Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Estudio de técnicas de control de congestión con diferentes tipos de tráfico de datos en NS3

Autor:

Cantera Álvarez, Álvaro

Tutor:

Álvarez Álvarez, Mª Teresa
Ingeniería de Sistema y Automática

Valladolid, Enero y 2023.

A mis amigos y amigas

por todas las horas compartidas

a lo largo de este viaje de conocimiento,

noches en vela arreglando

el mundo a través de la ciencia y el debate

haciendo que pudiera tener siempre fuerzas de seguir adelante

ayudándome en todo momento.

A mis familiares

por darme la oportunidad de estudiar

estando ahí en esas duras etapas

sin entender muy bien todo lo que hacía

pero mostrando interés en todo momento

celebrando cada nota a mi lado.

A quienes ya no están

por todos los consejos

a los que recurro en el recuerdo.

A mi tutora Teresa

por toda su paciencia y aportaciones

durante todo este tiempo.

Gracias por acompañarme en este camino que ahora empieza de nuevo.

Seguro que en este me caigo un poco menos, o al menos, esa es la intención.

Resumen

El crecimiento de los elementos IOT (*Internet Of Things)* tanto en la electrónica de consumo como en entornos industriales tienen como consecuencia una saturación de las redes pudiendo llegar a producir la congestión de las mismas con mayor frecuencia. En los últimos años se han desarrollado unas nuevas variantes de TCP con grandes resultados al implementarse en ciertos sistemas. Este trabajo pretende hacer una comparativa entre varias de esas variantes de TCP, junto con diferentes algoritmos de gestión de colas, particularizando en los AQM sobre el simulador NS3.

Palabras clave

NS3, BBR, AQM, Fq-Codel.

Abstract

The growth of IOT (*Internet Of Thing*s*)* elements both in consumer electronics and in industrial environments has as a consequence a saturation of the networks, being able to produce their congestion more frequently. In recent years, new variants of TCP have been developed with great results when implemented in certain systems. This work aims to make a comparison between several of these variant of TCP, together with different queue management algorithms, with particular emphasis on AQM on the NS3 simulator.

Keywords

NS3, BBR, AQM, Fq-Codel.

ÍNDICE:

Índice de Figuras

Índice de Tablas

1. Introducción y justificación del proyecto

1.1. Introducción y justificación

La congestión de redes es un problema poco conocido, pero muy presente en el día a día de las personas. En los últimos años con el crecimiento del volumen de datos transmitidos por redes tanto inalámbricas como cableadas se producen de forma más habitual situaciones no deseadas que afectan al buen funcionamiento de las comunicaciones. La transmisión de datos no se hace de forma constante, por lo que la red se puede llegar a saturar por la alta demanda de datos a transmitir. Ejemplos de estas situaciones serían por ejemplo la descarga masiva de datos de una simulación en un centro de investigación, como la conectividad de diferentes vehículos en movimiento o el visionado de un espectáculo en tiempo real emitido a todo el mundo mediante una plataforma de video.

Desde que este problema apareciera por primera vez en noviembre de 1988 hasta algoritmos con una implementación bastante eficiente como el planteado por Google en 2016, se han desarrollado muchas alternativas y cada vez más específicas para diferentes características de transmisión de datos para solventar este problema. En este trabajo se realizará una comparación de diferentes algoritmos AQM (*Active Queue Management*) frente a números diferentes de tráfico de datos y diferentes variantes de TCP (*Transmission Control Protocol*), analizar su comportamiento en un tipo de topología de red. Finalmente, se concluirá con una reflexión sobre la idoneidad de estos sistemas con el fin de lograr la implantación en entornos industriales.

Dentro de la terminología empleada en este trabajo, se observarán el uso de adjetivos calificativos tales como “bueno”, “malo”, “grande”, “pequeño”, etc. Con el fin de valorar algunas variables o elementos dentro del trabajo. Estos adjetivos han sido escogidos bajo el criterio de análisis global del proyecto.

1.2. Objetivo del proyecto

El objetivo de este proyecto es estudiar el comportamiento de diferentes variantes de TCP, haciendo hincapié en el tipo BBR (*Bottleneck Bandwidth and Round-trip propagation Time)* con diferentes características de red y diferentes algortimos AQM. Finalmente, se concluirá con una comparación entre los resultados obtenidos.

1.3. Organización de la memoria

La información en este documento está estructurada de la siguiente forma; Se han separado 7 capítulos y 2 Anexos. Siendo los capítulos, Introducción y justificación, donde se hará una presentación de lo que se va a tratar en este trabajo así como las motivaciones del mismo; Un segundo capítulo con el marco teórico donde se darán una serie de definiciones de los principales conceptos teóricos que irán apareciendo en las próximas páginas, dando unas nociones básicas para la correcta comprensión de lo explicado.

Un tercer capítulo con una aproximación teórica donde se explicará el funcionamiento de las principales variantes de TCP y de aquellas que se han utilizado en este trabajo. En el apartado 4 se realiza la misma aproximación teórica y de funcionamiento que en el apartado 3 pero ahora centrado en los algoritmos de gestión de colas. En el quinto apartado se hará un análisis de las simulaciones realizadas y presentes en el Anexo 1 explicando la información obtenida de las mismas y justificando su comportamiento. El apartado 6 recoge de forma más concreta y precisa las conclusiones mostradas en el apartado 5, sugiriendo unas posibles futuras líneas de trabajo. Finalmente en el último apartado se añade la bibliografía citada a lo largo del texto en formato APA 7ª edición.

Concluyendo con el documento se encuentran dos Anexos. El primero recoge los resultados gráficos de las simulaciones consideradas interesantes para este trabajo y que se han seleccionado para ser recogida aquí.

Para la realización de este trabajo y el análisis del comportamiento de los algoritmos dentro del simulador se han realizado más de 200 simulaciones.

Finalmente se añaden los códigos base para poder replicar las simulaciones a fin de auditar los resultados de este TFG. Si por alguna razón el código a la hora de ejecutarse diera algún problema por posibles futuras actualizaciones del simulador o del entorno donde se ejecuta, se insta a ponerse en contacto con el autor del trabajo para la resolución de los problemas.

2. Marco teórico

2.1. Estado del arte

En octubre de 1986 entre Lawrence Berkeley Laboratory y University of California at Berkeley estaban realizando pruebas de comunicación entre computadores mediante red cableada a 400 yardas (365,76 metros) de distancia. En estas pruebas se produjo un comportamiento que no esperaban en la red, la velocidad de transmisión se redujo de forma significativa por causas desconocidas.

Inicialmente se había asumido que en las redes de comunicación se cumplía el principio de conservación de masa, tomando como unidad básica de masa un paquete de datos. Por las características que tiene el paquete es "incompresible" y no puede variar su tamaño, además los lugares de intercambio de paquetes del contorno del balance están restringidos a los ordenadores y no puede haber ninguno más ni ninguno menos del que se haya enviado. Asumiendo esto y teniendo en cuenta la definición de ancho de banda expresada anteriormente, se observó que el ancho de banda paso de 32 (k bits)/segundo a 40 bits/segundo. Tras un análisis de la red se determino que se había producido una congestión de la red y se plantearon diversos algoritmos.

En esta temprana etapa del desarrollo tecnológico de las redes informáticas la comunicación se realizaba punto a punto, no se tardó en plantearse nuevas conexiones que fueran multipunto y que tuvieran nodos intermedios entre el ordenador que iniciaba la comunicación y final.

En estos nodos intermedios, que solían y suelen estar conectados en multipunto con otros ordenadores y nodos, se produce también congestión de red.

Una aproximación a algunos de los desarrollos en este campo son los siguientes:

En 1988 se publica por primera vez congestión en una red (Jacobson, 1995).

En 1990 se desarrollan TCP Tahoe (Fall & Floyd, 1996) y TCP Reno (Khan & Butt, 2018)

En 1993 se desarrolla el algoritmo AQM RED (Floyd & Jacobson, 1993)

En 1999 se desarrolla TCP New Reno (Song et al., 2021)

En 2001 se desarrolla TCP Westwood (Casetti et al., 2002)

En 2006 se publica TCP Compound en Microsoft (Tan et al., 2006).

En 2004 se desarrolla TCP Hybla(Caini & Firrincieli, 2004) y TCP BIC (Xu et al., 2004).

En 2008 se desarrolla TCP CuBIC (Ha et al., 2008).

En 2008 se presenta en el uso de control predictivo para la gestión de colas. (Alvarez et al., 2008).

En 2016 se presenta en el IETF BBRv1 desarrollado por Google(*Draft-Cardwell-Iccrg-Bbr-Congestion-Control-00*, n.d.).

En 2017 se presenta en el IETF el control de colas mediante PI(Pan et al., 2017)

En 2017 se presenta en el IETF DataCenter TCP desarrollado por Microsoft (Alizadeh et al., 2010)

2.2. Definiciones y conceptos teóricos

En este apartado se hará un breve resumen de los principales conceptos empleados en este trabajo, los cuales serán necesarios para una completa comprensión del mismo.

Paquete (Packet): Unidad básica de información en la capa de transporte

Ancho de banda (Bandwith): Flujo de paquetes en una conexión.

Conexión punto a punto: Topología de comunicación entre dos dispositivos mediante una única conexión entre ellos.

Conexión Multipunto: Tipo de conexión que se realiza entre un dispositivo (a) y otros n dispositivos (b) de forma que cada conexión entre a y b_n se realiza puno a punto.

Cola de paquetes: Secuencia de elementos que permite las operaciones de inserción y extracción con un tamaño máximo fijo o variable.

AQM (*Active Queue Management)*: Algoritmo que permite la gestión del número de paquetes en una cola, actuando sobre diferentes variables específicas de cada algoritmo.

RTT (Round-Trip-Time): Es el tiempo que transcurre desde el instante en que un emisor envía un paquete y el instante en que el emisor recibe el paquete de confirmación (ack) por parte del proveedor remoto.

Algoritmo de planificación de paquetes: Mecanismo para elegir a que cola se asigna un paquete según diferentes criterios.

DDR: Algoritmo de planificación Déficit Round Robin.(Shreedhar & Varghese, 1996)

Flujo: Se identifica como una 5-upla cuya información almacenada es la dirección ip de origen, la dirección ip de destino, el número de puerto de origen, número de puerto de destino y el número de protocolo TCP.

Quantum: Máxima cantidad de bytes que pueden ser desencoladas a la vez de una cola.

Hash: Valor numérico generado por un algoritmo criptográfico matemático a partir de unos valores concretos de entrada.

Tiempo de reenvío (*Retransmission TimeOut, RTO):* Tiempo que transcurre desde que se envía un paquete, no se recibe el ACK correspondiente y se vuelve a enviar ese mismo paquete.

Tiempo de residencia (*Sojourn time):* Cantidad de tiempo que un paquete pasa en una cola determinada. Desde el momento en que se introduce en la cola hasta aquel que pasa hasta que se desencola para su transmisión o para su descarte.

BDP (*Banwidth-delay product, Pipe Size):* Es el producto de la multiplicación del ancho de banda en el enlace que es cuello de botella, y, el RTT.

MIMO (*Multi input-Multi-Output):* Tipo de controlador donde tiene múltiples entradas y múltiples salidas, actuando en todas a la vez.

ECN (*Explicit Congestion Notification):* Dos bits que se incorporan en la cabecera de IP (bit 6 y 7) y otros dos que se incorporan en la cabecera TCP (bit 8 y bit 9) (Ramakrishnan et al., 2001)

Topología de red: Representación gráfica de la interconexión y disposición física de diferentes elementos en una red

TCP (*Transmission Control Protocol)* (*RFC 793 - Transmission Control Protocol*, n.d.): A continuación se detalla la pila y en que posición se enc

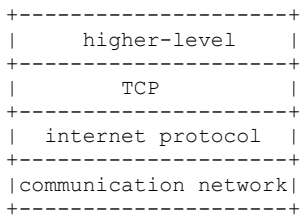

Figura 1 Capas de la pila TCP/IP (*RFC 793 - Transmission Control Protocol*, n.d.)

2.3. Descripción de la problemática con la congestión de red

Supongamos que tenemos varios ordenadores en una casa que están conectados a un router mediante cable. Estos ordenadores tienen una velocidad de transmisión máxima de 100Mbps cada uno, pero en ese momento la conexión a internet de la vivienda es de 100Mbps, pero los dos ordenadores si utilizaran la máxima capacidad del cable necesitarían 200Mbps. En ese momento, casualmente, desde los dos ordenadores se empiezan a subir los videos de la última cena navideña familiar a un servicio de alojamiento en la nube que tienen contratado. Como el ancho de banda disponible en el router es inferior a la suma de los anchos de banda disponibles en cada punto, lo que ocurrirá es que los dos ordenadores intentarán transmitir a la máxima velocidad que su enlace punto a punto con el router. Toda la información de la cena de navidad de los respectivos ordenadores se almacena en unas colas hasta el momento que se pueden transmitir a través del router. A esto se le llama una red congestionada. Siendo la principal característica de esta que la suma de las velocidades de transmisión de los ordenadores es superior a la mínima velocidad de transmisión de cualquier nodo de la red. Denominando a esta transmisión, “cuello de botella”.

Volviendo a la analogía anterior. Lo que ocurrirá es que poco a poco se irán “llenando” las colas que existen en el router. Cuando estás estén llenas el router no podrá recibir más información y tendrá que “rechazar la recepción” o vaciar las colas de alguna forma.

3. Descripción de variantes de TCP

3.1. TCP Tahoe

TCP Tahoe es uno de los primeros algoritmos de control de congestión y la base en la que se basaron las versiones más recientes propuestas como Reno o New Reno. Fue propuesto por Van Jacobson y Mike Karels en 1988 y fue implementado en el sistema operativo BSD, cuyo nombre en clave era "Tahoe", ese mismo año. Tahoe se basó en el principio de "conservación de paquetes". Según este principio, existe un punto de equilibrio en el que un enlace funciona de manera estable y mantiene ese estado solo añadiendo un paquete al enlace si otro paquete ha dejado el enlace. Por lo tanto, para obtener el mejor rendimiento, se necesitan dos algoritmos para abordar estas dos fases diferentes. El primero de los algoritmos busca el punto de equilibrio, mientras que el segundo se encargarga de mantener la conexión cerca de este punto. Llamaríamos a estas dos fases Slow-Start y Congestion Avoidance, respectivamente.

Como veremos, estos conceptos se utilizarán en los mecanismos de control de congestión que se desarrollarán en los siguientes años hasta la actualidad. Estos dos algoritmos actúan sobre la cantidad de datos enviados al enlace para maximizar el ancho de banda utilizado.

Para hacerlo, se utilizan dos parámetros: el primero es el tamaño de la ventana de congestión, cwnd, que controla el número de bytes que se pueden enviar de forma consecutiva y el segundo es sstresh que indica el número de paquetes que se pueden enviar hasta que la conexión entra en el modo de evitación de congestión.

En la fase de Slow-start, mientras cwnd es menor que sstresh, por cada ACK recibido, el remitente aumenta el tamaño de la ventana en un paquete. Ese es el llamado algoritmo Slow Start que, de hecho, no es tan lento, ya que lleva a un aumento exponencial de la ventana que tiene como objetivo alcanzar el ancho de banda máximo posible en el menor tiempo posible.

Cuando se produce una pérdida, este cwnd se reestablece a un paquete.

Cuando se alcanza el valor de sstresh, la conexión entra en el modo de evitación de congestión y comienza a aumentar la ventana de manera lineal, añadiendo 1 cwin al valor actual de la ventana. Si se pierde algún paquete, el remitente restablece la ventana a su valor inicial y restablece el umbral a la mitad del valor de la ventana de congestión cuando ocurrió la pérdida. Por

ejemplo, si comenzamos la fase de "*Congestion Avoidance"* enviando diez paquetes, se produce la pérdida cuando la ventana de congestión tiene treinta y seis, el valor del umbral será dieciocho y la ventana de congestión volverá a comenzar en diez paquetes.

TCP Tahoe también utiliza otras dos estrategias diferentes para detectar pérdidas: la estimación del tiempo de reenvío (RTO) y el Fast Retransmit. La estimación del RTO tiene como objetivo minimizar el tiempo de reacción a la pérdida de un paquete ajustando el tiempo de espera a las condiciones de la red y el algoritmo Fast Retransmit le da al remitente la capacidad de repetir la transmisión de un paquete perdido sin esperar a que expire su temporizador RTO. Para estimar el RTO, TCP Tahoe utiliza tres variables: srtt (tiempo de ida y vuelta suavizado), rttvar (variación del tiempo de ida y vuelta) y el RTO propio. Se establece inicialmente en 3 segundos, pero cuando se obtiene la primera medición de RTT, los valores se actualizan de la siguiente manera: srtt = (1-a) * srtt + a * rtt rttvar = (1-b) * rttvar + b * |rtt-srtt| RTO = srtt + 4 * rttvar Donde a y b son constantes y rtt es el tiempo de ida y vuelta medido en ese momento. El algoritmo Fast Retransmit se activa cuando se reciben cuatro ACK duplicados para el mismo paquete no confirmado. En ese caso, se reenvía el paquete sin esperar al temporizador RTO. Además, se reduce el umbral sstresh a la mitad del valor actual de la ventana de congestión y se restablece la ventana de congestión a su valor inicial. TCP Tahoe fue uno de los primeros intentos de controlar la congestión en la red y, aunque tuvo algunos problemas, sentó las bases para algoritmos posteriores como TCP Reno y TCP New Reno.

3.2. New Reno

TCP NewReno(Floyd & Henderson, 1999) es una evolución del TCP Reno original. y que tiene como objetivo mantener la transmisión de ventanas llenas cuando está en modo de recuperación mejorando el algoritmo de recuperación rápida que se implementó en TCP Reno. Aborda una desventaja específica que sufre TCP Reno cuando se producen varios eventos de pérdida concatenados. En esta situación, cuando TCP Reno recibe una confirmación de algunos de los paquetes perdidos, sale de la fase de recuperación rápida y no reenvía automáticamente los paquetes restantes perdidos como debería hacerse en Fast Retransmit, sino que espera a que expire su temporizador. En lugar de eso, New Reno controla las confirmaciones recibidas y no sale de la fase de Fast Retransmit hasta que todos los paquetes perdidos hayan sido confirmados. Como hemos mencionado antes, la variable principal que utiliza TCP New Reno y la mayoría de los algoritmos modernos de gestión de congestión de TCP para controlar la congestión en un enlace es el tamaño de la ventana, es decir, el número de paquetes que podemos enviar de forma consecutiva. Al modificar este parámetro, podemos modificar el tráfico enviado a un enlace y, por tanto, reducir o aumentar la congestión en ese enlace. El tamaño óptimo de la ventana (si solo hay un flujo TCP en el enlace) sería igual al producto ancho de banda retraso (BDP). Como no conocemos a priori el ancho de banda del enlace, el objetivo en el caso de TCP New Reno y todos los demás algoritmos de control de congestión basados en pérdidas es llevar la ventana de congestión al valor más alto que permita que el enlace funcione sin sufrir pérdidas.

TCP New Reno, al igual que sus predecesores, está compuesto por cuatro algoritmos: Slow Start, Congestion Avoidance, Fast Retransmit y Fast Recovery. También se podría dividir en dos fases principales: Slow Start, donde se lleva a cabo el algoritmo del mismo nombre y Congestion Avoidance, cuando se ajusta la ventana de congestión según las directivas de Congestion Avoidance. En esta fase, también pueden funcionar los algoritmos Fast Retransmit y Fast Recovery si se produce alguna pérdida.

Los algoritmos Slow Start y Congestion Avoidance en New Reno siguen los mismos principios que las implementaciones más antiguas de Tahoe y Reno, cambiando solo algunos detalles como el tamaño de ventana predeterminado

3.3. BIC

En el control de congestión de BIC, se considera que es un problema de búsqueda en el que el sistema puede proporcionar una retroalimentación sí/no a través de la pérdida de paquetes sobre si la tasa de envío actual (o ventana) es mayor que la capacidad de la red. Se puede estimar la ventana mínima actual como el tamaño de la ventana en el que el flujo no experimenta ninguna pérdida de paquetes. Si se conoce el tamaño máximo de la ventana, podemos aplicar una técnica de búsqueda binaria para establecer el tamaño de la ventana objetivo en el punto medio entre el máximo y el mínimo. Al aumentar hacia el objetivo, si produce alguna pérdida de paquetes, la ventana actual se puede tratar como un nuevo máximo y el tamaño de la ventana reducida después de la pérdida de paquetes puede ser el nuevo mínimo. El punto medio entre estos nuevos valores se convierte en un nuevo objetivo. Dado que la red sufre pérdidas alrededor del nuevo máximo pero no lo hace alrededor del nuevo mínimo, el tamaño de la ventana objetivo debe estar en medio de los dos valores. Después de alcanzar el objetivo y si no hay pérdida de paquetes, entonces el tamaño actual de la ventana se convierte en un nuevo mínimo y se calcula un nuevo objetivo. Este proceso se repite con el mínimo y el máximo actualizados hasta que la diferencia entre el máximo y el mínimo sea inferior a un umbral preestablecido, llamado el incremento mínimo (Smin). Esta técnica se llama aumento de búsqueda binaria.

3.4. BBRv1 (Bottleneck bandwidth and round-trip propagation time)

IETF 2016 - Google

En los variantes de TCP presentados anteriormente, la forma de detección de congestión es a través de la pérdida de paquetes. Sin embargo, la pérdida de paquetes no es equivalente a la congestión. La congestión se puede considerar como una condición que ocurre cuando una ruta de red opera de manera sostenida con más datos en tránsito de lo que es el BDP (*Banwidth Delay Product)* del enrutamiento. A medida que Internet ha evolucionado, el control de congestión basado en pérdidas es cada vez más problemático. Esta situación de pérdida de paquetes ocurre cada vez más en momentos que están disociados del comienzo de la congestión como pueden ser los siguientes:

- Buffers de capacidad limitada: en los buffers de capacidad limitada, la pérdida de paquetes suele ocurrir antes de la congestión. En los enlaces de larga distancia de alta velocidad de hoy en día que utilizan switches de consumo con estos variantes de buffers, el control de congestión basado en pérdidas puede resultar en un mal rendimiento debido a que la respuesta del algoritmo es demasiado agresiva, reduciendo de forma considerable la tasa de envío ante la pérdida de paquetes, incluso si dicha pérdida es resultado de picos puntuales de tráfico (este tipo de pérdida de paquetes puede ser bastante frecuente incluso cuando el enlace está mayormente inactivo). Debido a esta dinámica, es difícil alcanzar una utilización óptima.
- Buffers de gran capacidad: en enlaces cuello de botella con buffers de gran capacidad, la congestión suele ocurrir antes de la pérdida de paquetes. En el borde de Internet de hoy en día, el control de congestión basado en pérdidas puede causar el problema de "bufferbloat" (sobrecarga del buffer), al llenar repetidamente los buffers de gran capacidad en muchos enlaces de último kilómetro y causar retrasos de cola innecesarios de hasta varios segundos.

Como se comentará posteriormente, uno de estos casos se encuentra en entornos industriales.

Este algoritmo realiza una caracterización de la red mediante el ancho de banda de los cuellos de botella y el tiempo de transmisión. Supongamos que tenemos la siguiente red (en este ejemplo se están omitiendo los retardos asociados a la computación de los paquetes dentro de los nodos):

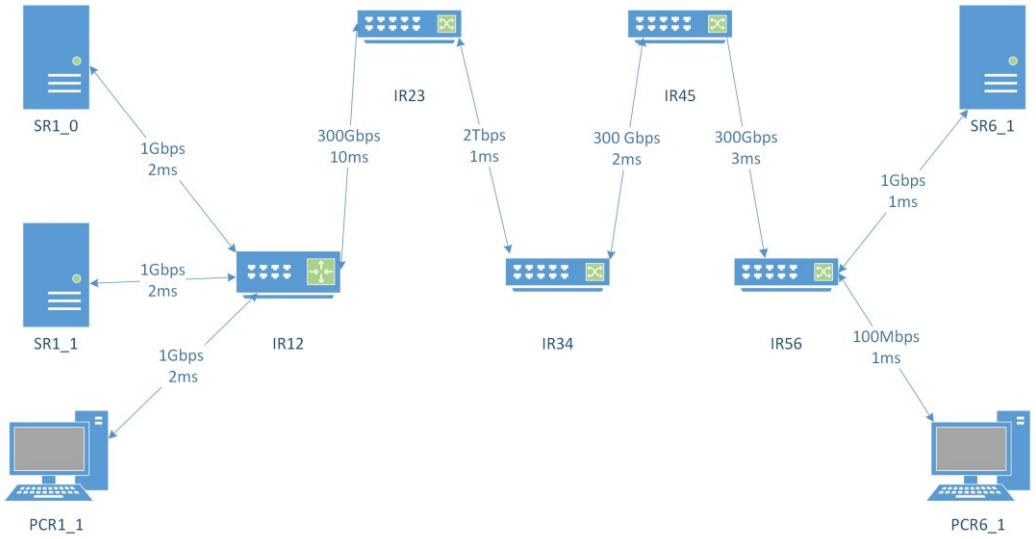

Figura 2 Topología Red Compleja BBR

En este caso queremos comunicar los ordenadores PCR1_1 y PCR6_1. Entre estos dos puntos como se puede ver hay 5 nodos intermedios. Entre cada nodo hay un tiempo de transmisión el cual representaremos a continuación.

$$\left(\begin{matrix} t_i \\ BW_i \end{matrix}\right)[=]\left[\begin{matrix} milisegundo \\ datos/_{segundo} \end{matrix}\right] \quad (1)$$

$$\left(\begin{matrix} t_i \\ BW_i \end{matrix}\right) = \left(\begin{matrix} 2 & 10 & 1 & 2 & 3 & 1 \\ 1\ Gbps & 300\ Gbps & 2\ Tbps & 300\ Gbps & 300\ Gbps & 100\ Mpbs \end{matrix}\right) \quad (2)$$

Pero, al igual que cuando queremos realizar experimentos, también podemos expresar esta red de una forma más simple, correspondiéndose con la siguiente:

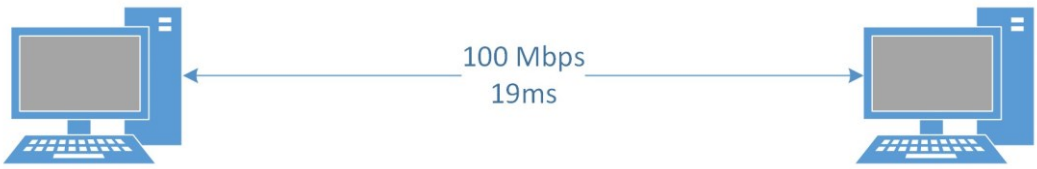

Figura 3 Topología de red simple BBR

El tiempo de transmisión ts se obtiene de la siguiente forma:

$$t_s = \sum_1^n t_i = 2 + 10 + 1 + 2 + 3 + 1 = 19\ ms \quad (3)$$

$$BW_s = \min BW_i = 100\ Mbps \quad (4)$$

El planteamiento de los desarrolladores del algoritmo es que realmente, como hemos explicado anteriormente, esperar a que haya pérdida de paquetes es una ineficiencia del sistema y cómo podemos realizar un modelo de la red de comunicación entre el punto de origen y el de destino, resulta más adecuado poder parametrizar correctamente este modelo y luego actuar sobre él. Para este ajuste el algoritmo funciona tal y como se muestra en la siguiente máquina de estados. La máquina de estados del funcionamiento sería la siguiente

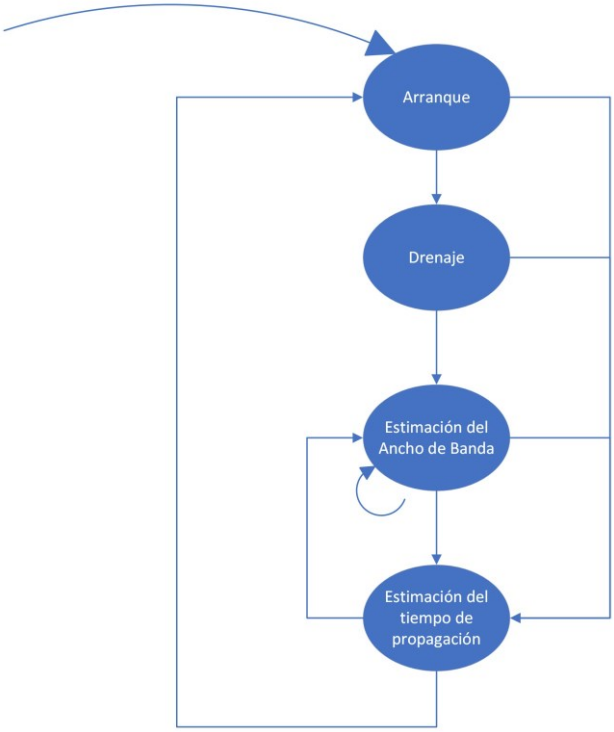

Figura 4 Diagrama de flujo BBR

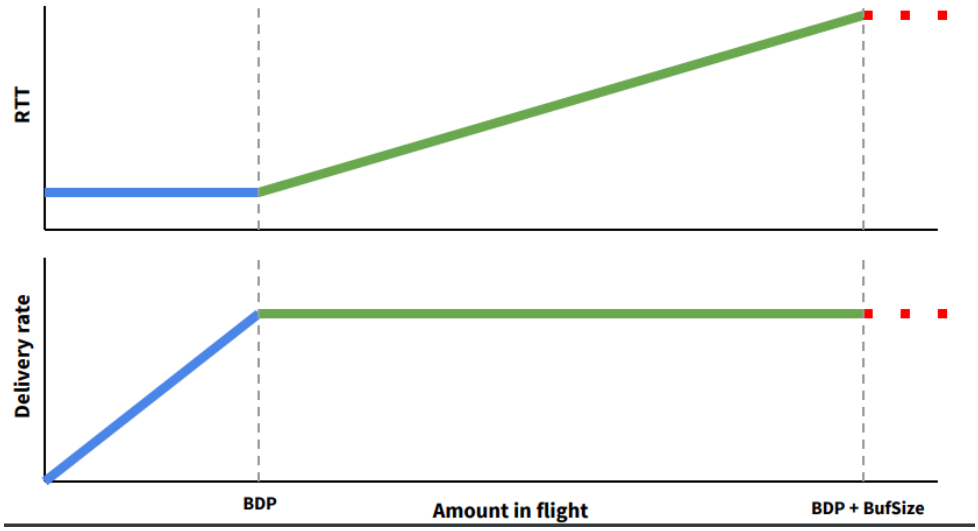

Figura 5 Comportamiento teórico de los paquetes en red (Neal Cardwell and Yuchung Cheng, 2017)

3.5. Data Center TCP

El objetivo de este algoritmo es poder reducir las necesidades de computación en los switches presentes en centros de datos con el fin de abaratar los costes de operación de estas infraestructuras.

El uso de ECN per se no permite conocer el nivel de congestión de la red. Solamente si está congestionada o no, es en este punto donde este algoritmo permite aportar algo utilizando el ECN para poder calcular el nivel de congestión de la red.

Para la correcta implementación se requieren de ciertas especificaciones mínimas en los componentes electrónicos y los respectivos controladores presentes en la red. Siendo este requerimiento que todos los nodos intermedios tienen que poder detectar el bit de congestión de la cabecera IP.

El ordenador de destino comunica la congestión al ordenador de origen a través del bit ECN-Echo que existe en la cabecera TCP tras lo cual el ordenador origen actuará en consecuencia para hacer desaparecer la congestión.

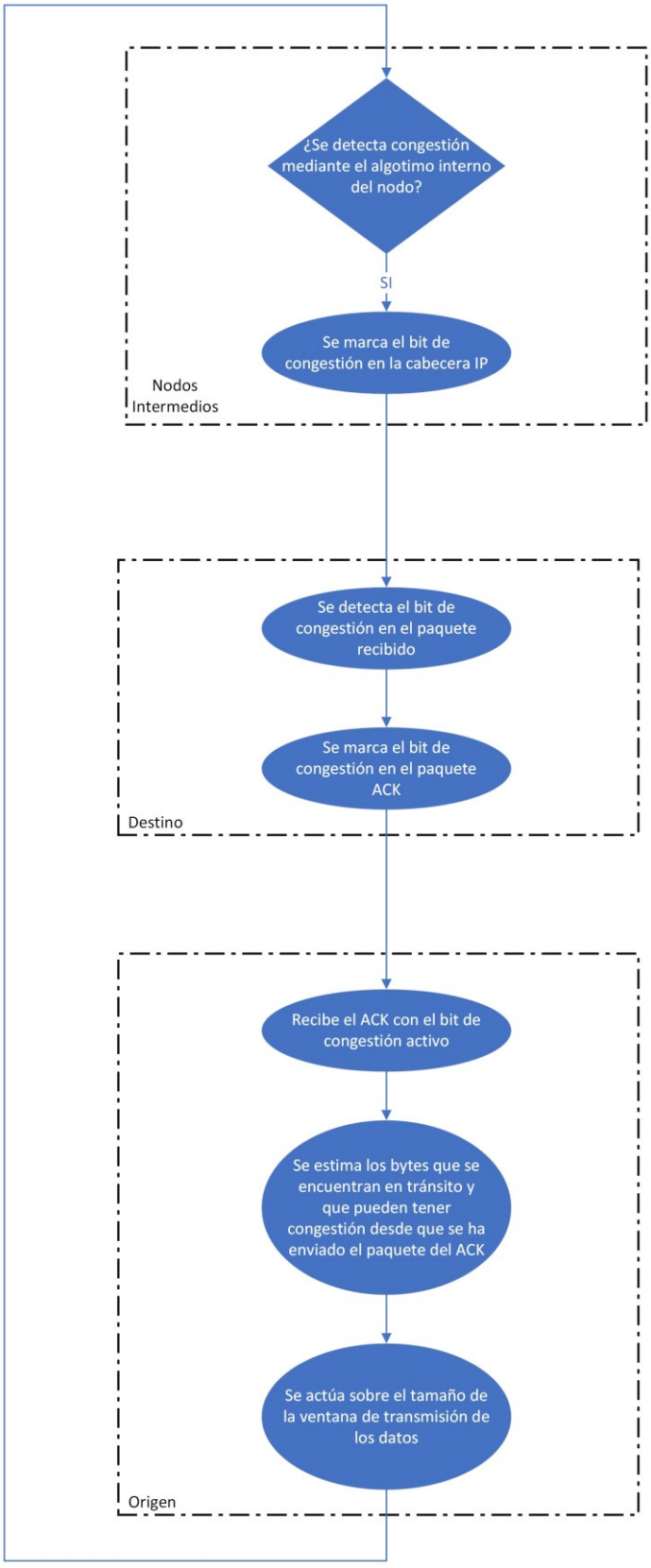

Figura 6 Diagrama de Flujo Datacenter TCP

4. Algoritmos de gestión de colas

4.1. Fifo Droptail

Una cola FIFO (del inglés "first in, first out", o "primero en entrar, primero en salir") es un tipo de estructura de datos en la que el primer elemento que se agrega es también el primer elemento en ser procesado. Es decir, los elementos se procesan en el orden en que se han ido agregando a la cola.

El término “Droptail” se corresponde con el funcionamiento del sistema0. En el instante en que el flujo de paquetes entrante sea superior al de salida, se producirá una acumulación de paquetes en la cola (aplicamos un balance de masa al sistema de la cola básico y llegamos a esta conclusión). Si esta circunstancia se prolonga en el tiempo entonces la cola se llenará y el sistema empezará a descartar todos los paquetes que vayan llegando hasta que la cola tenga espacio como para poder aceptarlos.

Podemos concluir que no tiene una lógica de optimización ni busca el mejor punto de funcionamiento en la cola. La principal ventaja de este algoritmo es su fácil implementación, la reducción de la carga de computación en su ejecución y su versatilidad para ser utilizado en estados iniciales del diseño de experimentos de redes para comprobar el buen funcionamiento de todos los elementos de la red.

4.2. RED

Este algoritmo (del ingés *Random Early Detection)* es de los conocidos como AQM. Tiene como principal objetivo la reducción del tamaño de la cola mediante la asignación a los valores de la probabilidad de ser descartados. Mediante un umbral sobre esta probabilidad decide que paquetes se descartan y cuáles no.

El funcionamiento de este algoritmo busca un pequeño tamaño de cola, siendo más resiliente frente a situaciones donde aumenta significativamente el tráfico de red reduciendo también los retrasos. Además, al descartar paquetes de forma aleatoria, se evita la sincronización global de la red.

Aquel momento donde el tamaño promedio de la cola supera un umbral predeterminado, los nuevos paquetes que llegan pasan a tratarse de una forma diferente. Pudiendo ser descartados o marcados con una cierta probabilidad de descarte. La probabilidad de descarte específica depende del tamaño promedio actual de la cola.

Cuando es necesario informar sobre la congestión en la red, el algoritmo RED elige a un emisor para notificarle que disminuya su tasa de transmisión. La forma de notificación puede ser explícita (devolviendo un paquete de control ICMP al origen) o implícita (descartando el paquete), y es independiente del propio algoritmo RED.

Aquel punto donde este algoritmo puede resultar más problemático es realizando una mala parametrización durante la etapa de diseño de la implementación. Llegando a ocasionar un descenso en el rendimiento de la red frente a otros algoritmos más inicialmente ineficientes como “Droptail”.

Funcionamiento:

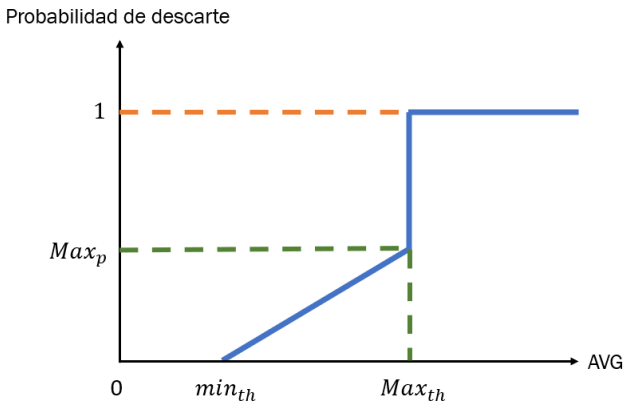

Figura 7 Asignación de probabilidad de descarte en RED

Para que RED funcione, se establecen dos umbrales: min_{th} (umbral mínimo) y Max_{th}(umbral máximo). Cuando llega un nuevo paquete, se comparan estos umbrales con la estimación del tamaño promedio de la cola (Qavg), calculada mediante el algoritmo EWMA (*Exponentially Weighted Moving Average* o Media Móvil con Ponderación Exponencial). Este algoritmo actúa como un filtro pasa-bajo y se define por la siguiente expresión:

$$Qavg = \left(1 - w_q\right) \cdot Qavg_{old} + w_q \cdot Qavg_{inst} \quad (5)$$

Donde:

w_q : Constante que permite asignar el peso para la media móvil, actuando como cota inferior del filtro.

$Qavg_{old}$: Ocupación media de la cola en la anterior evaluación de la cola.

$Qavg_{inst}$: Tamaño instantáneo de la cola.

Teniendo en cuenta el funcionamiento de este algoritmo inicialmente se puede pensar que no sea lo más adecuado para utilizar en un ambiente industrial debido a la aleatoriedad del descarte de paquetes. Sin embargo, se ha incluido este algoritmo en la comparación debido a que se puede utilizar los umbrales de funcionamiento de este para discriminar tráficos de datos. Con un ajuste fino del algoritmo podría llegar a ser útil, razón por la cual se ha tenido en cuenta para las comparativas de este trabajo.

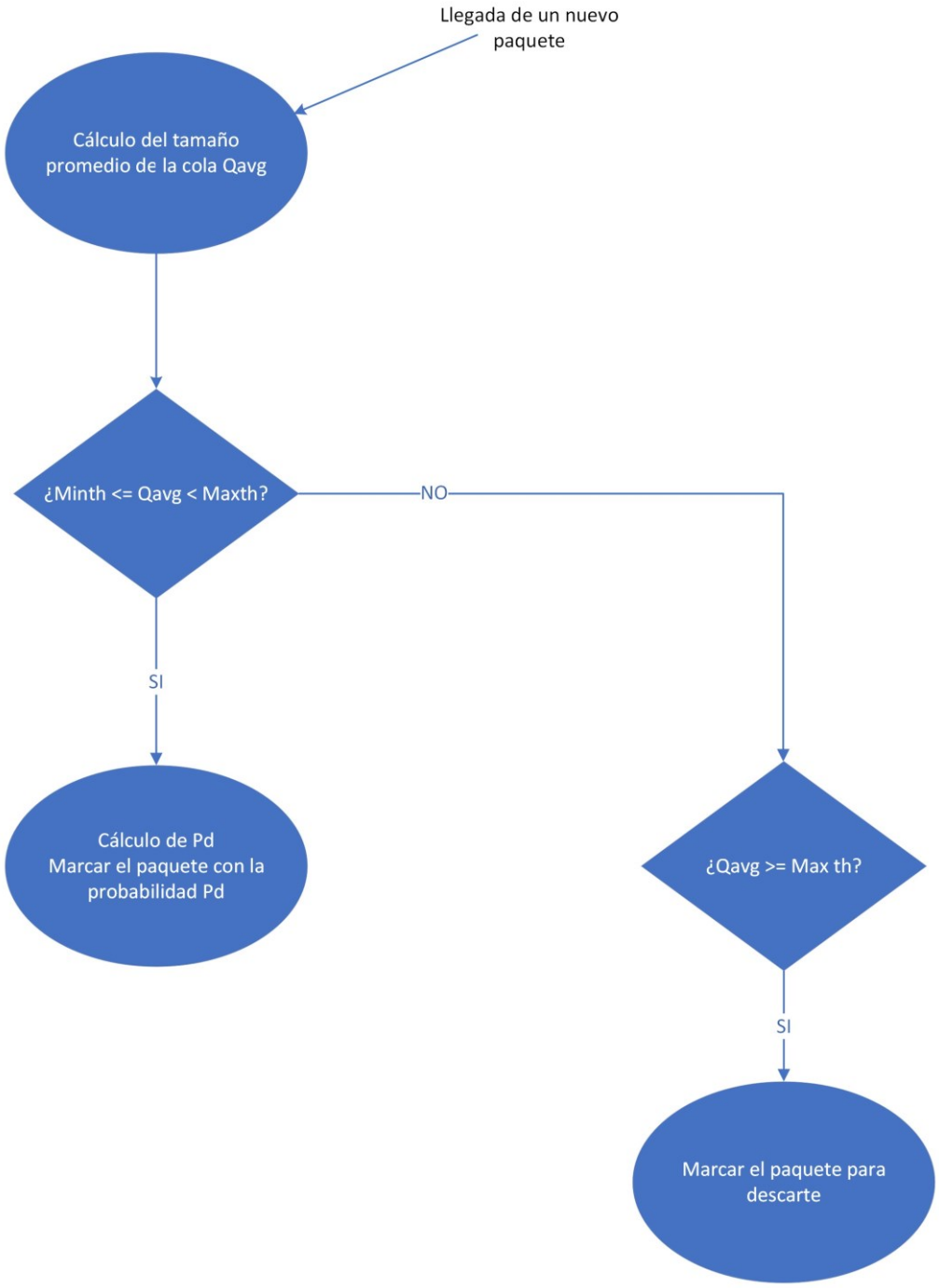

Figura 8 Diagrama de flujo RED

4.3. Codel

Codel(Nichols et al., 2018) discrimina entre “malas colas” aquellas que solo añaden “delay”. “Buenas colas”, que tienen una buena utilización. Se procede a desarrollar el funcionamiento de este algoritmo ya que aunque no se utilice en los experimentos como tal, es una parte importante de fq-codel, que se encuentra más adelante. Al igual que RED, este algoritmo se encuentra dentro de los denominados como algoritmos AQM.

Cuando el uso de las colas tiene una notoria importancia para el buen funcionamiento de la transmisión de datos es justo antes de los enlaces en los que se produce un cuello de botella. Si no hubiera cuello de botella el tamaño de la cola siempre tendría un tamaño pequeño, con algunas excepciones donde haya picos de tráfico. Manteniendo un funcionamiento estable independientemente del tipo de control de congestión de TCP presentes en los ordenadores que envían los paquetes, o del AQM existente en el nodo. Sin embargo, antes del cuello de botella, el tamaño de las colas será directamente dependiente del AQM y de la variante de TCP del tráfico.

Supongamos, por ejemplo, que tenemos una transmisión donde se están enviando 100 paquetes por ventana. Esta misma conexión tiene un BDP de 80 paquetes. Después del pico inicial, la cola se estabilizará en un tamaño aproximado de 20 paquetes. Este tamaño de cola es independiente de la tasa de transmisión de datos y se produce de la siguiente forma:

$$Paquetes\ en\ la\ cola\ \approx\ Paquetes\ en\ la\ ventana\ de\ congestión\ -\ BDP$$

La conexión tiene de forma permanente 100 paquetes en tránsito de los cuales solo llegan 80. En este caso, el 20% de los paquetes se está descartando de forma permanente y la cola del nodo tendrá de forma permanente esos paquetes sin ningún tipo de mejora en la transmisión y añadiendo tiempo al RTT. Este tipo de cola se corresponde con una cola “mala”.

El resto de las colas donde solo hay más o menos 1 paquete, son colas “buenas”

Codel se implementa mediante el uso de 3 componentes bien conocidos en el ámbito de la regulación y la automática.

Observador: Elemento encargado de realizar el muestro del tamaño de cola y de determinar si la cola observada es “buena” o mala”.

SetPoint: Elemento encargado del cálculo del SetPoint medi ante la

Controlador: Nos encontramos frente a un sistema MIMO que implementa un controlador en espacio de estados. Donde un estado sería "Cola mala" y otro estado "Cola buena". Ya que la linealidad del modelo es completamente diferente en cada uno de estos casos. Además, tiene un módulo de adaptación del algoritmo para mejorar el funcionamiento del controlador.

Es necesario mencionar que la implementación de este algoritmo en el simulador se hace de forma mucho más sencilla. Se permite desarrollar algoritmos propios para cada uno de los elementos mencionados anteriormente llegando a la complejidad descrita (sobre todo en el caso del controlador) en este caso el funcionamiento será el será el siguiente:

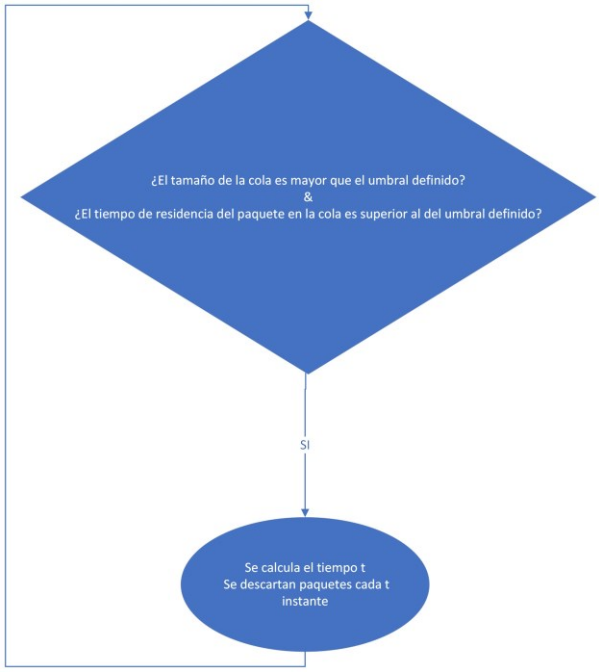

Figura 9 Diagrama de flujo Codel

El cálculo de t se hace mediante el método de Newton para el cálculo de ecuaciones recíprocas.

$$t_i = \frac{t_{i-1}}{\sqrt{n_{descartados}}} \quad (6)$$

Donde:

t_i: Tiempo calculado hasta descartar el próximo paquete.

t_{i-1}: Tiempo calculado para el descarte del paquete anterior.

$n_{descartados}$: Número de paquetes descartados desde que se ha iniciado la congestión de forma continuada.

Descartando los correspondientes paquetes y evitando la congestión de la red.

4.4. FQ Codel

Flow Queue Controlled Delay(Hoeiland-Joergensen et al., 2018) estrictamente hablando no es solo un algoritmo AQM, ya que, por su funcionamiento, también dispone de un algoritmo de asignación de paquetes.

Está diseñado para sacar todo su potencial en aquellas arquitecturas de red donde existen diferentes tráficos de datos con diferentes TCP. Funcionando mejor en tráfico bidireccional manteniendo tamaños de cola pequeños, reduciendo los costes del equipo donde se implementa al no requerir de tanta memoria como por ejemplo RED.

En este caso se le añade al algoritmo de Codel descrito anteriormente un algoritmo de filtrado de paquetes, y un algoritmo de creación de colas y de selección de extracción de cola. Esto permite generar una independencia entre los diferentes tráficos de datos existentes en el nodo donde se implementa el algoritmo. Generando diferentes colas internas mejorando la eficiencia de la transmisión.

El funcionamiento del algoritmo se corresponde con lo expresado en el diagrama que se encuentra a la derecha.

Para la planificación de la extracción de los paquetes de las diferentes colas cuenta con un algoritmo de paso de testigo. De forma que en el momento en que el clasificador crea una nueva cola, tiene que notificárselo al algoritmo de paso de testigo para que lo añada a la planificación. En las simulaciones que se expondrán más adelante, se podrá ver como se desenvuelve este algoritmo, aunque no utilicemos todo su potencial. En este trabajo se han utilizado como mucho 3 nodos de origen de datos, como se ha dimensionado un tamaño de cola estándar ciertamente generoso, no llega a hacer uso de una de las principales características. Siendo también problemática, nos referimos la creación de varias colas asociadas a una misma variante de TCP que en cierto momento dejen de transmitir. En el caso de una red industrial esto no sería particularmente problemático ya que siempre estarán presentes el mismo número de puntos de origen de datos transmitiendo de forma continua. En el caso de que algún punto deje de transmitir, antes o después volverá a hacerlo de nuevo.

Caso diferente a lo que podemos encontrar en los nodos de redes de internet donde el enrutamiento puede que elija un nodo para una retransmisión y no vuelva a escogerlo hasta pasado mucho tiempo. En este caso es necesario implementar un borrado de colas o reaprovechar las “colas zombies” con el uso de tráfico de datos cruzados.

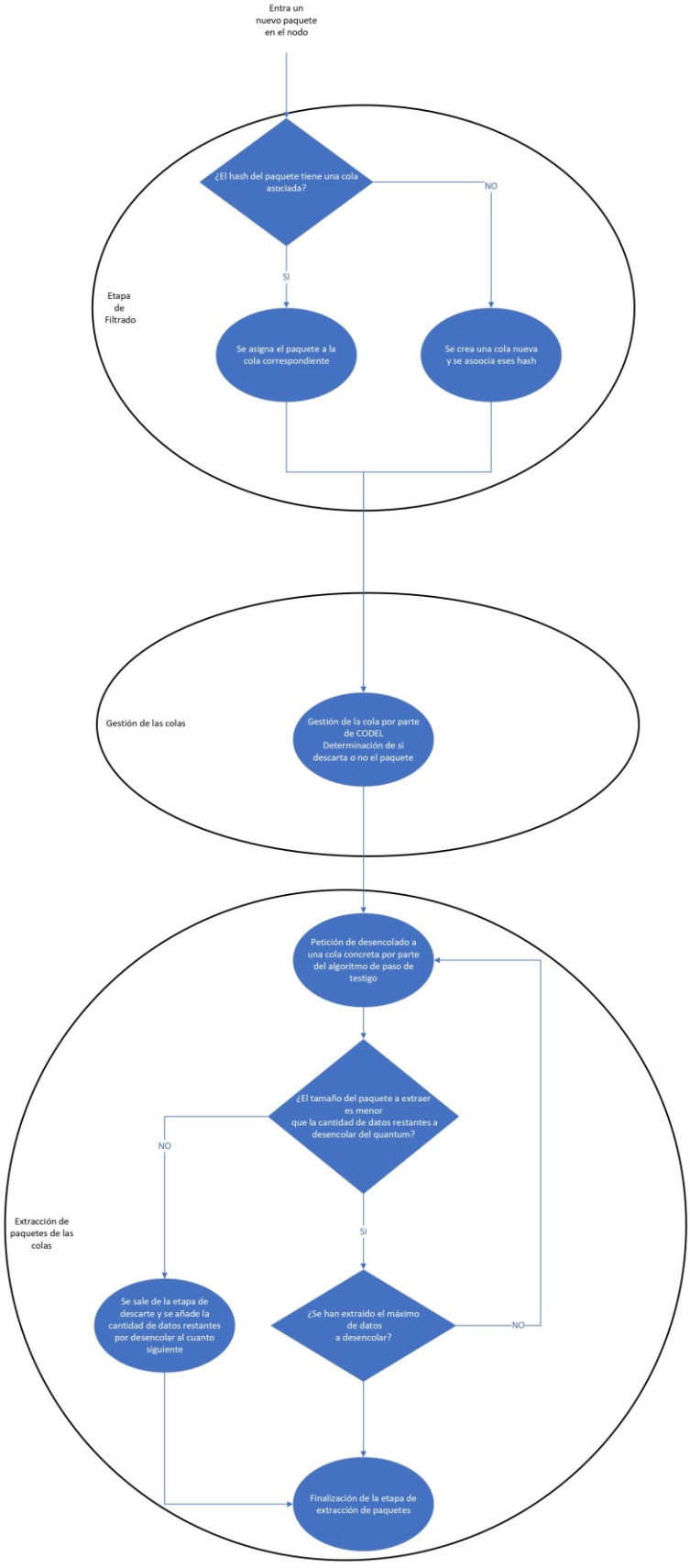

Figura 10 Diagrama de funcionamiento de fq-codel

5. Simulaciones

5.1. Aspectos generales

En el área del análisis del control de congestión es habitual utilizar el termino topología a la hora de describir las redes de ordenadores. Este concepto hace referencia a la comunicación física existente entre los diferentes elementos de una red.

Un ejemplo de topología de red es la llamada topología en estrella.

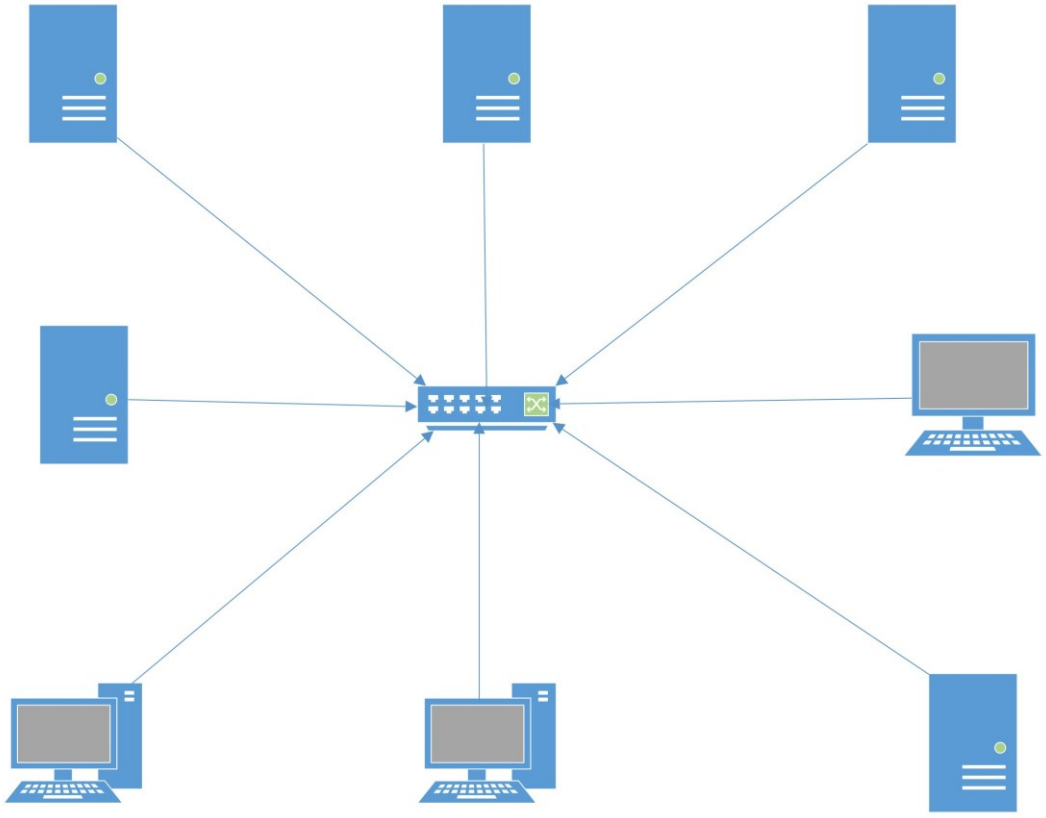

Figura 11 Topología de red en Estrella

Se opta por realizar las simulaciones con la topología1, al permitir que se pueda parametrizar de mejor forma en la que se implementa el retardo en la retransmisión entre nodos. Ya que se ha observado como la elección de este valor y su implementación afecta a nivel numérico a realizar la simulación. Dentro del simulador a la hora de ejecutarlo se implanto como argumento de las simulaciones el delay total. De forma que después de forma interna se asignaba un sexto de este valor a cada enlace punto a punto. La topología 2 es igual pero añadiendo un nodo más para el tráfico cruzado.

5.2. Topología de red

Topología 1

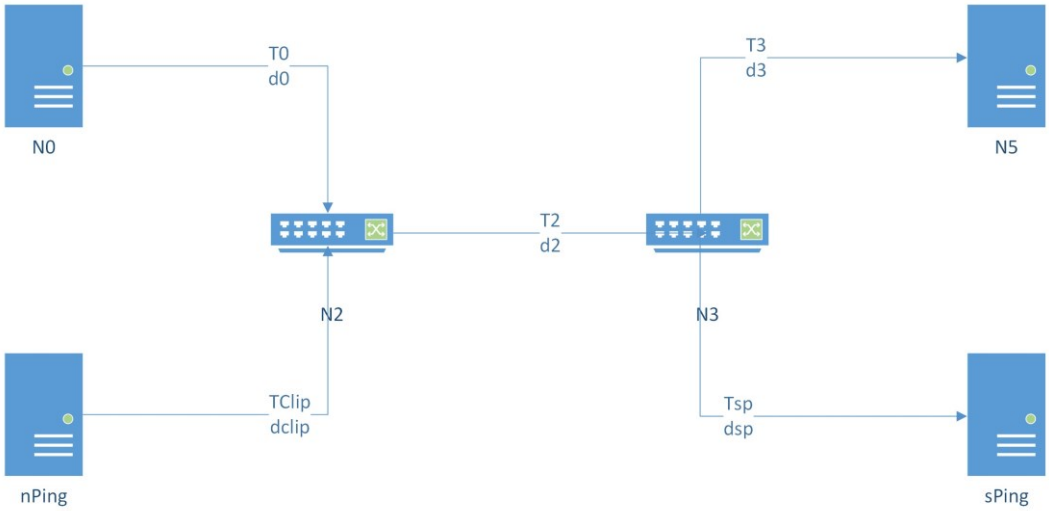

Figura 12 Topología de red 1 para simulación

Donde:

N0: punto donde se produce el origen de la transmisión de datos.

nPing: Se ha introducido otro generador de tráfico cuyo único fin es provocar unas ligeras perturbaciones en la red.

N2: Nodo intermedio donde se produce el cuello de botella y se monitorizaran las colas para el análisis de la congestión de la red.

N3: Nodo intermedio del enlace donde se produce cuello de botella.

N5: Punto de destino donde se reciben los datos enviados por N0 y se devuelven la confirmación de la recepción.

Txx: Ancho de banda máximo del enlace

Dx: Delay en la transmisión entre los nodos.

Para poder realizar las simulaciones correctamente es necesario implementar todos los componentes de la pila TCP/IP en todos los elementos del escenario.

Se utilizan en todos los puntos colas FIFO Droptail y un buffer entre la capa 3 y la capa 4 que varía según el escenario.

También se desactiva de forma predeterminada la detección de congestión por ECN.

Topología 2

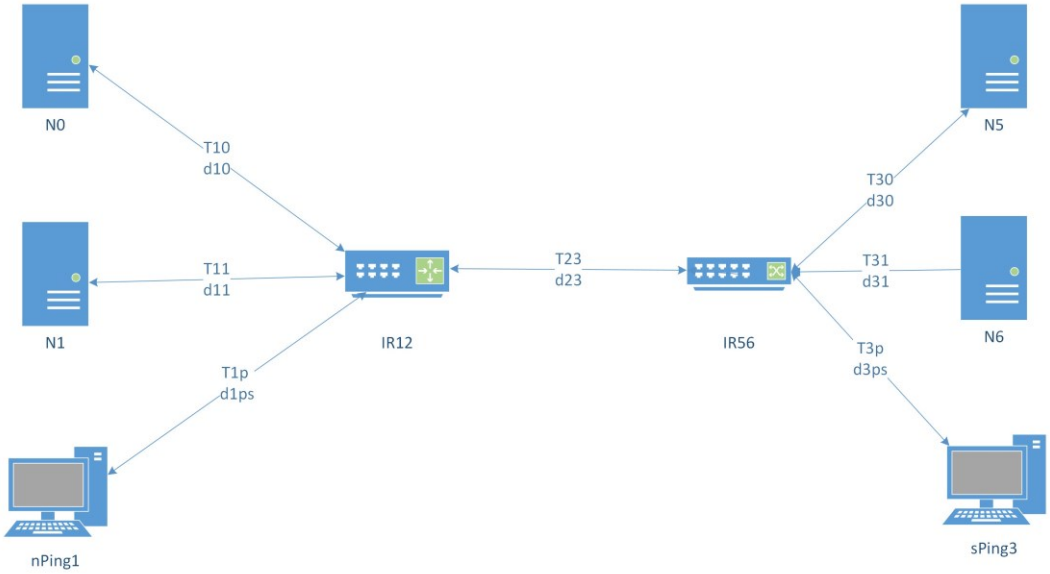

Figura 13 Topología de red 2 para simulación

Donde:

N0: punto donde se produce el origen de la transmisión de datos 1.

N1:punto donde se produce el origen de la transmisión de datos 2.

nPing1: Se ha introducido otro generador de tráfico cuyo único fin es provocar unas ligeras perturbaciones en la red.

N2: Nodo intermedio donde se produce el cuello de botella y se monitorizaran las colas para el análisis de la congestión de la red.

N3: Nodo intermedio del enlace donde se produce cuello de botella.

N5: Punto de destino donde se reciben los datos enviados por N0 y se devuelven la confirmación de la recepción.

N6: Punto de destino donde se reciben los datos enviados por N0 y se devuelven la confirmación de la recepción.

sPing3: Destino del tráfico de datos originado por nPing1

Txx: Ancho de banda máximo del enlace

Dx: Retraso en la transmisión entre los nodos.

5.3. Variables y entorno de estudio

Se recomienda leer este documento abierto en dos pantallas, sobre todo el apartado 5 de simulaciones, teniendo en una pantalla abierto el dicho apartado y en otra el Anexo 1.

Para realizar las simulaciones se opta por utilizar el ns3, simulador que ya tiene implementados el grueso de los algoritmos, permitiendo así agilizar el proceso de desarrollo técnico. También mencionar que las simulaciones de ns3 se pueden programar tanto en Python como en c++ (en este caso se han hecho en c++), lenguajes que forman parte del currículo académico de la titulación y ampliamente utilizados. Para el desarrollo del código y los escenarios de simulación se ha tomado como base el ejemplo de fq-codel (*Ns-3: Src/Traffic-Control/Examples/Fqcodel-L4s-Example.Cc File Reference*, n.d.)al cual se le han hecho profundas modificaciones.

Se ha optado por los siguientes algoritmos a comparar:

Como variantes de tráfico TCP: BIC, New Reno, BBR y datacenter TCP.

Como AQM: Fifo droptail. Red, y fq-codel.

Se han escogido estos algoritmos por diferentes motivos. En el caso de BIC por su importancia histórica siendo la base de TCP CuBIC, actualmente utilizado en múltiples sistemas operativos. Algoritmo principal del Kernel de Linux desde 2006, todos los lanzamientos de MacOS desde 2014 y todas las versiones de Microsoft Windows desde 2017.

Se ha escogido New Reno también por su amplia presencia en “redes de consumo”.

BBR se ha escogido con el fin de determinar si podría mostrar un desempeño suficientemente bueno como para poderse tener en cuenta en ciertas redes industriales. Donde se utilicen ciertos elementos que sin ser elementos de seguridad requieran un mínimo de robustez en la comunicación, pero no tanta como un elemento de seguridad.

DCTCP se ha escogido al presentar un planteamiento teórico que podría acercarse a las necesidades de ciertos entornos industriales.

Dentro de los Algoritmos AQM se ha optado por droptail al ser un indispensable en este tipo de trabajos y por su simplicidad.

La elección de RED también se debe a ser un algoritmo habitual en redes y por también ser habitual en este tipo de estudios.

En cuanto a fq-codel (*Ns-3: Ns3::TcpBbr Class Reference*, n.d.) se opta por este algoritmo al incorporar un módulo muy interesante de control. Aunque en este trabajo no se proceda al desarrollo de un controlador específico dentro del algoritmo, es necesario realizar una aproximación inicial para determinar si podría ser interesante dedicar esfuerzo a adaptar los múltiples algoritmos AQM desarrollados tanto por la tutora de este trabajo Maria Teresa Álvarez, como por el departamento de Automática de la Universidad de Valladolid.

La parametrización del muestro de las variables representadas es la siguiente:

Tabla 1 Parámetros de muestreo

Muestreo de la tasa de transmisión de los nodos	100 milisegundos
Muestreo del descarte de paquetes	200 milisegundos
Muestreo del Rtt	Predeterminado por el simulador

La parametrización de los algoritmos se ha mantenido constante en todos los escenarios. En todos los nodos las colas tienen una capacidad máxima de albergar 500 paquetes.

RED(*RED Queue Disc — Model Library*, n.d.)

Tabla 2 Parámetros de RED

Variable	Valor
Wait	True
Gentle	True
QW	0.002
Umbral -> mínimo	5
Umbral -> Máximo	15

BBR: Se ha optado por la parametrización predeterminada por el simulador. (*Ns-3: Ns3::TcpBbr Class Reference*, n.d.)

DCTCP: Se ha optado por la parametrización predeterminada por el simulador. (*Ns-3: Ns3::TcpDctcp Class Reference*, n.d.)

Se plantean los siguientes escenarios típicos a simular:

Estas simulaciones de hacen con la Topología 1 planteada en el apartado 5.1

Tabla 3 Características de los escenarios a simular

Escenario	AQM	Ancho de banda del enlace	Ancho de banda del cuello de botella	Distribución para la generación del tráfico aleatorio	Mínimo de la distribución	Máximo de la distribución	Media de la distribución
Escenario 1	Fifo Droptail	1 Gbps	100Mbps	Uniforme	0	1	-
Escenario 2	Fq-Codel	1 Gbps	100Mbps	Uniforme	0	1	-
Escenario 3	RED	1 Gbps	100Mbps	Uniforme	0	1	-
Escenario 4	Fifo Droptail	1 Gbps	100Mbps	Uniforme	7	15	-
Escenario 5	Fq-Codel	1 Gbps	100Mbps	Uniforme	7	15	-
Escenario 6	RED	1 Gbps	100Mbps	Uniforme	7	15	-
Escenario 7	Fifo Droptail	1 Gbps	100Mbps	Triangular	7	15	10
Escenario 8	Fq-Codel	1 Gbps	100Mbps	Triangular	7	15	10
Escenario 9	RED	1 Gbps	100Mbps	Triangular	7	15	10

Las diferencias principales que se plantean en los escenarios se encuentran en el algoritmo de gestión de colas en el nodo 2 (donde se produce el cuello de botella) y la generación de tráfico irregular como se puede apreciar en la tabla anterior. Más delante dentro de este apartado se comentarán las simulaciones realizadas que se escapan de los escenarios canónicos y se justificará la existencia de los mismos.

Durante todos los escenarios los nodos mandan 524277360 bytes a menos que se produzca un error en el algoritmo de control como veremos que ocurre en ciertos casos.

Se escoge representar las siguientes variables; Respecto a los puntos de origen del tránsito graficamos diferentes aspectos. La tasa de transmisión de datos en el tiempo nos permite comprobar el aprovechamiento del ancho de banda del cuello de botella que está haciendo, además de poder ver el momento en el que termina de realizar la transmisión, ya que se ha utilizado un tiempo de simulación superior al necesario para el envío de los datos. Además, se representa la ventana de congestión ya que va a ser útil de cara a explicar el funcionamiento de los diferentes variantes de TCP. También se representará el tiempo de retransmisión para comparar los diferentes variantes de TCP entre sí.

Respecto a las colas del nodo 2 donde se produce el cuello de botella representaremos únicamente los paquetes presentes en la cola y aquellos paquetes que se han descartado.

En ocasiones también se representará los valores de estimación del BDP calculados internamente por BBR.

Lo que buscamos es poder concluir que combinación de variante de TCP con que algoritmo de gestión de colas, ya sea AQM o no, tiene una menor pérdida de paquetes, con un menor tiempo en realizar la retransmisión. Después realizaremos unas simulaciones para poder combinar diferentes variantes de TCP y ver como se comportan los algoritmos AQM.

En el Anexo 1 se podrá encontrar todos los resultados de las simulaciones presentadas en este trabajo, consideradas por el autor como las significativas dentro del estudio realizado. En este apartado se comentarán y analizarán aquellos más representativos alineados con los fines de este trabajo.

En primer lugar, es necesario hacer hincapié en los paquetes descartados en los diferentes escenarios por BBR tal y como se aprecia en la siguiente tabla y el siguiente gráfico, en particular los escenarios 4,5,7 y 8 que se comentarán más adelante en profundidad,

Tabla 4 Escenarios

Escenario	BBR	BIC	DCTCP	New Reno
Escenario 1	2540	890	1966	501
Escenario 2	787	352	1824	235
Escenario 3	13297	149	5128	12
Escenario 4	0	1332	2871	732
Escenario 5	0	352	1824	235
Escenario 6	12064	2464	39227	395
Escenario 7	0	1515	2675	727
Escenario 8	0	352	1824	235
Escenario 9	11946	2468	38420	437

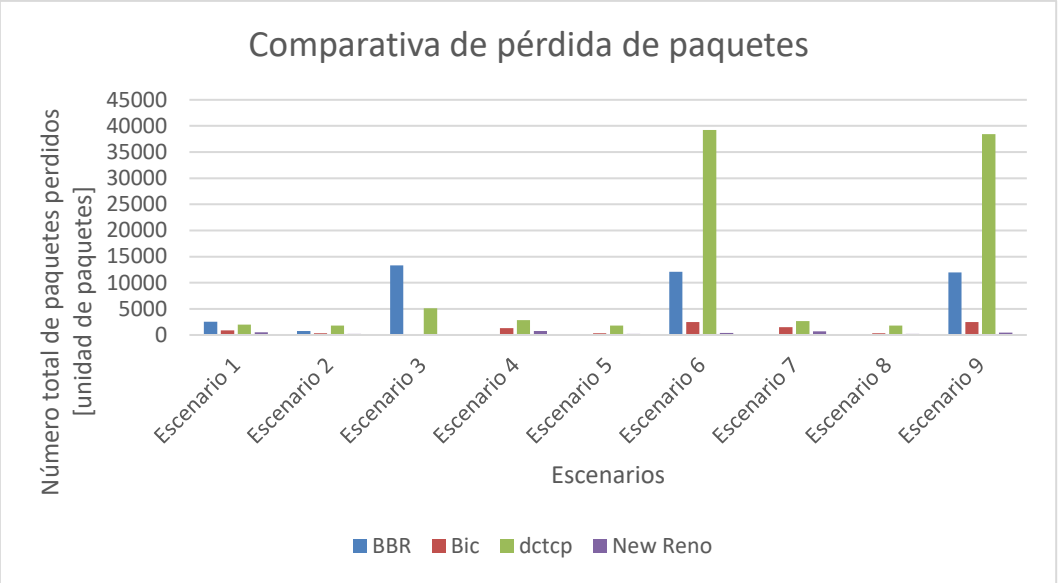

Figura 14 Pérdida de paquetes por escenario

Escenario 1 -FIFO, U(0,1)

En la gráfica de transmisión de datos desde N0 de la simulación 1 que se puede encontrar en el Anexo 1 correspondiente al escenario 1, se puede apreciar un comportamiento muy curioso dentro de BBR. Al ver la gráfica se puede apreciar como se producen unas interrupciones en la transmisión cada aproximadamente 10 segundos. En una primera vista a la gráfica podría no tener sentido, ya que los intervalos por los que se está produciendo la interrupción de la transmisión de datos es de valores entre 0 y 1. Sin embargo no se observa esa misma frecuencia en la gráfica en la disminución de la velocidad de transmisión. Esto se debe a que por el funcionamiento de BBR lo que está ocurriendo en esos instantes es que está entrando en los modos de estimación del ancho de banda del cuello de botella y del tiempo en transito de los paquetes.

Al comparar con la simulación 2 también presente en el Anexo 1 y que también corresponde al escenario 1 se observa como el algoritmo de DataCenterTCP presenta un comportamiento más estable y es menos susceptible frente a interrupciones en la transmisión de datos de estas características.

En las simulaciones 3 y 4 podemos ver el comportamiento esperable de estos variantes de TCP donde la cola permanece con mayor ocupación de paquetes mucho más tiempo que en el resto de los algoritmos.

Cabe también mencionar, que en la simulación 1 se puede observar como el tamaño de la cola solamente es mayor a 1 paquete cuando se reanuda la retransmisión de datos.

Escenario 2 -fq-codel, U(0,1)

Mientras que con una simple cola Fifo en el AQM, DataCenter TCP había tenido mejores resultados que BBR, destacando como había descartado menos paquetes, en este escenario no ocurre lo mismo, obteniendo BBR un mejor resultado. No se hace comenta con especial profundidad los resultados de New Reno o BIC, ya que siempre tienen un comportamiento similar independientemente del escenario por el propio diseño del algoritmo propio de cada uno. Aunque si que es notorio como el número de paquetes presentes en la cola de n2 en la simulación 8 con BIC se ha reducido significativamente respecto al caso anterior.

Se puede apreciar como el uso de fq-codel mejora significativamente el comportamiento de la red con una reducción del 45,39% la pérdida de paquetes. Y en consecuencia la reducción del tiempo necesario para la retransmisión de los datos.

Escenario 3 -RED, U(0,1)

No es de extrañar como, frente a RED, el número de paquetes descartados aumenta respecto a los otros algoritmos. Se puede destacar el buen funcionamiento que tiene este AQM con la variante de TCP New Reno, y aunque si que tenga un descarte para nada desdeñable BBR tiene un comportamiento mucho más estable que el puede presentar por ejemplo DCTCP en la simulación 10.

Escenario 4 -FIFO,U(7,15)

En la simulación 13 se puede apreciar como BBR presenta un comportamiento anómalo para el estudio de la congestión de red. Que es, precisamente, que no se produzca y que el algoritmo pueda actuar sobre el flujo de datos a tiempo para que no se produzca el descarte de ningún paquete. Desde luego para el estudio de AQM no resulta el algoritmo más adecuado, sin embargo, esta característica hace que sea idóneo para ciertas aplicaciones industriales.

Cabe recordar que estas simulaciones se están realizando con la configuración predeterminada del algoritmo y cualquier implantación de este en un entorno industrial requiere de un análisis detallado de las condiciones de red y de toda posible congestión que pueda llegar a ocurrir, acotando completamente que los rangos de trabajo que tenga el nodo se correspondan con los que tiene este funcionamiento, ya que, como se ha visto en el escenario 1,2 y 3 frente a situaciones no tan beneficiosas para BBR puede producir un descarte de paquetes importante al inicio o restablecimiento de las conexiones.

Respecto a los otros algoritmos en las simulaciones 14,15 y 16 se comportan de forma correcta.

Escenario 5 -fq-codel, U(7,15)

En este escenario es reseñable como en la simulación 19 New Reno no logra tener tan buen rendimiento como el resto de algoritmos. Mientras que el resto termina de mandar los datos en apenas unos 75 segundos (sin tener en cuenta la última retransmisión de datos que tienen algunos en el segundo 80 que los demás no necesitan) New Reno hasta el segundo 85 aún sigue transmitiendo.

Esto lo podemos ver claramente comparando las simulaciones 19 y 20. Mientras que en la simulación 20, BIC tiene una tasa de transferencia de datos mucho más estable logrando aprovechar el 100% del ancho de banda del cuello de botella, New Reno solo alcanza el 100%del ancho de banda de la retransmisión aproximadamente un 10% del tiempo de está transmisión.

Resto de escenarios

En el resto de los escenarios se pueden observar que se repiten los comportamientos analizados anteriormente y que por lo tanto no se va a repetir la correspondiente explicación.

Simulaciones con tráfico cruzado

En estas simulaciones se implementan la Topología 2 planteada en el apartado 5.1 y corresponden con las simulaciones desde la 37 hasta la simulación 49.

El primer origen de datos (N0) empieza a transmitir datos en el segundo 5, mientras que el otro (N1) comienza a transmitir en el segundo 15 de la simulación, esto se ha hecho para poder comprar el comportamiento de los algoritmos cuando están solos y ver la respuesta a la variación de las condiciones de la red.

Tras determinar el funcionamiento de los algoritmos se plantea estudiar el funcionamiento entre varios nodos, particularizando en el algoritmo de BBR, que hasta ahora nos había dado tan buenos resultados.

En la simulación 37, donde implementamos en el nodo N0 y N1 BBR, observamos que el resultado dista mucho de lo esperado y de lo que se ha observado con anterioridad. En este punto buscando una posible explicación a este resultado, acudimos a representar los estimadores que de forma interna calcula este algoritmo.

Para poder representar esta variable se ha modificado el código fuente del algoritmo BBR. La modificación consta en que cada vez que está calculando el BDP, se añade el valor de los estimadores a un archivo. En esta simulación se están ejecutando dos ordenadores con BBR y por la implementación de la representación no se puede discriminar entre las estimaciones de cada nodo. Independientemente de esto, el valor teórico(Editor & Board, 2005) al que tendría que llegar es de la mitad del ancho de banda del cuello de botella.

Se puede apreciar como el algoritmo de búsqueda del punto óptimo. Al parecer esto ocurre por las condiciones que hacen que BBR salga del estado de Drenaje (presente en el diagrama de flujo del apartado 3.4). Como consecuencia el algoritmo deja de enviar datos interrumpiéndose la comunicación.

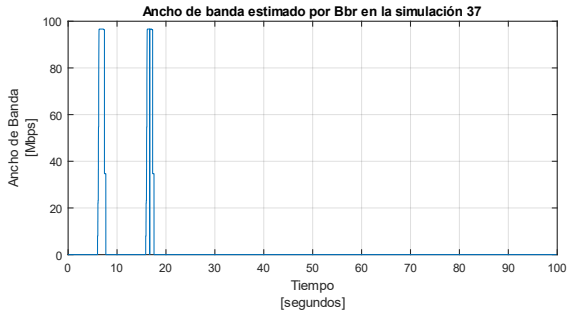

Ilustración 1 Ancho de banda estimado Simulación 37

A diferencia de los casos en que solo hay un tráfico de TCP, en esta colección de escenarios, tener como AQM a RED hace que al menos exista transmisión de datos en alguno de los ordenadores como se puede apreciar en la simulación 39.

Al igual que en los anteriores escenarios, se aprecia como en aquellas simulaciones donde el tiempo de transmisión de datos es superior, BBR funciona mejor. Las simulaciones 40,41 y 42 son una muestra de esto.

Si bien en el apartado 4.4 se explicaba la capacidad de fq-codel para discriminar tráfico mediante el filtrado de paquetes con el hash único de cada uno, se puede apreciar como este filtrado no funciona correctamente.

Podría estarse produciendo el mal funcionamiento del cálculo de BBR por la implementación de dos BBR simultáneos en el simulador, pero tras el análisis del código no parece ser está la causa. Ya que en la simulación 46 donde enfrentamos BBR con New Reno uitilzando fq-codel como AQM, obtenemos un comportamiento desfavorable, ocupando BBR prácticamente todo el ancho de banda del cuello de botella.

En la simulación 49 donde New Reno se encuentra en el origen de tráfico N0 y BBR en el N1, presenta un mejor comportamiento pero aún muy alejado de lo que se podría desear en un ambiente industrial.

6. Conclusiones

6.1. Conclusiones generales.

El problema de la congestión en las redes de ordenadores es importante ya que cuando hay un exceso de tráfico en el sistema, el rendimiento se ve afectado, causando retrasos y pérdida de paquetes. Para solucionarlo, el protocolo TCP cambia su velocidad de transmisión según el nivel de congestión. Sin embargo, con el rápido crecimiento de Internet y las aplicaciones, los métodos antiguos para controlar la congestión están dejando de ser efectivos. En respuesta a esto, surgen nuevos algoritmos llamados AQM que se implementan en los routers para controlar el tamaño de las colas y descartar o aceptar paquetes según sea necesario. Un ejemplo de estos algoritmos es RED, el cual ha sido propuesto como estándar por la IETF, pero su rendimiento no ha sido el mejor y por lo tanto, se han creado numerosos nuevos algoritmos para controlar las colas.

Se ha demostrado como BBR con unas condiciones de red favorables puede conseguir un gran ajuste de sus parámetros y de evitar que se produzca descarte de paquetes en la congestión de red. Si bien es cierto, que para hacer que esto ocurra es necesario un ajuste fino de los parámetros del algoritmo para que se adapte correctamente a las necesidades de la red.

Además, se ha confirmado el buen funcionamiento de fq-codel, dando pie a que se pueda seguir experimentando con este algoritmo adaptando los desarrollos en controladores, tanto predictivos como en otras técnicas de control e implementarlos de cara a poder mejorar el tráfico cruzado donde se han obtenido peores resultados.

6.2. Futuras líneas de investigación

A partir de los resultados obtenidos se puede ampliar el trabajo con las siguientes líneas de trabajo

- Mejora del algoritmo fq-codel en el tráfico cruzado de BBR
- Implementación de un módulo en fq-codel que permita variar el controlador con más facilidad de cara a poder realizar más experimentos sobre este algoritmo.
- Desarrollo de un experimento implementar BBR en un entorno real con elementos de IOT basados en Arduino Industrial y/o Raspberry junto con un abanico de algoritmos de gestión de colas particularizando en los escenarios que en este trabajo se han demostrado como más adecuados.

7. Bibliografía

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., & Sridharan, M. (2010). Data Center TCP (DCTCP). *Computer Communication Review*, *40*(4), 63–74. https://doi.org/10.1145/1851275.1851192

Alvarez, T., Alvarez, T., & Cristea, S. (2008). *AQM Control of TCP/IP Networks using Generalized Predictive Control Wind Tunnel View project Congestion Control View project AQM Control of TCP/IP Networks using Generalized Predictive Control*. https://www.researchgate.net/publication/236236176

Caini, C., & Firrincieli, R. (2004). TCP Hybla: A TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, *22*(5), 547–566. https://doi.org/10.1002/sat.799

Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M., & Wang, R. (2002). TCP Westwood: end-to-end bandwidth estimation for enhanced transport over wireless links. *Wireless Networks*, *8*(5), 467–479. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.7980&rep=rep1&type=pdf

draft-cardwell-iccrg-bbr-congestion-control-00. (n.d.). Retrieved January 6, 2023, from https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-00

Editor, S., & Board, E. (2005). The Mathematics of Internet Congestion Control. In *IEEE Transactions on Automatic Control* (Vol. 50, Issue 1). https://doi.org/10.1109/tac.2004.841398

Fall, K., & Floyd, S. (1996). Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review*, *26*(3), 5–21. https://doi.org/10.1145/235160.235162

Floyd, S., & Henderson, T. (1999). *The NewReno Modification to TCP's Fast Recovery Algorithm*. https://doi.org/10.17487/RFC2582

Floyd, S., & Jacobson, V. (1993). Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, *1*(4), 397–413. https://doi.org/10.1109/90.251892

Ha, S., Rhee, I., & Xu, L. (2008). CUBIC: A new TCP-friendly high-speed TCP variant. *Operating Systems Review (ACM)*, *42*(5), 64–74. https://doi.org/10.1145/1400097.1400105

Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., & Dumazet, E. (2018). *The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm*. https://doi.org/10.17487/RFC8290

Jacobson, V. (1995). Congestion avoidance and control. *Computer*

Communication Review, *25*(1), 157–173. https://doi.org/10.1145/205447.205462

Khan, J. A., & Butt, A. R. (2018). *Improving TCP Data Transportation for Internet of Things Improving TCP Data Transportation for Internet of Things*.

Neal Cardwell and Yuchung Cheng. (2017). BBR Congestion Control: An Update. *Proceedings of IETF-98*.

Nichols, K., Jacobson, V., McGregor, A., & Iyengar, J. (2018). *Controlled Delay Active Queue Management*. https://doi.org/10.17487/RFC8289

ns-3: ns3::TcpBbr Class Reference. (n.d.). Retrieved January 9, 2023, from https://www.nsnam.org/doxygen/d0/d4c/classns3_1_1_tcp_bbr.html

ns-3: ns3::TcpDctcp Class Reference. (n.d.). Retrieved January 9, 2023, from https://www.nsnam.org/doxygen/d2/d15/classns3_1_1_tcp_dctcp.html

ns-3: src/traffic-control/examples/fqcodel-l4s-example.cc File Reference. (n.d.). Retrieved January 9, 2023, from https://www.nsnam.org/docs/release/3.33/doxygen/fqcodel-l4s-example_8cc.html

Pan, R., Natarajan, P., Baker, F., & White, G. (2017). *Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem*. https://doi.org/10.17487/RFC8033

Ramakrishnan, K., Floyd, S., & Black, D. (2001). *The Addition of Explicit Congestion Notification (ECN) to IP*. https://doi.org/10.17487/RFC3168

RED queue disc — Model Library. (n.d.). Retrieved January 9, 2023, from https://www.nsnam.org/docs/models/html/red.html

RFC 793 - Transmission Control Protocol. (n.d.). Retrieved January 10, 2023, from https://datatracker.ietf.org/doc/rfc793/

Shreedhar, M., & Varghese, G. (1996). Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking*, *4*(3), 375–385. https://doi.org/10.1109/90.502236

Song, Y. J., Kim, G. H., Mahmud, I., Seo, W. K., & Cho, Y. Z. (2021). Understanding of BBRv2: Evaluation and Comparison with BBRv1 Congestion Control Algorithm. *IEEE Access*, *9*, 37131–37145. https://doi.org/10.1109/ACCESS.2021.3061696

Tan, K., Song, J., Zhang, Q., & Sridharan, M. (2006). A compound TCP approach for high-speed and long distance networks. *Proceedings - IEEE INFOCOM*. https://doi.org/10.1109/INFOCOM.2006.188

Xu, L., Harfoush, K., & Rhee, I. (2004). Binary increase congestion control (BIC) for fast long-distance networks. *Proceedings - IEEE INFOCOM*, *4*, 2514–2524. https://doi.org/10.1109/INFCOM.2004.1354672

ANEXOS

ANEXO 1: Resultados gráficos de las simulaciones

Simulación 1

Tabla 5 Simulación 1

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
1	BBR	-	fifo droptail	1Gbps	100Mbps	Uniforme	2540

Min_U	Max_U
0	1

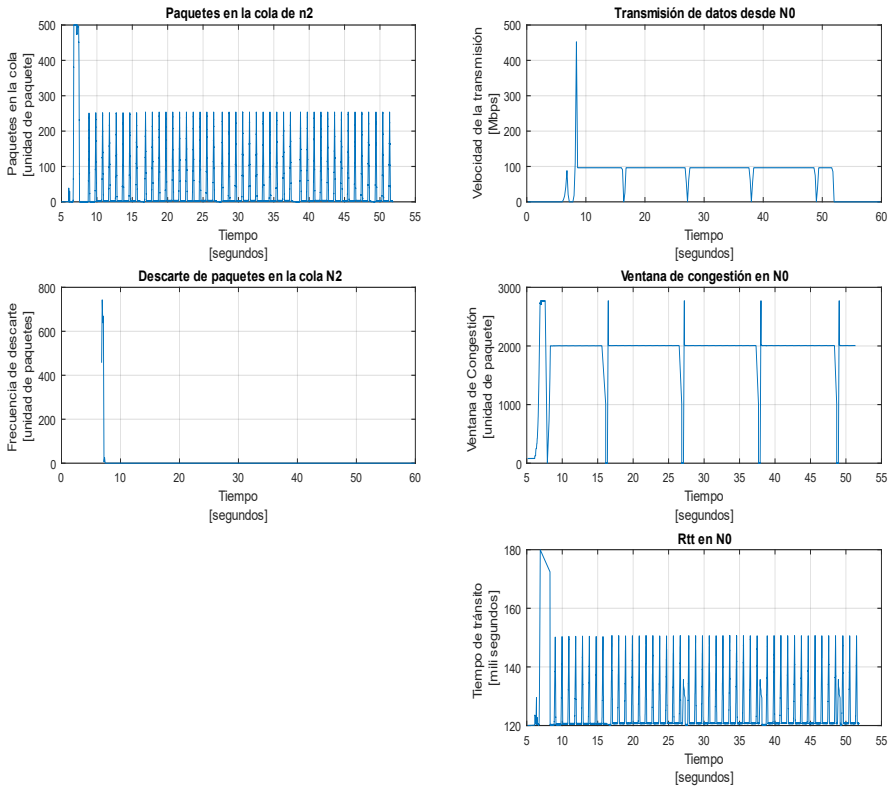

Figura 15 1672741931codigo_v150v

Simulación 2

Tabla 6 Simulación 2

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
2	DCTCP	-	fifo droptail	1Gbps	100Mbps	Uniforme	1966

Min_U	Max_U
0	1

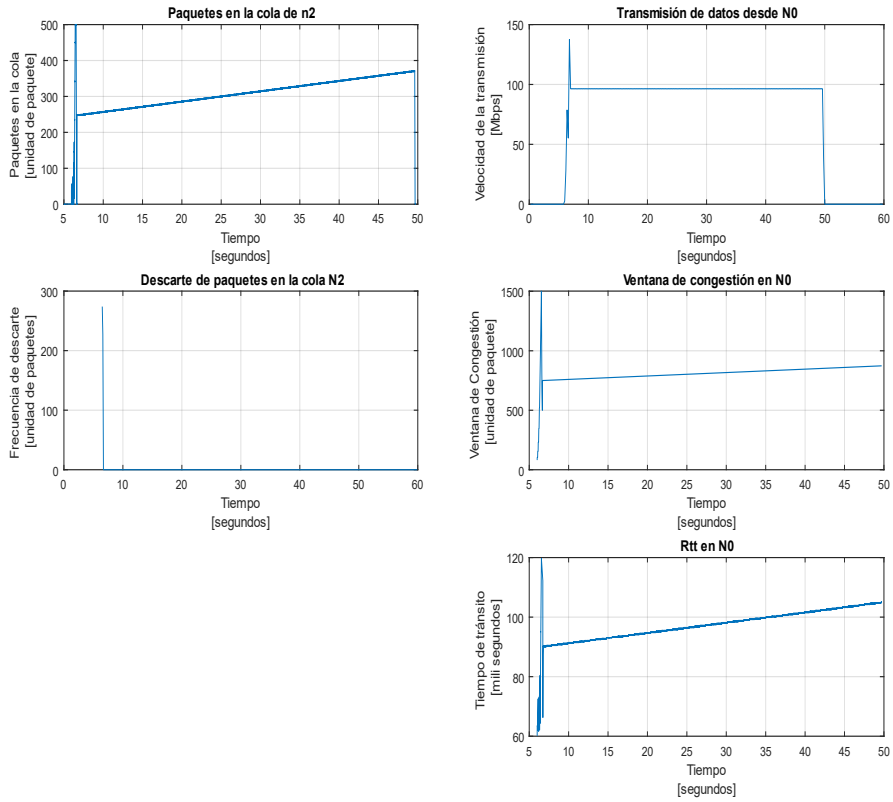

Figura 16 1672744366codigo_v150v

Simulación 3

Tabla 7 Simulación 3

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
3	reno	-	fifo droptail	1Gbps	100Mbps	Uniforme	501

Min_U	Max_U
0	1

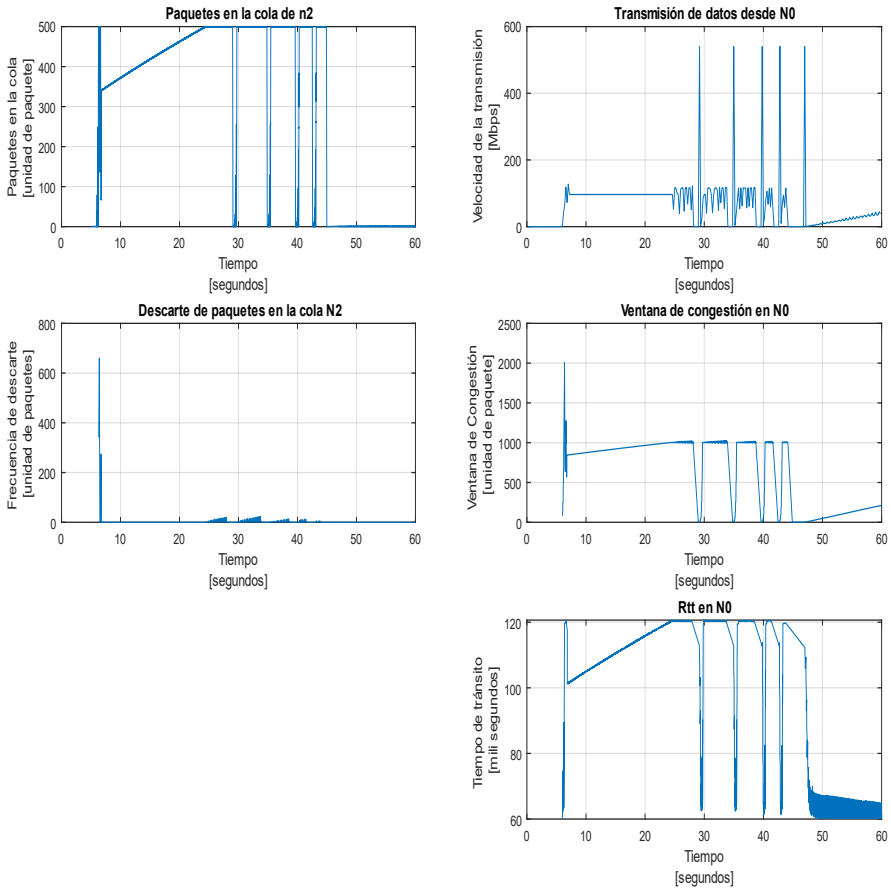

Figura 17 1672744913codigo_v150v

Simulación 4

Tabla 8 Simulación 4

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
4	BIC	-	fifo droptail	1Gbps	100Mbps	Uniforme	890

Min_U	Max_U
0	1

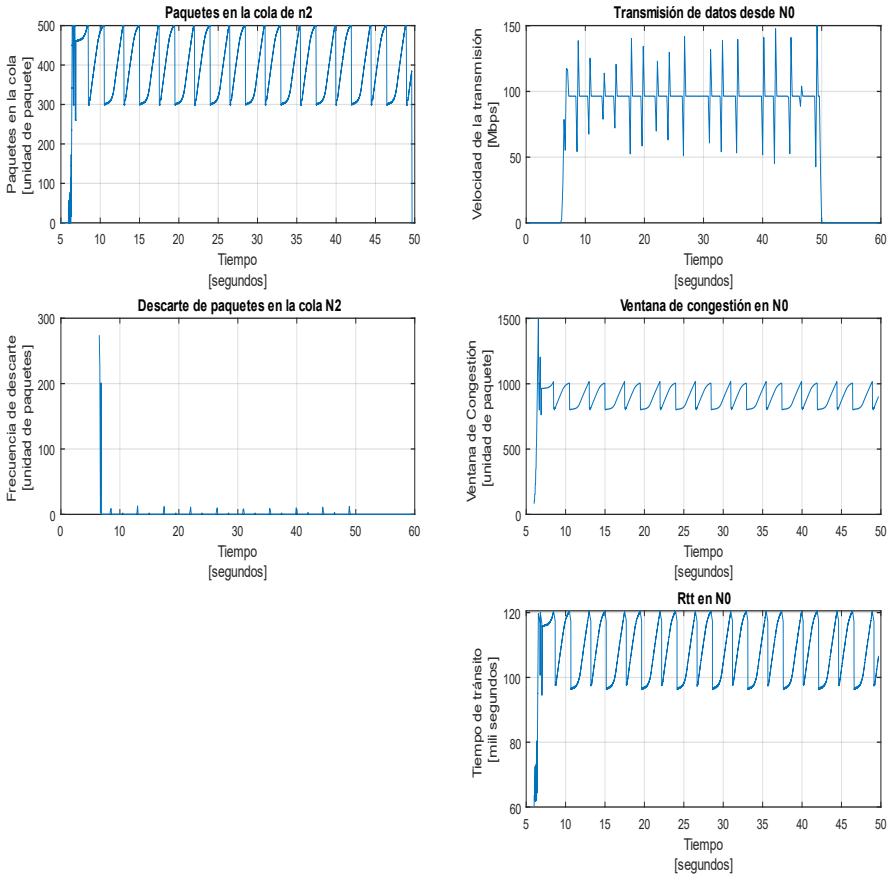

Figura 18 1672745661codigo_v150v

Simulación 5

Tabla 9 Simulación 5

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
5	BBR	-	fqcodel	1Gbps	100Mbps	Uniforme	787

Min_U	Max_U
0	1

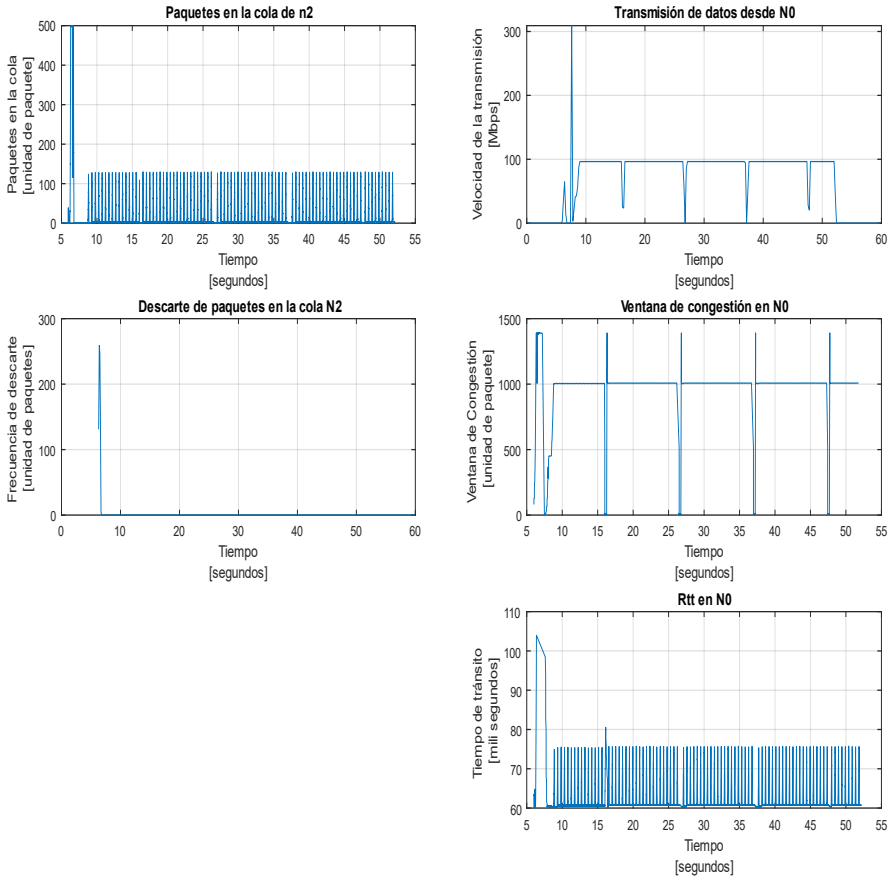

Figura 19 1672746394codigo_v150v

Simulación 6

Tabla 10 Simulación 6

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
6	DCTCP	-	fqcodel	1Gbps	100Mbps	Uniforme	1824

Min_U	Max_U
0	1

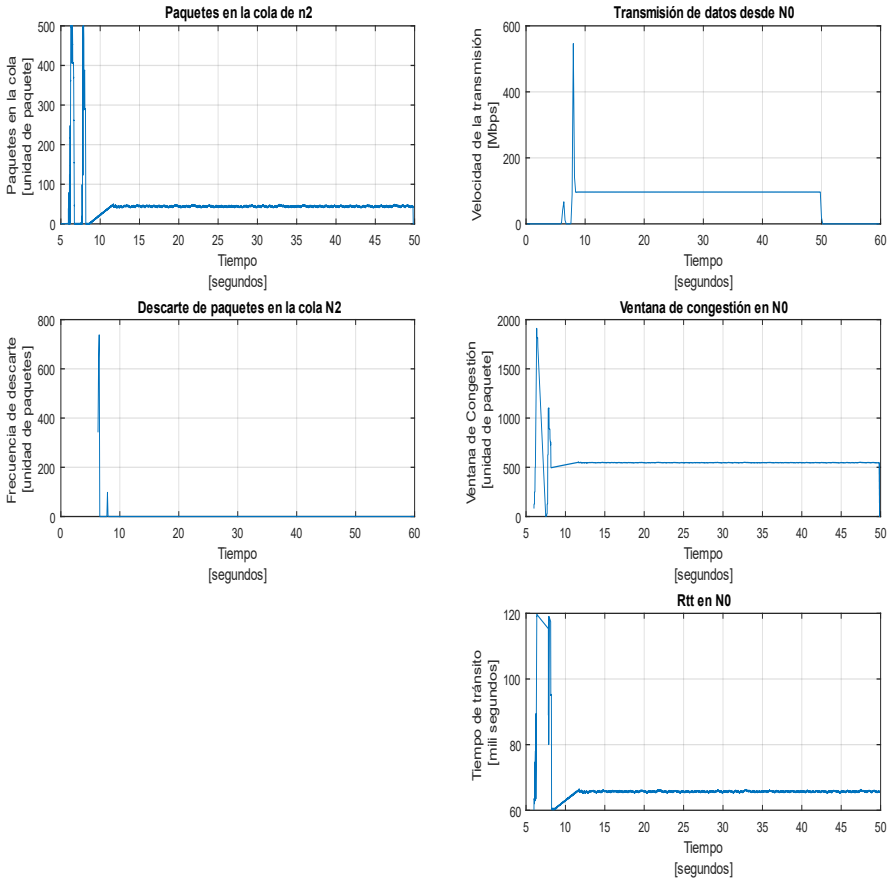

Figura 20 1672747030codigo_v150v

Simulación 7

Tabla 11 Simulación 7

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
7	reno	-	fqcodel	1Gbps	100Mbps	Uniforme	235

Min_U	Max_U
0	1

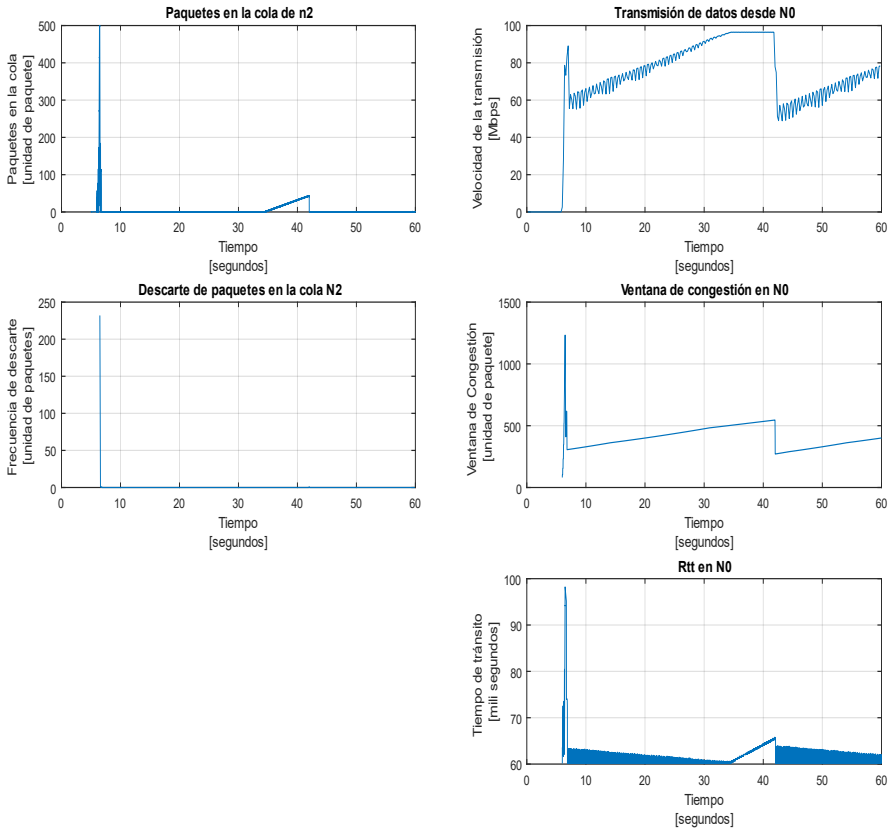

Ilustración 2 1672747303codigo_v150v

Simulación 8

Tabla 12 Simulación 8

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
8	BIC	-	fqcodel	1Gbps	100Mbps	Uniforme	352

Min_U	Max_U
0	1

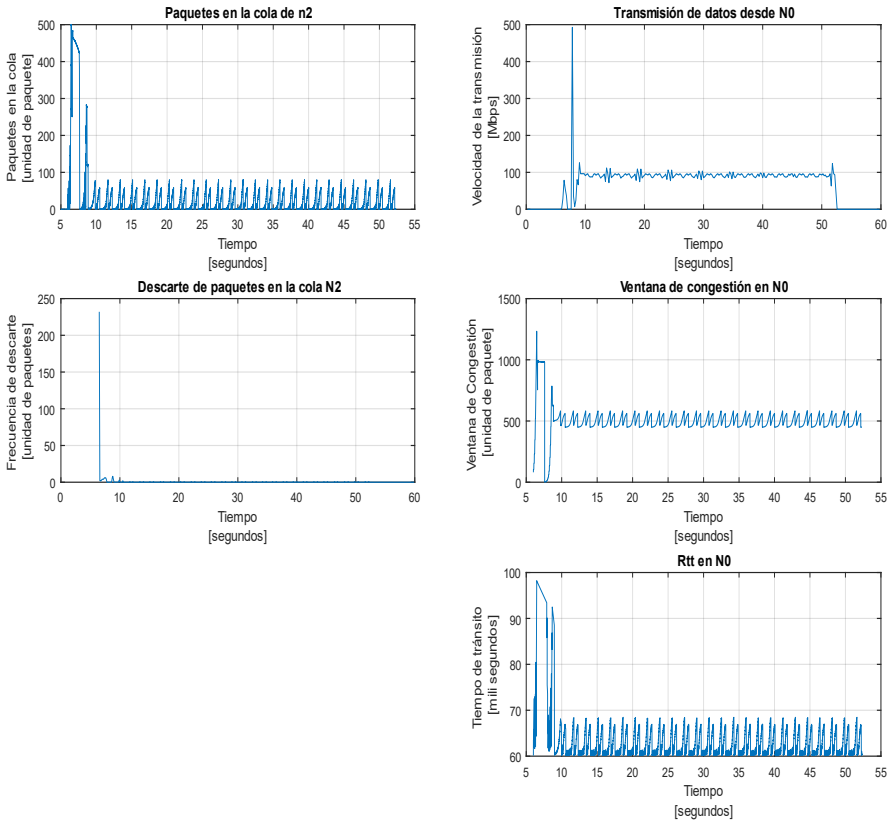

Ilustración 3 1672747508codigo_v150v

Simulación 9

Tabla 13 Simulación 9

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
9	BBR	-	RED	1Gbps	100Mbps	Uniforme	13297

Min_U	Max_U
0	1

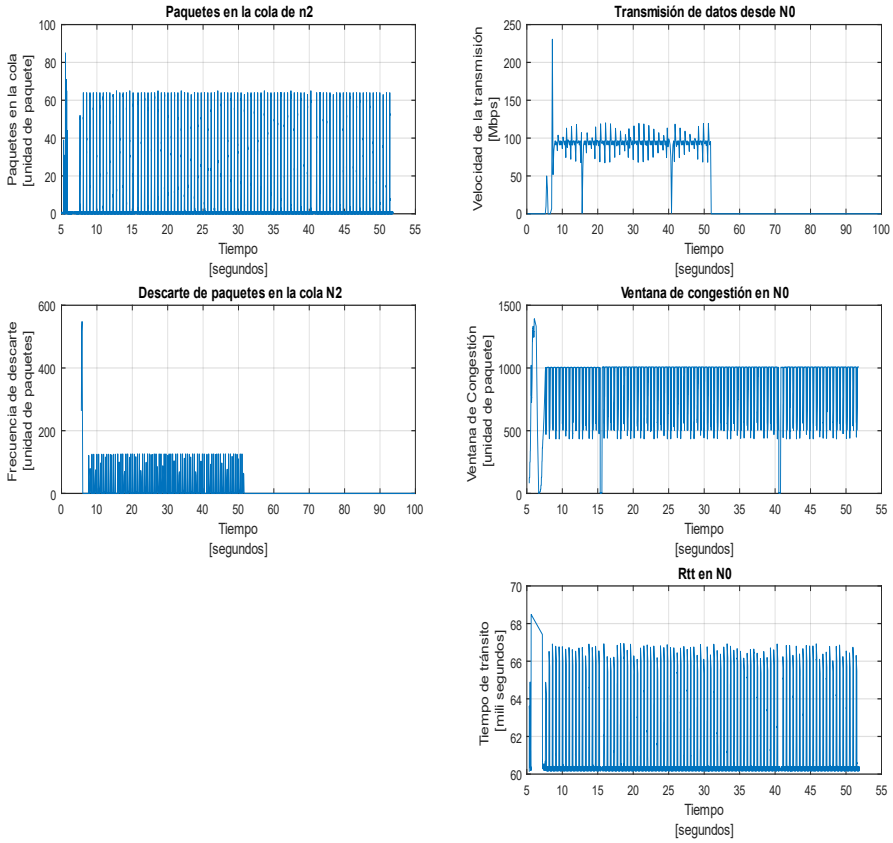

Ilustración 4 1672748471codigo_v150v

Simulación 10

Tabla 14 Simulación 10

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
10	DCTCP	-	RED	1Gbps	100Mbps	Uniforme	5128

Min_U	Max_U
0	1

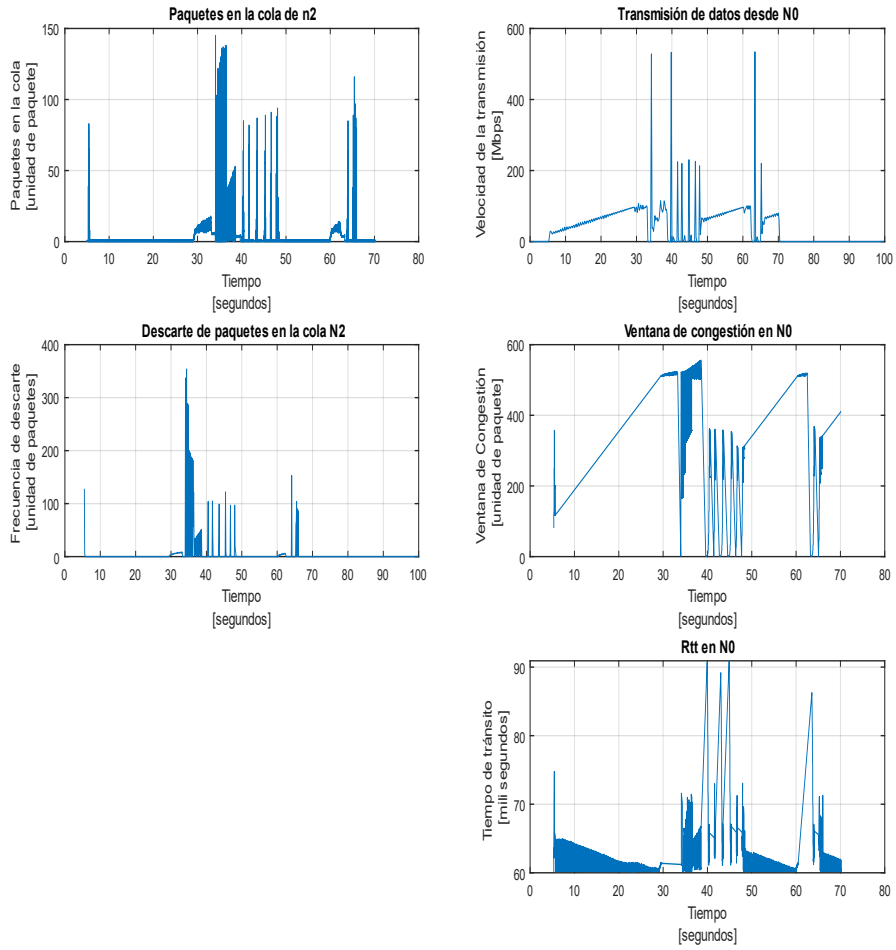

Ilustración 5 1672748824codigo_v150v

Simulación 11

Tabla 15 Simulación 11

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
11	reno	-	RED	1Gbps	100Mbps	Uniforme	12

Min_U	Max_U
0	1

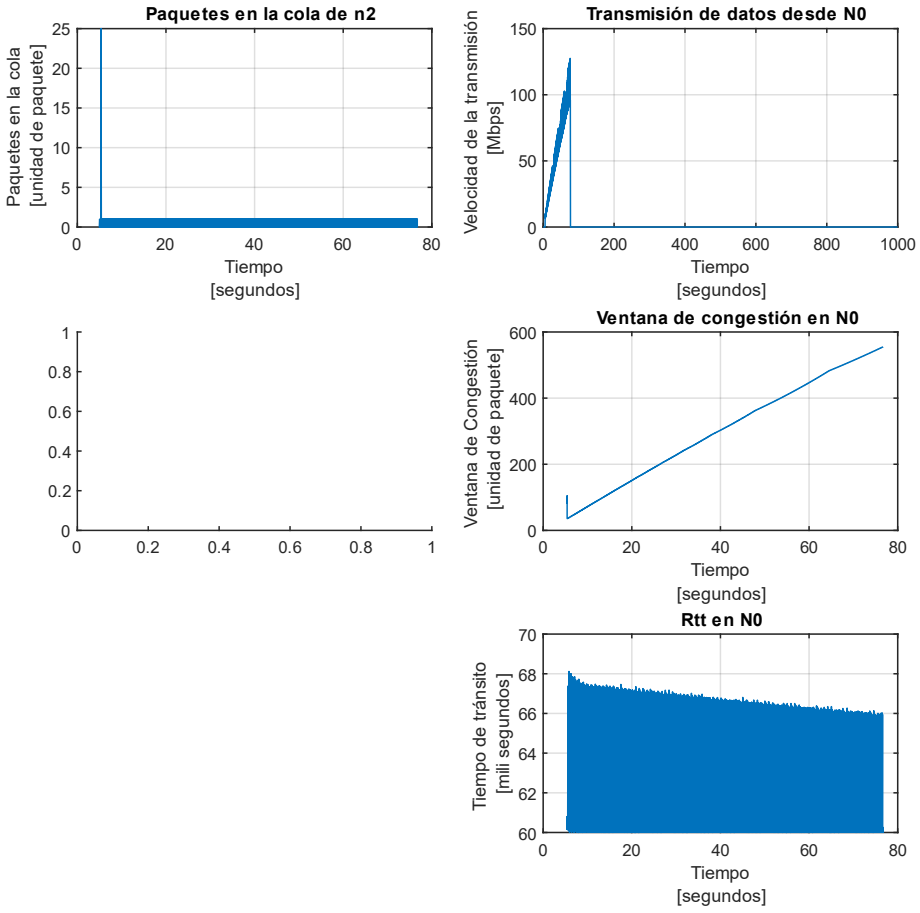

Figura 21 1673415278codigo_v150v

Simulación 12

Tabla 16 Simulación 12

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
12	BIC	-	RED	1Gbps	100Mbps	Uniforme	149

Min_U	Max_U
0	1

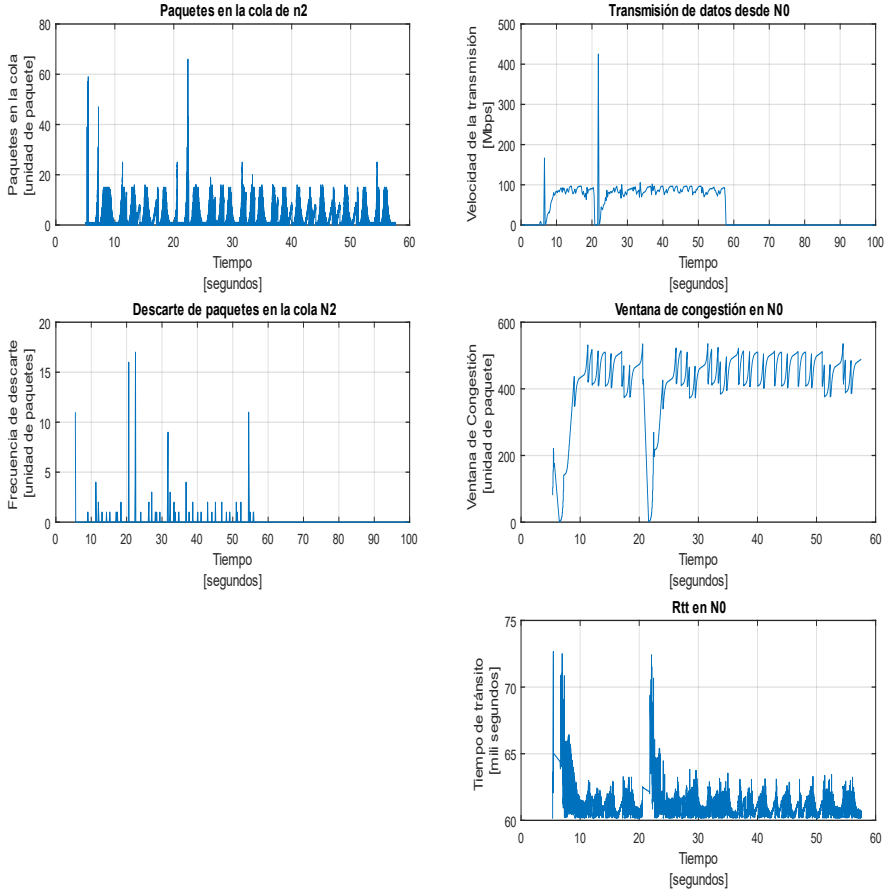

Figura 22 1672749290codigo_v150v

Simulación 13

Tabla 17 Simulación 13

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
13	BBR	-	fifo droptail	1Gbps	100Mbps	Uniforme	0

Min_U	Max_U
7	15

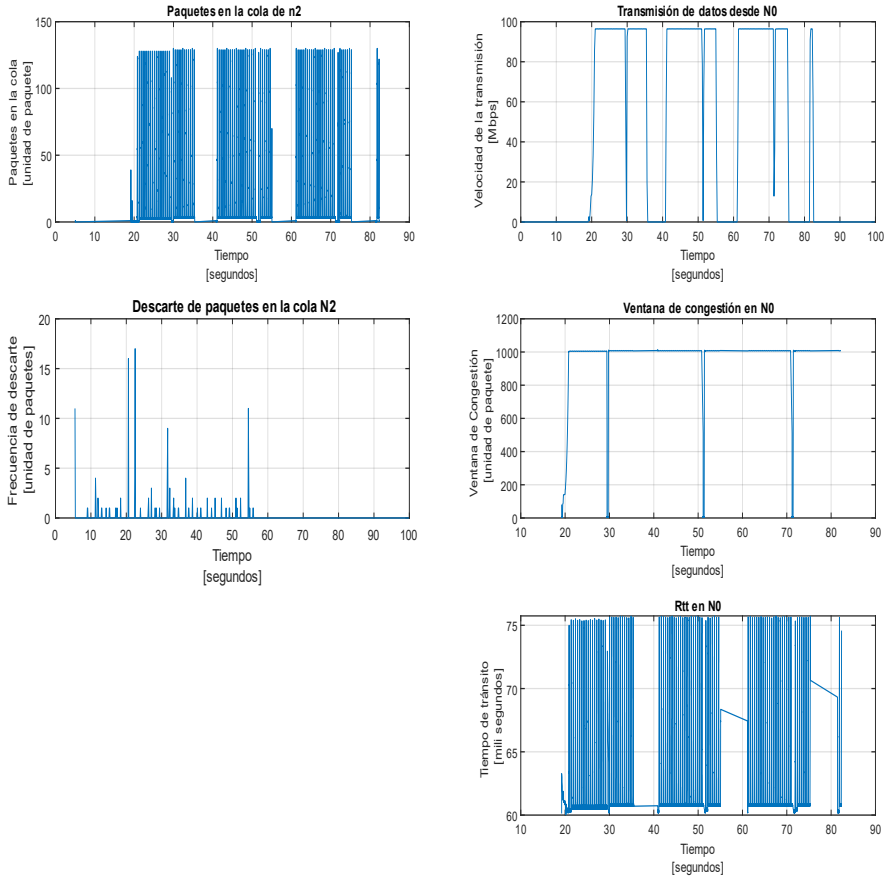

Figura 23 1672749563codigo_v150v

Simulación 14

Tabla 18 Simulación 14

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
14	DCTCP	-	fifo droptail	1Gbps	100Mbps	Uniforme	2871

Min_U	Max_U
7	15

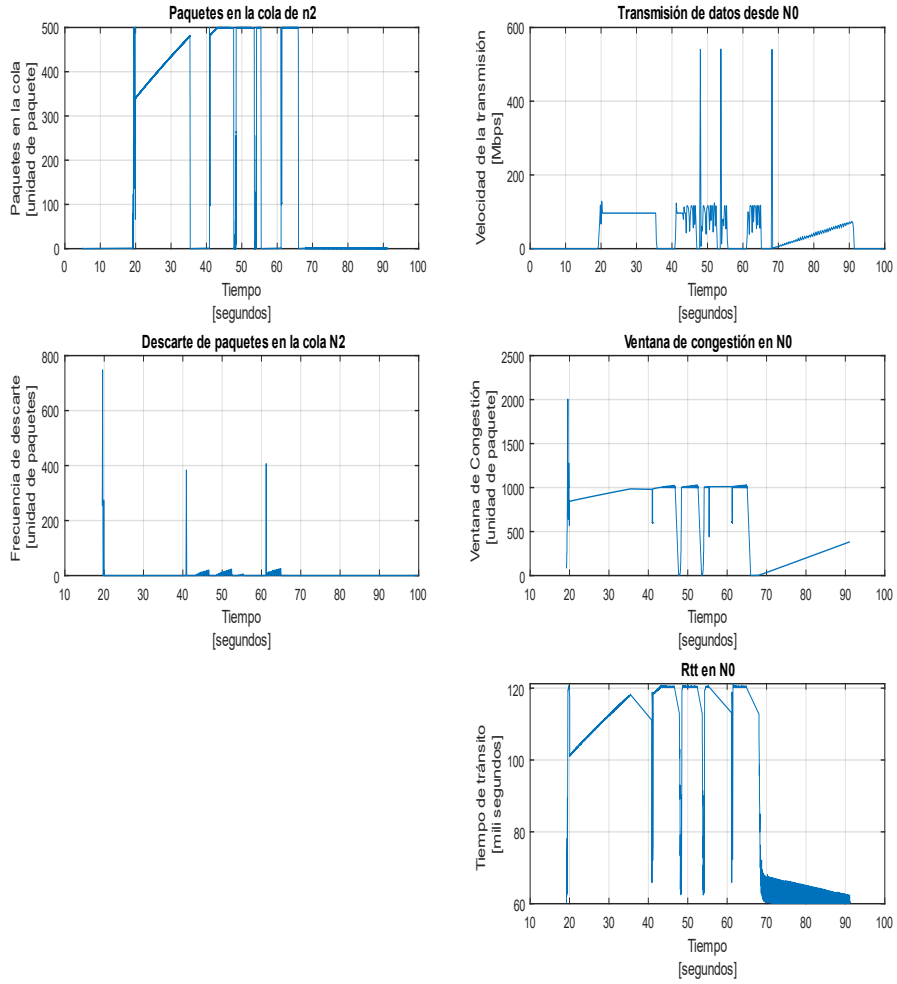

Figura 24 1672749738codigo_v150v

Simulación 15

Tabla 19 Simulación 15

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
15	reno	-	fifo droptail	1Gbps	100Mbps	Uniforme	732

Min_U	Max_U
7	15

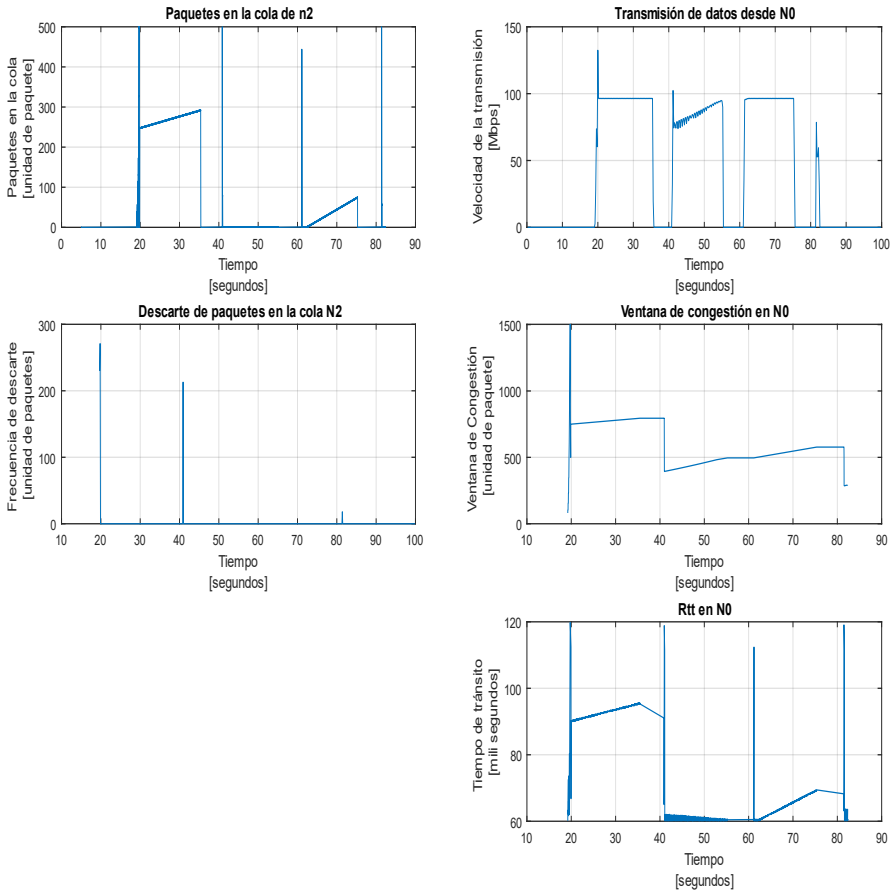

Figura 25 1672750412codigo_v150v

Simulación 16

Tabla 20 Simulación 16

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
16	BIC	-	fifo droptail	1Gbps	100Mbps	Uniforme	1332

Min_U	Max_U
7	15

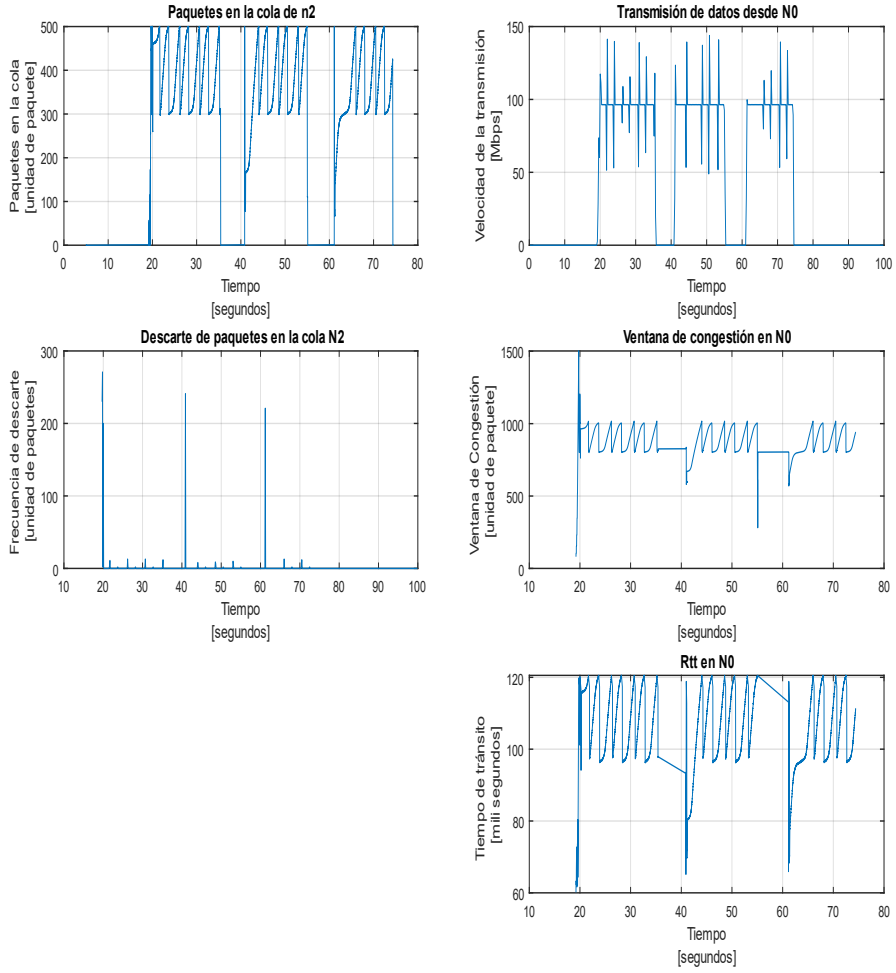

Figura 26 1672750573codigo_v150v

Simulación 17

Tabla 21 Simulación 17

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
17	BBR	-	fqcodel	1Gbps	100Mbps	Uniforme	0

Min_U	Max_U
7	15

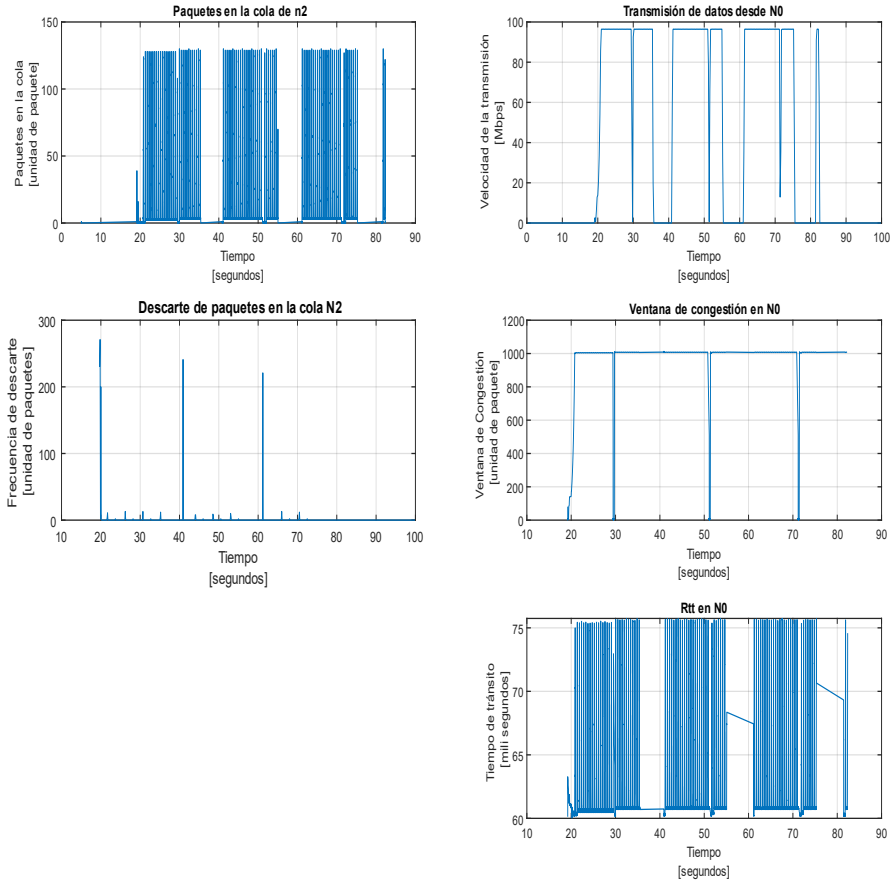

Figura 27 1672750761codigo_v150v

Simulación 18

Tabla 22 Simulación 18

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
18	DCTCP	-	fqcodel	1Gbps	100Mbps	Uniforme	1824

Min_U	Max_U
7	15

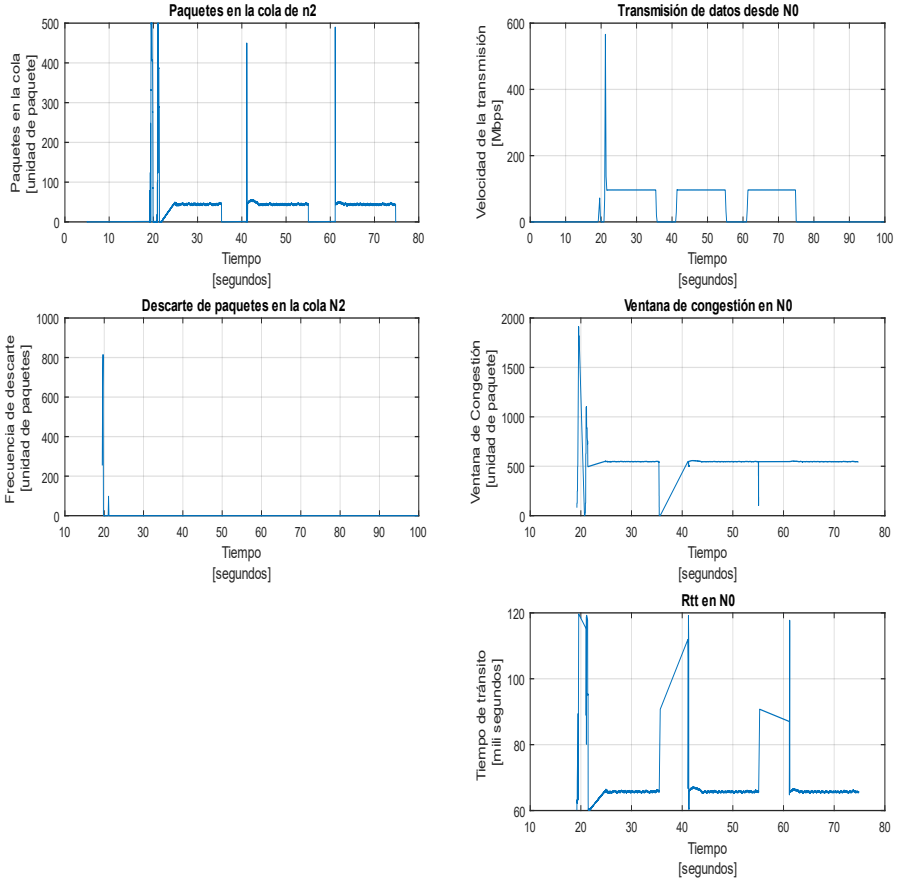

Figura 28 1672751042codigo_v150v

Simulación 19

Tabla 23 Simulación 19

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
19	reno	-	fqcodel	1Gbps	100Mbps	Uniforme	235

Min_U	Max_U
7	15

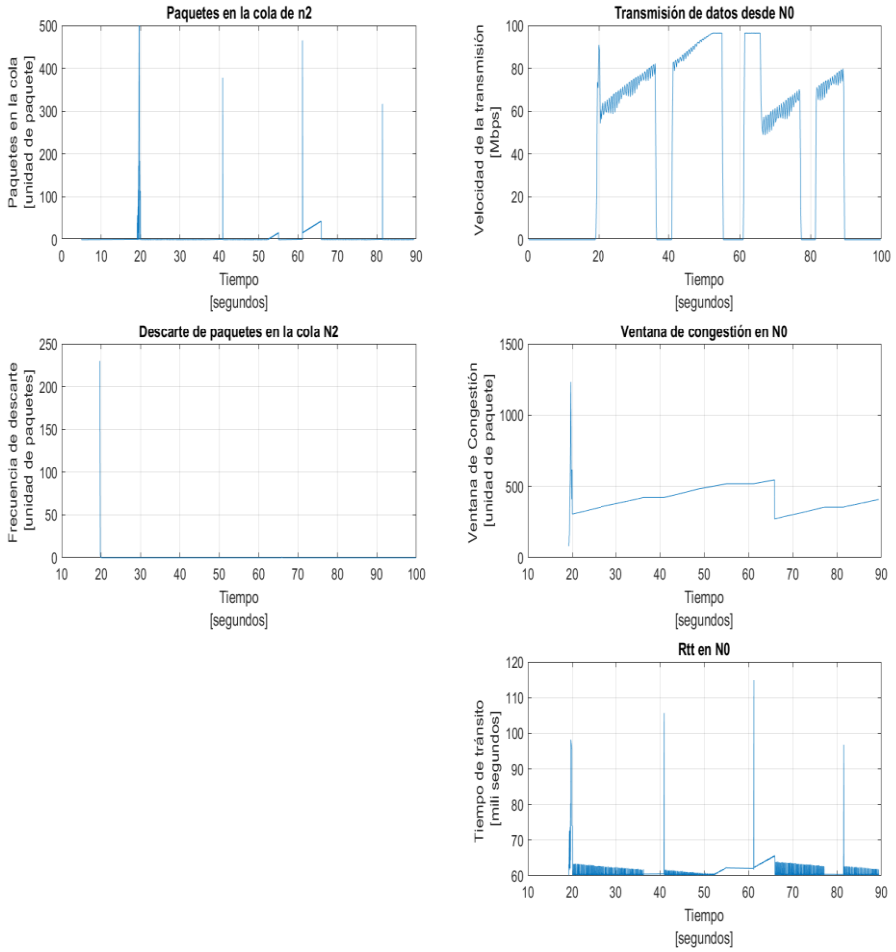

Figura 29 1672751435codigo_v150v

Simulación 20

Tabla 24 Simulación 20

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
20	BIC	-	fqcodel	1Gbps	100Mbps	Uniforme	352

Min_U	Max_U
7	15

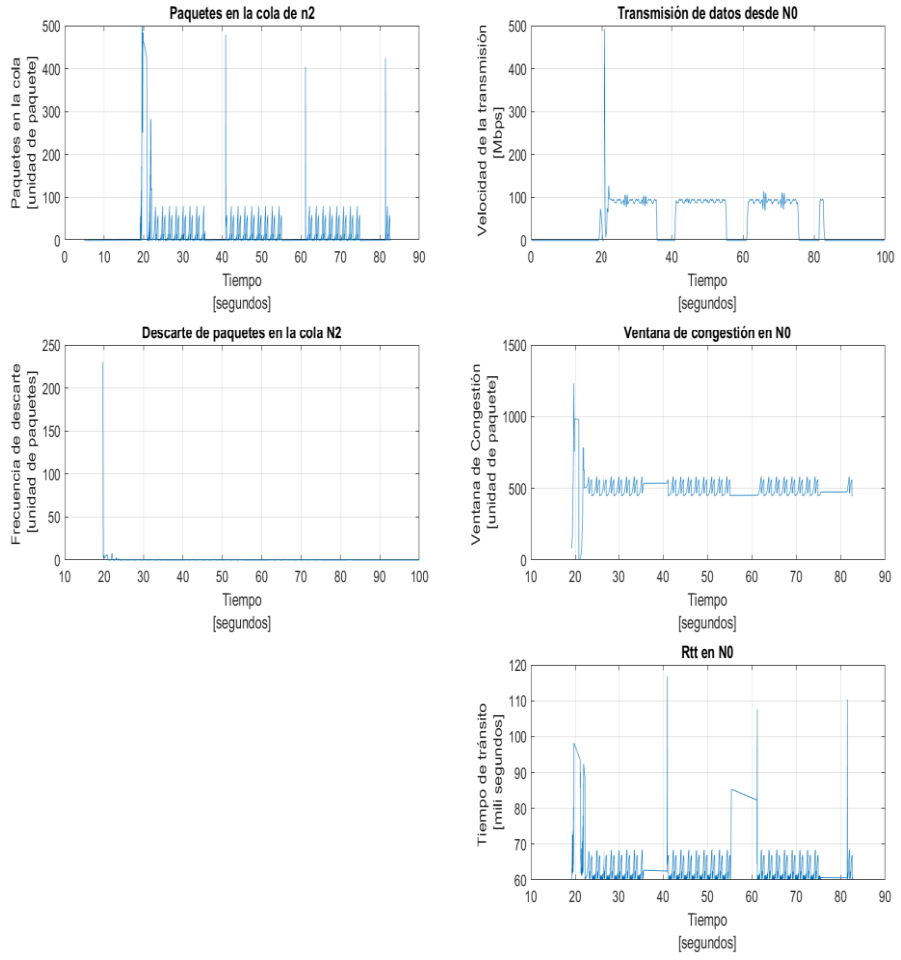

Figura 30 1672751696codigo_v150v

Simulación 21

Tabla 25 Simulación 21

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
21	BBR	-	RED	1Gbps	100Mbps	Uniforme	12064

Min_U	Max_U
7	15

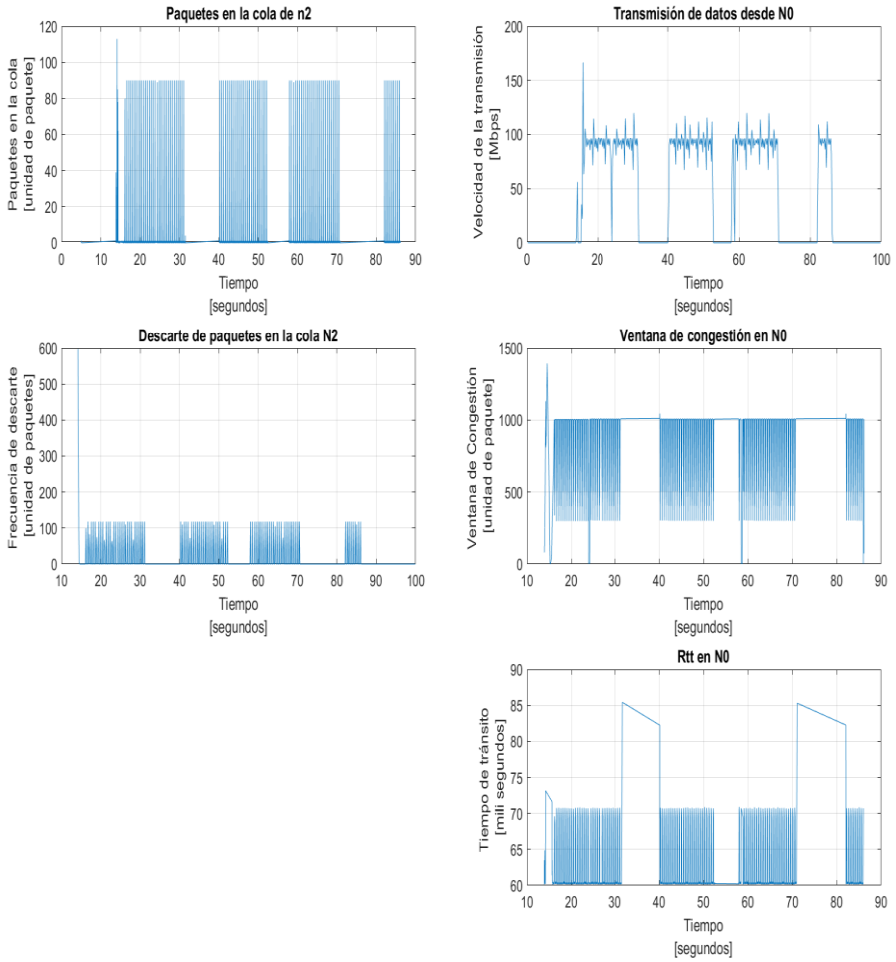

Figura 31 1672751898codigo_v150v

Simulación 22

Tabla 26 Simulación 22

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
22	DCTCP	-	RED	1Gbps	100Mbps	Uniforme	39227

Min_U	Max_U
7	15

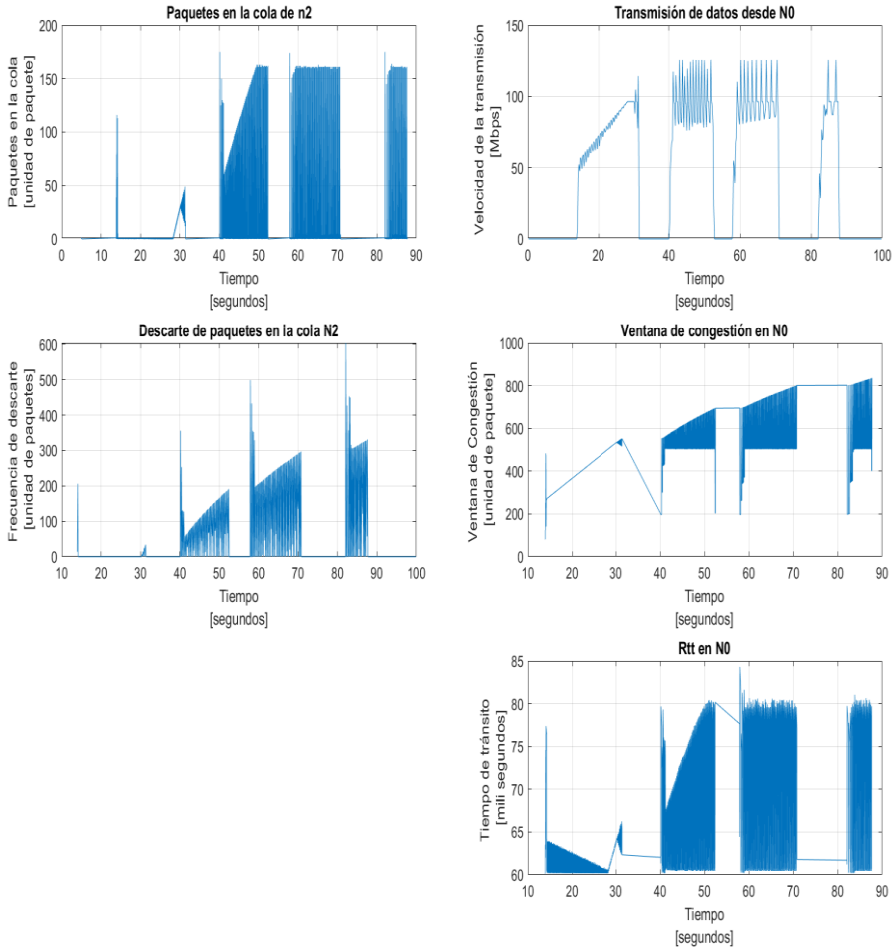

Figura 32 1672752132codigo_v150v

Simulación 23

Tabla 27 Simulación 23

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
23	reno	-	RED	1Gbps	100Mbps	Uniforme	395

Min_U	Max_U
7	15

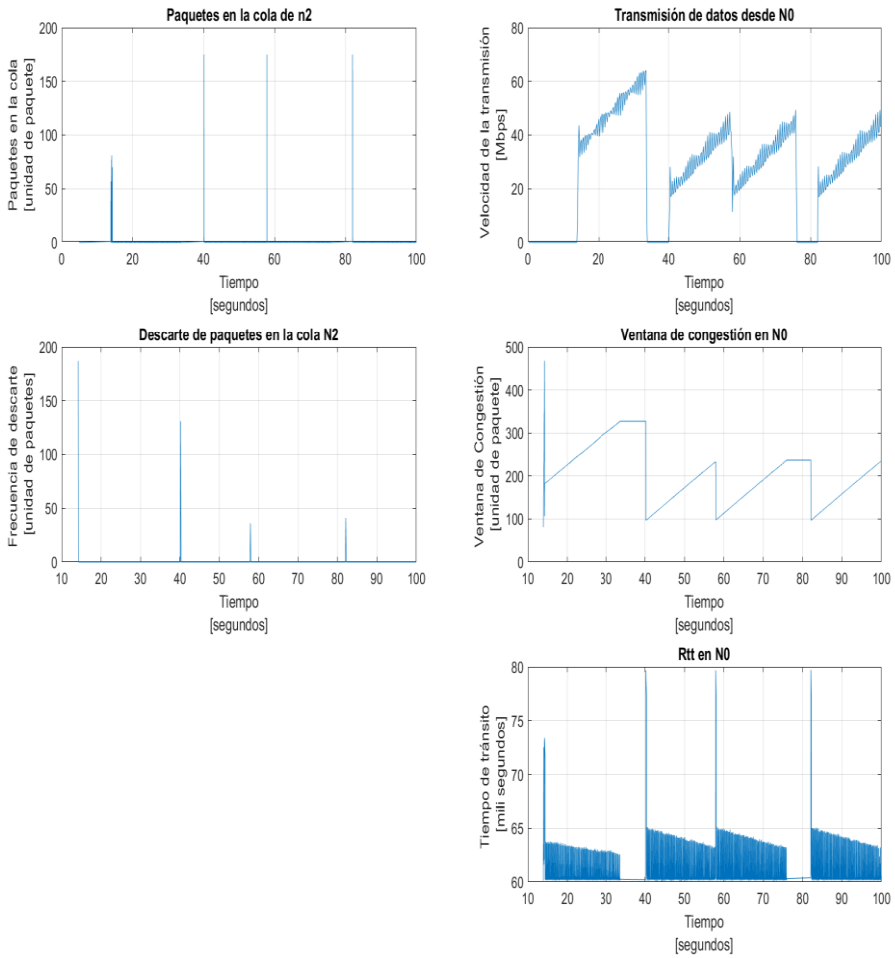

Figura 33 1672752336codigo_v150v

Simulación 24

Tabla 28 Simulación 24

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
24	BIC	-	RED	1Gbps	100Mbps	Uniforme	2464

Min_U	Max_U
7	15

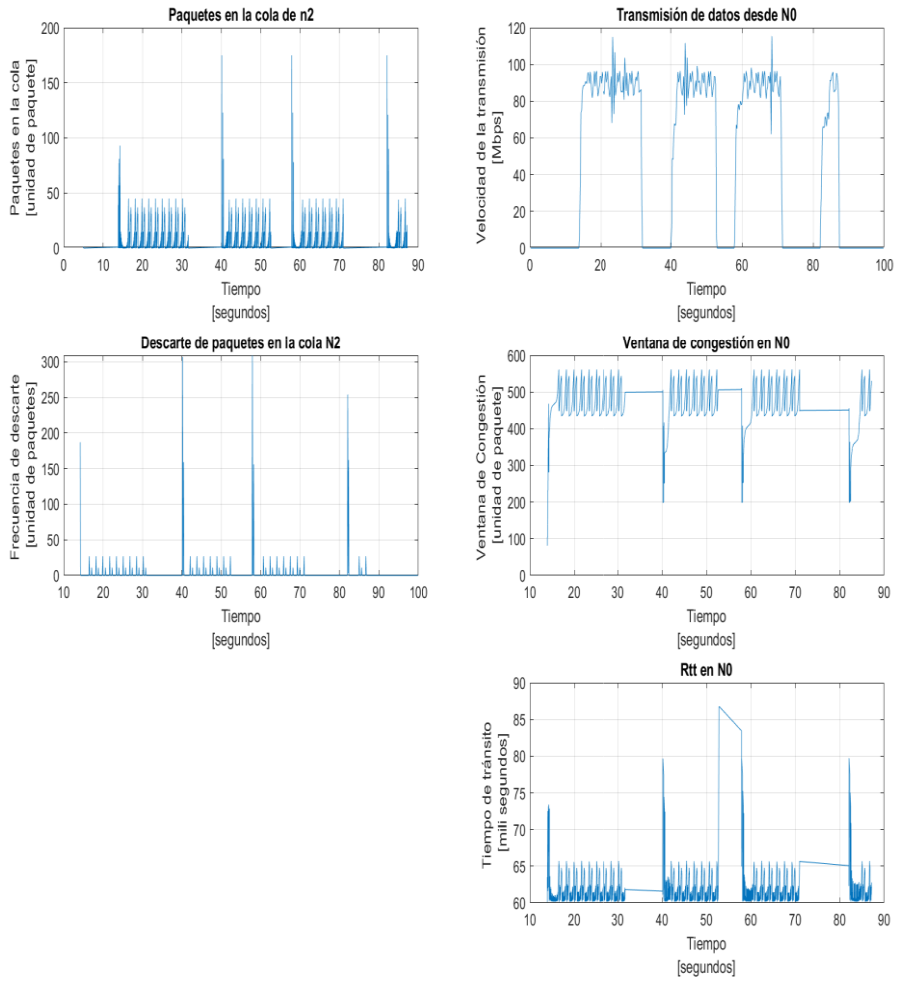

Figura 34 1672752505codigo_v150v

Simulación 25

Tabla 29 Simulación 25

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
25	BBR	-	fifo droptail	1Gbps	100Mbps	Triangular	0

Min_U	Max_U	Media
7	15	10

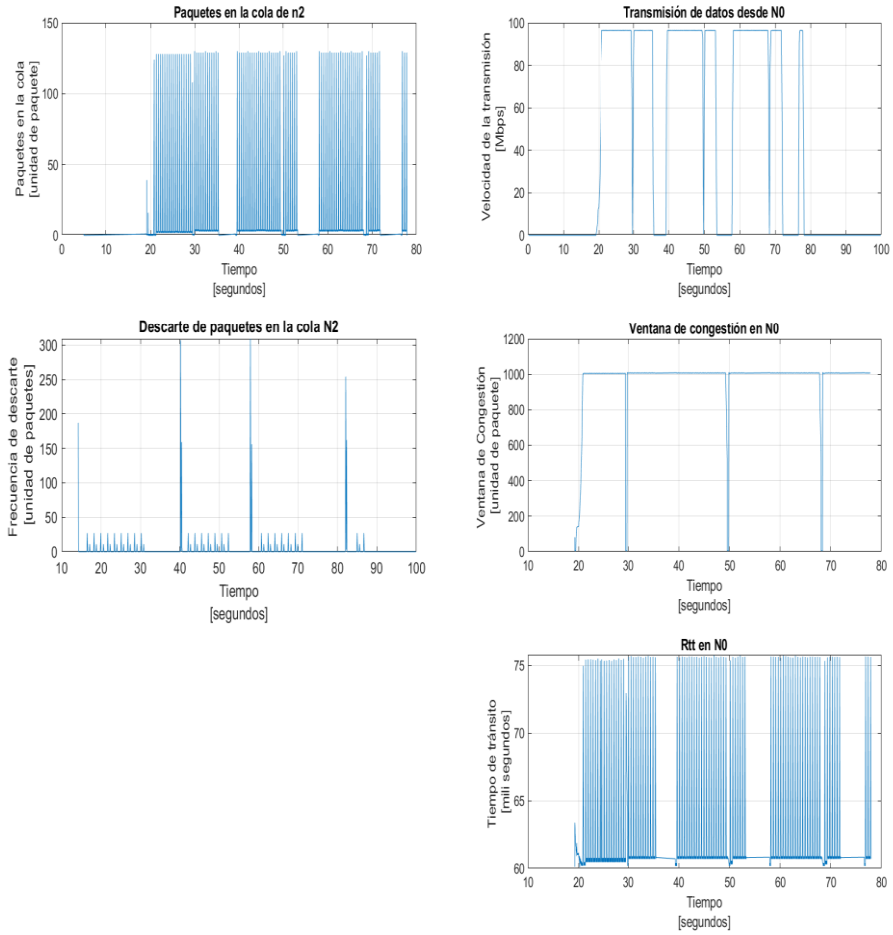

Figura 35 1672753230codigo_v150w

Simulación 26

Tabla 30 Simulación 26

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
26	DCTCP	-	fifo droptail	1Gbps	100Mbps	Triangular	2675

Min_U	Max_U	Media
7	15	10

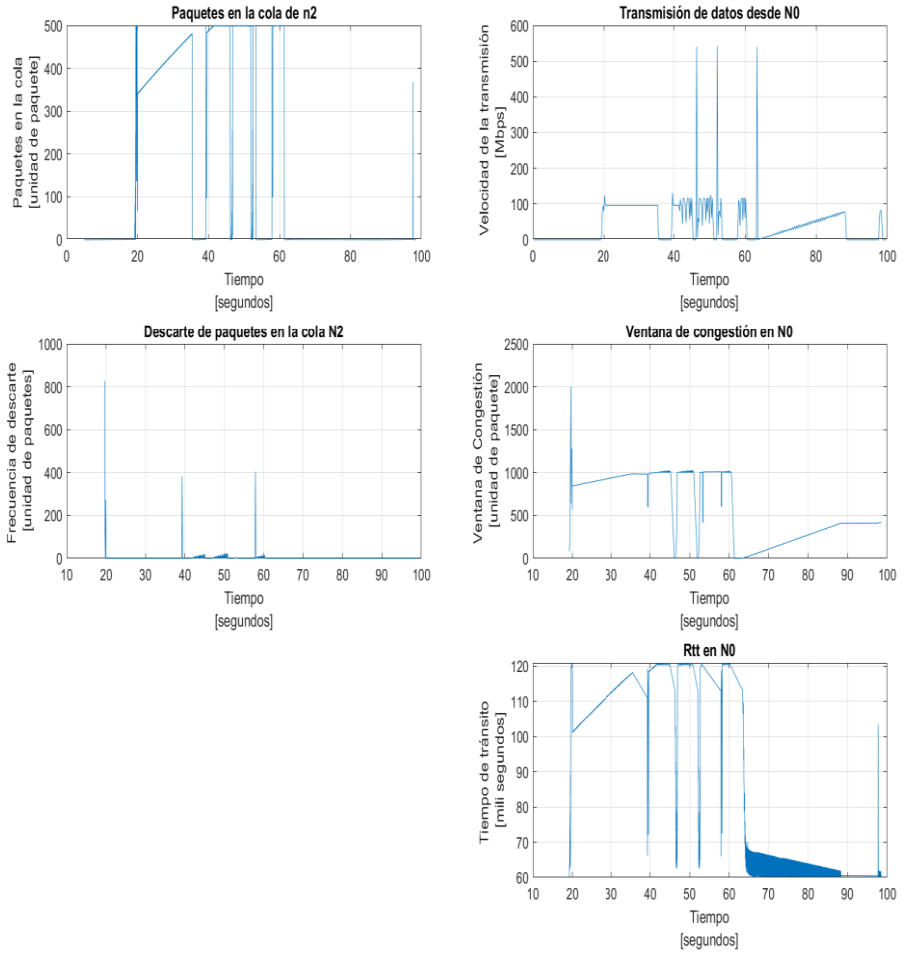

Figura 36 1672762257codigo_v150w

Simulación 27

Tabla 31 Simulación 27

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
27	reno	-	fifo droptail	1Gbps	100Mbps	Triangular	727

Min_U	Max_U	Media
7	15	10

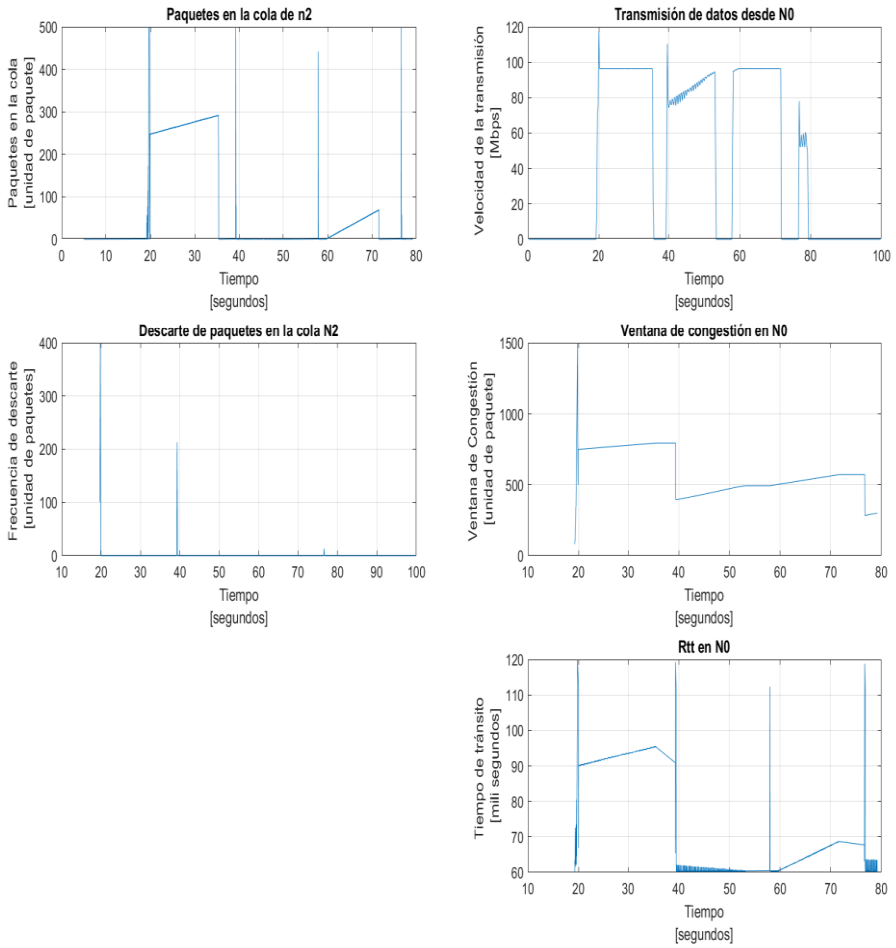

Figura 37 1672762501codigo_v150w

Simulación 28

Tabla 32 Simulación 28

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
28	BIC	-	fifo droptail	1Gbps	100Mbps	Triangular	1515

Min_U	Max_U	Media
7	15	10

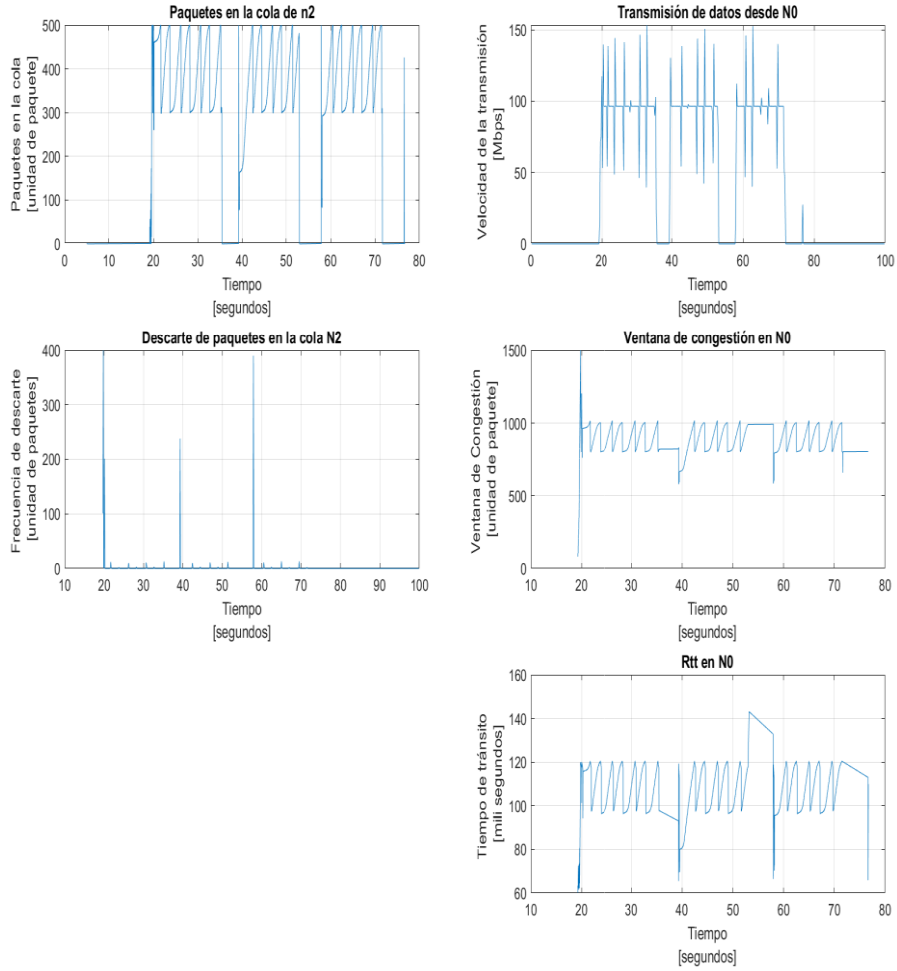

Figura 38 1672762683codigo_v150w

Simulación 29

Tabla 33 Simulación 29

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
29	BBR	-	fqcodel	1Gbps	100Mbps	Triangular	0

Min_U	Max_U	Media
7	15	10

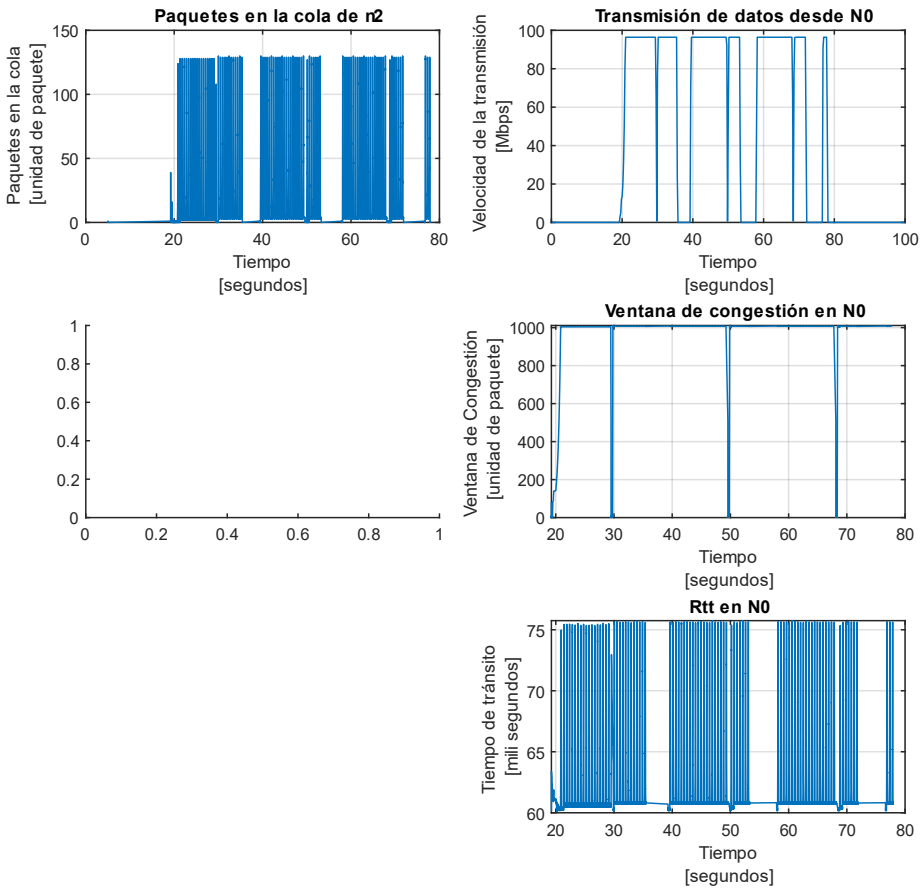

FIgura 39 1672762941codigo_v150w

Simulación 30

Tabla 34 Simulación 30

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
30	DCTCP	-	fqcodel	1Gbps	100Mbps	Triangular	1824

Min_U	Max_U	Media
7	15	10

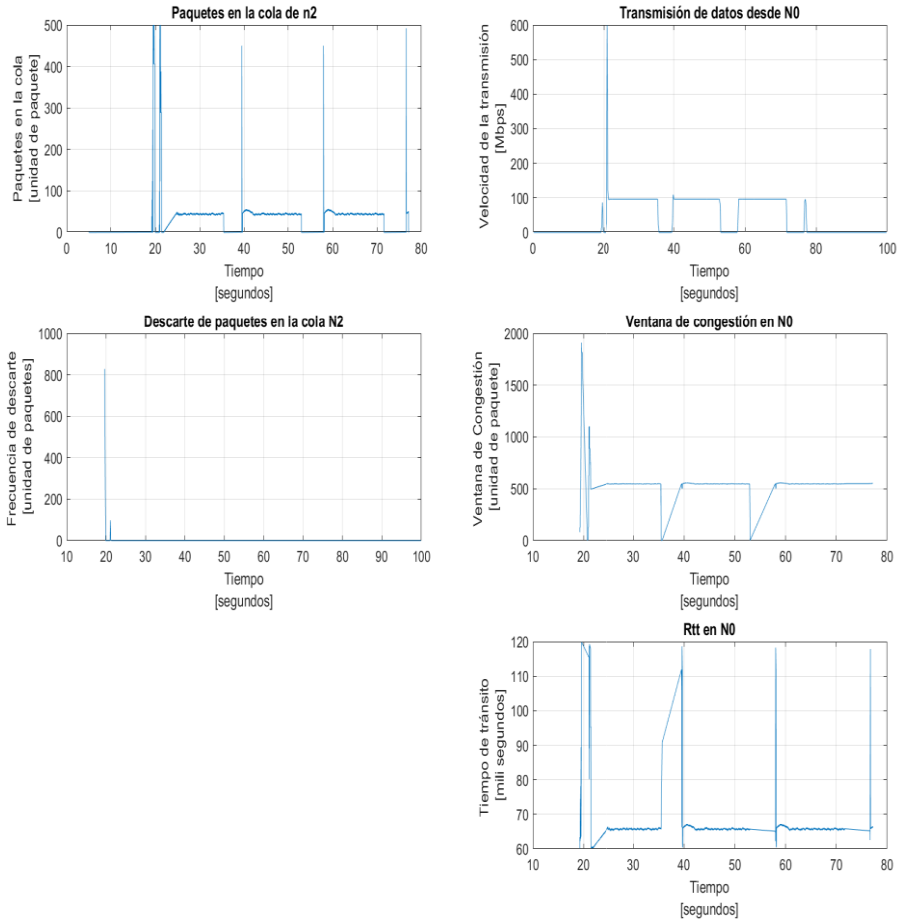

Figura 40 1672763173codigo_v150w

Simulación 31

Tabla 35 Simulación 31

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
31	reno	-	fqcodel	1Gbps	100Mbps	Triangular	235

Min_U	Max_U	Media
7	15	10

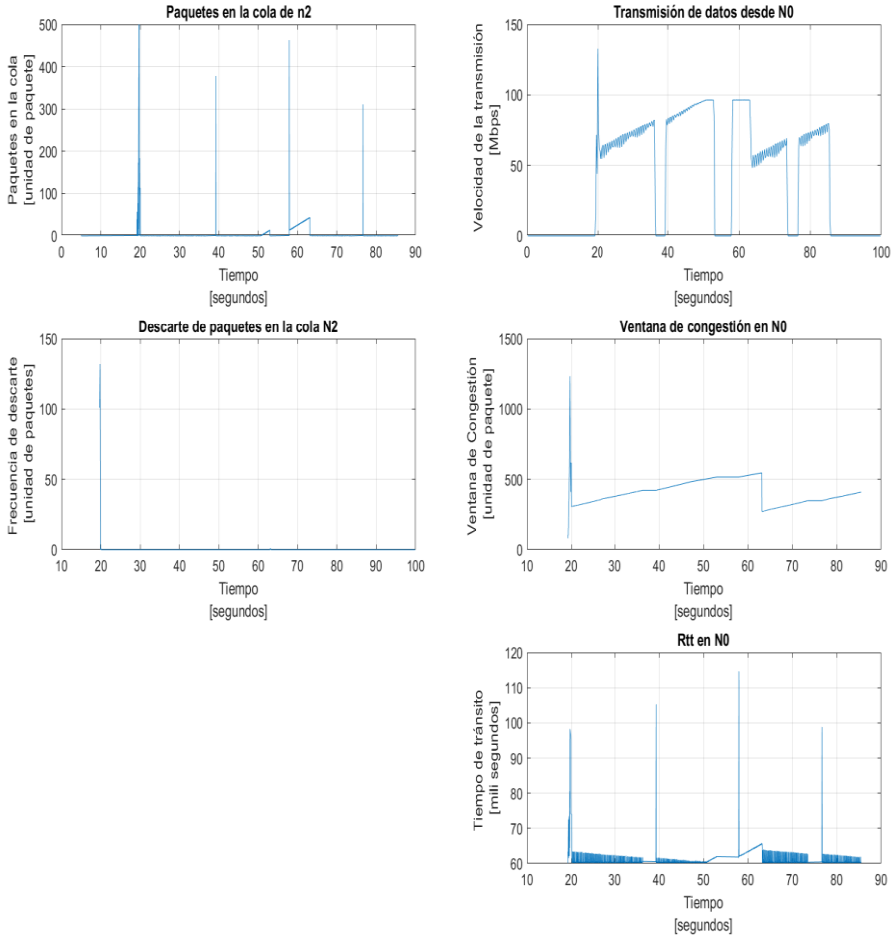

Figura 41 1672763527codigo_v150w

Simulación 32

Tabla 36 Simulación 32

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
32	BIC	-	fqcodel	1Gbps	100Mbps	Triangular	352

Min_U	Max_U	Media
7	15	10

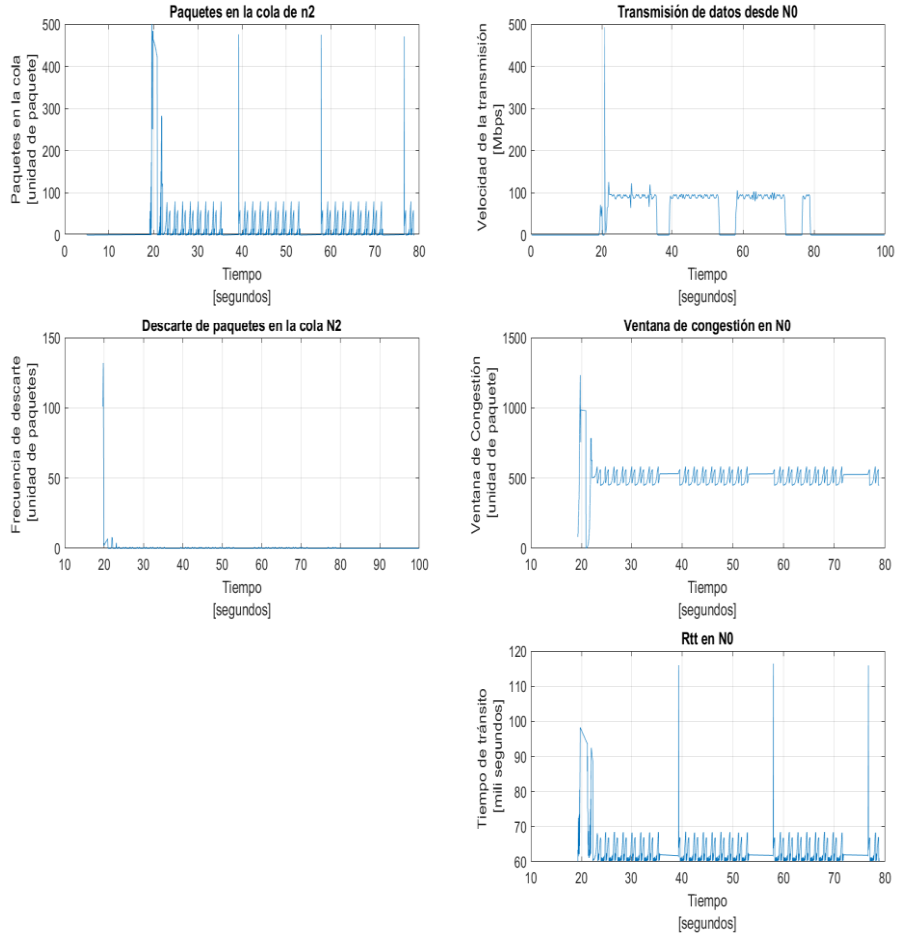

Figura 42 1672763758codigo_v150w

Simulación 33

Tabla 37 Simulación 33

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
33	BBR	-	RED	1Gbps	100Mbps	Triangular	11946

Min_U	Max_U	Media
7	15	10

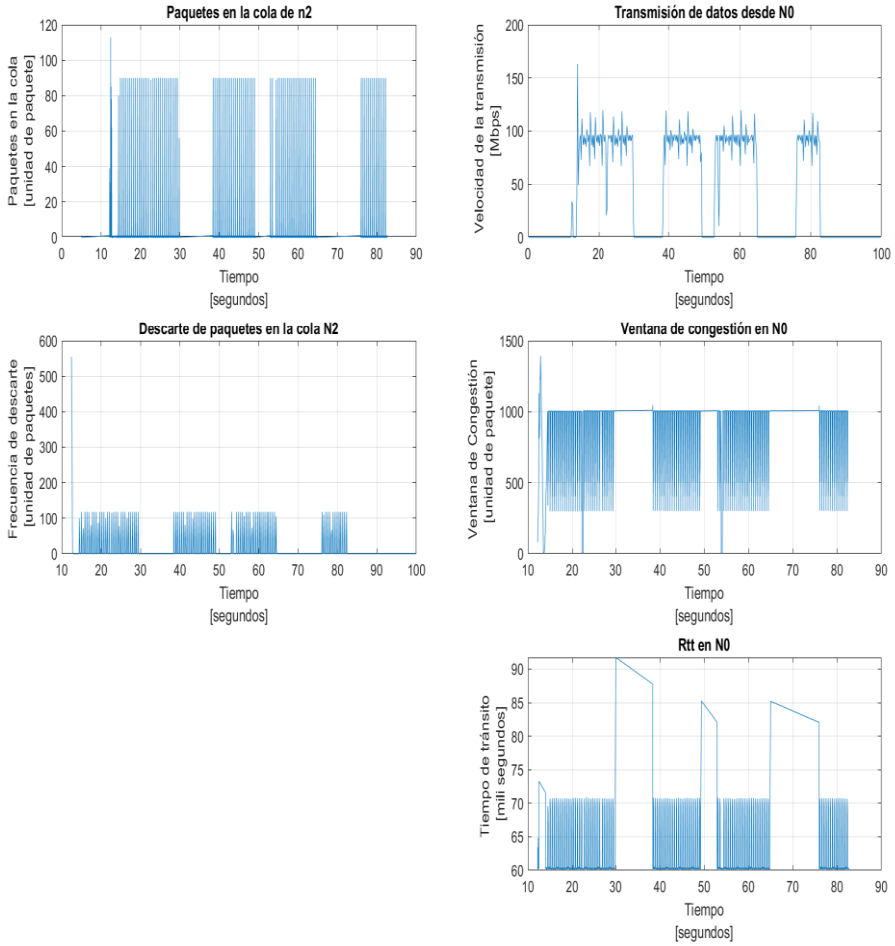

Figura 43 1672763991codigo_v150w

Simulación 34

Tabla 38 Simulación 34

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
34	DCTCP	-	RED	1Gbps	100Mbps	Triangular	38420

Min_U	Max_U	Media
7	15	10

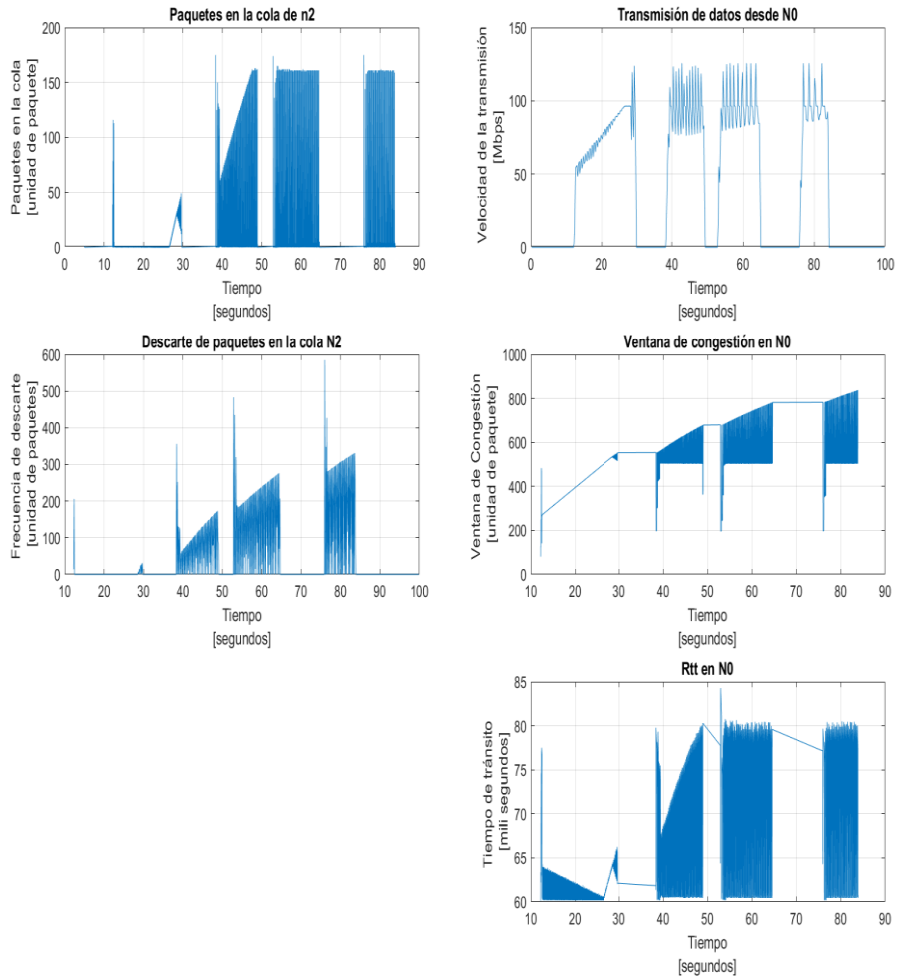

Figura 44 1672765711codigo_v150w

Simulación 35

Tabla 39 Simulación 35

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
35	reno	-	RED	1Gbps	100Mbps	Triangular	437

Min_U	Max_U	Media
7	15	10

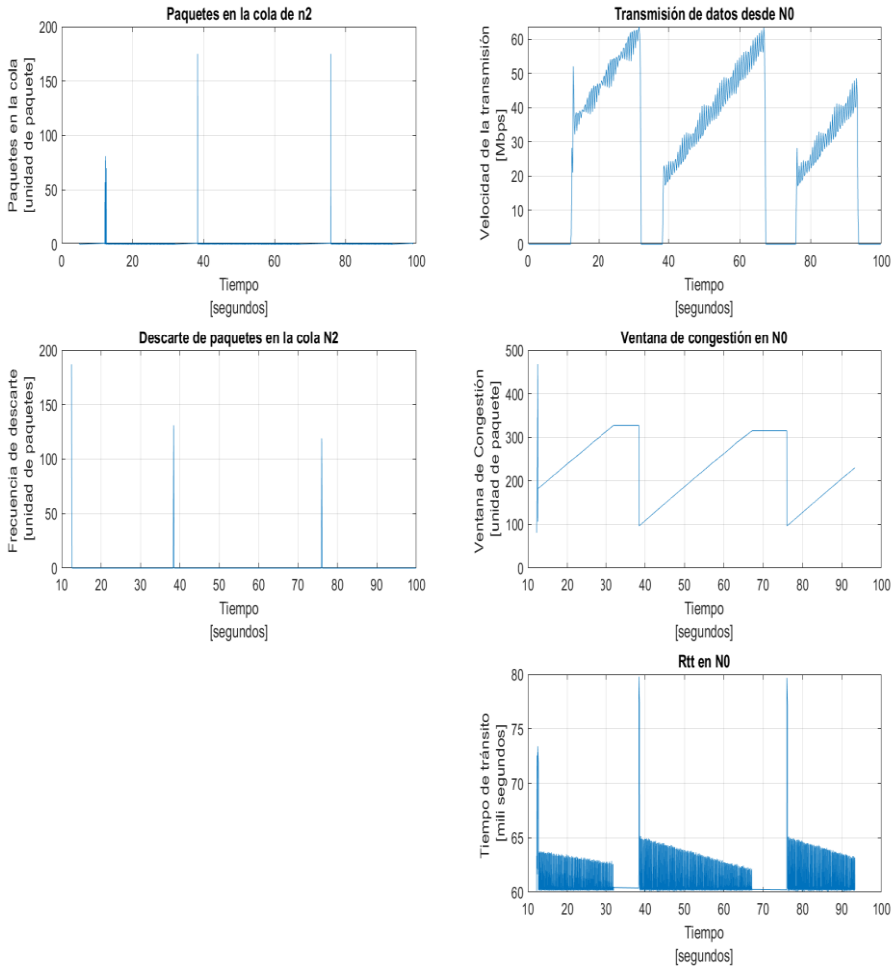

Figura 45 1672765969codigo_v150w

Simulación 36

Tabla 40 Simulación 36

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paquetes descartados
36	BIC	-	RED	1Gbps	100Mbps	Triangular	2468

Min_U	Max_U	Media
7	15	10

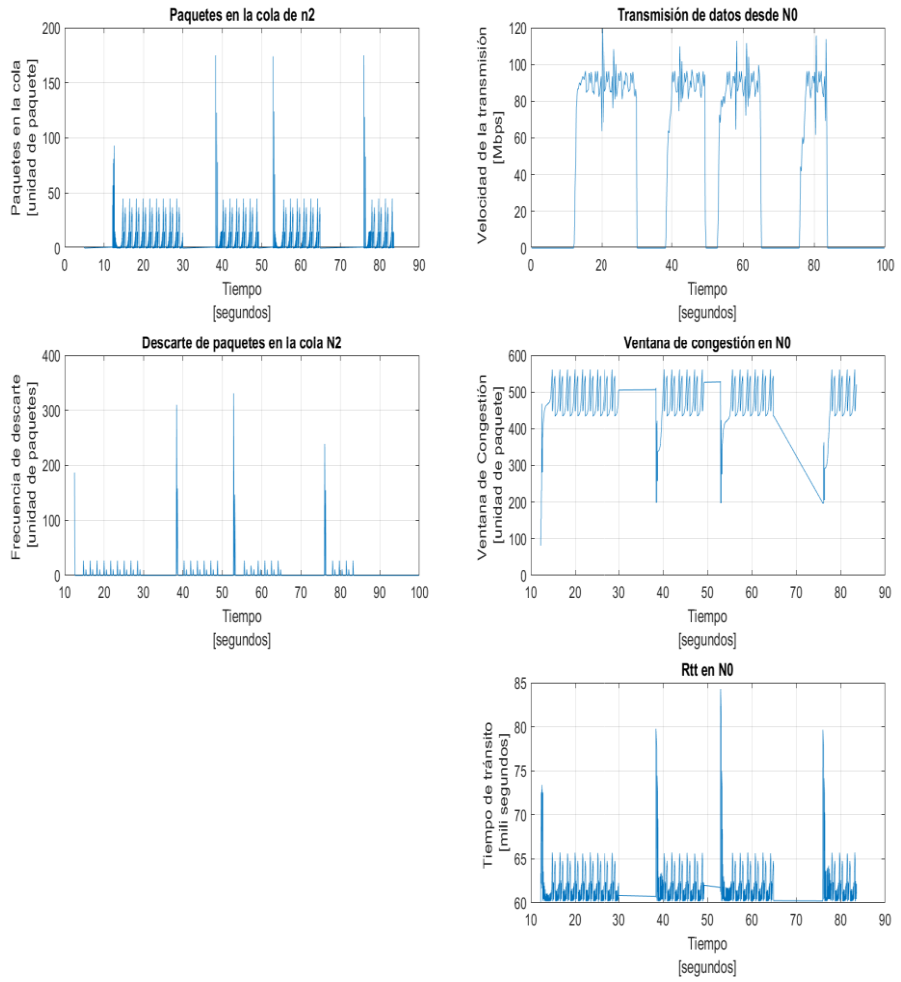

Figura 46 1672766145codigo_v150w

Simulación 37

Tabla 41 Simulación 37

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq Desc N0	Paq n1
37	BBR	BBR	fifo droptail	1Gbps	100 Mbps	Uniforme	823	824

Min_U	Max_U
0	1

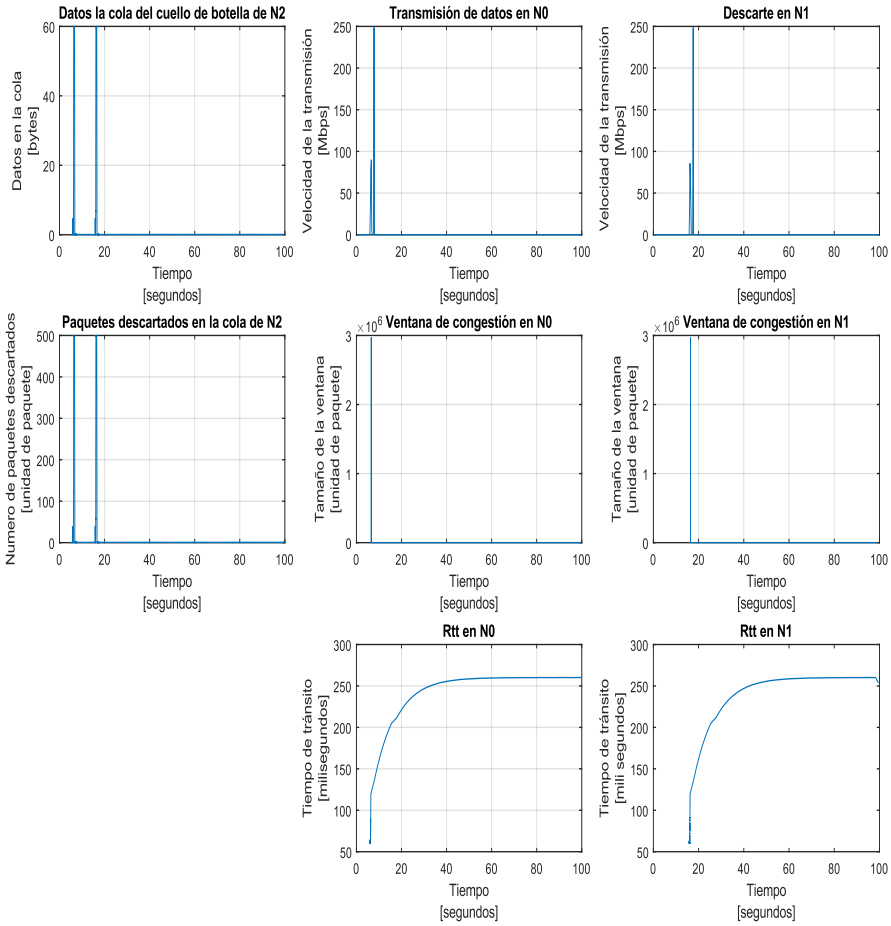

Figura 47 1672768754codigo_v150v

Simulación 38

Tabla 42 Simulación 38

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
38	BBR	BBR	fqcodel	1Gbps	100 Mbps	Uniforme	1389	2313

Min_U	Max_U
0	1

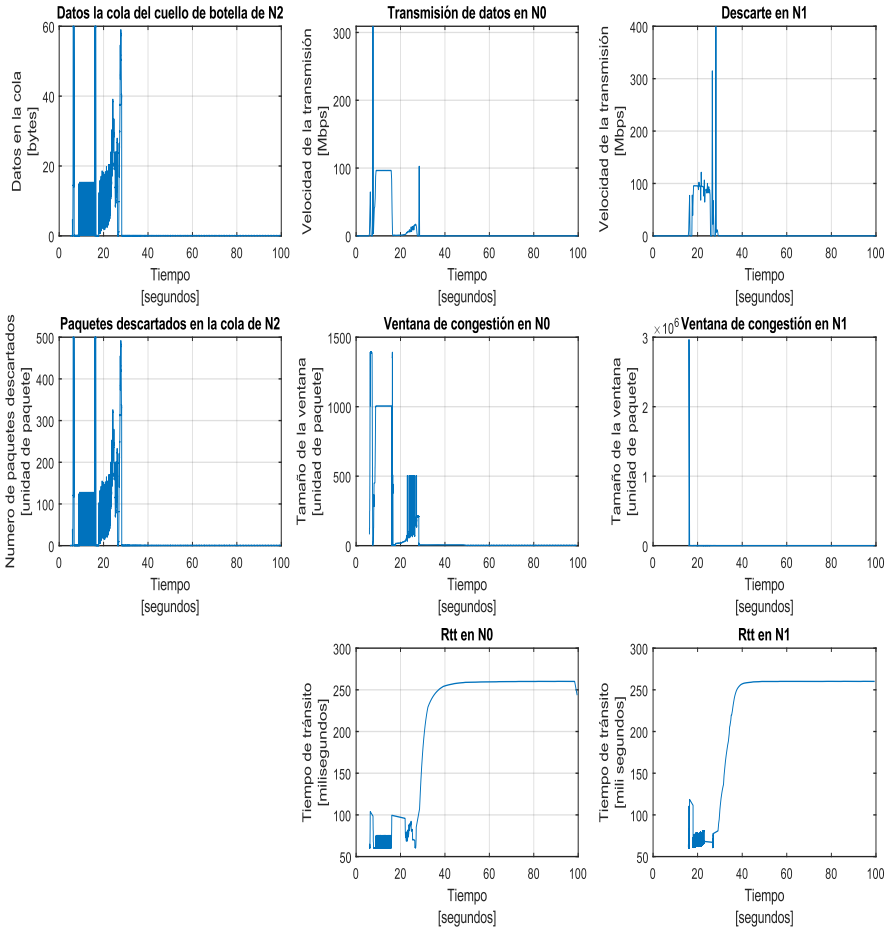

Figura 48 1672769014codigo_v150v

Simulación 39

Tabla 43 Simulación 39

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
39	BBR	BBR	red	1Gbps	100 Mbps	Uniforme	4658	12638

Min_U	Max_U
0	1

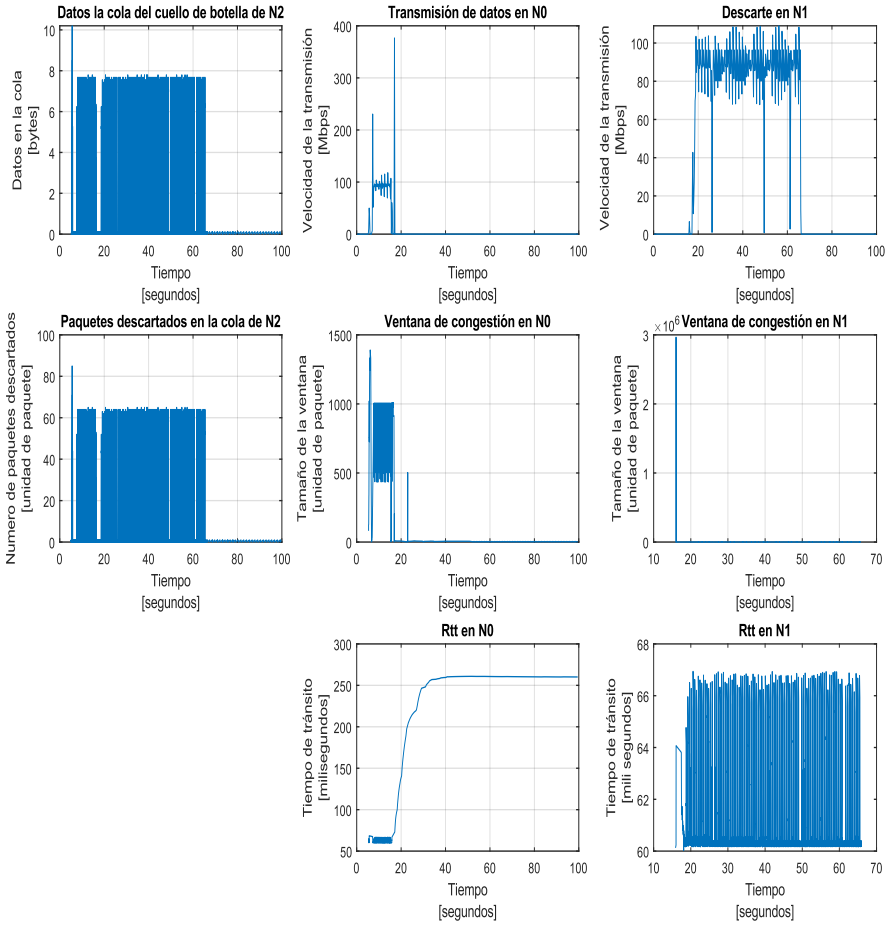

Figura 49 1672769429codigo_v150v

Simulación 40

Tabla 44 Simulación 40

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
40	BBR	BBR	Fifo droptail	1Gbps	100 Mbps	Uniforme	867	837

Min_U	Max_U
7	15

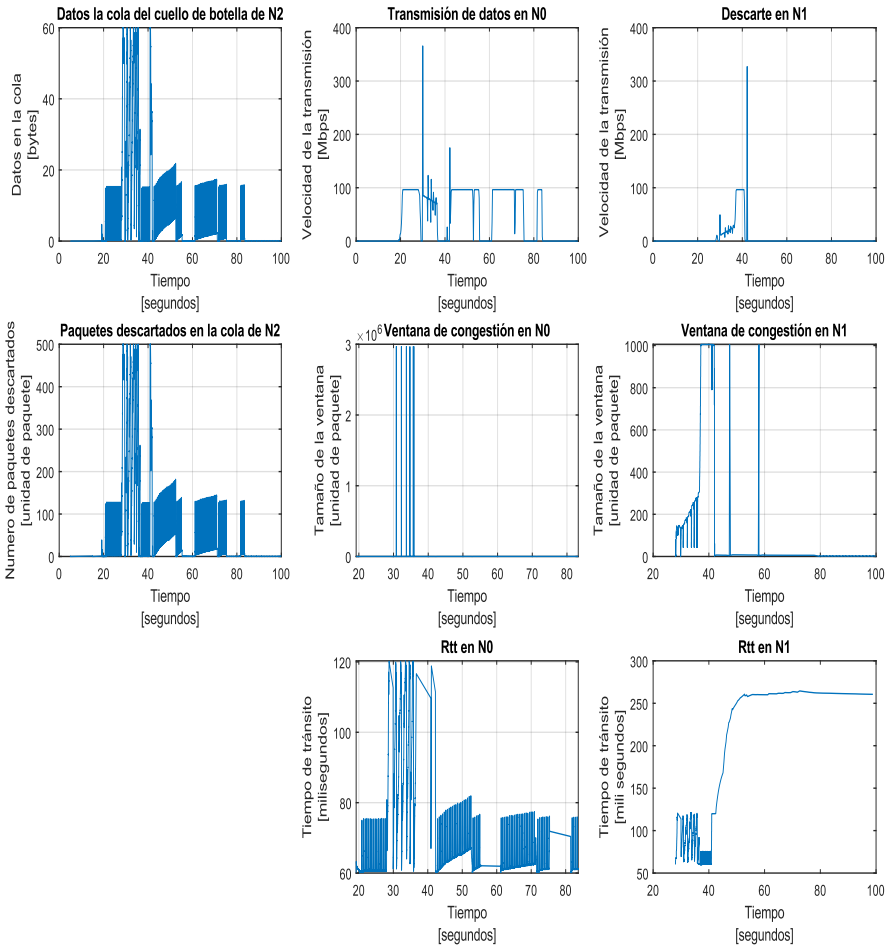

Figura 50 1672770283codigo_v150v

Simulación 41

Tabla 45 Simulación 41

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
41	BBR	BBR	fqcodel	1Gbps	100 Mbps	Uniforme	260	95

Min_U	Max_U
7	15

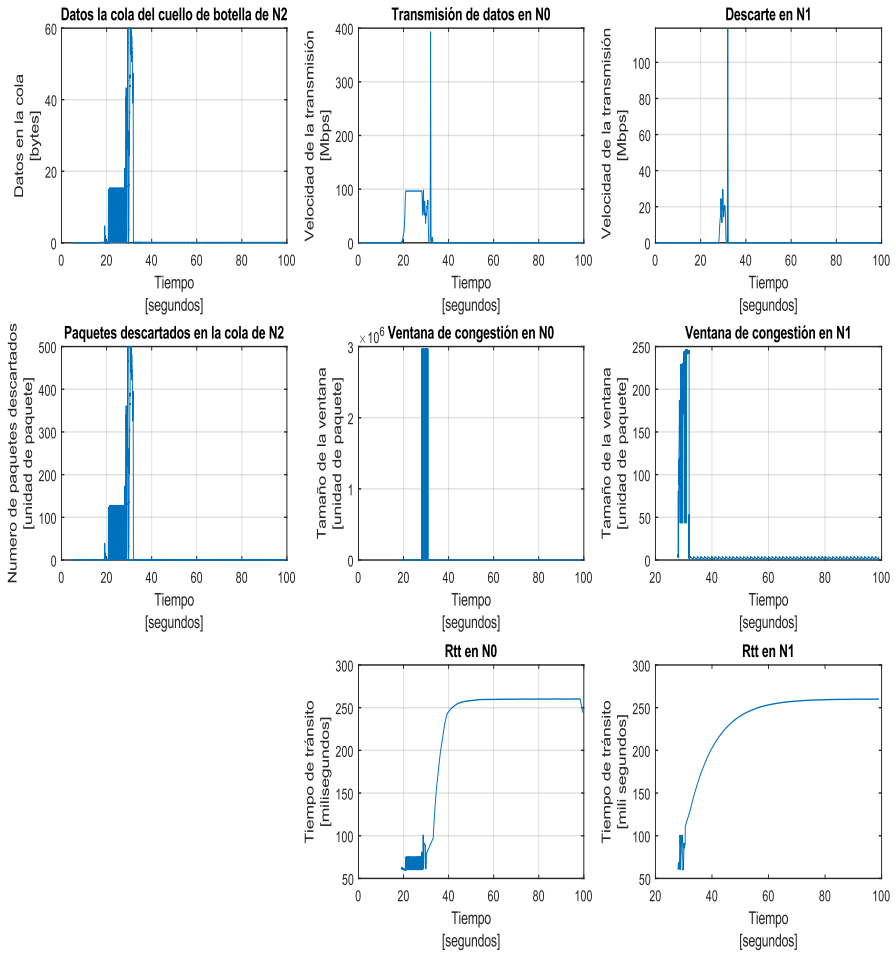

Figura 51 1672770543codigo_v150v

Simulación 42

Tabla 46 Simulación 42

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
	BBR	BBR	fqcodel	1Gbps (3p)	100 Mbps	Uniforme	588	433

Min_U	Max_U
7	15

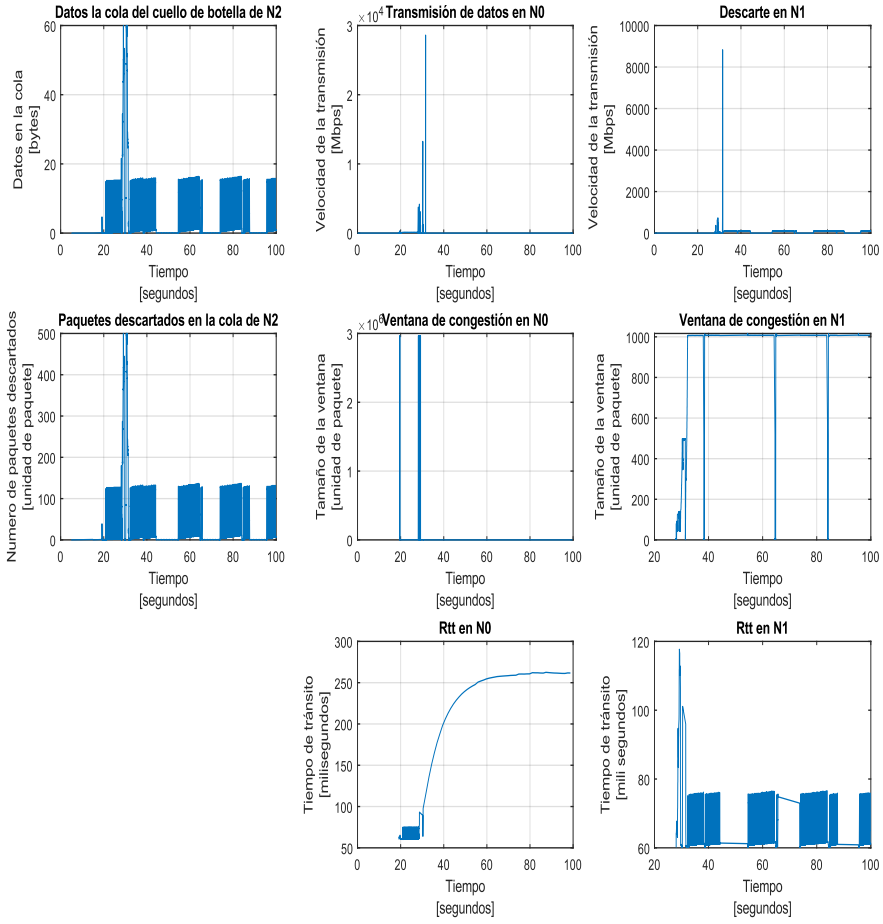

Figura 52 1672771352codigo_v150v

Simulación 43

Tabla 47 Simulación 43

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
	BBR	BBR	fqcodel	1Gbps (3p) dinamyc	100 Mbps	Uniforme	588	433

Min_U	Max_U
7	15

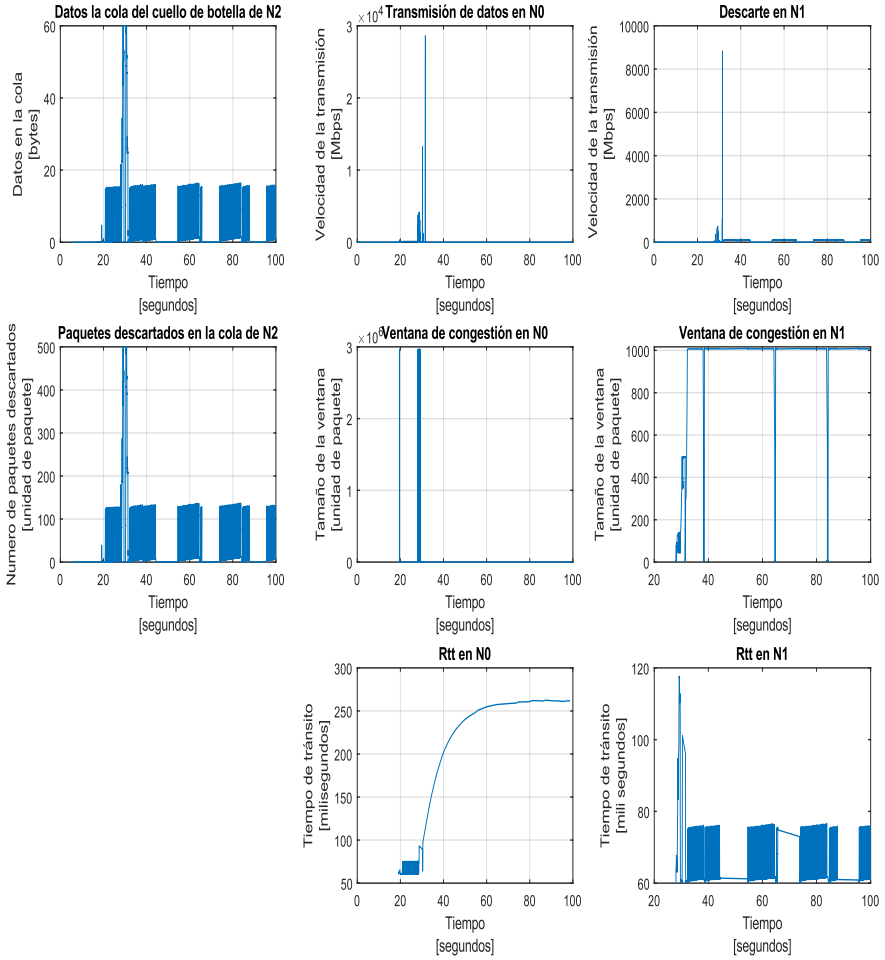

Figura 53 1672771674codigo_v150x

Simulación 44

Tabla 48 Simulación 44

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
42	BBR	BBR	red	1Gbps	100 Mbps	Uniforme	5110	59

Min_U	Max_U
7	15

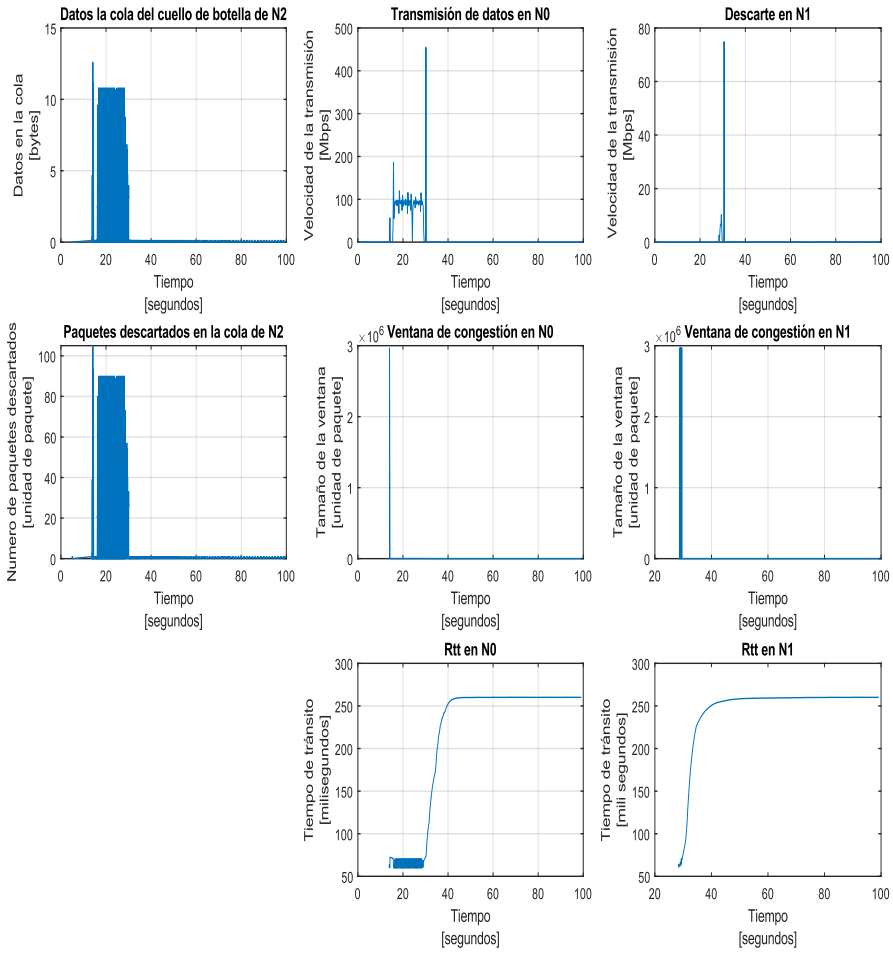

Figura 54 1672772299codigo_v150v

Simulación 45

Tabla 49 Simulación 45

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
43	BBR	BBR	Fifo droptail	1Gbps	100 Mbps	Triangular	130	123

Min_U	Max_U	Media
7	15	10

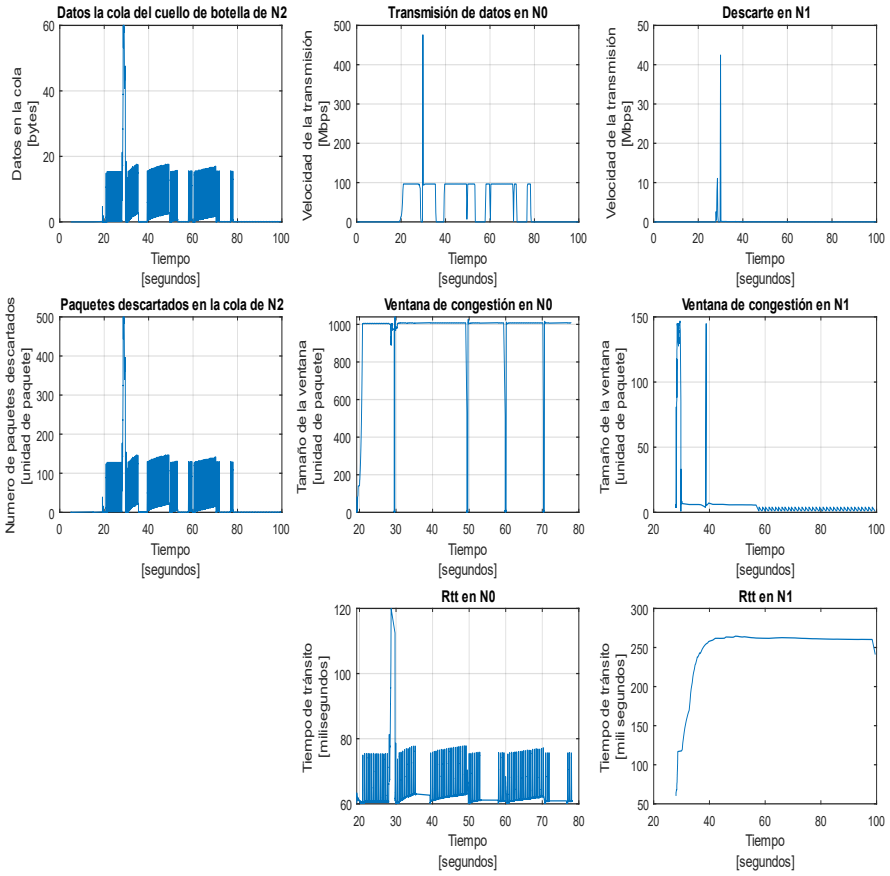

Figura 55 1672772823codigo_v150w

Simulación 46

Tabla 50 Simulación 46

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
44	BBR	BBR	fqcodel	1Gbps	100 Mbps	Tirangular	154	121

Min_U	Max_U	Media
7	15	10

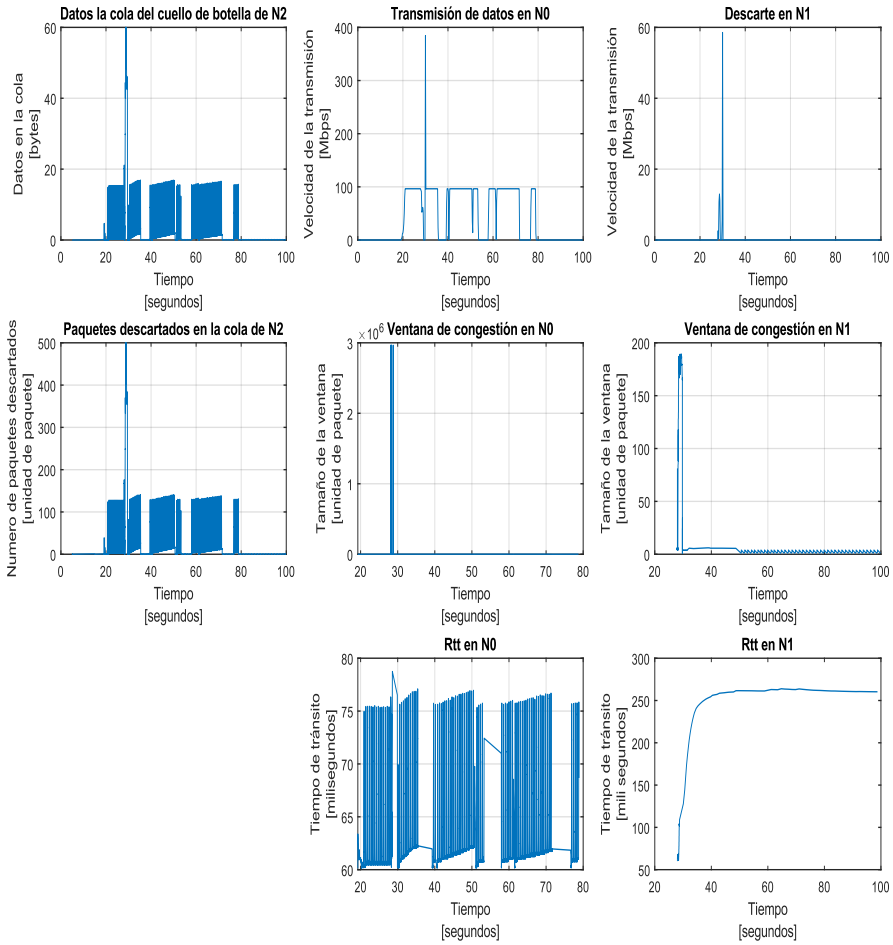

Figura 56 1672773035codigo_v150w

Simulación 47

Tabla 51 Simulación 47

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
45	BBR	BBR	red	1Gbps	100 Mbps	Triangular	6026	93

Min_U	Max_U	Media
7	15	10

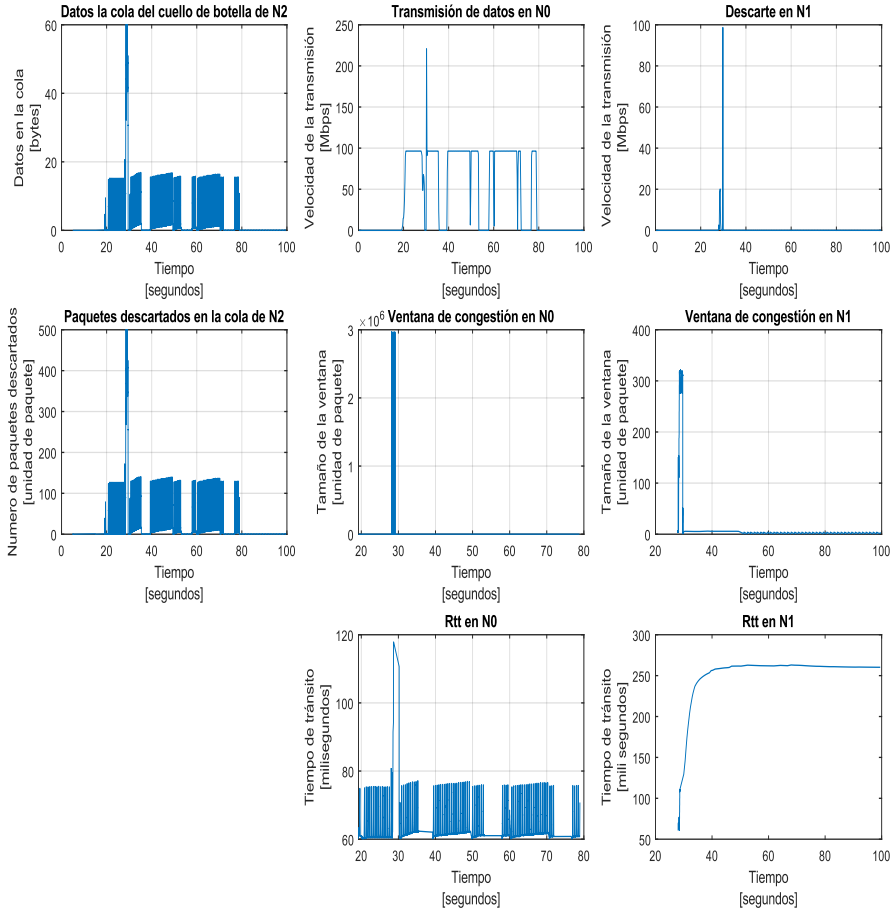

Figura 57 1672773205codigo_v150w

Simulación 48

Tabla 52 Simulación 48

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
46	BBR	reno	fqcodel	1 Gbps	100 Mbps	Uniforme	185	7

Min_U	Max_U
7	15

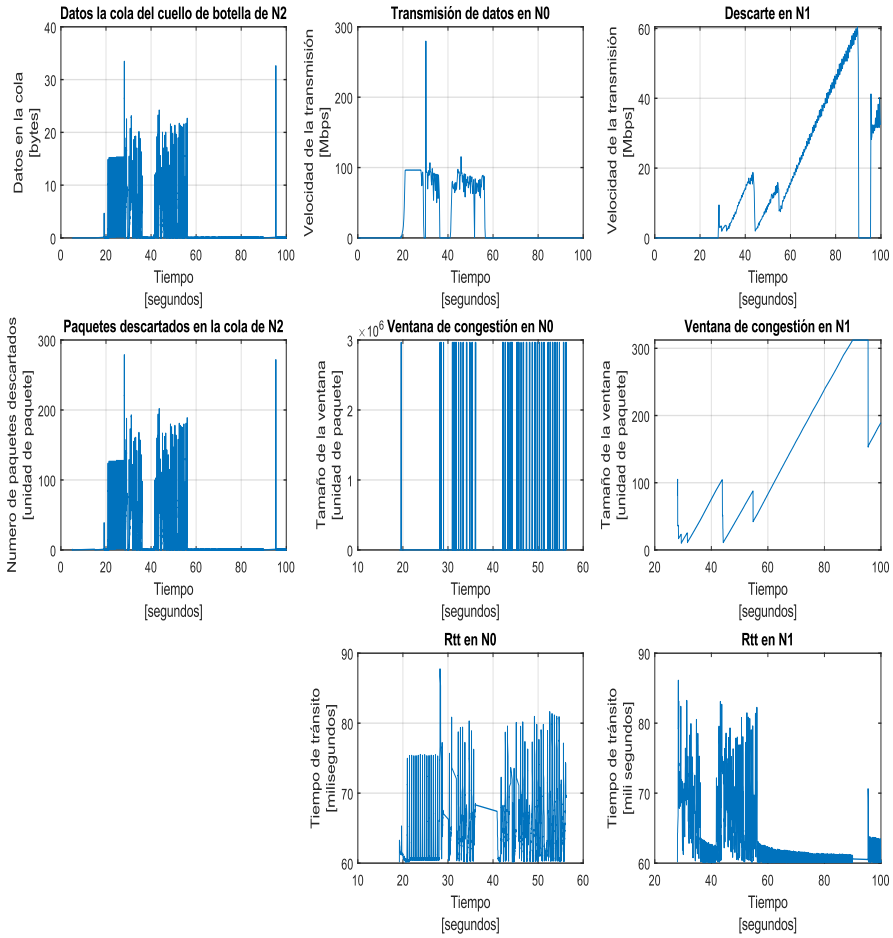

Figura 58 1672775833codigo_v150v

Simulación 49

Tabla 53 Simulación 49

ID	N0 TCP Type	N1 TCP Type	AQM	DataRate Link	DataRateBtlnck		Paq desc N0	Paquetes n1
	reno	BBR	fqcodel	100Mbps	10 Mbps	Uniform	26	419

Min_U	Max_U
3	5

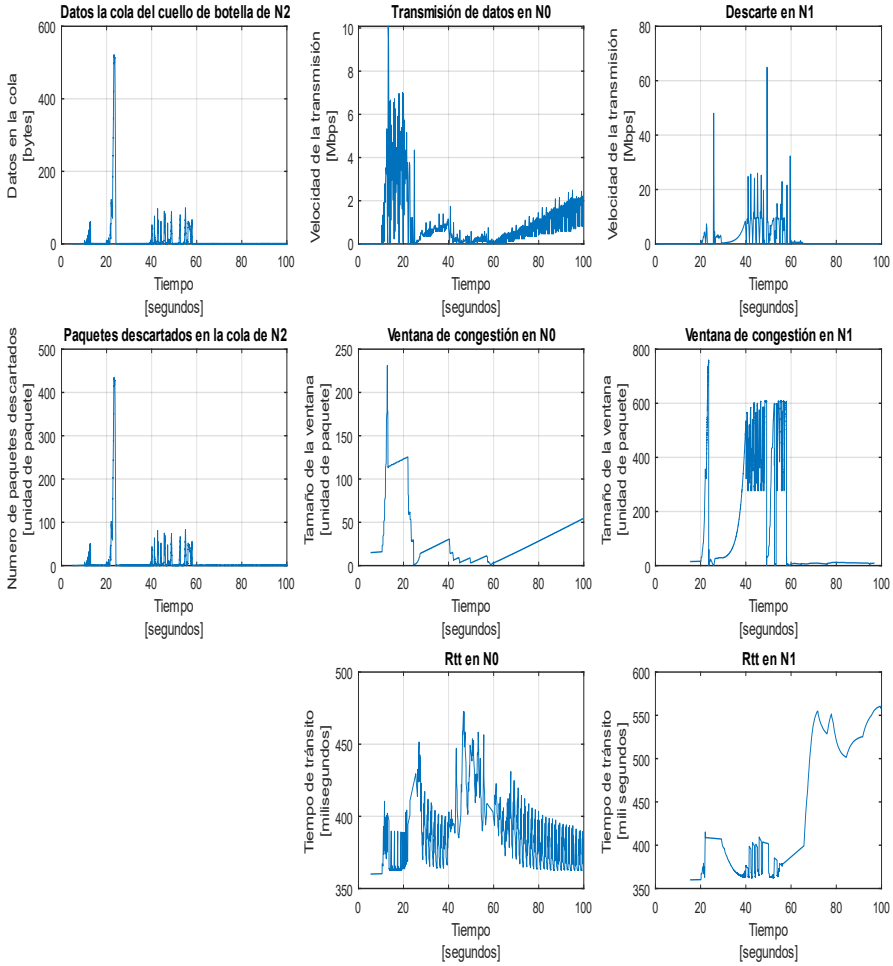

Figura 59 1672778724codigo_v150y

ANEXO 2: Código del simulador

7.1. Código de ns3

```
include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/internet-apps-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/traffic-control-module.h"
#include <iostream>
#include <ctime>
#include <string>
#include <fstream>
#include <cstdlib>
#include <cmath>
#include <vector>
#include <gnuplot-iostream.h>
#include "ns3/stats-module.h"
#include "ns3/fq-codel-queue-disc.h"



bool BBR = 0;
```

```
using namespace ns3;


NS_LOG_COMPONENT_DEFINE("Codigo_TFG");


uint32_t checkTimes;
double avgQueueDiscSize;


typedef struct Control_drop{
  uint32_t hash = 0;
  uint32_t g_dropsObserved = 0;
  uint32_t contador_linea = 0;
  double droptotal = 0;
} vcdbuf;
int drop_counter = 0;
std::vector<Control_drop> vcd;
std::vector<int> vcdl;
int contador_linea_general = 0;




void prep_inicio() {


  Control_drop c1;
  c1.hash = 5;
```

```
  c1.g_dropsObserved = 4;
  vcd.push_back(c1);
  vcdl.push_back(c1.hash);
  c1.hash = 4;
  vcd.push_back(c1);
  vcdl.push_back(c1.hash);
  int patata = vcd[0].hash;
  std::cout << "sdf" << vcd[1].hash <<  std::endl;
  std::cout << "sdf" <<  patata<< std::endl;
  std::vector<int>::iterator it = std::find(vcdl.begin(), vcdl.end(), 4);
  if (it != vcdl.end()){
    std::cout << "esta dentro " << std::distance(vcdl.begin(), it)<<std::endl;
  }
}


uint32_t g_n0BytesReceived = 0;
uint32_t g_n1BytesReceived = 0;
uint32_t g_marksObserved = 0;
uint32_t g_dropsObserved = 0;
uint32_t hash_para_representar = 0;
std::string  version = "codigo_v150v";

void
TraceN0Cwnd(std::ofstream* ofStream, uint32_t oldCwnd, uint32_t newCwnd)
{
```

```
  // TCP segment size is configured below to be 1448 bytes
  // so that we can report cwnd in units of segments
  *ofStream << Simulator::Now().GetSeconds() << " " << static_cast<double>(newCwnd) / 1448
  << std::endl;
}

void
TraceN1Cwnd(std::ofstream* ofStream, uint32_t oldCwnd, uint32_t newCwnd)
{
  // TCP segment size is configured below to be 1448 bytes
  // so that we can report cwnd in units of segments
  *ofStream << Simulator::Now().GetSeconds() << " " << static_cast<double>(newCwnd) / 1448
  << std::endl;
}

void
TraceN0Rtt(std::ofstream* ofStream, Time oldRtt, Time newRtt)
{
  *ofStream << Simulator::Now().GetSeconds() << " " << newRtt.GetSeconds() * 1000 << std::endl;
}

void
TraceN1Rtt(std::ofstream* ofStream, Time oldRtt, Time newRtt)
{
```

```
  *ofStream << Simulator::Now().GetSeconds() << " " << newRtt.GetSeconds()
* 1000 << std::endl;

}


void

TracePingRtt(std::ofstream* ofStream, Time rtt)

{

  *ofStream << Simulator::Now().GetSeconds() << " " << rtt.GetSeconds() *
1000 << std::endl;

}


void

TraceN0Rx(Ptr<const Packet> packet, const Address& address)

{

  g_n0BytesReceived += packet->GetSize();

}


void

TraceN1Rx(Ptr<const Packet> packet, const Address& address)

{

  g_n1BytesReceived += packet->GetSize();

}


void

TraceDrop(std::ofstream* ofStream, Ptr<const QueueDiscItem> item)

{

  std::stringstream ss;
```

```
  ss.precision(std::numeric_limits<long double>::digits10);
  ss << Simulator::Now().GetSeconds() << " " << item->Hash() << std::endl;
  drop_counter ++;
  // versión de serie del control de descarte
  *ofStream << ss.str();
  g_dropsObserved++;
  hash_para_representar = item -> Hash();


  //Versión Mia para control del descarte
  //std::vector<Control_drop>::iterator it = std::find((vcd.hash).begin(),
(vcd.hash).end(), 22);
  std::vector<int>::iterator it = std::find(vcdl.begin(), vcdl.end(), item->Hash());
  if (it != vcdl.end()){
    int d = std::distance(vcdl.begin(), it);
    //std::cout << "esta dentro " <<d <<std::endl;
    vcd[d].g_dropsObserved = vcd[d].g_dropsObserved + 1;
    vcd[d].droptotal = vcd[d].droptotal + 1;


  }
  else{
    Control_drop buf;
    buf.hash = item -> Hash();
    buf.g_dropsObserved = 1;
    buf.droptotal = 1;
    vcd.push_back(buf);
    vcdl.push_back(buf.hash);
```

```
  }




}

void
TraceMark(std::ofstream* ofStream, Ptr<const QueueDiscItem> item, const char* reason)
{
  *ofStream << Simulator::Now().GetSeconds() << " " << std::hex << item->Hash() << std::endl;
  g_marksObserved++;
}

void
TraceQueueLength(std::ofstream* ofStream, DataRate linkRate, uint32_t oldVal, uint32_t newVal)
{
  // output in units of ms
  *ofStream << static_cast<double>(Simulator::Now().GetSeconds()) << " " << std::fixed
  << static_cast<double>(newVal * 8) / (linkRate.GetBitRate() / 1000) <<" " <<newVal <<" "<< linkRate.GetBitRate()  <<std::endl;
  //NS_LOG_INFO ("newVal Trace QueueLength" + newVal);
}
```

```
void TraceQueuePackets(std::ofstream* ofStream, DataRate linkRate, uint32_t
oldVal, uint32_t newVal){

  *ofStream << static_cast<double>(Simulator::Now().GetSeconds()) << " " <<
std::fixed

        << static_cast<double>(newVal) << " " << std::fixed

              << static_cast<double>(oldVal) << std::endl;

        //Simulator::Schedule(throughputInterval,          &TraceQueuePackets,
ofStream, throughputInterval);

}


void

TraceDropsFrequency(std::ofstream* ofStream, Time dropsSamplingInterval)

{

  // *ofStream << Simulator::Now().GetSeconds() << " " << g_dropsObserved
<< std::endl;


  *ofStream << Simulator::Now().GetSeconds() << " ";


    for (int i=0; i< static_cast<int>(vcdl.size()); i++){

      *ofStream << vcd[i].g_dropsObserved << " " ;


      vcd[i].g_dropsObserved = 0;

      vcd[i].contador_linea ++;

    }

    *ofStream << std::endl;

  contador_linea_general ++;

  g_dropsObserved = 0;
```

```
  Simulator::Schedule(dropsSamplingInterval, &TraceDropsFrequency,
ofStream,dropsSamplingInterval);

}




void

TraceDropsFrequencyN0(std::ofstream* ofStream, Time
dropsSamplingInterval)

{

  *ofStream << Simulator::Now().GetSeconds() << " " <<
vcd[0].g_dropsObserved << std::endl;

  vcd[0].g_dropsObserved = 0;

  Simulator::Schedule(dropsSamplingInterval, &TraceDropsFrequencyN0,
ofStream,dropsSamplingInterval);

}



void

TraceDropsFrequencyN1(std::ofstream* ofStream, Time
dropsSamplingInterval)

{

  std::vector<int>::iterator it = std::find(vcdl.begin(), vcdl.end(), 4);

  *ofStream << Simulator::Now().GetSeconds() << " " <<
vcd[1].g_dropsObserved << std::endl;

  vcd[1].g_dropsObserved = 0;




  *ofStream << Simulator::Now().GetSeconds() << " ";
```

```
    for (int i=1; i< static_cast<int>(vcdl.size()); i++){
      *ofStream << vcd[i].g_dropsObserved << " " ;
      vcd[i].g_dropsObserved = 0;
    }
    *ofStream << std::endl;
  //g_dropsObserved = 0;
  Simulator::Schedule(dropsSamplingInterval, &TraceDropsFrequencyN1,
ofStream,dropsSamplingInterval);
}

void TraceMarksFrequency(std::ofstream* ofStream, Time
marksSamplingInterval) {
  *ofStream << Simulator::Now().GetSeconds() << " "<<  g_marksObserved <<
std::endl;
  g_marksObserved = 0;

Simulator::Schedule(marksSamplingInterval,&TraceMarksFrequency,ofStream
, marksSamplingInterval);
}

void TraceN0Throughput(std::ofstream* ofStream, Time throughputInterval) {
  *ofStream << Simulator::Now().GetSeconds() << " "
```

```
  << g_n0BytesReceived * 8 / throughputInterval.GetSeconds() / 1e6 <<
std::endl;

  g_n0BytesReceived = 0;

  Simulator::Schedule(throughputInterval, &TraceN0Throughput, ofStream,
throughputInterval);

}


void TraceN1Throughput(std::ofstream* ofStream, Time throughputInterval) {

  *ofStream << Simulator::Now().GetSeconds() << " "

  << g_n1BytesReceived * 8 / throughputInterval.GetSeconds() / 1e6 <<
std::endl;

  g_n1BytesReceived = 0;

  Simulator::Schedule(throughputInterval, &TraceN1Throughput, ofStream,
throughputInterval);

}


void ScheduleN0TCPCwndTraceConnection(std::ofstream* ofStream) {


Config::ConnectWithoutContext("/NodeList/1/$ns3::TCPL4Protocol/SocketList
/0/CongestionWindow",

  MakeBoundCallback(&TraceN0Cwnd, ofStream));

}


void ScheduleN0TCPRttTraceConnection(std::ofstream* ofStream) {


Config::ConnectWithoutContext("/NodeList/1/$ns3::TCPL4Protocol/SocketList
/0/RTT",

  MakeBoundCallback(&TraceN0Rtt, ofStream));

}
```

```
void ScheduleN0PacketSinkConnection() {

Config::ConnectWithoutContext("/NodeList/6/ApplicationList/*/$ns3::Packet
Sink/Rx",
  MakeCallback(&TraceN0Rx));
}

void ScheduleN1TCPCwndTraceConnection(std::ofstream* ofStream) {

Config::ConnectWithoutContext("/NodeList/2/$ns3::TCPL4Protocol/SocketList
/0/CongestionWindow",
  MakeBoundCallback(&TraceN1Cwnd, ofStream));
}

void ScheduleN1TCPRttTraceConnection(std::ofstream* ofStream) {

Config::ConnectWithoutContext("/NodeList/2/$ns3::TCPL4Protocol/SocketList
/0/RTT",
  MakeBoundCallback(&TraceN1Rtt, ofStream));
}

void
ScheduleN1PacketSinkConnection() {

Config::ConnectWithoutContext("/NodeList/7/ApplicationList/*/$ns3::Packet
Sink/Rx",
  MakeCallback(&TraceN1Rx));
}
```

```
// void ScheduleN0DropTailTraceConnection(std::ofstream* ofStream){

//
Config::ConnectWithoutContext("/Nodelist/1/$ns3::PointToPointNetDevice/Tx
Queue/$ns3:DropTailQueue", )

// }


//static void PacketEnqueue(std::ofstream* n0OfStream, std::ofstream*
n1OfStream, Ptr<const QueueDiscItem> item)




static void

PacketDequeue(std::ofstream* n0OfStream, std::ofstream* n1OfStream,
Ptr<const QueueDiscItem> item)

{

  Ptr<Packet> p = item->GetPacket();

  //uint32_t per = 0;


  uint32_t h = item -> Hash(0);


  //std::cout << "Hash descolado: " << h << std::endl;


  Ptr<const Ipv4QueueDiscItem> iqdi =

  Ptr<const                    Ipv4QueueDiscItem>(dynamic_cast<const
Ipv4QueueDiscItem*>(PeekPointer(item)));
```

```
  Ipv4Address address = iqdi->GetHeader().GetDestination();


  //std::cout << "Destino del paquete" << address << std::endl;

  Time qDelay = Simulator::Now() - item->GetTimeStamp();

  if (address == "192.168.2.2")

  {

    *n0OfStream << Simulator::Now().GetSeconds() << " " <<
qDelay.GetMicroSeconds() / 1000.0

    << std::endl;

  }

  else if (address == "192.168.3.2")

  {

    *n1OfStream << Simulator::Now().GetSeconds() << " " <<
qDelay.GetMicroSeconds() / 1000.0

    << std::endl;

  }

}

//

void Generar_Matlab(std::string directorio, std::string origen, std::string
nombre_salida){

  std::string linea;

  std::ifstream matlab_leer_or (origen);

  std::ofstream matlab_leer_fin(directorio + nombre_salida);



  if (matlab_leer_fin.is_open()) {

     if (matlab_leer_or.is_open()){
```

```
        while (std::getline (matlab_leer_or, linea)) {
            matlab_leer_fin<< linea << std::endl;
        }
        matlab_leer_or.close(); // No necesario, se cerrara al salir del bloque
    }
    else {
        std::cout << "No se ha podido abrir entrada.txt!";
    }
    matlab_leer_fin.close(); // No necesario, se cerrara al salir del bloque
  }
  else {
    std::cout << "No se ha podido crear salida.txt!";
  }

}



void gestion_frc_files(std::string file_name,  std::ofstream* N0,  std::ofstream*
N1){
  std::ifstream file(file_name);
  std::string line;
  while (std::getline(file, line)) {
    int i=0, j=0;
    std::stringstream ss(line);
    double col1, col2, col3;
```

```
    if (ss >> col1 >> col2 ){

      //*N0 << col1 << " " << col2 << std::endl;
       *N0 << col1 << " " << col2 << std::endl;
    }

    if(ss >> col3){
     *N1 << col1 << " " << col3 << std::endl;
    }
  }


  file.close();

}



  //void mover_archivo_colas (std::string directorio)



//
// }
// void plotData(std::string filename)
// {
```

```
//        vector<pair<double, double>> data; // Vector de pares (x,y) para almacenar los datos del archivo
//
//    // Abrir el archivo de datos en modo lectura
//    ifstream file(filename);
//    if (!file.is_open())
//    {
//        cerr << "Error: no se pudo abrir el archivo " << filename << endl;
//        return;
//    }
//
//    // Leer los datos del archivo y almacenarlos en el vector
//    double x, y;
//    while (file >> x >> y)
//    {
//        data.push_back(make_pair(x, y));
//    }
//
//    // Crear una instancia de Gnuplot
//    Gnuplot gp;
//
//    // Configurar el tamaño de la imagen jpg que se creará
//    gp << "set terminal jpeg size 768,768" << endl;
//
//    // Establecer los ejes x e y
//    gp << "set xlabel 'Tiempo en segundos'" << endl;
```

```
//     gp << "set ylabel 'Paquetes enviados'" << endl;
//
//     // Generar la imagen jpg y guardarla en el mismo directorio donde se está ejecutando el programa principal
//     gp << "set output 'plot.jpg'" << endl;
//
//     // Dibujar una línea usando los datos del archivo
//     gp << "plot '-' with lines" << endl;
//     gp.send(data);
//
//     std::cout << "Imagen jpg creada con éxito en el directorio actual" << endl;
// }


int main(int argc, char* argv[]) {

  NS_LOG_INFO("Inicio del programa");
  //prep_inicio();



  ////////////////////////////////////////////////////////
  // variables not configured at command line          //
  ////////////////////////////////////////////////////////
  Time stopTime = Seconds(70);
  Time baseRtt = MilliSeconds(80);
```

```
uint32_t pingSize = 100; // bytes
Time pingInterval = MilliSeconds(100000);
Time marksSamplingInterval = MilliSeconds(100);
Time throughputSamplingInterval = MilliSeconds(200);
DataRate bottleneckRate("100Mbps");




// Se añade std::string(+n1TCPType)
std::string dir = std::string("results/Final/");
time_t t=time(NULL);

std::string ver = std::to_string(static_cast<int>(t));
dir = dir + ver + version + "/";
std::string dirToSave = "mkdir -p " + dir;
if (system(dirToSave.c_str()) == -1)
{
  exit(1);
}

std::string MaxSizePacketinQueue = "3p";
std::string initCwnd = "500";
std::string pingTraceFile = dir + "ping.dat";
std::string n0TCPRttTraceFile = dir + "n0_TCP_rtt.dat";
std::string n0TCPCwndTraceFile = dir + "n0_TCP_cwnd.dat";
```

```
std::string n0TCPThroughputTraceFile = dir + "n0_TCP_throughput.dat";
std::string n1TCPRttTraceFile = dir + "n1_TCP_rtt.dat";
std::string n1TCPCwndTraceFile = dir + "n1_TCP_cwnd.dat";
std::string n1TCPThroughputTraceFile = dir + "n1_TCP_throughput.dat";
std::string dropTraceFile = dir + "drops.dat";
std::string dropsFrequencyTraceFile = dir + "zdrops_frequency.dat";
std::string dropsFrequencyTraceN0File = dir + "drops_frequency_N0.dat";
std::string dropsFrequencyTraceN1File = dir + "drops_frequency_N1.dat";
std::string lengthTraceFile = dir + "length.dat";
std::string packetsQueueFile = dir + "paquetes_cola.dat";
std::string markTraceFile = dir + "zmark.dat";
std::string marksFrequencyTraceFile = dir + "marks_frequency.dat";
std::string queueDelayN0TraceFile = dir + "queue_delay_n0.dat";
std::string queueDelayN1TraceFile = dir + "queue_delay_n1.dat";
//std::string colasn2file = dir +"colasn2.dat";

//////////////////////////////////////////////////////////
// variables configured at command line                 //
//////////////////////////////////////////////////////////
bool enablePcap = false;
bool useCeThreshold = false;
Time ceThreshold = MilliSeconds(1);
std::string n0TCPType = "BIC";
std::string n1TCPType = "";
bool enableN1TCP = false;
bool useEcn = false;
```

```
  std::string queueType = "fq";
  std::string linkDataRate = "1Gbps";
  uint32_t scenarioNum = 0;




  CommandLine cmd;

  cmd.AddValue("n0TCPType", "n0 TCP Type (cuBIC, BIC, DCTCP, reno or BBR",
n0TCPType);
  cmd.AddValue("n1TCPType", "n1 TCP Type (cuBIC, BIC, DCTCP, reno or BBR",
n1TCPType);
  cmd.AddValue("scenarioNum","Selection of the scenario from disgned to exp",
scenarioNum);
  cmd.AddValue("bottleneckQueueType", "n2 queue type fq or codel",
queueType);
  cmd.AddValue("baseRtt", "base RTT", baseRtt);
  cmd.AddValue("useCeThreshold", "use CE Threshold", useCeThreshold);
  cmd.AddValue("useEcn", "use ECN", useEcn);
  cmd.AddValue("ceThreshold", "CoDel CE threshold", ceThreshold);
  cmd.AddValue("bottleneckRate", "data rate of bottleneck", bottleneckRate);
  cmd.AddValue("linkRate", "data rate of edge link", linkDataRate);
  cmd.AddValue("stopTime", "simulation stop time", stopTime);
  cmd.AddValue("enablePcap", "enable Pcap", enablePcap);
  cmd.AddValue("pingTraceFile", "filename for ping tracing", pingTraceFile);
```

```
cmd.AddValue("n0TCPRttTraceFile", "filename for n0 rtt tracing", n0TCPRttTraceFile);

cmd.AddValue("n0TCPCwndTraceFile", "filename for n0 cwnd tracing", n0TCPCwndTraceFile);

cmd.AddValue("n0TCPThroughputTraceFile","filename for n0 throughput tracing",

        n0TCPThroughputTraceFile);

cmd.AddValue("n1TCPRttTraceFile", "filename for n1 rtt tracing", n1TCPRttTraceFile);

cmd.AddValue("n1TCPCwndTraceFile", "filename for n1 cwnd tracing", n1TCPCwndTraceFile);

cmd.AddValue("n1TCPThroughputTraceFile", "filename for n1 throughput tracing",

        n1TCPThroughputTraceFile);

cmd.AddValue("dropTraceFile", "filename for n2 drops tracing", dropTraceFile);

cmd.AddValue("dropsFrequencyTraceFile", "filename for n2 drop frequency tracing",

        dropsFrequencyTraceFile);

cmd.AddValue("lengthTraceFile", "filename for n2 queue length tracing", lengthTraceFile);

cmd.AddValue("packetsQueueFile", "filename for n2 queue packets tracing", packetsQueueFile);

cmd.AddValue("markTraceFile", "filename for n2 mark tracing", markTraceFile);

cmd.AddValue("marksFrequencyTraceFile", "filename for n2 mark frequency tracing", marksFrequencyTraceFile);

cmd.AddValue("queueDelayN0TraceFile", "filename for n0 queue delay tracing",

        queueDelayN0TraceFile);

cmd.AddValue("queueDelayN1TraceFile", "filename for n1 queue delay tracing",
```

```
            queueDelayN1TraceFile);
  cmd.AddValue("initialCwnd", "Initial congestion window for all TCP", initCwnd);
  cmd.AddValue("MaxSizePacketinQueue", "Max Packets in DroptailQueue
[numberp]",MaxSizePacketinQueue);
  cmd.Parse(argc, argv);
  Time oneWayDelay = baseRtt / 2;
  TypeId n0TCPTypeId;
  TypeId n1TCPTypeId;
  TypeId queueTypeId;

  //int envio =8192000,recepcion = 8192000, valinitCwnd = stoi (initCwnd);
  //int envio =41943040,recepcion = 62914565, valinitCwnd = stoi (initCwnd);
  int envio =41943040,recepcion = 41943040, valinitCwnd = stoi (initCwnd);
  //int envio =289600,recepcion = 289600, valinitCwnd = stoi (initCwnd);
  Config::SetDefault("ns3::TCPSocket::SegmentSize",UintegerValue(1448));
  Config::SetDefault("ns3::TCPSocket::SndBufSize", UintegerValue(envio));
  Config::SetDefault("ns3::TCPSocket::RcvBufSize", UintegerValue(recepcion));
  Config::SetDefault("ns3::TCPSocket::InitialCwnd",
UintegerValue(valinitCwnd));
  Config::SetDefault("ns3::TCPL4Protocol::RecoveryType",
            TypeIdValue(TCPPrrRecovery::GetTypeId()));
  //Config::SetDefault("ns3::FifoQueueDisc::MaxSize",
QueueSizeValue(QueueSize("3p")));
```

```
std::cout << "Cwnd Inicial" << initCwnd << std::endl;
std::cout << "\n N0 tipo: " << n0TCPType;

if (!scenarioNum){
 if (useEcn){
  Config::SetDefault("ns3::TCPSockeBase::UseEcn", StringValue("0ff"));
 }
 else if (n0TCPType == "reno"){
  n0TCPTypeId = TCPNewReno::GetTypeId();
 }
 else if (n0TCPType == "BIC"){
  n0TCPTypeId = TCPBIC::GetTypeId();
 }
 else if (n0TCPType == "DCTCP"){
  n0TCPTypeId = TCPDCTCP::GetTypeId();
 }
 else if (n0TCPType == "BBR"){
  n0TCPTypeId = TCPBBR::GetTypeId();
  BBR = 1;
 }

 else if (n0TCPType == "cuBIC"){
  n0TCPTypeId == TCPCuBIC::GetTypeId();
 }
```

```
else if (n1TCPType == "yeah"){
  n0TCPTypeId == TypeId::LookupByName("ns3::TCPYeah");
}
else if (n1TCPType == "lp"){
  n0TCPTypeId == TypeId::LookupByName("ns3::TCPLp");
}
else if (n1TCPType == "scalable"){
  n0TCPTypeId == TypeId::LookupByName("ns3::TCPScalable");
}

else if (n1TCPType == "HighSpeed"){
  n0TCPTypeId == TypeId::LookupByName("ns3::TCPHighSpeed");
}
else
{
  NS_FATAL_ERROR("Fatal error:  TCP unsupported");
}

if (n1TCPType == "reno"){
  enableN1TCP = true;
  n1TCPTypeId = TCPNewReno::GetTypeId();
}
else if (n1TCPType == "BIC"){
  enableN1TCP = true;
  n1TCPTypeId = TCPBIC::GetTypeId();
}
```

```
else if (n1TCPType == "DCTCP"){
  enableN1TCP = true;
  n1TCPTypeId = TypeId::LookupByName("ns3::TCPDCTCP");
}

else if (n1TCPType == "BBR"){
  enableN1TCP = true;
  n1TCPTypeId = TypeId::LookupByName("ns3::TCPBBR");
  BBR = 1;
}


else if (n1TCPType == "yeah"){
  n1TCPTypeId == TypeId::LookupByName("ns3::TCPYeah");
}
else if (n1TCPType == "lp"){
  n1TCPTypeId == TypeId::LookupByName("ns3::TCPLp");
}
else if (n1TCPType == "scalable"){
  n1TCPTypeId == TypeId::LookupByName("ns3::TCPScalable");
}

else if (n1TCPType == "HighSpeed"){
  n1TCPTypeId == TypeId::LookupByName("ns3::TCPHighSpeed");
}
else if (n1TCPType == ""){
```

```
    NS_LOG_DEBUG("No N1 TCP selected");
  }
  else{
    NS_FATAL_ERROR("Fatal error:  TCP unsupported");
  }
  if (queueType == "fq"){
    queueTypeId = FqCoDelQueueDisc::GetTypeId();
  }
  else if (queueType == "codel"){
    queueTypeId = CoDelQueueDisc::GetTypeId();
  }
  else if (queueType =="red"){
    queueTypeId = RedQueueDisc::GetTypeId();
    NS_LOG_INFO("Set RED params");
    uint32_t meanPktSize = 500;
    Config::SetDefault("ns3::RedQueueDisc::MaxSize",
StringValue(MaxSizePacketinQueue));
    Config::SetDefault("ns3::RedQueueDisc::MeanPktSize",
UintegerValue(meanPktSize));
    Config::SetDefault("ns3::RedQueueDisc::Wait", BooleanValue(true));
    Config::SetDefault("ns3::RedQueueDisc::Gentle", BooleanValue(true));
    Config::SetDefault("ns3::RedQueueDisc::QW", DoubleValue(0.002));
    Config::SetDefault("ns3::RedQueueDisc::MinTh", DoubleValue(5));
    Config::SetDefault("ns3::RedQueueDisc::MaxTh", DoubleValue(15));

  }
```

```
    else if(queueType == "fifo"){
      queueTypeId = FifoQueueDisc::GetTypeId();
    }
    else{
      NS_FATAL_ERROR("Fatal error:  queueType unsupported");
    }
    if (useCeThreshold){
      Config::SetDefault("ns3::FqCoDelQueueDisc::CeThreshold",
TimeValue(ceThreshold));
    }
  }


  std::ofstream pingOfStream;
  pingOfStream.open(pingTraceFile, std::ofstream::out);
  std::ofstream n0TCPRttOfStream;
  n0TCPRttOfStream.open(n0TCPRttTraceFile, std::ofstream::out);
  std::ofstream n0TCPCwndOfStream;
  n0TCPCwndOfStream.open(n0TCPCwndTraceFile, std::ofstream::out);
  std::ofstream n0TCPThroughputOfStream;
  n0TCPThroughputOfStream.open(n0TCPThroughputTraceFile,
std::ofstream::out);
  std::ofstream n1TCPRttOfStream;
  n1TCPRttOfStream.open(n1TCPRttTraceFile, std::ofstream::out);
  std::ofstream n1TCPCwndOfStream;
  n1TCPCwndOfStream.open(n1TCPCwndTraceFile, std::ofstream::out);
  std::ofstream n1TCPThroughputOfStream;
```

```
  n1TCPThroughputOfStream.open(n1TCPThroughputTraceFile,
std::ofstream::out);


  // Queue disc files
  std::ofstream dropOfStream;
  dropOfStream.open(dropTraceFile, std::ofstream::out);
  std::ofstream markOfStream;
  markOfStream.open(markTraceFile, std::ofstream::out);
  std::ofstream dropsFrequencyOfStream;
  dropsFrequencyOfStream.open(dropsFrequencyTraceFile, std::ofstream::out);
  std::ofstream dropsFrequencyOfStreamN0;
  dropsFrequencyOfStreamN0.open(dropsFrequencyTraceN0File,
std::ofstream::out);
  std::ofstream dropsFrequencyOfStreamN1;
  dropsFrequencyOfStreamN1.open(dropsFrequencyTraceN1File,
std::ofstream::out);


  std::ofstream marksFrequencyOfStream;
  marksFrequencyOfStream.open(marksFrequencyTraceFile,
std::ofstream::out);
  std::ofstream lengthOfStream;
  lengthOfStream.open(lengthTraceFile, std::ofstream::out);
  std::ofstream packetsOfStream;
  packetsOfStream.open(packetsQueueFile, std::ofstream::out);
  std::ofstream queueDelayN0OfStream;
  queueDelayN0OfStream.open(queueDelayN0TraceFile, std::ofstream::out);
  std::ofstream queueDelayN1OfStream;
```

```
queueDelayN1OfStream.open(queueDelayN1TraceFile, std::ofstream::out);
//std::ofstream colasn2;
//colasn2.open(colasn2file, std::ofstream::out);



////////////////////////////////////////////////////////
// scenario setup                                     //
////////////////////////////////////////////////////////

NS_LOG_INFO("Configuracion del escenario");
Ptr<Node> pingServer = CreateObject<Node>();
Ptr<Node> n0Server = CreateObject<Node>();
Ptr<Node> n1Server = CreateObject<Node>();
Ptr<Node> n2 = CreateObject<Node>();
Ptr<Node> n3 = CreateObject<Node>();
Ptr<Node> pingClient = CreateObject<Node>();
Ptr<Node> n4Client = CreateObject<Node>();
Ptr<Node> n5Client = CreateObject<Node>();

// Device containers
NS_LOG_INFO("Device Containers");
NetDeviceContainer pingServerDevices;
NetDeviceContainer n0ServerDevices;
NetDeviceContainer n1ServerDevices;
NetDeviceContainer n2n3Devices;
NetDeviceContainer pingClientDevices;
```

```
  NetDeviceContainer n4ClientDevices;
  NetDeviceContainer n5ClientDevices;

  PointToPointHelper p2p;
  NS_LOG_INFO("Configure queue");
  p2p.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("1p"));
  p2p.SetDeviceAttribute("DataRate",
DataRateValue(DataRate(linkDataRate)));
  // Add delay only on the server links
  p2p.SetChannelAttribute("Delay", TimeValue(oneWayDelay/3));
  pingServerDevices = p2p.Install(n2, pingServer);
  n0ServerDevices = p2p.Install(n2, n0Server);
  n1ServerDevices = p2p.Install(n2, n1Server);
  //p2p.SetChannelAttribute("DataRate",
StringValue(std::string("1.5Mbps")));//Lo que acabo de añadir €
  p2p.SetChannelAttribute("Delay", TimeValue(oneWayDelay/3));
  //p2p.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("500p"));
  n2n3Devices = p2p.Install(n2, n3);
  pingClientDevices = p2p.Install(n3, pingClient);
  n4ClientDevices = p2p.Install(n3, n4Client);
  n5ClientDevices = p2p.Install(n3, n5Client);

  Ptr<PointToPointNetDevice>          p          =          n2n3Devices.Get(0)-
>GetObject<PointToPointNetDevice>();
  p->SetAttribute("DataRate", DataRateValue(bottleneckRate));

  InternetStackHelper stackHelper;
```

```
stackHelper.InstallAll();

//Set the per-node TCP type here
Ptr<TCPL4Protocol> proto;
proto = n4Client->GetObject<TCPL4Protocol>();
proto->SetAttribute("SocketType", TypeIdValue(n0TCPTypeId));
proto = n0Server->GetObject<TCPL4Protocol>();
proto->SetAttribute("SocketType", TypeIdValue(n0TCPTypeId));
if (enableN1TCP){
  proto = n5Client->GetObject<TCPL4Protocol>();
  proto->SetAttribute("SocketType", TypeIdValue(n1TCPTypeId));
  proto = n1Server->GetObject<TCPL4Protocol>();
  proto->SetAttribute("SocketType", TypeIdValue(n1TCPTypeId));
}

// InternetStackHelper will install a base TrafficControLayer on the node,
// but the Ipv4AddressHelper below will install the default FqCoDelQueueDisc
// on all single device nodes.  The below code overrides the configuration
// that is normally done by the Ipv4AddressHelper::Install() method by
// instead explicitly configuring the queue discs we want on each device.



TrafficControlHelper tchFq;
```

```
TrafficControlHelper tch;

tch.SetRootQueueDisc ("ns3::FifoQueueDisc", "MaxSize", StringValue("3p"));

//tch.SetRootQueueDisc ("ns3::FifoQueueDisc");

//tch.SetAttribute("ns3::DropTailQueue", "MaxSize", StringValue("3p"));

//tch.SetQueueLimits("ns3::DynamicQueueLimits", "HoldTime",
StringValue("1ms"));

//tch.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("500p"));

//tch.AddInternalQueues (handle,, "ns3::DropTailQueue", "MaxSize",
StringValue ("500p"));


//tch.SetRootQueueDisc("ns3::FqCoDelQueueDisc", "MaxSize",
StringValue("2p"));


//tch.SetRootQueueDisc ("ns3::CoDelQueueDisc", "MaxSize", StringValue
("500p"));

//tch.SetQueueLimits("ns3::DynamicQueueLimits", "HoldTime",
StringValue("2ms"));

tch.Install(pingServerDevices);

tch.Install(n0ServerDevices);

tch.Install(n1ServerDevices);

tch.Install(n2n3Devices.Get(1)); // n2 queue for bottleneck link

tch.Install(pingClientDevices);

tch.Install(n4ClientDevices);

tch.Install(n5ClientDevices);

TrafficControlHelper tchN2;

tchN2.SetRootQueueDisc(queueTypeId.GetName(), "MaxSize",
StringValue(MaxSizePacketinQueue));

//tchN2.SetQueueLimits("ns3::DynamicQueueLimits", "HoldTime",
StringValue("200ms"));
```

```
  tchN2.Install(n2n3Devices.Get(0));









  //

  //

  // tchFq.SetRootQueueDisc("ns3::FifoQueueDisc");

  // //tchFq.SetRootQueueDisc("ns3::FqCoDelQueueDisc");

  //        tchFq.SetQueueLimits("ns3::DynamicQueueLimits",        "HoldTime",
StringValue("1ms"));

  // tchFq.Install(pingServerDevices);

  // tchFq.Install(n0ServerDevices);

  // tchFq.Install(n1ServerDevices);

  // tchFq.Install(n2n3Devices.Get(1)); // n2 queue for bottleneck link

  // tchFq.Install(pingClientDevices);

  // tchFq.Install(n4ClientDevices);

  // tchFq.Install(n5ClientDevices);

  // TrafficControlHelper tchN2;

  // //tchN2.SetRootQueueDisc(queueTypeId.GetName());

  // //tchN2.SetRootQueueDisc("ns3::FifoQueueDisc");

  //        tchN2.SetQueueLimits("ns3::DynamicQueueLimits",        "HoldTime",
StringValue("10ms"));

  // tchN2.Install(n2n3Devices.Get(0));

  //
```

```
NS_LOG_INFO("IP nodes");
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer pingServerIfaces = ipv4.Assign(pingServerDevices);
ipv4.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer n0ServerIfaces = ipv4.Assign(n0ServerDevices);
ipv4.SetBase("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer secondServerIfaces = ipv4.Assign(n1ServerDevices);
ipv4.SetBase("172.16.1.0", "255.255.255.0");
Ipv4InterfaceContainer n2n3Ifaces = ipv4.Assign(n2n3Devices);
ipv4.SetBase("192.168.1.0", "255.255.255.0");
Ipv4InterfaceContainer pingClientIfaces = ipv4.Assign(pingClientDevices);
ipv4.SetBase("192.168.2.0", "255.255.255.0");
Ipv4InterfaceContainer n4ClientIfaces = ipv4.Assign(n4ClientDevices);
ipv4.SetBase("192.168.3.0", "255.255.255.0");
Ipv4InterfaceContainer n5ClientIfaces = ipv4.Assign(n5ClientDevices);

Ipv4GlobalRoutingHelper::PopulateRoutingTables();


////////////////////////////////////////////////////////
// application setup                                  //
////////////////////////////////////////////////////////
```

```
// NS_LOG_INFO("Definición de aplicaciones");

// V4PingHelper pingHelper("192.168.1.2");

// pingHelper.SetAttribute("Interval", TimeValue(pingInterval));

// pingHelper.SetAttribute("Size", UintegerValue(pingSize));

// ApplicationContainer pingContainer = pingHelper.Install(pingServer);

// Ptr<V4Ping> v4Ping = pingContainer.Get(0)->GetObject<V4Ping>();

//                                   v4Ping->TraceConnectWithoutContext("Rtt",
MakeBoundCallback(&TracePingRtt, &pingOfStream));

// pingContainer.Start(Seconds(1));

// pingContainer.Stop(stopTime - Seconds(1));



// BulkSendHelper TCP("ns3::TCPSocketFactory", Address());

// // set to large value:  e.g. 1000 Mb/s for 60 seconds = 7500000000 bytes

// TCP.SetAttribute("MaxBytes", UintegerValue(7500000000));

// // Configure n4/n0 TCP client/server pair

uint16_t n4Port = 6000;



OnOffHelper TCP ("ns3::TCPSocketFactory", Address ());

//TCP.SetAttribute ("OnTime");

//TCP.SetAttribute ("OffTime");
```

```
  //std::string constOn = "200"
  //TypeId valor

  double rndmin = 0.01, max_On = 0.6, max_Off = 0.3;
  Ptr<UniformRandomVariable> Encendido = CreateObject<UniformRandomVariable>();
  Ptr<UniformRandomVariable> Apagado = CreateObject<UniformRandomVariable>();
  Encendido -> SetAttribute ("Min", DoubleValue(rndmin));
  Encendido -> SetAttribute ("Max", DoubleValue(max_On));
  Apagado -> SetAttribute ("Min", DoubleValue(rndmin));
  Apagado -> SetAttribute ("Max", DoubleValue(max_Off));

  double valor = Encendido->GetValue();



  TCP.SetAttribute("PacketSize", UintegerValue(1448));
  TCP.SetAttribute("DataRate", DataRateValue(DataRate(linkDataRate)));
  TCP.SetAttribute("MaxBytes",  UintegerValue(524277360));   //7500000000 524277360

  //TCP.SetAttribute ("OnTime", StringValue ("ns3::RandomVariableStream"));
  TCP.SetAttribute ("OnTime", StringValue("ns3::UniformRandomVariable"));
  TCP.SetAttribute ("OffTime", StringValue("ns3::UniformRandomVariable"));
```

```
  //TCP.SetAttribute ("ns3::UniformRandomVariable::Min",DoubleValue(0.1));

  //TCP.SetAttribute ("ns3::UniformRandomVariable::Max",DoubleValue(0.6));


  //std::cout << TCP.SetAttribute ->



  //TCP.SetAttribute
("ns3::UniformRandomVariable::m_min",DoubleValue(0.1));

  //TCP.SetAttribute
("ns3::UniformRandomVariable::m_max",DoubleValue(0.4));

  //TCP.SetAttribute ("OffTime", StringValue ("ns3::RandomVariableStream"));



  //TCP.SetAttribute                ("OffTime",                StringValue
("ns3::ConstantRandomVariable[Constant=1]"));

  ApplicationContainer n0App;

  InetSocketAddress n0DestAddress(n4ClientIfaces.GetAddress(1), n4Port);

    NS_LOG_INFO("Destino n0");

  NS_LOG_INFO(n4ClientIfaces.GetAddress(1));

  TCP.SetAttribute("Remote", AddressValue(n0DestAddress));
```

```
n0App = TCP.Install(n0Server);
n0App.Start(Seconds(5));
n0App.Stop(stopTime - Seconds(1));

Address n4SinkAddress(InetSocketAddress(Ipv4Address::GetAny(), n4Port));
PacketSinkHelper n4SinkHelper("ns3::TCPSocketFactory", n4SinkAddress);
ApplicationContainer n4SinkApp;
n4SinkApp = n4SinkHelper.Install(n4Client);
n4SinkApp.Start(Seconds(5));
n4SinkApp.Stop(stopTime - MilliSeconds(500));




// Configure second TCP client/server pair
if (enableN1TCP)
{

  uint16_t n5Port = 5000;
  ApplicationContainer secondApp;
  InetSocketAddress n1DestAddress(n5ClientIfaces.GetAddress(1), n5Port);
  NS_LOG_INFO("dESTINO N1");
  NS_LOG_INFO(n5ClientIfaces.GetAddress(1));
    OnOffHelper TCP2("ns3::TCPSocketFactory", n1DestAddress);
    TCP2.SetAttribute("PacketSize", UintegerValue(1448));
    TCP2.SetAttribute("DataRate", DataRateValue(DataRate(linkDataRate)));
```

```
    TCP2.SetAttribute("MaxBytes", UintegerValue(524277360)); //
50Megabytes // 36207 paquetes 524277360 //262138680

                                    //10 485 760

    //TCP.SetAttribute ("OnTime", StringValue ("ns3::RandomVariableStream"));

    TCP2.SetAttribute ("OnTime", StringValue("ns3::UniformRandomVariable"));

    TCP2.SetAttribute ("OffTime", StringValue("ns3::UniformRandomVariable"));



  TCP2.SetAttribute("Remote", AddressValue(n1DestAddress));

  secondApp = TCP2.Install(n1Server);

  secondApp.Start(Seconds(15));

  secondApp.Stop(stopTime - Seconds(1));


  Address n5SinkAddress(InetSocketAddress(Ipv4Address::GetAny(), n5Port));

  PacketSinkHelper n5SinkHelper("ns3::TCPSocketFactory", n5SinkAddress);

  ApplicationContainer n5SinkApp;

  n5SinkApp = n5SinkHelper.Install(n5Client);

  n5SinkApp.Start(Seconds(15));

  n5SinkApp.Stop(stopTime - MilliSeconds(500));

}



// Setup traces that can be hooked now

Ptr<TrafficControlLayer> tc;

Ptr<QueueDisc> qd;

tc = n2n3Devices.Get(0)->GetNode()->GetObject<TrafficControlLayer>();
```

```
  qd = tc->GetRootQueueDiscOnDevice(n2n3Devices.Get(0));

  qd->TraceConnectWithoutContext("Drop",    MakeBoundCallback(&TraceDrop,
&dropOfStream));

  qd->TraceConnectWithoutContext("Mark",   MakeBoundCallback(&TraceMark,
&markOfStream));

  qd->TraceConnectWithoutContext(

    "BytesInQueue",

    MakeBoundCallback(&TraceQueueLength,                &lengthOfStream,
bottleneckRate));

  qd ->TraceConnectWithoutContext("PacketsInQueue",

        MakeBoundCallback(&TraceQueuePackets,          &packetsOfStream,
bottleneckRate));


  qd->TraceConnectWithoutContext("Dequeue",

        MakeBoundCallback(&PacketDequeue,          &queueDelayN0OfStream,
&queueDelayN1OfStream));




  // Setup scheduled traces; TCP traces must be hooked after socket creation

  Simulator::Schedule(Seconds(5) + MilliSeconds(100),

                &ScheduleN0TCPRttTraceConnection,

                &n0TCPRttOfStream);

  Simulator::Schedule(Seconds(5) + MilliSeconds(100),

                &ScheduleN0TCPCwndTraceConnection,

                &n0TCPCwndOfStream);
```

```
  Simulator::Schedule(Seconds(5) + MilliSeconds(100),
&ScheduleN0PacketSinkConnection);
  Simulator::Schedule(throughputSamplingInterval,
                      &TraceN0Throughput,
                      &n0TCPThroughputOfStream,
                      throughputSamplingInterval);
  //
  // Simulator::Schedule(Seconds(5) + MilliSeconds(100),
  //                     &ScheduleN0DropTailTraceConnection,
  //                     &)

  // Setup scheduled traces; TCP traces must be hooked after socket creation
  if (enableN1TCP)
  {
    Simulator::Schedule(Seconds(15) + MilliSeconds(100),
    &ScheduleN1TCPRttTraceConnection,
    &n1TCPRttOfStream);
    Simulator::Schedule(Seconds(15) + MilliSeconds(100),
    &ScheduleN1TCPCwndTraceConnection,
    &n1TCPCwndOfStream);
    Simulator::Schedule(Seconds(15) + MilliSeconds(100),
&ScheduleN1PacketSinkConnection);
  }
  Simulator::Schedule(throughputSamplingInterval,
                      &TraceN1Throughput,
                      &n1TCPThroughputOfStream,
                      throughputSamplingInterval);
```

```
  Simulator::Schedule(marksSamplingInterval,
                      &TraceMarksFrequency,
                      &marksFrequencyOfStream,
                      marksSamplingInterval);
  Simulator::Schedule(Seconds(2),
                      &TraceDropsFrequency,
                      &dropsFrequencyOfStream,
                      marksSamplingInterval);

  // Simulator::Schedule(Seconds(5) + MilliSeconds(150),
  //                     &TraceDropsFrequencyN0,
  // //                      &dropsFrequencyOfStreamN0,
  // //                      marksSamplingInterval);
  // Simulator::Schedule(Seconds(15) + MilliSeconds(10),
  //                     &TraceDropsFrequencyN1,
  //                     &dropsFrequencyOfStreamN1,
  //                     marksSamplingInterval);

 if (enablePcap)
  {
    p2p.EnablePcapAll("FqCoDel-L4S-example", false);
  }


  NS_LOG_INFO(std::string("Algoritmo congestion n0:" + n0TCPType));
  NS_LOG_INFO(std::string("Algoritmo congestion n1:" + n1TCPType));
```

```
  //NS_LOG_INFO(std::string("Algoritmo congestion n2 _el cuello de botella_:" +
queueTypeId));

  NS_LOG_INFO(std::string("Algoritmo de la cola n2:" +queueType));

  //NS_LOG_INFO(std::string("Escenario que tiene:")+scenarioNum);



  NS_LOG_INFO("Stop Simulator time");

  Simulator::Stop(stopTime);

  time_t t_init, t_end;

  int min = 0, sec = 0;



  NS_LOG_INFO("Run Simulator");

  t_init = time(NULL);

  Simulator::Run();

  t_end = time(NULL);


  sec = t_end - t_init;


  if (sec >= 60 ){

    min = (static_cast<int>(sec));

    min = min / 60;

    sec = sec - min*60;


  }
```

```
  std::ofstream sacar(dir + "variables_simulacion.dat");


  sacar << version << std::endl;

  sacar <<"\n-----Variables de ejecución de la simulación-----" << std::endl;

  sacar <<"Tiempo de Simulación; " << stopTime << std::endl;

  sacar <<"base Rtt; "<< baseRtt << std::endl;

  sacar << "Tamaño del Ping ; " << pingSize << std::endl;

  sacar << "Muestro de los paquetes marcados; " << marksSamplingInterval <<
std::endl;

  sacar << "Muestreo de los paquetes enviados; " <<
throughputSamplingInterval << std::endl;

  sacar << "Ancho de banda del cuello de botella; " << bottleneckRate <<
std::endl;

  sacar << "cwndinicial: "<< initCwnd;



  std::cout << "\nDuración de la simulación: " << min<<" minutos y " <<sec <<
"segundos." << std::endl;

  sacar << "\nDuración de la simulación: " << min<<" minutos y " <<sec <<
"segundos." << std::endl;

  sacar << "N0 tipo; "<< n0TCPType << std::endl;

  sacar << "N1 tipo; "<< n1TCPType << std::endl;

  sacar << "Tipo de cola; " << queueType << std::endl;

  sacar << "Ancho de banda; " << linkDataRate << std::endl;

  sacar << "ECN; " << useEcn << std::endl;
```

```
  sacar << "PointToPoint Max Queue size:"<< MaxSizePacketinQueue << std::endl;

  sacar << "Tamaño cola salida, " << envio << std::endl;

  sacar << "Tamaño cola entrada, " << recepcion << std::endl;

  sacar << "\n\n_____"<< std::endl;

  sacar << std::endl << std::endl << std::endl << std::endl;

  sacar << "------Comparativa de paquetes descartados-----" << std::endl;

  sacar << "\n\nHash \t Paquetes descartados \t Paquetes totales \tPorcentaje de paquetes descartados" << std::endl;

  for (int i=0; i< static_cast<int>(vcdl.size()); i++){

    double porcentaje = vcd[i].droptotal/52427736;

    sacar << i<<" " << vcd[i].hash << "\t" << vcd[i].droptotal << " \t 52427736 \t" <<  porcentaje << std::endl;

  }

  sacar << "Descartes totales: " << drop_counter << std::endl;




  // Guarda los argumentos en el archivo

  for (int i = 1; i < argc; i++) {

    sacar << argv[i] << std::endl;

  }




  NS_LOG_INFO("Close created files");

  pingOfStream.close();
```

```
  n0TCPCwndOfStream.close();
  n0TCPRttOfStream.close();
  n0TCPThroughputOfStream.close();
  n1TCPCwndOfStream.close();
  n1TCPRttOfStream.close();
  n1TCPThroughputOfStream.close();
  dropOfStream.close();
  markOfStream.close();
  dropsFrequencyOfStream.close();

  marksFrequencyOfStream.close();
  lengthOfStream.close();
  packetsOfStream.close();
  queueDelayN0OfStream.close();
  queueDelayN1OfStream.close();
  // colasn2.close();
gestion_frc_files (dropsFrequencyTraceFile, &dropsFrequencyOfStreamN0,
&dropsFrequencyOfStreamN1);
dropsFrequencyOfStreamN0.close();
dropsFrequencyOfStreamN1.close();



  NS_LOG_INFO("Generando representación gráfica");
  std::string lectura = "octave/Directorio.m", dibujar = "octave/leer.m";
```

```
  Generar_Matlab(dir, lectura, "Directorio.m");
  Generar_Matlab(dir, dibujar, "leer.m");
  if (BBR){
     Generar_Matlab(dir,"Estimacion_BDP_BBR.dat","qBDP_Estimation.dat");
     std::ofstream file("Estimacion_BDP_BBR.dat");
     file.clear();
     file.close();

     Generar_Matlab(dir,"Delivery_Rate_BBR.dat","qRatio_entrega.dat");
     std::ofstream fileD("Delivery_Rate_BBR.dat");
     fileD.clear();
     fileD.close();

     Generar_Matlab(dir,"Estimacion_completa_BBR.txt",
"Estimacion_BBR.txt");
     std::ofstream fileE("Estimacion_completa_BBR.txt");
     fileE.clear();
     fileE.close();



  }
```

```
  NS_LOG_INFO("Archivos generados");

}
```

7.2. Código de matlab

Directorio.m:

```
clear all

% Obtener todos los archivos con extensión .dat en el directorio actual

c = char('Patata')
b = c(1:5-2)
  files = dir("*.dat");

% Cargar cada archivo en el workspace
 disp(length(files))
```

```
valor = leer(1,15)



j = figure(2)

plot(n0_TCP_throughput(:,1), n0_TCP_throughput(:,2))
xlabel('Tiempo [segundos]')
ylabel("Ancho de banda de la transmisión [Mbps]");
title('Transmisión del Nodo 1');

k = figure(3)
plot(n0_TCP_cwnd(:,1), n0_TCP_cwnd(:,2))
xlabel("Tiempo [segundos]")
ylabel("Tamaño de la ventana [Unidades de paquete]")
title('Ventana de congestión en N1');
tamagno = files(3).bytes

if (tamagno != 0 )



   h = figure(1)

   subplot(3,3,1)
   plot (length(:,1), length(:,2));
```

```
title("COla en n2");



subplot(3,3,4)
plot (paquetes_cola(:,1), paquetes_cola(:,2));
title("Paquetes en n2");



subplot (3,3,2)
plot (n0_TCP_throughput(:,1), n0_TCP_throughput(:,2));
title("Descarte en N0");

subplot (3,3,5)
plot (n0_TCP_cwnd(:,1), n0_TCP_cwnd(:,2));
title("Ventana de congestión en N0");

subplot (3,3,8)
plot (n0_TCP_rtt(:,1), n0_TCP_rtt(:,2));
title("Rtt en N0");



subplot(3,3,3)
plot (n1_TCP_throughput(:,1), n1_TCP_throughput(:,2));
title("Descarte en N1");
```

```
    subplot (3,3,6)
    plot (n1_TCP_cwnd(:,1), n1_TCP_cwnd(:,2));
    title("Ventana de congestión en N1");

    subplot (3,3,9)
    plot (n1_TCP_rtt(:,1), n1_TCP_rtt(:,2));
    title("Rtt en N1");

elseif (tamagno == 0 )

    h = figure(1)

    subplot(3,2,1)
    plot (paquetes_cola(:,1), paquetes_cola(:,2));
    xlabel("Tiempo [segundos]")
    ylabel("Paquetes en la cola [unidad de paquete]")
    title("Paquetes en la cola de n2");

    subplot (3,2,2)
    plot (n0_TCP_throughput(:,1), n0_TCP_throughput(:,2));
    xlabel("Tiempo [segundos]")
```

```
ylabel(" Velocidad de la transmisión [Mbps]")
title("Descarte en N0");

subplot (3,2,4)
plot (n0_TCP_cwnd(:,1), n0_TCP_cwnd(:,2));
xlabel("Tiempo [segundos]")
ylabel("Ventana de Congestión [unidad de paquete]")
title("Ventana de congestión en N0");

subplot (3,2,6)
plot (n0_TCP_rtt(:,1), n0_TCP_rtt(:,2));
ylabel("Tiempo de tránsito [segundos]")
xlabel("Tiempo [segundos]")
title( "Rtt en N0");

subplot(3,2,3)
plot (drops_frequency_N0(:,1), drops_frequency_N0(:,2));
ylabel("Frecuencia de descarte [unidad de paquetes]")
xlabel("Tiempo [segundos]")
title("Descarte en N0");

else
```

```
end


saveas(h,'Length_descartesn0yn1.tiff')
saveas(h,'Length_descartesn0yn1.jpg')
saveas(h,'Length_descartesn0yn1.m')
saveas(j, "n0_TCP_throughput.tiff")
saveas(k, "n0_TCP_cwnd.tiff")
```

leer.m

```
function [resultado] = leer(inicio, fin)


  files = dir("*.dat");


   %for i = 1:length(files)-1
 for i = inicio:fin
   %if(i<4 || i>4)
     disp(i)
     filename = files(i).name
     elnombre_bueno = length(files(i).name)
     nombre = filename(1: (elnombre_bueno-4))
     if (files(i).bytes > 0)
       a = 1-strcmp(nombre,'m')
       if (a)
         variable = load(filename);
         assignin("base", nombre, variable)
```

```
    %    endif
     endif

   endif

end
resultado = 1

end
```