UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

TRANSPORT CONTROL PROTOCOL CON SISTEMA INTEGRADO PARA LA EVASIÓN DE CONGESTIÓN BASADO EN UNA MÉTRICA ORIGINAL DE CONGESTIÓN

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA

LILIANA SOLEDAD ZURITA INOSTROZA

PROFESOR GUÍA:
CLAUDIO ESTÉVEZ MONTERO

PROFESOR CO-GUÍA:
MARCOS ORCHARD CONCHA

MIEMBROS DE LA COMISIÓN:
CÉSAR AZURDIA MEZA
SAMUEL MONTEJO SÁNCHEZ

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA
POR: LILIANA SOLEDAD ZURITA INOSTROZA
FECHA: 2023
PROF. GUÍA: CLAUDIO ESTÉVEZ MONTERO

TRANSPORT CONTROL PROTOCOL CON SISTEMA INTEGRADO PARA LA EVASIÓN DE CONGESTIÓN BASADO EN UNA MÉTRICA ORIGINAL DE CONGESTIÓN

El desempeño de TCP se degrada en condiciones de alto *Bandwidth Delay Product*, donde el producto del *delay* y el ancho de banda supera 10^5. Lo que ha motivado históricamente la investigación de algoritmos que lo resuelvan.

El trabajo presentado consiste en el diseño, implementación y evaluación de dos versiones del Protocolo de Control de Transmisión (TCP, *Transmision Control Protocol*) donde la ventana de congestión se administra empleando un controlador PD y uno difuso respectivamente. Esto difiere de los métodos AIMD-based tradicionales en que el incremento y el decremento de la ventana son diseñados con valores variables que dependen del grado de congestión percibido. Como referencia del controlador se utiliza una ecuación de predicción derivada de un modelo de *throughput* que se generaliza en la investigación *General Additive Increase Multiplicative Decrease Congestion Control*.

Se midió su desempeño en cuanto a utilización y *throughput* en escenarios donde el *Bandwidth Delay Product* es alto. Se usaron simulaciones diseñadas en OPNET y controladores sintonizados mediante métodos de optimización en MATLAB. Esto mejoró el rendimiento respecto de TCP Reno tanto en utilización del ancho de banda como en *fairness* y TCP-*friendliness* alcanzando incrementos en utilización entre 12 % y 18 %.

A mis padres.

Agradecimientos

Le agradezco a mis profesores del departamento de ingeniería eléctrica por la formación que recibí, en especial al profesor Claudio Estévez por intentar estimular el interés en la investigación en mí. Le agradezco también a todos los revisores de esta tesis por su dedicación y retroalimentación para mejorar este documento.

Le agradezco a mis padres y hermana por su apoyo siempre. A mis amigos Vale, Boris, Cris, Gus y Mati porque hicieron mi paso por la universidad un tiempo feliz. Además le agradezco a mis compañeros de laboratorio Boris y Diego por su apoyo y los buenos momentos compartidos.

Finalmente, le agradezco a Jens, porque su confianza en mi trabajo y aptitudes me volvió a acercar a la ingeniería y me motivó para volver a darle un cierre a la etapa universitaria. También le agradezco a Lucre por incentivarme respecto de lo mismo.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Abreviaciones

ABR Available bit rate. 61, 83, 147

ACK Acknowledgement. 5, 147

AI Additive Increase. 12, 147

AIMD Additive Increase Multiplicative Decrease. 32, 147

AQM Active Queue Management. 47, 147

ATM Asynchronous Transfer Mode. 61, 147

BDP Bandwidth Delay Product. 11, 147

BIC Binary Increase Congestion Control for Fast Long-Distance Networks. 19, 147

CEN Carrier Ethernet Networks. 83, 147

CIR Commited Information Rate. 147

CLTCP Congestion control algorithm-Coupling Logistic TCP. 19, 147

CUBIC CUBIC is an enhanced version of BIC. 19, 147

CWND Congestion Window. 147

DCC Delay-based Congestion Control. 81, 147

EIR Excess Information Rate. 84, 147

ERICA Explicit Rate Indication Control. 61, 147

ESTP Ethernet Services transport protocol. 83, 147

FCFS First-Come-First-Served. 66, 147

FIFO First-In-First-Out. 62, 147

FR Fairness Ratio. 81, 147

FTP File Transfer Protocol. 1, 147

GAIMD General Additive Increase Multiplicative Decrease. 9, 147

HBDP High Bandwidth Delay Product. 147

HSTCP HighSpeed TCP. 19, 147

HTCP TCP for high-speed and long-distance networks. 19, 147

LTCP Layered TCP. 147

MD Multiplicative Decrease. 12, 147

MLCP Multi-Level feedback Congestion control Protocol. 19, 147

PCC Packet-loss-based Congestion Control. 81, 147

PSO Particle Swarm Optimization. iv, 15, 147

RAS Rate Allocation Server. 66, 147

RED Random Early Detection Gateways for Congestion Avoidance. 44, 147

RTT Round Trip Time. 1, 147

rwnd Advertised Window. 147

SACK Selected Acknowledgement. 11, 147

SMTP Simple Mail Transfer Protocol. 1, 147

ssthresh Slow Start Threshold. 147

STCP Scalable TCP. 19, 147

TCP Transport Control Protocol. 1, 147

TD Triple Duplicated. 147

TELNET Teletype Network. 1, 147

VBR Variable bit rate. 62, 147

VC Virtual Circuit. 62, 147

VCP Variable-structure congestion Control Protocol. 19, 147

XCP Explicit Control Protocol. 19, 147

1. Introducción

1.1. Motivación

Desde el principio de los años 90 comenzó el incremento en el volumen de las transmisiones TCP [1], llegando a ser el tráfico dominante en Internet. En los años 2000s ciertas fuentes afirmaban que este había alcanzado una presencia de entre el 85 % y 95 % del total de transmisiones [2] [3]. La preponderacia que TCP alcanzó , y su incidencia dependía principalmente del tipo de red. La mayoría del tráfico en Internet, transferencia de archivos File Transfer Protocol (FTP), acceso remoto Teletype Network (TELNET), email Simple Mail Transfer Protocol (SMTP), era de tipo TCP [4]. En la actualidad TCP todavía constituye el núcleo de la red [5] y el desempeño de este protocolo impacta en el rendimiento de toda la Internet, es decir, en la transferencia de información, lo cual es relevante, por ejemplo, para el concepto de *Cloud Computing*. Pues esto requiere procesar una gran cantidad de información [6], y TCP ha mostrado que tiene un comportamiento deficiente en redes de alta velocidad y alto *delay*. Las redes de alta velocidad también son importantes en aplicaciones como física de alta energía, bioingeniería y radioastronomía que requieren el envío de grandes cantidades de datos [7]. La mejora de los protocolos basados en TCP puede contribuir al desempeño de toda la red lo que motiva su investigación hasta el presente. Además, en la actualidad el desarrollo de nuevas tecnologías como 5G y la internet de las cosas han dado origen a nuevos desafíos [8] [9] en el transporte en redes lo que sigue motivando su estudio.

Este trabajo tiene como antecedente el trabajo de memoria de Felipe Salas titulado "Pronóstico de la Tasa de Transmisión a Nivel de la Capa de Transporte" en el que se implementó un método de predicción para el ancho banda disponible del canal y se ajustaron las constantes de un controlador PI mediante un método heurístico, de ensayo y error. Las pruebas fueron desarrolladas en un servidor con sistema operativo Linux donde se programó el módulo de evasión de congestión propuesto y se analizaron simulaciones para diversos valores de *Round Trip Time* RTT con el fin de corroborar la validez de la ecuación de predicción del ancho de banda de referencia del controlador.

1.2. Problema a Abordar

TCP degrada su desempeño en contextos donde el *Bandwith Delay Product* es alto. Se ha considerado que este valor es alto cuando el producto entre el ancho de banda y el retardo supera 10^5 [10]. Esto ocurre, por ejemplo, en las redes de alta velocidad [7]. En ese tipo de escenarios se vuelve muy vulnerable a la congestión o a la pérdida de paquetes aleatoria. Esto ha motivado el diseño de múltiples protocolos que intentan reparar esta debilidad. Por otro lado, TCP disminuye su eficiencia en cuanto a utilización cuando está en presencia de tráfico sin control de la congestión. El cual suele acaparar los recursos disponibles. Por lo tanto, es necesario considerar que al diseñar un protocolo este tenga características de *fairness* o TCP-*friendliness* [11].

A causa de estos requerimientos se han desarrollado números estudios y propuestas de protocolos, entre ellos, los protocolos conocidos como AIMD-based[12]. En este trabajo se presenta y experimenta con una propuesta de protocolo que toma algunas características de este grupo, combinada con el uso de controladores de distintos tipos buscando medir su desempeño en cuanto a velocidad de datos transmitidos y *fairness*.

1.3. Hipótesis

En esta investigación se plantea que la modificación del algoritmo y *Congestion Avoidance* de TCP en sus parámetros de incremento y decremento, utilizando técnicas del control de sistemas permiten mejorar el *throughput* sin perjudicar la característica de TCP-*friendliness* respecto de TCP-Reno en escenarios de alto *Bandwidth Delay Product*, es decir, donde el producto del ancho de banda y el retardo de la red es alto. Se consideran dos tipos de controladores, un controlador PD y uno difuso que ajustan los factores de decremento multiplicativo y incremento aditivo de TCP. Se utiliza una referencia para controlador obtenida mediante una técnica predicción basada en un modelo de estimación de *throughput* teórico que se aplica mediante la contabilización de los paquetes perdidos.

1.4. Objetivos

1.4.1. Objetivos Generales

El objetivo general de esta investigación es diseñar, implementar y analizar dos propuestas de métodos de control de congestión sobre TCP utilizando técnicas de control de sistemas. El análisis considera la evaluación de su desempeño comparativo con TCP-Reno en cuanto a *throughput* y TCP-*friendliness* en un entorno de alto *Bandwidth-Delay Product* .

1.4.2. Objetivos Específicos

Los objetivos específicos son:

- Diseñar las propuestas de protocolos de congestión, eligiendo los tipos de controladores y los métodos de sintonización de parámetros.
- Diseñar experimentos para cuantificar el comportamiento de los protocolos propuestos en cuanto a TCP-*friendliness* y *throughput*.
- Implementar las pruebas diseñadas y comparar los resultados.

1.5. Metodología

La presente investigación es de tipo cuantitativo y experimental. Los protocolos a diseñar tienen como propósito responder de forma eficiente en un entorno de alto *Bandwidth Delay Product*. Es decir responder adecuadamente a las pérdidas manteniendo un buen *throughput* y utilización y sin acaparar los recursos del canal.

Algunas de las actividades realizadas para conseguir los objetivos de esta investigación son:

1. Estudio de protocolos propuestos que resuelven el problema y revisión bibliográfica.
2. Elección de tipos de controladores.
3. Elección de los métodos para sintonizar los controladores.
4. Diseño e implementación de los escenarios para simular el comportamiento de los los protocolos.
5. Diseño de los experimentos para medir las distintas cualidades de los protocolos.
6. Mediciones, evaluación de desempeño y propuestas de mejoras.

1.6. Estructura de la tesis

El orden en que se presenta el temario de la tesis es el siguiente:

En el capítulo 2 se expone la contextualización , es decir, un breve marco teórico con los conceptos más importantes que se tratan en este trabajo.

En la capítulo 3 se expone una investigación bibliográfica de trabajos relacionados, que proponen distintas técnicas, diferentes tipos de modelos matemáticos para predicción del *throughput* y protocolos planteados para la transmisión de información en la red con finalidades

similares.

En el capítulo 4 se detalla el diseño de protocolos que se testearán, así como su base teórica. También se describen las técnicas y algoritmos necesarios para la implementación de estos y las funciones que permitirán realizar las simulaciones y la evaluación de los protocolos. En el capítulo 5 se presentan los resultados de ambos métodos, se cuantifica su desempeño respecto, del *throughput*, utilización y TCP-*friendliness*. Finalmente se muestran las conclusiones y se propone el trabajo futuro.

2. Contextualización

En este capítulo se presentan los principales conceptos necesarios para comprender el diseño del protocolo planteado. En la sección 2.1 se explica que es el protocolo TCP, cómo funciona y los principales algoritmos de que se compone, el significado de incremento aditivo (AI) y decremento multiplicativo (MD). En la sección 2.2 se presenta el algoritmo que generaliza AI y MD y en el cual se basa el diseño del protocolo a evaluar en este trabajo. Luego, en la sección 2.3 se analiza el problema a resolver y finalmente en 2.4 se describen los algoritmos de control.

2.1. TCP

TCP es un protocolo [13] que permite que la conexión sea orientada, esto quiere decir que dos aplicaciones usando TCP deben establecer una conexión, esto es enviar segmentos preliminearmente para establecer los parámetros que antes de intercambiar información. La conexión en TCP es lógica y su estado reside los dos extremos [14].

La información que recibe la capa es fragmentada, la unidad de información transmitida por TCP a IP se llama segmento, y es en TCP donde se determina el tamaño del segmento. Al enviar TCP una unidad de datos espera desde el receptor una confirmación de recepción del segmento que consiste en otro paquete llamado Acknowledgement (ACK), si esto no es recibido en cierto intervalo tiempo entonces el segmento se retransmite.

Cada extremo de una conexión TCP tiene una cantidad limitada de almacenamiento. Ante esto TCP ejerce control de flujo, esto significa que es posible enviar solo cierto volumen de información acorde a la capacidad de almacenamiento del receptor.

TCP es un protocolo que se caracteriza por ser confiable, por eso, tiene implementado un *checksum* en su cabecera y su *data*, que es útil para determinar si hubo alguna alteración en la información que se está transmitiendo, en caso de que la hubiera, se descarta el segmentos que llega con error y este deberá ser retransmitido. Mediante transmisión TCP los datagramas podrían llegar en desorden, el receptor reordena los paquetes si es necesario entregando la información ordenada a la aplicación.

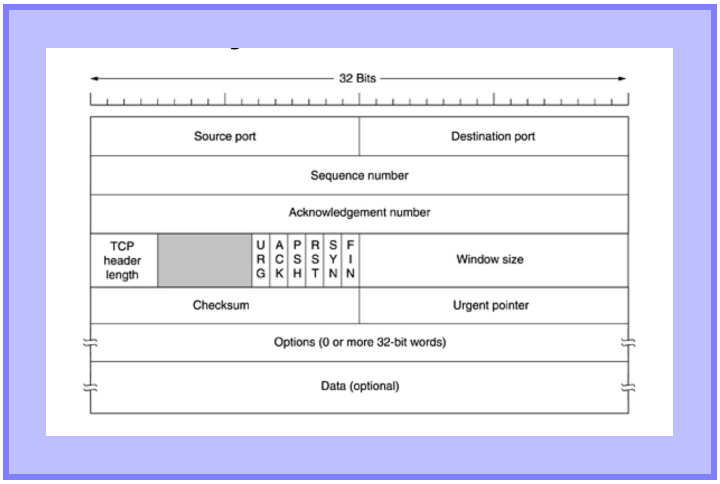


Figura 2.1: Header TCP

2.1.1. Header TCP

La data TCP se encapsula en un datagrama IP que consiste en una cabecera o *header* IP, una cabecera o *header* TCP [13]y los datos TCP. La longitud del *header* está limitada a 60 bytes, pero por lo general es de 20 bytes. Para identificar o precisar cada conexión los segmentos TCP contienen en su cabecera el número de puerto de origen y de destino. Se le llama *socket* a la combinación de dirección IP y número de puerto de cliente y servidor.

El número de secuencia identifica el byte que corresponde a ese segmento en el flujo de datos que viaja desde un emisor TCP a un receptor TCP. El primer byte de información enviado por el *host* será el *initial sequence number* (ISN) más uno, ISN está especificado en el *sequence number field*. Esto es porque al haberse establecido una conexión nueva el *flag* SYN fue activado, y este consume un número de secuencia.

El *acknowledgement number* es válido sólo si ACK *flag* es uno, lo que ocurre cuando se ha establecido la conexión. Este campo corresponde al número de secuencia más uno, y representa el paquete que se espera recibir o en otras palabras hasta el número de bytes que han sido recibidos con éxito. De acuerdo a lo anterior, la ventana deslizante que describe el protocolo TCP en su transmisión cambia.

2.1.2. Protocolo de Establecimiento de Conexión

Como ya se ha mencionado TCP [13] es un protocolo orientado a la conexión, lo que quiere decir que debe establecerse comunicación antes de comenzar a transmitir la información, por eso se ejecuta un protocolo específico para lograr esto.

Para establecer una conexión TCP el cliente envía un segmento llamado SYN que contiene información del número del puerto del servidor al que el cliente quiere conectarse y la secuencia inicial del número del cliente. El servidor envía como respuesta su propio segmento SYN que contiene la secuencia inicial del número. El servidor además confirma al cliente ha-

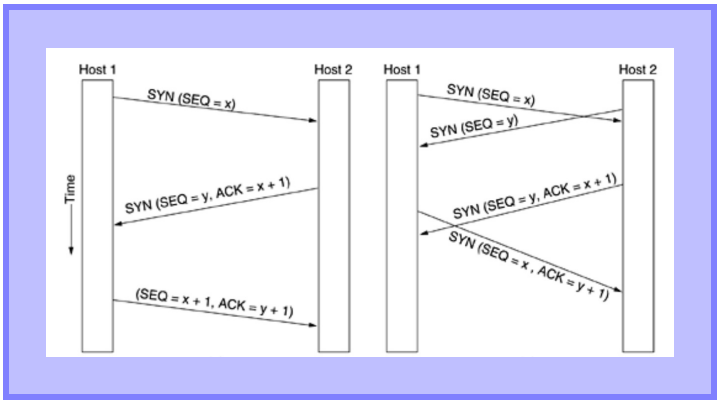


Figura 2.2: Establecimiento de conexión

ber recibido su SYN enviando el número ISN más uno. Finalmente el cliente confirma haber recibido el SYN desde el servidor enviando el ISN del servidor más uno.

A todo este proceso se le conoce comúnmente como *three-way handshake*. Se describe gráficamente en la Figura 2.2.

2.1.3. Implementación TCP: Slow Start y Congestion Avoidance

La implementación moderna de TCP incluye cuatro algoritmos [13] que nunca han sido totalmente documentados como estándares de internet.

- Slow Start
- Congestion Avoidance
- Fast Retransmit
- Fast Recovery

El Viejo TCP comenzaba la conexión inyectando múltiples segmentos a la red por parte del emisor. Algunos *routers* intermedios debían guardar los paquetes haciendo posible que el enrutador pudiera quedar sin espacio por lo que se hizo necesario implementar el algoritmo conocido como *Slow Start*.

Slow Start inicializa el flujo de datos en la conexión. Trabaja haciendo que la velocidad en que se inyectan nuevos paquetes concuerde con la razón en que los *acknowledgments* retornan. *Slow Start* requiere de la variable *congestion window* o ventana de congestión cwnd. Cuando se establece conexión con un host, la ventana de congestión se inicializa. La ventana de congestión se incrementa en un segmento si se recibe un ACK (cwnd se mantiene en bytes, pero *Slow Start* la incrementa por el tamaño de segmento). El control de flujo de la ventana de congestión es impuesta por el emisor. El emisor transmite un segmento y espera por su ACK. Cuando se recibe el ACK la ventana de congestión se incrementa en uno o dos segmentos y se pueden enviar dos segmentos. Cuando estos segmentos son recibidos (*ACKnowledged*) la ventana de congestión se incrementa a cuatro, esto determina un aumento exponencial.

Congestion Avoidance y *Slow Start* son algoritmos independientes con objetivos diferentes, mientras *Slow Start* inicializa el flujo de datos *Congestion Avoidance* se encarga de las pérdidas de paquetes.

En la transmisión de información se llega un punto en que se alcanza el límite de flujo, por lo que empieza a producirse pérdida de paquetes, esto se manifiesta como un *timeout* o bien ACKs duplicados. Para enfrentar este problema se recurre a *Congestion Avoidance*. Cuando ocurre congestión se desea aminorar la velocidad de paquetes hacia la red para luego regresar a la situación anterior usando *Slow Start* nuevamente, por lo tanto, se implementan ambos juntos.

Congestion avoidance y *Slow Start* requieren de dos variables:

- *Congestion window*, cwnd.
- *Slow start threshold size*, ssthresh.

El algoritmo *Congestion Avoidance* consiste en los siguientes pasos:

- Se inicializa poniendo *cwn*d a un segmento y *ssthresh* a 65535 *bytes*.
- Usando TCP nunca se envía más que el mínimo de *cwnd* y la *advertised window* del receptor.
- Al ocurrir congestión, lo que está dado por haber alcanzado el *timeout* o la recepción de ACKs duplicados la mitad del actual tamaño de la ventana, el mínimo de *cwn*d y la *advertised window* se guarda en *ssthresh*. Además si se llega a la congestión por *timeout*, se le da valor uno a *cwn*d.
- Si se recibe un ACK se incrementa *cwnd*, pero cuánto se incrementa depende del funcionamiento del *Slow Start* y *Congestion Avoidance*.

El algoritmo *Congestion Avoidance* determina que se incremente en $1/cwn$d la ventana cada vez que se confirma la recepción de un paquete a través de un ACK lo que corresponde a un incremento aditivo de la ventana *cwn*d en comparación con *Slow Start* que es exponencial.

En la Figura 2.3 se ve que la congestión sucede en *cwn*d con valor 32 y por lo tanto *ssthresh* toma el valor 16 y *cwn*d el de 1 segmento . Se puede notar que la primera parte tiene forma exponencial que corresponde a $SlowStart$. En el tiempo cero se envió un segmento se supone que el ACK fue recibido a tiempo con lo que hubo un aumento en un segmento, luego se reciben los dos ACKs a tiempo con lo que la ventana se incrementa a 4, etc, hasta llegar al valor máximo de *ssthresh* volviéndose lineal el incremento en esta zona.

2.1.4. Algoritmo de Rápida Recuperación y Rápida Transmisión

En 1990 se propusieron modificaciones al algoritmo, es importante primero darse cuenta que un ACK duplicado puede producirse por un segmento perdido o porque los paquetes han llegado en desorden. Si hay un reordenamiento habrá uno o dos duplicados, en cambio si se tiene dos o más entonces esto es indicador de que el segmento puede haberse perdido. El algoritmo de *fast recovery* se refiere a pedir la retransmisión del paquete antes de alcanzar el

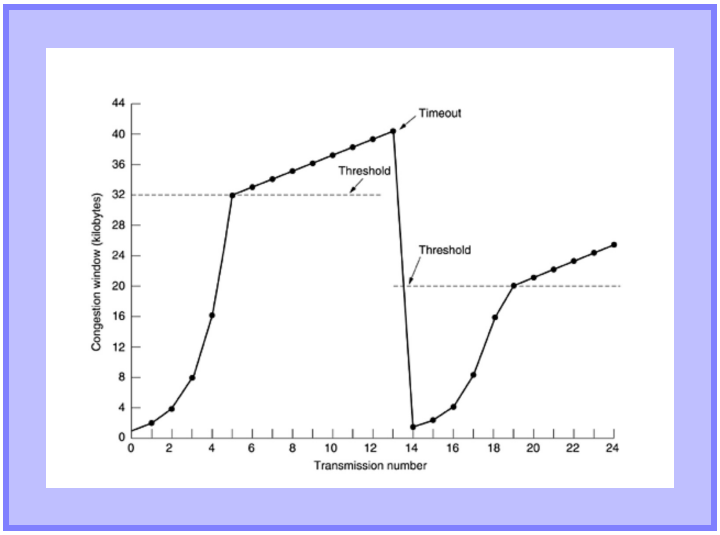


Figura 2.3: Algoritmos Slow-Start y Congestion Avoidance

timeout. No se aplica *Slow Start* después de haber recibido tres ACK duplicados, esto porque para que haya un ACK duplicado el receptor ha recibido otro segmento que está en el *buffer* del receptor lo que significa que está fluyendo la información entre emisor y receptor, y por lo tanto, no es necesario reducir bruscamente la tasa de transmisión empezando con *Slow Start* otra vez. Los algoritmos se implementan comúnmente:

- Cuando se recibe el tercer ACK duplicado se le da el valor de la mitad de la actual ventana de congestión a $ssthresh$, y se retransmite el segmento que falta. Se pone $cwnd$ a $ssthresh$ más tres veces el tamaño del segmento.
- Cada vez que otro ACK duplicado llega se incrementa la ventana y se transmite un paquete.
- Cuando llega el próximo ACK que confirma nueva *data*, se pone $cwnd$ a $ssthresh$.

2.2. GAIMD

Los métodos usados para diseñar los algoritmos de control de congestión que serán evaluados se basan en una técnica conocida como *General additive increase multiplicative decrease control congestion*, GAIMD [15]. Esta técnica es a su vez una variación de AIMD que se utiliza en TCP. AIMD incrementa la ventana en un uno (*Additive increase*, AI=1) por ventana de datos recibida y divide en dos (*Multiplicative decrease*, MD 0.5) el tamaño de la ventana por cada pérdida ocurrida. En GAIMD los valores de incremento pueden ser distintos, investigadores han mostrado que estos valores pueden llegar a tener un comportamiento *TCP-friendly* Los primeros que estudiaron GAIMD fueron Chiu y Jain [12], y se abocaron principalmente al estudio de estabilidad y *fairness*, encontrando que α y β deben satisfacer la siguiente relación.

$$\begin{aligned} 0 < \alpha\,, \\ 0 < \beta < \alpha\,. \end{aligned} \tag{2.1}$$

La expresión para la velocidad o *throughput* en función de los valores α y β es:

$$T_{\alpha,\beta}(p, RTT, t_0, b) = \frac{1}{RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}} + T_0\left(1{,}3\sqrt{\frac{(1-\beta^2)}{2\alpha}p}\right)p\,(1+32p^2)}\,. \tag{2.2}$$

Donde p representa la probabilidad de pérdida, T_o es el tiempo que espera el emisor para reenviar un paquete que no ha sido confirmado también llamado *timeout* y b el número de paquetes confirmados por cada ACK, que en el caso del desarrollo de este trabajo corresponde a uno.

De la fórmula de velocidad de envío de datos se desprende que es posible controlar la velocidad eligiendo pares (α, β). La fórmula fue calculada teóricamente haciendo algunos supuestos, por lo tanto su exactitud depende de los valores utilizados. En general, tiene buen desempeño cuando la pérdida es menor al 20 %.

Se definen $TD_{\alpha,\beta}$ y $TO_{\alpha,\beta}$ de la siguiente manera:

$$TD_{\alpha,\beta}(p, T_0, b) = RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p}\,, \tag{2.3}$$

$$TO_{\alpha,\beta}(p, T_0, b) = T_0\left(1{,}3\sqrt{\frac{(1-\beta^2)}{2\alpha}p}\right)p(1+32p^2)\,. \tag{2.4}$$

Si toda la congestión es del tipo triple ACK duplicado entonces en la expresión de *throughput* predomina el primer término $TD_{\alpha,\beta}$ 2.3, se considera la expresión $TO_{\alpha,\beta}$ 2.4, en cambio, cuando la congestión se debe tanto a triple ACK duplicados y *timeouts*.

De la ecuación de velocidad se desprende que es posible controlar el flujo mediante el ajuste de los valores (α, β). De GAIMD [15] se dedujo además que es posible relacionar las expresiones 2.3 y 2.4 con la fórmula de la velocidad para obtener relaciones *friendly* entre α y β. Para obtener relaciones *friendly* se reemplaza el valor α por el valor que se usa en TCP como incremento, es decir, 1 y β se evalúa con el factor de decremento, o sea, 1/2. Esto es así porque el *throughput* en una relación *friendly* debería ser equitativo con un flujo TCP transmitido al mismo tiempo. Se reemplaza en la ecuación de TD 2.5 y se obtienen las expresiones 2.6 y 2.7.

$$TD_{\alpha,\beta}(p, RTT, b) = TD_{1,\frac{1}{2}}(p, RTT, b) \tag{2.5}$$

$$\frac{1-\beta}{\alpha(1+\beta)} = \frac{1-0{,}5}{1+0{,}5} \tag{2.6}$$

$$\alpha = \frac{3(1-\beta)}{1+\beta} \tag{2.7}$$

Se puede hacer los mismo con TO en 2.4, se calcula reemplazando los valores $\alpha = 1$ y $\beta = 1/2$ como se indica en 2.8 resultando la ecuación 2.9.

$$TO_{\alpha,\beta} = TO_{1,\frac{1}{2}}(p, RTT, b) \tag{2.8}$$

$$\sqrt{\frac{1-\beta^2}{\alpha}} = \sqrt{\frac{1-0{,}5^2}{1}} \tag{2.9}$$

Obteniéndose finalmente la relación entre α y β en 2.10.

$$\alpha = \frac{4(1-\beta^2)}{3} \tag{2.10}$$

2.3. Análisis del problema

El protocolo TCP muestra deficiencias de desempeño en entornos donde el *bandwidth* (ancho de banda) del canal, por medio del cual se transmite la información, es muy grande y/o el *delay* (retardo) de propagación es muy alto. Más específicamente se evidencian inconvenientes en el rendimiento del algoritmo en los casos donde el producto del *bandwidth* y del *delay* supera cierto umbral. En estos escenarios disminuye la eficiencia de TCP en el uso del ancho de banda disponible y además aumenta la tendencia a la inestabilidad. Esto se refiere a la disminución brusca del *throughput* en presencia de pérdidas.

El trabajo de Jacobson y Braden [10] es uno de los primeros que deja en evidencia las dificultades que tiene TCP para afrontar correctamente ambientes donde el *bandwidth-delay product* (BDP, producto ancho de banda-retardo) es demasiado alto. En esta investigación se expone que los problemas empiezan a presentarse cuando el producto BDP alcanza un valor superior a 10^5 bits. Para resolver esto propusieron una modificación de TCP conocida como TCP SACK que mejoró hasta cierto punto el funcionamiento del protocolo de transporte.

TCP, al igual que la mayoría de los protocolos *end-to-end*, utiliza las pérdidas como indicador de congestión y, por lo tanto, la acción de control se puede ejercer sólo cuando la sobrecarga de la red ya ha ocurrido. Esto ha sido considerado por algunos investigadores como el impedimento que obstaculiza al algoritmo alcanzar altos valores de utilización [16]. Pero para Tom Kelly [7] el principal problema de TCP en las redes BDP es la dificultad para recuperarse después de una pérdida. Esto significa que toma demasiado tiempo recobrar la velocidad de transmisión que se tenía antes de que un paquete fuese descartado. La lentitud de TCP radica en el método de actualización de la ventana, en consecuencia, se espera que modificándolo sea posible mejorar la agilidad en el restablecimiento de la velocidad. Los protocolos sugeridos en esta tesis se basan, entre otras, en la premisa anterior y alteran TCP buscando que responda más rápido a las condiciones de la red.

El principal objetivo de este trabajo es desarrollar dos variantes del protocolo TCP obtenidas mediante la modificación del algoritmo *Congestion Avoidance* de la versión conocida

como TCP Reno. Se aplica teoría de control modelando el proceso de adaptación del tamaño de ventana al ancho de banda como un controlador de lazo cerrado que regula el incremento y decremento de esta. En esta tesis se diseñan y prueban dos tipos de controlador: un controlador lineal PD y un controlador de lógica difusa.

La proposición de insertar un controlador en el proceso *Congestion Avoidance* tiene como fin que el algoritmo pueda regular el grado de incremento y decremento de la ventana de congestión y el *throughput* acuerdo al estado del sistema respecto de la *throughput* libre. Para poder responder ágilmente en los casos donde el *bandwidth delay product* es muy grande.

El escenario donde se ejecutan las pruebas considera un servidor y un cliente en los extremos. La información se transmite a través de un canal que tiene un ancho de anda C, se asume que el canal tiene una tasa de pérdidas p y que los paquetes de descartan de manera aleatoria con una distribución probabilística uniforme. El sistema se modela con un *delay* del propagación d, no se tienen en cuenta en particular el *delay* de procesamiento, ni el *delay* de encolamiento.

El par AI (Additive Increase, incremento aditivo) y MD (Multiplicative Decrease, decremento multiplicativo) corresponden respectivamente a los parámetros que determinan el incremento y decremento de la ventana en el algoritmo *Congestion Avoidance*. En TCP estos operadores son fijos y toman los valores $[1, 0{,}5]$. Sin embargo, en el protocolo propuesto varían en el tiempo, además se les denota por α y β y la magnitud que alcanzan resulta de la decisión del controlador en el tiempo k. Por lo tanto, α y β constituyen la variable manipulada $u(k)$ 2.11. El regulador escoge los valores de la variable manipulada comparando el tamaño de la ventana con la estimación de la tasa de envío disponible que representa a la referencia R.

$$u(k) = [\alpha(k), \beta(k)] \tag{2.11}$$

La variable manipulada mantiene su valor mientras no ocurra un evento de pérdida y, por lo tanto, la acción de control se preserva durante el intervalo [k,k+1]. En ese periodo la ventana crecerá según el incremento aditivo $\alpha(k)$ cada vez que el emisor recibe un ACK. Esto continúa hasta el instante cuando ocurre el descarte de un paquete, en ese momento el tamaño de la ventana disminuye por el factor $\beta(k)$ y se recalcula la variable manipulada. En TCP Reno las pérdidas son detectadas mediante dos mecanismos *triple-duplicate ACK* y *timeout*. Estos procesos están implementados en las funciones *Fast Retransmit* y *Fast Recovery* y, en consecuencia, los códigos de estas son alterados para aplicar el controlador.

El sistema no es de paso fijo porque la decisión de control se ejerce en cuanto ocurre una pérdida y el intervalo de tiempo entre estos sucesos consecutivos es variable pues es de origen aleatorio, esto es considerado tanto en la modelación como en los cálculos. Además la dinámica del sistema es no lineal.

Las variables que decriben el sistema y sus interacciones se resumen en el esquema siguiente: En la figura 4.5 se indica la retroalimentación de la salida que es este caso corresponde al tamaño de la ventana $w(k)$. Además se observa que el controlador recibe como entrada la

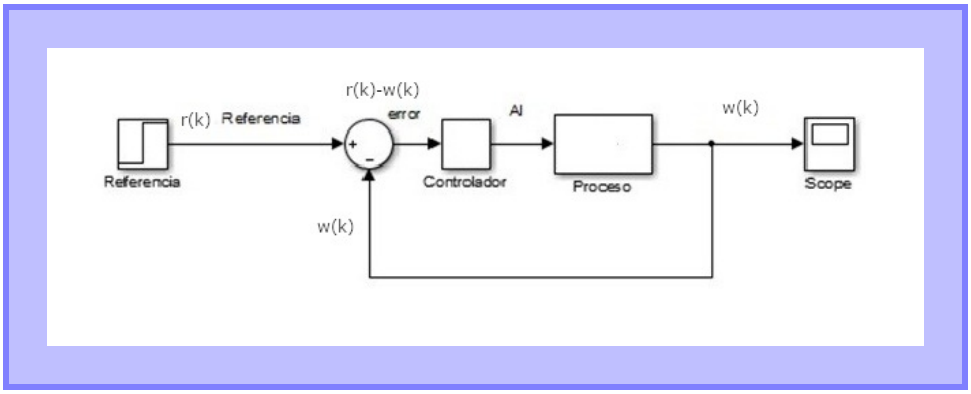


Figura 2.4: Esquema Controlador

diferencia entre la referencia $r(k)$ y la salida $w(k)$. Finalmente se muestra que la referencia es estimada usando la tasa de pérdidas registrada en el proceso.

2.4. Control

2.4.1. Controladores PID

El controlador PID [17] es la forma más común de retroalimentación, formó parte de los primeros controladores que existieron a partir de 1940 y está presente hasta nuestros días. Han sido construidos por cientos de miles año a año, tienen aplicación en las áreas de la producción de energía, transporte y manufactura. Además se han usado en conjunto con otras estrategias de control para crear sistemas de control más sofisticados. Los controladores han sobrevivido a los cambios en la tecnología evolucionando desde la neumática y mecánica hasta los transistores y circuitos integrados.

El algoritmo que describe a los controladores PID está dado por la ecuaciones:

$$u(t) = K\left(\mathrm{e}(t) + \frac{1}{T_\mathrm{i}}\int_0^t \mathrm{e}(\tau)\mathrm{d}\tau + T_\mathrm{d}\frac{\mathrm{de}(t)}{\mathrm{d}t}\right), \tag{2.12}$$

$$\mathrm{e} = r - y\,. \tag{2.13}$$

Donde y es la variable de proceso medida, r es la variable de referencia, u es la señal de control y e es el error de control. A la variable de referencia con frecuencia se llama también *set point*. La señal de control es la suma de la parte proporcional al error, la parte proporcional a la derivada del error y la parte proporcional la integral del error.

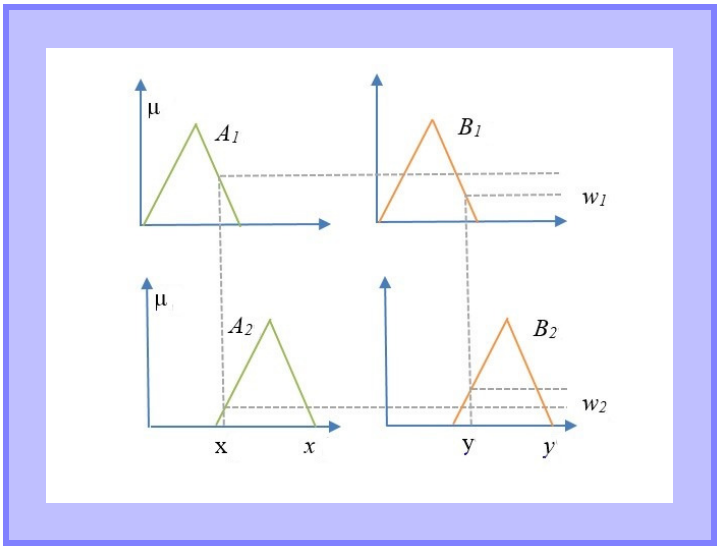


Figura 2.5: Grados de pertenencia de fuzzificación

2.4.2. Controlador Difuso

Los controladores difusos [17] han sido desarrollados para imitar el desempeño de operadores expertos codificando sus conocimientos en forma de reglas. La principal ventaja de un controlador difuso es la oportunidad de usar el conocimiento humano, la heurística y la intuición en el diseño del controlador. La principal desventaja es la falta de métodos para analizar el desempeño del controlador, en cuanto a su estabilidad y robustez.
Cualquier sistema difuso consta de 4 procesos:

1. Base de conocimiento: reglas y parámetros de las funciones de pertenencia.

2. Unidad de decisión: Operaciones de inferencia.

3. Interface de fuzzificación: transformar las entradas en grados de activación.

4. Interface de defuzzificación: transformación de los resultados de inferencia en una salida.

Hay muchos tipos de razonamiento difuso. Algunos de los más importantes son:

Método min-max: La salida de la función de pertenencia es la unión de los grupos asignados después de cortar el grado de pertenencia de la premisa correspondiente. La salida es comúnmente el centro de gravedad del grupo.

Takagi-Sugeno: Cada salida de regla es la combinación de las variables de entrada. La salida corresponde al promedio ponderado de cada salida de regla.

Aquí las reglas linguísticas son:

$$\begin{array}{lll} \mathrm{i}f\ x = A_1\ and\ y = B_1 & then & z_1 = f_1(x,y) = C_1\,, \\ \mathrm{i}f\ x = A_2\ and\ y = B_2 & then & z_2 = f_2(x,y) = C_2\,. \end{array} \tag{2.14}$$

En este caso la fuzzificación está dada por los grados de pertenencia de las variables x e y en las funciones de pertenencia definidas en Figura 2.5. La inferencia corresponde para cada i-ésima regla o premisa a:

$$min\left(\mu_{A_1}(x) and \mu_{b_\mathrm{i}}(y)\right) = \mu_{C_\mathrm{i}}(z)\,. \tag{2.15}$$

Con lo anterior se obtiene la interface de fuzzificación $\mu_{C_\mathrm{i}}(z)$ o grado de activación de cada regla, dos para este ejemplo. Las operaciones que se usan en la fase de inferencia comúnmente son mínimo o producto. Los grados de activación quedan:

$$\begin{array}{l} \mu_{C_1}(z_1) = w_1\,, \\ \mu_{C_2}(z_2) = w_2\,. \end{array} \tag{2.16}$$

Si $z_1 = f_1(x,y) = ax + by + c$ y $z_2 = f2(x,y) = px + qy + r$. Entonces la defuzzificación según el razonamiento de Takagi-Sugeno es:

$$z = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_2}\,. \tag{2.17}$$

2.4.3. Optimización PSO

Es un tipo de optimización [18], [19] que se basa en la habilidad de las sociedades de obtener conocimiento. Esto consiste en un grupo de individuos cuyo comportamiento es influenciado, por sus propias búsquedas, como también por el de quienes le rodean con el objetivo de tener éxito, bajo el supuesto de que existe alguna medición de bondad. Se realizan experimentos iterativos, donde las partículas, que son en realidad una representación de un punto en el espacio, calculan la distancia entre su resultado y el resultado deseado, cambiando consecuentemente su posición. Para evitar que la posición de la partícula cambie indefinidamente, se regula el cambio de esta mediante v_id.

$$x_\mathrm{id} = x_\mathrm{id} + v_\mathrm{id} \tag{2.18}$$

Cada partícula compara su error actual, con el mejor que han tenido es decir, el menor. Se le llama a este valor mínimo $pbest_\mathrm{i}$. y a la posición que logra este valor se llama $P_\mathrm{i}\mathrm{d}$. Al vector de cambio se le suma la diferencia entre la anterior mejor posición y la actual ponderada por un factor aleatorio, de manera que la posición tendrá una estocástica tendencia a regresar al anterior mejor.

$$v_\mathrm{id} = v_\mathrm{id} + \varphi_\mathrm{id}(P_\mathrm{id} - X_\mathrm{id}) \tag{2.19}$$

También se reconoce una variante del algoritmo que incluye una componente social ($P_{g\mathrm{d}} - X_\mathrm{id}$), representa la distancia entre la posición de una partícula y la mejor que ha sido encontrada por cualquiera de ellas. La ecuación 2.20 representa el cambio de posición para cada partícula x_id.

$$v_\mathrm{id} = v_\mathrm{id} = +\varphi_\mathrm{id1}(P_\mathrm{id} - X_\mathrm{id}) + \varphi_\mathrm{id2}(P_{g\mathrm{d}} - X_\mathrm{id}) \tag{2.20}$$

El modelo de 2.19 conocido como "*cognition model*" y que modela las partículas como entes independientes, comparado con el modelo completo tiene una tendencia levemente superior a quedarse atrapado en una solución local, y falla en encontrar las regiones óptimas y, con frecuencia, busca solo en torno a los valores en que fueron inicializadas las partículas.

El modelo "*social-only*" que se resume en 2.21, representa un proceso donde los individuos no tienden a regresar a los valores que les han resultado exitosos a ellos mismos en el pasado.

$$v_{\mathrm{id}} = v_{\mathrm{id}} + \varphi_{\mathrm{id}}(P_{g\mathrm{d}} - X_{\mathrm{id}}) \tag{2.21}$$

Este modelo converge más rápido al óptimo que el modelo completo y es levemente más vulnerable de caer en un óptimo local.

Variables a Considerar para PSO

Tal como se mencionó en 2.4.3 las variables que son necesarias para modelar un problema de optimización mediante la técnica de PSO son:

1. La posición de las partículas cada partícula en la i-ésima iteración $x_{\mathrm{i}} = [x_1^{\mathrm{i}} x_2^{\mathrm{i}} x_3^{\mathrm{i}} \dots x_n^{\mathrm{i}}]$
2. El vector de cambio de posición para cada partícula $v_{\mathrm{i}} = [v_1^{\mathrm{i}} v_2^{\mathrm{i}} v_3^{\mathrm{i}} \dots v_n^{\mathrm{i}}]$
3. $pBest_{\mathrm{i}} = [p_1^{\mathrm{i}} p_2^{\mathrm{i}} p_3^{\mathrm{i}} \dots p_n^{\mathrm{i}}]$ el vector con mejor posición para cada partícula, esto es lo mismo que P_{id} de 2.4.3
4. Valor *fitness* guarda el valor de la función objetivo o de bondad del vector x_{i}, es decir, de cada partícula.

2.4.4. Resumen del Capítulo

En este capítulo se describe el protocolo de transporte TCP y los algoritmos que lo componen. TCP que es un tipo de conexión orientada que permite la transmisión de datos entre dos extremos de manera segmentada. TCP mantiene el estado de la transmisión en los extremos y una de sus principales fortalezas es que es confiable. Está formado por cuatro algoritmos que fueron desarrollados para enfrentar las pérdidas en la red, estos son *slow start*, *congestion avoidance*, *fast retransmit* y *fast recovery*.

Slow Start inicializa el flujo de datos y Congestion Avoidance se encarga de las pérdidas de paquetes, ambos se implementan juntos. Cuando la transmisión se encuentra en Congestión Avoidance el incremento del tamaño de la ventana es aditivo (AI, additive increase) e igual a uno y el decremento es multiplicativo (MD, multiplicative decrease) e igual a un medio. Es decir $AI = 1$ y $MD = 1/2$.

En este capítulo además se detalla una técnica llamada GAIMD, (General Additive Increse Multiplicative Decrease) que consiste en la generalización de AIMD, donde los investigado-

res encontraron para qué valores AI y MD diferentes de 1 y 1/2 las transmisiones podrían mantenerse *friendly*.

También se analiza el problema y la proposición de sistema integrado a TCP, que se diseña utilizando los conceptos de la técnica GAIMD descrita. TCP degrada su rendimiento en condiciones alto *Bandwidth Delay Product* al ocurrir pérdidas pues la ventana disminuye a la mitad y demora su recuperación. Por eso se propone insertar un controlador donde la variable manipulada serán los valores AI y MD que se elegirán respetando al relación de *friendly* presentada en GAIMD y tomarán determinado valor de acuerdo al grado de congestión percibido.

Los controladores utilizados son de tipo clásico y difuso. Y sus parámetros son sintonizados usando optimización PSO (Particle Swarm Optimization). PSO es un tipo de Optimización que se basa en la habilidad de las sociedades de obtener conocimiento a consecuencia de las búsquedas de los individuos y la influencia entre ellos.

3. Antecedentes de Estudio

En este capítulo se hace un recorrido por los distintos protocolos planteados por los investigadores para mejorar la velocidad de transmisión en contextos de alto *Bandwidth Delay Product*, múltiples pérdidas y congestión. Se comienza exponiendo la evolución de TCP en 3.1 y luego las distintas maneras en que se ha modelado TCP para entender su comportamiento y que se utilizado para dar origen a nuevos algoritmos que intentan mejorar su desempeño. También se presentan algunos algoritmos provenientes de esos tipos de modelamiento y otros que usan técnicas alternativas. Según sus características estos se clasifican en diferentes tipos.

Para facilitar el estudio y nuevos planteamientos distintos autores han agrupado los algoritmos de control de congestión de acuerdo a sus propiedades y origen. Por ejemplo, en la publicación *Adaptive Delay-based Congestion Control for High Bandwidth-Delay Product Networks* Jung [20] los agrupa en *router-supported* y *end-to-end* (Ver Figura 3.1). Los de tipo *router-supported* tienen un comportamiento destacado en redes de alto *bandwidth delay product* pero requiren de compatibilidad con los protocolos de flujo TCP más antiguos y son más difíciles de implementar. Los del conjunto *end-to-end* a su vez Jung los divide en *delay-based* y *packet loss-based*. Los algoritmos *delay-based* utilizan la medición del RTT y del cambio de la velocidad como indicador de congestión. En cambio los de tipo *packet loss-based* miden las pérdidas para cuantificar la congestión y algunos de ellos modifican el algoritmo AIMD característico de TCP para mejorar la utilización.

Por otro lado, Yang [15] también menciona una diferenciación de los protocolos y considera los grupos *AIMD-based* y *Formula-based* (Ver Figura 3.2). Aquellos que pertenecen a *AIMD-based* imitan el comportamiento de TCP para hacer crecer y decrecer el tamaño de la ventana de congestión. Por otro lado, los *formula-based* usan modelos estocásticos para derivar el *throughput* a partir del RTT, pérdidas y *timeout*.

Desde otra perspectiva, Estevez [21] distingue entre los algoritmos a las clases de los basados en modelos de segmento-tiempo,también los que se desarrollan sobre modelos markovianos y además algunas técnicas que se fundamentan en los modelos de fluidos y la teoría de control.

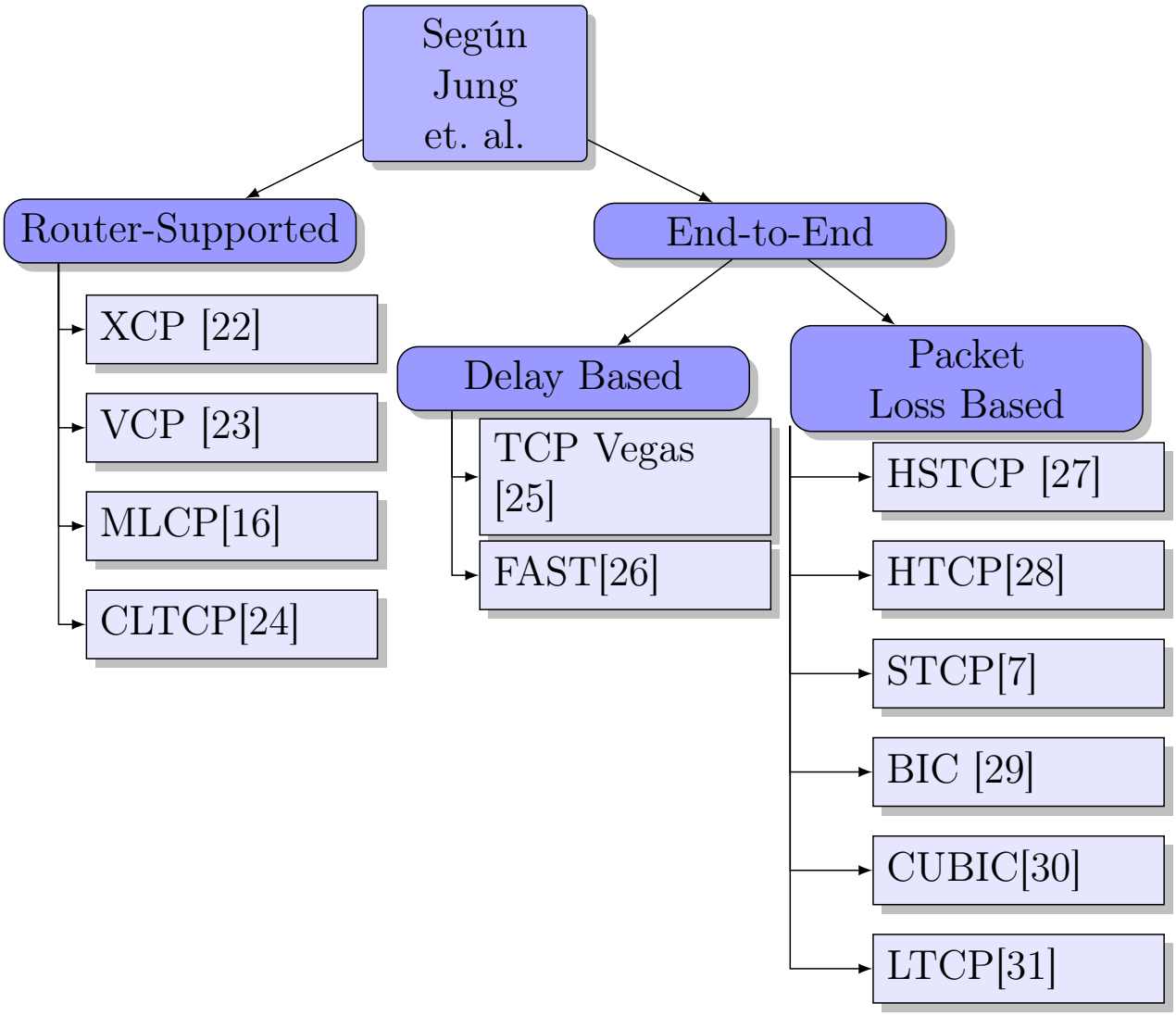


Figura 3.1: Clasificación según Jung [20]

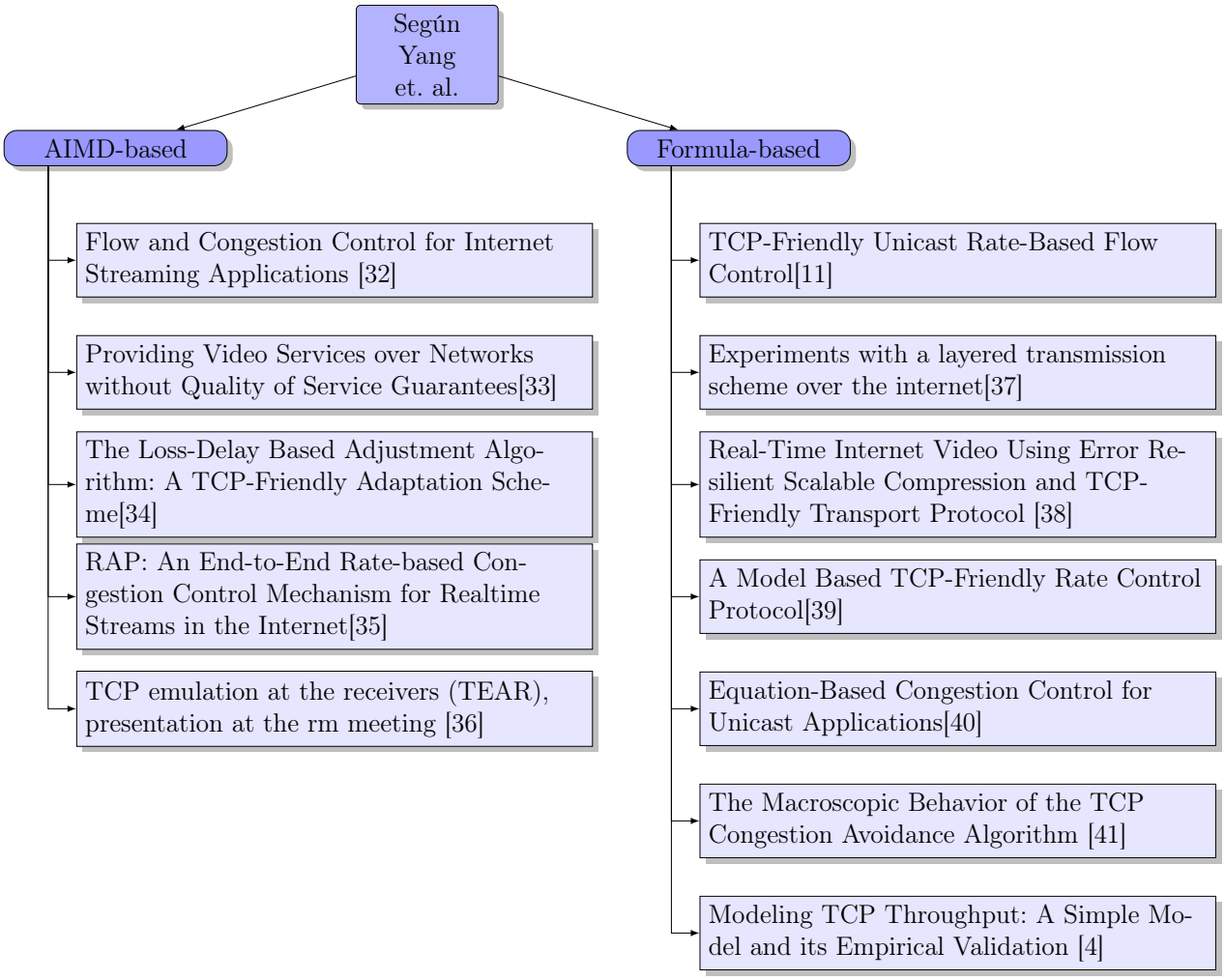


Figura 3.2: Clasificación según Yang [15]

3.1. Evolución de TCP

En la década de los ochenta debido a la explosiva expansión de las redes de computadores se evidenciaron problemas de congestión en la transmisión de información que se tradujeron en pérdidas del 10 % de los paquetes enviados a consecuencia de los desbordes en *buffers* y una drástica disminución en la velocidad de transmisión. Se hizo necesario investigar los protocolos existentes e introducir modificaciones para enfrentar esta situación. Hasta la actualidad se continúa investigando el desempeño de estos mecanismos de control de transporte, así han aparecido a lo largo del tiempo distintas versiones que se diferencian por los algoritmos incorporados. Entre estos se puede nombrar:

- Reno
- Tahoe
- New Reno
- SACK Reno

Según W. Stevens [42] las implementaciones modernas de TCP incluyen cuatro algoritmos denominados: *Slow Start*, *Congestion Avoidance*, *Fast Retransmit* y *Fast Recovery*. *Slow Start* y *Congestion Avoidance* fueron los primeros en ser desarrollados, en RFC1121 [43] se indica que deben ser implementados en conjunto de acuerdo a lo planteado por el trabajo de [44]. En cambio, *Fast Retransmit* y *Fast Recovery* son posteriores a RFC1121, primero se incluyó *Fast Retransmit* que apareció por primera vez en 4.3BDS Tahoe *release* y después *Fast Recovery* que fue aplicado en 4.3BSD Reno *release*.

Slow Start fue sugerido para mantener la estabilidad en el tráfico de paquetes, pues se observó que los colapsos por congestión podían evitarse haciendo que la transmisión respetara el principio de conservación de paquetes, donde conservativo se refiere a que es posible agregar un paquete solo si otro deja la red. Las versiones anteriores de TCP introducían una gran cantidad de paquetes a la red haciendo que estos fuesen encolados en *routers* intermedios lo que provocaba la disminución brusca del *throughput*. *Slow Start* impide que esto pase pues ingresa paquetes monitoreando la tasa en que regresan los *acknowledgments*, así es capaz de sondear lentamente la red para poder determinar la capacidad disponible. De acuerdo a lo anterior *Slow Start* es el algoritmo utilizado para iniciar la transmisión o bien aquel invocado tras ocurrir una pérdida.

El proceso *Slow Start* utiliza una variable llamada ventana de congestión (*congestion window*, $cwnd$), que es inicializada en uno al comenzar la transferencia de datos o después de una pérdida y que se incrementa por cada ACK recibido en una unidad. Se usa además la variable *advertised window* ($rwnd$) que corresponde al máximo de paquetes admisibles por el receptor, las variables $cwnd$ y $rwnd$ limitan la transmisión que quedará restringida al mínimo entre ambas. Otro parámetro importante es el llamado *Slow Start Threshold* ($ssthresh$) que permite discriminar si el proceso va a ejecutar *Slow Start* o bien *Congestion Avoidance*.

Congestion Avoidance es una función cuyo fin es hacer frente a la congestión de la red disminuyendo el flujo de paquetes. En otras palabras, el mecanismo que caracteriza este algoritmo contiene dos partes: la primera es que debe ser capaz de hacer notar a los *endpoints*

cuando la red está congestionada o bien en qué momento está al borde de ello; la segunda es que debe hacer que el *sender* disminuya el flujo de información en tránsito al detectar congestión. La pérdida de una unidad de información puede considerarse un indicador de congestión pues la probabilidad que esta sea causada por daño del paquete es muy baja y menor al 1 % .Los eventos conocidos como ACK duplicado y *timeout retransmit* permiten identificar que ha ocurrido la pérdida de un paquete por lo que son utilizados por *Congestion Avoidance* como indicio de congestión. *Congestion Avoidance* es un algoritmo independiente de *Slow Start*, ambos son usados ante la ocurrencia de una pérdida, pero su objetivo es diferente y deben ser implementados en conjunto.

La forma en que *Congestion Avoidance* disminuye la tasa de paquetes en la red es mediante decremento multiplicativo de la ventana de congestión *cwn*d. Esto es así escogido porque el promedio del tamaño de las colas aumenta de manera exponencial cuando la red está sobreutilizada y para poder contrarrestarlo es necesario que el decremento sea del mismo tipo. Por otro lado, el incremento de *cwn*d es escogido como aditivo, pues un incremento multiplicativo es demasiado rápido y puede con alta probabilidad sobrestimar la capacidad disponible lo que produciría más costos que beneficios. De acuerdo a lo anterior, el tamaño de la ventana aumenta en un segmento por cada RTT y disminuye a la mitad en caso de detectar una pérdida. Así el crecimiento de *Congestion Avoidance* es lineal a diferencia de *Slow Start* que es exponencial. La variable *ssthresh* permite elegir cuál de los dos algoritmos será invocado. Si $cwnd < ssthresh$ se ejecutará *Slow Start* y si $cwnd > ssthresh$ entonces lo hará *Congestion Avoidance*.

Fast Retransmit surge de forma experimental en 1990 cuando se presentan algunas modificaciones al algoritmo *Congestion Avoidance*, entre ellas se menciona el envío de un ACK duplicado ante el arribo de un segmento no consecutivo a los recibidos. Varias son las causas posibles de un ACK duplicado, una de estas es la pérdida de un paquete. El objetivo de *Fast Retransmit* es detectar mediante ACKs duplicados cuando se ha producido un paquete pérdido y solicitar la retransmisión de este antes de que se cumpla el plazo del *retransmit timeout*.

Un ACK duplicado puede ser originado por reordenamiento de los paquetes de la red, por replicación de un ACK o un segmento además de por pérdida de un paquete. *Fast Retransmit* es capaz de distinguir cuando se trata de una pérdida imponiendo un umbral a la cantidad de ACKs duplicados, pues se asume que siendo la causa el reordenamiento deberían recibirse sólo uno o dos ACKs duplicados. El umbral se fija en 3 ACKs duplicados.

Fast Recovery es el algoritmo ejecutado para enviar nuevos paquetes después del reenvío del paquete perdido por *Fast Retransmit*, este ejecuta *Congestion Avoidance* en vez de *Slow Start*, pues los ACKs duplicados muestran que existe flujo de paquetes y que no se quiere reducir este de manera brusca. Esto permite que el *throughput* se mantenga alto ante congestión es moderada. *Fast Recovery* controla el flujo de paquetes hasta que se reciben ACKs no duplicados de nueva data. Este es implementado en conjunto con *Fast Retransmit*.

Posteriormente a que *Fast Retransmit* reenvía el paquete perdido, el algoritmo *Fast Recovery* fija el valor de *ssthresh* a la mitad de la ventana de congestión *cwn*d, luego iguala el valor de *cwn*d al de *ssthresh* e infla el tamaño de la ventana *cwn*d en un segmento por cada uno de los tres ACK duplicados de *Fast Retransmit* que han dejado la red. Además cada vez

que llega un ACK adicional , se aumenta *cwn*d en un segmento . Así si el tamaño de *cwn*d y *rwn*d lo permiten transmitirá un nuevo paquete. *Fast Recovery* finaliza cuando se recibe un ACK de data nueva decrementando el tamaño de *cwn*d hasta el valor de $ssthresh$.

3.1.1. Tahoe

Las primeras versiones de TCP [44] usaban un sistema de ACKs acumulativos que seguía el modelo *go-back-n* , requerían que se cumpliera el tiempo de *timeout retransmit* para reenviar un paquete. En algunos otros casos presentaron problemas de inestabilidad pues comenzaban las transmisiones inyectando una gran cantidad de paquetes.

Las implementaciones más modernas desarrollaron algoritmos que permitieron mejoras en la transmisión, como evitar esperar el *timeout retransmit* para la retransmisión de un paquete. Ejemplo de ello es la conocida como Tahoe que utiliza los algoritmos *Slow Start*, *Congestion Avoidance* y *Fast Retransmit*, y que consta de una versión de estimador de RTT modificado que es útil para establecer los tiempos de *retransmit timeout*.

Tahoe a diferencia de algoritmos TCP más antiguos es capaz de reenviar un paquete sin tener que esperar que expire el *retransmit timeout*. Esto es posible gracias a la función *Fast Retransmit* que distingue la ocurrencia de una pérdida por medio de ACKs duplicados. La evolución y variaciones aplicadas a *Fast Retransmit* dieron origen a otras versiones conocidas de TCP.

3.1.2. Reno

Este protocolo [45] conserva las mejoras que se hicieron en Tahoe, pero añade variaciones en su algoritmo *Fast Retransmit* agregando la función *Fast Recovery* que considera cada ACK duplicado como un paquete que dejó el canal, además evita que el canal quede vacío después de una pérdida y que sea necesaria la invocación de *Slow Start*.

Se ejecuta *Fast Recovery* cuando se alcanza el umbral de ACK duplicados recibidos, se retransmite un paquete y se reduce la ventana de congestión a la mitad. No se ejecuta *Slow Start* y en cambio la ventana se infla por el número de ACKs duplicados se hayan recibido y así *cwn*d queda representada por el valor $min(\alpha|owin, cwnd + ndup)$ donde αwin es la *advertised window* del receptor, *cwn*d es la ventana de congestión del receptor y $ndup$ el número de ACKs duplicados obtenidos. Se sale de *Fast Retransmit* con la llegada de ACKs de nueva data.

La ventajas de Reno debidas al algoritmo *Fast Recovery* descrito es que permite mantener el *throughput* en valores mayores que Tahoe y que es capaz de responder adecuadamente ante la pérdida de un paquete. Sin embargo, presenta dificultades cuando las pérdidas son múltiples.

3.1.3. New Reno

Este algoritmo es llamado New Reno [46] pues corresponde a una pequeña modificación del algoritmo Reno [47]. La primera implementación de Reno se hizo en 1990, este aplicó una nueva función conocida como *Fast Recovery* que se caracterizaba por permitir la retransmisión de un solo paquete sin tener que ejecutar *Slow Start* después de una pérdida manteniendo así alto *throughput*. New Reno es de hecho una modificación de *Fast Recovery* de la versión Reno que tiene como objetivo enfrentar la situación de pérdidas múltiples.

La diferencia principal con la versión anterior es que New Reno es capaz reaccionar ante un *partial ACK*, esto es, un ACK que confirma el recibo de una cierta parte de los paquetes pendientes, pero no de la totalidad de ellos. Recibir un *partial ACK* puede ser interpretado según J.Hoe [48] como una señal de que el paquete siguiente al ACK obtenido se ha perdido. Esta información es usada para solicitar la retransmisión de aquel segmento sin necesidad de esperar a que expire el *retransmit timeout*. Otra diferencia importante de New Reno es que el algoritmo no sale de la función *Fast Recovery* cuando recibe un *partial ACK* a diferencia de Reno.

El proceso de *Fast Recovery* comienza en New Reno cuando sucede *triple duplicate* ACK (triple ACK duplicado) y termina cuando se recibe un acuse de recibo para todos los paquetes pendientes antes del inicio del proceso. La modificación del algoritmo incluye la creación de una variable llamada *recover* que guarda el *sequence number* (número de secuencia) del segmento de más alto valor transmitido. Después de recibir un triple ACK duplicado se infla la ventana de congestión por cada uno de los segmentos que han dejado la red, se hace el mismo incremento si se reciben ACKs duplicados adicionales. En caso de lo permitan los tamaños de *cwn*d y *rwn*d entonces se envía nueva data. Cuando se recibe un ACK de nueva data: si este confirma el recibo de todos los paquetes incluido el valor *recover* entonces se decrementa *cwn*d y se sale de *Fast Recovery*, en cambio, si se trata de un partial ACK entonces se reenvía el paquete perdido y se desinfla la ventana por el número de paquetes cuyo recibo ha sido confirmado, se envían nuevos paquetes si el tamaño de la ventana de congestión lo permite.

Para evitar las ráfagas de paquetes en los algoritmos New Reno y SACK se utilizan una variable llamada *maxburst* que limita la cantidad de segmentos enviados, más allá de lo autorizado por *cwn*d. En New Reno este valor se fija en cuatro paquetes y restringe el flujo cuando la transmisión deja *Fast Recovery*. En Tahoe la variable *maxsburst* no es requerida pues *Slow Start* previene el envío excesivo de segmentos.

3.1.4. SACK

TCP Reno tiene mal desempeño cuando ocurren múltiples pérdidas en una ventana, pues como utiliza un sistema de ACKs acumulativos la información es escasa y solo tiene dos inconvenientes posibilidades de abordar el problema: esperar un RTT para solicitar el reenvío de cada paquete perdido o bien retransmitir un conjunto de paquetes aunque eso signifique retransmitir algunos que ya se han recibido correctamente.

TCP SACK [46] deriva de TCP Reno y ambos tienen mucho en común. TCP SACK mantiene todas las buenas cualidades de su predecesor. Usa los mismos algoritmos que TCP Reno para variar el tamaño de la ventana de congestión, también utiliza *timeout retransmit* como último recurso de recuperación y es capaz de afrontar adecuadamente el recibo de paquetes en desorden. Las modificaciones en TCP SACK buscan mejorar el rendimiento del protocolo en un escenario donde ocurren diversas pérdidas.

La principal diferencia de SACK con las versiones TCP anteriores es la capacidad que tiene de hacer saber al *sender* cuáles paquetes se han recibido exitosamente para que sean reenviados sólo aquellos que se han perdido. Esto lo logra mediante la utilización de bloques SACK que representan los bloques no contiguos de data obtenida por el *receiver*. Los bloques SACK se incluyen en los ACKs sin que cambie el significado original ni el algoritmo de los acuses de recibo.

Existen diversas implementaciones, la referida por Mathis et al. [49] fue diseñada como tal para reducir el impacto sobre el protocolo TCP por eso se aplica mediante dos opciones. La primera opción se conoce como *SACK-permitted* y corresponde a una opción de habilitación que se envía en un segmento SYN señalando que puede usarse la opción SACK al concretarse la conexión; la segunda se llama SACK *option* y se envía cuando el enlace ya se ha establecido y habilitado el permiso. SACK *option* permite enviar información adicional a través de los bloques SACK donde cada bloque identifica a un conjunto no contiguo de datos. Se usan dos enteros de 32 bits para definir un bloque SACK. La información transmitida por SACK pretende optimizar el reenvío de paquetes al receptor. Los segmentos que se reciben en SACK son marcados mediante un *flag* en la cola de reenvío para evitar su retransmisión.

Otra diferencia de SACK con las anteriores es el uso de la variable *pipe* que es una estimación de la cantidad de paquetes pendientes y que se utiliza para decidir cuando enviar un paquete de data nueva en *Fast Recovery.* Este valor se incrementa cuando se envía un paquete o se reenvía uno y se decrementa cuando se recibe un ACK duplicado.

Como se ha dicho, han sido publicadas distintas versiones de este método. La descrita en los párrafos anteriores data de 1996 y apareció en el artículo *"TCP Selective Acknowledment Options"* [49], está basada en un trabajo previo de Jacobson y Braden [10] y presenta algunas modificaciones al algoritmo conocido de SACK. Las variaciones en [49] tienen como objetivo hacer más fácil la implementación, estas se resumen en usar *sequence numbers* de 32 bit y enviar *selective acknowledgement* para la *data* recibida más recientemente lo que permite que la longitud de SACK *option* sea más pequeña. El trabajo de Jacobson y Braden [10] es una de las implementaciones más antiguas aparece en el trabajo titulado *"TCP Extension for Long-Delay Path"* del año 1988 que discute un conjunto de soluciones para los problemas ocasionados en canales de transmisión con grandes anchos de banda y largos *round-trip delays.* Se menciona en [10] que un motivo del desempeño deficiente de TCP bajo este escenario en la red es el uso de ACKs acumulativos que no aportan suficiente información de los paquetes bien recibidos ante lo cual se propone una implementación de SACK TCP. Posteriormente, en el 2000 es expuesta por Floyd et al. una extensión del algoritmo TCP SACK que describe cómo debe actuar SACK cuando se reciben paquetes duplicados, lo que permitiría deducir el orden de los paquetes que han llegado hasta el receiver fuera de orden y así evitar retransmisiones innecesarias, la extensión es compatible con las implementaciones anteriores de SACK. Más

recientenmente en el año 2012 Blanton et al. [50] expusieron un algoritmo de recuperación de pérdidas conservativo, que está basado en el algoritmo detallado en [46] y que se obtiene de un cambio aplicado a Fast Recovery.

Se han realizado numerosos estudios experimentales que avalan la robustez y eficiencia de SACK por sobre otras versiones de TCP como Tahoe o Reno [49] obteniendo mejores resultados de *throughput* incluso en casos de pérdidas en ráfaga, un trabajo destacado que muestra esto es el de Fall y Floyd [46] que compara el rendimiento en casos desde uno hasta cuatro paquetes perdidos en una ventana usando simulaciones. Además este protocolo ha sido usado en el diseño de diversos algoritmos de transporte NETBLT [51], XTP[52], RDP[53], NDIR [54] y VMTP[55].

3.1.5. Comparación de Desempeño TCP

La comparación de los desempeños de las diferentes versiones de TCP se describen en el trabajo titulado *"Simulation-based Comparison of Tahoe, Reno and SACK TCP"* [46]. Se realizaron simulaciones que permitieron observar las diferencias de comportamiento de los algoritmos bajo distinta cantidad de pérdidas desde uno a cuatro paquetes.

Se pudo establecer que al producirse la pérdida de un paquete Tahoe requiere de *Slow Start*, en cambio Reno, New Reno y SACK TCP pueden hacerlo suavemente mediante *Fast Recovery*. En este escenario TCP Reno tiene un desempeño óptimo y no hay diferencias con los comportamientos de New Reno y SACK. La simulación que modela la pérdida de dos paquetes mostró que Tahoe los recupera con la función *Slow Start*, TCP Reno logra recuperar los paquetes con dificultad, SACK y New Reno tienen una recuperación rápida y suave. En la simulación que examina el caso que en que se producen tres pérdidas, se observó como en el anterior que Tahoe se recupera a través de *Slow Start*, TCP Reno tiene una recuperación lenta pues debe esperar el tiempo de retransmisión. No se observan grandes diferencias entre los comportamientos de New Reno y SACK TCP que se recuperan suavemente. En el experimento que evalúa el comportamiento al ocurrir la pérdida de cuatro paquetes se observa que Tahoe recupera los paquetes mediante el algoritmo *Slow Start*, TCP Reno presenta problemas en su desempeño pues debe esperar el tiempo de retransmisión para reenviar paquetes. New Reno utiliza cuatro *round-trip times* para recuperar los paquetes en cambio SACK logra una recuperación rápida y suave.

En conclusión los algoritmos Reno, New Reno y SACK tienen un comportamiento similar en el caso de una pérdida. New Reno y SACK tienen mejor desempeño ante varias pérdidas, sin embargo, la diferencia de desempeño entre SACK y New Reno se incrementa con el número de paquetes pérdidos donde TCP SACK toma la ventaja.

3.2. Modelos TCP

El estudio y desarrollo de los modelos TCP [21] permite la utilización de estos en simulaciones, y mediciones con el fin de analizar su desempeño respecto de diferentes aspectos. Una de las motivaciones de la correcta modelación de los flujos de información TCP están relacionadas a posibilidad de determinar si un flujo no TCP puede ser considerado como *"fair share"* o *"TCP-friendly"*, conceptos que han sido estudiados en diversos trabajos. Existen dos tipos de modelos que son más utilizados:

- Modelo del área de segmento-tiempo.
- Modelo del proceso de Markov.

3.2.1. Modelo del Área de Segmento-Tiempo

El modelo de área segmento-tiempo corresponde a una representación del estado estacionario del *throughput* como función de la tasa de pérdidas y del *round trip time*. Este refleja el impacto del proceso *Fast Retransmit* y de las situaciones de *timeout* y es capaz de hacer una buena predicción en un amplio rango de tasas de pérdida.

En una primera aproximación se considera que las pérdidas son solamente de tipo triple-ACK duplicado (TD, triple-duplicate) y que el tamaño de la ventana no está limitado por el *advertised window* del receptor. Se define N_t como el número de paquetes transmitidos en el intervalo $[0, t]$, y $B_t = N_t/t$ describe el *throughput* en ese intervalo. La cantidad B_t representa el *throughput* de la conexión pues se contabilizan todos los paquetes independiente de si fueron correctamente recibidos. A partir de este es posible inferir el valor del *throughput* B como el estado estacionario de B_t según se indica en la ecuación 3.1.

$$B = \lim_{t\to\infty} B_t = \lim_{t\to\infty} \frac{N_t}{t} \tag{3.1}$$

Se nombra la probabilidad de que se pierda un paquete como p y se busca obtener la relación $B(p)$. Se establece el periodo TDP como el intervalo de tiempo entre dos pérdidas de clase TD. Para cada periodo i-ésimo Y_{i} representa el número de paquetes enviados en el periodo A_{i}, de allí se puede deducir que B correponde a la ecuación 3.2.

$$B = \frac{E[Y]}{E[A]} \tag{3.2}$$

El primer paquete perdido en TDP_{i} es denominado como α_{i} y el viaje en que la pérdida sucedió, X_{i}, lo que se puede apreciar en la figura 3.3. Después de que esta ocurre $W_{\mathrm{i}} - 1$ paquetes son enviados antes de que otra tenga lugar y, por lo tanto, en la ronda $X_{\mathrm{i}} + 1$ se transmitirán $Y_{\mathrm{i}} = \alpha_1 + W_{\mathrm{i}} - 1$ segmentos. De lo anterior se deriva el resultado de la ecuación 3.3.

$$E[Y] = E[\alpha] + E[W] - 1 \tag{3.3}$$

Si se supone que $\{\alpha_{\mathrm{i}}\}_{\mathrm{i}}$ es un proceso aleatorio donde α_{i} es el total de paquetes enviados antes que se produzca una pérdida de tipo TD y que los eventos de la serie son independientes e

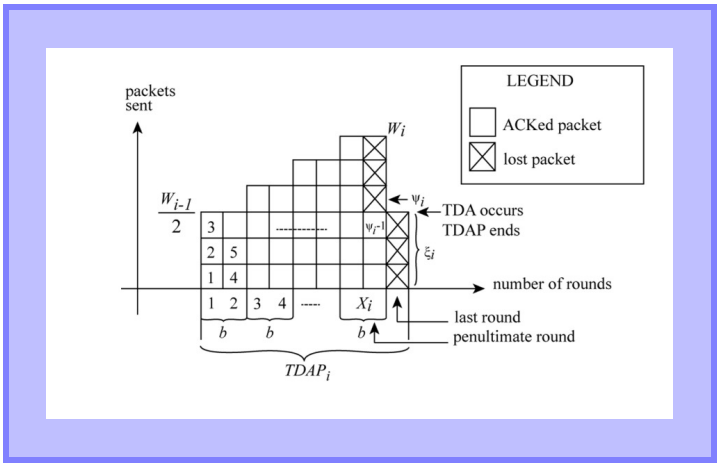


Figura 3.3: Modelo Área Segmento-Tiempo [21]

identicamente distribuidos entonces se puede establecer que la probabilidad de que $\alpha_{\mathrm{i}} = k$ es igual a la probabilidad indicada en la ecuación 3.4, lo que corresponde a la probabilidad de que k-1 paquetes sean recibidos correctamente.

$$P[\alpha = k] = (1-p)^{k-1} p \tag{3.4}$$

De lo anterior, se desprende que el promedio de α es lo que se expresa en la ecuación 3.5.

$$E[\alpha] = \sum_{k=1}^{\infty} (1-p)^{k-1} p = \frac{1}{p} \tag{3.5}$$

Sustituyendo el término 3.5 en la ecuación 3.3 se obtiene 3.6.

$$E[Y] = \frac{1-p}{p} + E[W] \tag{3.6}$$

Se puede obtener el promedio del periodos TDP_{i}, que también se denotan por A_{i} mediante la expresión (3.7) donde se calcula el producto entre $E[r]$, que representa el promedio de los *round trip times* o RTT, y la media del número de rondas por cada periodo.

$$E[A] = (E[X] + 1)\, E[r] \tag{3.7}$$

La relación entre $E[X]$ y $E[W]$ proviene del modelo de crecimiento de W_{i} respecto de X_{i}, se puede inferir desde la figura 3.3 y se expone en 3.8. Allí b representa el número de segmentos cuya llegada es confirmada mediante un ACK.

$$E[W] = \frac{2}{b} E[X] \tag{3.8}$$

El promedio de todos los paquetes enviados en los periodos TDP_{i} se puede obtener sumando el área bajo la curva en la gráfica de figura 3.3, es decir, haciendo la adición del área triangular $\frac{1}{2}\frac{E[W]}{2}E[X]$ y el área rectagular $\frac{E[W]}{2}E[X]$.

$$E[Y] = \frac{E[W]}{2}E[X] + \frac{1}{2}\frac{E[W]}{2}E[X] + E[\beta] \tag{3.9}$$

En la igualdad anterior β representa el número de paquetes enviados en la última ronda como se aprecia en la figura 3.3. Se considera que β es una variable uniformemente distribuida en el intervalo $[0, W_{\mathrm{i}}]$ de lo que se desprende que $E[\beta] = \frac{E[W]}{2}$. Reemplazando (3.6) en (3.9) resulta (3.10).

$$\frac{1-p}{p} + E[W] = \frac{E[X]}{2}\left(\frac{E[W]}{2} + E[W]\right) + E[\beta] \tag{3.10}$$

Sustituyendo (3.8) y $E[\beta] = \frac{E[W]}{2}$ en (3.10) y resolviendo la ecuación cuadrática se obtiene 3.11.

$$E[W] = \sqrt{\frac{8}{3bp}} + o(1/\sqrt{p}) \tag{3.11}$$

En otras palabras, para pequeños valores de p se cumple que $E[W] \approx \sqrt{\frac{8}{3bp}}$. Así mismo como se calculó $E[W]$ es posible deducir $E[X]$ resultando 3.12.

$$E[X] = \sqrt{\frac{2b}{3p}} + o(1/\sqrt{p}) \tag{3.12}$$

Los valores $E[W]$ y $E[X]$ permiten derivar $B(p)$. Utilizando las ecuaciones (3.2), (3.6) y (3.7) se logra el resultado mostrado en 3.13.

$$B(p) = \frac{\frac{1-p}{p} + E[W]}{E[A]} = \frac{\frac{1-p}{p} + E[W]}{(E[X]+1)\,RTT} \tag{3.13}$$

Sustituir en la ecuación (3.13) las expresiones calculadas para $E[X]$ y $E[W]$ proporciona el valor del *throughput* en segmentos por segundo para pequeños valores de p como se observa en 3.14.

$$B(p) = \frac{1}{RTT}\sqrt{\frac{3}{2bp}} + o(1/\sqrt{p}) \tag{3.14}$$

Este resultado es muy importante para la investigación porque permite calcular teóricamente para un rango de pérdidas el *throughput* de una transmisión lo posibilita hacer comparaciones con otros protocolos y simulaciones.

3.2.2. Modelos Estocásticos

El protocolo TCP ha llegado a ser considerado el protocolo de transporte más usado de internet [56], una cantidad considerable del tráfico en internet se realiza mediante este algoritmo [57]. Numerosas investigaciones se han realizado para analizar diferentes aspectos de este, uno de ellos consiste en la observación del desempeño en condiciones de pérdida en redes. Pues se ha podido observar mediante experimentos que no todas las pérdidas son provocadas por desbordamiento de un *buffer* [58]. Varios trabajos relacionados se encargaron de examinar el comportamiento de los protocolos TCP mediante modelos estocásticos.

Cadenas de Markov

Una cadena de Markov $[X_n, n = 1, 2, \dots]$ es un proceso [59] que toma un número finito o contable de valores posibles. Así el proceso estará caracterizado por un conjunto de enteros y se dirá que el proceso en el tiempo n se encuentra en el estado i ($X_n = \mathrm{i}$). La probabilidad de que el proceso que se encuentra en el estado i pueda pasar al estado j se denota por $P_{\mathrm{i}j}$. En forma extendida esto se expresa

$$P\{X_{n+1} = j | X_n = \mathrm{i}, X_{n-1} = \mathrm{i}_{n-1}, \dots, X_1 = \mathrm{i}_1, X_0 = \mathrm{i}_0\} = P_{\mathrm{i}j} \tag{3.15}$$

Para todos los estados $\mathrm{i}_0, \mathrm{i}_1, ..., \mathrm{i}_{n-1}, \mathrm{i}, j$ y todos los $n > 0$ Las probabilidades $P_{\mathrm{i}j}$ son no negativas, debido a esto y a que el proceso hará alguna transición de estado, entonces se cumple la siguiente relación.

$$P_{\mathrm{i}j} \geq 0, \quad \mathrm{i}, j \geq 0; \quad \sum_{j=0}^{\infty} P_{\mathrm{i}j} = 1, \quad \mathrm{i} = 0, 1, ... \tag{3.16}$$

La matriz de transición de un paso queda de la forma siguiente

$$\left\| \begin{array}{cccc} P_{00} & P_{01} & P_{02} & \cdots \\ P_{10} & P_{11} & P_{12} & \cdots \\ \vdots & \vdots & \vdots & \\ P_{\mathrm{i}0} & P_{\mathrm{i}1} & P_{12} & \cdots \\ \cdots & \cdots & \cdots & \end{array} \right\| \tag{3.17}$$

La probabilidad de que un procesos en estado i pase a un estado j en n transiciones se denomina probabilidad de n-pasos lo que denota $P_{\mathrm{i}j}^n$. En forma extendida esto queda

$$P_{\mathrm{i}j}^n = P\{X_{n+k} = j | X_k = \mathrm{i}\}, \quad n \geq 0, \mathrm{i}, j \geq 0 \tag{3.18}$$

Las equaciones de Chapman-Kolmogorov permiten calcular esta probabilidad a través de la siguiente expresión

$$P_{\mathrm{i}j}^{n+m} = \sum_{k=0}^{\infty} P_{\mathrm{i}k}^n P_{kj}^m, \quad \text{for all } n, m \geq 0, \quad \text{all i}, j \tag{3.19}$$

En la ecuación 3.19 el factor $P_{\mathrm{i}k}^n P_{kj}^m$ es la probabilidad de que el proceso pase desde el estado i al estado j en $n + m$ transiciones y pasando por un estado intermedio denominado k, por los tanto la suma de todas las probabilidades pasando por todos los estados intermedios k posibles representa la probabilidad de la transición de n-pasos. Se define la matriz de transiciones de n pasos como $P^{(n)}$ esta cumple la relación

$$P^{(n+m)} = P^{(n)} P^{(m)} \tag{3.20}$$

La propiedad en 3.20 puede ser demostrada mediante inducción.

Para un estado i en el que comienza una cadena de Markov se define la probabilidad de que el proceso vuelva a entrar en el mismo estado i como f_1. Se denomina como recurrente el estado i si $f_{\mathrm{i}} = 1$ y como transiente si $f_{\mathrm{i}} < 1$, es decir un estado recurrente es aquel que si comienza en el estado i hará infinitas transiciones sobre sí mismo. Se dice que un estado es

positivamente recurrente si el tiempo que demora en retornar al estado i es finito. Se puede demostrar que aunque los estados sean recurrentes no positivos si la cadena de Markov es de estado finito entonces sus estados recurrentes serán positivos recurrentes. Los estados positivos recurrentes y además aperiódicos también son llamados ergódicos y cumplen con el teorema que dice que si una cadena de Markov ergódica e irreducible cuyo límite $lim_{n\to\infty}P_{ij}^{n}$ existe y es independiente de i y si se llama a la probabilidad así $\pi_j = lim_{n\to\infty}P_{ij}^{n}$ entonces π_j es la única solución no negativa de

$$\begin{array}{c} \pi_j = \sum_{i=0}^{\infty} \pi_i P_{ij}, \quad j \geq 0, \\ \sum_{j=0}^{\infty} \pi_j = 1 \end{array} \tag{3.21}$$

Entre algunos ejemplos de modelos estocásticos se encuentra el presentado por Mishra, Sangui y Tripathi que considera pérdidas aleatorias [60]. Allí analizaron el funcionamiento del algoritmo propuesto por Jacobson y Karels [44], lo que les permitió llegar a la conclusión que un 1 % de pérdidas aleatorias de paquetes produce una disminución significativa del *throughput*. De la misma forma posteriormente Lakshman y Madhow [61] estudiaron el desempeño de TCP/IP en casos donde el producto retardo-ancho de banda es alto comparado con el *buffering* y cuando ocurren pérdidas aleatorias debida a congestión transiente por fluctuaciones en tiempo real del tráfico o por *wireless links* en la conexión. Se estudiaron dos versiones de TCP, la versión Tahoe y la versión Reno que incluye el proceso conocido como *Fast Retransmit*. A diferencia de trabajos anteriores este pone hincapié en los efectos detallados analíticamente de ciertos parámetros. Esto permitió obtener como resultados que TCP Reno produce menos tráfico en ráfaga que TCP Tahoe, pero es más vulnerable a efectos de fase. Ambas versiones presentan desventajas al enfrentarse a pérdidas aleatorias que repercuten reduciendo el *throughput* drásticamente especialmente cuando el producto de la probabilidad de pérdida y el cuadrado del retardo-ancho de banda son grandes, por ejemplo cercanos a diez o más. En transmisiones con alto *bandwidth-delay* el *throughput* se ve todavía más deteriorado cuando el *delay* es mayor estableciéndose una relación inversamente proporcional.

3.3. Algunas investigaciones, y propuestas de soluciones para redes con alto *bandwidth delay product*

En esta sección se revisan algunas publicaciones que plantean métodos para estudiar el comportamiento de TCP y mejorar el rendimiento en condiciones de *high bandwidth delay product*. Se repasan técnicas diversas y que, por lo tanto, se pueden relacionar a distintos grupos de los considerados en las clasificaciones a que se ha hecho referencia al principio de este capítulo. Los trabajos presentados en 3.4.1, 3.4.2 y 3.4.3 corresponden a modelamientos estocásticos para analizar el comportamiento de TCP.

Por otro lado, en 3.5.1 se resume el planteamiento de algoritmo conocido como RED, este utiliza el *gateway* y, por lo tanto, es un protocolo de tipo *router-supported*. Las investigaciones que se describen en 3.5.2 y 3.5.3 si bien son modelamientos estocásticos, analizan algoritmos de control de congestión basados en manejo del encolamiento en *routers* y que, por lo tanto,

podrían clasificarse dentro del tipo *router-supported* en vez de *end-to-end* como es TCP. En los modelos se usan ecuaciones de fluidos y teoría de control respectivamente.

En 3.6.1 se exponen las diferencias entre control de congestión, evitación de congestión y control de flujo. En 3.6.2 se resume un algoritmo que utiliza teoría de control y en particular el predictor de Smith para realizar el control de paquetes. Este método utiliza un nodo intermedio y, por lo tanto, no se puede clasificar como *end-to-end*. En 3.6.3 se muestra de manera breve una técnica de control de flujo basada en teoría de control. El último, como el anterior, también utiliza nodos intermedios por lo que se agrupa en la clase *router-supported*.

Finalmente, en 3.7.2 3.7.1, 3.7.3 3.7.4 y 3.7.5 se presentan algoritmos de tipo *end-to-end*. El primero es uno de los algoritmos más antiguos que se desarrollaron para enfrentar los problemas en redes *high badwidth delay product* y consiste en una modificación a TCP llamada TCP-SACK. Los restantes, como el primero, también se caracterizan en que utilizan las pérdidas de paquetes para cuantificar la congestión en la red. Además tienen en común entre sí que modifican el algoritmo AIMD de TCP para reaccionar a la congestión y obtener mejor utilización.

3.4. Ejemplos de modelamientos estocásticos para análisis de TCP

Los trabajos descritos en 3.4.1, 3.4.2 y 3.4.3 corresponden a modelamientos estocásticos para analizar el comportamiento de TCP y cómo cambia el throughput. En 3.4.1 se considera un escenario que presenta pérdidas aleatorias, se modela cómo cambia la ventana de congestión en el tiempo y se calculan las probabilidades de los estados lo que permite deducir la matriz de transición de Markov y la distribución de probabilidad. A partir de lo cual se logró estudiar como cambia el throughput según la probabilidad de pérdidas. En 3.4.2 se modelaron los distintos algoritmos de TCP que son utilizados en sus diferentes versiones pudiendo estudiar el rendimiento de cada version de TCP mediando el modelamiento. Finalmente en 3.4.3 se menciona un tipo de modelamiento estocástico basado en teoría de fluidos. Es importante considerarlo pues este tipo de modelamiento es la base de algunas propuestas planteadas basadas en teoría de control.

3.4.1. Análisis de control de flujo en redes con pérdidas

Mishra et al. [60] estudiaron el comportamiento TCP en dos diferentes casos, uno en presencia de pérdidas aleatorias y el otro en ausencia de estas, este análisis se realizó en el entorno de conexión TCP más simple que consiste en una transmisión entre dos *hosts* y se asume que no hay tráfico entre los *gateways* intermedios.

En condiciones ideales donde no son descartados paquetes aleatoriamente se producirá una pérdida cuando el tamaño de la ventana que incrementa su tamaño en las etapas de *Slow Start* y *Congestion Avoidance* alcance la capacidad máxima del canal de transmisión.

Durante *Slow Start*, es decir, mientras *cwn*d esta debajo de *ssthresh*, el tamaño de la ventana se duplica tras haberse confirmado el recibo de todos ellos. Se definen los conceptos de época y ciclo, un ciclo es el intervalo de tiempo entre dos pérdidas, en cambio una época es el tiempo trascurrido mientras una ventana recibe los acuses de recibo para cada uno de sus paquetes, de ahí que un ciclo esta conformado por varias épocas. De acuerdo a lo anterior al final de la época j-ésima en *Slow Start* la ventana tomará el valor 2^j. Y si se establece que el valor *ssthresh* cumple la siguiente condición:

$$2^k + x - 1 \leq ssthresh < 2^k + x \quad , k \geq 0\,. \tag{3.22}$$

entonces en *Congestion Avoidance* el tamaño de la ventana al final de la j-ésima época corresponde a $2^k + x + (j - k - 1)$.

El modelo descrito por Mishra primero requiere determinar la cantidad de paquetes enviados en cada época, lo cual se resume en la siguiente ecuación por partes:

$$n(j) = \begin{cases} 2^{j-1}, & si \quad 1 \leq j \leq k \\ x, & si \quad j \leq k+1 \\ 2^k + x + (j - k - 2) & si \quad j > k+1\,. \end{cases} \tag{3.23}$$

La función que precisa la cantidad de paquetes acumulados entre las épocas 1 a la $j - 1$ es:

$$N(j) = \sum_{\mathrm{i}=1}^{j-1} n(\mathrm{i})\,. \tag{3.24}$$

Al analizar el caso donde existen pérdidas aleatorias se puede notar que el tamaño de la ventana decrece cuando ocurre que un paquete no llega a destino aun cuando esta no haya alcanzado el máximo posible, además el tamaño de *ssthresh* se reduce a la mitad, ambos hechos repercuten en el *throughput* promedio resultando en una significativa reducción en la velocidad de transmisión.

El escenario que considera pérdidas puede ser modelado como una cadena de Markov si se asume que no hay pérdidas múltiples en una época pues el valor de la ventana W^j, que representa el tamaño de la ventana al final de un ciclo j, está relacionada solamente al valor de *ssthresh* que es la mitad de W^{j-1}. Para que la cadena de Markov quede definida debe obtenerse la matriz de transición que la caracteriza, esta se logra averiguando la probabilidad de que la ventana llegue a tomar el valor i al final del ciclo j-ésimo, es decir $P[W^j] = \mathrm{i}$. Las probabilidades dependen de la etapa en que se encuentre el protocolo de transmisión: *Slow Start* o *Congestion Avoidance*.

En *Slow Start* para que la ventana sea i el primer paquete perdido en la época α-ésima tiene que ser el paquete x-ésimo y se cumple que:

$$\begin{gathered} 2^{\alpha-1} \leq \mathrm{i} \leq 2^{\alpha}\,, \\ \mathrm{i} = 2^{\alpha-1} + x - 1\,. \end{gathered} \tag{3.25}$$

La probabilidad de que este evento ocurra se expresa en 3.26, donde se asume que p es la probabilidad de que un paquete sea aleatoriamente descartado.

$$p \cdot (1-p)^{N(\alpha)} \cdot (1-p)^{x-1} \tag{3.26}$$

En *Congestion Avoidance* se obtiene una ventana de tamaño i en el final de un ciclo si se pierde un paquete en la época α-ésima tal que se cumplan las expresiones en 3.27.

$$\begin{array}{c} 2^k + x - 1 \leq ssthresh < 2^{k+1} \quad , 1 < x < 2^k \\ \mathrm{i} = 2^k + x + (\alpha - k - 2) \end{array} \tag{3.27}$$

La probabilidad de que lo anterior acontezca corresponde a la expresión:

$$(1-p)^{N(\alpha)} p * \sum_{z=0}^{\mathrm{i}-1} (1-p)^z \,. \tag{3.28}$$

Por último, también es analizado el caso donde la ventana alcanza en la α-ésima época la capacidad máxima C como consecuencia de la ausencia de pérdidas aleatorias. Esto queda matemáticamente representado como:

$$\begin{array}{l} 2^k + x - 1 \leq ssthresh \leq, \quad 1 < x < 2^k \,, \\ C = 2^k + x + (\alpha - k - 2) \,. \end{array} \tag{3.29}$$

La correspondiente probabilidad de la situación supuesta está expresada por la ecuación 3.30.

$$(1-p)^{N(\alpha)} * \left[p * \sum_{z=0}^{C-1} (1-p)^z + (1-p)^C \right] \tag{3.30}$$

Mediante la matriz de transición de la cadena de Markov fue posible obtener la distribución de probabilidad y con ello analizar el comportamiento de la ventana ante diferentes probabilidades de pérdida. Se pudo constatar comparando lo obtenido por medio de simulaciones y los valores derivados del modelo, que este último provee resultados que se aproximan a los del experimento cuya diferencia es debida a que en el modelo no fueron consideradas multiples pérdidas en una misma ventana. También se observó que a mayor probabilidad de pérdidas aleatorias disminuye el tamaño promedio de la ventana de congestión durante la transmisión y que con sólo un 1 % de pérdidas el tamaño de la ventana en cierto escenario puede llegar a alcanzar el 40 % de su capacidad.

Para resolver estos inconvenientes los investigadores propusieron un esquema que tiene la propiedad de distinguir si la pérdida es causada por la congestión del enlace o por aleatoriedad. Para poder discriminar entre ambos casos se usaron dos caracteríscas distintivas de una transmisión: las pérdidas por congestión raramente son únicas en una ventana y, por el contrario, la pérdidas aleatorias dificilmente son más de una en una ventana si la probalidad de pérdida es pequeña. La variante del protocolo actúa tal como el original cuando la pérdida es identificada como de congestión, en cambio, cuando esta es aleatoria sólo retransmitirá el paquete dejando la ventana intacta. Se obtiene como resultado de esta propuesta el aumento del *throughput* promedio de la transmisión

3.4.2. Análisis comparativo del desempeño de distintas versiones de TCP en una red local con pérdidas

En 1998 Kumar presentó el trabajo [62] en el que a través de un modelo estocástico se estudia el desempeño del *throughput* de varias versiones adicionales TCP [62]: Tahoe,

OldTahoe, Reno y NewReno. El análisis está basado en dos investigaciones ya mencionadas en este apartado [61] y [60], al igual que en ellas se considera la estructura cíclica del proceso que permite modelarlo como una cadena de Markov donde cada ciclo de TCP comienza en el momento en que empieza la recuperación de las pérdidas del ciclo anterior. El objetivo de determinar un modelo detallado del protocolo TCP es poder pronosticar el desempeño del algoritmo ante valores diferentes de los parámetros considerados y comparar los efectos que las variaciones tienen con trabajos experimentales [63], [64], [65], [66] . Con los modelos analíticos se intenta medir el *throughput* de las cuatro versiones de TCP a través de procesos de *renewal-reward* de Markov. Entre las características estudiadas y consideradas en la construcción del modelo se encuentran los algoritmos *Slow Start*, *Fast Retransmit* y *Fast Recovery*, el efecto de *timeouts* altos y, por otro lado, se investigó el resultado de reducir el número de ACKs duplicados para desencadenar la ejecución del algoritmo *Fast Retransmit*, además se asumieron pérdidas aleatorias y no correlacionadas donde el *throughput* es función de la probabilidad de pérdida. Se diferencia de los trabajos de Mishra [60] y Lakshman [61] en que se intenta examinar el modelo con tanta exactitud como sea posible, incluyéndose en la investigación aspectos no tomados en cuenta en los anteriores, los cuales abarcaron en su análisis solo versiones más antiguas de TCP. *Fast Retransmit* se puede nombrar como ejemplo de una función no observada previamente cuyo estudio permitió establecer las ventajas de su uso en la reducción de la frecuencia de *timeouts* grandes.

Antes de describir los modelos obtenidos es necesario mencionar ciertas generalidades acerca del *receiver* y *transmitter* . Todas las versiones estudiadas tienen como supuesto el mismo proceso en el *receiver*. El *receiver* es el encargado de enviar los ACK para cada bloque de información correctamente recibido, estos son acumulativos lo que quiere decir que un ACK de valor n confirma el recibo de los $n-1$ paquetes anteriores y a la vez indica el número de paquete esperado . El *receiver* permite además recibir paquetes desordenados y almacenarlos en un TCP *buffer*, se define una variable llamada tamaño de ventana máximo W_{max} que limita la cantidad de paquetes en espera que no han confirmado recibo. Por otro lado, el *transmiter* maneja una serie de variables que es importante tener en cuenta para el modelamiento, entre las que se encuentran:

- $A(t)$ que representa el límite inferior de la ventana donde toda la información con numeración menor y hasta $A(t)-1$ ha sido transmitida y confirmado su recibo.
- $W(t)$ corresponde a la ventana de congestión cuyos paquetes son numerados con n de tal forma que se cumple la siguiente relación $A(t) \leq n < A(t)+W(t)$ y además el tamaño de la ventana no puede superar el máximo $W(t) < W_{max}$.
- $W_{th}(t)$ también llamada *Slow Start threshold* controla los incrementos de $W(t)$. Si se cumple que $W(t) < W_{th}$ entonces se está ejecuta el algoritmo de *Slow Start* y $W(t)$ se incrementa en uno, en cambio si $W(t) \geq W_{max}$ entonces se está en la etapa *Congestion Avoidance* y la ventana es incrementada en $1/W(t)$

Si se le llama A y M a los valores finales de las variables $A(t)$ y $W(t)$ respectivamente, es decir, A es la cantidad de paquetes recibidos y confirmados y M, el tamaño de la ventana de pérdida entonces la secuencia del último paquete transmitido antes del paquete descartado es $A+M-1$.

Del análisis de todos los algoritmos considerados se puede ver que tienen en común un

comportamiento cíclico donde se presenta una alternacia entre periodos de transmisión de paquetes y etapas de recuperación de paquetes. Un ciclo es definido como la etapa entre dos épocas donde t_k denota la época donde comienza el ciclo. Se asume que se inicia el traspaso de paquetes en el tiempo $t_0 = 0$ y que el tamaño de la ventana en ese instante es $W(0) = 1$ y $W_{th}(0) = W_{max}/2$. Se establece que k-ésimo ciclo es el intervalo $(t_{k-1}, t_k]$ y se le llama l_k a la época perteneciente al k-ésimo ciclo en la que ocurre la primera pérdida de un paquete. Para $k \geq 1$ se define $U_k = W(l_k)$ el tamaño de la ventana en la época l_k del k-ésimo ciclo en que ocurre un descarte de paquete.

Al estudiar el comportamiento de U_k para cada versión de TCP se puede ver, como se mostrará en los párrafos siguientes, que U_{k+1} depende probabilisticamente de U_k de lo que se desprende que $\{U_k\}$ es una cadena de Markov de tiempo discreto cuyo espacio de estado es $\{1, 2, \ldots, W_{max}\}$. Conocer el comportamiento de U_k y de la variable estacionaria aleatoria U asociada a U_k permite derivar una expresión para el *throughput* que es es el objetivo de la investigación [62].

TCP-OldTahoe y Tahoe tienen en común que comienzan cada nuevo ciclo en *Slow Start*. Y se diferencian en que Old-Tahoe espera un *timeout* para recuperar los paquetes y Tahoe, en cambio, es capaz de retransmitir el paquete perdido tras la ocurrencia de un triple ACK duplicado, esto es, antes de que ocurra un *timeout*. Si se considera el k-ésimo ciclo y que la pérdida ocurre en la época l_k entonces si, además de los $U_k - 1$ de la ventana del paquete perdido, se reciben K paquetes con éxito, en Tahoe se gatillará *Fast Retransmit*. Para ambas versiones de TCP, U_k define los valores que toman las variables $W(t_k^+)$ y $W_{th}(t_k^+)$ como se indica en 3.31, de lo que se deriva que U_{k+1} depende de U_k.

$$W(t_k^+) = 1 \quad y \quad W_{th}(t_k^+) = \tfrac{U_k}{2} \tag{3.31}$$

TCP-Reno y New Reno tienen implementado el algoritmo *Fast Retransmit* tal como TCP-Tahoe pero difieren con este en la etapa de recuperación, en la que el *transmitter* retransmite sólo el paquete pérdido. Si se supone que K ACKs duplicados ocurren en la época t_0 el *transmitter* define los valores de las variables $W(t)$ y $W(t)_{th}$ como en la ecuación 3.32.

$$W(t_0^+) = \tfrac{M}{2} + K \qquad W_{th}(t_0^+) = \tfrac{M}{2} \tag{3.32}$$

Ambas versiones se diferencian de Tahoe y Old-Tahoe en que después de la pérdida pueden ejecutar *Fast Retransmit* y *Fast Recovery* recuperando los paquetes mediante *Congestion Avoidance*, o en caso contrario, si no se gatilla *Fast Retransmit*, por medio del ya conocido *Slow Start*. Luego de que se pierde un paquete en el ciclo k-ésimo, existe la probabilidad de que el proceso entre en periodo de *Fast Recovery*, la que se denota F_{U_k}, que esta posibilidad se concrete determina el tamaño de las variables W y W_{th} en el siguiente ciclo. Si sucede *Fast Retransmit* el siguiente ciclo comenzará con *Congestion Avoidance* y los tamaños de las ventanas serán:

$$W(t_k^+) = W_{th}(t_k^+) = \frac{U_k}{2}\,. \tag{3.33}$$

O bien, es posible con probabilidad $1 - F_{U_k}$ que no se ejecute *Fast Recovery* pasando en el siguiente ciclo a una fase *Slow Start* donde las ventanas tomarán los siguientes valores:

$$W(t_k^+) = 1 \qquad W_{th}(t_k^+) = \tfrac{U_k}{2}\,. \tag{3.34}$$

Para cada version se puede calcular el *throughput* T_{ver} del proceso de renovación-recompensa de Markov se obtiene de la expresión 3.35 donde el sumando $\frac{1-p}{p}$ representa los paquetes enviados antes de la pérdida , $(1-p)(EU_{ver}-1)$ corresponde a los enviados luego de esta, donde C_{ver} es el periodo promedio de ciclo y U_ver se refiere a la variable aleatoria estacionaria del proceso $\{U_k\}$.

$$T_{ver} = \frac{\frac{1-p}{p} + (1-p)(EU_{ver} - 1)}{C_{ver}} \tag{3.35}$$

En el cálculo del *throughput* son necesarios el periodo promedio de ciclo y la distribución estacionaria de la cadena de Markov $\{U_k\}$ como se puede notar en la ecuación 3.35, en los párrafos siguientes se expone cómo fueron deducidos por los investigadores expresiones de la cadena de Markov $\{U_k\}$ de la que luego se puede derivar la distribución estacionaria y del promedio de ciclo para cada una de la versiones de TCP estudiadas.

Para el estudio de la cadena de Markov U_k para cada versiones de TCP lo primero que se debe obtener es la matriz de transición de probabilidades que es necesaria para encontrar la distribución estacionaria de la cadena de Markov. Para ello se define $U_k = M$ tal que M $\in \{1, 2, 3, \ldots, W_{max}\}$ y se busca precisar $P\{U_{k+1} = j | U_k = M\}$ para $1 \leq j \leq W_{max}$. La probabilidad de que el próximo ciclo, consecuencia de una pérdida, comience con *Slow Start* se establece como $\bar{f_M} = 1 - f_M$, donde f_M representa la probabilidad de comenzar en la fase de *Congestion Avoidance*. Además de denota $\bar{p} = 1 - p$ y $m = \frac{M}{2}$.

Con el fin de la matriz de transición se consideran los distintos valores que podría tomar U_{k+1} y su probabilidad de ocurrir, c.p significa con probabilidad. El primer caso supone que j alcanza un valor menor estricto que m lo que quiere decir que el ciclo se encuentra necesariamente en *Slow Start*, para que la ventana llegue a tomar ese valor partiendo desde 1, es necesario que se envíen correctamente $j-1$ paquetes. El segundo caso posible es que la ventana incremente su valor superando m, estando este en el intervalo $[m, W_{max})$ lo que puede suceder comenzando el ciclo con *Slow Start*, o bien, con *Congestion Avoidance*. Si se comienza con *Slow Start* entonces para llevar a una ventana a tamaño m, es decir $U_{k+1} = m$, deben enviarse $m-1$ paquetes correctamente y no todos los siguientes m que aumentarían en uno la ventana pues se ha pasado a *Congestion Avoidance*. Si la ventana $U_{k+1} = m+1$ entonces se deberán enviar sin pérdidas los primeros $m-1$, los siguientes m que incrementarán en uno la ventana, pues se ha pasado a la etapa de *Congestion Avoidance*, y alguno de los siguientes $m+1$ debe fallar para que la ventana se mantenga en el valor anterior. La generalización de lo previo se muestra en las ecuaciones 3.36 y 3.37, donde se especifican las probabilidades de ocurrencia de cada posibilidad y los paquetes que deben enviarse sin problemas, respectivamente.

$$U_{k+1} = \begin{cases} j, & c.p. & \bar{f}_M \bar{p}^{j-1} p \\ & para & 1 \leq j \leq m-1 \\ m+k, & c.p. & \bar{f}_M \bar{p}^{\mathrm{d}(k)}(1-\bar{p}^{m+k}) + f_M \bar{p}^{h(k)}(1-\bar{p}^{m+k}) \\ & para & 0 \leq k \leq (W_{max}-1-m) \\ W_{max} & c.p. & \bar{f}_M \bar{p}^{\mathrm{d}(W_{max}-m)} + f_M \bar{p}^{h(W_{max}-m)} \end{cases} \tag{3.36}$$

$$\begin{aligned} \mathrm{d}(k) &= (m-1) + m + (m+1) + \ldots + (m + (k-1)) \\ &= (k+1)\left((m-1) + \tfrac{k}{2}\right) \\ h(k) &= 0 + m + (m+1) + \ldots + (m + (k-1)) \\ &= \mathrm{d}(k) - (m-1) \end{aligned} \tag{3.37}$$

El valor de f_M, variable que representa la probabilidad de comenzar el siguiente ciclo en *Congestion Avoidance* cuando ha ocurrido una pérdida en la ventana de congestión de tamaño M, depende de la versión de TCP y del tamaño de la ventana. En los siguientes párrafos se exponen los resultados de esta variable para cada versión y cómo fueron calculadas por los investigadores.

Los casos en que es más simple de estudiar la magnitud que alcanza f_M son TCP-OldTahoe y TCP-Tahoe. En ambas versiones se comienza cada ciclo con *Slow Start*, por lo tanto, no hay posibilidad que tras una pérdida en el siguiente ciclo se ejecute *Congestion Avoidance* y entonces $f_M = 0$ para $1 \leq M \leq W_{max}$.

A diferencia de las anteriores versiones, para TCP-Reno resultan distintos valores de f_M según el valor de M. Cuando $1 \leq M \leq K$ no es posible que K ACKs se reciban pues $K > M - 1$ y entonces no se puede desencadenar *Fast Retransmit* , por lo tanto, $f_M = 0$. Al considerar el caso donde $M - 1 \geq K$, si K ACKs llegan al receiver se retransmitirá el paquete perdido tomando las ventanas los valores $W_{th} = \frac{M}{2} = m$ y $W = W_{th} + K$. Si sólo se ha descartado un paquete entonces la retransmisión de este producirá que se confirme el recibo de todos los paquetes. En cambio, si hay más de una pérdida la retransmisión generará un ACK que indica el siguiente paquete que ha fallado y sólo se podrá gatillar una nueva retransmisión si K paquetes se enviaron con éxito después de la primera, pues eso permitirá que se generen los ACKs necesarios para desencadenar una segunda retransmisión, esto es así requerido pues se asume que se está utilizando un *transmiter* rápido que envió todos los paquetes de la ventana antes de que el primer paquete fuese retransmitido. Matemáticamente, la condición para que luego del primer reenvío el sistema tenga la capacidad de transmitir K ACKs duplicados está dado por la expresión 3.38.

$$m + n \geq M + K \tag{3.38}$$

Donde n representa el número de paquetes que se han recibido correctamente tras el paquete descartado. Ya se ha asumido que se está en el caso donde $M > K$, además debe cumplirse $n \leq M - 1$ pues se ha hecho el supuesto de al menos la pérdida de un paquete. Se puede ver que no es posible cumplir la desigualdad 3.38 si $m + (M - 1) < M + K$, pero si hay un único paquete descartado entonces se puede cumplir si $m + (M - 1) = M + K$. Además, reemplazando m por $\frac{M}{2}$ en $m + (M - 1) \leq M + K$ se obtiene que $M \leq 2(K + 1)$, por lo tanto, para M comprendido en el intervalo $K + 1 \leq M \leq 2(K + 1)$ la probabilidad f_M de que se invoque *Fast Recovery* corresponde a:

$$f_M = (1-p)^{M-1}(1-p)\,. \tag{3.39}$$

Esta representa a la probabilidad de enviar correctamentemente todos los paquetes después del que ha fallado y a que la retransmisión se realice con éxito. Para el resto de los valores de M que pertencen al siguiente intervalo $2(K + 1) \leq M \leq W_{max}$ se calcula la probabilidad

de que sucedan hasta dos pérdidas y se obtiene que f_M toma el valor en 3.40 .

$$\begin{aligned} f_M &= (1-p)^{M-1}(1-p) + Pr\{\text{una pérdida más en los M-1 paquetes}\}\cdot \\ &\quad Pr\{(M-2)-(M-m) \text{ paquetes se envíen correctamente}\}(1-p)^2 \\ &= (1-p)^{M-1}(1-p)+ \\ &\quad (M-1)p(1-p)^{M-2}(1-p)^{m-2}(1-p)^2 \end{aligned} \tag{3.40}$$

Al estudiar que magnitud tiene la varible f_M para la versión de TCP-New Reno se puede ver que comparte con TCP Reno el requerimiento de K ACKs duplicados para desencadenar *Fast Retransmit* y *Fast Recovery*. Entonces en el intervalo $1 \leq M \leq K$,

$$f_M = 0\,. \tag{3.41}$$

En cambio, si M satisface la expresión $K+1 \leq M \leq W_{max}$,

$$f_M = \sum_{n=K}^{M-1} \binom{M-1}{n} (1-p)^n p^{M-1-n}\,. \tag{3.42}$$

A partir de la matriz de transición de la cadena de Markov $\{U_k\}$ se puede calcular a través de métodos tradicionales la distribución estacionaria de la cadena de Markov que se denota como u_M.

También este trabajo aborda la tarea de calcular el promedio del periodo de ciclo o *mean cycle time*. Un ciclo de tiempo fue definido como $(t_{k-1}, t_k]$ que para su estudio se divide en dos partes $(t_{k-1}, l_k]$ durante el que se realizan transmisiones correctamente y $(l_k, t_k]$ que corresponde al periodo donde ocurre la recuperación, este a su vez contiene otras dos partes, en la primera se reenvían los paquetes pendientes y los paquetes perdidos y la segunda se alcanza si el proceso no fue capaz de realizar *Fast Recovery* y sobrepasa el tiempo de *timeout*. Se define el periodo de ciclo promedio como $\gamma_{\mathrm{i},j}^{(m)}$ para los intervalos $1 \leq j \leq W_{max}$ y $0 \leq \mathrm{i} \leq j$, donde j representa la ventana, m es el *Slow Start threshold* e i son los paquetes encolados en el *transmitter*, este se calcula con diferentes expresiones dependiendo de la version de TCP que corresponda. Para el cálculo del periodo de ciclo promedio se utiliza la distribución estacionaria de la cadena de Markov u_M.

En las versiones TCP Old-Tahoe y Tahoe cada ciclo comienza con Slow Start y la expresión para el tiempo de ciclo promedio es el expresado en la ecuación 3.43.

$$\sum_{M=1}^{W_{max}} u_M \gamma_{0,1}^{\lceil \frac{M}{2} \rceil} \tag{3.43}$$

Para las variantes de TCP-Reno y New Reno suponiendo que la pérdida ocurre cuando la ventana de congestión es de tamaño M, el próximo ciclo comienza con la ventana en el tamaño uno o $m = \frac{M}{2}$ con probabilidad $1 - f_M$ y f_M respectivamente para *Slow Start* y *Congestion Avoidance*, la expresión para el promedio de tiempo de ciclo correspondiente se muestra en la ecuación 3.44.

$$\sum_{M=1}^{W_{max}} u_M \left(f_M \gamma_{0,\lceil \frac{M}{2} \rceil}^{\lceil \frac{M}{2} \rceil} + (1-f_M)\gamma_{0,1}^{\lceil \frac{M}{2} \rceil} \right) \tag{3.44}$$

En base a los modelos obtenidos en este trabajo se pudo obtener resultados que muestran cómo afecta la ejecución de *Fast Retransmit* en evitar *timeouts* y cómo repercuten los distintos valores *timeout* en el desempeño de los protocolos. En efecto Tahoe, Reno y New Reno mejoran su rendimiento hasta la probabilidad 0.01 y desde ahí disminuyen debido a *timeouts*. En particular TCP Reno se ve impactado con probabilidades de pérdida superiores a 0.02 ya que TCP Reno no es capaz de funcionar efectivamente ante multiples pérdidas pues requiere K ACKs adicionales para invocar a *Fast Recovery* confirmándose los resultados de los autores K.Fall y S.Floyd en [46].

Además, se pudo concluir que el desempeño de Old Tahoe es relativamente pobre incluso para probabilidades inferiores a 0.001 y que este cae llegando el *throughput* a 0.5 de la velocidad del canal cuando aumenta la probabilidad de pérdida a 0.01, este resulado es mejor que el que obtuvieron los autores de [61] pues ellos consideraron grandes *delays* de propagación.

3.4.3. Modelación de TCP basada en modelos de fluidos

Existen algunas tecnicas adicionales que si bien son menos frecuentes, pueden dar resultados más precisos en determinados casos. Entre estos se pueden mencionar los trabajos de Misra, Gong y Towsley [67], [68] que estudian TCP modelándolo como un fluido y obteniendo una ecuación diferencial que lo describe, relacionado con estos estan los trabajos de Hollot, Misra, Towsley y Gong [69] que utilizan la teoría de control para modelar la transmisión de paquetes por medio de una técnica conocida como *Random Early Detection* (RED).

Modelación de ecuación estocástica

En 1999 Misra, Gong y Towsley desarrollaron un modelo diferente a los existentes para el estudio de TCP, cuya principal diferencial es la forma en que se modelan las pérdidas, esto se hace comúnmente asumiendo que los paquetes salen teniendo cierta probabilidad de pérdida p que puede ser constante o variable según el caso, en cambio, en este trabajo se esquematiza a TCP como fluido, poniendo a la red como la fuente de pérdidas, donde se transmiten indicadores de pérdida como un proceso de Poisson de velocidad λ lo que permite que el comportamiento de la ventana calce con una ecuación de Poisson estocástica impulsada por el contador de Poisson (PCSDE) [70].

La teoría que sustenta de esta formulación, es que las conexiones en Internet ocurren a través de *routers* o saltos, los paquetes son encolados en cada *router* o descartados si el *router* está lleno. Si el proceso de rebasamiento de los *routers* es ordenado y estacionario, estonces el proceso completo de congestión es la suma de estos y comienza a asemejarse a un proceso de Poisson mientras mayor sea el número de saltos según el teorema de Khinchine [71]. Se presentan distintos tipos de pérdidas entre las que se puede nombrar triple ACK duplicado y pérdidas por *timeout* que se asumen como llegadas de Poisson con diferentes velocidades $\lambda_{TD}\lambda_{T0}$.

Antes de comenzar con el modelado se deben tener en cuenta las estructuras necesarias para hacer la formulación, se debe considerar un proceso de Poisson N con velocidad de llegada λ definido como en 3.45.

$$\begin{aligned} \mathrm{d}N &= \begin{cases} 1, & llegada\,de\,Poisson \\ 0, & en\,otro\,caso \end{cases} \\ E[\mathrm{d}N] &= \lambda \mathrm{d}t \end{aligned} \tag{3.45}$$

También se debe plantear una ecuación diferencial $\psi(x(t))$ de la forma 3.46.

$$\mathrm{d}x = f(x(t))\mathrm{d}t + \sum_{\mathrm{i}=1}^{m} g_{\mathrm{i}}(x(t))\mathrm{d}N_{\mathrm{i}} \tag{3.46}$$

La ecuación 3.46 según la regla de Ito también puede escribirse como 3.47.

$$\mathrm{d}\psi = \frac{\partial \psi}{\partial x} + \sum_{\mathrm{i}_1}^{m} (\psi(x + g_{\mathrm{i}}(x)) - \psi(x))\mathrm{d}N_{\mathrm{i}} \tag{3.47}$$

El comportamiento de la ventana de congestión puede ser expresado a través de la ecuación 3.48 cuyo segundo término representa el decremento multiplicativo y el tercero a la conducta relacionada a *timeouts* que se han considerado como singulares para simplificar el problema. El modelo planteado considera sólo *Congestion Avoidance* sin abarcar *Slow Start*, si bien es posible realizar su análisis este no se incluye pues no impacta en el resultado.

$$\mathrm{d}W = \frac{\mathrm{d}t}{RTT} + (-W/2)\mathrm{d}N_{TD} + (1 - W)\mathrm{d}N_{T0} \tag{3.48}$$

Se quiere determinar el comportamiento de la ventana, y por lo tanto, se aplica el valor esperado de la ecuación diferencial que representa a la ventana de congestión 3.47 resultando 3.48.

$$E[\mathrm{d}W] = E\left[\frac{\mathrm{d}t}{RTT}\right] + \left(-\frac{E[W]}{2}\right) E[\mathrm{d}N_{TD}] + (1 - [W])E[\mathrm{d}N_{T0}] \tag{3.49}$$

Luego al intercambiar las operaciones de valor esperado y diferencial se obtiene la ecuación 3.50.

$$\begin{aligned} \mathrm{d}E[W] &= \frac{\mathrm{d}t}{RTT} + \left(-\frac{E[W]}{2}\right)\lambda_{TD}\mathrm{d}t + (1 - E[w])\lambda_{T0}\mathrm{d}t \\ \frac{\mathrm{d}E[W]}{\mathrm{d}t} &= \frac{1}{RTT} + \frac{-E[W]}{2}\lambda_{TD} + \lambda_{T0} - E[w]\lambda_{T0} \\ &= \left(\frac{1}{RTT} + \lambda_{T0}\right) - \left(\frac{\lambda_{TD}}{2} + \lambda_{T0}\right) E[W] \end{aligned} \tag{3.50}$$

La ecuación diferencial anterior se resuelve y se obtiene la expresión 3.51

$$E[W] = \frac{\frac{1}{RTT} + \lambda_{T0}}{\frac{\lambda_{TD}}{2} + \lambda_{T0}} + C\mathrm{e}^{-\left(\frac{\lambda_{TD}}{2} + \lambda_{T0}\right)t} \tag{3.51}$$

De 3.51 se puede notar que si t tiende al infinito entonces el segundo término se anula y, por lo tanto, se tiene que la solución toma el valor de 3.52.

$$E[W] = \frac{\frac{1}{RTT} + \lambda_{T0}}{\frac{\lambda_{TD}}{2} + \lambda_{T0}} \tag{3.52}$$

A partir de esto se puede calcular el *throughput* R 3.53 dividiendo la ventana de congestión esperada por el valor de RTT.

$$R = \frac{1}{RTT} \frac{\frac{1}{RTT} + \lambda_{T0}}{\frac{\lambda_{TD}}{2} + \lambda_{T0}} \tag{3.53}$$

Para obtener un análisis más preciso se considera el tamaño máximo de la ventana en la formulación lo que tiene un efecto importante en la solución resultante. Esto se logra multiplicando el primer término de la ecuación diferencial 3.48 por un indicador $I_M(W)$ que es de la forma 3.54.

$$I_M(W) = \begin{cases} 1, & W < M \\ 0, & W = M \end{cases} \tag{3.54}$$

El indicador garantiza que la ventana detenga su incremento cuando alcanza el valor máximo M. Tras esta operación la ecuación diferencial queda según se muestra en 3.55.

$$\mathrm{d}W = I_M(W)\frac{\mathrm{d}t}{RTT} + \left(-\frac{W}{2}\right)\mathrm{d}N_{TD} + (1-W)\mathrm{d}N_{T0} \tag{3.55}$$

Aplicando el operador del valor esperado resulta 3.56.

$$E[\mathrm{d}W] = E[I_M(W)]\frac{\mathrm{d}t}{RTT} + E\left[-\frac{W}{2}\right]\lambda_{TD}\mathrm{d}t + (1-E[W])\lambda_{T0}\mathrm{d}t \tag{3.56}$$

Al intercambiar las operaciones de valor esperado y diferenciación se obtiene 3.57.

$$\begin{aligned} \frac{\mathrm{d}E[W]}{\mathrm{d}t} &= E[I_M(W)]\frac{1}{RTT} + \left(-\frac{\lambda_{TD}}{2}\right)E[W] + \lambda_{T0} - E[W]\lambda_{T0} \\ &= P[W < M]\frac{1}{RTT} + \left(-\frac{\lambda_{TD}}{2}\right)E[W] + \lambda_{T0} - E[W]\lambda_{T0}E[W] + \lambda_{T0} - E[W]\lambda_{T0} \end{aligned} \tag{3.57}$$

Tal como se resolvió la ecuación diferencial anterior se deriva que la solución de esta ecuación diferencial es 3.58.

$$E[W] = \frac{(1 - P[W = M])\frac{1}{RTT} + \lambda_{T0}}{\frac{\lambda_{TD}}{2} + \lambda_{T0}} \tag{3.58}$$

Para obtener el resultado completamente es necesario conocer el valor de $P[W = M]$ en función de los parámetros mencionados. Para ello los investigadores definieron la función:

$$V = M - W\,. \tag{3.59}$$

Esto se define así pues $V(t)$ corresponde a la función conocida como *Virtual Waiting Time* en una cola $M/G/1$ ya que las pérdidas fueron modeladas como llegadas de Poisson. Se calcula el trabajo promedio $\int_0^t V(u)\mathrm{d}u$ y las contribuciones que hacen a este los servicios de dos clientes supuestos que llegan a una cola hipotética. La parte del trabajo aportada por cada cliente se obtiene mediante construcciones geométricas. Con estos cálculos se obtienen relaciones que hacen posible despejar $P[W = M]$, como se muentra en 3.60, este valor permite derivar una expresión para el *throughput* en función del M, RTT,λ_{TD} y λ_{T0}.

$$P[W = M] = \frac{2\lambda_{T0}^2 + \lambda_{T0}\lambda_{TD} + 2\lambda_{T0}\frac{1}{RTT} + \left(\frac{1}{RTT}\right)^2 + 2\left(\frac{1}{RTT}\right)}{\left(\frac{1}{RTT} + 1\right)\left(2M\lambda_{T0} + M\lambda_{T0} + M\lambda_{TD} + 2K\right)} \tag{3.60}$$

Para obtener los resultados los investigadores utilizaron una muestra de datos de los obtenidos por J.Padhye V.Firoiu y D.Towsley en [4], y estimaron los valores de RTT y $T0$ tal como en el paper. Se estimaron los valores de λ_{T0} y λ_{TD} contando las pérdidas tramos de 100 segundos. Pudieron observar que los resultados del *throughput* son semejantes a los valores reales cuando el *throughput* es medianamente alto y bajo, en cambio para valores de *throughput* muy bajo o muy alto la aproximación no es buena, lo que se debe a que en esta circunstancia ocurren múltiples *timeouts* situación que no fue considerado en el modelo donde se asumieron *timeouts* singulares. En conclusión, los resultados obtenidos por los investigadores son cercanos a los reales pese a que fueron despreciados los efectos de etapas como *Slow Start* y *Fast Recovery* y se plantea por su simplicidad como una buena técnica para otro tipo de análisis como los protocolos de congestión *multicast*.

3.5. Propuestas basadas en modelos estocásticos de tipo router-supported

En esta sección se revisan algoritmos de tipo router-supported que si bien tienen una implementación más compleja tienen la ventaja de poder detectar la congestión más precisamente que los algoritmos de tipo end-to-end. El primero mencionado es el algoritmo conocido como RED 3.5.1, este utiliza el *gateway* y observa las colas lo que le permite identificar congestión desde el principio. Las investigacion que se resume en 3.5.2 es el Análsis de red de routers AQM que considera un conjunto de ecuaciones estocásticas de fluidos que modelan la transmisión de información y a través de la cual se analiza cómo afectan los parámetros del modelo en la estabilidad y la responsividad de la técnica AQM. Finalmente en 3.5.3 se hace un análisis de RED usando teoría de control que está basado en el modelo del trabajo anterior, haciendo ciertas simplificaciones. Se analizan los parámetros y se buscan relaciones usando la teoría de control para obtener valores de diseño que aseguren estabilidad.

3.5.1. RED definición

Es una técnica de control de congestión que fue por primera vez presentada en 1993 por Sally Floyd y Van Jacobson en el trabajo titulado "*Random Early Detection Gateways for Congestion Avoidance*" [72]. Este algoritmo es planteado ante la necesidad de obtener altos valores *throughput* pero manteniendo bajo el tamaño promedio de las colas en redes de alta velocidad y conexiones con gran *delay bandwith product*. Limitar a valores bajos los tamaños de cola permite que el *delay* promedio de la red también esté acotado. TCP por sí solo no es capaz de alcanzar los objetivos mencionados en redes de alta velocidad pues sólo detecta congestión después de una pérdida.

En la época existían métodos para detectar congestión que carecían de retroalimentación del *gateway*, estos lograban identificarla midiendo características de la red como el tiempo estimado de servicio en el cuello de botella, los cambios en el *throughput*, el retardo *end-to-end*, o bien, el total de paquetes perdidos. Sin embargo, el mejor método para detectar la congestión es el que ocurre en el *gateway* pues tiene una visión más amplia que le confiere

la posibilidad de observar las distintas conexiones que pasan a través de él. Estas tienen diferentes *round-trip times*, tolerancia al *delay*, *throughput*, etc. Una importante ventaja de los mecanismos de control de congestión implementados sobre el *gateway* es que funcionan aun cuando no todos los *gateways* en Internet apliquen el mismo esquema.

DECbit es uno de los primeros algoritmos de control de congestión que utiliza el *gateway* para identificarla y notificarla por medio de retroalimentación cuando el promedio del tamaño de la cola sobrepasa determinado umbral establecido. El método RED se diferencia de DECbit en que está enfocado a redes que son aptas para comunicar que existe congestión a través de una acción sobre un sólo paquete el que puede ser marcado o bien descartado, en cambio DECbit requiere contabilizar la cantidad de paquetes que han sido marcados respecto del total.

La técnica RED fue diseñada para usarse en redes que reaccionan ante la congestión y su enfoque general permite que pueda ser utilizado incluso con protocolos que no se caracterizan por ser cooperativos. Entre los protocolos que responden a la congestión puede aplicarse tanto a los que como TCP controlan el flujo monitoreando el tamaño de la ventana conocidos como *window-based* y también a los que, en cambio, hacen seguimiento a la velocidad, denominados *rate-based*.

El objetivo fundamental de RED es lograr control de congestión a través de la supervisión y control del promedio del tamaño de cola, otros objetivos también importantes son evitar la sincronización global y en caso de que el protocolo de transporte carezca de cooperación entonces deberá ser capaz de mantener el promedio de la longitud de la ventana bajo cierta cota superior.

En general, los algoritmos que se denominan de evitación de congestión o *congestion avoidance* apuntan a que el *throughput* alcance valores altos y que el *delay* permanezca bajo. Es conveniente que el esquema de control ocurra en el *gateway* pues es posible detectar la congestión desde el inicio al observar las colas en todo momento, además porque se puede dar seguimiento a los paquetes de cada fuente y avisar a la correspondiente en caso de notar congestión para que se realice una acción preventiva. Algunos de los medios que emplea RED para notificar que ha detectado congestión a una fuente son marcar un *bit* en el *header* del paquete, descartar un paquete o alguna otra señal que sea identificada por el protocolo de transporte en particular. Al avisar de la congestión a una fuente esta reducirá el tamaño de su ventana de congestión. Un problema que el algoritmo debe considerar y evitar es enviar la misma indicación a todas las fuentes, lo que se conoce como *global synchronization*, este fenómeno en el cual todas las ventanas reducen su tamaño simultáneamente, es muy perjudicial pues disminuye significativamente el *throughput*. Para prevenir que esta situación ocurra, el algoritmo RED implementa aleatoriedad al marcar los paquetes y decidir a cuáles conexiones notificará. Otro inconveniente que RED debe evitar es la distorsión o *bias* en caso de tráfico abultado o *bursty traffic*. Al igual que *global synchronization* esto se puede sortear mediante haciendo aleatorio el proceso de marcar los segmentos.

El algoritmo RED tiene dos partes principales en una se calcula el promedio del tamaño de la cola que determinará hasta qué nivel las ráfagas de paquetes serán admitidas en la cola del *gateway*, la segunda parte que lo conforma consiste en el cálculo de la probabidad con que se marcarán ciertos paquetes bajo ciertas condiciones de congestión, este proceso está

destinado a evitar ciertos problemas como *global synchronization* y distorsión y también se busca que el tamaño de la cola sea controlado apropiadamente.

El algoritmo RED usa un filtro pasabajo para calcular el promedio del tamaño de la cola con un sistema de pesos móviles, en inglés esto de designa como *exponencial weighted moving average* (promedio exponencial de pesos móviles, EWMA) y se define matemáticamente en 3.61.

$$avg \leftarrow (1 - w_q)avg + w_q q \tag{3.61}$$

Además, se establecen dos umbrales, uno máximo y otro mínimo contra los cuales se compara el promedio y se realizarán diferentes acciones en caso que este valor se encuentre sobre el máximo, bajo el mínimo, o bien, entre los umbrales. Cuando se cumple la primera condición todos los paquetes serán marcados, esto puede referirse concretamente a descartar los paquetes dependiendo del protocolo de transporte usado, dada esta situación ocurrirá que se mantendrá el promedio del tamaño de la cola cercano al umbral superior. Si se satisface la segunda condición ningún paquete será marcado, en cambio, si el valor del promedio se encuentra entre los umbrales entonces los paquetes se marcarán con la probabilidad p_a que está relacionado al tamaño promedio de la cola *avg* y que resulta ser aproximadamente proporcional al ancho de banda compartido en el *gateway*.

Análogamente al promedio *avg* que varía entre min_{th} y max_{th} la probabilidad fluctúa linealmente entre 0 y max_p tal como se indica en la ecuación 3.62.

$$p_b \leftarrow max_p \frac{(avg - min_{th})}{max_{th} - min_{th}} \tag{3.62}$$

La probabilidad del marcado de paquetes final aumenta lentamente con el incremento de la cuenta que se lleva respecto del último paquete marcado según la expresión 3.63. La forma elegida para el cálculo de la probabilidad es pertinente pues permite evitar que pase demasiado tiempo entre dos paquetes marcados.

$$p_a \leftarrow \frac{p_b}{1 - count \cdot p_b} \tag{3.63}$$

La investigación de Floyd y Jacobson también entrega ciertas pautas para escoger adecuadamente los valores de los parámetros del algoritmo como lo son w_q, min_{th} y max_{th}. El peso w_q corresponde a la constante de tiempo que caracteriza al filtro pasabajo que usa RED para medir el tamaño promedio de la cola. No se puede escoger demasiado grande el valor de w_q pues es en ese caso no se estará filtrando convenientemente el transiente del tamaño de la cola.

Para deducir cotas que limiten al parámetro w_q, se calcula el promedio de la cola al recibirse el L-ésimo paquete. Esto se obtiene a partir de la ecuación 3.61, que describe el cálculo del promedio de la cola del algoritmo RED, suponiendo que la cola está vacía en el comienzo y tiene un promedio igual a cero. Resulta la expresión 3.64 que permite establecer la relación 3.65 pues se aceptarán ráfagas de hasta L paquetes.

$$\begin{aligned} avg_L &= \sum_{\mathrm{i}=1}^{L} \mathrm{i} w_q (1 - w_q)^{l-\mathrm{i}} \\ &= w_q (1 - w_q)^L \sum_{\mathrm{i}=1}^{L} \mathrm{i} \left(\frac{1}{1-w_q}\right) \mathrm{i} \\ &= L + 1 + \frac{(1-w_q)^{L+1} - 1}{w_q} \end{aligned} \tag{3.64}$$

$$L + 1 + \frac{(1 - w_q)^{L+1} - 1}{w_q} < min_{th} \tag{3.65}$$

La ecuación 3.65 define una cota superior para el peso w_q dados los valores del mínimo umbral min_{th} y el parámetro L que representa la cantidad de paquetes en una ráfaga admisible. Por otro lado, se afirma que un valor demasiado bajo en el peso w_q puede llevar a que los cambios en la cola no se reflejen oportunamente en el valor avg calculado, en general se recomienda que $w_q \geq 0{,}001$. En la mayoría de las simulaciones se eligió $w_q = 0{,}002$.

Los valores min_{th} y max_{th} también deben ser escogidos con cuidado pues su magnitud afectará el desempeño del sistema. En el trabajo se sugiere que min_{th} debe fijarse con un valor alto para maximizar el poder de la red, que es el cuociente entre el *throughput* y el *delay*. El valor del parámetro min_{th} se debe seleccionar acorde al promedio de cola deseado y al nivel de tráfico en el *link*. Si el promedio de la cola resulta ser demasiado bajo, entonces se estará desaprovechando la capacidad del *link* y este quedará infrautilizado. Además la diferencia max_{th}-min_{th} se aconseja que sea elegida de modo que alcance un valor grande y mayor al incremento típico del promedio de la cola durante un *round-trip time* con el fin de evitar *global synchronization* y oscilaciones del promedio de la cola sobre max_{th}. Una regla de dedo que se entregó es elegir max_{th} al menos duplicando min_{th}.

La denominada probabilidad inicial p_b de marcar una paquete se calcula como función del promedio de la cola. En este trabajo se comparan dos métodos para calcular el valor de la probabilidad final de marcar paquetes cuando el promedio de cola se mantiene constante. Se muestran las características de cada método y las ventajas del escogido por sobre el otro. En el primer sistema se modela como una variable geométrica la cantidad de paquetes que llegan entre dos que han sido marcados, en cambio en el segundo esto se plantea con una variable aleatoria uniforme. Para el primer caso donde se expresa como X el número de paquetes no marcados entre dos que sí lo fueron, la ecuación 3.66 representa la probabilidad de que ocurra este evento. Para el mismo caso se calcula que la esperanza es $E(X) = 1/p_b$.

$$Prob[X = n] = (1 - p_b)^{n-1} p_b \tag{3.66}$$

Por otra parte, el modelo de variable X aleatoria uniforme de $[1, 2, \ldots, 1/p_b]$ se obtiene si la probabilidad de marcar un paquete es $p_b/(1 - count \cdot p_b)$ donde $count$ es el número de paquetes no marcados desde el último que lo fue. En ese caso se cumple la ecuación 3.67 y su esperanza es $E(X) = 1/(1p_b) + 1/2$.

$$Prob[X = n] = \frac{p_b}{1-(n-1)p_b} \prod_{\mathrm{i}=0}^{n-2} \left(1 - \frac{p_b}{1-\mathrm{i}p_b}\right) = p_b \quad para \quad n > \frac{1}{P_b} \tag{3.67}$$

$$Prob[X = n] = 0 \quad para \quad n > 1/p_b \tag{3.68}$$

El primer método resulta incoveniente pues marca muchos paquetes juntos y también deja otros con grandes intervalos entre sí, sin embargo se persigue justamente lo contrario, que exista regularidad entre los paquetes marcados. El método de variable geométrica puede causar *global synchronization* en el sistema y con ello reducir el tamaño de las ventanas y el *throughput*. Las simulaciones de RED permitieron observar que el algoritmo tiene mejor desempeño cuando la probabilidad de marcar un paquete cambia suavemente respecto de las variaciones en el promedio del tamaño de la cola, pues esto aminora las oscilaciones en la anterior y en la probabilidad de marcar un paquete.

Las simulaciones que se realizaron para esta investigación [72] mostraron que el algoritmo RED no presenta distorsión o *bias* para tráfico en forma de ráfagas. Este tipo de tráfico se puede dar en tráfico de vídeo de *bit-rate* variable, en tráfico interactivo y también en una conexión FTP con *long delay-bandwith product* usando un tamaño de ventana pequeña. Este último es el que se utiliza en el escenario experimental creado. Se pudo comprobar con una de estas pruebas que el protocolo *RED gateway* resulta exitoso controlando el promedio de la cola en respuesta a un cambio de carga dinámico, se observó que al incrementarse el número de conexiones también aumenta la frecuencia con la que el *gateway* descarta paquetes. Estas simulaciones también se realizaron utilizando otras técnicas de evitación de congestión como lo son *Drop Tail*, *Random Drop Gateways* y DECBit. Pudieron notar que *Drop Tail* y RED *gateway* mantienen el mismo tamaño del promedio de la ventana. Además observaron que con *Drop Tail* o *Random Drop Gateway* el nodo tiene una cantidad desproporcionada de paquetes perdidos y presentan *bias* para tráfico en ráfaga. Las simulaciones para DECbit también revelan *bias* o distorsión en el tráfico que puede ser corregido mediante retroalimentación selectiva.

En conclusión, *Ramdom Early Gateways* es un algoritmo de evitación de congestión relativamente sencillo que tiene buen rendimiento al combinar su acción con otros protocolos de transporte. Una caracterísca que lo define es que notifica congestión a una conexión con una probabilidad proporcional al ancho de banda compartido en el *gateway*. Este es exitoso evitando distorsión o *bias* en un tráfico en ráfagas y también logra prevenir *global synchonization* con lo que las múltiples conexiones no disminuyen simultáneamente el tamaño de su ventana favoreciendo al *throughput* del sistema.

3.5.2. Análisis de red de routers AQM

En el año 2000 Misra, Gong y Towsley presentaron un trabajo que trata acerca de ecuaciones diferenciales estocásticas utilizadas para modelar flujos en un sistema de *routers* de *Active Queue Management*, estos métodos que fueron propuestos en [73] y [72]. Este trabajo a diferencia del anterior considera un conjunto de ecuaciones diferenciales acopladas donde las pérdidas no son independientes del flujo de información.

El planteamiento requiere tomar en cuenta un *router* de capacidad C, que aplica la política de *Active Queue Management* (Manejo Activo de Colas, AQM), esta última queda descrita por la función de pérdida $p(x)$, donde x es un estimador del promedio de la longitud de la cola que se denomina $q(t)$. Si la política AQM es RED la función $p(x)$ está comúnmente definida por 3.69.

$$p(x) = \begin{cases} 0, & 0 \leq x < t^{min} \\ \frac{x - t^{min}}{t^{max} - t^{min}}, & t^{min} \leq x \leq t^{max} \\ 1, & t^{max} < x \end{cases} \tag{3.69}$$

Las funciones que corresponden al tamaño de la ventana TCP y al *round trip time* se denotan por $W_{\mathrm{i}}(t)$ y $R_{\mathrm{i}}(t)$ respectivamente para N flujos TCP i $= 1, \ldots, N$. Por otro lado, $B_{\mathrm{i}}(t)$ se define como el *throughput* instantáneo de un flujo de i-ésimo flujo. Acorde a lo anterior $R_{\mathrm{i}}(t)$ sigue la relación 3.70, donde a_{i} es un retardo de propagación fijo y $q(t)/C$ es el retardo del encolamiento. Además se debe tener en cuenta que se modelan las pérdidas de un flujo i-

ésimo como un proceso de Poisson $\{N_{\mathrm{i}}(t)\}$ con velocidad de tiempo variable $\lambda_{\mathrm{i}}(t)$ donde $N_{\mathrm{i}}(t)$ representa el número de pérdidas de cada flujo y $\lambda_{\mathrm{i}}(t)$ se puede relacionar con los esquemas de marcas que se presentan en AQM.

$$R_{\mathrm{i}}(t) = a_{\mathrm{i}} + q(t)/C \tag{3.70}$$

Usando las definiciones anteriores, el primer paso para el estudio del proceso es proponer la ecuación que describe la evolución del tamaño de la ventana $W_{\mathrm{i}}(t)$, esto es, la ecuación diferencial 3.71. La expresión refleja el comportamiento aditivo de TCP a través de su primer término y el multiplicativo por medio del segundo. Según el segundo sumando de 3.71 , la ventana es dividida a la mitad al momento de producirse una pérdida, es decir, cuando se cumple que $\mathrm{d}N_{\mathrm{i}}(t) = 1$.

$$\mathrm{d}W_{\mathrm{i}}(t) = \frac{\mathrm{d}t}{R_{\mathrm{i}}(q(t))} - \frac{W_{\mathrm{i}}(t)}{2}\mathrm{d}N_{\mathrm{i}}(t) \tag{3.71}$$

Al aplicar el operador valor esperado en la ecuación 3.71 se obtiene 3.72.

$$\begin{aligned} E[\mathrm{d}W_{\mathrm{i}}(t)] &= E\left[\frac{\mathrm{d}t}{R_{\mathrm{i}}(q(t))}\right] - \frac{E[W_{\mathrm{i}}(t)\mathrm{d}N_{\mathrm{i}}(t)]}{2} \\ \mathrm{d}E[W_{\mathrm{i}}] &\approx E\left[\frac{\mathrm{d}t}{R_{\mathrm{i}}(q)}\right] - \frac{E[W_{\mathrm{i}}]\lambda_{\mathrm{i}}(t)}{2}\mathrm{d}t \end{aligned} \tag{3.72}$$

La igualdad anterior corresponde a una aproximación pues las variables no son independientes en $E[W_{\mathrm{i}}(t)\mathrm{d}N_{\mathrm{i}}(t)]$ y, por lo tanto, no es igual a $E[W_{\mathrm{i}}(t)]E[\mathrm{d}N_{\mathrm{i}}(t)]$. El indicador de pérdida $\lambda(t)$ se atrasa un *round trip time* respecto de la marca o descarte de un paquete cumpliéndose las ecuaciones 3.73.

$$\begin{aligned} t &= R(q(t')) + t' \\ t' &= t - \tau \end{aligned} \tag{3.73}$$

El indicador de pérdida $\lambda(t)$ en el tiempo t es proporcional al ancho de banda, así cumple la relación $p(\bar{x})B(t-\tau)$ donde $B(t-\tau)$ representa el valor del *throughput* en el tiempo $t-\tau$. La variable B también puede ser representadoa de la forma $\bar{W}_{\mathrm{i}}(t-\tau)/R_{\mathrm{i}}(\bar{q}(t-\tau))$. Reemplazando en la ecuación 3.72 y asumiendo que $E[f(x)] \approx f(E[x])$ se obtiene la expresión 3.74.

$$\frac{\mathrm{d}\bar{W}_{\mathrm{i}}}{\mathrm{d}t} = \frac{1}{R_{\mathrm{i}}(\bar{q})} - \frac{\bar{W}_{\mathrm{i}}\bar{W}_{\mathrm{i}}(t-\tau)}{2R_{\mathrm{i}}(\bar{q}(t-\tau))}p(\bar{x}(t-\tau)) \tag{3.74}$$

Como segundo paso, se necesita encontrar la ecuación diferencial que representa el cambio en la variable promedio de la cola x. Se supone que x se calcula a través de un promedio con pesos móviles donde se pondera $q(t)$ por α cada δ segundos como se muestra en la ecuación 3.75 . El factor α debe cumplir la condición $0 < \alpha < 1$.

$$x((k+1)\delta) = (1-\alpha)x(k\delta) + \alpha q(k\delta) \tag{3.75}$$

La ecuación diferencial que podría calzar con la ecuación recurrente anterior es 3.76.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Ax(t) + Bq(t) \tag{3.76}$$

La solución de la ecuación diferencial es la ecuación 3.76

$$x(t_{k+1}) = \mathrm{e}^{A(t_{k+1}-t_k)}x(t_k) + \int_{t_k}^{t_{k+1}} \mathrm{e}^{A(t_{k+1}-\tau)}B\mathrm{d}\tau q(t_k) \tag{3.77}$$

Igualando los primeros términos de las expresiones 3.75 y 3.76 se encuentra la relación entre parámetros 3.78.

$$1 - \alpha = \mathrm{e}^{A\delta} \tag{3.78}$$

Entonces los valores de A y B cumplen la igualdad 3.79.

$$A = \frac{log_{\mathrm{e}}(1-\alpha)}{\delta} = -B \tag{3.79}$$

Reemplazando A y B en la ecuación 3.76 se obtiene la expresión 3.80.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{log_{\mathrm{e}}(1-\alpha)}{\delta}x(t) - \frac{log_{\mathrm{e}}(1-\alpha)}{\delta}q(t) \tag{3.80}$$

Si además se aplica el operador valor esperado en ambos lados de la ecuación diferencial 3.80 se obtiene la igualdad 3.81.

$$\frac{\mathrm{d}\bar{x}}{\mathrm{d}t} = \frac{log_{\mathrm{e}}(1-\alpha)}{\delta}\bar{x(t)} - \frac{log_{\mathrm{e}}(1-\alpha)}{\delta}\bar{q(t)} \tag{3.81}$$

Para terminar y completar el conjunto de ecuaciones que caracterizan al sistema se utiliza la ecuación de Lindley que representa el comportamiento de una cola q como se observa en 3.82. El primer sumando de la ecuación describe la disminución del largo de la cola, mientras esta es mayor a cero, a consecuencia del servicio de los paquetes, y el segundo término expresa el aumento del largo de la cola por la llegada de paquetes desde los N flujos.

$$\frac{\mathrm{d}q(t)}{\mathrm{d}t} = -1_{q(t)}C + \sum_{\mathrm{i}=1}^{N}\frac{W_{\mathrm{i}}}{R_{\mathrm{i}}(q)} \tag{3.82}$$

Al tomar el valor esperado de la ecuación 3.82 se obtiene 3.83.

$$\begin{array}{rl} \frac{\mathrm{d}\bar{q}(t)}{\mathrm{d}t} & = E[-1_{q(t)}]C + \sum_{\mathrm{i}=1}^{N} E\left[\frac{W_{\mathrm{i}}}{R_{\mathrm{i}}(q)}\right] \\ & \approx E[-1_{q(t)}]C + \sum_{\mathrm{i}=1}^{N}\frac{\bar{W}_{\mathrm{i}}}{R_{\mathrm{i}}(q)} \end{array} \tag{3.83}$$

Se puede aproximar $E[-1_{q(t)}]$ a uno para el caso de una cola con cuello de botella pues se cumple que $q(t) > 0$ con probabilidad cercana a uno.

$$\frac{\mathrm{d}\bar{q}}{\mathrm{d}t} \approx -C + \sum_{\mathrm{i}=1}^{N}\frac{\bar{W}}{R_{\mathrm{i}}(\bar{q})} \tag{3.84}$$

La solución de las ecuaciones acopladas 3.74, 3.82 y 3.84 se puede obtener mediante métodos numéricos, pues se tienen $N + 2$ ecuaciones y $N + 2$ incógnitas y permite conocer el comportamiento de la longitud de la cola, el tamaño de la ventana y su evolución en el tiempo.

Es posible generalizar el análisis anterior a la red, para ello se considera un conjunto V de *routers* y cada *router* v perteneciente a V está limitado a una capacidad C_v y tiene una

función de pérdida $p_v(x_v)$. La variable x_v representa el promedio de la cola del *router* v y forma parte un vector que se expresa como $\mathbf{x}$, análogamente $q_v(t)$ denota al tamaño de la cola de cada *router* y $\mathbf{q}$ al vector que conformado por los valores q_v. Además, δ_v y α_v describen un intervalo de muestreo y un factor de ponderación o peso de promedio, respectivamente, que se utilizan en el cálculo de x_v. La generalización de la ecuación 3.70 que describe el *round trip time* promedio para cada flujo i es 3.85.

$$R_{\mathrm{i}}(\mathbf{q},t) = a_{\mathrm{i}} + \sum_{v \in V_{\mathrm{i}}} q_v(t)/C_v \tag{3.85}$$

Para la extensión de la ecuación 3.74 se considera el vector de probabilidades de cada nodo $\mathbf{P}$ y la matriz $\mathbf{A}$ cuyas filas indican los flujos y sus columnas, las colas de los *routers*. El producto $\mathbf{AP}$ representa el producto entre las columnas de A y los elementos de P. La ecuación 3.74 se transforma a 3.86, donde la probabilidad de una pérdida en un flujo i se denota por $\hat{p}_{\mathrm{i}}(\bar{x})$.

$$\begin{aligned}\frac{\mathrm{d}\bar{W}_{\mathrm{i}}}{\mathrm{d}t} &= \frac{1}{R_{\mathrm{i}}(\bar{q}(t-\tau))} - \left(\frac{\bar{W}_{\mathrm{i}}\bar{W}_{\mathrm{i}}(t-\tau)}{2R_{\mathrm{i}}(\bar{q}(t-\tau))}\right) \\ &\quad \left(1 - \left(\prod(1 - \mathbf{AP}(\mathbf{x})_{\mathrm{i}})\right)\right) \\ &= \frac{1}{R_{\mathrm{i}}(\bar{q}(t-\tau))} - \left(\frac{\bar{W}_{\mathrm{i}}\bar{W}_{\mathrm{i}}(t-\tau)}{2R_{\mathrm{i}}(\bar{q}(t-\tau))}\right) \cdot \hat{p}_{\mathrm{i}}(\bar{x}(t-\tau))\end{aligned} \tag{3.86}$$

La ecuación que describe el promedio estimado de las longitudes de las colas se mantiene igual que en la formulación para un flujo y la ecuación de Lindley para F_v flujos que pasan a través de la cola v se modifica a 3.87.

$$\frac{\mathrm{d}\bar{q_v}}{\mathrm{d}t} = -1_{\bar{q}_v(t)C} + \sum_{\mathrm{i} \in F_v} \frac{\bar{W}_{\mathrm{i}}}{R_{\mathrm{i}}(\bar{q}_v)} \tag{3.87}$$

En la expresión 3.87 es posible hacer la aproximación $E[1_{q_v(t)}] \approx 1$ pues para una cola que es un cuello de botella se cumple que $q_v(t) > 0$ con probabilidad cercana a uno, en cambio, para una que no es cuello de botella $q_v > 0$ ocurre con probabilidad cercana a cero. Con las últimas se completa un conjunto de $N + 2|V|$ ecuaciones que se pueden resolver a través de métodos numéricos para obtener las funciones que describen el comportamiento de la ventana, las colas, etc. en un sistema con múltiples fuentes y flujos.

Para considerar los *timeouts* en el modelo se debe modificar la ecuación diferencial que caracteriza a la ventana de congestión. Se hace la diferencia entre pérdidas por *timeout* y por ACK duplicado que ocurren en el proceso. Se utiliza la función de pérdidas a $Q(w) = min(1, 3/w)$ que entrega la probabilidad que se pierda un paquete por *timeout*. En la formulación de la anterior se considera que todos los paquetes se pueden perder con la misma probabilidad y se asume a lo más una pérdida por vuelta. De acuerdo esto se pueden producir pérdidas por *timeout* si se descartan cualquiera de los últimos paquetes de la ronda. La ecuación diferencial de la ventana modificada queda como se muestra en 3.88.

$$\begin{aligned}\frac{\mathrm{d}W_{\mathrm{i}}}{\mathrm{d}t} &\approx \frac{1}{R_{\mathrm{i}}(q)} + (1 - Q(\bar{W}_{\mathrm{i}}))\left(-\frac{W_{\mathrm{i}}W_{\mathrm{i}}(t-\tau)}{2R_{\mathrm{i}}\bar{q}(t-\tau)}\right)\bar{p}_{\mathrm{i}}(\bar{x}(t-\tau)) \\ &\quad + (1 - \bar{W}_{\mathrm{i}})\frac{\bar{W}_{\mathrm{i}}(t-\tau)}{R\mathrm{i}_{\mathrm{i}}(\bar{q}(t-\tau))}Q(\bar{W}_{\mathrm{i}})\bar{p}_{\mathrm{i}}(\bar{x}(t-\tau))\end{aligned} \tag{3.88}$$

También se tomó en cuenta el caso donde los flujos son iguales, es decir, tienen la misma ruta y *round trip time* lo que da origen a un conjunto de ecuaciones más simples de resolver. Se consideran F clases de flujos que contiene a su vez n_j flujos idénticos donde j indica la clase del flujo, $R_j(q)$ es el *round trip time* correspondiente y V_j la ruta respectiva. La ecuación diferencial del promedio del tamaño de la cola que incluye la variación mencionada se presenta en 3.89.

$$\frac{\mathrm{d}\bar{q}_v(t)}{\mathrm{d}t} = -1_{\bar{q}_v} C_{\mathrm{i}} + \sum_{j \in F_v} n_j \frac{\bar{W}_j}{R_j(\bar{q}_v)} \tag{3.89}$$

El número de ecuaciones a resolver para el escenario analizado disminuye desde $\sum_{\mathrm{i}_1}^{F} n_{\mathrm{i}} + 2|V|$ a $F + 2|V|$ y, por lo tanto, es más fácil encontrar su solución requiriéndose menos tiempo de cálculo.

Los investigadores hicieron experimentos donde compararon los resultados de simulaciones con lo que se obtiene de la resolución de las ecuaciones diferenciales del modelo planteado. Observaron que los resultados son semejantes, pero difieren en el comienzo pues el modelo no incluye el impacto de *Slow Start*. A través de los experimentos también se pudo notar que el tamaño de las colas oscila en el tiempo. Las oscilaciones en el sistema son un comportamiento no deseado que se busca aminorar porque pueden aumentar el largo de las colas llegando a rebasar el *buffer*, también pueden producir la variablidad de los *delays* e incrementar el *jitter* en los *delays*. Todo esto tendría un impacto negativo sobre el *throughput* pues se acrecentarían las pérdidas. Uno de los motivos por los que se observa comportamiento oscilatorio en las colas está implícito en la ecuación 3.90que describe su comportamiento.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{log_{\mathrm{e}}(1-\alpha)}{\delta} x(t) - \frac{log_{\mathrm{e}}(1-\alpha)}{\delta} q(t) \tag{3.90}$$

Denominando $-K$ al término $log_{\mathrm{e}}(1-\alpha)/\delta$ resulta la ecuación

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -Kx(t) + Kq(t) \tag{3.91}$$

Al tomar la transformada de Laplace de la ecuación, se puede ver la función de transferencia de la ecuación es de la forma $\frac{K}{K+s}$ lo que corresponde a un filtro pasabajo cuya entrada en la longitud instantánea de la longitud de la cola q y la salida en es promedio estimado de la cola x. Un filtro pasabajo permite el paso de frecuencias bajas y amortigua las frecuencias altas respecto de un parámetro. En este caso la constante K indica la responsividad del filtro: frecuencias más bajas que K serán permitidas y más altas, atenuadas. Si el valor de K es alto se obtendrán notorias oscilaciones pues la función AQM seguirá a la longitud instantánea de la cola.

Los investigadores observaron que el periodo de muestreo efectivo δ tiene gran importancia en la estabilidad del sistema y, por lo tanto, en la sintonización de los parámetros de RED se busca establecer el valor adecuado de este. Se dieron cuenta el valor de δ está relacionado al de la capacidad del *link* y al promedio de tamaño de los paquetes. Otras variables que también afectan la estabilidad son α y los niveles de carga. Al aumentar la capacidad del enlace al triple, se provoca la disminución de δ el periodo de muestreo, esta a su vez causa el incremento de la constante K que se traduce en aumento en las oscilaciones del tamaño de la cola. Por el contrario, incrementar el valor de δ resulta en la estabilización del sistema, pero

si eleva demasiado se pueden causar tiempos de subida mayores y el aumento el sobrepaso inicial de la longitud de cola instantánea. Como se mencionó δ está también relacionado con el tamaño de los paquetes y la capacidad del enlace, otro forma de acrecentar su valor es aumentar el tamaño de los paquetes enviados. Se busca para establizar el sistema que δ sea independiente de la capacidad y del tamaño de los paquetes porque un algoritmo cuya estabilidad depende de este tipo de factores es vulnerable a usuarios maliciosos.

En el año 2000 Firou et. al [74] recomendaron que δ debería ser igual al *round trip time más* pequeño del enlace y además expusieron algunas pautas de elección de los parámetros de RED. También plantearon que las oscilaciones en RED se deben a la discontinuidad de la función de pérdida y deberían remediarse cambiando esta característica. Sin embargo, esto fue descartado en [68] realizando un experimento donde se eliminó la discontinuidad mostrando que el comportamiento oscilatorio se mantiene. Así se demostró que las causas de las oscilaciones son diversas y no sólo se relacionan a la discontinuidad de la función de pérdida de paquetes. A través de los experimentos también pudieron ver que existe un *trade off* entre la responsividad y la estabilidad en el control RED, pues se observó bajo ciertos parámetros se logra hacer el sistema estable pero este se vuelve lento en su respuesta en el tiempo.

Finalmente, se concluye que en [68] se desarrolló un método para modelar y obtener resultados numéricos del comportamiento de la técnica conocida como AQM, que además resultan similares a los obtenidos mediante simulaciones, demoran menos tiempo computacional y permitieron reconocer algunos problemas o dificultades que se tienen con el mecanismo de control RED.

3.5.3. Análisis de RED basado en teoría de control

El trabajo denominado "A control theoretic analysis of RED"[69] está basado en [68]. Las primeras investigaciones acerca de RED no eran concluyentes respecto de la sintonización de parámetros, este punto constituía, por lo tanto, un problema de la técnica a consecuencia de lo cual se propusieron muchas varientes de RED para subsanar esto. Pero al mismo tiempo hubo otros que intentaron de definir pautas para la elección de parámetros [74], los autores de [69] buscaron profundizar en esto y abordarlo de manera más formal a través de teoría de control utilizando el modelo que ya había sido obtenido en [68].

El análisis en [69] se fundamenta en el modelo no lineal que describe al método RED obtenido en [68], este es simplificado ignorando la etapa de *timeout* y además linealizado para poder aplicar las herramientas que se utilizan en la teoría de control y así diseñar un sistema estable. El modelo es de tipo estocástico diferencial y las ecuaciones no lineales acopladas que lo conforman se detallan en 3.92.

$$\begin{aligned} \dot{W}(t) &= \frac{1}{R(t)} - \frac{W(t)W(t-R(t))}{2R(t-R(t))} p(t-R(t)) \\ \dot{q}(t) &= \frac{W(t)}{R(t)} N(t) - C \end{aligned} \tag{3.92}$$

Donde $\dot{W}$ y $\dot{q}$ representan las derivadas de W y q respectivamente y las variables utilizadas se exponen a continuación.

$$\begin{array}{rcl} \mathrm{W} & = & \text{magnitud de ventana TCP esperada}, \\ \mathrm{q} & = & \text{longitud de la cola esperada}, \\ \mathrm{R} & = & \text{round-trip time}=\frac{q}{C}+T_p\,, \\ \mathrm{C} & = & \text{capacidad del enlace[packets/s]}\,, \\ T_p & = & \text{delay de propagacion[s]}\,, \\ \mathrm{N} & = & \text{factor de carga (número de la sesión TCP)}\,, \\ \mathrm{p} & = & \text{probabilidad de pérdida del paquete}\,. \end{array}$$

Al resolver las ecuaciones se debe tener en cuenta que W y q son cantidades acotadas por la capacidad del *buffer* $\bar{q}$ y el tamaño máximo de la ventana $\bar{W}$ respectivamente y, por lo tanto, se cumple que $q \in [0, \bar{q}]$ y $W \in [0, \bar{W}]$. También p está limitado $p \in [0, 1]$, pues representa a la probabilidad de pérdida.

La linealización se consigue asumiendo un punto de operación (W_0, q_0, p_0) tal que además se cumple $\dot{W} = 0$ y $\dot{q} = 0$. Obteniéndose las relaciones 3.93.

$$\begin{array}{rcl} \dot{W} = 0 & \rightarrow & W_0^2 p_0 = 2 \\ \dot{q} = 0 & \rightarrow & W_0 = \frac{R_0 C}{N} \end{array} \tag{3.93}$$

Se define R_0 en 3.94.

$$R_0 = \frac{q_0}{C} + T_p \tag{3.94}$$

Se realiza la linealización en torno al punto de operación encontrado y se asumen valores constantes para las funciones $R(t) = R_0$ y $N(t) = N$ y resultan las ecuaciones 3.95 y 3.96.

$$\delta\dot{W}(t) = -\frac{N}{R_0^2 C}(\delta W(t) + \delta W(t - R_0)) - \frac{R_0 C^2}{2N^2}\delta p(t - R_0) \tag{3.95}$$

$$\delta\dot{q}(t) = \frac{N}{R_0}\delta W(t) - \frac{1}{R_0}\delta q(t) \tag{3.96}$$

Se define δW, δq y δp en 3.97.

$$\begin{array}{rcl} \delta W & = & W - W_0 \\ \delta q & = & q - q_0 \\ \delta p & = & p - p_0 \end{array} \tag{3.97}$$

Otros autores también han abordado el modelado del mecanismo de control de la ventana de TCP. Por ejemplo, Mascolo [75] afirmó que el modelo TCP se puede representar utilizando una estructura conocida como regulador de Smith, pero esto no coincide con lo planteado por Hollot et. al, pues en ese caso el término que multiplica a δW debería ser $-\frac{C^2}{2N}\frac{1}{s+\frac{1}{R_0}}\mathrm{e}^{sR_0}$.

El sistema de ecuaciones considerado se puede simplificar asumiendo que se cumple la desigualdad 3.98.

$$\frac{N}{R_0^2 C} \ll \frac{1}{R_0} \tag{3.98}$$

De cumplirse 3.98 se puede despreciar el término e^{-sR_0} en la fórmula de la dinámica de la ventana. La desigualdad anterior se cumplirá siempre que $W_0 \gg 1$, esto porque la ecuación 3.99 siempre se cumple en el punto de operación.

$$\frac{N}{R_0^2 C} = \frac{1}{W_0 R_0} \tag{3.99}$$

Así si W_0 alcanza un valor mucho mayor que uno se puede ignorar el término que indica retardo y el sistema de ecuaciones a resolver se reduce a 3.100.

$$\begin{aligned} \delta\dot{W} &= -\tfrac{2N}{R_0^2 C}\delta W(t) - \tfrac{R_0 C^2}{2N^2}\delta p(t - R_0) \\ \delta\dot{q} &= \tfrac{N}{R_0}\delta W(t) - \tfrac{1}{R_0}\delta q(t) \end{aligned} \tag{3.100}$$

Los eigenvalores de la ecuación TCP linealizada y de la ecuación dinámica de la cola, en 3.100, son $\frac{-2N}{R_0^2 C}$ o $\frac{-2}{W_0 R_0}$ y $\frac{-1}{R_0}$. Que los valores sean negativos muestra que el estado de equilibrio es localmente asintóticamente estable.

$$\delta\dot{W} = -\lambda_0\delta W(t) - \frac{R_0 C^2}{2N^2}\delta p(t - R_0) \tag{3.101}$$

Teniendo las ecuaciones que describen la dinámica del sistema se diseña el controlador, que en este caso se denomina "la ley de control AQM". Los objetivos del compensador son obtener sistema de *loop*-cerrado estable, mantener una respuesta transiente aceptable y ser robusto ante las variaciones de los parámetros. Por lo tanto, el compensador se diseña de modo que guarde márgenes de estabilidad que aseguren buen funcionamiento en cierto dominio. Hay dos aspectos que deben ser considerados al momento de bosquejar el compensador: el margen de ganancia y el margen de fase. El margen de ganancia se puede asociar a la incerteza del nivel de carga N que admite el sistema. Y al margen de fase se relaciona al valor del retardo en el *round trip time* que puede tolerar el sistema antes de volverse inestable. Los márgenes de estabilidad se pueden comprobar observando el diagrama de Bode que caracteriza al sistema. El margen de ganancia corresponde a la magnitud de la respuesta donde la respuesta en fase es de -180°. El margen de fase ϕ_m corresponde a 3.102, donde w_{ph} es la respuesta en fase en el punto en que la magnitud es unitaria, es decir, vale 0 dB.

$$\phi_m = w_{pm} - 180 \tag{3.102}$$

La función de transferencia de la planta queda definida por $P(s) = P_{tcp}P_{queue}(s)$ en 3.105, donde P_{tcp} y P_{queue} están representadas en 3.103 y 3.104.

$$P_{tcp}(s) = \frac{\frac{R_0 C^2}{2N^2}}{s + \frac{2N}{R_0^2 C}} \tag{3.103}$$

$$P_{queue}(s) = \frac{\frac{N}{R_0}}{s + \frac{1}{R_0}} \tag{3.104}$$

$$P(s) = \frac{\left(\frac{C^2}{2N}\right)\mathrm{e}^{-sR_0}}{\left(s + \frac{2N}{R_0^2 C}\right)\left(s + \frac{1}{R_0}\right)} \tag{3.105}$$

Para estabilizar el sistema se propone una técnica de manejo de colas activo o AQM, en un modelo cerrado y retroalimentado. Este deberá encargarse de compensar, por ejemplo, la

eventual inestabilidad que produce la ganancia de alta frecuencia $\frac{C^2}{2N}$ de la planta $P(s)$, donde un incremento en el valor de N, el número de sesiones TCP, puede aumentar la respuesta oscilatoria. Otra variable, cuyas fluctuaciones debe cuidar el controlador es el retardo de propagación T_p. La estabilidad del sistema debe manterse independiente de las magnitudes que toman las variables mencionadas. Lo anterior constituye un propósito del controlador conocido como robustez que permite conservar el desempeño ante las variaciones de la red. Otras metas del controlador son la eficiencia en el retardo de la cola y utilización de la cola.

El retardo de la cola se refiere al tiempo necesario para que le paquete sea atendido y su valor se obtiene a través del cuociente $\frac{q}{C}$, esta cantidad junto con T_p, el retardo de propagación, configuran el retardo de la red. Se busca mantener bajos estos valores y por ello se procura que la longitud de las colas sean también reducidas, sin embargo, se debe evitar llegar al extremo de la infrautilización del enlace.

La eficiencia en la utilización significa que la cola, tal como se mencionó en el parráfo anterior, no debe ser infrautilizada o a permanecer vacía, pero tampoco debe rebasarse, pues esto tendrá como consecuencias pérdidas y retransmisiones que perjudicarán a la red.

Entre las estrategia AQM se pueden mencionar los tipos *tail-drop* y RED. *Tail-drop* es equivalente al control *on-off*, su desventaja que produce oscilaciones que pueden tener comportamiento caótico, en cambio, RED es capaz de evitar esto obteniendo estabilidad. El control implementado por Hollot et. al es de tipo RED y su función de transferencia está descrita por la ecuación 3.106.

$$C_{\text{red}}(s) = \frac{L_{\text{red}}}{s/K + 1} \tag{3.106}$$

Las constantes L_{red} y K corresponden a 3.107 y 3.108.

$$L_{\text{red}} = \frac{p_{max}}{max_{th} - min_{th}} \tag{3.107}$$

$$K = \frac{log_{\text{e}}(1-\alpha)}{\delta} \tag{3.108}$$

Con el fin de encontrar valores apropiados para L_{red} y K tales que permitan mantener la estabilidad se asumen cotas para el número de sesiones TCP N y para el *round-trip time* R_0, estas se denotan por N^- y R^+ respectivamente y cumplen las relaciones $R_0 \leq R^+$ y $N \geq N^-$. El sistema debe mantenerse estable para todo espectro de valores de N Y R_0 que consideran los intervalos delimitados por las cotas. Los investigadores Hollot et. al. propusieron que el controlador $C(s)$ es estable para todo $N \geq N^-$ y para todo $R_0 \leq R^+$ si los parámetros del controlador L_{red} y K cumplen la relación 3.109.

$$\frac{L_{\text{red}}(R^+C)^3}{(2N^-)^2} \leq \sqrt{\frac{w_g^2}{K^2} + 1} \tag{3.109}$$

donde

$$w_g = 0{,}1min\left(\frac{2N^-}{(R^+)^2C}, \frac{1}{R^+}\right). \tag{3.110}$$

Para demostrar que se cumple lo sugerido se debió probar que la función de transferencia de la planta compensada $L(jw)$, bajo las condiciones impuestas, satisface las relaciones $|L(jw_g)| \leq 1$ y $\angle L(jW_g) > 180^o$ En efecto, para mostrar que la función de transferencia guarda la primera relación, se tiene en cuenta que la ecuación que modela a la planta junto al controlador tiene la forma 3.111.

$$\begin{aligned} L(jw) &= C_{\text{red}}(jw)P(jw)\mathrm{e}^{-jwR_0} \\ &= \frac{L_{\text{red}}\frac{(R_0C)^3}{(2N)^2}\mathrm{e}^{-jwR_0}}{\left(\frac{jw}{K}+1\right)\left(\frac{\frac{jw}{2N}}{R_0^2C}+1\right)\left(\frac{jw}{\frac{1}{R_0}}+1\right)} \end{aligned} \tag{3.111}$$

La ecuación 3.111 se puede aproximar usando 3.110 obteniéndose la expresión 3.112 que es válida en el intervalo $w \in [0, w_g]$.

$$L(jw) \approx \frac{L_{\text{red}}\frac{(R_0C)^3}{(2N)^2}\mathrm{e}^{-jwR_0}}{\frac{jw}{K}+1} \tag{3.112}$$

Al tomar el módulo de la fracción 3.112 es posible afirmar que se cumple la ecuación 3.113 para $N \geq N^-$ y para $R_0 \leq R^+$. Usando la proposición planteada en 3.109 se comprueba que $|L(jw_g)| \leq 1$ para todo $N \geq N^-$ y para $R_0 \leq R^+$.

$$|L(jw_g)| \leq \frac{L_{\text{red}}\frac{(R^+C)^3}{(2N^-)^2}}{\sqrt{\frac{w_g^2}{K^2}+1}} \tag{3.113}$$

Es necesario demostrar el segundo requerimiento para satisfacer al criterio de estabilidad de Nyquist que determina la estabilidad en *loop*-cerrado. De hecho, usando la proposición en 3.110 en 3.114 se concluye.

$$\angle L(jw_g) \geq \angle\frac{K_{\text{red}}\frac{(R^+C)^3}{(2N^-)^2}}{\frac{jw_g}{p_{\text{red}}}+1} - w_gR_0 \geq -90^o - 0{,}1\frac{180^o}{\pi} > -180^o \tag{3.114}$$

Otro punto sugerido por los investigadores enuncia que para cualquier controlador de tipo RED con función de transferencia C_{red} que cumple las condiciones 3.109 y 3.110, el margen de ganancia (MG) y el margen de fase (MF) guardan las relaciones 3.115. Y por lo tanto, el sistema lineal se conservará estable si $R_0 < 15R^+$ o $N > \frac{1}{5\pi N^-}$.

$$MG \geq 5\pi; \quad MF \geq 85^o \tag{3.115}$$

Para demostrar lo anterior se toma la ecuación 3.114 y se calcula su valor aproximado, tal como se muestra en 3.116.

$$\angle L(jw_g) \geq \angle\frac{K\text{red}\frac{(R^+C)^3}{(2N^-)^2} - w_gR_0}{\frac{jw_g}{p_{\text{red}}+1}} \geq -90^o - 0{,}1\frac{180^o}{\pi} \approx -95^o \tag{3.116}$$

De modo que al usar esta aproximación para estimar el margen de fase MF se demuestra la primera relación planteada, tal como se detalla en 3.117.

$$MF = 180^o + \angle L(jw_g) \geq 85^o \tag{3.117}$$

El retardo de fase debido a un *round-trip time delay* ΔR es $\angle \mathrm{e}^{jw_g\Delta R} = -w_g\Delta R$. Por lo tanto, en radianes $w_g\Delta R = 85\left(\frac{\pi}{180}\right)$, y utilizando la desigualdad $w_g \leq \frac{0{,}1}{R_0}$ extraída de 3.109 se obtiene que es válida la relación $\Delta R \leq 14{,}8R$.

Para demostrar lo que fue afirmado respecto del margen de ganancia se comienza considerando la aproximación de la planta que se deriva de la primera proposición que se expone en la ecuación 3.118.

$$P(jw) \approx \frac{L_{\mathrm{red}}\frac{(R_0C)^3}{(2N)^2}}{\frac{jw}{K}+1}, \quad \forall w \in [0, w_g] \tag{3.118}$$

A consecuencia de 3.118 se desprende 3.119.

$$\angle P(jw_g) \geq -90^o \tag{3.119}$$

Y pues es válido que $\angle \mathrm{e}^{-jwR_0} = -90^o$ entonces se concluye $\angle L\left(j\frac{\pi}{2R_0}\right) \geq -180^o$. También se desprende a partir de $|L(jw_g)| \leq 1$ que $1/L\left(j\frac{\pi}{2R_0}\right) \approx \frac{\frac{\pi}{2R_0}}{w_g}$ es válida, esta última constituye una cota inferior para el margen de ganancia. De la relación $w_g \leq 0{,}1R_0$ se infiere que $MG \geq 5\pi$.

En resumen, el trabajo de los investigadores tuvo como principal propósito encontrar pautas, técnicas o relaciones que permitieran diseñar y establecer los parámetros del sistema de forma que se obtuviera un comportamiento estable. Además pudieron comprobar mediante simulaciones en ns que la implementación del sistema funciona acorde a lo planeado. Sin embargo, vislumbraron en su análisis que la estabilidad de la longitud de la cola no debe ser el principal objetivo de diseño de controladores tipo RED pues ocurre que aumentar la estabilidad produce sistemas más lentos.

Su investigación permitió encontrar algunas relaciones entre parámetros de diseño que comprometen el rendimiento del controlador. Por ejemplo, la estabilidad del sistema va en desmedro de la rapidez del mismo, como ya se mencionó. A la inversa, un sistema que tiene una rápida respuesta en el tiempo puede producir un gran *delay jitter*, mayores oscilaciones en los niveles de carga y utilización ineficaz de los recursos. Además las fluctuaciones podrían perjudicar la ejecución de protocolos que se fundamentan en la predicción de las pérdidas y el retardo como lo son aquellos se derivan del protocolo TCP y los que tienen características TCP-*friendly*. En la ejecución de los controlared RED, también se encuentran enlazados entre sí la longitud de la cola y las pérdidas, que tienen como consecuencia a la vez en la dependencia entre la carga y la longitud de las colas lo que tendrá como consecuencia que al producirse niveles de carga altos entonces se reducirá el ancho de banda.

Para finalizar, se puede concluir que el estudio otorga directrices para el diseño AQM basadas en la teoría de control que considera algunas relaciones entre variables y sus efectos en el rendimiento del diseño conocimiento que permite lo que posibilita planear el esquema dependiendo de la aplicación que se requiera.

3.6. Control de Congestion: Definición y Otras Técnicas que Utilizan Teoría de Control de Sistemas

En esta sección se presenta en 3.6.1 la definición de control de congestión, evitación de congestión y control de flujo. Además se describen los objetivos de cada método y cuáles son las características que se esperan en su desempeño. Luego en 3.6.2 se resume una técnica que consiste en un diseño basado en control de sistemas. Se modeló cómo se mueven los paquetes a través de los links y nodos intermedios y también el encolamiento. Se dedujo la función de transferencia del sistema. Se observa que la presencia de grandes delays puede producir inestabilidad y se propone un controlador con predictor de Smith para contrarrestar esto. Para terminar en 3.6.3 se expone un método de control de flujo basado en teoría de control. Se contruye un modelo que representa un conjunto de *router* y *switches* unido por links donde los paquetes fluyen desde una fuente a un destino. Y se busca una ley de control para el modelo presentado. Este trabajo es interesante de revisar porque se utiliza un controlador difuso para determinar un parámetro de importancia de un predictor.

3.6.1. Control de Congestión, Evitación de Congestión y Control de Flujo

En 1998 se presentó el trabajo denominado "Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology"[76] donde se comparan y caracterizan distintas técnicas para contrarrestar la congestión en la red.

La congestión es un fenómeno que ocurre cuando el tráfico en la red sobrepasa el ancho de banda de la entidad que lo recibe, esto se da principalmente en *routers* (nodos intermedios, *gateways* o IMPs) o en *links*. Los problemas de congestión no pueden ser resueltos simplemente mediante aumento del tamaño de la memoria o del incremento del ancho del *link*.

Los conceptos *Congestion Control* (Control de Congestión), *Congestion Avoidance* (Evitación de la Congestión) y *Flow Control* (Control de Flujo) si bien tienen algunas características en común son diferentes. Estos difieren tanto en la manera en que funcionan como en el lugar de la red en que solucionan un problema. Y es importante mostrar sus particularidades antes de presentar ciertos ejemplos de algoritmos de cada tipo.

Flow Control se refiere al mecanismo que al transmitir información entre dos nodos conectados por un *link* impide que el destino sea sobrepasado por los paquetes enviados por la fuente. El principal objetivo de este algoritmo es asegurar el segmento enviado encuentre lugar en el *buffer*, para ello el destinatario comunica el máximo número de paquetes que es capaz de recibir . Algunos ejemplos de esto son *window flow-control*, Xon/Xo [58], *rate flow-control*[51], etc.

Congestion Control a diferencia del anterior no se enfoca en proteger al destinatario sino que a toda la red, pues si se envían paquetes a un ritmo más rápido del que la red puede soportar, entonces se generarán problemas como *queuing* (encolamiento), *buffer overflow*,

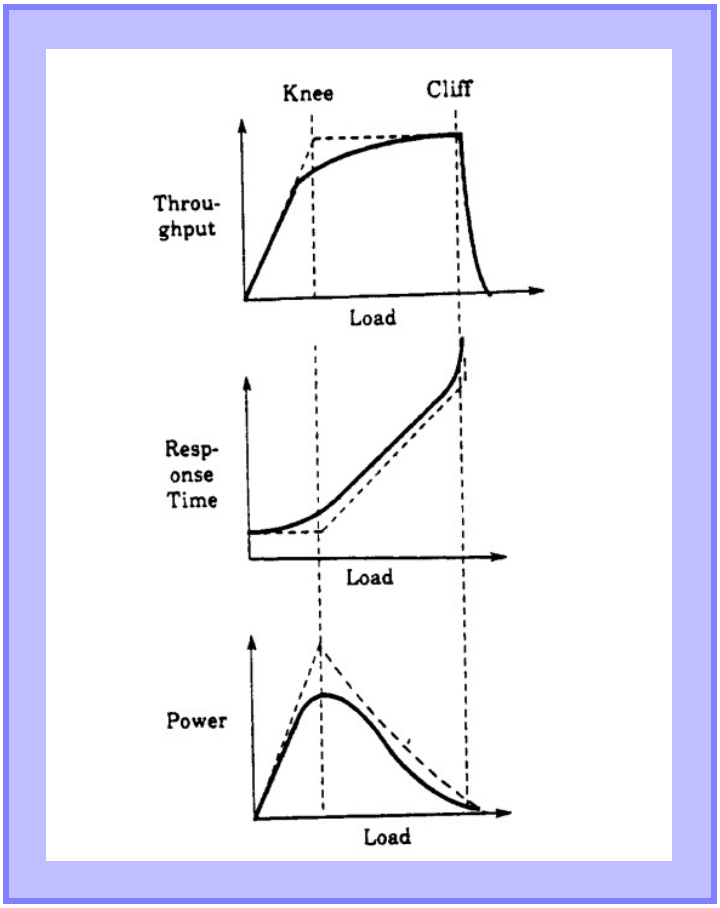


Figura 3.4: Modelo Área Segmento-Tiempo [76]

pérdidas de paquetes, retransmisiones, todos los cuales impactan en el rendimiento de la red. El cuello de botella en congestion control está en los *links* y *routers*, en cambio, en *flow control* este se encuentra en el destinatario. *Congestion Control* actúa cuando la congestión ya ha ocurrido, usando estrategias como *timeout* y *choke packets* [77]. Algunos ejemplos de *Congestion Control* fueron revisados y estudiado por Jain [78] y Ramakrishnan [79].

Congestion Control tiene como función detectar cuando los paquetes enviados han hecho que la red alcance un punto de colapso de congestión también denominado *cliff* que se ejemplifica en la Figura 3.4. En este, la respuesta en el tiempo del sistema cuyo incremento es lineal en principio, alcanza un aumento dramático.En cambio, *Congestion Avoidance* pretende encontrar un punto de operación donde se maximice el *throughput* minimizando la respuesta en el tiempo, el cual se conoce como *knee* o punto rodilla. *Congestion Avoidance* al igual que *Congestion Control* está conformado principalmente por dos mecanismos: retroalimentación y control. El proceso de retroalimentación busca saber, en cada algoritmo, si el sistema se encuentra operando por sobre o bajo los puntos *knee* y *cliff* (Figura 3.4) en *Congestion Avoidance* y *Congestion Control* respectivamente. El mecanismo de control está dedicado a variar el nivel de carga de la red de acuerdo a la información que se recibe de la retroalimentación.

Algunas medidas de rendimiento que son importantes de considerar en cualquier análisis de desempeño de una red y de sus protocolos de control son el *throughput*, el *delay* y la

potencia. El *throughput* se refiere a la velocidad en que se transmite la información en la red, es decir son los *bits* recibidos por unidad de tiempo. El *delay* o retardo corresponde al tiempo que demora el paquete en alcanzar su destino y la potencia, a una combinación de los dos. Al aumentar en magnitud el *throughput* también lo hace el *delay*, sin embargo, uno de los fines esperados en el rendimiento es contrario a esto: se busca máximizar el *throughput* mientras se minimiza el *delay*. Se define la potencia como el cuociente entre el *throughput* y el *delay*, como se expresa en la ecuación 3.120, pues con esta variable se puede evaluar directamente si se cumple el propósito mencionado. Según cuál sea la aplicación, esta puede verse más afectada en su ejecución por determinada variable, por ejemplo, la transferencia de archivos o el correo se ven impactados proncipalmente por el nivel de *throughput* que exista, por otra parte, para *remote login* es la respuesta en el tiempo la que incide mayormente.

$$Power = \frac{Throughput}{Response\_time} \tag{3.120}$$

En el diseño de un esquema de *Congestion Avoidance* además de las variables de desempeño nombradas anteriormente existen otros objetivos que también deben ser tomados en cuenta al momento de evaluar su funcionamiento. Entre estos encontramos eficiencia, *responsiveness*, oscilación mínima, convergencia, *fairness*, robustez, simplicidad, sensibilidad baja a los parámetros, entropía de la información.

Se espera que el algoritmo de *Congestion Avoidance* sea eficiente, la eficiencia se refiere a que el sistema esté en el punto de operación donde se maximiza la potencia, que como se ha mencionado, es conocido como punto rodilla o *knee*. La eficiencia puede verse desde dos enfoques: el global y el local. El sistema es globalmente eficiente cuando el desempeño del sistema entero es el óptimo; en cambio, lo es localmente, si se considera solo a un usuario sin que importe la eficiencia del resto. Otra característica que debe evaluarse es la llamada *responsiveness*, esta es la capacidad del mecanismo de reaccionar apropiadamente a los cambios en la red que hacen variar donde se encuentra el punto de mayor eficiencia. Existe una relación entre la *responsiveness* y las oscilaciones en el sistema, pues para que el sistema opere en el punto *knee* es necesario que se mantenga entorno a este valor. Suprimir las variaciones afectaría negativamente a la *responsiveness*, y por lo tanto, se prefiere que el sistema tenga oscilaciones pequeñas. También es importante la propiedad llamada convergencia y es motivo de desechar el mecanismo no lograr mantener estable el punto de operación cuando la configuración de la red no cambia. La cualidad llamada *Fairness* evalúa si n usuarios de la red comparten equitativamente los recursos obteniendo el mismo *throughput*, y puede cuantificarse por medio de la expresión 3.121, donde x_{i} es el *throughput* del i-ésimo usuario. Es conveniente, además, que el sistema cumpla con ser robusto, es decir, que funcione correctamente en un entorno de ruido, debe ser simple sin sacrificar rendimiento y no debe ser demasiado sensible a los parámetros.

$$Fairness = \frac{(\sum_{\mathrm{i}=1}^{n} x_{\mathrm{i}})\,2}{n\sum_{\mathrm{i}=1}^{n} x_{\mathrm{i}}^2} \tag{3.121}$$

Los algoritmos *Congestion Avoidance* están formados por un mecanismo de control y otro de *feedback* como ya se mencionó. Esta clasificación de técnicas puede hacerse más precisa y específica dividiendo al proceso en políticas de usuario que contienen a los algoritmos:

deteccion de congestión, filtro retroalimentado y selector de retroalimentación; y por otro lado, las políticas de red dentro de las que se encuentran los algoritmos: señal de filtro, función de decisión y algoritmo de incrementro/decremento.

Se debe determinar el estado de carga del sistema para poder entregar información de retroalimentación, la etapa que permite conocer si el sistema se encuentra sobrecargado o infracargado se denomina detección de congestión o *congestion detection*. Luego de esto, se pasa al periodo denominado filtro de retroalimentación donde debe comprobarse si el estado de carga es pasajero o si perdura en tiempo. Algunos de ejemplos de filtros de retroalimentación son: *exponencial weighted average*, promedio móvil de utilización de procesador, utilización de *link* o longitud de cola. Posteriormente, se tomará la decisión de ejercer una acción sobre el sistema o no. Para que la retroalimentación entregue información que permita tomar adecuadamente la acción necesaria se necesita un filtro-pasabajos que deje pasar sólo los estados que tienen una duración significativa esta es la fase llamada selector de retroalimentación.

Después de las políticas de usuario se aplican las de red. Una vez que son enviadas las señales de retroalimentación estas deben ser interpretadas, para ello deben ser acumuladas y analizadas, etapa conocida como filtro de señal. El estado de la red cambia en el tiempo, por tanto, se recibirán señales contradictorias y es el conjunto de ellas que permitirá discriminar la acción adecuada. Cuando ya se conoce el estado de la carga de la red se debe decidir si aumentarla o diminuirla, para ello se utiliza una función que contiene dos partes: la parte que discrimina cuál de las dos acciones tomar, denominada función de decisión y otra que define en qué cantidad, llamada algoritmo de incremento/decremento. El algoritmo de incremento/decremento es la acción que se realiza para controlar el sistema de acuerdo a la información recibida y de la que depende la eficiencia y el *fairness* del sistema.

En resumen, este trabajo expone las diferencias entre los esquemas: *Flow Control*, *Congestion Control* y *Congestion Avoidance*. Además muestra que el último esquema, al actuar de manera preventiva ayuda a llegar a un punto de operación óptimo global. También se describen las etapas que caracterizan a *congestion avoidance* y se definen las cualidades que permiten discriminar si un mecanismo de este tipo es adecuado y tiene buen rendimiento.

3.6.2. Control de Congestión usando el Principio de Smith

Este método fue desarrollado por Saverio Mascolo y fue presentado en 1999 en el trabajo "*Congestion control in high-speed communication networks using the Smith principle*"[75]. Un problema que motiva la investigación de Mascolo es que las redes de alta velocidad pueden tener inestabilidades de ciclo cerrado. La propuesta planteada consiste en una ley de control basada en el principio de Smith que asegura estabilidad y utilización completa. La efectividad del algoritmo pudo ser demostrado analíticamente. La ley de control diseñada se aplicó a tráfico ABR en redes ATM y también se aplica a Internet. Los resultados fueron comparados con algoritmos *Explicit Rate Indication Control* (ERICA).

La mayoría de los algoritmos que existían hasta el momento para tráfico ABR en redes ATM, como los esquemas de *feedback* binario, mostraban problemas de estabilidad, comportamiento oscilatorio y requerían gran cantidad de *buffer* para evitar pérdidas. Además

estos carecían de dos importantes aspectos: el análisis de la dinámica de ciclo-cerrado y la interacción con tráfico VBR the *real-time* and the *non-real-time* variable *bit-rate*.

En este estudio el sistema es representado mediante funciones de transferencia según la teoría de control a partir de lo cual se diseña el controlador. El comportamiento dinámico de cada cola se modela como un integrador más un *delay* en cascada. Se utiliza el principio de Smith en la ley de control, pues del *delay* de propagación se vuelve signicativo en redes de alta velocidad.

Se utiliza un modelo de red llamado *store-and forward packet switching service* que consiste en paquetes que entran a la red donde son almacenados y reenviados a través de nodos y *links* intermedios. Los nodos y *links* que conforman la red se denominan respectivamente $N = \{n_\mathrm{i}\}$ y $L = \{l_\mathrm{i}\}$. Los nodos son aquellos que almacenan y reenvían la información que viaja a través de cierto camino y corresponden principalmente a los *switches* y *routers*. Estos están compuestos por un conjunto de colas de entrada y otro conjunto de colas de salida, se asume que su velocidad de procesamiento de cada nodo es mayor que la velocidad de transmisión de los *links* a los que está conectado. Por otro lado, un *link* es la unión entre dos nodos, cada *link* está definido por su capacidad de transmisión $ci = 1/t_\mathrm{i}$ y por su *delay* de propagación t_di. Además, la red está organizada en pares de nodos *source/destination* (fuente/destino) $(S, D) \in N \times N$ que representan a una conexión, un flujo o un *virtual circuit* (VC). Los paquetes son enviados desde el nodo S al nodo D a través de un camino, o secuencia de *links* que se expresa como $p(S, D)$. El flujo de paquetes se describe mediante la variable continua $u(t)$ en paquetes/segundo y el *bandwith delay product* se puede expresar como t_di/t_j, este valor es importante pues tiene relación con la estabilidad del sistema en ciclo-cerrado.

Se asume que cada salida al *link* del *switch/router* tiene un encolamiento *per-VC first in-first out* (FIFO) que se caracteriza por separar en colas los diferentes flujos lo que favorece la cualidad de fairness de la red. La dinámica de la cola en la red se representa como un integrador simple. Se denomina $x_\mathrm{i}j(t)$ al nivel de cola relacionada con la conexión (S_i, D_j) y el link l_j. Además se designa a la velocidad de flujo de entrada a VC_i como $u_\mathrm{i}(t) \geq 0$ y al ancho de banda en el *link* l_j como $b_{av,\mathrm{i}j}(t) \geq 0$. Considerando además que la cola en el comienzo es cero $x_\mathrm{i}j(0) = 0$, entonces $x_\mathrm{i}j$ queda determinado por la ecuación 3.122, donde $T_\mathrm{i}j$ es el retardo de propagación en el *link* y $\mathrm{d}_\mathrm{i}j$ es la reducción de la velocidad de la j-ésima cola.

$$x_\mathrm{i}j = \int_0^t \left[u_\mathrm{i}(\tau - T_\mathrm{i}j) - \mathrm{d}_\mathrm{i}j(\tau)\right] \mathrm{d}\tau \tag{3.122}$$

La dinámica de las colas en la red se modela a través de funciones de transferencia usando técnicas de control clásico. El diagrama que representa esto es presentado en la figura 3.5. Se asume que una conexión tiene un almacenamiento per-VC en un circuito virtual donde un *link* en particular será el cuello de botella. El objetivo de control es utilizar completamente el *link* en el cuello de botella sin generar sobre-flujo de paquetes. La dinámica del nivel de la cola se clasifica como *single-input-single-output*. Se utiliza un integrador para modelar el *per-flow buffer* donde la longitud de salida de la cola en el cuello de botella es $x_\mathrm{i}j(t)$ en el j-ésimo *link* y la i-ésima conexión . Algunos términos que incluye el modelo son: la perturbación $\mathrm{d}_\mathrm{i}j(t)$ que se modela como una función determinística no conocida en vez de aleatoria, la función de transferencia $\mathrm{e}^{-sT_{fw,\mathrm{i}j}}$ que representa el tiempo de propagación $T_{fw,\mathrm{i}j}$ desde el circuito virtual VC_i hasta el *link* j-ésimo, la función de transferencia $\mathrm{e}^{-sT_{fb,\mathrm{i}j}}$ que representa el tiempo de

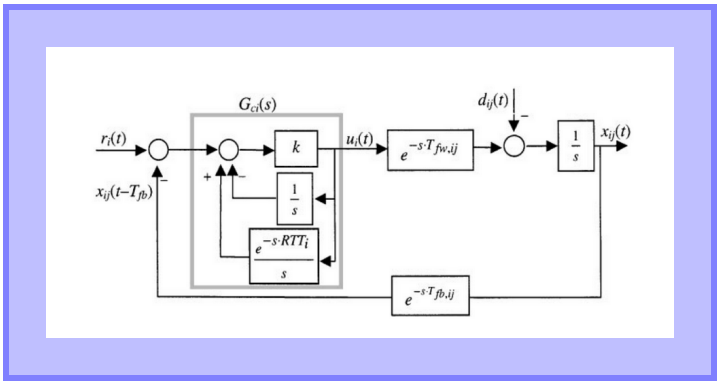


Figura 3.5: Diagrama de bloques de una cola en un cuello de botella controlada

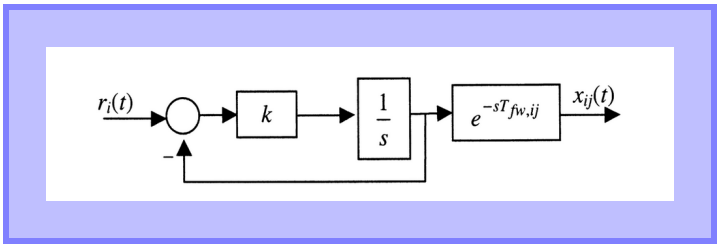


Figura 3.6: Diagrama de bloques de la dinámica de entrada y salida deseada

propagación $T_{fb,ij}$ desde una cola en el cuello de botella en el *link* j-ésimo hasta la fuente i-ésima, la función de transferencia del controlador $G_{ci}(s)$, la velocidad de la fuente de entrada que se denota por $u_i(t)$ y finalmente la señal de referencia $r_i(t)$.

Las entradas del sistema son la referencia $r_i(t)$ y la perturbación $d_{ij}(t)$. El controlador $G_{ci}(s)$ recibe la diferencia entre la referencia $r_i(t)$ y el nivel de la cola retrasado en el cuello de botella $x_{ij}(t - T_{fb,ij})$, con ello calcula la velocidad de entrada. El *round-trip time* en este modelo siempre se puede expresar, para cualquier j-ésimo *link* en el *path* VC_i, como $RTT_i = T_{fw,ij} + T_{fb,ij}$ donde $T_{fw,ij}$ representa el *delay* de propagación y $T_{fb,ij}$ corresponde al *delay* inverso. Por lo tanto, el esquema planteado es un caso general de un cuello de botella en un camino VC.

La referencia r_i constituye un umbral para la longitud de cola en el cuello de botella. Esta señal se denota por $r_i(t) = r_i^0(t - t_{fb})$. La salida está dada por $x_{ij}(t)$ y, por lo tanto, la retroalimentación del sistema corresponde a $(r_i^0(t-T_{fb})-x_{ij}(t-T_{fb}))$ e indica la disponibilidad de espacio en el *buffer*.

La presencia de grandes *delays* de propagación puede producir oscilaciones e inestabilidad es por eso que el controlador tendría un papel importante. Se propone que ese sea diseñado utilizando predictor de Smith, esta técnica es adecuada para el caso estudiado, pues tiene la capacidad de eliminar el *delay* en la ecuación característica de ciclo cerrado.

La ley de control tiene dos objetivos fundamentales: la estabilidad del sistema y la utilización completa del *link* que se pueden definir matemáticamente respectivamente como se indica en las ecuaciones 3.123. En la primera expresión, r^0 es la capacidad de la cola en el cuello de botella y, por lo tanto, la relación señala que la cola se encuentra acotada lo que

evita la pérdida de paquetes. Con ambas condiciones se busca que el nivel de cola sea siempre mayor que cero y menor que la capacidad.

$$\begin{array}{ll} x_{ij} \leq r^0 & para \quad t > 0 \\ x_{ij} > 0 & para \quad t \geq RTT \end{array} \tag{3.123}$$

La planta está formada por un retraso directo, un integrador y un retraso inverso. Para poder hacer el diseño de controlador de Smith se asumen conocidos los retardos y se busca un controlador tal que logre eliminar el retardo en el lazo de control y hacer equivalente el sistema a uno donde el retardo se encuentre fuera de este y en cascada. Es decir, se espera obtener un sistema de entrada y salida como el de la figura 3.6 a partir del sistema en 3.123. Además se aspira a que el sistema sea de primer orden. Así se obtiene que el controlador que cumple con los objetivos de diseño está descrito por la igualdad 3.124.

$$G_c = \frac{k}{1 + (K/s)(1 - \mathrm{e}^{-RTT \cdot s})} \tag{3.124}$$

Del diagrama de bloques se desprende la ecuación de la velocidad corresponde a la ecuación 3.125.

$$u(t) = k \left(r(t - T_{fb}) - x(t - T_{fb}) - \int_{t-RTT}^{t} u(\tau) \mathrm{d}\tau \right) \tag{3.125}$$

De la expresión 3.124 se entiende que la velocidad de entrada es proporcional al espacio disponible en *buffer* menos el número de celdas liberadas en el último *round trip time*. Utilizar la tranformada de Laplace permite hacer análisis matemático de la solución planteada y con ello se demuestra que preserva la estabilidad y logra la utilización completa del *link* cuello de botella.

El mecanismo estudiado por Mascolo fue aplicado al protocolo TCP/IP lo que permitió ver que este ya tiene incorporado en su funcionamiento un predictor de Smith y además logró mejorar el rendimiento de la técnica.

El protocolo TCP/IP, como ya se ha mencionado en otras secciones, es uno de los más efectivos y más utilizados en la actualidad. Este basa su funcionamiento en las medidas de dos variables: la *advertised window* y la *congestion window*. La primera mide el grado de congestión que hay en el *buffer* y la segunda, la congestión en la red. El protocolo a través de la llamada *advertised window* lleva la cuenta del espacio disponible en el *buffer* mediante la relación matemática 3.126, con el fin de evitar sobrecarga.

$$AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - NextbyteRead) \tag{3.126}$$

En 3.126, $MaxRcvBuffer$ es la cantidad máxima de *bytes* que es posible de almacenar en el *buffer*, *LastByteRcvd* es el último *byte* recibido y $NextByteRcvd$ es el próximo *byte* a leer. El protocolo calcula cuantos paquetes puede enviar la fuente a través de las ecuaciones 3.127 y 3.128.

$$MaxWindow = MIN(CongestionWindow, AdvertisedWindow) \tag{3.127}$$

$$\begin{aligned} W &= MaxWindow - (LastByteSent - LastByteAcked) \\ &= MaxWindow - OutstandingPackets \end{aligned} \tag{3.128}$$

El protocolo TCP/IP utiliza una técnica de incremento/decremento para sondear la capacidad del *link*, cada vez que ocurre una pérdida de un paquete el tamaño de la ventana es reducido, y luego, vuelve a incrementarse hasta que sucede una nueva pérdida. Cuando la ventana de congestión es demasiado pequeña el *link* no será aprovechado apropiadamente, por el contrario, cuando la ventana de congestión es muy grande se producen pérdidas y puede llegar a incrementarse el tamaño de los retardos o *delays* lo que se reflejará negativamente en el funcionamiento de las aplicaciones. Para aplicar el controlador de Smith sobre TCP primero debe reinterpretarse la ecuación 3.125 pues el protocolo TCP es una estratagia de control de tipo *window-based*,de ello se obtiene la igualdad 3.129.

$$u(t)\tau = r^0(t - T_{fb}) - x(t - T_{fb}) - \int_{t-RTT}^{t} u(\tau)\mathrm{d}\tau \tag{3.129}$$

En la ecuación 3.129 el término $u(t)\tau$ representa a la ventana de paquetes W que es posible enviar en el tiempo t, además la integral en la expresión corresponde a los paquetes enviados por la fuente que no han sido confirmados, es decir, los paquetes pendientes o *outstanding packets*. A los términos $r^0(t - T_{fb}) - x(t - T_{fb}))$ se les designa como *Generalized Advertised Window* y corresponde al mínimo espacio disponible considerando todas las conexiones. Si B_{i} es la disponibilidad en el i-ésimo *buffer* entonces lo anterior se puede expresar a través de la ecuación 3.130.

$$\begin{aligned} GAW &= (r^0(t - T_{fb}) - x(t - T_{fb})) \\ &= min\{B_{\mathrm{i}}, AdvertisedWindow\} \end{aligned} \tag{3.130}$$

Usando las definiciones la ecuación 3.129 se puede reescribir como la igualdad 3.131.

$$W = GAW - OutstandingPackets \tag{3.131}$$

De la ecuación 3.128 se observa que el mecanismo propuesto es compatible con TCP incluso si se utiliza una estimación de GAW como lo sería $estimated(GAW) = B/N$ donde B es la capacidad y N es número de colas.

$$MaxWindow = MIN(CongestionWindow, GAW) \tag{3.132}$$

Se puede notar que las ecuaciones 3.128 y 3.131 son iguales de ello se infiere que TCP ya tiene una estructura de predictor de Smith en su algoritmo con la diferencia que la estimación del espacio libre en el *buffer* en el método propuesto se hace mediante la ecuación 3.127. Se consideró la similitud de los mecanismos como una confirmación de la efectividad del método propuesto, pues TCP es un protocolo que ha dado pruebas de su correcto y exitoso funcionamiento en la red.

3.6.3. Control de Flujo Mediante Teoría de Control

El 1991 fue presentando el trabajo denominado *"A control-theoretic approach to flow control"* [80] de Srinivasan Keshav que entrega un método para hacer control de flujo en redes

que no reservan ancho de banda a través de técnicas basadas en la teoría de control. Esto busca adaptar las transmisiones de información a los cambios de estado de la red con el fin de lograr los objetivos de control.

Se había probado en trabajos anteriores que hacer reservaciones para cada canal aseguraba buen desempeño en cuanto a *throughput*, *delay* y *delay jitter*. No obstante, esta estrategia perjudicaba la multiplexación estadística, lo que constituyó una de las motivaciones para el estudio de mecanismos de control en canales sin reserva de ancho de banda. La técnica desarrollada permite que el flujo se adapte a los cambios en la red mediante la teoría de control. Esto requiere la observación permanente de los estados de la red, lo que puede ser medido sin dificultad en redes cuyos servidores son de tipo *Rate Allocating Server*(RAS) y que implementan la técnica de sondeo *Packet-Pair* [81] [82].

Rate Allocating Server (RAS) se refiere a los esquemas de servicio de cola. Son agrupados bajo esta denominación debido a su parecido los algoritmos *Virtual Clock* y *Fair Queuing*. RAS en redes sin reserva de ancho de banda dividirá los recursos tal como lo hace el algoritmo *Fair Queuing*, entre todos los canales que se encuentren funcionando, porque el servidor no puede rechazar canales a diferencia de los que ocurre en redes con reservación orientada. Bajo esta disciplina la velocidad del servicio en cada canal puede modificarse si cambia el número de canales activos o si el canal tiene una velocidad de llegada más baja que el ancho de banda reservado. Aún así, RAS tiene ventaja respecto de los servidores que aplican la técnica *First-Come-First-Served* (FCFS) pues las transmisiones son independientes y no se encuentran ligadas, por lo tanto, aunque ocurra una ráfaga en una no perjundicará el funcionamiento del resto.

En este trabajo se plantea un modelo analítico para la red RAS. Se escoge un esquema determinístico con el fin de estudiar el periodo transiente, pues este tipo permite un cálculo más exacto y menos complicado que si fuese modelado como un sistema de colas. También se consideró una extensión estocástica del planteamiento para examinar las variaciones que se producen en la velocidad de servicio..

El modelo determinístico propuesto consiste en un conjunto *routers* o *switches* unidos por *links*, donde los paquetes fluyen desde una fuente hasta un destino a través de un grupo nodos intermedios. El tiempo de servicio de cada *router* es finito y determinístico y se define como s_{i}, en consecuencia, la velocidad de servicio en cada servidor es $\rho_{\mathrm{i}} = 1/s_{\mathrm{i}}$. La velocidad de envío de paquetes desde una fuente se designa como λ y además se asume que la fuente envía datos en intervalos de tiempo $s_0 = 1/\lambda$. El tiempo de servicio en cuello de botella de una transmisión se determina por medio de la expresión 3.133, donde b es el índice del servidor cuello de botella.

$$s_b = max(s_{\mathrm{i}} | 0 \leq \mathrm{i} \leq n) \tag{3.133}$$

La velocidad de servicio en el cuello de botella de denota por μ y corresponde a $\frac{1}{s_b}$. Las definiciones también se realizaron en tiempo discreto reemplazando la variable tiempo t por k. Por ejemplo, el total de paquetes pendientes en el tiempo k se denota como $S(k)$.

$$\hat{n}_b(k+1) = \hat{n}_b(k) + S(k) - RTT(k)\hat{u}(k) \tag{3.134}$$

$$\hat{n}_b(k+2) = \hat{n}_b(k+1) + S(k+1) - RTT(k+1)\hat{u}(k+1) \tag{3.135}$$

A diferencia del modelo determinístico, donde μ se asume constante, el modelo estocástico permite que este parámetro sea variable. Se dice que el valor de μ variará en torno a un valor nominal μ_0, o lo que es lo mismo, $\mu(k+1)$ debería estar próximo a $\mu(k)$, pues si el número de canales activos N_{ac} es grande, el cambio magnitud de N_{ac} debería representar un valor marginal y, así también, la fluctuación de μ. Por lo tanto, μ es modelado como una variable aleatoria con media cero y pequeña varianza 3.136.

$$\mu(k+1) = \mu(k) + \omega(k) \tag{3.136}$$

En la ecuación anterior $\omega(k)$ representa un ruido blanco gaussiano de media cero. Esto es modelado así por los siguientes motivos: la velocidad de servicio es no correlacionada; la distribución gaussiana aproxima al menos el primer orden de cualquier tipo de distribución; la mayoría de las fuentes de ruido de la naturaleza son de este tipo gaussiano y finalmente, se requiere ruido blanco gaussiano para hacer estimaciones de Kalman.

La estrategia de diseño de la técnica de control de flujo se basa en el Teorema de Separación [83]. Esta proposición plantea que en un sistema lineal estocástico, donde se estima el estado del sistema por un observador, los eigenvalores del estado estimador y del controlador se pueden separar. Esto permite obtener una ley de control usando los estados estimados.

El objetivo del control es mantener el número de paquetes en la cola del cuello de botella n_b en cierto valor establecido como *setpoint*. La medida de este último estará enlazada con las variables *packet delay*, *packet loss* y *bandwith loss*, es decir, según si el *setpoint* es elegido de mayor o menor magnitud afectará de diferente forma el valor alcanzado por las variables mencionadas. En efecto, si el *setpoint* es pequeño la probabilidad de *bandwith loss* (pérdida de paquetes) se incrementa mientras que el *mean packet delay* y la probalidad de *packet loss disminuyen*. Por otro lado, si el valor del *setpoint* es grande se incrementa la probabilidad de packet loss y el *mean delay* y se reduce la probabilidad de pérdida de ancho de banda. En el trabajo se fija el *setpoint* como $B/2$ donde B es el total de búferes en cada canal, esta elección pretende equilibrar los *tradeoffs* observados.

En el trabajo de Keshav se asumió que el *propagation delay* R (retardo de propagación) es constante en un canal y que el tiempo de *round trip time* es largo respecto del tiempo entre paquetes de confirmación. Además, se convino que el *round trip time* en la época $k+1$ puede ser bien aproximado por $RTT(k)$, esto se fundamenta en que el sistema en equilibrio tendrá más o menos la misma longitud de cola en épocas consecutivas y por ello el cambio en el *queueing delay* (retardo de encolamiento) se considera despreciable respecto del *propagation delay* (retardo de propagación).

Para encontrar la adecuada ley de control se asumieron varias condiciones para el modelo.

El tiempo fue dividido en épocas de longitud RTT y se permitió una acción de control en cada época. Por otro lado, el escenario que bosqueja el sistema considera varias variables para determinar las ecuaciones matemáticas que caracterizan al esquema planteado y permiten definir la ecuacion del controlador, entre estas se puede enumenar: $RTT(k)$, el *round trip time* en la k-ésima época; $S(k)$, la cantidad de paquetes pendientes; el estimador del promedio de la velocidad de servicio $\hat{\mu}(k+1)$, $n_b(k)$ la cantidad de paquetes en el comienzo de la época k-ésima.

Se proponen las ecuaciones que describen el comportamiento del sistema. La cantidad de paquetes pendientes $S(k)$ en la época k-ésima, es decir, aquellos enviados en ese mismo periodo, se puede aproximar como el producto entre la velocidad de envío por el intervalo de envío en la época k-ésima como muestra la ecuación 3.137. Esto porque ya se ha confirmado su recepción de todos los paquetes enviados en la época $k-1$.

$$S(k) = \lambda(k)RTT(k) \tag{3.137}$$

La cantidad de paquetes en el cuello de botella en el inicio de la época $(k+1)$-ésima, $n_b(k+1)$, se puede calcular conociendo la cantidad de paquetes en el comienzo de la época k-ésima y el número de paquetes que salieron en esta misma época, haciendo la sustracción de ambos como se expresa en la ecuación 3.138.

$$n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k) \tag{3.138}$$

Combinando 3.138 y 3.137 lo anterior también puede ser declarado según la igualdad 3.139.

$$n_b(k+1) = n_b(k) + S(k) - \mu(k)RTT(k) \tag{3.139}$$

Se intenta hacer el control sobre $n_b(k+2)$ pues $n_b(k+1)$ está determinado por lo acontecido en la k-ésima época.

$$n_b(k+2) = n_b(k+1) + (\lambda(k+1) - \mu(k+1))RTT(k+1) \tag{3.140}$$

A partir de la ecuaciones 3.139 y 3.140 se obtiene la expresión 3.141.

$$n_b(k+2) = n_b(k) + S(k) - \mu(k)RTT(k) + \lambda(k+1)RTT(k+1) - \mu(k+1)RTT(k+1) \tag{3.141}$$

El *setpoint* como ya se ha mencionado se estableció en $B/2$, esto se aplica para obtener el control sobre la ecuación 3.141 como se muestra en 3.142.

$$n_b(k+2) = B/2 = n_b(k) + S(k) - \mu(k)RTT(k) + \lambda(k+1) - \mu(k+1)RTT(k+1) \tag{3.142}$$

De la expresión 3.142 se desprende la igualdad 3.143.

$$\lambda(k+1) = \frac{1}{RTT(k+1)}\left[B/2 - n_b(k) - S(k) + \mu(k)RTT(k) + \mu(k+1)RTT(k+1)\right] \tag{3.143}$$

Al reemplazar los estimadores en la ecuación anterior se obtiene la expresión 3.144.

$$\lambda(k+1) = \frac{1}{\hat{RTT}(k+1)}\left[B/2 - \hat{n}_b(k) - S(k) + \hat{\mu}(k)RTT(k) + \hat{\mu}(k+1)\hat{RTT}(k+1)\right] \tag{3.144}$$

Finalmente, asumiendo: $\hat{\mu}(k) = \mu(\hat{k+1})$, pues ambos valores son desconocidos; y $RTT(k) = RTT(k+1)$, suposición que ha sido justificada anteriormente, resulta la ley de control que se expone en la ecuación 3.145.

$$\lambda(k+1) = \frac{1}{RTT(k)}[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)] \tag{3.145}$$

En este trabajo se realiza un análisis de estabilidad sobre la base del modelo teórico contruido que permite conocer bajo que circunstacias el sistema se mantiene estable. El análisis de estabilidad se consigue a partir de la ecuación de estado 3.138, reemplazando el término $\lambda(k)$ por la derivación de la ley de control 3.145. No obstante, esta última corresponde a $\lambda(k+1)$ en vez de $\lambda(k)$ entonces debe utilizarse la ecuación de estado $n_b(k+2)$ 3.140 con el fin de poder hacer la sustitución, así la ecuación 3.140 combinada con 3.144 da como resultado 3.146.

$$n_b(k+2) = n_b(k+1) - \mu(k+1)RTT(k+1) + \frac{RTT(k+1)}{RTT(k)}[B/2 - \hat{n}_b - S(k) + 2\hat{\mu}(k)RTT(k)] \tag{3.146}$$

En la expresión anterior, a consecuencia de que se ha supuesto que los valores de *round trip time* en dos épocas consecutivas deberían ser idénticos y se puede reemplazar $\frac{RTT(k+1)}{RTT(k)} = 1$. Si además se retrocede la ecuación dos pasos hacia atras, entonces se obtiene la igualdad 3.147.

$$n_b(k) = n_b(k-1) - \mu(k-1)RTT(k-1) + B/2 - \hat{n}_b(k-2) - S(k-2) + 2\hat{\mu}(k-2)RTT(k-2) \tag{3.147}$$

A esta última, 3.147, se le puede aplicar la transformada Z si se supone que $n_b(k-2) = \hat{n}_b(k-2)$ obteniéndose la igualdad 3.148.

$$n_b(z) = z^{-}1n_b(z) - z^{-2}\mu z RTT(z) + B/2 - z^{-2}n_b(z) - z^{-2}S(z) + 2z^{-4}\hat{\mu}(z)RTT(z) \tag{3.148}$$

Tomando a n_b como un estado variable se llega a la ecuación característica 3.149.

$$z^{-2} - z^{-1} + 1 = 0 \tag{3.149}$$

Para que el sistema sea asintóticamente estable los eigenvalores del sistema (en la ecuación 3.150) deben estar dentro del círculo unitario en el plano complejo. No obstante, en este caso los valores propios se encuentran sobre el círculo unitario pues el módulo, que corresponde a la distancia a los polos desde el origen, da uno como resultado. Por lo tanto, el sistema no es asintóticamente estable.

$$z^{-1} = \frac{1 \pm \sqrt{1-4}}{2} = \frac{1 \pm \sqrt{3}\mathrm{i}}{2} \tag{3.150}$$

Pero es posible situar el polo de manera que cumpla con el criterio de estabilidad modificando la ley de control a la expresión 3.151.

$$\lambda(k+1) = \frac{\alpha}{RTT}[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)] \tag{3.151}$$

Para la nueva ley de control la ecuación característica varía a 3.152.

$$az^{-2} - z^{-1} + 1 = 0 \tag{3.152}$$

Por tanto, los eigenvalores en este caso son 3.153 y se debe cumplir la condición de estabilidad desarrollada en la expresión 3.154.

$$z^{-1} = \frac{1 \pm \sqrt{4\alpha - 1}\mathrm{i}}{2\alpha} \tag{3.153}$$

$$\begin{array}{rcl} |z^{-1}| & > & 0 \\ \sqrt{\left(\frac{1}{2\alpha}\right)^2 + \left(\frac{\sqrt{4\alpha-1}}{2\alpha}\right)^2} & > & 1 \\ \frac{1}{\sqrt{\alpha}} & > & 1 \\ \alpha & < & 1 \end{array} \tag{3.154}$$

Se deriva de 3.154 que si se cumple el requisito $\alpha < 1$, entonces el sistema es asintóticamente estable. En otras palabras, la fuente debe enviar los paquetes a velocidad levemente inferior a la propuesta en la ecuación 3.145 para que el sistema se mantenga estable. Pero además para obtener buenas características de responsividad α no debe ser escogido demasiado bajo. Las pruebas experimentales del trabajo de Keshav mostraron que $\alpha = 0{,}9$ es un valor apropiado.

El sistema es no lineal en los bordes, pues por las características físicas del sistema es imposible que $n_b(k+1)$ esté fuera del rango $[0, B]$, esto invalida la demostración de estabilidad anterior para grandes oscilaciones en torno a un valor de *setpoint* dentro del intervalo, esto se debe a que la técnica corrobora la estabilidad solo en casos de sistemas lineales . Sin embargo, la estabilidad en un escenario no lineal puede ser demostrada por medio del segundo método de Liapunov. En la práctica, la no linealidad del sistema no tiene consecuencias ya que la ecuación de control asegura $\lambda(k+1)$ tal que el sistema se mueva hacia $n_b(k+2) = B/2$.

Para obtener los estimadores requridos por la ley de control se utiliza la estimación de estado de Kalman debido a su robustez, este estimador se caracteriza además por minimizar la covarianza. Para aplicarlo se necesita definir el sistema en el espacio estado para ello se puede utilizar la ecuación 3.155.

$$\begin{array}{c} x(k+1) = Gx(k) + Hu(k) + v_1(k) \\ y(k) = Cx(k) + v_2(k) \end{array} \tag{3.155}$$

En la expresión 3.155 los estados están dados por los vectores de largo n, m y r denominados x, u e y respectivamente. H y G son matrices,$v_1(k)$ y $v_2(k)$ representan el ruido blanco y gaussiano del sistema y de observación correspondientemente.

Para el sistema estudiado $u(k)$ es un escalar que corresponde a la ley de control $u(k) = \lambda(k)$. El vector de estado está formado por tres variables n_b, $\mu(k)$ y $\mu(k-1)$, la última proviene de la ecuación de observación de estado $y(k) = \mu(k-1) + v_2$. Las matrices G, H y C se pueden deducir observando las ecuaciones de estado 3.156.

$$\begin{array}{c} n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k) \\ \mu(k+1) = \mu(k) + \omega(k) \\ \mu_{-1}(k+1) = \mu(k) \end{array} \tag{3.156}$$

Conforme a la ecuación 3.155 y a las ecuaciones de estado 3.156 se obtienen las matrices

3.157.

$$G = \begin{bmatrix} 1 & -RTT & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix},\ x = \begin{bmatrix} n_b \\ \mu(k-1) \\ \mu(k) \end{bmatrix},\ H = \begin{bmatrix} RTT \\ 0 \\ 0 \end{bmatrix},\ v_1 = \begin{bmatrix} 0 \\ \omega \\ 0 \end{bmatrix},\ C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{3.157}$$

Luego se debe construir el estimador de Kalman donde se utilizarán además las matrices de estado ya derivadas. Para ello se deben encontrar las matrices Q, S y R que cumplen las relación 3.158, donde δ representa a una delta de Kroneckener.

$$E\left[\begin{bmatrix} v_1(k) \\ v_2(k) \end{bmatrix} \begin{bmatrix} v_1^T(\theta) & v_2(\theta) \end{bmatrix}\right] = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \delta(t-\theta) \tag{3.158}$$

Haciendo la multiplicación de matrices en el lado izquierdo usando las matrices de estado se obtiene que las matrices requeridas son las expuestas en 3.159.

$$Q = E\begin{bmatrix} 0 & 0 & 0 \\ 0 & \omega^2 & 0 \\ 0 & 0 & 0 \end{bmatrix},\ R = E\left(V^2\right),\ S = E\begin{bmatrix} 0 \\ \omega v_2 \\ 0 \end{bmatrix} \tag{3.159}$$

El estimador $\hat{x}$ se obtiene a través de la ecuación 3.160.

$$\begin{gathered} \hat{x}(k+1) = G\hat{x}(k) + K(k)\left[y(k) - C\hat{x}(k)\right] + Hu(k) \\ \hat{x}(0) = 0 \end{gathered} \tag{3.160}$$

En la expresion 3.160, K representa a la matriz de Kalman que se deriva de la ecuación de Ricatti 3.161.

$$\begin{gathered} K(k) = \left[G\Sigma(k)G^T + Q - K(k)\right]\left[C\Sigma(k)C^T + R\right]K(k)^T \\ \Sigma(0) = \Sigma_0 \end{gathered} \tag{3.161}$$

En la relación 3.160, Σ_0 corresponde a la covarianza de x en el tiempo 0 que se puede asumir como cero. De la ecuaciones se puede deducir que estimador de Kalman no puede ser utilizado si no se tiene información de las varianzas del sistema y por eso en este caso resulta inconveniente y se recurrió a un método de estimación difusa que puede prescindir de información sobre el sistema y el ruido de observación.

El mecanismo difuso de estimación sugerido por Keshav se fundamenta en la predicción del k-ésimo elemento de la serie de una variable en el tiempo según la ecuación 3.162.

$$\theta_k = \theta_{k-1} + \omega_{k-1} \tag{3.162}$$

En 3.162, ω_k es una variable aleatoria de distribución desconocida. No es posible conocer con exactitud θ_k ya que está alterado por el ruido de observación, en consecuencia, se busca un predictor óptimo que lo represente.

El predictor utilizado tiene su raíz en el método conocido como promedio exponencial que se caracteriza por ser robusto y por tener numerosas aplicaciones. Este queda definido por la fórmula 3.163 donde α representa un peso que le da mayor o menor ponderación al pasado.

$$\hat{\theta}_{k+1} = \alpha\hat{\theta}_K + (1-\alpha)\hat{\theta}_k \quad o \quad \hat{\theta}_k = \textstyle\sum_{\mathrm{i}=0}^{k-1}(1-\alpha)\hat{\theta}_\mathrm{i}\alpha^{k-\mathrm{i}-1} + \alpha^k\hat{\theta}_0 \tag{3.163}$$

Se ha asumido que el sistema tiene un comportamiento desde estable hasta ruidoso y así la dificultad de la técnica planteada es encontrar el valor de α adecuado. Si el sistema es estable entonces α deberá ser escogido grande para darle mayor ponderación pasado. Por el contrario si el sistema es ruidoso, entonces será apropiado que α sea pequeño para darle mayor importancia a la información reciente. La estrategia de control difuso juega su rol en determinar el valor de α más adecuado.

Para su funcionamiento el estimador difuso necesita una variable que cuantifique el grado de ruido en el sistema. Esto se logra a través del error de estimación, ya que si el sistema es estable la predicción será más precisa con menor error. A la inversa, si el sistema es ruidoso entonces el error de predicción será mayor. Las leyes de control que definen al controlador son:

Si el error es bajo, entonces α es alto ,
Si el error es medio, entonces α es medio ,
Si el error es alto entonces α es bajo .

Como entrada al controlador se usa un valor proporcional al error descrito por $\frac{\tilde{\theta}_k - \hat{\theta}_k}{\hat{\theta}_k}$. Como salida de controlador debe obtenerse el valor más adecuado de α, para ello debe ocurrir un proceso conocido como defuzzificación que en este caso particular fue realizado por medio de la técnica del centroide.

Otra forma presentada para calcular $\hat{n_b}(k+1)$ utiliza la medición del *delay* de propagación. El *delay* de propagación existe debido al tiempo de procesamiento en *switches* e interfaces. Este retardo puede ser aproximado al *round trip time* si no está presente *queueing delay* o retardo de encolamiento. El *queueing delay* puede ser evitado midiendo el *round-trip time* del primer paquete del primer *packet-pair*. Usando R es posible calcular los paquetes en el cuello de botella haciendo la diferencia de los paquetes transmidos y aquellos de los que no se ha recibido confirmación, matemáticamente esto queda según se expresa en 3.164.

$$\hat{n}_b(k+1) = S(k) - R\hat{\mu}(k) \tag{3.164}$$

Esta expresión permite tener menos riesgo de error sistemático y además ejecutar la acción de control con mayor frecuencia que una vez por *round-trip time*.

Finalmente, algunos resultados obtenidos por el algoritmo son: rapidez de respuesta a los cambios de estado en la red, rendimiento comparable a DECbit, buen comportamiento donde el *bandwith-delay product* es grande, implementación sencilla y dinámica simple.

3.7. Técnicas de Control de Congestion de Tipo end-to-end

En esta sección se describen técnicas de tipo end-to-end, estos son de implementación más sencilla que los router-supported y miden el estado de congestión de la red a través de las pérdidas o el delay. En 3.7.1 se revisa HighSpeed que es interesante porque tiene en común con

el protocolo propuesto en este trabajo que varía los valores de los parametros de incremento y decremento carácterísticos de TCP. Luego se describe un trabajo llamado Extensión de TCP para Trayectos de delay largo,3.7.2. Este fue uno de los primeros métodos planteados para enfrentarse a entornos de alto *bandwidth delay product* que dio origen a la versión de TCP conocida como SACK donde se utilizan bloques para determinar y retransmitir los paquetes perdidos. En 3.7.3 se menciona Scalable TCP y tiene en común con HighSpeed y el protocolo propuesto en esta tesis la variación de los parámetros de *Congestion Avoidance*. En 3.7.4 se describe el método llamado *Adaptative Delay-based Congestion Control* que tiene como particularidad que evalúa la congestión en la red utilizando el *delay* que lo distingue de los métodos anteriores considerados en esta sección además implementa la medición de una variable denominada *fairness-ratio* para evitar acaparar los recursos. Finalmente, en 3.7.5 se estima el nivel de congestion midiendo la cantidad de paquetes enviados correctamente y se determina el factor de descremento haciendo un mapeo.

3.7.1. HighSpeed TCP

HighSpeed TCP es un método de control de congestión presentado por Sally Floyd en el año 2003 en el documento titulado "High Speed for Large Congestion Window"[27]. Esta técnica fue propuesta para enfrentar el problema que ocurre en condiciones de ventanas de congestión de gran de tamaño. En este caso se espera que se cumpla la relación de promedio de la ventana de congestión que rige para Standard TCP $1{,}2/\sqrt{p}$, donde p representa a la tasa de pérdidas. La anterior relación impone grandes limitaciones para alcanzar valores altos del *throughput*. El algoritmo sugerido se comporta tal como TCP Standard para escenarios de congestión alta y mediana, pero modifica la respuesta TCP cuando las ventanas de congestión son más grandes.

El mecanismo planteado consiste en modificar los parámetros de incremento y decremento característicos de TCP con el fin de obtener requerimientos más realistas comparados con los que exige Standard TCP respecto de la tasa de pérdidas y la cantidad de *round-trip time* entre pérdidas. Se espera con lo anterior alcanzar *throughput* altos en las conexiones. Otros objetivos considerados por el algoritmo son evitar *delays* altos al recuperarse de múltiples retransmisiones, alcanzar un *throughput* alto con rapidez durante Slow Start, lograr rendimiento comparable a TCP Standard en entornos con congestión media o alta, prescindir de *feedback* adicional desde *receivers* o *routers* , y obtener desempeño aceptable en transiente.

Las limitaciones en TCP Standard se reflejan en la función de respuesta de TCP $w = 1{,}2/\sqrt{p}$ que permite relacionar el tamaño promedio de la ventana en segmentos MSS con la probabilidad de pérdida p en estado estacionario. Esta función es consecuencia del mecanismo de incremento aditivo y decremento multiplicativo característicos de TCP. El protocolo HighSpeed modifica la función de respuesta y para que esta quede determinada se definen tres parámetros *Low_ Window*, *High_ Window* y *High_ P*. La variación de la función se ejecuta para valores de ventana mayores al parámetro *Low_ window*, en cambio, para tamaños menores se conserva la función de respuesta original de modo que se mantenga el comportamiento de Standard TCP en casos de congestión media y alta. *Low_ Window* es fijado en 38 MSS segmentos que corresponden a una tasa de pérdidas de paquetes de 10^{-3} de TCP. El parámetro *High_ Window* es fijado en 83000 segmentos y *High_ P* como 10^{-7}. La nueva

función de respuesta queda como se muestra en la ecuación 3.165.

$$W = \left(\frac{p}{Low\_P}\right)^{S} Low\_Window \tag{3.165}$$

En 3.165 Low_P es la tasa de pérdida respectiva a Low_Window y S corresponde a la expresión 3.166.

$$S = \frac{log(High\_Window) - log(Low\_Window)}{log(High\_P) - log(Low\_P)} \tag{3.166}$$

Se busca que la nueva función de respuesta sea compatible con TCP Standard para valores de tasa de pérdidas en el intervalo $[10^{-3}, 10^{-1}]$ y a valores de pequeña pérdida permita tasas de caídas de paquetes acordes a un conexión de *throughput* de 10 Gbps.

Se puede evaluar el *fairness* del protocolo HighSpeed respecto de TCP Standard a través de sus funciones de respuesta, por ejemplo, para una tasa de pérdida p se hace el cuociente entre la ventana de congestión $WStandard$ de TCP Standard y la ventana $WHighspeed$ de TCP HighSpeed. Según los datos manejados en la investigación para valores de tasa de ventana menor a 10^{-4} el sistema se comporta de manera "injusta", sin embargo, los beneficios que tendría el algoritmo en ese contexto son mayores que esta desventaja. Sin embargo, este tiene buen comportamiento en cuanto a *fairness* en la etapa de *congestion avoidance*, teniendo mejor convergencia que TCP Standard. Además, a lo anterior se puede agregar que el trabajo de Chiu y Jain [12] plantea que el algoritmo AIMD converge a *fairness* en un ambiente de congestión de eventos sincronizados.

Ya se ha dicho que la función de respuesta de TCP Standard está ligada al mecanismo AIMD que le caracteriza, por lo tanto, la nueva función de respuesta diseñada para el algoritmo HighSpeed no puede lograrse mediante AIMD y entonces este debe ser modificado. Así que la varible de incremento multiplicativo $a = 1/2$ de TCP Standard es modificada a la función $a(w)$ que modela análogamente el incremento por *round-trip time* tal como lo indica la expresión 3.167.

$$w \leftarrow w + a(w)/w \tag{3.167}$$

Por otro lado, el parámetro de decremento $b = 1$ en TCP Standard es reemplazado por $b(w)$ y, por lo tanto, el tamaño de la ventana ante un evento de congestión cambia según la ecuación 3.168.

$$w \leftarrow (1 - b(w))w \tag{3.168}$$

La ecuación que relaciona ambas funciones para $w = High\_Window$ es

$$a(w) = High\_Window^2 * High\_p * 2 * \frac{b(w)}{2 - b(w)}\,, \tag{3.169}$$

para utilizar esta función se establece el decremento y se obtiene de la ecuación el valor del incremento.

El protocolo diseñado define los valores de $b(w)$ para el tamaño de ventana $High\_Window$ como se mencionó antes y también para Low_Window. Estos últimos valores coinciden con los de TCP Standard, para valores intermedios de ventana se detemina la variación lineal como $log(w)$ tal como indica la ecuación 3.170.

$$b(w) = (High\_Decrease - 0{,}5)\frac{log(w) - log(W)}{log(W_1) - log(W)} + 0{,}5 \tag{3.170}$$

Y el parámetro de incremento queda definido por la expresión 3.171 donde $p(w)$ representa la tasa de pérdidas.

$$a(w) = w^2 p(w) * 2\frac{b(w)}{2 - b(w)} \tag{3.171}$$

Una variante de esta técnica considera una función de respuesta lineal en HighSpeed y estableciendo los parámetros $Low\_Window = 38$, $High\_Window = 380000$, $High\_P = 10{-}7$ y $Low\_P = 10^{-3}$ se llega a la función lineal $W = 0{,}038/p$ válida para el intervalo donde $w > Low\_P$ y para esta función respuesta los parámetros de incremento y decremento serían tales que si se elige $b(w) = 1/2$ entonces $a(w) = w/Low\_Window$ o $a(w) = 1/Low\_Window$.

El *fairness* de la técnica alternativa denominada Linear HighSpeed resulta ser menor que el de Standard TCP pues es una técnica más agresiva, sin embargo este tiene las atractivas característica de ser escala-invariante, es decir tiene un incremento fijo en la ventana de congestión por *acknowledgement* y una cantidad fija de *round-trip time* entre pérdidas.

Para las funciones respuesta de la forma $w = c/p^{\mathrm{d}}$, donde c es una constante, se asume que las únicas admisibles como tales serían aquellas donde el exponente d cumpla que $1/2 \leq \mathrm{d} \leq 1$. De ese grupo de funciones aquella con $\mathrm{d} = 1/2$ corresponde a TCP Standard y la de $\mathrm{d} = 1$ coincide con Linear HighSpeed, en medio de estas se encuentra HighSpeed con $\mathrm{d} = 0{,}835$. Las funciones respuesta con exponente menor que $1/2$ son rechazadas porque podrían tener peor desempeño que TCP Standard en la acomodación de conexiones. También aquellas que tienen exponente mayor que uno se descartan porque el número de *round-trip times* entre pérdidas decrece cuando la ventana w se incrementa, a diferencia de TCP Standard donde el número de *round-trip times* entre eventos de pérdida es lineal con w y en contraste con Linear HighSpeed donde el número de *round-trip times* es fijo entre episodios de descarte de paquete.

Los bruscos incrementos en las técnicas presentadas tienen ciertas desventajas, por ejemplo, en un entorno donde esté actuando Drop-Tail podría producir un gran número de paquetes descartados en un solo evento de pérdida, en cambio, para ambientes que usan AQM el problema sería la incapacidad del sistema para evitar las fluctuaciones que con los grandes incrementos se producirían. La agresividad de HighSpeed puede llevar a que ocurran pérdidas sincronizadas en múltiples flujos lo que repercutiría negativamente en el *throughput* del sistema.

En conclusión, el mecanismo HighSpeed está basado en la definición de tres parámetros, la fijación de estos determina ciertas ventajas y desventajas en el funcionamiento del protocolo, observándose en algunos aspectos mejor o peor desempeño que Standard TCP según las circunstancias.

3.7.2. Extensión de TCP para Trayectos de Delay Largo

Este trabajo fue desarrollado por V. Jacobson y R. Braden en 1988 [10]. Fue motivado porque en ese entonces TCP funcionaba apropiadamente para canales ente 800 Mbit/s y 300 bit/s, pero tenía problemas de desempeño cuando el ancho de banda era alto y el *round-trip delay*(RTT) era largo. A esa condición se le denominó *"high bandwith-delay product"* y se presenta más precisamente cuando el producto de los parámetros, ancho de banda y *delay*, supera 10^5 bits. Los canales que tienen estas características se denominan *"long, fat pipe"* y a la red de la que forman parte, LFN. Algunos ejemplos de casos donde se observa este inconveniente son los canales de alta capacidad en satélites y canales de fibra óptica.

El algoritmo TCP presenta principalmente tres problemas en entornos de este tipo. El primero es la restricción en el tamaño de la ventana pues el TCP *header* usa 16 bit para reportar la magnitud de la que ha sido recibida al emisor y, por lo tanto, la ventana de mayor tamaño que se puede utilizar es de 65 Kbytes, como solución se propuso aceptar ventanas de mayor tamaño implementando un factor multiplicativo en el *header* para comunicar el verdadero tamaño de la ventana. El segundo problema en TCP es la retransmisión innecesárea de algunos paquetes que han llegado al *receiver* pero que han sido encolados para lo que se plantea como mejora un algoritmo, extensión de TCP, conocido como *Selective Acknowledgement* y que permite que se retransmitan efectivamente sólo los paquetes que se han descartado. El tercer problema es la medición del *round-trip time*, la medición de este parámetro es importante para la adaptación del tráfico y para evitar inestabilidad. Para ahorrar esfuerzo computacional este valor suele ser calculado sobre un segmento de la ventana, pero esto se vuelve insuficiente para canales LFN donde la estimación no alcanza a ser buena y que empeora cuando hay errores, para resolver esto se sugiere el protocolo TCP *echo* donde cada segmento lleva su propia marca de tiempo.

El *Window Scale Factor* es el factor de escalamiento de la ventana para poder comunicar el recibo de tamaños mayores de esta al emisor. Una manera posible de implementarlo es a través de la definición de una opción inicial llamada *Window Scale* de TCP. Esta opción se aplica a los segmentos SYN lo que es adecuado pues su transmisión es confiable. Se utiliza una opción de tres bytes en un segmento SYN donde el primero indica que está listo para hacer ambos envíos; el segundo muestra el factor de escala a la ventana recibida, codificado como potencia de dos.

El protocolo de ACK Selectivos que extiende de TCP utiliza dos *TCP options* para aminorar el impacto sobre el protocolo original. Las *TCP options* se transmiten a través de un segmento SYN. El primero es *SACK-permitted* que señala que puede usarse la opción cuando se establece la conexión y el otro es *SACK-option* que se envía en una conexión ya establecida posterior a *SACK-permitted*. *SACK option* transmite información desde el *receiver* hasta el *transmitter* acerca de los bloques no contiguos que se han recibido y encolado, esperando que la información faltante llegue sea recibidaa. El proceso de avance del campo del número de *Acknowledgement* no es modificado y permanece tal como en TCP. Las variaciones del algoritmo y la nueva información enviada permiten optimizar el proceso de retransmisiones.

Los bloques de información son expresados sobre *SACK Option* por dos enteros de 16 bit estos son el origen relativo y el tamaño del bloque. El origen relativo se refiere al primer

número de sequencia del bloque en relación al campo *Acknowledgement Number* y el tamaño del bloque es el tamaño en octetos del bloque. Se puede aplicar *Window Scaling* en SACK de dos formas: expandiendo el campo de SACK option a 24 o 32 bits, o bien, escalando los campos de SACK por el mismo factor que la ventana. Entre las dos formas mencionadas la más apropiada es la que utiliza factor de escalamiento pues, a diferencia de la otra, no disminuye el número de bloques posibles de comunicar. Sin embargo, el segundo método tiene la desventaja de aumentar la imprecisión de la información al expresar los orígenes y la longitud de los bloques como múltiplos del factor de escalamiento $S = rcv.scal$e. El problema de precisión del método no es importante si hay seguimiento de los límites de los segmentos y en caso de que no existiese solo se retransmitirían innecesariamente una cantidad pequeña.

El método de *SACK option* evita las retransmisiones a través de un *flag bit* o "ACK'd"que forma parte de cada segmento. En el transmisor cuando un segmento es enviado este se agrega a la cola de retransmisión con el "ACK'd"bit en *off*, este bit se encenderá al recibirse *SACK option*, y así se prevenirá que sea enviado inútilmente. Lo anterior se cumplirá para cada segmento perteneciente a cada bloque indicado en *SACK option*.

TCP echo es un mecanismo para medir el RTT de un segmento de una manera más precisa. Consiste en aplicar una marca de tiempo a un segmento que el *receiver* debe devolver en el ACK correspondiente. Esta opción requiere 4 bytes que son enviados de vuelta con la misma cantidad de bits por medio de la opción *TCP Echo Reply*.

TCP echo también debe medir correctamente el RTT en los casos donde no existe una relación inyectiva entre la información enviada y los segmentos ACK situación que analizado en la aplicación del método. En ciertos casos el acuse de recibo se realiza en TCP cada cierta cantidad K de segmentos, se debe medir el RTT efectivo para evitar retransmisiones innecesáreas, lo que se logra replicando el *Echo option* desde el segmento más antiguo no recibido. Otra situación posible es que exista un paquete que se haya perdido de la secuencia y que, por lo tanto, *Echo Reply option* transmita el RTT de un paquete reciente que no corresponde con el número de secuencia indicado en el ACK, pues este es el paquete perdido. La última condición considerada es que en presencia de una pérdida al calcular el RTT desde en un segmento más antiguo se pueda estar incluyendo el tiempo de *timeout* de la retransmisión, lo que hace imprecisa la estimación.

Se propone para enfrentar lo anterior y según sea el caso que el *Echo option* se retorne desde el segmento más antiguo sin acuse de recibo, o bien, desde el más reciente. Para ello se realizan implementaciones en el *receiver* y el *sender*. En el *receiver*se agrega un espacio de 32 bits a la información de *Echo option* llamada rcv.echodata y también una bandera que indica si existe el rcv.echodata llamada rcv.echopresent, al verificar la existencia de rcv.echopresent se añade una opción *echo-reply* que contiene rcv.echodata. Si el segmento contiene un *echo option* pero no tiene activado rcv.echopresent, el valor de *echo option* se copia a rcv.ecodata y se activa rcv.echopresent. Si no se encuentra activado rcv.echopresent y si el segmento está a la izquierda de la ventana, *echo option* se copia a rcv.echodata en caso contrario se ignora. La implementación en el *sender* consiste en un bit bandera denominado snd.echoallowed que es activado si se recibe un SYN con *TCP echo option*.

3.7.3. Scalable TCP

Scalable TCP [7] es un algoritmo propuesto por Tom Kelly que surgió como respuesta al mal desempeño de TCP en redes con alto *bandwidth-delay product* (BDP,producto retardo-ancho de banda) debido a su lenta respuesta y al gran tamaño que la ventana de congestión alcanza. El mecanismo propuesto logra mejor utilización en redes de alta velocidad. Además utiliza los mismos *routers* y *receivers* que TCP tradicional y es capaz de comportarse como Standard TCP cuando el tamaño de las ventanas es pequeño.

Al algoritmo Standard TCP en las redes de HighSpeed le toma largo tiempo alcanzar la velocidad de envío anterior a un evento de pérdida. Este lapso, llamado tiempo de recuperación, es además mucho mayor que los tiempos entre periodos transientes de congestión y, por lo tanto, se desencadena una baja utilización de la capacidad de canal incluso cuando no hay congestión. No obstante, si se modifican los parámetros que controlan la evolución de la ventana se puede disminuir el tiempo que demora restauración de la magnitud de la ventana.

TCP bajo determinadas características de red se desempeña de forma adecuada, tiene la capacidad de utilizar agresivamente el ancho de banda disponible y de que este sea compartido por distintos flujos con similar *round-trip time* en forma equitativa. Esto lo logra por medio de su mecanismo de ajuste de la ventana también conocido como *Additive Increase/-Multiplicative Decrease*. Sin embargo, la actualización de ventana no opera apropiadamente y presenta problemas en redes con alto BDP. En escenarios donde la velocidad es superior a 100 Mbps y el *round trip time* toma valores sobre 50ms, el tiempo que tarda la ventana en recuperarse es demasiado alto. Para solucionar esto se propuso en el trabajo de Kelly modificar el algoritmo característico de TCP cambiando los parámetros que le caracterizan. Así en forma general, el incremento de la ventana por cada *acknowledgement* recibido queda ahora representado por 3.172 y análogamente la variación de la ventana al detectar congestión se cambia a 3.173.

$$cwnd \leftarrow cwnd + a \tag{3.172}$$

$$cwnd \leftarrow -[b * cwnd] \tag{3.173}$$

En las ecuaciones 3.172 y 3.173, a y b son constantes tales que $0 < a < 1$ y $0 < b < 1$. En Scalable TCP el tiempo de recuperación de pérdida es proporcional solamente al *round-trip time* de la conexión a diferencia de TCP donde este intervalo es proporcional al tamaño de la ventana de conexión y al *round-trip time*. Es esta propiedad de invarianza respecto del tamaño de la ventama lo que hace a Scalable TCP idóneo para las redes HighSpeed. Como resultado de las implementaciones y pruebas del algoritmo se recomienda en este trabajo el uso de los parámetros $a = 0{,}01$ y $b = 0{,}125$.

Scalable TCP y TCP Standard tienen distintos valores en el promedio de sus ventanas para distintas magnitudes de pérdida P_r pues cada una tiene una función de respuesta diferente. La función de respuesta es aquella que relaciona el tamaño de la ventana de congestión con la velocidad de pérdida. La que corresponde a Scalable TCP es 3.174 y la función de respuesta

de Standard TCP está expresada en 3.175.

$$cwnd_r \approx \frac{a}{b}\frac{1}{Pr} \tag{3.174}$$

$$cwnd_r = \sqrt{\frac{1{,}5}{P_r}} \tag{3.175}$$

El objetivo del comportamiento distinto es, de hecho, lograr una mayor utilización del ancho de banda disponible. TCP Standard no puede alcanzar tamaños de ventana superiores a cierto valor denominado *legacy window size* $lwnd$, pues el tamaño del *buffer* no se lo permite. Scalable TCP, en cambio, puede superar $lwnd$ y se define como umbral para que el protocolo de transporte modifique el algoritmo de actualización de ventana que aplica. Es decir, si $cwnd \leq lwnd$ se ajecuta el mismo mecanismo de variación de ventana que TCP y si $cwnd > lwnd$ entonces se usa el nuevo método descrito. Por otro lado, la variable p_l indica la máxima probabilidad de pérdida que mantiene el tamaño de la ventana sobre $lwnd$. Para asegurar que la curva de respuesta sea decreciente, esta debe pasar por el punto $(p_l, lwnd)$ lo que establece una relación entre a y b que se expresa en la ecuación 3.176.

$$\frac{a}{b} = p_l * lwnd \approx \sqrt{1{,}5p_l} \tag{3.176}$$

Otra propiedad importante a considerar en el diseño del protocolo es la variación instantánea de la velocidad. Esta fluctuación permite al protocolo sondear la capacidad disponible pero se recomienda que mantenga un valor pequeño. El estudio de esta característica y su relación con los parámetros de diseño de Scalable TCP indican que b debe ser escogido pequeño para lograr menor oscilación de velocidad y como se desea que estas sean menores que las de TCP tradicional entonces se concluye que b debería elegirse cumpiendo la relación $b \leq \frac{1}{2}$. Además b debe ser mayor a cero para satisfacer la condición de la ventana de que su variación dure más de un *round-trip time*.

La convergencia del protocolo se refiere al tiempo que toma responder a los cambios en la capacidad disponible en la red. En presencia de un evento de congestión y el incremento de P_r la fuente disminuirá la velocidad de envío de paquetes en un factor de $\frac{1}{2}$ en un tiempo menor al presentado la ecuación 3.177.

$$\frac{log\left(\frac{1}{2}\right)}{log(1-b)} \tag{3.177}$$

Ante la situación contraria, es decir, el incremento en la capacidad del canal la velocidad de envío aumentará en un factor de dos en tiempo expresado en 3.178.

$$\frac{log(2)T_r}{log(1+a)} \tag{3.178}$$

En cambio, para responder a la misma variación Standard TCP requiere $cwnd_r(t_0)$ *round-trip times* lo que confirma la superioridad de Scalable TCP en la convergencia en escenarios de transmisión donde las ventanas alcanzan gran tamaño. De las ecuaciones se desprende que es conveniente para reducir el tiempo de reacción escoger los parámetros a y b con valores grandes, sin embargo, esto resultaría inapropiado para mantener la variación instantánea de

la velocidad baja. En el trabajo de Kelly se escogió el valor $b = \frac{1}{8}$ pues tiene un buen equilibrio en ambas las características.

Fue demostrado en [84] que el algoritmo Scalable TCP es localmente estable entorno al equilibrio dado por la expresión 3.179.

$$a < \frac{p_j(\hat{y}_j)}{\hat{y}_j p'_j(\hat{y}_j)} \qquad \forall j \in J \tag{3.179}$$

Donde $\hat{y}_j$ es la velocidad de equilibrio de cada *link*, $p_j(y)$ corresponde a la probabilidad de pérdida del link j-ésimo para una velocidad y donde J corresponde al conjunto de *links*. Además se puede construir un sistema estable Scalable TCP mediante el uso de ECN configurando la red de *buffer* según se afirma en otro trabajo del mismo autor [85].

Los experimentos que consistieron en el envío de un archivo entre Chicago y Génova usando 4 *servers* y sus respectivos *receivers*. Se compararon tres casos variando el *sender*: TCP en *kernel* de Linux 2.4.19 inalterado, TCP en kernel de Linux 2.4.19 con modificaciones de *gigabit*, TCP Scalable en el kernel de Linux 2.4.19 con modificaciones de gigabit. La variación del kernel se refiere al cambio del funcionamiento de un *driver* de Linux para privilegiar la velocidad a la eficiencia al espacio. Se observó que el *throughput* se incrementa de un 34 % a un 175 % usando Scalable TCP respecto de TCP con modificación de gigabit en el kernel respectivamente . Usando un flujo en cada una de las cuatro máquinas mediante Scalable TCP se obtiene el 78 % del rendimiento máximo, en cambio, la transmisión de 16 flujos en las cuatro máquinas Linux 2.4.19 con modificaciones del kernel sólo alcanza el 61 % del mejor desempeño posible y el kernel Linux 2.4.29 standard apenas llega al 38 % del máximo.

En la investigación de Kelly también se hicieron experimentos para analizar si el uso de TCP Scalable perjudica a los usuarios de Standard TCP y se llegó a la conclusión de que el efecto que tiene el protocolo estudiado sobre el rendimiento de TCP tradicional se puede considerar despreciable. Por ejemplo, el *throughput* TCP en Linux que aplica las modificaciones del *kernel* se encontró que obtiene un 52 % de eficiencia respecto del máximo mientras que el tráfico que usa Scalable TCP alcanza un total transferido de 75 % respecto del máximo posible.

3.7.4. Adaptative Delay-based Congestion Control

Es un algoritmo que fue presentado por Hyungsoo Jung et. al.[20], se caracteriza por usar la estimación del tamaño de la cola en el cuello de botella y además la medición del *fairness* en su ejecución para inferir el grado de congestión. Tiene como objetivo mejorar el desempeño en redes *High Bandwith-Delay Product* en cuanto a las características de utilización, rapidez de convergencia y uso equitativo del ancho de banda en el cuello de botella.

Existen diversos algoritmos que han buscado solucionar los problemas que se generan en Standard TCP en escenarios *High Bandwith-Delay*, estos se pueden clasificar en protocolos de tipo *router-supported* y *end-to-end*. El segundo grupo, al que pertenece el mecanismo propuesto por Hyungsoo Jung et. al. expuesto en este apartado, tiene la ventaja de no requerir condiciones de compatibilidad en *routers*, sin embargo, necesita alcanzar un comportamiento

TCP-friendly para no perjudicar las transmisiones bajo TCP que son predominantes en la red. La categoría de algoritmos *end-to-end* a su vez se puede subdividir en dos: *Delay-based Congestion Control* (DCC) y *Packet loss-based Congestión control* (PCC).

Los protocolos DCC utilizan las fluctuaciones en la variable RTT para distinguir congestión. Algunos algoritmos pertenecientes a DCC son TCP-Vegas [86] y FAST [26] ambos mecanismos lograron aumentar el *throughput* considerablemente respecto de TCP-Reno, pero también los dos presentan problemas de *fairness*. Por otro lado, los mecanismos PCC basan su funcionamiento en las pérdidas de paquetes como indicador de congestión. La mayoría de los algoritmos PCC usan como actualización de la ventana el sistema AIMD, que si bien asegura *TCP-friendliness* no logra buena utilización en redes de *High-Bandwidth product*. En esta categoría ciertos ejemplos de algoritmos propuestos para superar las desventajas mencionadas son HSTCP[27], HTCP [28], STCP [7], BIC[29], CUBIC [30], LTCP [31], etc, algunos de los cuales utilizan la medición de ciertas variables escogidas para calcular el parámetro de incremento.

Los protocolos PCC y DCC, que distiguen congestión a partir de la pérdida de paquetes y el delay respectivamente, están en desventaja respecto de los de la categoría *router-supported* pues no tienen la capacidad de identificar el estado de carga del cuello de botella y, en cambio, usan técnicas de testeo para encontrar el ancho de banda disponible incrementando y disminuyendo en tamaño de la ventana ante la presencia de un indicador de congestión. El método de incremento aditivo de TCP no es tan rápido como se quisiera para encontrar y ocupar el ancho de banda libre. Los protocolos HSTCP, STCP, CUBIC y, en general, aquellos con incremento multiplicativo tienen aumentos de ventana más rápidos que TCP, sin embargo en algunos casos presentan problemas de estabilidad. Para enfrentar la dificultad de mantener la estabilidad en el trabajo de Jung et. al. se propuso una medida denominada *Fairness Ratio* (FR), que es el cuociente entre el ancho de banda actual y el ancho de banda compartido en equilibrio. De esta medición resulta, por ejemplo, que se permita a un flujo con bajo FR aumentar su ventana con mayor agresividad para obtener una convergencia rápida y equidad al compartir el ancho de banda.

Otro aspecto considerado en la propuesta fue la reacción del protocolo ante las pérdidas. En casos donde se usan los algoritmos PCC la respuesta a un paquete pérdido es el decremento multiplicativo de la ventana. Este proceso lleva a reducir más de lo necesario la utilización del canal y, por lo tanto, los investigadores sugirieron observar la diferencia entre la velocidad de envío (*throughput*) y la velocidad de recibo (*goodput*) para cuantificar el grado de congestión en el cuello de botella y decidir en cuánto disminuir el tamaño de la ventana.

El protocolo ACP requiere una serie de medidas y parámetros para su funcionamiento que le permiten sondear el estado de la red. Los dos principales son el crecimiento de la cola y el *Fairness Ratio*. Para deducir si existe congestión se define la medida $\Phi = goodput - throughput$ en una época. Cuando el *link* recibe paquetes a una velocidad mayor que su capacidad entonces $\Phi < 0$ lo que significa que la red está congestionada y t_{d}, el retardo *forward delay*, se incrementa. Por el contrario, el caso $\Phi > 0$ señala que el *buffer* del cuello de botella se ha vaciado más rápido que la llegada de paquetes. Además se hace seguimiento de las variaciones del *forward delay* para estimar la longitud de la cola en el cuello de botella. El protocolo además requiere de la variable denominada estimación de crecimiento de la cola

que se generaliza denotándose Q^T 3.180 para calcular la estimación dado cualquier periodo T.

$$G^T = \frac{P^T}{T+\Delta t_{\mathrm{d}}} \quad Q^T = G^T \Delta t_{\mathrm{d}} \tag{3.180}$$

Donde G^T 3.180 representa el *goodput* en un un periodo T, P^T es el total de paquetes enviados en el periodo y ΔT_{d} es el incremento en el *forward queuing delay.*

La medición de *Fairness Ratio Estimation* tiene por objetivo mejor la velocidad de convergencia la ancho de banda de disponible. Se puede inferir si se está usando el ancho de banda más allá de lo que se considera *fair* si el incremento del *delay* es mayor de lo que se espera y, por el contrario, un incremento del *delay* menor de lo esperado indica infrautilización del canal. Según lo anterior, se define el *Fairness Ratio Estimation* como:

$$\hat{F}_{\mathrm{i}} = \frac{Q_{\mathrm{i}}^{t_c}}{\Delta cwnd}\,, \tag{3.181}$$

donde $Q_{\mathrm{i}}^{t_c}$ es el crecimiento de la cola durante el tiempo t_c para el i-ésimo flujo. Las dos principales variables que detectan el estado de la red y discriminar la acción que toma el protocolo son Φ y F según los valores que estas alcancen, que definen seis estados, se ejecutan determinadas acciones representadas por las ecuaciones 3.182. Debido a que se ajusta a las diferentes situaciones el protocolo toma el nombre de .ªdaptativoz las acciones son llamadas Incremento adaptativo y Decremento adaptativo.

$$\begin{aligned} AI: &\quad cwnd(t+tc) = cwnd(t) + f_{AI} \\ AD: &\quad\quad cwnd \leftarrow cwnd(t) - Q(t) \end{aligned} \tag{3.182}$$

Si $\Phi > 0$ la red no está congestiondad y, por lo tanto, la acción permite incrementar rápidamente la ventana en α segmentos por un periodo t_c,

$$f_{AI}(t) = \begin{cases} \alpha & si \quad \Phi \leq 0 \\ \alpha & si \quad \Phi < 0, \hat{F} \geq 1 \\ \alpha + \kappa(1-\hat{F})^2 & si \quad \phi < 0, 0 \leq \hat{F} < 1 \end{cases} \tag{3.183}$$

donde $\alpha > 0$,$\kappa > 0$. La función de f_{AI} 3.183 es lograr rápida convergencia pero a la vez satisfacer la propiedad de fairness y por lo tanto la función F_{AI} debe tomar distintos valores según las condiciones en que se encuentre la red. Cuando $\Phi < 0$ la congestion en la red es inminente y el valor de la función f_{AI} se decide según el fairness ratio. Si $\hat{F} \geq 1$ se incrementa el tamaño de la ventana en α segmentos, en cambio, si $0 \leq \hat{F} < 1$ el incremento es mayor que en el caso anterior dependiendo en magnitud del fairness ratio F de modo que más grande es cuanto más se acerque $\hat{F}$ a cero, y además del parámetro κ cuyo valor incidirá en la convergencia del protocolo.

En el algoritmo se introdujo un estado para conservar la propiedad de TCP-*friendliness* aun cuando la utilización del canal es alta a este se le denominó early control state este ocurre en dos casos: cuando Q alcanza cierta fracción de la ventana de congestión y cuando un flujo detecta que otro comenzó una disminución del tamaño de su ventana. En el segundo caso ocurre un aumento de goodput en el periodo t_c que se denota por ϕ

Se realizaron experimentos donde se comparó el rendimiento de ACP respecto de TCP-NewReno, CUBIC, FAST, HSTCP sobre un régimen de encolamiento drop tail. Se puedo

observar que ACP logra un mejor desempeño en cuanto a throughput, utilización, tamaño de las colas y de las ventanas de congestion. Se estudió la convergencia a través de una simulación en la que el cuello de botella es de 300Mbps y donde confluyen cinco flujos, esto permitió comprobar la adecuada reacción del protocolo a los cambios en la red, además se hizo pruebas variando los tipos de flujos en el tiempo con que se comprobó el rápido tiempo de reacción del protocolo con el que alcanzó alta utilización y fairness.

Se midió la eficiencia de los protocolos para diferentes capacidades de buffer del bandwith-delay product para 100Mbps y 1Gbps y se mostró la superioridad en la utilización alcanzada por los protocolos CUBIC y ACP y la incapacidad de FAST para obtener una utilización aceptable para un bajo largo de cola. Para analizar la propiedad de *TCP-Friendliness* en ACP se hicieron competir flujos TCP con flujos ACP en un link de cuello de botella 500Mbps y se pudo ver que ACP es tan *TCP-friendly* como otros protocolos considerados y es capaz de mantener alta utilización del canal.

3.7.5. Carrier Ethernet Orientado a Protocolo de Transporte

La motivación para investigar *Carrier Ethernet* se basa en las convenientes propiedades con que esto cuenta, entre las que se pueden nombrar, escalabilidad, bajo costo de operación, flexibilidad, entre otros. Mediante esto son posibles canales de alto ancho de banda y bajo costo, sin embargo en estas condiciones de *Bandwith delay Product*, TCP no opera correctamente y se recurre al diseño del protocolo ESTP [87]. Este consiste en el ajuste de la ventana según el estado de la red inferido en base a información en la capa de transporte exclusivamente.

Muchos protocolos habían sido desarrollados para enfrentar las dificultades de TCP en circunstancias de BDP, pero estos eran poco adecuados ,o bien, incompatibles con *Carrier Ethernet Networks* (CEN) como, por ejemplo, aquellos que dependen del router. ESTP se basa en un método que permite estimar el *throughput* usando información de cwnd, la ventana de congestion, y del *round-trip time*.

La capacidad en un canal de tipo CEN es analizada definiendo un modelo de tráfico donde los paquetes que ingresan a Available bit rate (ABR) se marcan con prioridad alta, en cambio, un paquete que sale de la región CIR será marcado con baja prioridad. Y los paquetes tienen distinta probabilidad de pérdida según si fueron marcados con prioridad alta o baja que corresponden a probalilidad p_1 y p_2 respectivamente. Así se puede representar la capacidad como:

$$C = CIR(1 - p_1) + EIR(1 - p_2)\,. \tag{3.184}$$

El límite inferior es:

$$C_{min} = CIR(1 - p_1)\,. \tag{3.185}$$

El algoritmo ESTP pretente incluir la información de los parámetros QoS para mejorar el control de transporte de datos. Consiste en tres etapas: la primera de estimación del nivel de congestion, la segunda donde se hace un mapeo y la tercera , una fase de *congestion avoidance*.

El nivel de congestión se mide contabilizando los paquetes enviados correctamente que se denota por α entre dos pérdidas que luego se mapea para encontrar el factor de decremento multiplicativo. Estrictamente α es la cantidad de paquetes entre pérdidas más uno y por lo tanto esta variable puede tomar los valores en el intervalor [1, ∞[y el intervalo al que sea mapea corresponde a [1,2], es decir, el factor de decremento multiplicativo según el protocolo puede estar en ese rango. El mapeo se realiza de modo que al haber gran congestión el factor sea elegido grande para disminuir de manera más agresiva el envío de paquetes y ,en caso contrario, cuando no hay congestión y, por lo tanto, α es grande entonces se hace decrecer la ventana en menor grado, sin embargo, nunca se alcanza el factor uno, pues el intervalo es abierto en el infinito.

Para obtener la función de mapeo se considera la función distribución de probabilidad que caracteriza el tiempo entre dos paquetes perdidos, esto se modela como exponencial pues se asume que la cantidad de paquetes perdidos tienen una distribución de Poisson. La función distribución se expresa como:

$$f(\alpha, \lambda) = \begin{cases} \frac{1}{\lambda} \mathrm{e}^{-\frac{\alpha}{\lambda}} & \alpha \geq 0\,, \\ 0 & \alpha < 0 \end{cases} \tag{3.186}$$

donde α es $E[\alpha]$. Para definir la función de mapeo se tienen en cuenta las condiciones de borde. $map(1) = 2$ y $lim_{\alpha \to \infty} map(\alpha) = 1$. Y con ello se obtiene que la función de mapeo es:

$$map(\alpha) = \mathrm{e}^{-\frac{\alpha - 1}{\tau}} + 1\,. \tag{3.187}$$

Y por lo tanto la variación en cwnd queda:

$$cwn\mathrm{d}_{n+1} = cwn\mathrm{d}_n \left(\mathrm{e}^{-\frac{\alpha - 1}{\tau}} + 1 \right)^{-1}\,. \tag{3.188}$$

Al agregar la información de los servicios Ethernet se obtiene la relación 3.189 para el decrecimiento multiplicativo que tiene ciertas ventajas en el rendimiento en cuanto a throughput.

$$cwn\mathrm{d}_{n+1} = \frac{cwn\mathrm{d}_n - cwn\mathrm{d}_{MIN}}{\mathrm{e}^{-\frac{\alpha - 1}{\tau}} + 1} + cwn\mathrm{d}_{MIN} \tag{3.189}$$

$$cwn\mathrm{d}_{n+1} = \frac{cwn\mathrm{d}_n - cwn\mathrm{d}_{MIN}}{\mathrm{e}^{\frac{\alpha - 1}{\tau}} + 1} \tag{3.190}$$

Donde $cwn\mathrm{d}_{MIN} = RTT \cdot CIR$. La expresión para el incremento aditivo sería

$$cwn\mathrm{d}_{n+1} = min \left(cwn\mathrm{d}_{MAX}, cwn\mathrm{d}_n + \frac{1}{cwn\mathrm{d}_n} \right) \tag{3.191}$$

donde $cwn\mathrm{d}_{MAX} = RTT \cdot EIR$ Usar estas ecuaciones en el algoritmo asegura que el throughput se mantenga sobre CIR y bajo EIR. El throughput en ESTP está ligado al modelo de canal definido, este contiene una bifurcación donde el tráfico que es menor a CIR se envía por la ruta 1 y si es mayor a CIR se envía por la ruta 2 que deternminará el nivel de pérdida al que está sometido. El throughput de best effort se puede expresar como

$$T_{BE} = \frac{1}{RTT \sqrt{\frac{2b(1-\beta)}{\gamma(1+\beta)} p}}\,. \tag{3.192}$$

Por otro lado el throughput de alta prioridad corresponde a $T_{HP} = CIR$. Para encontrar el throughput en estado estacionario se debe resolver la ecuación

$$T = CIR + \frac{1}{RTT\sqrt{\frac{2b(1-\beta)}{\gamma(1+\beta)}\left(\frac{T-CIR}{T}p_2 + \frac{CIR}{T}p_1\right)}}. \tag{3.193}$$

En los experimentos realizados permitieron comprobar que el protocolo mantiene el throughput en una banda a diferencia de TCP que no es capaz de alcanzar la completa utilización del canal. Se pudo ver que las estimaciones teóricas de throughput promedio para TCP y para ESTP coinciden con cierto error aceptable con los resultado de la simulaciones. Además se compararon los resultados para los protocolos TCP-SACK y ESTP en un canal de tipo CEN, se realizaron simulaciones donde se varió el parámetro RTT de 5ms a 100ms, se pudo ver que los resultados teóricos se aproximan a las simulaciones resultantes . Se pudo observar que al comparar ESTP con TCP-Sack el primero siempre supera en el throughput obtenido y que al variar la magnitud del RTT TCP-Sack disminuye de forma más rápida la utilización alcanzada.

3.8. Últimos Avances y propuestas

En la actualidad se sigue investigando en torno a TCP, muestra de ello es que se sigue publicando acerca del estudio comparativo de desempeño de propuestas de protocolos de tipo end-to-end como TCP Reno, TCP Vegas y TCP SACK bajo diferentes condiciones [88]. Así como también, se han presentado investigaciones y análisis de protocolos *end-to-end* que utilizan la medición del *delay* debido a sus ventajas respecto de aquellos que infieren la congestión a partir de las pérdidas de paquetes [89] .

En los años recientes también han aparecido nuevos protocolos basados en TCP para enfrentar nuevos problemas que han surgido del uso de nuevas tecnologías. Las redes móviles, 5G y la internet de las cosas han puesto nuevos desafíos en el funcionamiento de TCP.

Un ejemplo de propuesta para hacer frente a los desafíos de las nuevos avances es DL-TCP [90], una técnica que utiliza *Deep Learning* para ser aplicado en redes 5G que es reconocida como una red adecuada para la transmisión de información en casos de desastre. La banda donde trabaja 5G es susceptible a los obstáculos produciéndose cortes que degradan el desempeño de TCP. El Protocolo DL-TCP trabaja prediciendo las desconexiones de la red, gracias a que aprende de la información de se moviliza y de la fuerza de la señal ajustando la ventana de acuerdo a eso. Y tiene un rendimiento mejor en cuanto estabilidad y *throughput* que TCP NewReno, TCP Cubic y TCP BBR. También Dynamic TCP [91] es una aplicación diseñada para encarar las dificultades en condiciones de canal microndas y LTE, este trabaja estimando las fluctuacions del ancho de banda disponible en el canal. En condiciones de redes *wireless* las pérdidas aleatorias de paquetes son considerables lo que degrada el desempeño de TCP, *Dynamic* TCP es propuesto para contrarrestar aquello. Este algoritmo estima el ancho de banda del cana y la ventana de congestion varía de manera adaptativa de acuerdo a lo estimado.

Varios de los métodos propuestos recientemente utilizan técnicas de *machine learning* como por ejemplo TCP-DQN que se basa en *Deep Q network* [92] en el cual el proceso de TCP se abstrae como un proceso de Markov de decisión parcialmente observada y el tamaño de la ventana de congestión se ajusta según el estado observado. Y se utiliza una función de recompensa para balancear el *delay* y el *throughput* y una doble capa neuronal. Se obtiene como resultado la disminución de las oscilaciones. Otro protocolo que utiliza inteligencia fue llamado NeuRoc [93] y se caracteriza por usar el *changepoint* y una *deep neural network* para una inferencia rápida, se utiliza un metodo de *reinforcement learning* para predecir el *bandwidth delay product* y ajustar la ventana. Esto permite acercarse al punto de operación óptimo conocido como Kleinrock. También se puede mencionar un método de discriminación de pérdida basado en *machine learning* ML-LDA [94] en el cual se usa un perceptrón de multicapa para detectar pérdidas por congestión.

Por otro lado, se han presentado investigaciones de protocolos específicos para aplicaciones en IoT como el protocolo MQTT [9] que utiliza TCP en el transporte. Para mejorar el desempeño de TCP en redes heterogeneas usadas para aplicaciones IoT se planteó un protocolo basado en un modelo de decisión de Markov [95]. Este protolo se nombra TCP-Siam y es una mezcla de TCP-Illinois y modificado con un estado MDP.

4. Diseño e Implementación

4.1. Introducción

La hipótesis que se demuestra en este trabajo es que modificando el algoritmo Congestion Avoidance utilizando métodos del control de sistemas para variar los parámetros AI y MD según la congestión medida se obtienen resultados de *throughput* y utilización del ancho de banda disponible mejores que el protocolo clásico TCP Reno en escenarios alto *Bandwidth Delay Product*.

En este capítulo se expone el diseño teórico de los métodos usados en dos protocolos de transporte basados en técnicas de control como también la correspondiente implementacion de cada uno necesaria para su cuantificación de desempeño; el primero emplea técnicas de control clásicas y el segundo, procedimientos de control difuso.

El objetivo de los experimentos planteados en esta sección y en el capítulo 4 es evaluar y comparar el funcionamiento de los dos protocolos de control de congestión propuestos respecto del protocolo clásico de transporte TCP Reno en cuanto a *throughput*, utilización de un canal compartido y *TCP-friendliness*.

Es posible dividir conceptualmente el problema del diseño de un protocolo de control de congestión en dos partes:

- Estimación del ancho de banda disponible.
- Ajuste al valor del ancho de banda disponible.

Las subsecciones en este capítulo describen la teoría, algoritmos e implementaciones de las soluciones propuestas a los dos puntos mencionados.

TCP Reno es un protocolo clásico que tiene la capacidad de ajustarse y estimar el ancho de banda disponible simultáneamente, mediante un método de *probing*, sin embargo tiene una desventaja debida al comportamiento de su función *congestion avoidance*, que produce grandes fluctuaciones en los valores de tamaño de su ventana. Así ocurrirá que al subir rápidamente el tamaño de sus ventanas llegará sucesivamente a un punto en que superarán entre ambas el ancho de banda, lo que producirá una pérdida en uno de los servidores y, por lo tanto, una caída brusca en el tamaño de esta, lo que determinará finalmente que cada servidor no pueda alcanzar en promedio el *throughput* disponible.

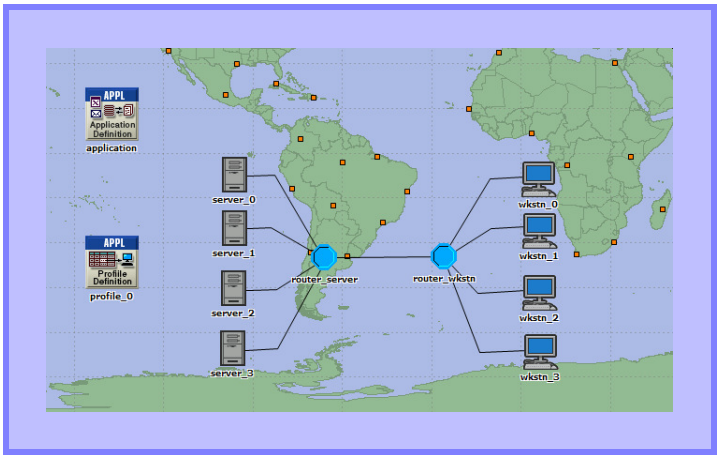


Figura 4.1: OPNET

Los experimentos diseñados en esta investigación serán realizados mediante simulaciones en OPNET, un software que permite analizar el rendimiento en redes de computadores.

Las implementaciones en este apartado consideran dos escenarios; el primero se usa en las ejecuciones del protocolo con control clásico y consta de dos servidores y dos *workstations*. El segundo comprende cuatro servidores y cuatro *workstations* que se utilizarán en las realizaciones del protocolo con control difuso y en las pruebas de *TCP-friendliness* del protocolo con controlador PD. En cada escenario se simula la transmisión de archivos: los servidores envían paquetes de información a sus respectivos *workstations* hasta completar el tamaño de archivo *FILE SIZE* escogido para la simulación, los servidores de cada escenario comparten la línea de transmisión. Los valores de *data rate* estudiados en los experimentos son de 12 Mb/s, 16 Mb/s y 24 Mb/s. Al usar un protocolo de transmisión *friendly*, los servidores dividirán el *throughput* disponible equitativamente.

4.2. Escenarios

Los escenarios usados en los experimentos de esta investigación están formados por un conjunto de servidores y *workstations* que comparten el canal por donde transmiten; los parámetros TCP de cada servidor y *workstation* deben ser configurados, los valores escogidos son los que se especifican en la Tabla 4.1; un enrutador permite enviar los paquetes de un servidor al *workstation* correspondiente. Además para que la transmisión de las unidades de información se realice de una manera equitativa y justa se emplea un algoritmo en el *router* conocido como *round-robin* que permite agendar los paquetes del *buffer* a medida que llegan cambiando el turno cada vez lo que permite alternar la salida en caso de haber paquetes procedentes de distintos servidores. Además se define en cada escenario un bloque de *application* y otro de *profile*. Un *profile* describe la actividad de un usuario o grupo de usuarios en términos de la aplicación usada en un periodo de tiempo. Es posible definir ciertos parámetros de perfil algunos de los más importantes son *Start time* que indica cuando comienza la operación de ese usuario y *Operation mode* indica como comenzará la *application*, simultáneamente o en forma secuencial. Cambiar los valores en *Start time* permitirá hacer

Maximum segment size (bytes)	1460
Receiver buffer	65535
Windows scaling (SACK)	Enables
Initial RTO (sec)	0.5
Minimum RTO (sec)	0.25
Maximum RTO (sec)	30
Timer Granularity	0.0001

Tabla 4.1: TCP Parameters

que los servidores comiencen desfasados y en un tiempo especificado.

A cada *profile* se le asocia una *application* o varias de ellas. Las *applications* son fuentes predominantes de tráfico en la red y cada una tiene un patrón de tráfico único y por lo tanto crea una carga característica en servidores y redes, el modelo usado en los escenarios de este trabajo es FTP.

4.2.1. Implementación Round Robin

El algoritmo *round-robin* es implementado en el *Process Model round-robin* Figura 4.3 del *Node Model* del *router* consta de varios estados, los paquetes de cada servidor se almacenan en cuatro colas distintas, tantas como servidores haya. En el estado *init* se inicializan las variables, en el estado *service* al cual se accede cuando la variable *SERVICE* toma el valor uno, que sucede cuando acontece un *interrupt self*, se alterna el turno de las colas y se envía un paquete de la que corresponde. En el estado *queue* se entra si *QUEUE* es uno, lo que ocurre cuando hay un *interrupt stream*, allí si la suma total de paquetes almacenados en cada cola supera cierta cantidad determinada se remueve y destruye un paquete de la cola que tiene mayor cantidad de paquetes o de aquella en que se hayan eliminado paquetes hace más tiempo; en este estado además se agenda un *interrupt self* y con ello el envío de un paquete.

4.3. Estimación del Ancho de Banda Disponible

Este tema es de suma importancia, ya que este es el valor de referencia usado en el protocolo de transporte propuesto, independiente del sistema de control implementado. La estimación del ancho de banda disponible está basada en GAIMD. Se conocen teóricamente expresiones para la velocidad de transmisión. En los casos considerados, por las condiciones escogidas para la transmisión en la modelación del escenario, las pérdidas serán producidas por *triple duplicate* ACK y no por *timeouts*, la expresión que mejor representa el *throughput* es:

$$\frac{1}{T} = TD_{\alpha,\beta}(p, T_0, b) = RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p}\,. \tag{4.1}$$

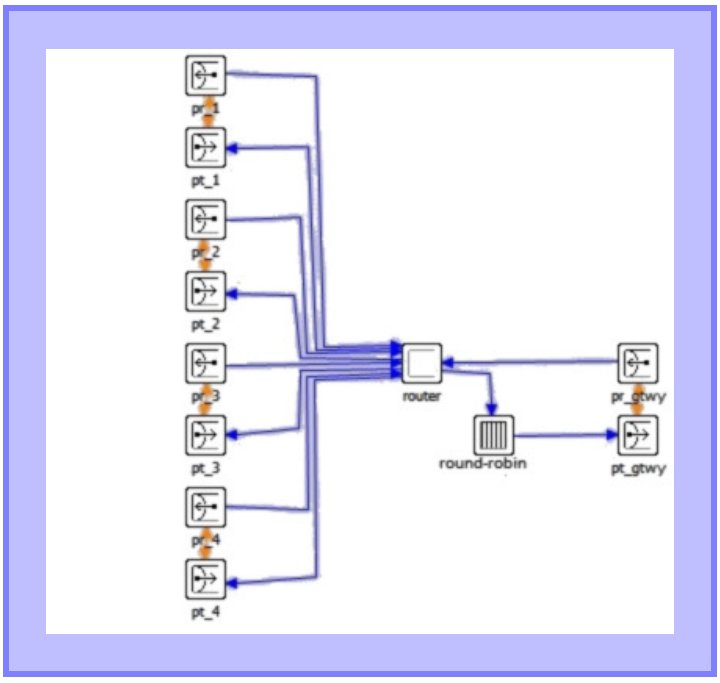


Figura 4.2: Node Model

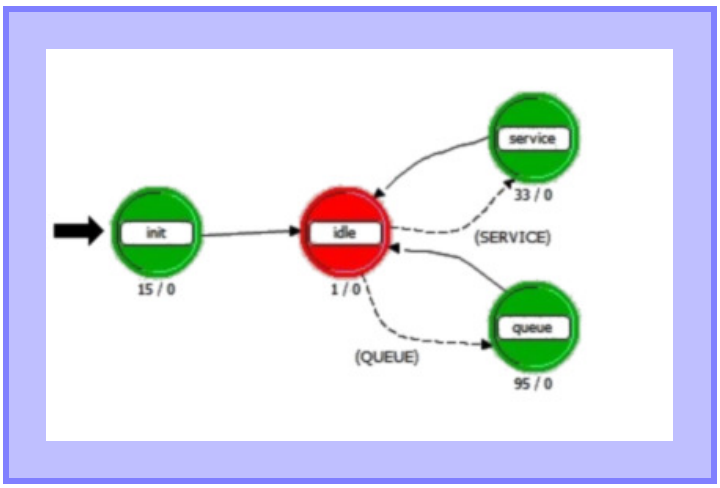


Figura 4.3: Round Robin

Ya que la transmisión debe ser *TCP-friendly* la magnitud de la velocidad debe ser la misma que para TCP por lo tanto:

$$\begin{aligned} T &= \left(RTT\sqrt{\tfrac{2b(1-\beta)}{\alpha(1+\beta)}p}\right)^{-1} \\ &= \left(RTT\sqrt{\tfrac{2b(1-\frac{1}{2})}{\alpha(1+\frac{1}{2})}p}\right)^{-1} \\ &= \left(RTT\sqrt{\tfrac{2}{3}p}\right)^{-1} \\ &= \left(\tfrac{1}{RTT}\sqrt{\tfrac{3}{2p}}\right) . \end{aligned} \tag{4.2}$$

De la ecuación 4.2 se ve que en condiciones de flujo *TCP-friendly* se puede deducir el valor de *throughput*, es decir, el ancho de banda disponible a partir de la medición de las pérdidas.

4.3.1. Implementación de la Predicción de Ancho de Banda Disponible

En TCP el *throughput* o velocidad real de transmisión está relacionado con la cantidad de paquetes enviados en cada RTT, es decir, la ventana de congestión $cwnd$. En la implementación se utiliza $cwnd_{ref}$ una medida de la variable $cwnd$ como la estimación del ancho de banda de referencia, en vez del *data rate* porque $cwnd$ es una variable de simulación a la que se tiene acceso directamente lo que permite hacer una comparación entre el valor real que alcanza y la estimación o predicción calculada.
La ecuación que relaciona T y $cwnd$ es:

$$T = \frac{cwnd}{RTT} . \tag{4.3}$$

De 4.3 y 4.2 se tiene que

$$cwnd = \sqrt{\frac{3}{2p}} . \tag{4.4}$$

La pérdida p en 4.5 se aproxima contando la distancia en paquetes entre dos pérdidas que se define como Δ, como se aprecia en la Figura 4.4.

$$p = \frac{1}{\Delta} \tag{4.5}$$

Así la ecuación 4.4 queda

$$cwnd_{predIns} = \sqrt{\frac{3\Delta}{2}} . \tag{4.6}$$

El cálculo fue realizado en la función *Fast Retransmit* de TCP `static void tcpfast_retrans (void)` porque cada vez que el programa entra en ésta, es debido a que ha ocurrido un *triple duplicate ACK*, *duplicate ACK* quiere decir que quien envía recibe más de un ACK del receptor con el mismo número, triple se refiere a que esto ocurre tres veces. Se contabiliza allí la cantidad de paquetes que se han transmitido desde la anterior ocasión en que el programa llamó a esta función y mediante 4.7 se obtiene $cwnd_{pred}(n)$ que corresponde a un

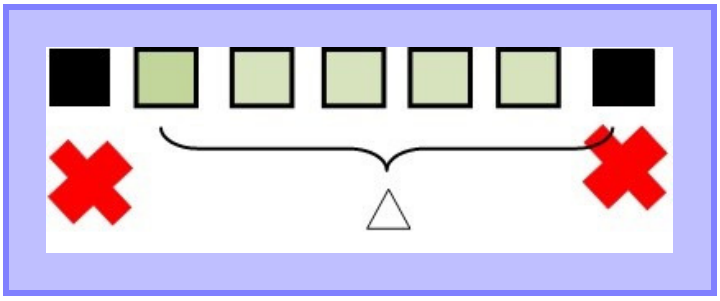


Figura 4.4: Delta, aproximación de la pérdida

promedio en el tiempo de los valores anteriores calculados.

$$cwnd_{pred}(n) = \left(1 - \frac{1}{n}\right) cwnd_{predIns}(n-1) + \frac{1}{n} cwnd_{predIns} \tag{4.7}$$

La ventana de referencia calculada como en 4.7 representa el promedio total en todo el tiempo transcurrido al momento, de los valores instantáneos de 4.6. Otra forma de obtener una aproximación de la predicción es hacer un promedio parcial que considere una ventana y no todo el lapso transcurrido 4.8.

$$cwnd_{pred}(j) = \frac{\sum_{i=j}^{j+N-1} cwnd_{predIns}(i)}{N} \tag{4.8}$$

4.4. Metodos de Ajuste al Ancho de Banda Disponible

4.4.1. Diseño Controlador PD

Para ajustarse al *throughput* disponible se propone el uso de controladores PD. Un controlador PD permite el seguimiento de una referencia en retroalimentación, midiendo el error e entre variable de salida $y(t)$ y la referencia r, compensándolo mediante la acción de la magnitud calculada de la variable de control $u(t)$. En la Figura 4.5 se muestra el esquema que representa este proceso.

Se considera como variable controlada el tamaño de la ventana $cwnd$ y como variable manipulada, la magnitud de AI, la referencia es el ancho de banda del canal expresado como tamaño de ventana $cwnd_{ref}$ en paquetes. Se escogió el valor de AI como variable manipulada porque la mayor parte de las veces que se entra en esta función, la ventana está creciendo y solo disminuye según MD cada vez que hay una pérdida, y lo que se quiere es disminuir los eventos de pérdida. En otras palabras, basándose en GAIMD, se usarán valores variables de AI para controlar el crecimiento de la ventana y disminuir las posibilidades de pérdida y ajustarse a la tasa de envío disponible. Para asegurarse que el comportamiento sea amigable se utilizarán valores de AI y MD relacionados como se indica en la ecuación 2.7, es decir, la variable independiente es AI que se denota como α y se despeja β:

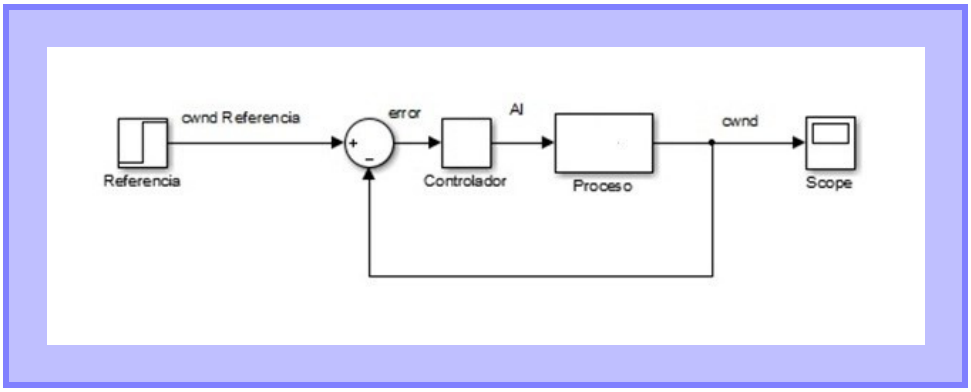


Figura 4.5: Esquema Controlador

$$\beta = \frac{3-\alpha}{\alpha+3}\,. \tag{4.9}$$

De las inecuaciones de 2.1 se tiene que:

$$\beta = \tfrac{3-\alpha}{\alpha+3} > 0 \quad y \quad \alpha > 0\,. \tag{4.10}$$

Resolviendo las inecuaciones 4.10 resulta que $\alpha \in (0, 3)$.

4.4.2. Implementación Controlador PD

La implementación del PD (Figura 4.6) se realizó en la función `static void tcp_cwnd_update` del *Process Model* `tcp_conn_v3` Figura 4.7. En esta función se actualiza el tamaño de la ventana de cada servidor cada vez que se recibe un acuse de recibo o ACK. Cuando el programa entra a la función compara el actual valor de la ventana con la referencia o predicción, calcula el error 4.11 y a partir de eso la magnitud de la variable controlada 4.12.

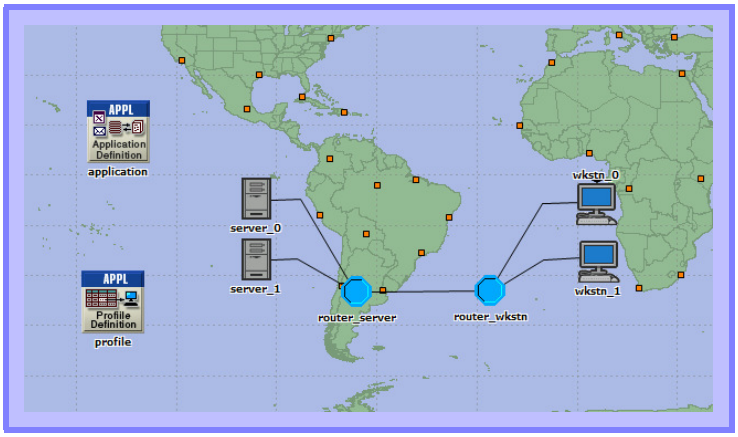


Figura 4.6: Sintonización PD

Figura 4.7: Process Model

$$\mathrm{e}(n) = cwnd_{ref}(n) - cwnd(n) \tag{4.11}$$

$$AI(n) = \mathrm{e}(n)K_p + \frac{\mathrm{e}(n) - \mathrm{e}(n-1)}{\Delta t}K_\mathrm{d} \tag{4.12}$$

En la práctica para tener mayor estabilidad se decidió usar valores de α tales que,$\alpha \in$ [0.2;2.8]. Los valores de K_p y K_d se obtuvieron mediante un método de optimización. La sintonización se realizó en el primer escenario, con dos servidores, para los valores siguientes de la variable *data rate* 12 Mb/s, 16Mb/s y 24 Mb/s.

El escenario empleado en las ejecuciones comprende dos servidores y dos *workstations*; se enviarán archivos de 250 MB, en la sintonización del controlador PD, es decir, en el proceso necesario para encontrar los valores de K_p y K_d adecuados que se detalla en la sección 4.4.3.

4.4.3. Sintonización Controlador PD: Optimización PSO Particle Swarm Optimization

Para obtener los valores de K_p y K_d se utilizó un algoritmo de optimización conocido como *Particle Swarm Optimization*. Más específicamente se usó la variante *only-social*, mencionada en la sección 2.4.3, que considera en cada iteración la posición de la partícula con mejor *fitness* para calcular el cambio en las coordenadas de estas. Se sabe que este tipo tiene como ventaja una mejor velocidad de convergencia y como desventaja cierta tendencia a quedar atrapado en óptimos locales [18].

En este caso el algoritmo fue ejecutado usando 20 partículas, cada partícula está definida por dos coordenadas que corresponden a K_p y K_d. Estas son inicializadas aleatoriamente en un rango determinado, el rango fue escogido observando la relación entre las constantes y la variable de control. Cada sumando de 4.12 debería ser menor al máximo valor de AI.

$$K_p \mathrm{e}_{max} < 3 \tag{4.13}$$

$$\frac{1}{2}\left(\frac{\sum_{j=1}^{n} |\mathrm{e}_1(j)|\,\Delta t}{T} + \frac{\sum_{j=1}^{n} |\mathrm{e}_2(j)|\,\Delta t}{T}\right) \tag{4.14}$$

$$\frac{D\mathrm{e}_{max}}{\Delta t}K_\mathrm{d} < 3 \tag{4.15}$$

La función de *fitness* escogida 4.14 representa el promedio de la sumatoria del error 4.15 en valor absoluto respecto de una referencia escogida para cada ventana o transmisión, pues la optimización se ejecutó en el escenario con dos servidores. Como los valores del error registrados ocurren a intervalos de tiempo irregulares, se ponderó este por el valor Δt transcurrido y se dividió por el valor de tiempo total T. Se busca que los tamaños de las ventanas de la cada transmisión sean mínimos respecto de una referencia dada.

Data rate [Mb/s]	$cwnd_{ref}$[paquetes]
12	51
16	68
24	102

Tabla 4.2: Valor de referencia para cada PD

$cwnd_{ref}$[paquetes]	$\frac{De_{max}}{\Delta t}$[Mb/s]
51	-50
68	-100
102	-176

Tabla 4.3: Valores máximos de $\frac{De_{max}}{\Delta t}$

Los valores de *data rate* estudiados en esta implementación son 12 Mb/s, 16 Mb/s y 24 Mb/s.

Ya que el diseño de control tiene como propósito ser *TCP-friendly* y equitativo, la ventana de referencia para cada servidor corresponderá a la mitad de la tasa de envío disponible de la línea. Teniendo en cuenta que 1 paquete pesa 1460 B y que se usó RTT=0.1 entonces:

$$\mathrm{e_i} = cwnd_{ref} - cwnd_{\mathrm{i}}\,, \tag{4.16}$$

$$cwnd_{ref} = \frac{data\_rate}{2 \cdot 1460 \cdot 8} RTT[paquetes]\,. \tag{4.17}$$

Los valores de la ventana de referencia $cwnd_{ref}$ en paquetes para la sintonización de cada PD de acuerdo a la ecuación 4.17 se muestran en la Tabla 4.2.

Podemos considerar el $error_{max}$ igual al valor de $cwnd_{ref}$ y utilizando la inecuación 4.13 obtener los rangos para K_p indicados en la Tabla 4.4, Para obtener los rangos de K_{d} y K_p es necesario obtener los valores de $(De_{max})/\Delta t$ Pues se sabe que los valores máximos de estas expresiones se obtienen de la transmisión que usa los valores máximos de AI, se corren simulaciones fijando AI=2.8 y así se consiguen los resultados en la Tabla 4.3.

4.4.4. Implementación Sintonización Controlador PD: PSO

Existen numerosos *tools* para optimización mediante PSO disponibles en la *web*, en este trabajo se utilizó como base un sencillo código de ejemplo de la página de Matlab que fue

Data Rate[Mb/s]	K_p	K_{d}
6	[0 0.0588]	[0 0.0600]
8	[0 0.0441]	[0 0.0300]
12	[0 0.0294]	[0.0170]

Tabla 4.4: Rangos de inicialización para K_p y K_{d}

adaptado a lo que se requería. No es posible usar directamente un algoritmo de PSO ya programado, pues se está trabajando en una plataforma distinta a *Matlab*, donde la evaluación de la función de *fitness* de cada partícula debe ser obtenida de tantas simulaciones en OPNET como partículas se hayan definido para cada iteración de PSO; la información debe ser guardada y accedida apropiadamente a fin de calcular partículas de la siguiente iteración y realizar las n simulaciones de OPNET de esta.
Para conseguirlo fue necesario:

- Unir *Matlab* y OPNET.
- Se crearon funciones en *Matlab* y OPNET.

Funciones OPNET

```
double *prePso(void)
```

Permite la inicialización o bien la lectura de coordenadas de las partículas para la presente iteración. Llama a la función *prePso()* de *Matlab*, tiene como salida los vectores *ind*, `vel` y *n_iter*, que corresponden a las coordenadas de los individuos, el vector de cambio de las partículas o individuos y el número de la iteración.

```
double* postPso(double peso_data, double *ind_dat, double *vel_dat, double
*valorF_data, double n_it_data)
```

Tiene como entradas *peso_data* que representa el factor de aleatoriedad de cambio, *ind_dat* los valores de las actuales partículas, *vel_dat* el vector de cambio de las partículas, *valorF_data*, el vector con los valores de la función de fitness para cada partícula, es decir, la sumatoria del error en valor absoluto respecto de la referencia de cada simulación (ver ecuación 4.14) y *n_it_data* el número de la simulación. Esta función llama a la función *postPso* de *Matlab* y entrega como salida f los dos valores mínimos de *fitness* para las partículas de la última iteración y p, las coordenadas de las partículas que corresponden a esos valores.

```
double *valF(int ni_t,double Val)
```

Abre un archivo llamado *Val.txt*, lee su contenido avanzando hasta la última línea, escribe en esta el valor de la variable *Val*. El contenido leído y el valor Val lo guarda en un vector de salida que retorna.

```
int *cuentaRep(void)
```

Lee y escribe en el archivo *prueba2.txt*. Escribe una línea por cada simulación donde anota dos valores el número de simulaciones y el número de iteraciones PSO. Cada vez que se completan tantas simulaciones como partículas se suma uno al valor de la iteración. Retorna el número de iteración de PSO y el número de simulación de partícula.

Funciones Matlab

`function [ind,vel,n_iter]=prePso()`
Inicializa las partículas o bien lee desde un archivo y entrega las partículas para la presente iteración. Lee los archivos *prueba.txt* y *prim_ it.txt*, si están vacíos significa que se trata de la primera iteración PSO y por lo tanto se inicializan aleatoriamente las partículas en un rango escogido, se entrega como salida *ind*, las coordenadas de las partículas, *vel*, el vector de cambio, que es cero, en la primera iteración, y `n_iter`, el número de la iteración, si se trata de la primera iteración estos serán guardados en *prim_ it.txt*. Si el archivo *prueba.txt* está vacío y *prim_ it.txt* no lo está quiere decir que se está en una de las $N-1$ simulaciones de la primera iteración se lee la línea y se guarda en *ind*, *vel* de salida es un vector con ceros y *n_ iter* es uno. Si *prueba.txt* no está vacío, se lee y se guarda su contenido en los vectores de salida.

`function [f_min,p_min]=PostPso(pesoStoc,ind,v,valF,n_iter)`
Esta función se ocupa de calcular las coordenadas de los individuos de la siguiente iteración. Tiene como entradas *pesoStoc* el factor de aleatoriedad al calcular el cambio de las partículas, *ind*, las partículas actuales, *v* el vector de cambio, *valF* el valor de *fitness* de cada partícula y *n_ iter* el número de la iteración. La función *PostPso* invoca a la función *pso4* de *Matlab* y retorna como salida *f_ min*, los dos mejores valores de *fitness*, y *p_ min* las dos partículas con mejor *fitness*. Se ejecuta el cambio de las coordenadas de partículas, estos valores son anotados en un archivo llamado *prueba.txt* que después leerá *PrePso* para que con esas coordenadas de las partículas sea evaluada la función de *fitness*, es decir, se ejecuten la simulaciones en OPNET.

`function [p_min, f_min,radius,ind,v]=pso4(pesoStoc,ind,v,valF)`
Tiene como entradas *pesoStoc*, el factor de aleatoriedad al cambiar las coordenadas de las partículas, *ind* las coordenadas actuales de las partículas, *v* el cambio de las partículas y *valF* el vector con el valor de *fitness* de cada partícula. Esta función calcula la mejor partícula, es decir la que tiene menor *fitness* y usando como referencia la posición de ella modifica la magnitud de su vector de cambio *v* con el que hará variar el valor de las coordenadas de las partículas considerando un factor aleatorio, también calcula *radius*, una variable que cuantifica la convergencia de las partículas.

Sintonización Controlador PD: Estructura del Programa

En cada simulación de OPNET en el nodo *init* del *process model tcp_ conn_ v3_ owl* se llama a la función *prePso()* con lo que se inicializan las coordenadas de las partículas si se está en la primera iteración y primera simulación. Cada iteración de PSO tiene N repeticiones o simulaciones, donde N es el número de partículas elegidas en el diseño. En cada simulación se ejecutará una transmisión de archivo usando como variables K_p, K_d las coordenadas de una de las partículas inicializadas. La función *cuentaRep()* cuenta y lleva registro del número

*D*e/e	Negativo	Cero	Positivo
Positivo	Ai=0.2 Md=0.3793	Ai=0.2 Md=0.875	Ai=1.6 Md=0.8
Cero	Ai=0.2 Md=0.465	Ai=0.2 Md 0.875	Ai=1.3 Md=0.8
Negativo	Ai=0.2 Md=0.575	Ai=0.2 Md=0.875	Ai=1 Md=0.8

Tabla 4.5: Set de reglas del controlador difuso

de la iteración PSO y del número de simulación de la iteración PSO , y entrega estos dos valores como salida, el número de simulación es utilizado para obtener las coordenadas de la partícula correspondiente a la simulación en curso desde el vector de salida de *prePso* de OPNET que retorna un vector con las coordenadas de todas las partículas de la iteración.

En el nodo *CLOSED* del *process model tcp_ conn_ v3* Figura 3.8 se guarda el valor de la función de *fitness* para esa partícula y simulación, esto se hace mediante la invocación de la función *ValF* de OPNET que guarda en el archivo *ValF.txt* el resultado de la función de *fitness* (ver ecuación 4.14) de cada simulación de la iteración en una línea y además retorna el vector con estos valores a OPNET. Cuando el número de repeticiones es igual al número de partículas definidas, entonces se llama a *postPso* en OPNET obteniéndose las partículas y *fitness* mínimos para tal iteración.

4.4.5. Diseño Controlador Difuso

Se diseñó un controlador PD difuso de tipo Takagi-Sugeno donde las entradas son, por lo tanto, el error $\mathrm{e} = ref(n) - y(n)$, la diferencia entre el valor de referencia y la salida que en este caso es el tamaño de la ventana de congestión *cwn*d, y $\Delta\mathrm{e} = \mathrm{e}(n) - \mathrm{e}(n-1)$, la diferencia del error.

Las salidas en este controlador son dos variables α y β, que corresponden a AI y MD respectivamente. En este caso, a diferencia del controlador PD, son variables independientes entre sí, y sus valores se escogen ya no conservando una relación que sea justa o *TCP-friendly*, si no que permita un comportamiento agresivo o amistoso en función de la magnitud que tenga la entrada, es decir, la ventana *cwn*d, respecto de la ventana de referencia según el ancho de banda disponible. Esto es posible gracias a que naturalmente un controlador difuso permite diferentes acciones mediante la activación de distintas reglas según los conjuntos a que pertenezcan las entradas.

El diseño del controlador considera tres conjuntos de pertenencia, Negativo, Cero y Positivo por cada variable de entrada lo que implica que existen nueve reglas las que se resumen en la Tabla 4.5 a continuación.

Las reglas se fijaron de modo que cuando el error es muy pequeño, o sea, cercano a cero, entonces independiente del valor que toma Δe, AI es pequeño para obtener bajas oscilaciones

y la magnitud de MD es aquella obtenida de la relación GAIMD en la ecuación 4.9.

Si el error es negativo entonces el valor de la ventana es mayor que la referencia por eso en cualquier caso el valor de AI es bajo e igual a 0.2 para evitar que el tamaño de la ventana crezca demasiado y MD se escoge en general amistoso, pero distinto según Δe; se está en una posición en que es necesario ceder porque se ha sobrepasado el valor de la referencia. Si Δe es negativo significa que la ventana de congestión está subiendo estando sobre la referencia y, por lo tanto, es necesario que en caso de pérdida baje en mayor proporción a que si Δ es positivo que indica que la ventana de congestión ya está decreciendo y, por lo tanto, se escoge MD mayor.

En la situación en que el error e es positivo el valor de ventana está bajo la referencia, por lo tanto, AI se elige más grande con el fin de acelerar el crecimiento, MD se fija pequeño de modo que el ascenso sea agresivo, pues se está bajo la tasa de envío potencial disponible.

Los valores de las columnas Negativo y Positivo fueron determinados de modo de equilibrar la agresividad y amigabilidad del protocolo. Por ejemplo, para $AI = 1$ el valor *friendly* que le corresponde es $MD = 0{,}5$, en este caso se escogió $MD = 0{,}8$ que es más agresivo con una diferencia de 0.3, por lo tanto, en la columna de la misma fila, se elige un valor de MD más amistoso, este debería ser 0.875 según la ecuación 4.9, pero se elige tanto más amistoso como agresivo se fijó en la otra columna $MD = 0{,}875 - 0{,}3 = 0{,}575$. Análogamente se determinan las otras filas de ambas columnas.

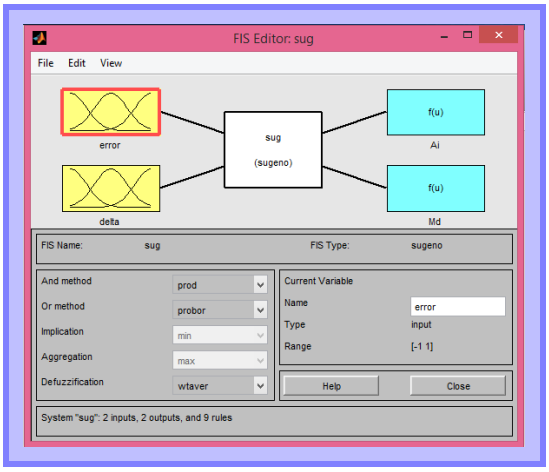
Las pruebas o experimentos de desempeño de los controladores difusos utilizaron dos bases de reglas diferentes a la de la sintonización. En un método AI y MD se consideran independientes y el otro dependientes según la relación GAIMD 4.9 y se comparó el resultado entre ambas. Esto se detalla en la sección 5.8 en Tabla 5.21 y Tabla 5.22.

4.4.6. Implementación Controlador Difuso

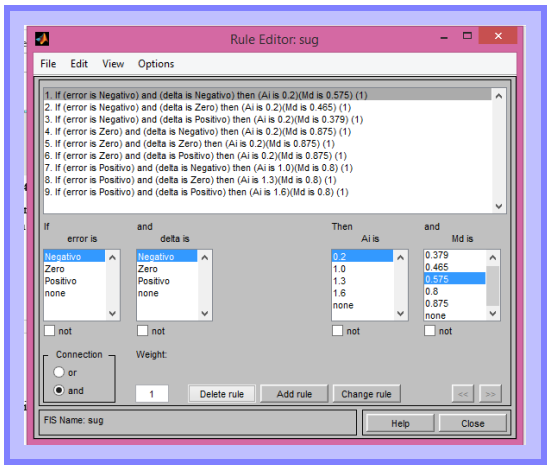
La implementación se realizó en Matlab, pero fue necesario realizar una optimización para encontrar los parámetros de los conjuntos difusos, esto se detalla en las sección 4.4.7. El controlador difuso fue implementado utilizando el *toolbox fuzzy* de *Matlab*.

Se definen dos entradas el error e y Δe y dos salidas AI y MD, se determina que el método *And* de las variables de entrada corresponde al producto y que la defuzzificación se calculará a través de los promedios ponderados de cada salida de las reglas, el rango de las entradas es entre [-1,1], lo que quiere decir que serán normalizadas multiplicando por un factor.

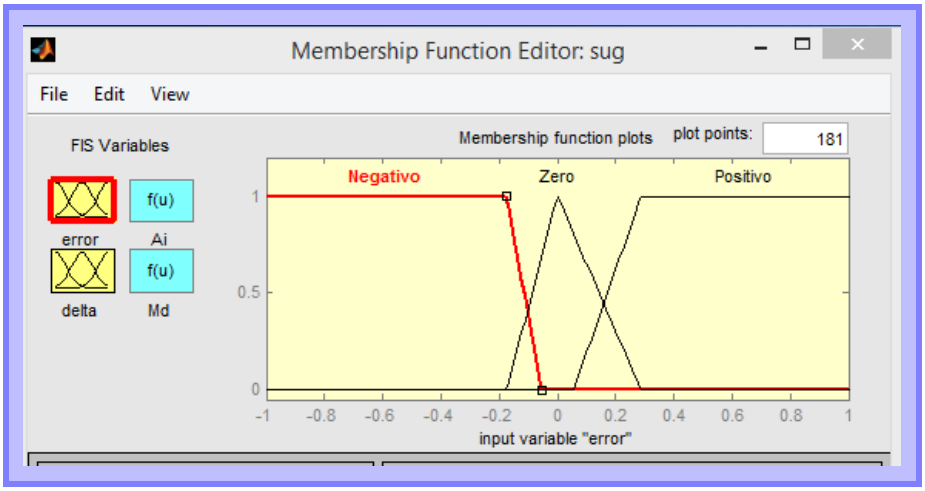
Las funciones de pertenencia o conjuntos difusos tanto para e como Δe son tres y corresponden a dos trapezoides y un triángulo para cada entrada, denominados positivo, cero y negativo como se ilustra en Figura 4.8c.



(a) Fuzzy Toolbox



(b) Reglas difusas



(c) Conjuntos de pertenencia

Figura 4.8: Controlador Difuso en Matlab

Las reglas fueron definidas como se describió en la sección 4.4.5 y como se muestra en la Figura 4.8b. El protocolo de congestión que utiliza control difuso es ejecutado en un escenario con cuatro servidores y cuatro *workstations* Figura 4.1.

Funciones Utilizadas y Estructura del Programa

Para invocar al bloque difuso diseñado con el toolbox se utiliza la función *function out = fuzzy_fun2(in)* de *Matlab* que tiene como entradas un vector de dos coordenadas que corresponden a e y Δe obteniéndose como salida AI y MD.

Para invocar esta función desde OPNET se utiliza la función *double *fuzzyp2 (double *input_data)* que tiene como entradas e y Δe, y llama a la función de *Matlab*, ya mencionada, entregándole estos parámetros, y recibiendo las variables de salida de esta que luego serán las suyas propias. Esta función que representa al controlador se utiliza en la función de TCP que

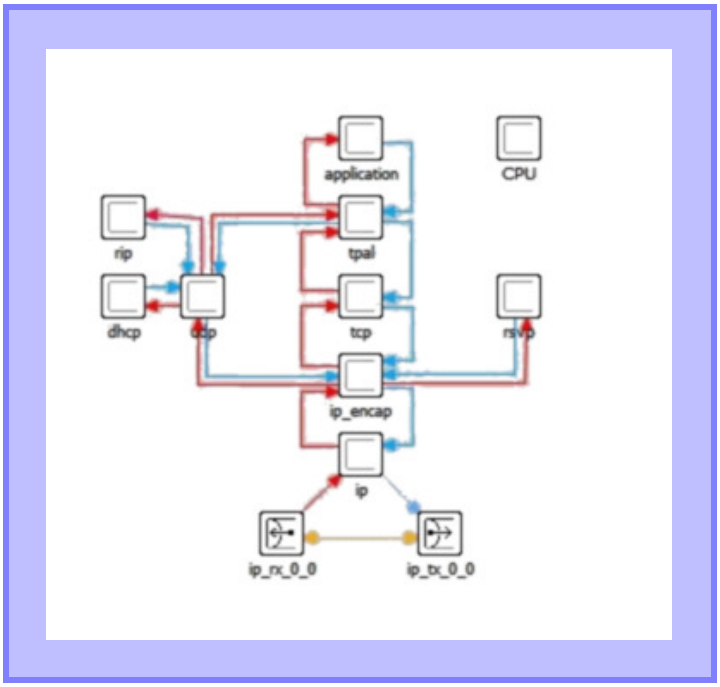


Figura 4.9: Node Model ppp_server_adv

actualiza el valor de la ventana de congestión conocida como *static void tcp_cwnd_update (TcpT_Size acked_bytes)* localizada en *tcp_conn_v3* Figura 4.7 que es el *process child* del *process model tcp_manager_v3* que a su vez pertenece al procesador tcp del *Node Model ppp_server_adv* Figura 4.9 y también del *ppp_wkstn_adv*.

4.4.7. Diseño Sintonización Controlador Difuso: Optimización PSO

La optimización para el controlador PD difuso consiste en la búsqueda de la mejor n-tupla de valores que representan a los parámetros de los conjuntos de pertenencia, dos trapecios y un triángulo, para cada variable: error e y delta error Δe. Se utilizan seis partículas, por lo que en cada iteración se ejecutan seis simulaciones, luego de las cuales se calculan 6 nuevas partículas y un óptimo parcial. El escenario OPNET escogido para la optimización consiste de cuatro servidores que transmiten desfasados en el tiempo entre sí de modo que el ancho de banda disponible para los servidores 1 y 2 cambia en el tiempo; al principio es 24 [Mb/s] y posteriormente es 12[Mb/s]. La evaluación de la función de *fitness* consiste en la sumatoria del error en valor absoluto respecto de la referencia, es decir, $(s_1 + s_2)/T$ donde s_1, s_2 y s_3 se describen en la expresión 4.18 y T es el periodo total, transcurrido.

$$f = \begin{cases} s_1 = \sum_{j=1}^{n} |\mathrm{e}_1(j)| & si \quad 155[s] < t < 230[s] \\ s_2 = \sum_{j=1}^{n} |\mathrm{e}_2(j)| & si \quad 230[s] < t < 330[s] \end{cases} \tag{4.18}$$

En la ecuación 4.18 $\mathrm{e}_1(j) = 102 - y(j)$, $\mathrm{e}_2(j) = 51 - y(j)$ y $\Delta t = t(j) - t(j-1)$. En otras palabras, la sintonización considerará el seguimiento de una referencia no constante. Posteriormente se probará su desempeño inclusive considerando una referencia diferente de las usadas en la sintonización.

4.4.8. Implementación Sintonización Controlador Difuso: Optimización PSO

La implementación PSO realizada es similar a la de la sintonización del controlador PD en el apartado 4.4.4. Consta de un conjunto de funciones en *Matlab* necesarias para ejecutar PSO y almacenar adecuadamente la información que se utilizará en las siguientes iteraciones, además se usan un conjunto de funciones en OPNET que invocan a las funciones de Matlab permitiendo la comunicación entre ambos programas y que las variables calculadas por PSO sean utilizadas como atributos de las simulaciones OPNET en las correspondientes iteraciones. Las funciones utilizadas tanto en Matlab como en OPNET son las mismas ya mencionadas en 4.4.4 por ello no se describirán detalladamente en esta sección, sin embargo, fue necesario modificar algunas funciones con el fin de aplicarlo al controlador difuso, esas diferencias se expondrán a continuación.

La función de *Matlab function [ind,vel,n_ iter]=prePso()* fue reemplazada por la función *function [ind,vel,n_ iter]=prePsoFuzzy(nrep)*, que tiene las mismas salidas que la anterior, pero como entrada recibe el número de la simulación *nrep* de la iteración, el total de simulaciones en una iteración es igual al número de partículas. Conocer *nrep* es necesario pues, esta función lo utilizará para acceder a la información y construir el controlador. La función *prePsoFuzzy()* tal como *prePso()* obtiene los datos guardados en la anterior iteración que consiste en las coordenadas de las N partículas calculadas y los N vectores de cambio de coordenadas *vel* y lo almacena en un vector de salida, se seleccionan de este vector las coordenadas de la n-ésima partícula, para definir las funciones de pertenencia del controlador de la simulación en curso mediante el uso de *nrep*.

Además, al igual *prePso()* la función cumple con la tarea de la inicialización si se está en la primera iteración, pero como la inicialización de la matriz de coordenadas de las N partículas no es completamente aleatoria y requiere cumplir ciertos requisitos esto se realiza invocando a la función function *[ind]=inicPart(npart)*, que tiene como entrada el número de partículas y genera como salida una matriz con tantas filas como partículas y 14 columnas que corresponden a los 14 puntos que determinan las funciones de pertenencia de cada variable como muestra Figura 4.10, donde el orden i-ésimo ubicado en la parte inferior son de las funciones de pertenencia de la variable e y el orden i-ésimo ubicado en la parte inferior de la figura, a la segunda variable Δe.

Algunas de las relaciones que se consideraron para la correcta implementación del módulo son:

1. i_1 e i_2 son aleatorios y pertenecen al intervalo [-0.4,-0.15] y [0.15, 0.4] respectivamente para la variable e; i_3 e i_4 pertenecen al intervalo [-0.2,-0.1] y [0.1, 0.2] respectivamente para la variable Δe. Fueron elegido así los intervalos, resultado de la observación de la ejecución del controlador PD bajo las mismas características de la simulación que será empleada para la optimización PSO del controlador difuso. Esta consiste en la transmisión de cuatro servidores con cierto desfase en el tiempo, como se mencionó en la sección 4.4.6, de la observación de una simulación que usa el controlador PD se

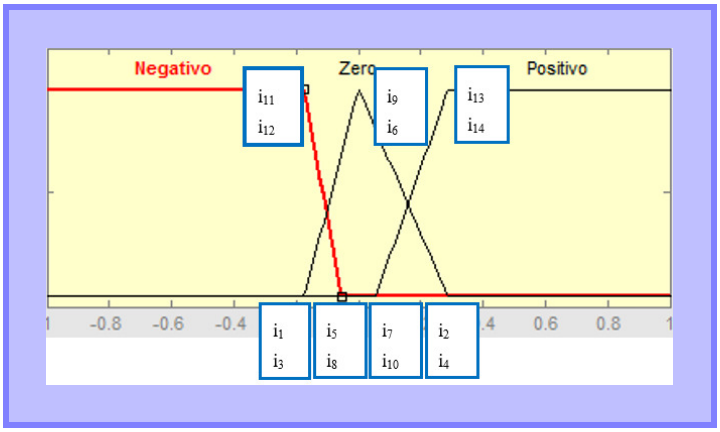


Figura 4.10: Relación entre conjuntos de pertenencia y partículas

	Positivo		Negativo	
	Min	Max	Min	Max
Error e	0.0082	0.8039	-0.956	-0.052
Delta error Δe	-	-	-0.19	-0.013

Tabla 4.6: Intervalos observados para e y Δe

pueden obtener las magnitudes máximas y mínimas de los positivos y negativo de las variables de entrada en lazo cerrado.

2. Para la variable Δe solo se observaron negativos, no fueron explorados todos los valores, sino una muestra, y se asumió que estos serían de la misma magnitud. Se escogieron los intervalos mencionados en 1 de modo que pertenecieran a los de la Tabla 4.6.

3. i_6 e i_9 se definen como cero para que los triángulos de ambas variable se encuentren centrados.

4. i_7 se define aleatorio y perteneciente [0.04 ,0.1] y $i_7 = -i_5$.

5. i_{10} se define aleatorio y perteneciente [0 ,0.1] y $i_{10} = -i_8$.

6. $i_{11} = i_1$ y análogamente para las coordenadas de Δe

7. $i_2 = i_{13}$ y análogamente para la coordenadas de Δe

Es importante mencionar que si bien la inicialización se realiza en determinado intervalo, el algoritmo PSO permite encontrar el óptimo incluso en una vecindad de esta región.

En esta función finalmente se crea módulo *FIS* de *Matlab* invocando a la función *function crearFis2(v1,v2)* que recibe en v_1 y v_2 los valores de la n-tupla ind de una determinada partícula con las coordenadas de las funciones de pertenencia inicializadas o leídas por *preP-soFuzzy(nrep)* asociadas a e y Δe respectivamente.

Otra importante modificación necesaria para poder ejecutar PSO es la de la función de *Matlab* es el reemplazo de *function [p_ min, f_ min,radius,ind,v] = pso4(pesoStoc,ind,v,valF)* por *function [p_ min, f_ min,radius,ind,v] = pso4fuzzy(pesoStoc,ind,v,valF)*. Ambas funciones tienen por misión calcular las nuevas partículas i*n*d y vectores de cambio v de la siguiente iteración, sin embargo, en esta optimización no todas las n-tuplas de una partícula son evaluables, es necesario que cumplan con ciertas condiciones para crear un módulo *FIS* que no produzca errores. Es por eso que *pso4fuzzy(pesoStoc,ind,v,valF)* se invoca a la función function *[bool,m]=condicion_ lin2(m,leader)* que verifica:

1. Que todas las coordenadas estén entre [-1,1].

2. Que se cumplan las relaciones y hacer la permutación en caso que no se cumplan:

   - $i_2 > i_1$
   - $i_4 > i_3$
   - $i_5 > i_{11}$
   - $i_8 > i_{12}$
   - $i_{13} > 1_7$
   - $i_{14} > i_{10}$
   - $i_7 > i_5$
   - $i_{10} > i_8$

3. Que $i_1 < i_6 <_i 2$ y $i_3 < i_7 < i_4$ y que caso que no se cumpla i_6 pasa a tomar el mismo valor del extremo que está más cerca, análogamente con i_7.

4.4.9. Resumen del Capítulo

En este capítulo se presentaron por un lado, el diseño y la implementación del mecanismo de predicción que determina el grado de congestión de la red y la referencia para el mecanismo de control. Y también se expusieron el diseño e implementación de un controlador PD y un controlador difuso.

La predicción del ancho de banda disponible se diseña a partir de un modelo de TCP que permite calcular el *throughput* [4] mediante las probabilidad de las pérdidas y el RTT. Contando los paquetes perdidos para cuantificar probabilidad de las pérdidas y haciendo un promedio en el tiempo del *throghput* se estima el ancho de banda disponible en el momento.

El diseño del controlador PD considera como variable manipulada el valor AI de *Congestion Avoidance*. Además AI y MD se relacionan mediante una expresión que asegura *fairness* y que proviene de la técnica conocida como GAIMD. Las constantes del controlador PD fueron ajustadas mediante optimización *Particle Swarm Optimizaction* PSO. PSO es un método de optimización que se basa en el el éxito de las poblaciones para encontrar valores óptimos. La implementación se hizo utilizando simulaciones en OPNET en conjunto con ejecuciones en Matlab.

El diseño del controlador difuso es de tipo Takagi-Sugeno y recibe como entradas el error

respecto de la referencia y el delta error. Los conjuntos difusos fueron ajustados usando PSO. La variable manipulada es AI y MD. Para este controlador la optimización PSO se realizó mediante simulaciones en OPNET en conjunto con Matlab donde se utilizó *Simulink* para implementar el controlador difuso.

5. Resultados

5.1. Introducción

En este apartado se presentan los resultados de las simulaciones realizadas de los dos tipos de protocolos usados diseñados para el control de congestión. El primer protocolo emplea un método de ajuste al *throughput* que corresponde a un controlador PD y el segundo, un controlador difuso.

Del primer protocolo se exponen los valores resultantes de la sintonización, o sea, los parámetros adecuados del controlador, ajuste que se ha realizado en un escenario con 2 servidores y dos *workstations*, además se muestran en la parte 5.3 las simulaciones del método con control clásico empleando como estimación de la tasa de transmisión 4.7 el promedio ponderado de todos los valores resultantes de las predicciones de ventana en el tiempo de 3 *throughputs* de línea diferentes 12 Mb/s , 16 Mb/s y 24 Mb/s lo que además fue comparado con el desempeño de TCP Reno bajo las mismas condiciones en un escenario del mismo tipo de la sintonización, con un par de servidores y *workstations*. En la sección 5.4 se exponen los resultados de una serie de 20 simulaciones del protocolo testeado en el mismo escenario anterior usando diferentes semillas lo que determinará un comienzo aleatorio de los servidores entre 0.001 s y 0.01 s, el objetivo de estas pruebas es analizar las medias de los promedios de *throughput* y utilización y la desviación estándar. En la sección 5.5 se estudia el comportamiento del protocolo con diferentes tamaños de archivo a enviar y se compara con el funcionamiento de TCP Reno, el escenario es el de los casos anteriores. Por otro lado, en la sección 5.6 en un escenario con cuatro servidores y cuatro *workstations* y empleando la estimación de *data rate* de acuerdo a 4.8 se realizaron las pruebas de *TCP-friendliness* del protocolo en cuestión; los servidores transmiten en algunos casos al mismo tiempo y en otros con un desfase en el tiempo por medio de la configuración en el bloque *profile*; el fin de las pruebas es evaluar el ancho de banda del protocolo al compartir equitativamente los recursos del canal con servidores que transmiten mediante el protocolo TCP Reno. Finalmente, en la sección 5.7 se muestran los resultados de los mismos experimentos presentados en el apartado 5.6 pero con modificaciones en el diseño del algoritmo implementado.

Los resultados del segundo protocolo, el cual utiliza un controlador difuso se presentan en la sección 5.8. Las pruebas se realizaron en un escenario con cuatro servidores y la estimación de la tasa de envío diponible como en la ecuación 4.8. Se presentan los resultados de los parámetros de la optimización PSO y de la ejecución del protocolo con dos variantes de reglas.

Data rate [Mb/s]	Ventana Referencia [pkt]	Número de iteraciones PSO	K_p	K_d
12	51	18	56.6590e-003	-2.1974e-003
16	68	10	42.12e-003	-3.7465e-003
24	102	8	28.1850e-003	-1.0433e-003

Tabla 5.1: Constantes controlador PD para *data rate* 12 Mb/s, 16 Mb/s y 24 Mb/s

	Ventana Servidor 1[B]	Ventana Servidor 2[B]	Throughput[b/s]	Utilización[%]
Promedio	7.47e4	7.46e4	1.15e7	96.1
Desviacion Estándar	3.96e3	3.99e3	9.98e6	8.32

Tabla 5.2: Promedio poderado y desviación estándar *data rate* 12 Mb/s

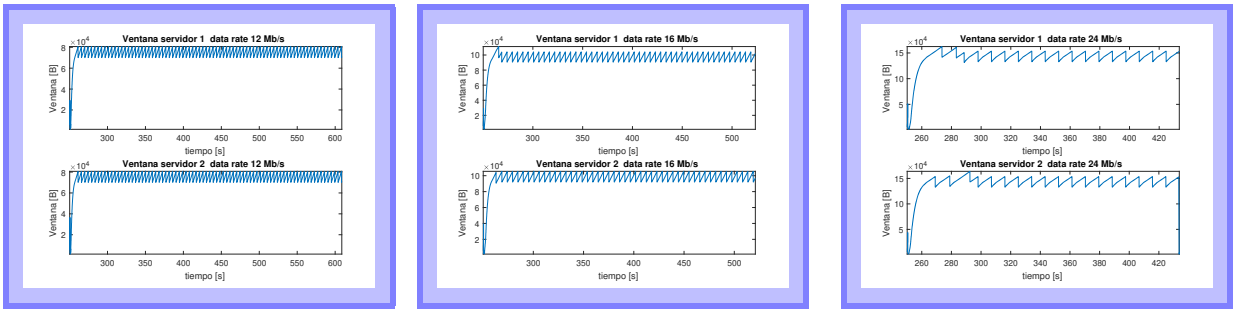
5.2. Sintonización PD

Como se indica en 4.4.2 fueron sintonizados 3 controladores PD para *data rates* de línea 12 Mb/s, 16 Mb/s y 24 Mb/s.
En la Tabla 5.1 se muestran los valores de las constantes K_p y K_d calculadas para cada *data rate*. Los controladores fueron sintonizados usando una referencia fija del tamaño de ventana en paquetes correspondiente al valor máximo posible de acuerdo al *data rate* de la línea, al número de servidores transmitiendo y a una repartición equitativa. Los gráficos siguientes se presentan los resultados de las simulaciones de los valores óptimos encontrados para cada controlador usando la misma referencia utilizada en la simulación de optimización y la misma semilla. Se puede observar de aquí cuán cercano es el ajuste respecto del objetivo.

Se puede notar de la Figura 5.1a que la magnitud de las ventanas obtenidas en promedio es cercana a la referencia donde el promedio de las ventanas de cada servidor ponderado en el tiempo es de 50.958 pkt y 50.953 pkt respecto de 51 pkt que es el valor en relación al que se calcula el error. Además los valores de *throughput* y utilización son próximos a los valores máximos posibles, donde la utilización alcanza el 96 %.

	Ventana Servidor 1[B]	Ventana Servidor 2[B]	Throughput[b/s]	Utilización[%]
Promedio	9.62e4	9.71e4	1.51e7	94.5
Desviacion Estándar	5.63e3	5.69e3	1.87e6	11.7

Tabla 5.3: Promedio ponderado y desviación estándar data rate 16 Mb/s



(a) 12 Mb/s (b) 16 Mb/s (c) 24 Mb/s

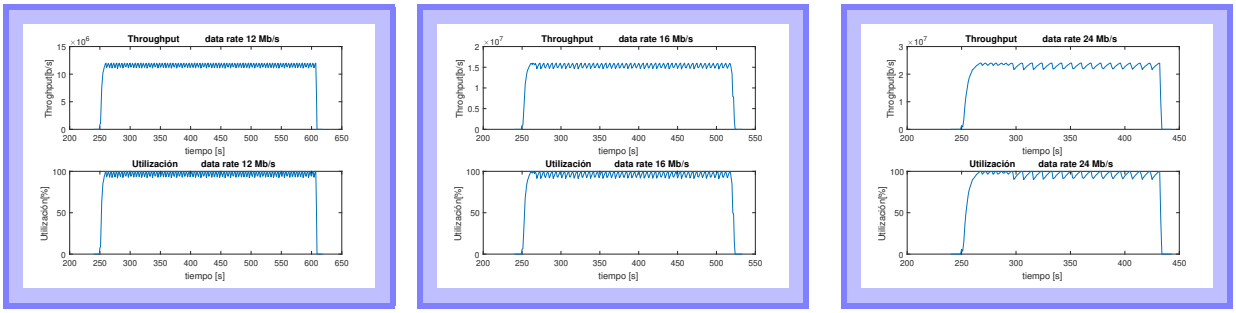
Figura 5.1: Sintonización PD

	Ventana Servidor 1[B]	Ventana Servidor 2[B]	Throughput[b/s]	Utilización[%]
Promedio	1.41e5	1.41e5	2.26e7	94.0
Desviacion Estándar	1.06e4	1.04e4	3.34e6	13.9

Tabla 5.4: Promedio poderado y desviación estándar *data rate* 24 Mb/s

Para el caso en que al *data rate* se le asigna magnitud 16 Mb/s el controlador óptimo obtenido da como resultado ventanas ponderadas en el tiempo 66.453 pkt y 65.558 pkt respecto de 68 pkt que es tamaño de ventana que se usó por referencia, de donde se ve que el ajuste es bastante cercano, pero no mejor que el caso 12 Mb/s. Los valores de *throughput* y utilización se aproximan a los máximos posibles, llegando la última al 94 % que es inferior al obtenido en la experiencia para *data rate* 12 Mb/s . En cuanto a la desviación estándar se puede ver que alcanza magnitudes superiores a las que toma cuando el *data rate* es 12 Mb/s.

Las simulaciones del último caso donde el *data rate* de línea es 24 Mb/s también muestran un buen ajuste. Las ventanas de los servidores tienen valores 96.9 pkt y 97.1 pkt respecto de los 102 pkt que se usaron como referencia. La utilización llega a un 94 % y está próxima al máximo al igual que el *throughput* alcanzado, además la magnitud de las desviaciones respecto del promedio es mayor a ambos casos anteriores. Esto se debe en el caso de las gráficas de ventanas, a que el comportamiento inicial de ellas corresponde a un intervalo creciente, mayor en tiempo a los de *data rates* anteriores, que se estabiliza en un valor final cercano a 24 Mb/s. Lo mismo ocurre en los casos de *throughput* y utilización, pues los datos consideran también una zona de aumento de *throughput* desde cero y al ser más alto el valor final entonces la dispersión calculada será mayor.



(a) 12 Mb/s (b) 16 Mb/s (c) 24 Mb/s

Figura 5.2: Sintonización PD Throughput

5.3. Comparación TCP Reno v/s Protocolo con Control PD

5.3.1. Introducción

En esta sección se hace una comparación entre el desempeño del tradicional protocolo TCP Reno respecto del protocolo diseñado en a base de controladores PD. A continuación, se exponen los resultados de seis distintas simulaciones, tres corresponden al protocolo con control PD y están asociadas a los tres niveles de *data rates* (12 Mb/s, 16 Mb/s y 24 Mb/s) para los que se ajustaron controladores. Las otras tres utilizan el protocolo TCP Reno para los mismo *data rates*.

El tamaño de archivo de las transmisiones es de 400 MB y se ejecutaron con una semilla diferente de la que se usó para sintonización, por lo que estas pruebas dan una idea de cuán buena es la sintonización para algún caso general en que los servidores comienzan su transmisión en un momento aleatorio entre 0.001 s y 0.01 s. El valor de la referencia en estas no es fija como en la sintonización, sino que se implementan ambas partes del protocolo, es decir, se usa la predicción de la ventana de congestión por referencia; más específicamente se utiliza la que calcula el promedio de todos los valores ocurridos en el intervalo tiempo (ecuación 4.7). Se exponen los gráficos de ventana de ambos servidores, *throughput*, y utilización alternadamente de cada protocolo.

5.3.2. Comparación Protocolo Control PD v/s TCP Reno *data rate* 12 Mb/s

Se puede observar en Figura 5.3a que el valor promedio de la ventana alcanzada es superior para el protocolo que utiliza control PD, que llega a valores cercanos a 74e+03 B, en cambio, TCP Reno toma magnitudes aproximadas a 61e+03 como se puede apreciar en más detalle en la Tabla 5.5. Además, las oscilaciones de las ventanas en TCP Reno tienen mayor amplitud lo que coincide con valores de deviación estándar más altos. Por otro lado,

	Ventana Servidor 1[B]	Ventana Servidor 2[B]	Throughput [b/s]	Utilización[%]
Promedio Control PD	7.46e4	7.50e4	1.15e7	95.7
Desviacion Estándar	7.24e3	7.24e3	1.53e6	12.8
Promedio TCP Reno	6.17e4	6.12e4	9.43e6	78.6
Desviacion Estándar	1.30e4	1.30e4	1.49e6	12.5

Tabla 5.5: Promedio y desviación estándar para *data rate* 12 Mb/s en control PD y TCP Reno

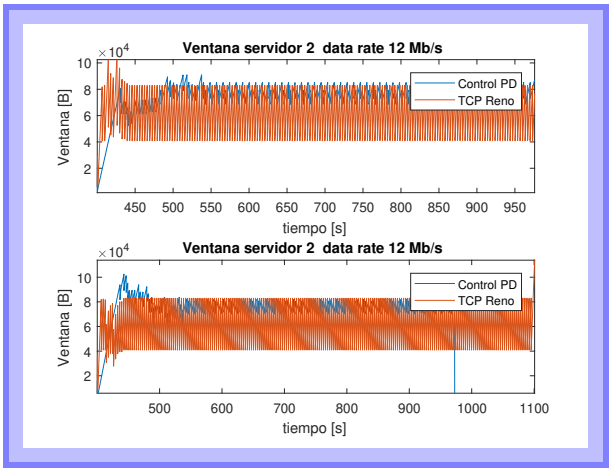
	Ventana Servidor 1[B]	Ventana Servidor 2[B]	Throughput[b/s]	Utilización[%]
Promedio Control PD	9.40e4	9.52e4	1.48e7	92.4
Desviacion Estándar	9.79e3	1.06e4	2.59e6	16.2
Promedio TCP Reno	7.86e4	7.89e4	1.23e7	77.2
Desviacion Estándar	1.69e4	1.68e4	2.10e6	13.1

Tabla 5.6: Promedio y desviación estándar para *data rate* 16 Mb/s en control PD y TCP Reno

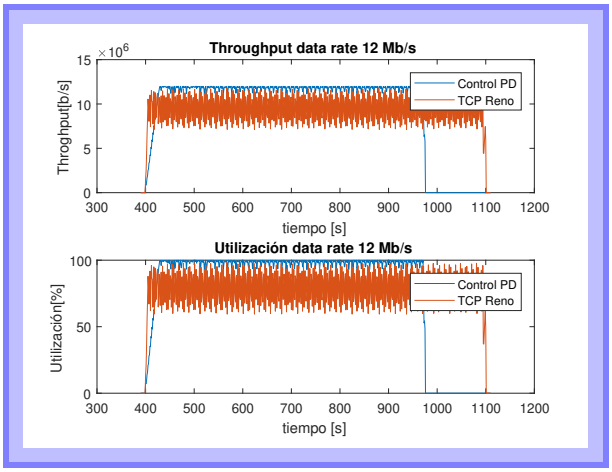
el *throughput* y la utilización Figura 5.3b del protocolo implementado resultan mayores que los de TCP Reno , la utilización alcanza casi un 96 % en cambio en TCP Reno sólo llega a 78 %, los niveles de dispersión son semejantes tomado valores del mismo orden Tabla 5.5. Consecuentemente con lo expuesto, se ve que el tiempo de transmisión del archivo de 400 MB es superior en la prueba que usa el protocolo clásico, demorando alrededor de 130 s más que lo que tarda el protocolo que utiliza control PD. En la Figura 5.3c se aprecia como la ventana sigue a la referencia que se obtiene de la predicción a través de la ecuación 4.7. Finalmente, en la Figura 5.3d se aprecian los valores MD y AI que tienen una región transiente entre 400 s y 570 s donde fluctúa entre 0.2 y 0.9 y luego se estabilizan en 0.2 aproximadamente.

5.3.3. Comparación Protocolo Control PD v/s TCP Reno *data rate* 16 Mb/s

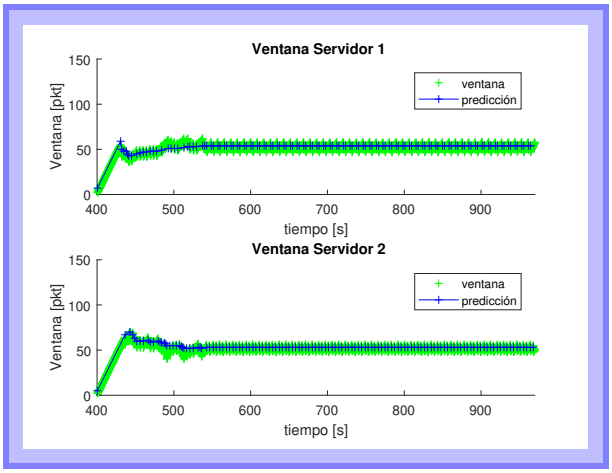
Se puede ver de Figura 5.4a que el promedio de ventana resultante es mayor al utilizar el protocolo implementado que al utilizar TCP Reno. Esta magnitud se aproxima a los 94e+03 B en el primer caso contra los 78e+03 B a que llega en el segundo (Tabla 5.6). La



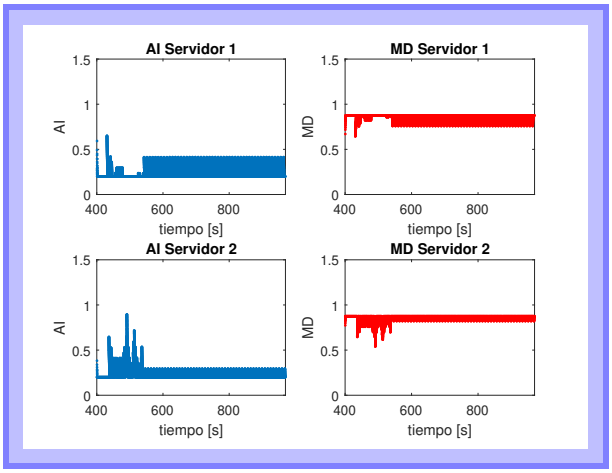
(a) Ventanas:Control PD v/s TCP Reno



(b) Throughput: Control PD v/s TCP Reno

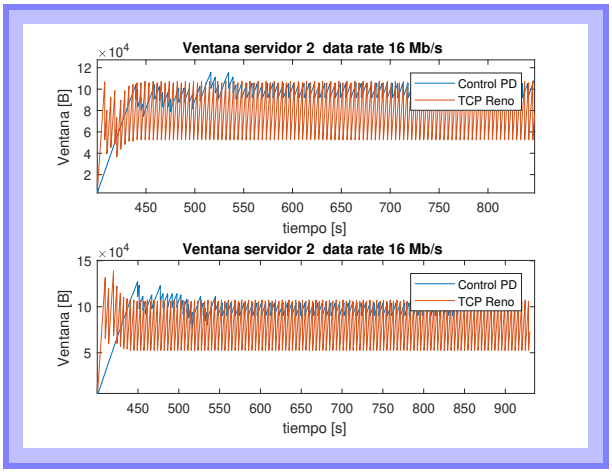


(c) Predicción Control PD

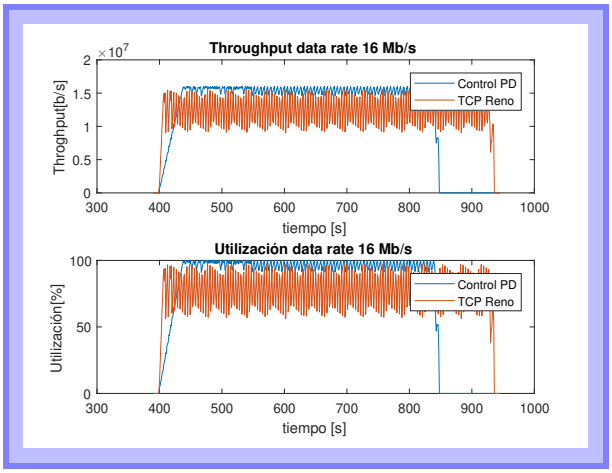


(d) AI y MD Control PD

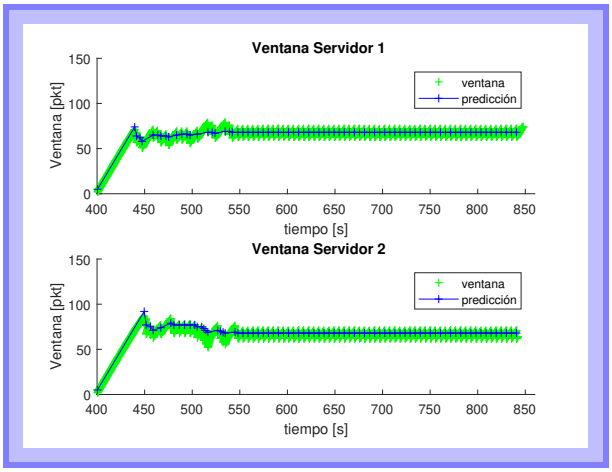
Figura 5.3: Resultados Controlador PD Escenario 12 Mb/s



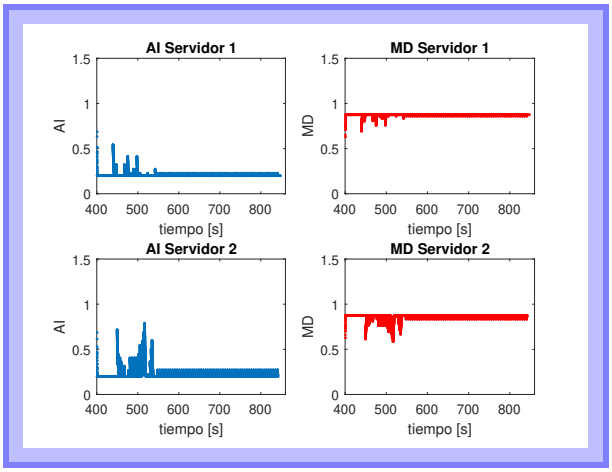
(a) Ventanas: Control PD v/s TCP Reno



(b) Throughput: Control PD v/s TCP Reno



(c) Predicción Control PD



(d) AI y MD Control PD

Figura 5.4: Resultados Controlador PD Escenario 16 Mb/s

desviación estándar de las ventanas es mayor para las simulaciones que usan TCP Reno, lo que indica mayor dispersión, tal como se observa en las gráficas. El *throughput* y utilización son más altos para el protocolo con control PD que toma valores de utilización de 92 % versus 77 % en el caso clásico. La desviación estándar para el *throughput* y la utilización tiene magnitudes similares, siendo levemente superior para el protocolo con control PD. Acorde a lo ya mencionado es posible ver que el tiempo de transmisión del archivo de 400 MB es superior en la simulación con el protocolo clásico demorando alrededor de 100 s más que el otro. En la Figura 5.4c se aprecia cómo la ventana sigue a la referencia que se obtiene de la predicción a través de la ecuación 4.7. Además en la Figura 5.4d se pueden observar los valores de AI y MD, se distingue una región transiente entre 400 s y 500 s donde AI fluctúa entre 0.2 y 0.8 aproximadamente y posteriormente al mantenerse las condiciones se estabiliza en 0.2.

	Ventana Servidor 1[B]	Ventana Servidor 2[B]	Throughput[b/s]	Utilización[%]
Promedio Control PD	1.32e5	1.32e5	2.12e7	88.2
Desviacion Estándar	1.87e4	1.87e4	5.25e6	21.9
Promedio TCP Reno	1.14e5	1.14e5	1.83e7	76.4
Desviacion Estándar	2.34e4	2.34e4	3.54e6	14.8

Tabla 5.7: Promedio y desviación estándar para *data rate* 24 Mb/s en control PD y TCP Reno

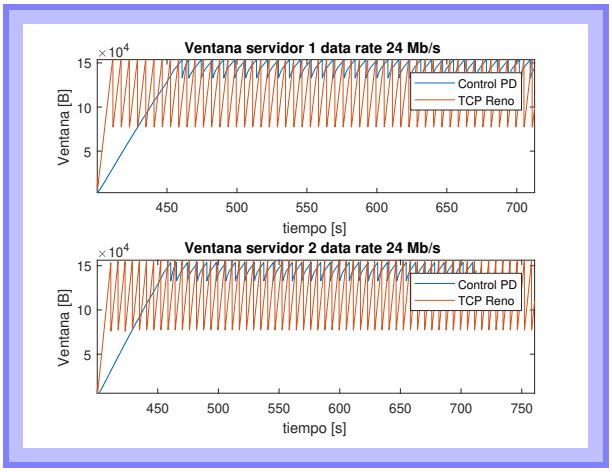
5.3.4. Comparación Protocolo Control PD v/s TCP Reno *data rate* 24 Mb/s

Se observa de la Figura 5.5a que el promedio de ventana a que llega el protocolo que utiliza control PD es superior al otro protocolo, esto se acerca a 1.32e+05 contra 1.14e+05 (Tabla 5.7) donde la dispersión respecto de las ventanas es superior para la prueba que usa TCP Reno. El *throughput* y la utilización que se muestran en Figura 5.5b tienen mejores resultados para el protocolo con control PD llegando a un 88 % de utilización contra un 76 % que alcanza el protocolo clásico. La dispersión de los datos es mayor para el protocolo con control PD. Coherentemente con lo ya mencionado el tiempo de transmisión del archivo es mayor para la simulación que utiliza el protocolo clásico, que demora aproximadamente 50 s más que el protocolo con control PD. En la Figura 5.5c se puede ver que las ventanas se encuentran muy próximas a la predicción: además en la Figura 5.5d se ilustran los valores de AI y MD en cada servidor, de donde se aprecia que el primero fluctúa entre 0.2 y 0.8 aproximadamente y se mantiene estable a partir de cierto valor acorde con lo que ocurre con las ventanas.

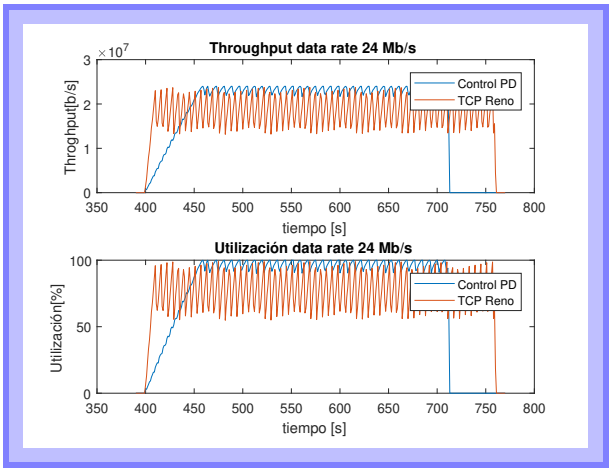
Finalmente, se desprende de las pruebas realizadas que para todos los casos el desempeño del protocolo con controlador PD es mejor que el del protocolo clásico permitiendo una ventaja en tiempo de transmisión de un archivo de 400 MB que varía entre los 50 s y 130 s y una utilización de canal entre 88 % y 95 % dependiendo del ancho de banda de canal disponible para los dos servidores transmitiendo.

También es importante notar que la desviación estándar de las ventanas es menor en el caso del protocolo con control PD para cualquiera de los *data rates* estudiados, es decir, se confirma que con el protocolo implementado las oscilaciones de ventana en el tiempo son menores lo que se asocia con un mayor *throughput* obtenido.

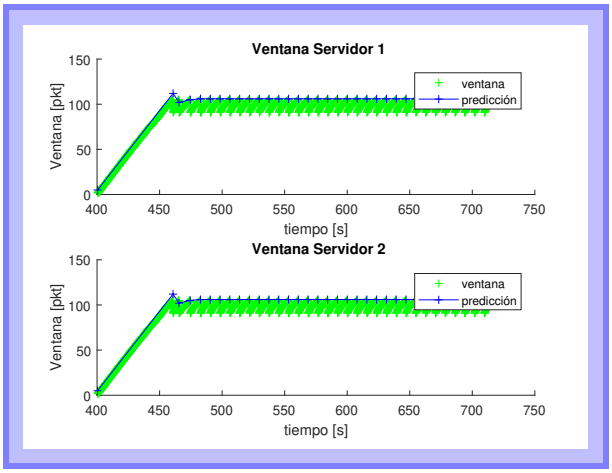
Además, es posible observar de los gráficos de las ventanas resultantes de las simulaciones



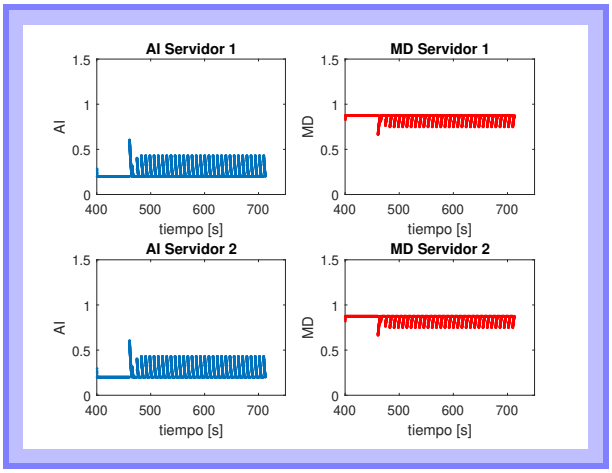
(a) Ventana: Control PD v/s TCP Reno



(b) Throughput: Control PD v/s TCP Reno



(c) Predicción Control PD



(d) AI y MD Control PD

Figura 5.5: Resultados Controlador PD Escenario 24 Mb/s

Data Rate	Protocolo	Ventana Servidor 1[B]	Ventana Servidor 2[B]	Throughput [b/s]	Utilización [%]	Mejora
12[Mb/s]	Control PD	7.46e4	7.50e4	1.15e7	95.7	17.1 %
	TCP Reno	6.17e4	6.12e4	9.43e6	78.6	
16[Mb/s]	Control PD	9.40e4	9.52e4	1.48e7	92.4	15.2 %
	TCP Reno	7.86e4	7.89e4	1.23e7	77.2	
24[Mb/s]	Control PD	1.32e5	1.32e5	2.11e7	88.2	11.8 %
	TCP Reno	1.14e5	1.14e5	1.83e7	76.4	

Tabla 5.8: Resumen

que usan el protocolo con control PD un crecimiento en forma de rampa antes de oscilar en torno al valor en que finalmente se estabiliza, este periodo transiente es más largo cuando el ancho de banda disponible de la línea es mayor y va aproximadamente entre los 30 s y 70 s, alcanzando el máximo en el caso de *data rate* 24 Mb/s. Esto afecta el *throughput* promedio, pues en el intervalo de crecimiento de la ventana el *throughput* disponible sube de la misma forma que ésta tardando en llegar a su máximo. Si se comparan los gráficos de la sección 5.3 con los gráficos del apartado 5.2 se ve que el tiempo de estabilización es mucho menor en la sintonización, de allí se puede concluir que la razón del transiente se asocia a la naturaleza de la referencia, que a diferencia del proceso de sintonización donde se establece un valor fijo igual al *throughput* de la línea, en estas pruebas corresponde a la predicción que depende de los paquetes perdidos y que varía y crece hasta estabilizarse finalmente en una cantidad cercana al ancho de banda disponible para el servidor.

El tiempo de estabilización del protocolo con control PD depende del comportamiento de la predicción de ancho de banda y de la magnitud real del ancho de banda de la línea, de lo que se desprende que si el archivo es más largo y la transmisión dura más tiempo, el desempeño es mejor porque menos ponderación tiene ese tiempo inicial en el total, esto se analizará en más detalle en la sección 5.5.

5.3.5. Resumen

Se puede observar que en cada uno de los casos analizados, el protocolo propuesto muestra una mejora respecto de TCP Reno que supera en cada caso el 10 %. Esta ventaja en el rendimiento disminuye mientras crece el ancho de banda disponible canal de transmisión. Estos resultados se resumen en la tabla 5.8

Data Rate[Mb/s]	Promedio throughput [b/s]	Desviación estandar throughput[b/s]	Promedio Utilización [%]	Desviación estándar Utilización [%]
12	1.14e7	6.50e4	95.1	0.54
16	1.47e7	1.47e5	91.9	0.92
24	2.07e7	4.68e5	86.2	2.0

Tabla 5.9: Promedio y desviación estándar de throughput y utilización de canal para 20 simulaciones con distintas semillas

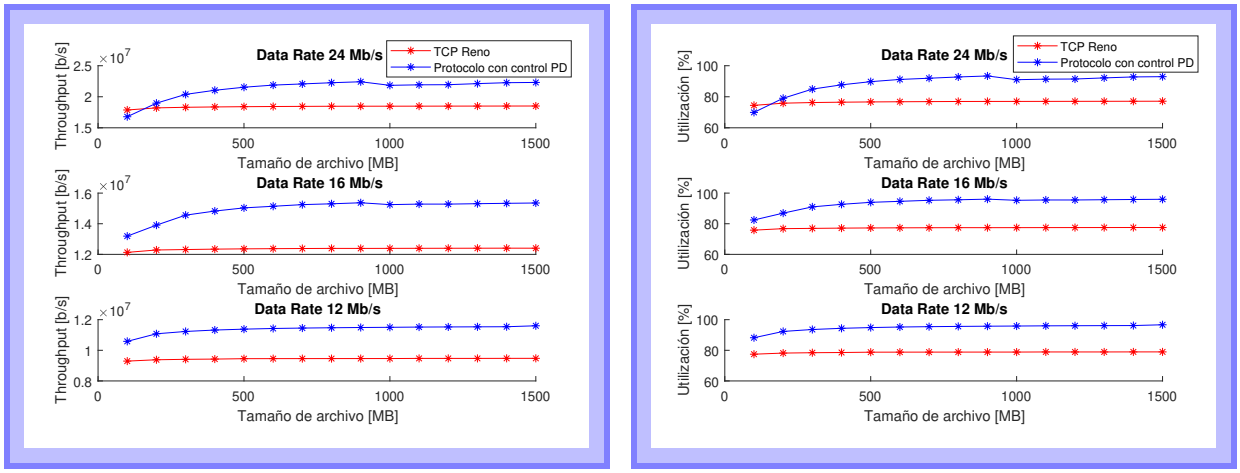
5.4. Resultados de Simulaciones con Protocolo con Control PD Usando Diferentes Semillas

Para analizar si el resultado de *throughput* obtenido depende de la semilla con la que se generó la simulación, o bien se trata de un resultado más general, se hicieron 20 simulaciones para cada nivel de *data rate* que corresponden a la transmisión de un archivo de 400 MB usando el protocolo con control PD. Cada semilla determinará el tiempo aleatorio de inicio de la simulación en cada servidor entre 0.001 s y 0.01 s. Se calculó el promedio de las medias de *throughput* y utilización obtenidas para cada una de las simulaciones y además la desviación estándar. Los resultados se resumen en la Tabla 5.9. De la Tabla 5.9 se puede concluir que la dispersión de los datos es baja en magnitud en relación con la magnitud del promedio de las medias de *throughput* o utilización. Además, al comparar con las medias de *throughput* obtenidos por el protocolo con TCP Reno en 5.3 es posible notar que las medias del *throughput* del protocolo con control PD son superiores que los valores conocidos de promedios del *throughput* de la simulaciones con TCP Reno.

5.5. Resultados Protocolo con Control PD v/s TCP Reno para Distintos Tamaños de Archivo

En esta sección se analiza el desempeño de ambos protocolos en la transmisión utilizando un valor de semilla de generación fija y para distintos tamaños de archivos que van desde los 100 MB hasta los 1500 MB. Los resultados se presentan en Figura 5.6.

Se puede notar de la Figura 5.6 que el *throughput* promedio y utilización promedio obtenidos mediante el protocolo con control PD no son constantes respecto del tamaño de archivo, a diferencia de lo que se obtiene con el protocolo TCP Reno donde sí lo son, manteniéndose estos últimos bajo el 80 % del ancho de banda disponible en cada *data rate* estudiado. En el caso de los *data rates* de línea 12 Mb/s y 16 Mb/s el *throughput* promedio y la utilización promedio del protocolo con control PD son mayores para cada tamaño de archivo considerado que los obtenidos con el protocolo TCP Reno. En cambio cuando el *data rate* es 24 Mb/s, el



(a) Throughput (b) Utilización

Figura 5.6: Resultados frente a Diferentes Tamaños de Archivo

throughput promedio y la utilización promedio del protocolo con control PD son mayores a partir de los 200 MB. La curva de utilización en cada gráfico tiende a una asíntota cercana, pero menor al máximo 100 %, además ésta tiene una apariencia más plana en las simulaciones con *data rate* 12 Mb/s, lo que quiere decir, que las simulaciones con tamaño de archivo más pequeño tienen un *throughput* relativo al maximo disponible mejor respecto de los *data rates* mayores para el mismo tamaño de archivo, cuando el tamaño de archivo es más pequeño.

5.6. Análisis de TCP Friendliness

5.6.1. Introducción

Para visualizar si el protocolo con control PD es amistoso con TCP se realizaron cuatro pruebas con el objetivo es analizar si se reparten equitativamente los recursos del canal, o si alguno toma ventaja respecto de otro. En el primer experimento se realizaron simulaciones de transmisiones simultáneas de cuatro servidores usando dos de ellos TCP Reno y los otros dos restantes, el protocolo estudiado; los servidores comienzan la transmisión en distintos momentos lo que hace que el ancho de banda disponible para cada servidor varíe en el tiempo. En una segunda prueba se hizo la misma simulación pero con todos los servidores usando TCP Reno a fin de comparar ambos experimentos y ver si las ventanas de los servidores con TCP Reno, servidor 3 y servidor 4, alcanzan valores semejantes en ambos experimentos, lo que permitirá discernir si el protocolo en cuestión es amigable con TCP. En el tercer ensayo se hizo una transmisión semejante a la primera, con dos servidores para cada protocolo, con la diferencia que no existe desfase, o sea todos los servidores comienzan al mismo tiempo y el ancho de banda, por lo tanto, es fija y no variable. Finalmente, la cuarta experiencia es análoga a la tercera, con todos los servidores iniciando su proceso al mismo tiempo, pero utilizando todos TCP Reno.

5.6.2. Pruebas TCP Friendliness

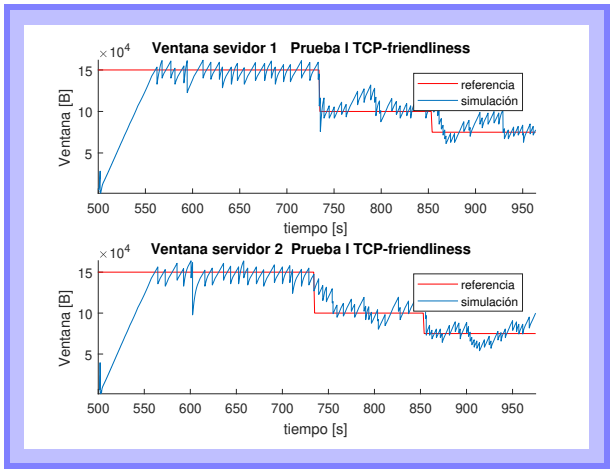
Prueba I TCP Friendliness

En la primera simulación los servidores 1 y 2 Figura 5.7a transmiten usando el protocolo que emplea control PD y los servidores 3 y 4 Figura 5.7c, TCP Reno. Los dos primeros empiezan a mandar información al mismo tiempo en 500 s , posteriormente en 735 s inicia el servidor 3, finalmente en 855 s parte el proceso el servidor 4. Si se incorpora un nuevo servidor a la tarea de envío de archivo el recurso disponible debe ser compartido entre los participantes, lo que en un escenario justo se refiere a que el ancho de banda total sea dividida en partes iguales. Esto es lo que indica la línea llamada Referencia, en la Figura 5.7b, es decir, teóricamente, el valor de ventana que le corresponde a un servidor en el tiempo, acorde a una repartición equilibrada. Es posible ver que las ventanas de los servidores 1 y 2 Figura 5.7a se ajustan a la referencia teórica en cada transición, es decir, no se apropian de los recursos, sino que ceden bajando el tamaño de sus ventanas permitiendo que los servidores que parten su transmisión puedan alcanzar un *throughput* adecuado.

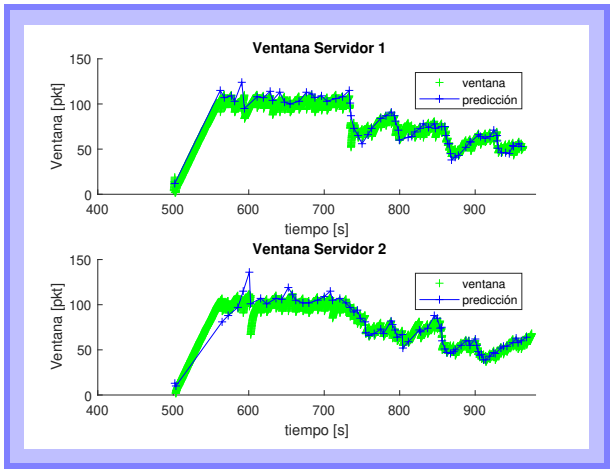
En la Figura 5.7b se aprecia la referencia real obtenida del promedio de una ventana móvil de seis elementos (ecuación 4.8) , aquí no se utiliza la predicción como el promedio total de todos los elementos como en los experimentos de la sección 5.3. Se ve que la ventana sigue a la referencia y que la predicción en los dos últimos tramos entre 735 s y 960 s oscila alrededor de el ancho de banda teórico que se ilustra en Figura 5.7a, posiblemente por la naturaleza del algoritmo de predicción usado. En el primer intervalo entre 500 s y 735 s, en cambio, la referencia tiene un crecimiento de rampa estabilizándose luego. En la Figura 5.7e se observa el comportamiento de AI y MD, esto a diferencia de 5.3.3 donde se ve que a partir de cierto tiempo toman un comportamiento regular, mientras que aquí se aprecian más fluctuaciones a causa de la variación de condiciones propia del experimento.

El promedio ponderado en el tiempo de las ventanas de los servidores 1 y 2 hasta el tiempo 960 s es cercano a 110000 B , considerando todas las variaciones en los diferentes intervalos, esto se muestra en detalle en la Tabla 5.10. Si se analizan los intervalos independientemente se puede ver que en el primero que considera el tiempo entre 500 s y 735 s el promedio de las ventanas de los servidores 1 y 2 es cercana a 129000 B , en el segundo entre 735 s y 855 s las ventanas son aproximadamente 107000 B y en el último tramo entre 855 s y 960 s el servidor 1 es aproximadamente 81900 B en cambio el servidor 2 alcanza 74400 B. El servidor 3 tiene un promedio cercano a 60000 B Tabla 4.9 considerando todo el tiempo. Los promedios por intervalo de cambio del tercer servidor son 65700 B en el primer tramo entre 735 s y 855 s; en el segundo, entre 855 s y 960 s es 53600 B. El servidor cuatro tiene un promedio ponderado de 51900 B aproximadamente.

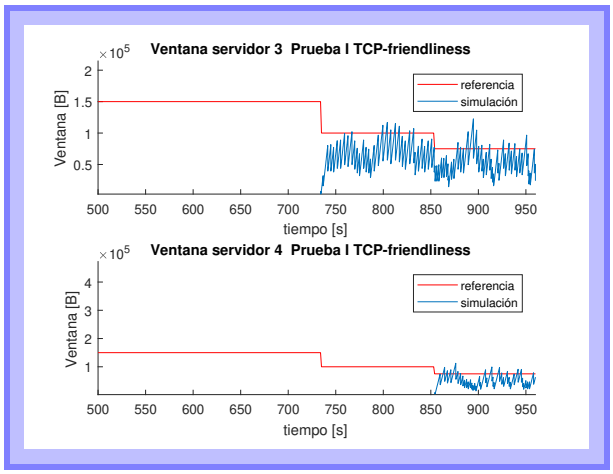
Se puede ver que las ventanas de los servidores 1 y 2 son casi simétricas entre sí salvo en el último tramo donde la del servidor 2 se queda un poco más arriba que la del servidor 1, también se puede apreciar que los promedios ponderados de ventanas alcanzados por tramos



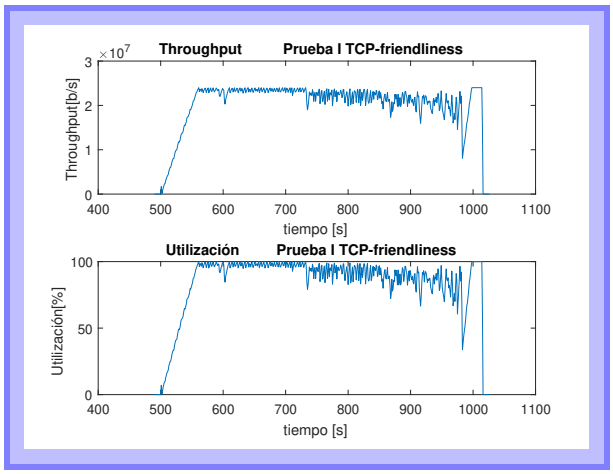
(a) Ventana v/s Referencia Servidores 1 y 2



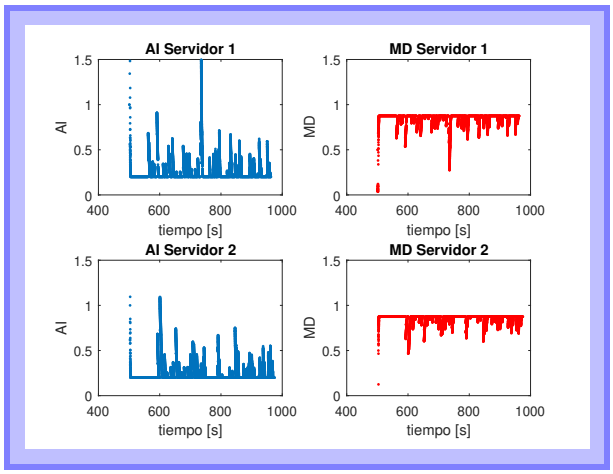
(b) Predicción Control PD Servidores 1 y 2



(c) Ventana v/s Referencia Servidores 3 y 4



(d) Throughput y Utilización



(e) AI y MD Control PD

Figura 5.7: Resultados Prueba I TCP-Friendliness

	Ventana Servidor1 [B]	Ventana Servidor2 [B]	Ventana Servidor3 [B]	Ventana Servidor4 [B]	Throughput [b/s]	Utilización [%]
Promedio	1.13e5	1.11e5	5.98e4	5.19e4	2.10e7	87.4
Desviación	2.91e4	3.02e4	2.01e4	1.98e4	4.58e6	19.1

Tabla 5.10: Pruebas I TCP friendliness

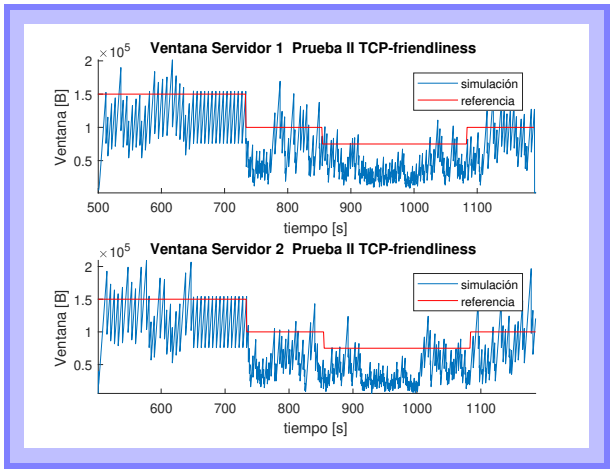
son cercanos a lo que se obtuvo en la sección 5.3; en el caso del servidor 1 el primer tramo se obtiene un promedio ponderado algo inferior al de la parte 5.3.4 donde llega a 132000 B aproximadamente, lo que se debe probablemente a que el tiempo de transmisión considerado es más corto. En los tramos siguientes se tienen muy similares que los de los apartados 5.3.3 (94000 B aprox.) y 5.3.2 (74000 B aprox.).

El *throughput* en Figura 5.7d tiene un promedio ponderado aproximado de 21 Mb/s y un 87 % de utilización, se observa de la gráfica que estos alcanzan valores mayores en el primer intervalo cuando están transmitiendo los servidores 1 y 2 que son los que utilizan el protocolo con control PD, en cambio, cuando comienzan el proceso los servidores 3 y 4 la banda de utilización y *throughput* aumentan disminuyendo por lo tanto el promedio, esto es consecuente con lo que se obtuvo en 5.3.

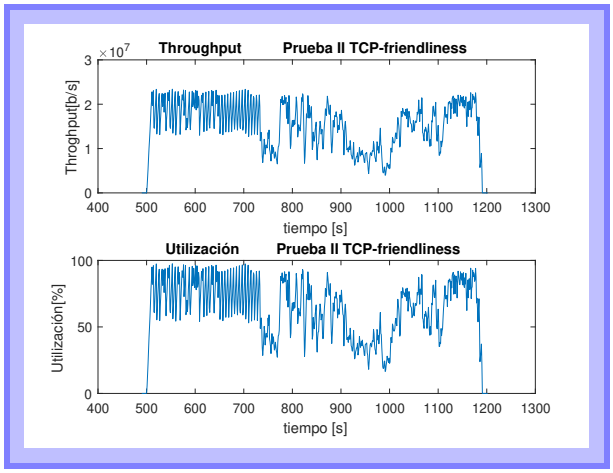
Prueba II TCP Friendliness

La segunda prueba consistió en una simulación con los mismos desfases de tiempo en otras palabras, los servidores 1 y 2 comienzan como antes en 500 s, el servidor 3 en 735 s y el 4 en 855 s, pero en cambio, el protocolo TCP Reno se usará en todos los servidores con el fin de medir los tamaños de ventanas que se alcanzan. Se puede notar de la Figura 5.8a que el promedio ponderado de las ventanas de los servidores 1 y 2 a diferencia de la prueba anterior se quedan bajo la línea del ancho de banda teórica disponible, no aprovechando adecuadamente el ancho de banda libre. El promedio de ventana ponderado de los servidores 1 y 2 es aproximadamente 75000 B. Mas se distinguen en el gráfico cuatro intervalos con diferente *throughput* teórico; analizando los promedios ponderados por intervalo se ve que en el primer tramo en el servidor 1 es cercano a 112000 B y en el servidor 2 es aproximadamente 119000 B. En el segundo tramo entre 735 s y 855 s es 68000 B y 56006 B para el servidor 1 y 2 respectivamente. En el tercer tramo entre 855 s y 1084 s son aproximadamente ambos 40000 B. En el cuarto tramo entre 1084 s y 1182 s los promedios son 76000 B y 81000 B respectivamente los promedios de ventana de los servidores 1 y 2. En cuanto al servidor 3 esto llega a 56800 B en su primer intervalo entre 735 s y 855 s y el segundo intervalo entre 855 s y 1084 s es 39100 B. El cuarto servidor entre 855 s y 1084 s es 39193 B en el último tramo entre 1084 s y 1182 s es 69100 B.

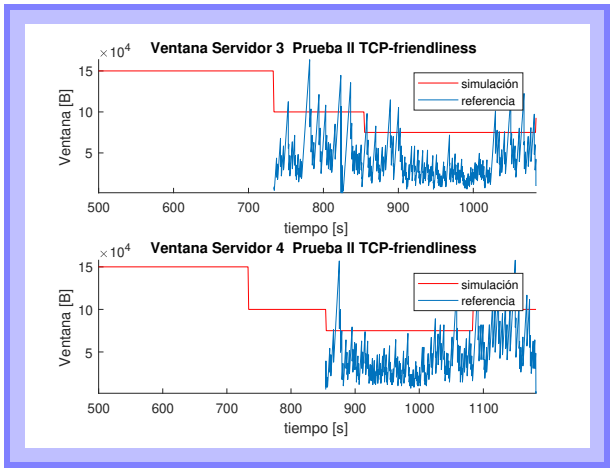
De lo anterior y de lo que a simple vista se percibe de Figura 5.8a y Figura 5.8c el pro-



(a) Ventana v/s Referencia Servidores 1 y 2



(b) Throughput y Utilización



(c) Ventana v/s Referencia Servidores 3 y 4

Figura 5.8: Resultados Prueba II TCP-Friendliness

medio por intervalo es bastante inferior al ancho de banda libre en cada tramo. El tamaño de las ventanas de los servidores 3 y 4 son inferiores a los que se obtiene cuando se realizó la prueba donde los servidores 1 y 2 usan el protocolo con control PD, esto se ve claramente de la Tabla 5.10 y también de lo expuesto en el párrafo anterior. Por ejemplo, cuando la tasa disponible es 100000 B el servidor 3 obtiene una ventana de tamaño 56800 B en la prueba 2, en cambio en la prueba 1 es 65700 B; cuando la tasa disponible es 75000 B en la prueba 2 el servidor 3 llega a un promedio de 39100 B, en cambio, en la prueba 1 el promedio del servidor 3 en las mismas condiciones es 53600 B. Se puede concluir que el protocolo con control PD no sólo es amistoso con TCP permitiendo que sus ventanas crezcan rápidamente sino que también permite que el protocolo TCP logre valores promedio más altos que los que consigue cuando todos los servidores transmiten con TCP. Esto se debe posiblemente a que TCP puede encontrar un promedio mayor por las bajas oscilaciones del protocolo con control PD.

	Ventana Servidor1 [B]	Ventana Servidor2 [B]	Ventana Servidor3 [B]	Ventana Servidor4 [B]	Throughput [b/s]	Utilización [%]
Promedio	7.54e4	7.62e4	4.53e4	4.81e4	1.55e7	64.6
Desviación	3.88e4	4.35e4	3.20e4	3.16e4	4.97e6	20.7

Tabla 5.11: Pruebas II TCP friendliness

	Ventana Servidor1 [B]	Ventana Servidor2 [B]	Ventana Servidor3 [B]	Ventana Servidor4 [B]	Throughput [b/s]	Utilización [%]
Promedio	7.78e4	7.44e4	6.38e4	6.27e4	1.98e7	82.5
Desviación	1.60e4	1.28e4	3.95e4	4.71e4	3.29e7	13.7

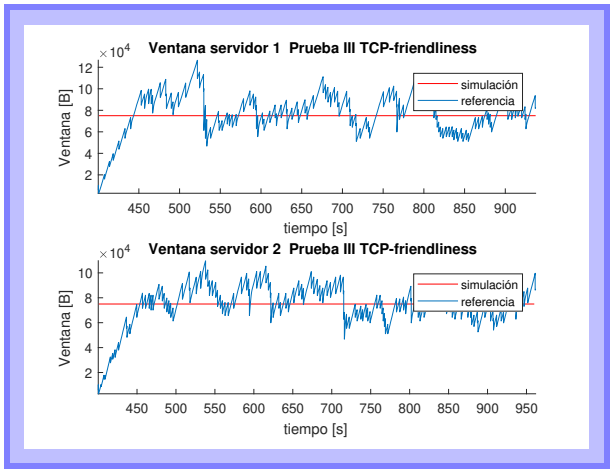
Tabla 5.12: Prueba III TCP friendliness

Prueba III TCP Friendliness

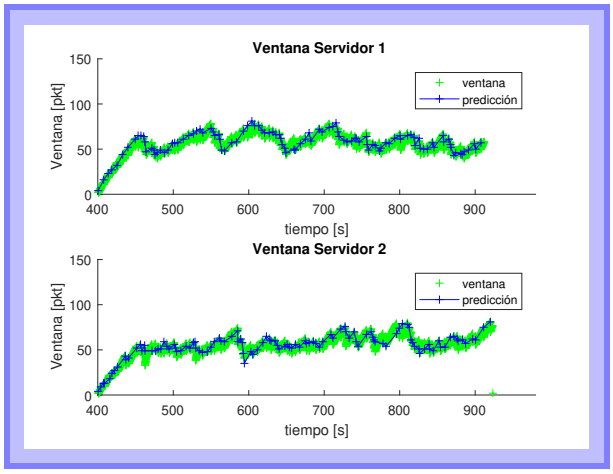
En el tercer experimento se realizaron transmisiones sin desfase de tiempo usando 4 servidores: los dos primeros utilizan el protocolo en estudio y los otros TCP Reno. Se puede observar de la Tabla 5.12 que los servidores que usan el protocolo con control PD se ajustan a la tasa disponible que en este caso es 75000 B para cada servidor. Los servidores 1 y 2 tienen un promedio ponderado de 77000 B y 74000 B aproximadamente. Las ventanas de los servidores con protocolo TCP en cambio tienen un promedio cercano 63000 B y 62000 B respectivamente. Sin embargo, en estos último 2 se distinguen dos zonas, una en que están compartiendo el canal con los otros servidores y otra donde transmiten solos los servidores 3 y 4. Calculando el promedio en el intervalo entre 400 s y 937 s se obtiene 52146 B y 50883 B respectivamente en los servidores 3 y 4.

En Figura 5.9b se ve la referencia resultado de la predicción y la ventana, se observa que la ventana sigue la referencia, y que esta última tiene un comportamiento oscilatorio que se debe al método para obtener la predicción usado en los experimentos que sólo considera la media de seis valores (ecuación 4.8) lo que determina un resultado variable en el tiempo, a diferencia de lo que resulta si la referencia consigue del promedio de todos los valores como en el apartado 5.3 donde ésta y la ventana tienen una apariencia casi constante a partir de cierto tiempo. Se utiliza este algoritmo para tener la predicción porque favorece el desempeño cuando ocurren variaciones de ancho de banda como se describe en la sección 5.7.

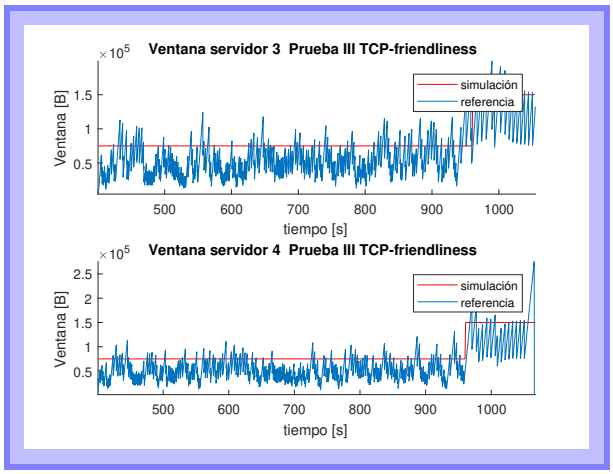
En la Figura 5.9e se aprecia las variaciones en el tiempo de la variable de control AI, y MD. Tienen un comportamiento más regular que en la prueba uno de *TCP friendliness* en Figura 5.7e, pero debido a las oscilaciones que presenta la predicción no son constantes a contar de un tiempo como ocurre con los gráficos de AI y MD de la sección 5.3.



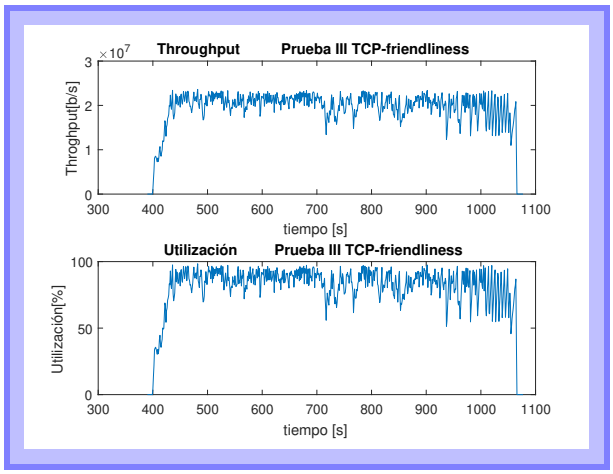
(a) Ventana v/s Referencia Servidores 1 y 2



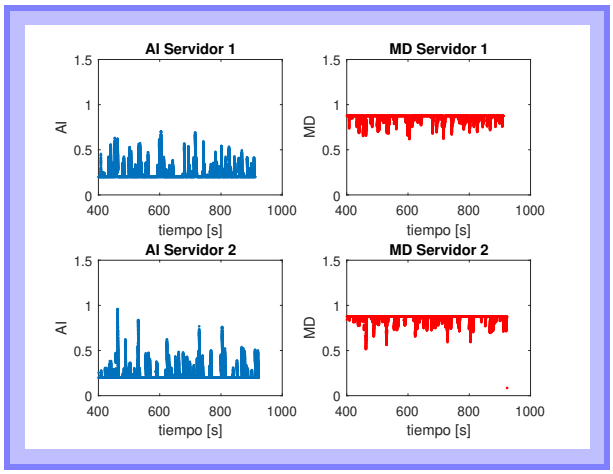
(b) Predicción Control PD Servidores 1 y 2



(c) Ventana v/s Referencia Servidores 3 y 4

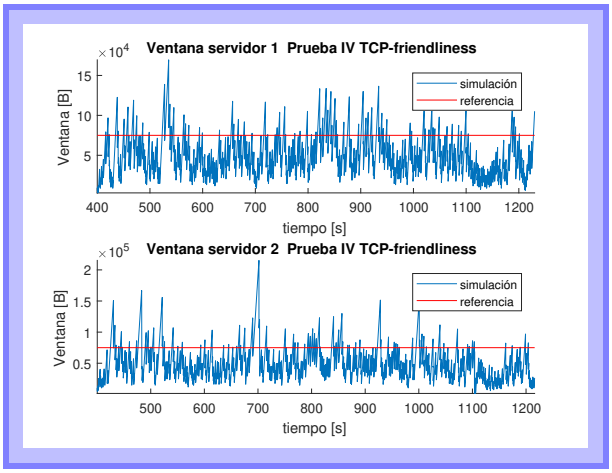


(d) Throughput y Utilización

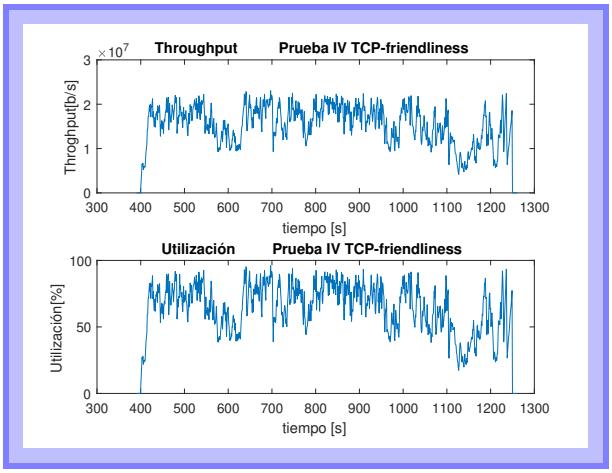


(e) AI y MD Control PD

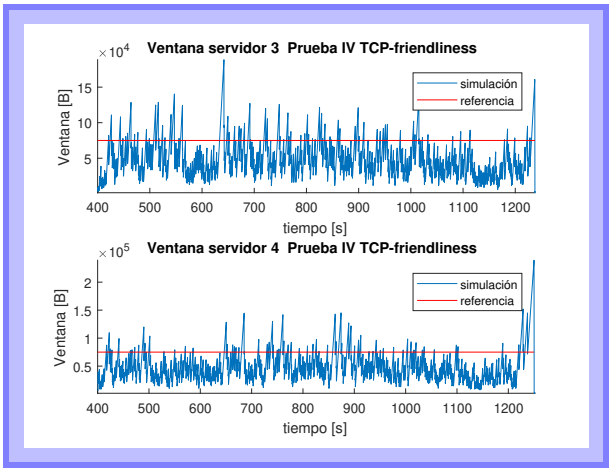
Figura 5.9: Resultados Prueba III TCP-Friendliness



(a) Ventana v/s Referencia Servidores 1 y 2



(b) Throughput y Utilización



(c) Ventana v/s Referencia Servidores 3 y 4

Figura 5.10: Resultados Prueba IV TCP-Friendliness

Prueba IV TCP Friendliness

Se realizó finalmente una cuarta prueba donde los cuatro servidores transmiten un archivo del mismo tamaño 400 MB , sin desfase en el tiempo y usando TCP Reno. Los valores promedios ponderados de las ventanas obtenidas en este caso son cercanos a 50000 B Tabla 5.13 y esto es menor al ancho de banda potencial del canal 75000 B para cada servidor indicada por la línea roja de la Figura 5.10a. Además, observando los resultados de la tercera prueba, se puede notar que los servidores 3 y 4, que en ambas experiencias funcionan mediante el protocolo TCP Reno tienen un promedio ponderado de su ventana de congestión levemente superior en el caso en que los servidores 1 y 2 utilizan el protocolo con control PD.

	Ventana Servidor1 [B]	Ventana Servidor2 [B]	Ventana Servidor3 [B]	Ventana Servidor4 [B]	Throughput [b/s]	Utilización [%]
Promedio	5.05e4	5.14e4	5.00e4	4.93e4	1.55e7	64.7
Desviación	2.74e4	3.40e4	2.97e4	3.69e4	4.02e6	17.9

Tabla 5.13: Prueba TCP friendliness sin desfases en el tiempo

	Ventana Servidor1 [B]	Ventana Servidor2 [B]	Ventana Servidor3 [B]	Ventana Servidor4 [B]	Throughput [b/s]	Utilización [%]
Prueba I	1.13e5	1.11e5	5.98e4	5.19e4	2.10e7	87.4
Prueba II	7.54e4	7.62e4	4.53e4	4.81e4	1.55e7	64.6
Prueba III	7.78e4	7.44e4	6.38e4	6.27e4	1.98e7	82.5
Prueba IV	5.05e4	5.14e4	5.00e4	4.93e4	1.55e7	64.7

Tabla 5.14: Resumen Promedios Pruebas TCP Friendliness

5.6.3. Resumen

En conclusión, de los cuatro experimentos realizados se tiene que el protocolo con control PD tiene ventanas con promedio ponderado cercano a la tasa potencial disponible en ese tiempo, y eso se cumple tanto para los casos donde el ancho de banda disponible se mantiene constante en el tiempo o bien cuando esto varía. Además este protocolo permite alcanzar un *throughput* mayor cuando se usa en todos los computadores que transmiten, o incluso si sólo se aplica en algunos de ellos mejorando el resultado total. Además es un protocolo amistoso que permite que quien se integra a la transmisión rápidamente alcance un tamaño promedio de ventana adecuado que puede incluso llegar a ser mayor que el que se obtendría si todos los servidores usaran TCP Reno.

5.7. Análisis del Throughput respecto de las Variaciones en la Condiciones de Diseño

5.7.1. Introducción

Se analiza en esta sección cómo afectan tres diferentes variantes de diseño al *throughput* y al error de seguimiento, que es el error de la variable de salida respecto de la referencia teórica de ancho de banda (ver ecuación 4.11). Las variantes corresponden a modificaciones en las técnicas utilizadas en las pruebas de la sección anterior 5.6. Los resultados se compararán con los que se obtuvieron en el apartado 5.6. En las tablas Tabla 5.16, Tabla 5.17 y Tabla 5.18 el error 1 es el error respecto del ancho de banda teórico y el error 2 o de seguimiento

Caso	N° Controladores	Tipo de Predicción	Tipo de simulación
Caso A	3	P^2	I TCP Friendliness
Caso B	1	P^2	I TCP Friendliness-V
Caso C	3	P^1	I TCP Friendliness-V
Caso E	1	P^2	III TCP Friendliness
Caso D	1	P^1	III TCP Friendliness-V

Tabla 5.15: Casos de Estudio

es el error respecto de la referencia real o predicción.

Los cambios se realizan sobre dos características:

- El número de controladores usados.
- El tipo de predicción utilizada.

Las modificaciones se realizan sobre las pruebas I y III (ver secciones 5.6.2 y 5.6.2) de *TCP friendliness* y así a partir de ellas se definen una serie de casos que se resumen en la tabla 5.15. Se debe recordar que la prueba I de TCP friendliness consiste en cuatro servidores que transmiten con cierto desfase en el tiempo sobre el mismo canal lo que provoca una fluctuación en el ancho de banda disponible. Por lo anterior son utilizados tres controladores PD uno para cada nivel que alcanza el *throughput* de referencia. Además la prueba mencionada utiliza un tipo de predicción de ventana móvil que promedia seis valores de ventana del pasado y descarta al resto, este tipo de predicción aquí es denominada como P_2. La prueba I TCP Friendliness en esta sección es etiquetada con el nombre Caso A.

A partir de la prueba I se da origen a dos variantes los denominados Caso B y Caso C. Se produce el caso B reemplazando los tres controladores utilizados en la prueba I TCP friendliness por uno solo. Como en este experimento la tasa de envío disponible varía en el tiempo se espera que se obtenga un peor desempeño con esta alteración. El caso C proviene de la modificación del tipo de predicción, el método usado que promedia un número de valores de la ventana es cambiado por otro que calcula la predicción mediante el promedio todos los valores de la ventana.

Por otro lado, la prueba III TCP *Friendliness* corresponde a una simulación donde los cuatros servidores comienzan una transmisión al mismo tiempo. En esta simulación no se requiere de varios controladores basta solo con uno pues el ancho de banda disponible del canal se mantiene constante en el tiempo. La predicción utilizada en esta prueba coincide con la prueba I TCP Friendliness o Caso A en este apartado. La prueba III TCP Friendliness es aquí nombrada como Caso C. En base a este se deriva el Caso E que difiere del anterior en el tipo de predicción utilizada y que coincide con la denominada P_1. Para la prueba III no es posible generar una variante modificando el número de controladores, esto no tendría sentido pues originalmente se utiliza solo uno por manterse constante el ancho de banda disponible.

Caso B				Caso A			
Throughput [b/s]	Utilización [%]	Error1 [pkt]	Error 2 [pkt]	Throughput [b/s]	Utilización [%]	Error1 [pkt]	Error2 [pkt]
2.09e7	87.2	12.3	12.1	2.10e7	87.4	11.1	11.3

Tabla 5.16: Comparación de throughput cuando se usa un controlador

Caso C				Caso A			
Throughput [b/s]	Utilización [%]	Error1 [pkt]	Error2 [pkt]	Throughput [b/s]	Utilizacion [%]	Error1 [pkt]	Error2 [pkt]
2.02e7	84.1	15.2	16.3	2.10e	87.4	11.2	11.3

Tabla 5.17: Comparación de *throughput* cuando se usa la predicción del promedio total

5.7.2. Variantes del Caso A

La primera situación de estudio corresponde al caso B donde se usa un solo controlador en vez de tres , como se hizo en la primera prueba de *TCP-friendliness*. Los resultados se resumen en Tabla 5.16 el error 1 se calcula en los servidores 1 y 2 sumados hasta 960 s de la simulación. Es posible notar que utilizar tres controladores tiene un mejor resultado en cuanto a seguimiento y un mejor *throughput* que si se utiliza un solo controlador, esto es coherente con lo que se sabe del modelo del sistema; este es no lineal y la referencia no es constante. Un controlador lineal tiene mejor desempeño en el entorno del punto para el que fue diseñado y peor al alejarse, entonces sintonizar controladores en varios puntos del intervalo donde varía la referencia mejora el desempeño.

En segundo lugar se analizó el denominado caso C donde la prueba I TCP friendliness se modifica en su fórmula de predicción la que en su transformación calcula el promedio de todos los elementos para obtener la mencionada estimación. Los resultados se exponen en la Tabla 5.17.

En el caso A (ver Tabla 5.10), donde existe variación del ancho de banda en el tiempo, el seguimiento y el *throughput* son mayores cuando se usa la predicción que hace el promedio de seis elementos de una ventana móvil, el método usado en la sección anterior. Al usar el promedio de todos los elementos la utilización disminuye en casi un 3 % Tabla 5.17 porque el valor final no refleja bien lo que ocurre realmente en cuanto al *throughput* y eso finalmente afecta el desempeño del controlador pues causa mayores oscilaciones.

5.7.3. Variante Caso E

Cuando se hace el experimento en la prueba 3 de *TCP-friendliness* Tabla 5.12, donde la tasa de envío se mantiene constante en el tiempo, el resultado es más parecido entre sí

Caso E				Caso D			
Throughput [b/s]	Utilización [%]	Error1 [pkt]	Error2 [pkt]	Throughput [b/s]	Utilización [%]	Error1 [pkt]	Error2 [pkt]
2.00e7	83.3	10.1	5.87	82.5	1.98e7	10.4	4.26

Tabla 5.18: Comparación de *throughput* cuando se usa un controlador

Caso	Prueba Asociada	Utilización[%]	Throughput
Caso A	I TCP Friendliness	87.4	2.10e7
Caso B		87.2	2.09e7
Caso C		84.1	2.02e7
Caso E	III TCP Friendliness	82.5	1.98e7
Caso D		83.3	2.00e7

Tabla 5.19: Resumen

Tabla 5.18, pero al contrario de la experiencia anterior el *throughput* levemente mayor es el obtenido de la implementación de de la predicción de acuerdo a la ecuación 4.7, esto es porque la referencia generada de esa forma tiene menos fluctuaciones y a la vez es capaz de hacer un buen pronóstico del ancho de banda real disponible a consecuencia de lo cual las ventanas oscilan menos y resulta una mejor utilización del recurso libre.

5.7.4. Resumen

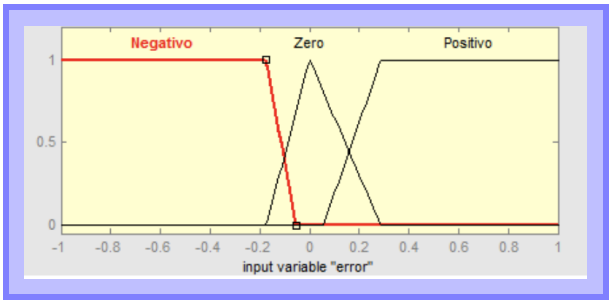
En resumen, se puede concluir de estos experimentos que el tipo de predicción P^1 que promedia toda la ventana resulta mejor cuando no hay cambio del *throughput* disponible. Y que usar un controlador en vez de tres cuando hay cambio de ancho de banda resulta peor, lo que es coherente con que el proceso es no lineal 5.19.

5.8. Resultados Controlador Difuso

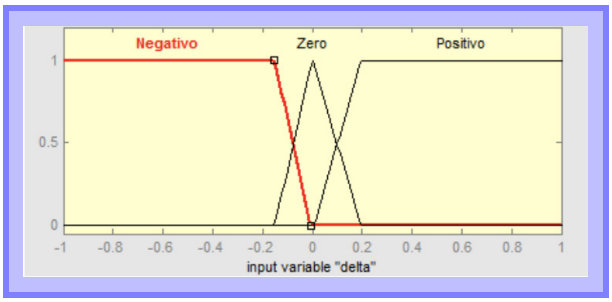
En este apartado se exponen los resultados de la implementación del método de optimización que permite encontrar los parámetros de un controlador difuso idóneo y también se presentan los resultados de simulaciones y desempeño para dos versiones de controlador que emplean la solución hallada por la optimización.

De acuerdo a lo explicado en la parte 4.4.7 se ejecutó la optimización PSO para encontrar las posiciones de los conjuntos de pertenencia óptimos que produjeran una menor sumatoria de error ponderado (ver ecuación 4.14) respecto del ancho de banda teórico. Se realizaron 6 iteraciones con cinco partículas cada una. Las funciones de pertenencia resultantes se muestran en la Figura 5.11b y Figura 5.11b.

Esto fue obtenido de simulaciones realizadas con la base de reglas indicada en la Ta-



(a) Funciones de Pertenencia Error



(b) Funciones de Pertenencia Delta Error

Figura 5.11: Funciones de Pertenencia Obtenidos mediante PSO

Variable	Trapecio Negativo	Triangulo Cero	Trapecio Positivo
Error	[-1 -1 -0.1736 -0.05541]	[-0.1736 0 0.2867]	[0.05541 0.2867 1 1]
Δ Error	[-1 -1 -0.1515 -0.006078]	[-0.1515 0 0.196]	[0.006078 0.196 1 1]

Tabla 5.20: Funciones de pertenencia resultado PSO

bla 5.21. Sin embargo se observó que los valores resultantes de AI y MD aumentaron las oscilaciones, pues ocurre que cuando el tamaño de ventana es mayor que la referencia, se activan reglas que implican una brusca caída al momento de una pérdida lo que incrementa las fluctuaciones y, por lo tanto, disminuye el *throughput*. Con el fin de mejorar esto en los experimentos a continuación se utilizó otra base de reglas descrita en la Tabla 5.21. El método para decidir las reglas es el mismo que se usó en las de la Tabla 4.6, con la diferencia que se eligieron valores más cercanos a los de la relación GAIMD *TCP-friendly* en (ver ecuación 4.9).

En el primer experimento se realizó la misma simulación de la prueba 1 de *TCP friendliness* que considera variaciones de la tasa de envío teórica debida al comienzo en desfase de las transmisiones de los servidores empleando ahora el controlador PD difuso descrito en Tabla 5.21.

En el segundo experimento se llevó a cabo la misma simulación anterior usando un controlador difuso con los mismos parámetros de las funciones de pertenencia del anterior, es decir, los obtenidos de la optimización PSO realizada pero con una base de reglas que respeta la relación GAIMD *TCP-friendly* en 4.10; el objetivo es comparar el desempeño de ambos y también respecto de la simulación de los controladores PD. El tipo de transmisión para ambos experimentos es el mismo que el de la sección 5.6 (Figura 4.27 y Figura 4.28), es decir,

De/e	Negativo	Cero	Positivo
Positivo	AI=0.2 MD=0.5293	AI=0.2 MD=0.875	AI=1.6 MD=0.8
Cero	AI=0.2 MD=0.6203	AI=0.2 MD 0.875	AI=1.3 MD=0.65
Negativo	AI=0.2 MD=0.725	AI=0.2 MD=0.875	AI=1 MD=0.65

Tabla 5.21: Tabla base de reglas AI y MD independientes

De/e	Negativo	Cero	Positivo
Positivo	AI=0.2 MD=0.875	AI=0.2 MD=0.875	AI=1.6 MD=0.3043
Cero	AI=0.25 MD=0.8462	AI=0.2 MD 0.875	AI=1.3 MD=0.3953
Negativo	AI=0.2 MD=0.8182	AI=0.2 MD=0.875	AI=1 MD=0.5

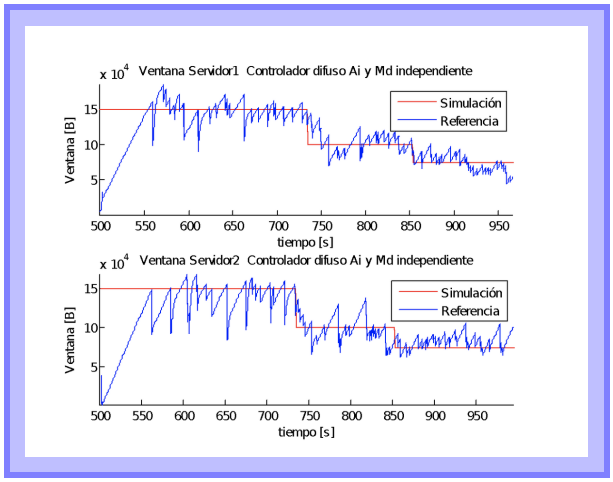
Tabla 5.22: Tabla de base de reglas AI y MD dependientes

	Ventana Servidor1 [B]	Ventana Servidor2 [B]	Ventana Servidor3 [B]	Ventana Servidor4 [B]	Throughput [b/s]	Utilización [%]
Promedio AI y MD Dependientes	1.13e5	1.11e5	6.09e4	4.86e4	2.09e7	87.3
Desviación estándar AI y MD Dependientes	3.03e4	3.06e4	2.10e4	1.81e4	4.44e6	18.5
Promedio AI y MD Independientes	1.12e5	1.07e5	6.18e4	5.34e4	2.08e7	86.7
Desviación estándar AI y MD Independientes	3.21e4	2.96e4	2.35e4	1.76e4	4.25e6	17.7

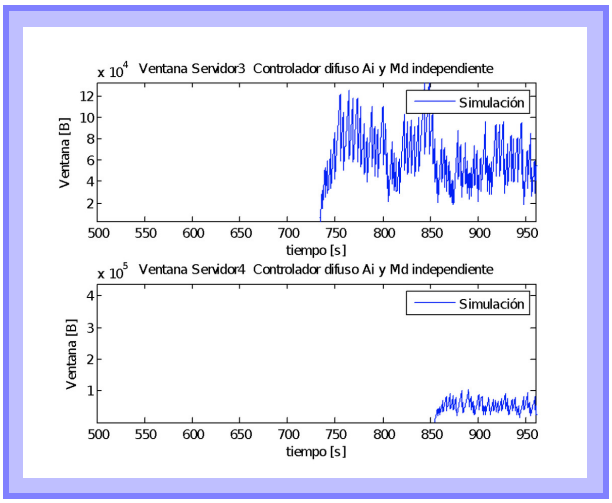
Tabla 5.23: Prueba controladores PD difusos

comienzan el proceso los servidores 1 y 2 en 500 s que usan el protocolo en estudio, luego en 735 s comienza el servidor 3 que envía paquetes mediante TCP Reno y finalmente en 855 s el servidor 4 que también usa TCP Reno.

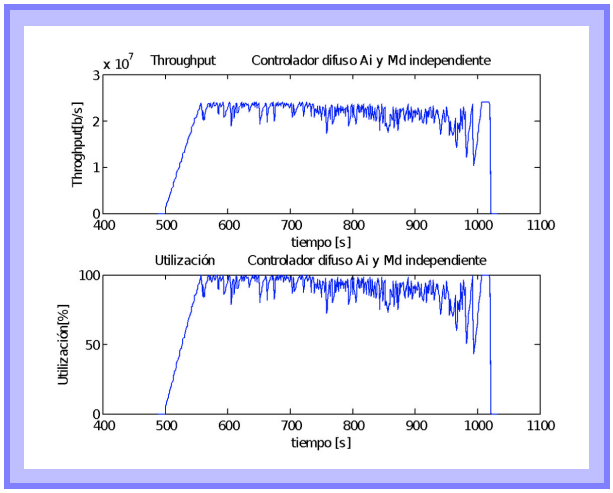
En la Figura 5.12a y Figura 5.12b se presentan las ventanas de congestión resultantes del primer experimento donde se utiliza el controlador difuso con AI y MD independientes, se ve que siguen la línea que representa la tasa de envío teórica para cada servidor, sin embargo, también es posible notar que ocurren bajadas bruscas cuando hay una pérdida y el tamaño de la ventana está sobre la referencia, esto sucede porque en ese instante la variable de salida también estaba sobre la referencia real, la estimada por la predicción, así al momento de producirse la pérdida se activan reglas con un valor de MD muy bajo que provocan finalmente esta abrupta caída. El promedio ponderado de las ventanas de congestión de los servidores 1 y 2 es levemente inferior en este caso (Tabla 5.23) respecto del que utiliza los tres controladores PD (Tabla 5.10), para los servidores 3 y 4 es al revés, es decir, son levemente superiores. El *throughput* y la utilización son inferiores que las del protocolo con controladores PD Tabla 5.23. El error de ajuste respecto de la referencia teórica promedio de los servidores 1 y 2 es 12.768 pkt.



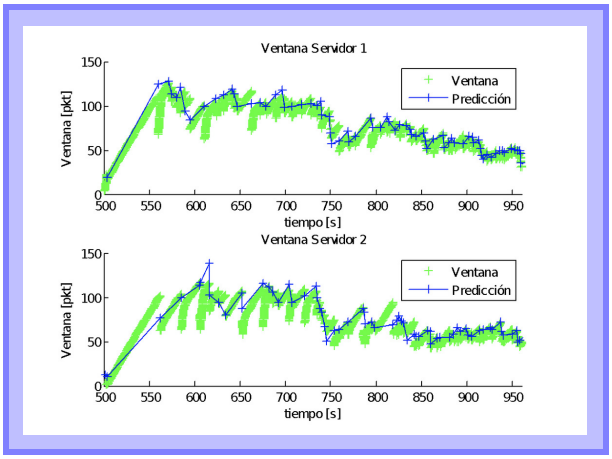
(a) Ventana v/s Referencia Servidores 1 y 2



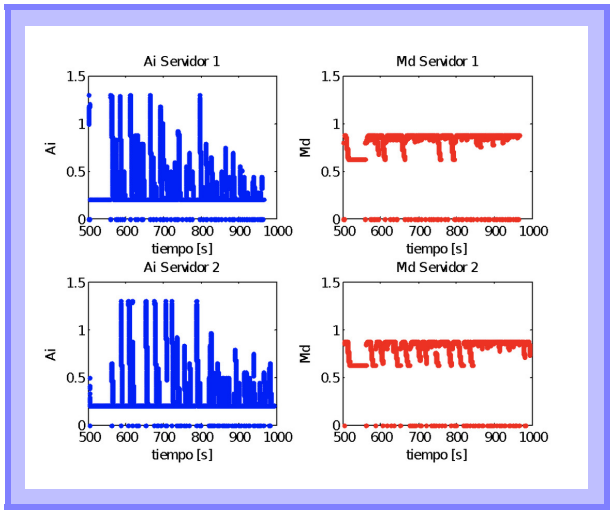
(b) Ventana v/s Referencia Servidores 3 y 4



(c) Throughput y Utilización

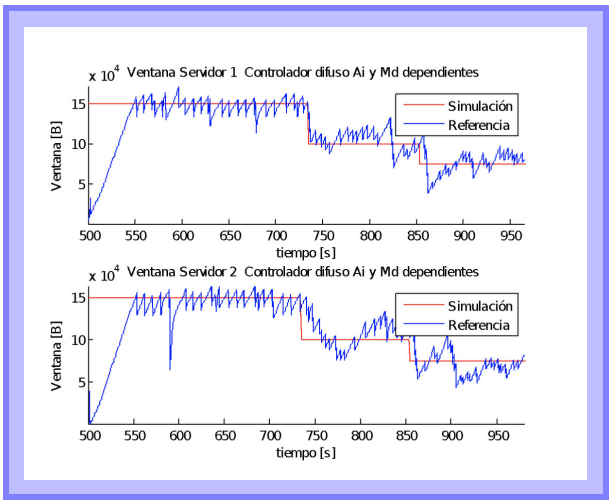


(d) Ventana v/s Predicción Servidores 1 y 2

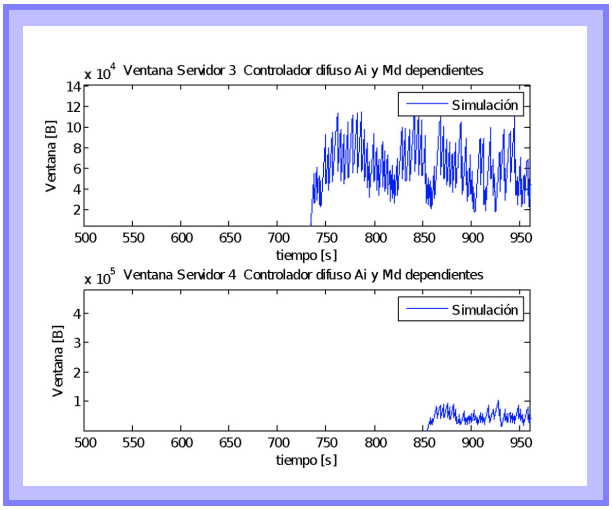


(e) AI y MD

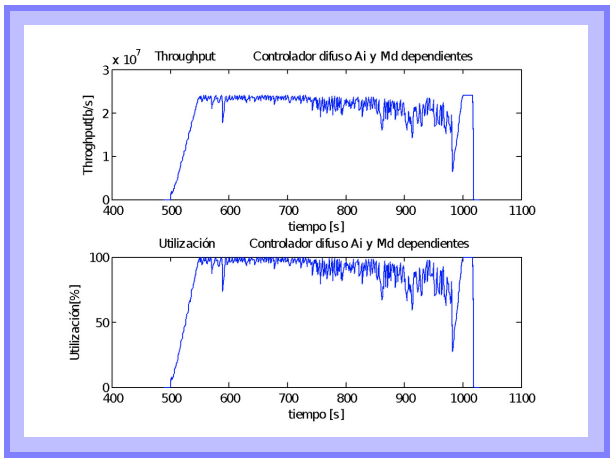
Figura 5.12: Resultados Controlador Difuso Caso 1



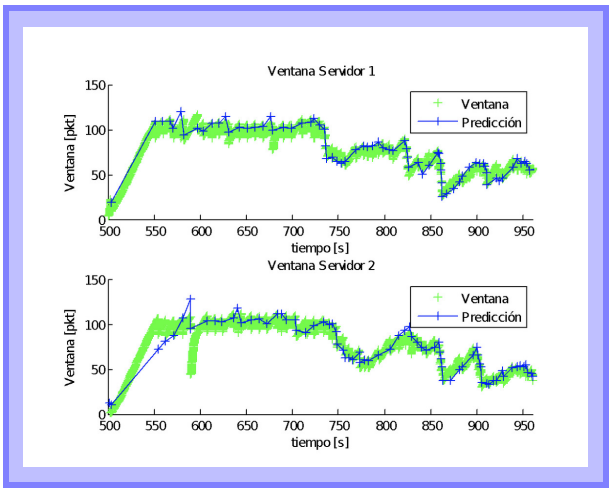
(a) Ventana v/s Referencia Servidores 1 y 2



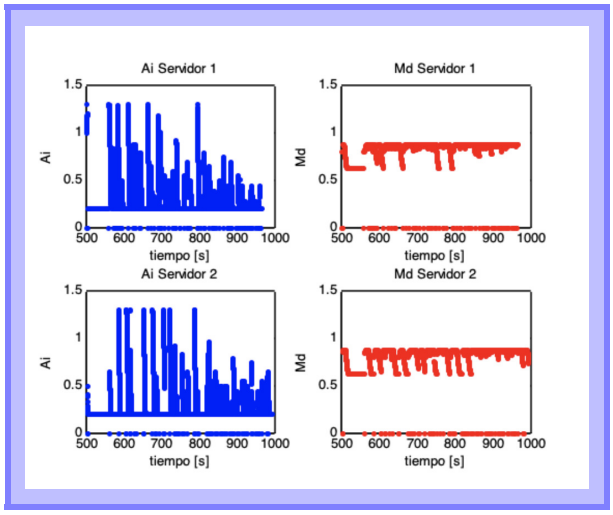
(b) Ventana v/s Referencia Servidores 3 y 4



(c) Throughput y Utilización



(d) Ventana v/s Predicción Servidores 1 y 2



(e) AI y MD

Figura 5.13: Resultados Controlador Difuso Caso 2

El intervalo entre 735 s y 855 s del servidor 3 tiene un promedio ponderado de 71100 B aproximadamente y el intervalo entre 855 s y 960 s tiene un promedio ponderado cercano a 51800 B.

El servidor 4 entre 855 s y 960 s tiene una ventana de congestión cuyo promedio ponderado es 53800 B aproximado.

De los resultados obtenidos en la sección 5.6 en la prueba 2 de *TCP-friendliness* Tabla 5.11 donde todos los servidores transmiten con TCP Reno es claro que el protocolo con controlador difuso y AI y MD independientes es amistoso, pues en la situación en que el ancho de banda libre es 100000 B el tamaño de ventana del servidor 3 era cercano a 56800 B ,cuando todos los servidores funcionaban con TCP Reno, en cambio aquí el promedio es 71100 B . Cuando la tasa de envío es 75000 B el servidor 3 alcanzaba apenas 39100 B con el protocolo clásico, en cambio en este caso 53800 B.

Los resultados del segundo experimento donde se utiliza un controlador difuso con AI y MD dependientes se exponen en la Figura 5.13. Se puede notar que menos hay oscilaciones y que las ventanas se ajustan de mejor manera al ancho de banda existente. El error de ajuste promedio respecto de la referencia teórica de los servidores 1 y 2 es 11.670 pkt, esto es menor que para el controlador difuso con AI y MD independientes; el error es mayor que en la prueba 1 de *TCP friendliness* que usa tres controladores, pero es menor y por lo tanto mejor que el caso que usa un solo controlador PD (Tabla 5.17 error 1). Esto muestra el buen desempeño que tiene un controlador PD difuso incluso cuando la sintonización no fue demasiado exhaustiva para el conjunto de reglas y no se consideraron todos los puntos en diseño que se usaron en el examen de funcionamiento.

Los promedios ponderados Tabla 5.23 de las ventanas de congestión de sus servidores son levemente inferiores que las que produce el controlador difuso donde AI y MD son independientes y además el *throughput* y utilización son mejores y sobrepasan el 87 %. Al comparar con el desempeño del protocolo que usa tres controladores PD Tabla 5.10 se puede apreciar que los promedios de ventana de los servidores 1, 2 y 4 son levemente inferiores en el protocolo con controlador difuso, y al revés en el servidor 3. El throughput y la utilización son ligeramente superiores en el protocolo con controladores PD. No obstante, el *throughput* y la utilización son mayores que la prueba realizada en 5.6.2 donde se usó solo un controlador PD cuyo resultado se muestra en la Tabla 5.17 , lo que confirma el buen desempeño de un controlador difuso para un sistema no lineal respecto de un lineal.

En la Figura 5.13d se aprecian las ventanas respecto de la referencia y en Figura 5.13e se observa el comportamiento de AI y MD bajo el uso de este controlador.

5.8.1. Resumen del Capítulo

En este capítulo se presentaron los resultados de la sintonización de un controlador PD y de un controlador difuso. Se comparó el desempeño en un escenario de dos servidores y dos *workstation*, en un caso se utiliza TCP Reno en ambos transmitiendo al mismo tiempo y en el otro caso ambos utilizan el protocolo propueto. Se encontró que en todos los casos

es superior el protocolo propuesto a TCP Reno y la mejora es aproximadamente del 10 % en caso. Esto se debe a que las oscilaciones son menores, el ajuste al ancho de banda es mejor gracias al control y por lo tanto, el *throughput* en promedio es mayor.

Se hicieron pruebas de simulaciones tanto para el protocolo usando el controlador PD como TCP Reno con diferentes semillas y se encontró que la dispersión de los datos es baja comparada con el promedio de los valores de *throughput* y utilización. Además se obtuvo que el promedio de *throughput* y utilización en mayor. Además se probaron los protocolos en la transmisión de archivos de distintos tamaños y la utilización es más cercana al máximo 100 % en caso del protocolo que utiliza un controlador PD. Para archivos más grandes la utilización es mayor pues la transmisión permanece más tiempo operando en el valor de ancho de banda disponible que encontrando la predicción, la predicción tarda en encontrar ese valor y esto tiene impacto en la transmisión de archivos más pequeños. De este experimento se puede observar que la predicción no es tan rápida como se quisiera. Esto es porque la predicción tiene un comportamiento incremental y está basada en las perdidas obtenidas que ocurren incluso al principio de la transmisión.

Las pruebas de *TCP friendliness* encontraron que al ejecutarse ambos protocolos, el que usa un controlador PD y TCP Reno simultáneamente. TCP Reno puede alcanzar un *throughput* incluso mayor que en el caso en que todos los servidores ejecutan TCP Reno de lo que se puede concluir que el protocolo que integra un controlador PD se comporta de manera justa sin acaparar los recursos. Esto se debe a que se ajusta al ancho de banda que detecta la predicción lo que permite al otro flujo poder transmitir a buena velocidad.

Se obtuvieron los valores de sintonización del controlador difuso y se hicieron pruebas de comportamiento de este. En los experimentos que utilizaron el controlador difuso en conjunto con TCP Reno en transmisiones simultáneas se obtuvo que TCP Reno accede al *throughput* disponible tal como ocurriría si todos los servidores transmitieran usando TCP Reno de modo que se alcanza un promedio de utilización incluso mayor que en ese caso. De allí se puede afirmar que el protocolo que usa un controlador difuso tiene un comportamiento *TCP-friendly*. Se probó el controlador difuso utilizando dos set de reglas. Esto se debe a que se ajusta al ancho de banda de acuerdo a la predicción y no tiene demasiadas oscilaciones que disminuyan el ancho de banda para el flujo con el que comparte el canal.

6. Conclusiones

En este trabajo se estudió el desempeño de un protocolo de transporte basado en TCP en términos del aprovechamiento del canal, velocidad de transmisión y *TCP-friendliness*. Se programaron en OPNET las simulaciones necesarias para estudiar el comportamiento de los dos métodos de control de congestión planteados basados en control clásico y control difuso. Fue posible sintonizar los parámetros de los controladores PD para tres referencias distintas mediante simulaciones del algoritmo PSO que permitieron obtener una utilización promedio del canal, en condiciones de referencia fija y al enviar un archivo de 400[MB], entre 93.4 %, 94.5 % y 96.0 % para *throughputs* de línea 24 Mb/s, 16 Mb/s y 12[Mb/s] respectivamente. Al evaluar el desempeño de los controladores PD usando la referencia resultante de la predicción de la tasa de envío disponible se obtuvo que se logran utilizaciones de 95.7 %, 92.3 % y 88.2 % para 12 Mb/s, 16 Mb/s y 24 Mb/s respectivamente; al comparar esto con las utilizaciones promedio de una transmisión que se logran con algoritmo TCP Reno: 78.6 %, 77.2 % y 76.4 % correspondientemente se concluye que el algoritmo en evaluación permite una mejor utilización en cada uno de los casos y por lo tanto un mejor provecho de los recursos lo que se traduce en *throughputs* mayores y menor tiempo de transmisión en cada caso. Se pudo ver una relación entre valor del ancho de banda y utilización alcanzada; si el *throughput* de línea es mayor la utilización es menor, se desprendió de esta información y también de los gráficos que lo anterior se debe al periodo inicial del funcionamiento del algoritmo y a las características de la predicción. Se dedujo pues, que el tamaño del archivo podría influir en el grado de utilización y se construyeron 3 gráficos, uno para cada ancho de banda estudiado 12 Mb/s, 16 Mb/s y 24 Mb/s, con los promedios de *throughput* y utilización en archivos que van desde los 100 MB hasta los 1500 MB se observó de allí que para los *throughputs* de 12 Mb/s y 16 Mb/s se obtienen mejores utilizaciones y velocidades de envío en cualquiera de los tamaños de archivo respecto de lo que se obtiene con TCP Reno, aunque para los archivos más pequeños la diferencia entre ellos es menor; en el caso de un datarate de 24 Mb/s el protocolo con control PD supera al TCP Reno a partir del archivo de 200 MB.

Con el fin de determinar si los resultados de utilización obtenidos del protocolo con control PD dependen de la semilla de generación que determina el tiempo aleatorio de inicio de la transmisión entre 0.001 s y 0.01 s se hicieron 20 simulaciones para datarates 12 Mb/s, 16 Mb/s y 24 Mb/s y se calculó la media de los promedios de *throughput* y utilización y también la desviación estándar, resultó que las medias de los promedios de *throughput* y utilización son bastante semejantes a las que se obtuvieron en la prueba de comparación de desempeño del protocolo contra TCP Reno, y además que las deviaciones estándar son pequeñas: 0.54, 0.92 y 1.95 respectivamente lo que indica que los resultados y la consecuente evaluación del protocolo no son casuales, la dispersión es pequeña y los promedios de utilización serán

cercanos independiente de si existe cierto desfase en las transmisiones de los servidores.

Se realizó también la medición del grado de *TCP friendliness* del protocolo con control PD, esto se llevó a cabo en un escenario con cuatro servidores donde dos de ellos emplean el protocolo en evaluación y los otros dos TCP Reno se realizó una prueba donde todos los servidores transmiten al mismo tiempo de esto se obtuvo que los servidores que emplean el algoritmo con control PD logran alcanzar un promedio de ventana mayor que el de los servidores que transmiten, aproximadamente 76000 B contra 63000 B; al realizar una simulación con todos los servidores con TCP Reno, se ve que todos los servidores tienen ventanas casi del mismo tamaño 50000 B aproximadamente, sin embargo , esto es mucho menor que lo que se obtiene cuando se simuló usando el protocolo con control PD en dos servidores y el protocolo clásico en los otros dos, de allí se desprende, y también de un experimento semejante pero donde las transmisiones incluyen desfases, que el protocolo analizado no sólo es amigable con TCP Reno permitiéndole incrementar rápidamente su ventana hasta llegar a niveles justos y adecuados de transmisión sino que además favorece a los servidores con TCP Reno permitiéndoles llegar a velocidades de transmisión incluso mayores gracias al mejor provecho de los recursos que determina.

También se implementó un protocolo con técnicas de control difuso, se logró sintonizar los parámetros del controlador que determinan las funciones de pertenencia mediante un algoritmo de optimización PSO. La referencia teórica de tasa de envío disponible usada en la sintonización contiene una variación a diferencia de la sintonización del controlador clásico en principio es 24 Mb/s y 12 Mb/s. Se hicieron pruebas de desempeño para dos bases de reglas diferentes en una AI y MD son valores independientes entre sí y en la otra tiene una relación de dependencia como se establece en el trabajo sobre GAIMD [1]. Las pruebas realizadas involucraban comienzo en desfase de las transmisiones de los servidores y variación de la tasa de envío dispoible de la línea pasando por los niveles 24 Mb/s, 16 Mb/s y 12 Mb/s se obtuvo una utilización promedio en toda la simulación de 87.2 % y 86.6 % para AI y MD dependientes e independientes respectivamente. Un experimento con los mismos desfases y TCP Reno en cada servidor obtiene 64.6 % como promedio de utilización, de esto se desprende que los protocolos con control difuso permiten lograr una mejor utilización y *throughput*, además cumplen con *TCP friendliness* pues las ventanas de los servidores 3 y 4 llegan a tamaños cercanos a 50000 B y 60000 B respectivamente que es cercano e incluso mayor en casos que en experimento donde todos los servidores emplean el algoritmo TCP Reno. Si se compara la mismo prueba pero con el protocolo con los controladores clásicos PD se puede notar que el resultado es muy cercano, 87 % en ambos, levemente superior por algunas décimas en este último. De aquí se puede concluir que a pesar que la sintonización del controlador difuso, haya sido más simple que exhaustiva, pues se consideraron sólo dos niveles de cambio del ancho de banda de línea se pueden obtener en todo caso resultados muy favorables, lo que permite afirmar que esto sería una ventaja de usar este tipo de controlador: una sintonización más simple e incluso intuitiva que no perjudica el resultado final.

En resumen, los protocolos testeados logran resultados mejores en *throughput* y utilización que TCP Reno en cada caso estudiado permitiendo aprovechar mejor los recursos y por lo tanto menores tiempos de transmisión; al compartir el canal con servidores con TCP Reno tienen un comportamiento *TCP friendly* favoreciendo inclusive en casos el desempeño de estos.

Respecto de trabajos futuros se propone realizar las mismas pruebas y experimentos en un entorno más real, es decir, en una red de computadores para validar el resultado. Además sería interesante plantear el diseño de un protocolo que pueda realizar sintonización *on line* o métodos adaptativos para asegurar el buen funcionamiento en caso de cambios en las condiciones de la red. Por otro lado, también podría estudiarse una versión que TCP que utilizara técnicas del control óptimo.

Bibliografía

[1] C. Williamson, «Internet traffic measurement», *IEEE Internet Computing*, vol. 5, no. 6, pp. 70–74, 2001.

[2] K. Thompson, G. J. Miller, and R. Wilder, «Wide-area internet traffic patterns and characteristics», *IEEE network*, vol. 11, no. 6, pp. 10–23, 1997.

[3] C. Fraleigh, S. Moon, B. Lyles, *et al.*, «Packet-level traffic measurements from the sprint ip backbone», *IEEE network*, vol. 17, no. 6, pp. 6–16, 2003.

[4] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, «Modeling tcp throughput: A simple model and its empirical validation», *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 303–314, 1998.

[5] G. Paliwal, K. P. Sharma, S. Taterh, and S. Varshney, «A new effective tcp-cc algorithm performance analysis (ns3)», in *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, IEEE, 2019, pp. 630–635.

[6] N. Oguchi and S. Abe, «Reconfigurable tcp: An architecture for enhanced communication performance in mobile cloud services», in *2011 IEEE/IPSJ International Symposium on Applications and the Internet*, IEEE, 2011, pp. 242–245.

[7] T. Kelly, «Scalable tcp: Improving performance in highspeed wide area networks», *ACM SIGCOMM computer communication Review*, vol. 33, no. 2, pp. 83–91, 2003.

[8] Y. Ren, W. Yang, X. Zhou, H. Chen, and B. Liu, «A survey on tcp over mmwave», *Computer Communications*, vol. 171, pp. 80–88, 2021.

[9] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, «Internet of things: Survey and open issues of mqtt protocol», in *2017 international conference on engineering & MIS (ICEMIS)*, Ieee, 2017, pp. 1–6.

[10] V. Jacobson and R. T. Braden, «Tcp extensions for long-delay paths», Tech. Rep., 1988.

[11] J. Mahdavi, «Tcp-friendly unicast rate-based flow control», *Note sent to end2end-interest mailing list*, 1997.

[12] D.-M. Chiu and R. Jain, «Analysis of the increase and decrease algorithms for congestion avoidance in computer networks», *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[13] A. S. Tanenbaum, D. J. Wetherall, *et al.*, *Computer networks*, 1996.

[14] J. Kurose and K. Ross, *Computer networks: A top down approach featuring the internet*, 2010.

[15] Y. R. Yang and S. S. Lam, «General aimd congestion control», in *Network Protocols, 2000. Proceedings. 2000 International Conference on*, IEEE, 2000, pp. 187–198.

[16] I. A. Qazi and T. Znati, «On the design of load factor based congestion control protocols for next-generation networks», *Computer networks*, vol. 55, no. 1, pp. 45–60, 2011.

[17] J. Godjevac, «Comparison between pid and fuzzy control», *Ecole Polytechnique Federale de Lausanne, Departement d'Informatique, Laboratoire de Microinformatique, Internal Report*, vol. 93, 1993.

[18] J. Kennedy, «The particle swarm: Social adaptation of knowledge», in *Evolutionary Computation, 1997., IEEE International Conference on*, IEEE, 1997, pp. 303–308.

[19] J. Kennedy, «Swarm intelligence», in *Handbook of nature-inspired and innovative computing*, Springer, 2006, pp. 187–219.

[20] H. Jung, S.-g. Kim, H. Y. Yeom, S. Kang, and L. Libman, «Adaptive delay-based congestion control for high bandwidth-delay product networks», in *2011 Proceedings IEEE INFOCOM*, IEEE, 2011, pp. 2885–2893.

[21] C. Estevez, «Transporting the cloud», in *Mobile Networks and Cloud Computing Convergence for Progressive Services and Applications*, IGI Global, 2014, pp. 135–156.

[22] D. Katabi, M. Handley, and C. Rohrs, «Congestion control for high bandwidth-delay product networks», in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, 2002, pp. 89–102.

[23] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, «One more bit is enough», *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 37–48, 2005.

[24] X. Huang, C. Lin, F. Ren, G. Yang, P. D. Ungsunan, and Y. Wang, «Improving the convergence and stability of congestion control algorithm», in *2007 IEEE International Conference on Network Protocols*, IEEE, 2007, pp. 206–215.

[25] R. Jain, «A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks», *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 5, pp. 56–71, 1989.

[26] C. Jin, D. X. Wei, and S. H. Low, «Fast tcp: Motivation, architecture, algorithms, performance», in *IEEE INFOCOM 2004*, IEEE, vol. 4, 2004, pp. 2490–2501.

[27] S. Floyd, «Highspeed tcp for large congestion windows», Tech. Rep., 2003.

[28] D. Leith and R. Shorten, «H-tcp: Tcp for high-speed and long-distance networks», in *Proceedings of PFLDnet*, vol. 2004, 2004.

[29] L. Xu, K. Harfoush, and I. Rhee, *Binary increase congestion control for fast long-distance networks ieee infocom*, 2004.

[30] S. Ha, I. Rhee, and L. Xu, «Cubic: A new tcp-friendly high-speed tcp variant», *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[31] S. Bhandarkar, S. Jain, A. N. Reddy, *et al.*, «Improving tcp performance in high bandwidth high rtt links using layered congestion control», *PFLDNet05*, 2005.

[32] S. Cen, J. Walpole, and C. Pu, «Flow and congestion control for internet media streaming applications», in *Multimedia Computing and Networking 1998*, International Society for Optics and Photonics, vol. 3310, 1997, pp. 250–264.

[33] S. Jacobs and A. Eleftheriadis, «Providing video services over networks without quality of service guarantees», in *World Wide Web Consortium Workshop on Real-Time Multimedia and the Web*, 1996.

[34] D. Sisalem and H. Schulzrinne, «The loss-delay based adjustment algorithm: A tcp-friendly adaptation scheme», in *Proceedings of NOSSDAV*, Citeseer, vol. 98, 1998, pp. 215–226.

[35] R. Rejaie, M. Handley, and D. Estrin, «Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet», in *IEEE INFOCOM'99. Confer-*

ence on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320), IEEE, vol. 3, 1999, pp. 1337–1345.

[36] V Ozdemir and I Rhee, *Tcp emulation at the receivers (tear), presentation at the rm meeting*, 1999.

[37] T. Turletti, S. F. Parisis, and J.-C. Bolot, «Experiments with a layered transmission scheme over the internet», Tech. Rep., 1997.

[38] W.-T. Tan and A. Zakhor, «Real-time internet video using error resilient scalable compression and tcp-friendly transport protocol», *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 172–186, 1999.

[39] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, «A model based tcp-friendly rate control protocol», in *Proceedings of NOSSDAV'99*, Citeseer, 1999.

[40] S. Floyd, M. Handley, J. Padhye, and J. Widmer, «Equation-based congestion control for unicast applications», *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 43–56, 2000.

[41] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, «The macroscopic behavior of the tcp congestion avoidance algorithm», *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[42] W. R. Stevens, «Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms», 1997.

[43] R. Braden, «Rfc-1122: Requirements for internet hosts», *Request for Comments*, pp. 356–363, 1989.

[44] V. Jacobson, «Congestion avoidance and control», in *ACM SIGCOMM computer communication review*, ACM, vol. 18, 1988, pp. 314–329.

[45] W. R. Stevens and G. R. Wright, *TCP/IP illustrated, volume 2*. Addison-wesley, 1996.

[46] K. Fall and S. Floyd, «Simulation-based comparisons of tahoe, reno and sack tcp», *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[47] S. Floyd and T. Henderson, «Rfc 2582», *The NewReno Modification to TCP's Fast Recovery Algorithm*, 1999.

[48] J. C. Hoe, «Start-up dynamics of tcp's congestion control and avoidance schemes», Ph.D. dissertation, Massachusetts Institute of Technology, 1995.

[49] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, «Tcp selective acknowledgment options», Tech. Rep., 1996.

[50] E. Blanton, M. Allman, L. Wang, I. Jarvinen, M. Kojo, and Y. Nishida, «A conservative loss recovery algorithm based on selective acknowledgment (sack) for tcp», Tech. Rep., 2012.

[51] D. D. Clark, M. L. Lambert, and L. Zhang, «Netblt: A bulk data transfer protocol», Tech. Rep., 1987.

[52] W. T. Strayer, B. J. Dempsey, and A. C. Weaver, *XTP: The Xpress transfer protocol*. Addison Wesley Longman Publishing Co., Inc., 1992.

[53] D. Velten, R. Hinden, and J. Sax, «Reliable data protocol», Tech. Rep., 1984.

[54] C. Huitema and I. Valet, «An experiment on high speed file transfer using satellite links», in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 11, 1981, pp. 254–257.

[55] D. R. Cheriton, «Vmtp: Versatile message transaction protocol: Protocol specification», Tech. Rep., 1988.

[56] J. Postel, «Transmission control protocol», 1981.

[57] K. Thompson, G. J. Miller, and R. Wilder, «Wide-area internet traffic patterns and characteristics», *IEEE network*, vol. 11, no. 6, pp. 10–23, 1997.
[58] M. Gerla and L. Kleinrock, «Flow control: A comparative survey», *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 553–574, 1980.
[59] S. M. Ross, *Introduction to probability models*. Academic press, 2014.
[60] P. P. Mishra, D. Sanghi, and S. K. Tripathi, «Tcp flow control in lossy networks: Analysis and enhancement.», in *NETWORKS*, 1992, pp. 181–192.
[61] T. Lakshman and U. Madhow, «The performance of tcp/ip for networks with high bandwidth-delay products and random loss», *IEEE/ACM Transactions on Networking (ToN)*, vol. 5, no. 3, pp. 336–350, 1997.
[62] A. Kumar, «Comparative performance analysis of versions of tcp in a local network with a lossy link», *IEEE/ACM Transactions on Networking (ToN)*, vol. 6, no. 4, pp. 485–498, 1998.
[63] A. Bakre and B. Badrinath, «I-tcp: Indirect tcp for mobile hosts», in *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, IEEE, 1995, pp. 136–143.
[64] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, «A comparison of mechanisms for improving tcp performance over wireless links», *IEEE/ACM transactions on networking*, vol. 5, no. 6, pp. 756–769, 1997.
[65] R. Caceres and L. Iftode, «Improving the performance of reliable transport protocols in mobile computing environments», *IEEE journal on selected areas in communications*, vol. 13, no. 5, pp. 850–857, 1995.
[66] V. K. Josyula, R. B. Bunt, and J. J. Harms, «Measurements of tcp performance over wireless connections», in *Proc. WIRELESS*, vol. 96, 1996.
[67] V. Misra, W.-B. Gong, and D. Towsley, «Stochastic differential equation modeling and analysis of tcp-windowsize behavior», in *Proceedings of PERFORMANCE*, vol. 99, 1999.
[68] V. Misra, W.-B. Gong, and D. Towsley, «Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red», in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 30, 2000, pp. 151–160.
[69] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong, «A control theoretic analysis of red», in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, IEEE, vol. 3, 2001, pp. 1510–1519.
[70] R. Brockett, «Stochastic control», *Lecture Notes, Harvard University*, 2009.
[71] A. Y. Khinchin, D. Andrews, and M. Quenouille, *Mathematical methods in the theory of queuing*. Courier Corporation, 2013.
[72] S. Floyd and V. Jacobson, «Random early detection gateways for congestion avoidance», *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 4, pp. 397–413, 1993.
[73] D. D. Clark and W. Fang, «Explicit allocation of best-effort packet delivery service», *IEEE/ACM Transactions on networking*, vol. 6, no. 4, pp. 362–373, 1998.
[74] V. Firoiu and M. Borden, «A study of active queue management for congestion control», in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, IEEE, vol. 3, 2000, pp. 1435–1444.
[75] S. Mascolo, «Congestion control in high-speed communication networks using the smith principle», *Automatica*, vol. 35, no. 12, pp. 1921–1935, 1999.

[76] R. Jain, K. Ramakrishnan, and D.-M. Chiu, «Congestion avoidance in computer networks with a connectionless network layer», *arXiv preprint cs/9809094*, 1998.

[77] J. Majithia, M. Irland, J. Grange, N. Cohen, and C. O'Donnell, «Experiments in congestion control techniques», in *Proc. Int. Symp. Flow Control Computer Networks, Versailles, France*, 1979, pp. 211–234.

[78] R. Jain, «A timeout-based congestion control scheme for window flow-controlled networks», *IEEE Journal on Selected Areas in Communications*, vol. 4, no. 7, pp. 1162–1167, 1986.

[79] K. Ramakrishnan, «Analysis of a dynamic window congestion control protocol in heterogeneous enviromments including satellite links», in *Proc. Computer Network Symposium, 1986*, 1986, pp. 94–101.

[80] S. Keshav, «A control-theoretic approach to flow control», in *Proceedings of the conference on Communications architecture & protocols*, 1991, pp. 3–15.

[81] S. Singh, A. K. Agrawala, and S. Keshav, *Deterministic analysis of flow and congestion control policies in virtual circuits*. University of Maryland, 1990.

[82] S. Keshav, A. K. Agrawala, and S. Singh, *Design and analysis of a flow control algorithm for a network of rate allocating servers*. University of Maryland, 1990.

[83] B. D. Anderson and J. B. Moore, *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

[84] G. Vinnicombe, «On the stability of networks operating tcp-like congestion control», *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 217–222, 2002.

[85] T. Kelly, *On engineering a stable and scalable TCP variant*. University of Cambridge, Department of Engineering, 2002.

[86] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*, 4. ACM, 1994, vol. 24.

[87] C. Estevez, S. Angulo, A. Abujatum, G. Ellinas, C. Liu, and G.-K. Chang, «A carrier-ethernet oriented transport protocol with a novel congestion control and qos integration: Analytical, simulated and experimental validation», in *2012 IEEE International Conference on Communications (ICC)*, IEEE, 2012, pp. 2673–2678.

[88] A. Abadleh, A. Tareef, A. Btoush, *et al.*, «Comparative analysis of tcp congestion control methods», in *2022 13th International Conference on Information and Communication Systems (ICICS)*, 2022, pp. 474–478. DOI: 10.1109/ICICS55353.2022.9811217.

[89] R. Al-Saadi, G. Armitage, J. But, and P. Branch, «A survey of delay-based and hybrid tcp congestion control algorithms», *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3609–3638, 2019.

[90] W. Na, B. Bae, S. Cho, and N. Kim, «Dl-tcp: Deep learning-based transmission control protocol for disaster 5g mmwave networks», *IEEE Access*, vol. 7, pp. 145 134–145 144, 2019.

[91] M. R. Kanagarathinam, S. Singh, I. Sandeep, *et al.*, «Nexgen d-tcp: Next generation dynamic tcp congestion control algorithm», *IEEE Access*, vol. 8, pp. 164 482–164 496, 2020.

[92] Y. Wang, L. Wang, and X. Dong, «An intelligent tcp congestion control method based on deep q network», *Future Internet*, vol. 13, no. 10, p. 261, 2021.

[93] W. Li, S. Gao, X. Li, Y. Xu, and S. Lu, «Tcp-neuroc: Neural adaptive tcp congestion control with online changepoint detection», *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2461–2475, 2021.

[94] K. Han, J. Y. Lee, and B. C. Kim, «Machine-learning based loss discrimination algorithm for wireless tcp congestion control», in *2019 international conference on electronics, information, and communication (ICEIC)*, IEEE, 2019, pp. 1–2.

[95] T. Toprasert and W. Lilakiataskun, «Tcp congestion control with mdp algorithm for iot over heterogeneous network», in *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, IEEE, 2017, pp. 1–5.

[96] S Floyd, «Rfc 5348 tcp-friendly rate control (tfrc) protocol specification», *RFC 5348 Proposed Standard*, 2008.

Anexos

Anexo A

A.1. Modelo OSI

El modelo Sistema de Interconexión Abierto (*Open System Interconnection*, OSI) se denomina así porque aplica a los sistemas que están abiertos a la comunicación con otros sistemas. El modelo está compuesto por siete capas. Cada capa representa una abstracción diferente y cumple una función distinta de las restantes de manera que se minimice la información que fluye a través de cada una. Estas capas son: La capa física se ocupa de la transmisión de bits a través de un canal de comunicación. Los aspectos considerados al momento de realizar el diseño en esta capa son las interfaces mecánicas, eléctricas y de temporización además del medio a utilizar en la transmisión.

La capa de enlace de datos trasforma el medio de transmisión en una línea de comunicación, su objetivo es evitar los errores de transmisión en la capa red. Trabaja fragmentando los datos y transmitiendo secuencialmente, el receptor confirma la recepción correcta devolviendo datos. Aquí además se regula el tráfico con el fin de evitar saturaciones.

La capa red controla las funciones de la subred. Determina cómo enrutar los datos desde origen a destino también controla la congestión aunque esta operación no es exclusiva de esta capa sino que puede ser una labor compartida con la capa de transporte. En otras palabras, esta se encarga de aspectos de la calidad de servicio como retardo, tiempo de tránsito, inestabilidad, etc. La capa de transporte recibe los datos que vienen de las capas superiores los divide y traspasa a la capa de red de modo que lleguen todos los fragmentos a destino en forma correcta, eficiente y segura. Es decir, en la capa de transporte se realiza una conexión de extremo a extremo donde un programa máquina conversa con otro parecido.

La capa de sesión permite que establezcan sesiones los usuarios de máquinas distintas. Las

sesiones ofrecen varios servicios como el control de diálogo, es decir, determinar a quién le toca transmitir, esto puede ser simultáneo full-dúplex en ambos sentidos o alternado half-dúplex en las dos direcciones, además estas se ocupan de administrar el token y de la sincronización.

La capa de presentación se encarga de la sintaxis y semántica de la información enviada. Aquí se estadariza la codificación y representación de las estructuras de datos que se intercambiarán para que pueda existir comunicación efectiva entre dos computadoras.

La capa de aplicación permite acceder a los protocolos que utilizan frecuentemente. Entre estos se cuentan aquellos que se encargan de la transferencia de archivos, envío de correo electrónico acceso a páginas web, etc. Este último es conocido como protocolo de transferencia de hipertexto (*Hipertext Transfer Protocol*, HTTP) que es la base de *World Wide Web*, este protocolo se usa para comunicarse con el servidor respectivo cuando el navegador intenta ingresar a una página web.

A.2. Modelo TCP/IP

Es una arquitectura que al ser diseñada tenía como objetivo evitar los cortes de comunicación en aquellos casos en que hubiese pérdida hardware de la subred, es decir, cuando alguna máquina quedase fuera de servicio. Esta estructura consta de varias capas: la capa de interred es no orientada a la conexión, aquí se define el formato de un paquete y el conocido como Protocolo de Internet (*Internet Protocol*, IP). Su función más importante es el enrutamiento y entrega de los paquetes al receptor a fin de evitar la congestión.

La capa de transporte permite que las entidades iguales en los host de origen y destino "conversen". A este nivel se han definido dos protocolos de transporte: Protocolo de Control de Transmisión (*Transport Control Protocol*, TCP) y Protocolo de Datagrama de Usuario (*User Datagram Protocol*, UDP). TCP es un protocolo orientado a la conexión que cuya misión es transmitir bytes de información que deben llegar sin errores al hardware de destino, también se ocupa de la fragmentación de los datos y maneja el control de flujo con el fin de evitar saturaciones entre emisores y receptores que trabajan a diferente velocidad. UDP es un protocolo no confiable y no orientado a la conexión para aplicaciones que no necesitan control de flujo; se usa, por ejemplo, en transmisión de voz y vídeo.

La capa de aplicación contiene los protocolos de más alto nivel. De los más antiguos se pueden nombrar terminal virtual (Telecommunication Network, TELNET), transferencia de archivo (*File Transfer Protocol*, FTP) y correo electrónico o Protocolo para la Transferencia Simple de Correo (*Simple, Mail Transfer Protocol*, SMTP). Posteriormente se añadieron otros como Sistemas de nombres de dominio (*Domain Name System*, DNS) para la resolución de nombres de host; *Network News Transport* Protocol (NNTP) para transportar artículos de noticias, Hypertext Transfer Protocol (HTTP) para páginas *World Wide Web*, etc. La capa de host, el modelo TCP/IP indica que el host se debe conectar a la red mediante el mismo protocolo para que le puedan enviar paquetes IP- Este protocolo no está definido y varía de un host a otro.

A.3. Implementación

A.3.1. PSO

El código PSO en que se basa este trabajo fue codificado en Matlab por Andrea Cirillo y se encuentra disponible en la url: `https://www.mathworks.com/matlabcentral/fileexchange/30660-simple-example-of-pso-algorithm`.

Este fue adaptado o modificado para poder ser usado en Opnet con integración a Matlab. Fue necesario dividirlo en partes para ejecutarlo secuencialmente con las simulaciones en Opnet.

A.3.2. Opnet

Los archivos de la implementación en Opnet se encuentran disponibles en `https://github.com/she-rand/tesis-imp-opnet`

A.3.3. Matlab

El código en Matlab desarrollado para la implementación de las simulaciones se encuentra disponible en `https://github.com/she-rand/tesis-imp-matlab`

Anexo B

Definiciones

acknowledgement acuse de recibo de un paquete.

additive increase incremento aditivo, se refiere a la forma en que se incrementa la ventana durante la etapa de congestion avoidance.

bandwidth delay product es el producto entre el valor del ancho de banda y el retardo en una red.

congestion avoidance evitación de congestión, es un algoritmo de TCP que se activa cuando se detecta congestión y que incrementa el valor de la ventana en uno y la decrementa a la mitad.

congestion control control de congestión.

congestion window ventana de congestión.

delay retardo entre la emisión y recepción.

fast recovery rápida recuperacion, es un algoritmo de TCP que permite una rápida recuperación inflando la ventana de congestión por cada nuevo paquete recibido .

fast retransmit rápida retransmisión, es un algoritmo de TCP que permite retransmitir sin esperar un timeout.

multiplicative decrease decremento multiplicativo se refiere al factor en que se disminuye la ventana cuando ocurre congestión, esto es a la mitad.

particle swarm optimization optimización por enjambre de partículas.

round trip time tiempo que demora un paquete en ir y volver al emisor.

slow start comienzo rápido, se refiere al nombre de un algoritmo de TCP con que inicia la transmisión de paquetes.

TCP-friendliness se refiere a una característica de los protocolos de congestión que se comportan amistosamente con el tráfico TCP sin acaparar el uso del ancho de banda.

throughput es la velocidad o tasa de flujo de paquetes o información en una red.

timeout es el tiempo máximo de espera a que un paquete sea recibido.

transport control protocol protocolo de control de transporte.