Universidad Nacional de Córdoba
Facultad de Matemáticas, Física, Astronomía y Computación

Laboratorio 3: Transporte

Redes y Sistemas distribuídos

Alvaro Medina, Ezequiel Pastore, Lorenzo Canovas y Matías Kühn

Córdoba, Argentina

Resumen

Este informe presenta un análisis del comportamiento de una red simulada en OMNET++, con el propósito de estudiar mecanismos de control de flujo y congestión que permitan un mejor manejo de recursos limitados, como tasas de transmisión acotadas y buffers de tamaño reducido. En el marco de la materia Redes y Sistemas Distribuidos, se desarrollaron simulaciones experimentales que permitieron observar métricas clave del desempeño de la red, tales como el retardo, la pérdida de paquetes, la carga ofrecida y la carga útil. A partir de estas observaciones, se ha propuesto y se ha implementado un algoritmo inspirado en el mecanismo de "parada y espera", cuya efectividad fue evaluada en distintos escenarios. Los resultados obtenidos muestran que, si bien dicho algoritmo mejora significativamente la confiabilidad de la entrega de datos, introduce un retardo adicional que se intensifica con el aumento de la carga ofrecida. Además, se identificaron diferencias en el comportamiento de la red: en situaciones de control de flujo, permite una mayor carga ofrecida antes de que el retardo por paquete comience a ser exponencial, en comparación del control de congestión. Este estudio permite comprender el impacto de los mecanismos de control sobre el desempeño de la red, y sienta las bases para posibles mejoras en el diseño de soluciones de comunicación más eficientes y adaptables.

Palabras clave: Parada y espera, red, nodo, OMNET++, control de flujo, congestión, buffers, queue, drop, carga Útil, carga ofrecida, Retardo, transmisor, receptor, paquetes, algoritmo, ventana, RTT, ACK, datarate, GenerationInterval.

1. Introducción

En el presente informe se trata una problemática planteada por la cátedra de Redes y Sistemas Distribuidos, en la cual debemos llevar a cabo un análisis del tráfico de red bajo tasa de datos acotadas y buffers con tamaños limitados. Esto implica proponer y diseñar soluciones de control de congestión y flujo, donde tendremos que comprender y generar modelos de red para poder realizar simulaciones de los eventos de interés.

Con una concepción más global, podemos ver que en la actualidad, la internet se ha vuelto indispensable en nuestra vida cotidiana, tanto en ámbitos personales como empresariales. Las implementaciones de redes eficientes y robustas han sido el principal desafío para brindar una infraestructura y conexión que puedan soportar y brindar un servicio de calidad frente al crecimiento exponencial de usuarios y dispositivos en la red. Sin embargo este crecimiento trae consigo un problema crítico: *la congestión de red*, la cual ocurre cuando un emisor transmite datos a un receptor a una tasa que excede la capacidad de carga de la red, volviéndola incapaz de entregar los paquetes a la velocidad de llegada. Este fenómeno genera pérdida de paquetes *(packet drops)* y colapsos en los buffers de la red particularmente en los enrutadores, afectando así a servicios como el streaming, transferencia de datos en tiempo real y videoconferencias.

Por otro lado, la congestión de red refiere a dos grandes cosas, el control del flujo y el control de congestión. El *control de flujo* se encarga de regular la transmisión de paquetes, para evitar la saturación del buffer del receptor, mientras que el control de congestión se encarga de ajustar dinámicamente las tasas de transmisión del ancho de banda para evitar la saturación de los buffers de los enrutadores. Para poder entender mejor sobre el problema, utilizamos la siguiente analogía: Al abrir la canilla(Ajuste de la tasa de transmisión) de un grifo de agua(Generator) y una mangue-

ra(Red de transmisión) podemos llenar un sumidero(Sink) como se puede observar en la **Figura 1**, notando que el agua representaría a los paquetes en circulación por la red.

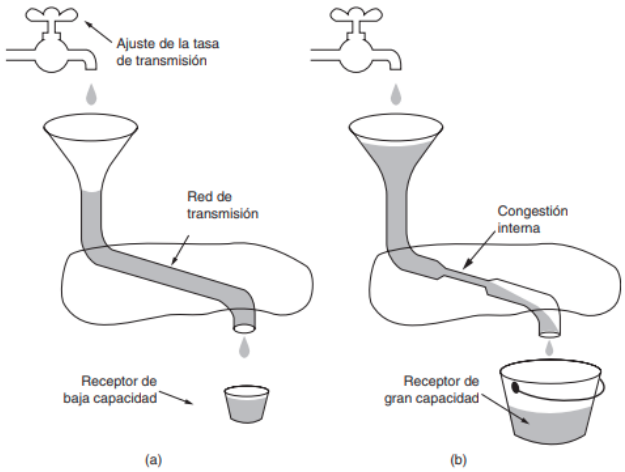


Figura 1: (a) Una red veloz que alimenta a un receptor de baja capacidad.
(b) Una red lenta que alimenta a un receptor de alta capacidad.

Entonces, para abordar esta situación y evaluar distintas situaciones vamos a utilizar simulaciones. Pero ¿Cómo haremos las simulaciones? Mediante la plataforma de simulación, *Omnet++*, la cual nos permite ver de forma gráfica la analogía que describimos con anterioridad, el comportamiento de la red en un intervalo de tiempo determinado y el flujo de la transmisión de paquetes ¿Por qué? Porque en la vida real, la transmisión de paquetes es demasiado rápida, por lo que se vuelve imposible realizar un estudio con esas velocidades, además de que se necesitaría una gran infraestructura para poder realizar estas pruebas.

El respectivo *análisis*[*] se desarrollará en dos partes:

- **Parte 1:** Análisis de control de congestión y flujo.
- **Parte 2:** Algoritmo de entrega de datos confiable para intentar solucionar el control de congestión y flujo.

Modelo de red en Omnet++

El modelo principal de la *Parte 1* se puede observar en la **Figura 2**, compuesta por *NodeTx*, *Queue* y *NodeRx*, nodo para la transmisión de paquetes, una representación de la red y un nodo para la recepción de los paquetes, respectivamente. La transmisión se realiza de forma unidireccional a una tasa de transmisión dada por una distribución exponencial con media $\lambda \in (0, 1)$, la cual se puede configurar desde *Omnet++*.

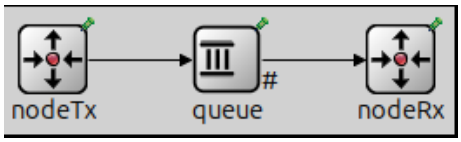
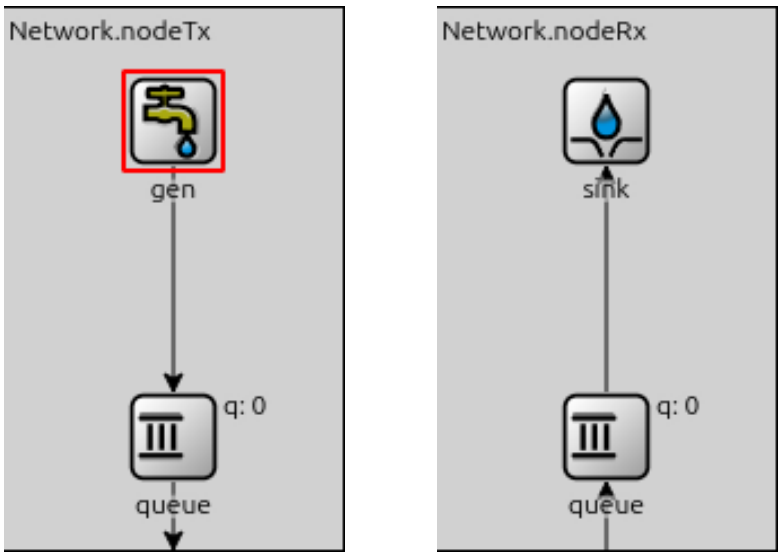


Figura 2: Modelo de red principal

Individualmente, *NodeTx* estará compuesta por uno de los componentes que utilizamos en la analogía del grifo de agua en la **Figura 1**, llamado *gen* (de generador), a su vez contendrá una *Queue*, la cual representará a un buffer. Dicha representación de *NodeTx* podemos encontrarla en

[*]A lo largo del informe se utilizarán términos concretos los cuales estarán definidos en la *sección 5 - Anexo*.

la **Figura 3a**. Idénticamente para *NodeRx*, estará compuesta por un sumidero, *Sink*, y un buffer. Podemos observarla en la **Figura 3b**.



(a) NodeTx (b) NodeRx

Figura 3: Modelos de transmisión y recepción

2. Métodos

Planteo de posibles soluciones

Dado el escenario de redes descrito en la **Sección 1**, se propone implementar un algoritmo de entrega de datos confiable, el cual debe garantizar *fiabilidad* y *eficiencia* en el uso de los recursos, minimizando el desbordamiento de buffers, teniendo así control en el flujo de datos. No necesariamente el algoritmo solucionará el control de congestión. Para la solución del problema, se pueden plantear diversos algoritmos; en particular, hay dos paradigmas muy importantes: Parada y Espera, y Tubería. Cada enfoque responde a las exigencias previstas de formas distintas.

Algoritmo de Parada y Espera

El método de *Parada y espera* se utiliza en contextos donde la simplicidad y la fiabilidad son importantes, sacrificando un poco de eficiencia. Generalmente, se aplica en contextos de baja latencia donde el **RTT** (round-trip time) es bajo comparado con el tiempo de copiar un paquete, permitiendo que solo se pueda enviar un paquete cuando llegue el ACK(confirmación de recepción). Para garantizar una entrega fiable y ordenada, al momento de implementar el algoritmo debemos suponer que se cumplen las siguientes condiciones:

- Los paquetes tienen número de secuencia.
- Se trabaja con *ACK's*: el receptor debe especificar el número de secuencia del paquete siendo confirmado, resultando importante para la comprobación de errores en los paquetes.
- Se usan retransmisiones de paquetes.

Funcionamiento/comportamiento

En primer lugar, el emisor envía todos los bits correspondientes al paquete P y se queda esperando un determinado tiempo la recepción del ACK del paquete P. Si llega el ACK a tiempo se confirma la correcta recepción del paquete por parte del receptor y se procede a enviar el

siguiente paquete. Si el ACK se pierde, se produce un *timeout* y se reenvía el paquete, ya que el emisor interpreta que este nunca llegó porque no llegó a recibir el ACK en la ventana de tiempo esperada.

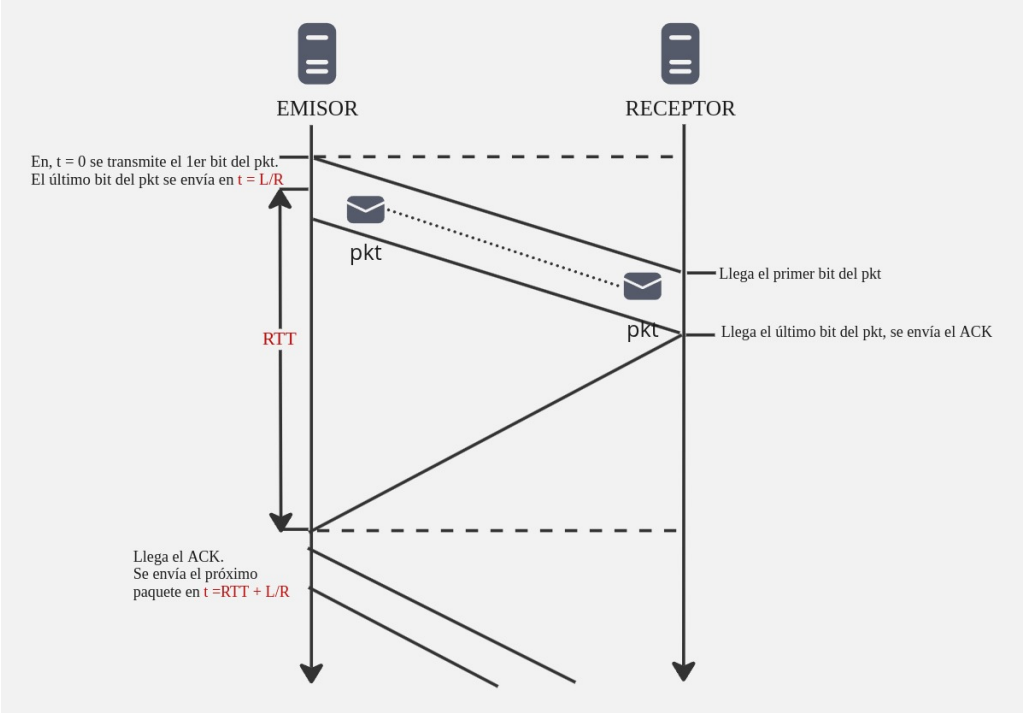


Figura 4: Representación a un ejemplo del algoritmo de Parada y espera

Algoritmos de Tubería

Los algoritmos de tipo tubería permiten al emisor enviar múltiples paquetes antes de recibir confirmaciones ($ACKs$), usando de forma más eficiente el canal (principalmente si hay latencia). Se destacan dos algoritmos principales: *Retroceso-N* y *Repetición Selectiva*.

Algoritmo de Retroceso-N

En este, el emisor mantiene una ventana de tamaño N (donde $N \leq MAX\_SEQ$)[**], que representa los paquetes que puede enviar sin recibir confirmación. El emisor transmite hasta N paquetes seguidos y espera confirmaciones del receptor, el cual envía ACKs acumulativos indicando el número del último paquete recibido en orden. Si ocurre una pérdida o un error y un paquete no es confirmado, el emisor activa un único temporizador asociado al paquete más antiguo no confirmado. Al expirar, se retransmiten todos los paquetes no confirmados desde ese punto en adelante, incluso si algunos fueron recibidos correctamente. El receptor descarta cualquier paquete fuera de orden y continúa enviando ACKs duplicados del último paquete correcto que recibió.

La ventana del emisor incluye:

- Paquetes enviados y pendientes de confirmación.
- Paquetes preparados para ser enviados, si hay espacio disponible.

[**] **MAX_SEQ**: Máximo número de secuencia, generalmente $(2^{32} - 1)$)

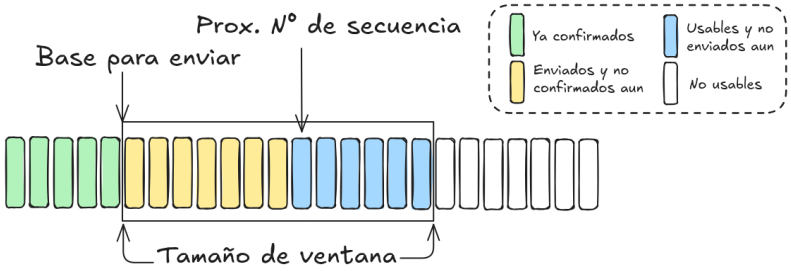


Figura 5: Representación de la ventana del algoritmo de Retroceso-N

Algoritmo de Repetición Selectiva

Este mejora el rendimiento en enlaces con alta latencia o alta tasa de error. Tanto el emisor como el receptor mantienen una ventana de tamaño N (donde $N \leq \frac{MAX\_SEQ+1}{2}$). A diferencia del *Retroceso-N*, cada paquete tiene su propio temporizador, y el emisor retransmite únicamente aquellos paquetes cuya confirmación no ha llegado a tiempo. El receptor acepta y almacena paquetes que llegan fuera de orden, y responde con un ACK individual por cada paquete recibido. Cuando posteriormente llegan los paquetes faltantes, puede ordenar y entregar los datos en orden al nivel superior.

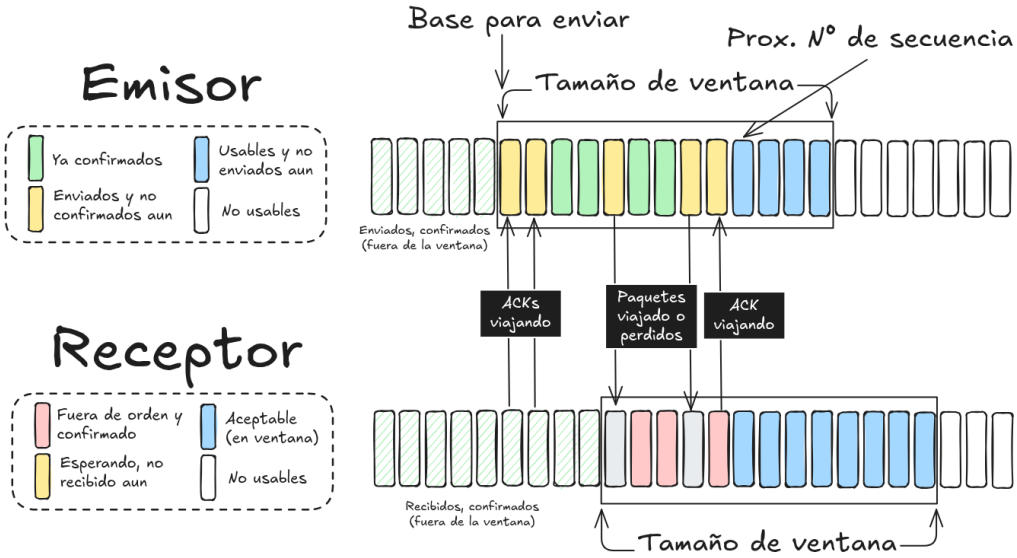


Figura 6: Representación de la ventana del algoritmo de Repetición Selectiva

La ventana del emisor incluye:

- Paquetes enviados pero aún no confirmados.
- Paquetes ya confirmados pero aún dentro de la ventana (por desorden).
- Paquetes listos para ser enviados.

La ventana del receptor incluye:

- Paquetes ya recibidos (fuera de orden o en orden esperando a ser consumidos).
- Espacio para paquetes esperados.

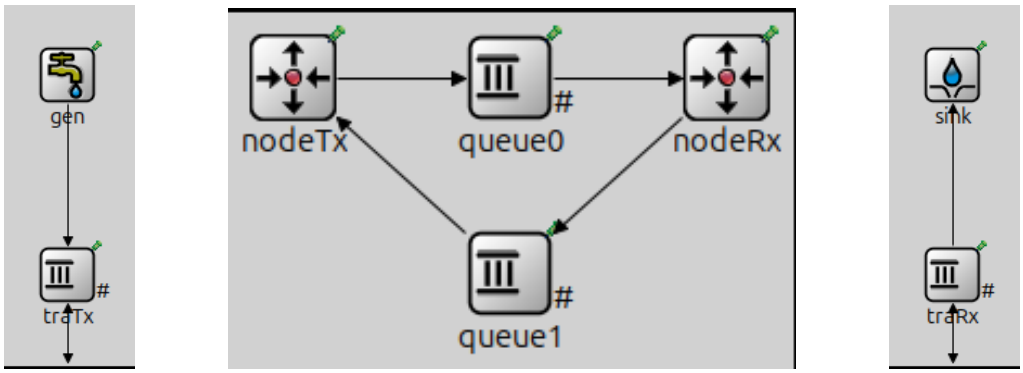
Solución al problema de congestión y control de flujo

Como solución a nuestro problema, es decir, al control de flujo y de congestión, decidimos implementar un algoritmo muy similar al de entrega de datos confiable llamado *Parada y Espera*, visto en la parte teórica de la materia, salvo que en esta nueva versión, no contamos con un temporizador ya que como no hay saturación de buffers, el uso de un temporizador es innecesario. Cabe aclarar que en la vida real, es indispensable el uso de un temporizador ya que sí ocurren saturaciones de buffer y no conocemos el comportamiento interno de una red.

El grupo optó por este algoritmo debido a su simplicidad a la hora de implementarlo y entender su funcionamiento, en comparación con otros algoritmos como lo son los de tubería, de los cuales se dio una explicación resumida antes.

Modelo que soporta el algoritmo de Parada y espera

Para la correcta implementación de este algoritmo, debimos hacer ciertas modificaciones en la estructura de la red anterior. A continuación, podemos apreciar la red resultante en la **Figura 7**.



(a) NodeTx (b) Network (c) NodeRx

Figura 7: Modelos de transmisión y recepción

Vamos a explicar cada sección en detalle:

- **Network:** Está compuesta por los mismos nodos que antes, salvo que ahora se agrega una segunda cola. ¿Cómo funciona? *NodeTx* le va a mandar paquetes de datos de a uno por vez a *NodeRx* por medio de una *queue0*, la cual representa la red (enrutadores interconectados). Luego, *NodeRx* enviará la respectiva confirmación de recepción al *NodeTx* si le llegó el paquete, por medio de la *queue1*.
- **NodeTx:** Está compuesto por el *Generator* y un *buffer*, al igual que antes, pero ahora el buffer (*traTx*) va a tener dos entradas y una salida. La salida es usada para mandar los paquetes de datos hacia *NodeRx*, una de las entradas sirve para recibir los paquetes que va generando *Generator* y la otra sirve para recibir las confirmaciones de recepción de *NodeRx*.
- **NodeRx:** Está compuesto también por el *Sink* y un *buffer*, al igual que antes, con la diferencia de que ahora nuestro buffer (*traRx*) tendrá una entrada y dos salidas. La entrada será para recibir los paquetes que le llegan de *NodeTx*, mientras que las salidas, una será para pasarle los paquetes al *Sink* y la otra será para mandarle las confirmaciones de recepción al *NodeTx*.

Detalles a tener en cuenta: *queue1* tiene el mismo objetivo que *queue0* y es proveer una abstracción y representación de la red y de todos los enrutadores conectados, con la diferencia de que queue1 tiene una capacidad de procesamiento infinita, por lo que jamás vamos a tener problemas de congestión en esta parte, ya que no es nuestro objetivo en este laboratorio.

3. Resultados

3.1. Casos de Estudio

A continuación, analizaremos 2 casos de estudio con el objetivo de visualizar, comparar y comprender, el funcionamiento y la eficiencia del algoritmo propuesto como solución. Para ello, vamos a realizar los casos antes de la implementación del algoritmo y también una vez que esté implementado. De esta manera, podremos analizar y obtener conclusiones de la solución.

Los casos de estudio son:

- **Caso 1:**
  - NodeTx a Queue: datarate = 1Mbps y delay = 100 μs
  - Queue a NodeRx: datarate = 1Mbps y delay = 100 μs
  - QueueNodeRx a Sink: datarate = 0.5 Mbps

  Delay se refiere a la velocidad de procesamiento de datos, mientras que *datarate* es el tiempo que va a tardar un paquete en ir desde el origen hacia el destino. Podemos ver que de *NodeTx* a Queue y de Queue a *NodeRx*, los paquetes van a ser procesados todos a la misma velocidad y todos van a tardar el mismo tiempo en ir desde un lugar hacia el otro, por lo que en *NodeTx* y Queue no habrá problemas de congestión ni de flujo. El problema, reside en el *NodeRx*, ya que los datos en la Queue del *NodeRx* son enviados a la aplicación con la mitad de tasa de transferencia que la que se los envía Queue, por lo que, eventualmente, el buffer comenzará a llenarse hasta saturarse y provocar la pérdida de paquetes. Esto es lo que se le denomina como problema de control de flujo.

- **Caso 2:**
  - NodeTx a Queue: datarate = 1Mbps y delay = 100 μs
  - Queue a NodeRx: datarate = 0.5 Mbps y delay = 100 μs
  - QueueNodeRx a Sink: datarate = 1Mbps

  En este segundo caso, podemos ver que los datos desde *NodeTx* a Queue y desde QueueNodeRx a Sink, van a ser procesados y enviados a la misma velocidad, por lo que en y *NodeRx* no se generarán problemas. El problema reside en Queue, cuando los datos son enviados desde Queue a *NodeRx*, con la mitad de tasa de transferencia de datos que la que se los envía *NodeTx*, por lo cual Queue comenzará a llenarse hasta saturarse y perder paquetes. Por otro lado, Queue es simplemente una sola cosa (una simple cola), podríamos pensar que este problema se trata de un problema de control de flujo, ya que estaríamos saturando como un solo nodo, pero en realidad, en este laboratorio, Queue es una representación de toda la red y todos los nodos conectados que hay en el medio entre el *NodeTx* y *NodeRx*, por lo que si Queue se satura, estaríamos frente a una saturación general de la red, por lo que el problema en este segundo caso, es un problema de control de congestión.

Análisis de los resultados

Métricas

Dados los casos de estudio, que servirán como base para el análisis, corresponde introducir las métricas que se utilizaron para comparar el desempeño de las redes simuladas, es decir, aquellas que ponen a prueba el funcionamiento de la red con el algoritmo y sin el algoritmo.

Una consideración importante a tener en cuenta para la Parte 1 y Parte 2 es la variable *GenerationInterval*, utilizada en *Omnet++*

GenerationInterval:

- **Parte 1:** Para poder simular un caso más completo, en el cual la red pueda generar pérdida de paquetes, demora en la entrega de paquetes, saturación de buffers y congestión, nos pareció adecuado fijarlo a un valor de $\lambda = 0{,}1$.

- **Parte 2:** Para obtener dinamismo en los resultados en las distintas simulaciones, disminuyendo la tasa de generación de paquetes, mediante el cálculo de $\left(\frac{10}{Nsim+1}\right)$. Esta métrica es fundamental para la Parte 2, ya que en cada corrida de la simulación, se obtienen diferentes cantidades de paquetes enviados y recibidos. Y esto para ver si la implementación de la red se ve afectada por ella.

Carga Útil vs Carga Ofrecida:

Con la variación del *GenerationInterval* es fácil ver que la *Carga Ofrecida*, la cual representa a los paquetes producidos por segundo por el *Generator*, tiene un gran impacto en la red en cuanto a lo que recibe el *Sink*. Esto se debe a que, mientras más rápidamente se generen los paquetes y se lancen a la red dependiendo del caso que se esté simulando, más rápido se saturará el buffer receptor.

Al saturarse dicho buffer, se empezarán a descartar paquetes entrantes. Por lo tanto, nos interesa conocer cuántos paquetes se utilizan efectivamente (*Carga Útil*) en función de los que se ofrecen.

Retardo vs Carga Ofrecida:

Sea Δt el tiempo que transcurre desde que un paquete es creado por *Generator* hasta que llega a *Sink*, tal que:

$$\Delta_t = t_{\text{llegada}} - t_{\text{salida}}$$

donde:

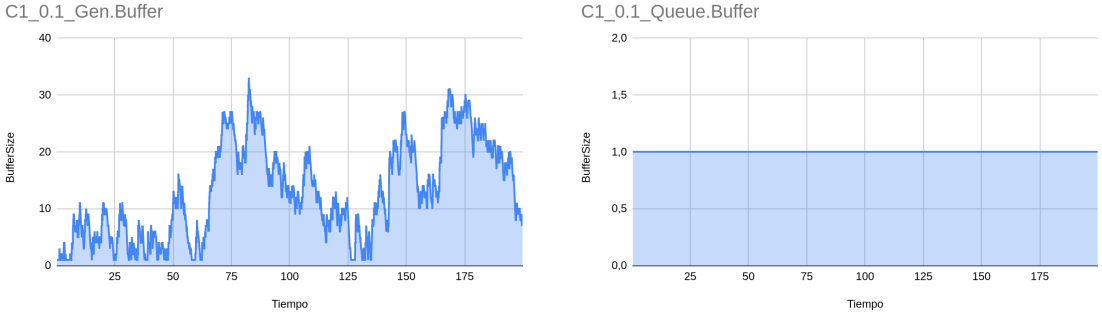
- t_{llegada}: Tiempo en que el paquete llega al *Sink*.
- t_{salida}: Tiempo en que el paquete es generado por el *Generator*.

Queremos cuantificar Δt para comparar y analizar el desempeño en las distintas situaciones para cada paquete transmitido, y así ver cómo varía el retardo promedio en función de la carga ofrecida.

Gráficas

Parte 1 - Caso 1:

En los gráficos de la **Figura 8** se puede observar cómo el tamaño del buffer del emisor (Gen) varía entre 1 y 35 paquetes. Por otro lado, el buffer de la Queue (que representa a la red) se mantiene constantemente en 1 ya que no hay un problema de control de congestión, mientras que el buffer del receptor (Sink) comienza a crecer de forma lineal con una pendiente pronunciada, alcanzando su capacidad máxima de 200 paquetes aproximadamente a los 46 segundos de la simulación, observado en la **Figura 9**. En conjunto con esto, en la gráfica de Drop se puede apreciar que a partir de ese mismo instante comienzan a aparecer paquetes descartados.

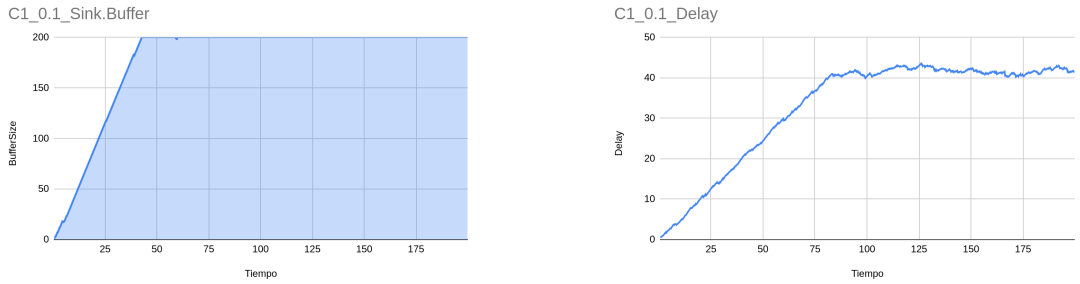


(a) Buffer de NodeTx

(b) Buffer de Queue

Figura 8: Resultados obtenidos en la simulación - Caso 1

Esto evidencia que el emisor está saturando el buffer del receptor, ya que la velocidad de consumo de la aplicación receptora es menor que la carga ofrecida por el emisor. Esta situación corresponde claramente a un problema de control de flujo.



(a) Buffer de Sink (b) Delay de la red

Figura 9: Resultados obtenidos en la simulación - Caso 1

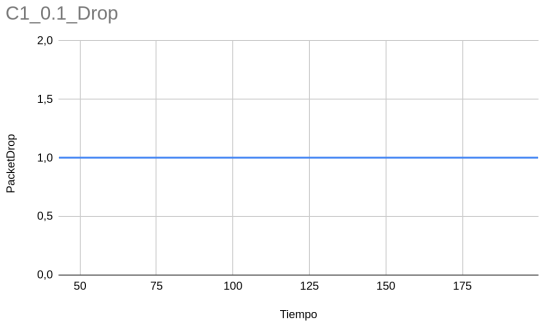
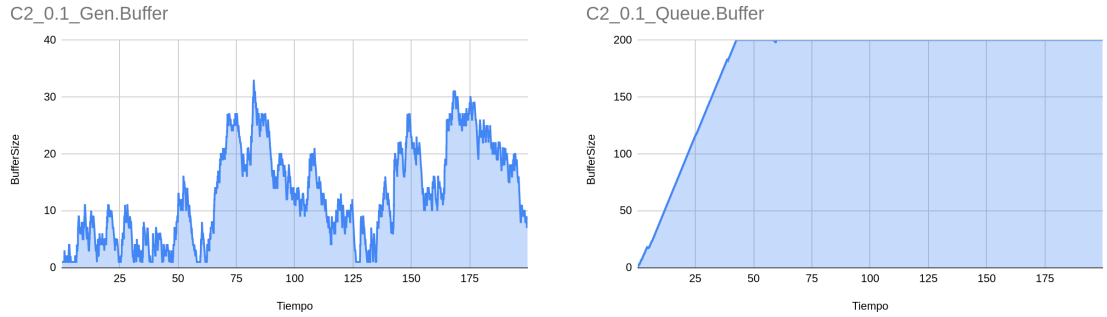


Figura 10: Resultados obtenidos en la simulación - Caso 1

Parte 1 - Caso 2:

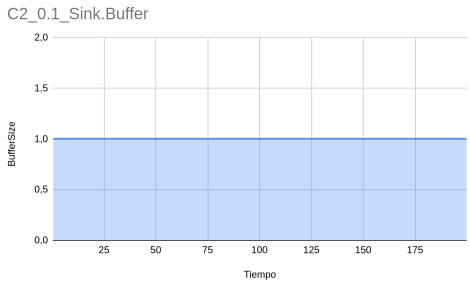
En los gráficos de la **Figura 11** se puede observar cómo el tamaño del buffer del emisor (Gen) varía entre 1 y 32 paquetes. El buffer del receptor (Sink) se mantiene constantemente en 1 ya que no hay problema de control de flujo, mientras que el buffer de la Queue (que representa a la red) comienza a crecer de forma lineal con una pendiente pronunciada, alcanzando su capacidad máxima de 200 paquetes aproximadamente a los 46 segundos de simulación. De manera conjunta, en la gráfica de Drop, se aprecia que a partir de ese momento se comienza a descartar paquetes.



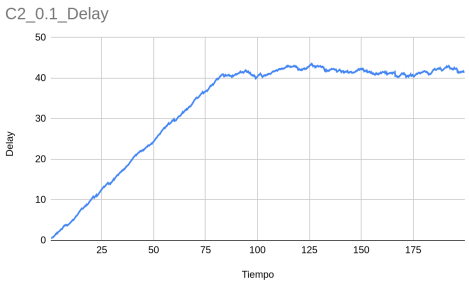
(a) Buffer de NodeTx (b) Buffer de Queue

Figura 11: Resultados obtenidos en la simulación - Caso 2

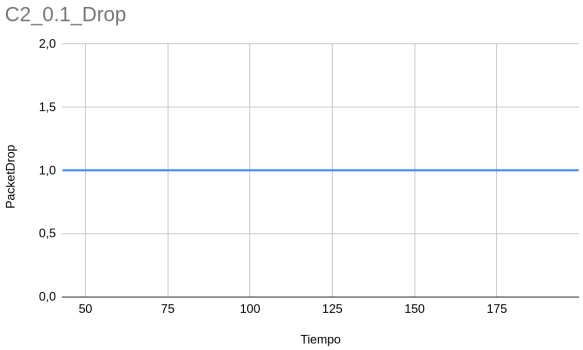
Esto indica que el emisor está saturando el buffer de la Queue, es decir, está saturando la red. Esto se debe a que para esta configuración de la red, el emisor transmite paquetes el doble de rápido de lo que puede transmitir Queue al receptor. Esta situación corresponde claramente a un problema de control de congestión.



(a) Buffer de Sink



(b) Delay de la red



(c) Pérdida de paquetes

Figura 12: Resultados obtenidos en la simulación - Caso 2

El siguiente análisis se realiza sobre los gráficos de la **Figura 13 y 14**, los cuales representan la "Carga Útil vs Carga Ofrecida" y "Retardo vs Carga Ofrecida" sin algoritmo de entrega confiable.

Carga Útil vs Carga Ofrecida:

A medida que la carga ofrecida aumenta, la carga útil también lo hace de manera casi lineal hasta que la carga ofrecida alcanza un valor aproximado de 5. A partir de dicho punto, la carga útil se estabiliza (se pone constante en 5), lo cual nos muestra que la red alcanzó su máxima capacidad efectiva de transmisión, es decir, esto indica que la red está funcionando a plena capacidad y ya no puede aprovechar mayor carga ofrecida debido a la congestión, por lo que los paquetes empiezan a perderse al llenarse los buffers de la red, ya que son desechados por la red.

Retardo vs Carga Ofrecida:

El retardo es mínimo al principio, una vez que la red alcanza su máxima capacidad efectiva de transmisión que vimos en el análisis de la **Figura 13 y 14** (aprox. 5), el retardo se incrementa bastante, indicando congestión en la red. Esto nos dice que el tráfico sigue aumentando, pero la red ya no puede procesarlo eficientemente. Hay inestabilidad visible (zigzagueo), lo cual es porque el gráfico está compuesto por resultados de simulaciones diferentes. En el momento en el que llega a una carga ofrecida cercana a 10, se puede observar un crecimiento brusco debido a la congestión.

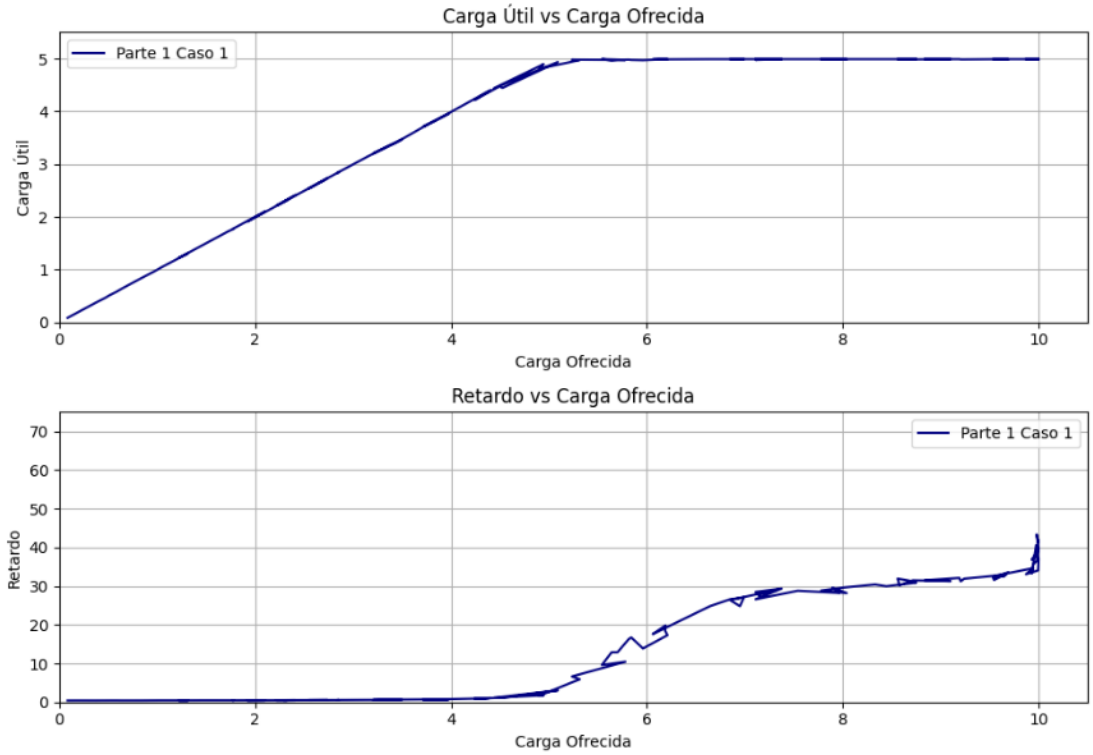


Figura 13: Evaluación de las métricas - Parte 1 Caso 1

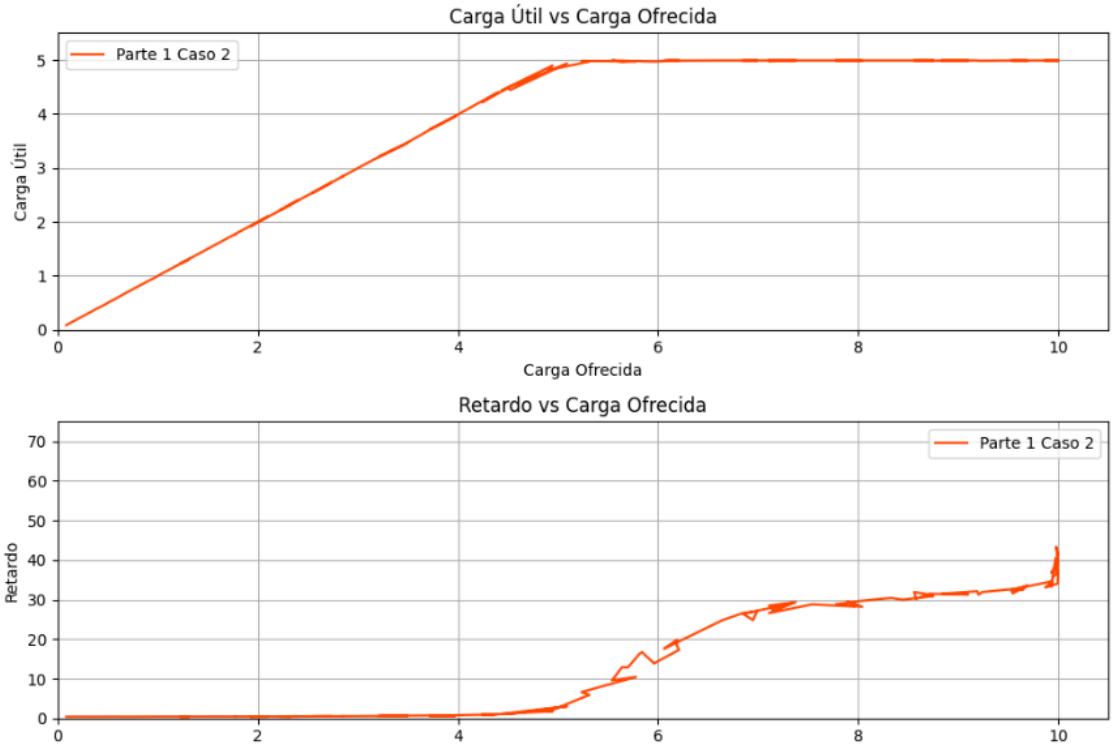


Figura 14: Evaluación de las métricas - Parte 1 Caso 2

Parte 2 - Caso 1 y 2:

El siguiente análisis se realiza sobre los gráficos de la **Figura 15 y 17**, los cuales representa la "Carga Útil vs Carga Ofrecida" y **Figura 16** para "Retardo vs Carga Ofrecida", en una red con problemas de congestión con un algoritmo de entrega confiable propuesto por nuestro grupo:

Carga Útil vs Carga Ofrecida:

Debido a que el algoritmo que implementamos, se puede observar que el gráfico es lineal, esto se debe por que no importa cual sea la tasa de generación de paquetes del emisor ya que solo envía un paquete nuevo al recibir el ACK del paquete anterior.

Retardo vs Carga Ofrecida:

Como resultado del algoritmo, se observa como aumenta exponencialmente el retardo de los paquetes a medida de que aumenta la tasa de generación de paquetes del emisor, ya que se generan muchos paquetes, pero la velocidad de la red queda estancada y se empiezan a acumular los paquetes en el buffer del emisor.

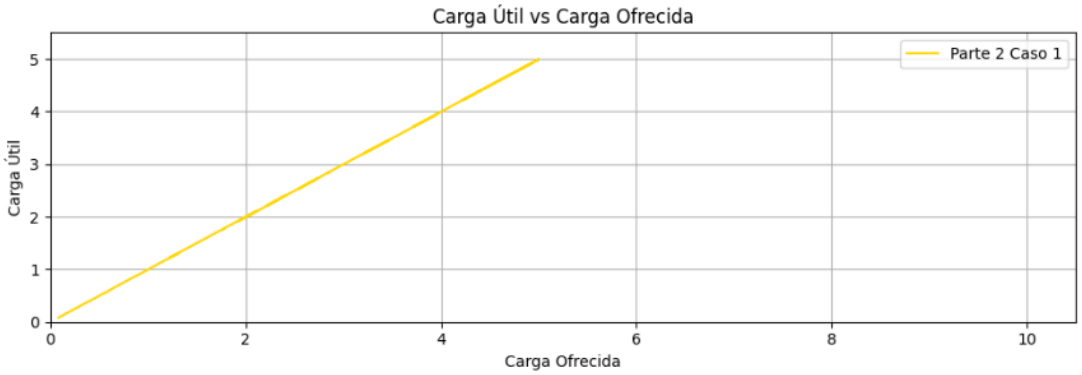


Figura 15: Evaluación de las métricas - Parte 2 Caso 1

Figura 16: Evaluación de las métricas - Parte 2 Caso 1

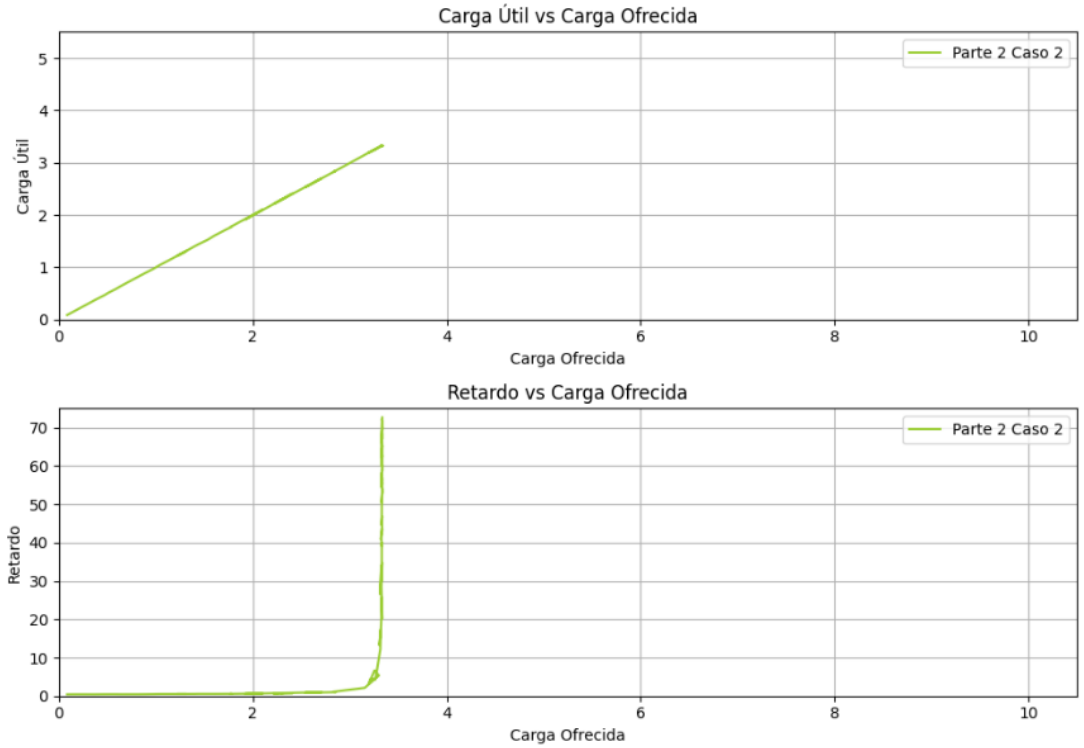


Figura 17: Evaluación de las métricas - Parte 2 Caso 2

4. Conclusión

En la **Figura 18** y la **Figura 19** se presentan los gráficos de "Carga Útil vs Carga Ofrecida" y "Retardo vs Carga Ofrecida" para los diferentes casos de estudio, tanto con como sin la implementación del *algoritmo de entrega confiable*.

En primer lugar, cuando no se utiliza el algoritmo de entrega confiable, se observa que, al saturarse la red, se pierden numerosos paquetes. Sin embargo, el retardo se mantiene relativamente bajo incluso a altas tasas de carga ofrecida. Este comportamiento (similar al utilizar UDP) puede ser beneficioso en aplicaciones donde la pérdida ocasional de paquetes es tolerable, como en servicios de *streaming*. No obstante, al incorporarlo, se garantiza la entrega de los datos pero a costa de un aumento significativo en el retardo.

En situaciones de control de flujo (Parte 2 - Caso 1), se logra transmitir y recibir una mayor cantidad de paquetes en comparación con escenarios de congestión de red (Parte 2 - Caso 2). Esto se debe a que, en presencia de control de flujo, los paquetes llegan rápidamente al receptor (aunque puedan demorar en ser procesados), permitiendo enviar el ACK de forma inmediata. En cambio, en situaciones de congestión, los paquetes tardan más en llegar al receptor debido al tráfico en la red, incrementando el tiempo total hasta recibir el ACK.

En cuanto al retardo promedio, este se ve fuertemente influenciado por la acumulación de paquetes en el buffer del emisor. A medida que aumenta la carga ofrecida, más paquetes quedan en espera debido al mecanismo del algoritmo, lo que puede producir una acumulación en el buffer del receptor.

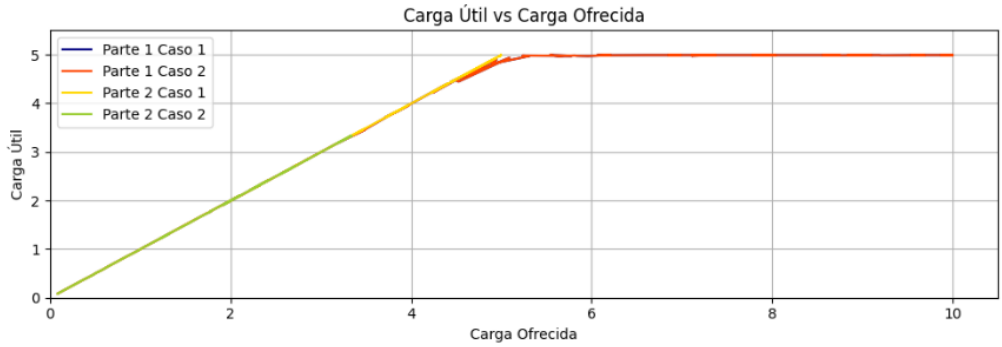


Figura 18: Representación del desempeño de todos los casos

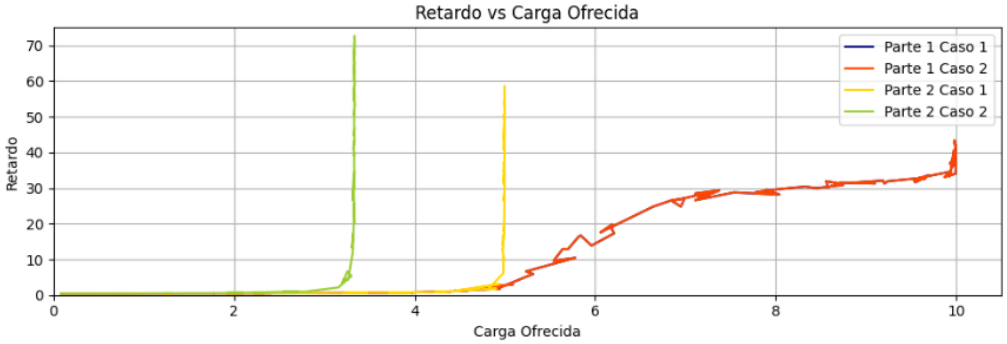


Figura 19: Representación del desempeño de todos los casos

Discusión

Limitaciones

- En la figura *Carga útil vs carga ofrecida* podemos observar que en un momento la carga útil se mantiene constantemente en 5 en términos de la carga ofrecida, lo cual se debe a que la red no se cae a pesar de estar saturada y, por ende, no se ve reflejada la pérdida de paquetes. Esto es una limitación en el diseño de la red proporcionada por la cátedra.
- Conocimiento del grupo sobre *Omnet++*, otros tipos de gráficos para representar y analizar el problema.
- La falta de experiencia en redacción técnica por parte de los integrantes del equipo generó demoras en la elaboración del informe, reduciendo el tiempo disponible para realizar ajustes y mejoras adicionales.

Mejoras

- El *timeout* del algoritmo de Parada y Espera no fue implementado debido a la situación dada por la simulación (tamaños de buffers grandes, velocidad de la red), y al incremento de complejidad en el código. Debido a esta omisión, pueden presentarse problemas de *deadlock*.
- Graficar la pérdida de paquetes mencionada en el apartado de *Limitaciones*.
- Que los paquetes de *feedback* no sean instantáneos para que sea más realista.
- Que los paquetes en general pasen por una red en la cual hay otros paquetes en circulación.
- Que el paquete de *feedback* de información sea sobre su tamaño de *buffer*.

5. Anexo

A continuación, se describen los principales componentes y definiciones que se utilizaron a lo largo del informe para otorgar una comprensión clara y coherente del funcionamiento del sistema, permitiendo así recorrer su análisis y diseño de manera fluida y estructurada.

- **Parte 1:** Hace referencia a la parte de análisis, en el cual se busca observar los problemas de control de congestión y flujo.
- **Parte 2:** Hace referencia a la parte en la cual se da un algoritmo de control de congestión y de flujo para la red.

Transmisor

- **Parte 1:** `NodeTx`, es el que genera y envía paquetes de datos al receptor.
- **Parte 2:** `NodeTx`, es el que genera, envía paquetes de datos y recibe paquetes de *feedback* del receptor.

Receptor

- **Parte 1:** `NodeRx`, es el que recibe los paquetes de datos del transmisor.
- **Parte 2:** `NodeRx`, es el que recibe los paquetes de datos y envía paquetes de *feedback* al transmisor.

Red

- **Parte 1:** `queue`, recibe paquetes de datos del transmisor y los envía al receptor.
- **Parte 2:** `queue0` y `queue1`, los cuales sirven para enviar los paquetes de datos al receptor y enviar paquetes de *feedback* al transmisor.

Buffer

Es un arreglo que contiene paquetes esperando a ser enviados a la red desde el transmisor, procesados por el receptor y para simular la red (`queue`, `queue0`, `queue1`, `traTx` y `traRx`).

Tipos de paquetes

- **Paquete enviado:** Es un paquete de datos que salió del transmisor hacia la red.
- **Paquete recibido:** Es un paquete de datos que salió del transmisor, llegó al receptor y fue procesado por el `sink`.
- **Paquete dropeado:** Es un paquete que llegó a un elemento de la red que contiene *buffer* y, al estar lleno, elimina el paquete, por lo que no llega a ser procesado por el `sink`.
- **Paquete en la red:** Hace referencia a los paquetes que estaban siendo enviados y a aquellos que se encontraban en algún *buffer* al terminar la simulación.