Laboratorio 3: Transporte

Análisis de flujo y congestión

Grupo 22
Integrantes: Lazaro Manuel Cugat, Ezequiel Lauret, Santiago Morales

Abstract

En este laboratorio analizaremos una red básica mediante la herramienta de simulación de eventos OMNET + +, con la que estudiaremos el comportamiento de dos escenarios típicos en la capa de transporte :

1) Problema de flujo
2) Problema de congestión

Luego se modificará esta red añadiendo la posibilidad de enviar paquetes de control, que nosotros usaremos para tratar estos problemas con el algoritmo explicado en este informe, en el cual se busca básicamente bajar la velocidad de envío de los paquetes cuando ocurre un cuello de botella.

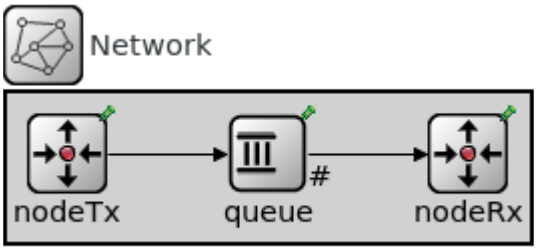
Introducción

Las redes están compuestas por nodos y conexiones entre estos, con distintas capacidades y tasas de transferencia, lo que produce lo que llamamos cuellos de botellas, haciendo referencia al problema donde los nodos reciben paquetes más rápido de lo que pueden procesarlos. Cuando estos se encuentran en el nodo receptor es un **problema de flujo (Caso 1)**, y cuando pasa entre el transmisor y el receptor es un **problema de congestión (Caso 2)**, .

Estructura de red

Lógica del Network

Nuestra red está compuesta por un generador (gen) una cola (queue) y un destino (sink)
El esquema de esta red sería el siguiente:

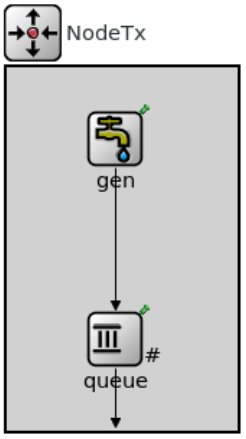


Queue intermedia:

Recibe paquetes desde su “gate in” (entrada) y los almacena en un buffer esperando para enviarlos por “gate out” (salida) a medida que es posible. En el caso en que el buffer se encuentra saturado, descarta los nuevos paquetes que llegan y se pierden.

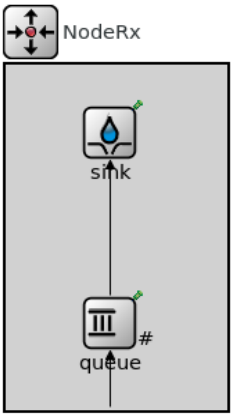
NodeTx:

Está compuesto por un generador y una queue, el generador produce paquetes cada cierto intervalo de tiempo y la queue recibe paquetes producidos por el generador y los envía a la queue intermedia.



NodeRx:

Está compuesto por una queue y un consumidor (sink), la queue recibe paquetes enviados por la queue intermedia y los envía al consumidor.



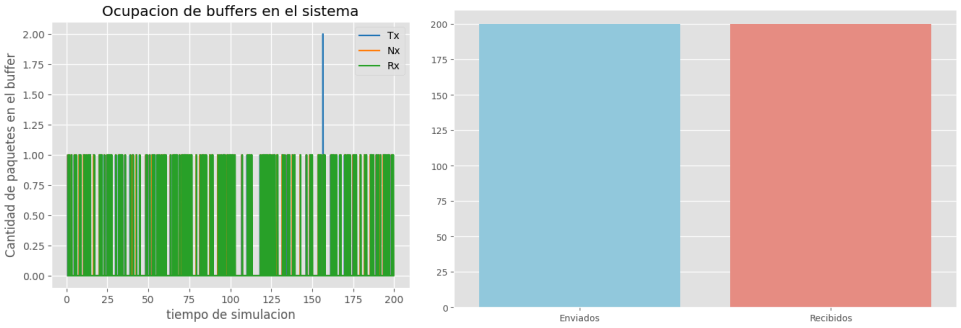
Experimentos

Caso de estudio 1

El primer problema a analizar será el control de flujo (buffer del receptor lleno). En esta situación, llegarán paquetes a la queue de NodeRx a mayor velocidad de lo que la misma los envía, de esta manera se satura la queue y se perderán paquetes al no poder ser almacenados en el buffer, porque se encuentra lleno.

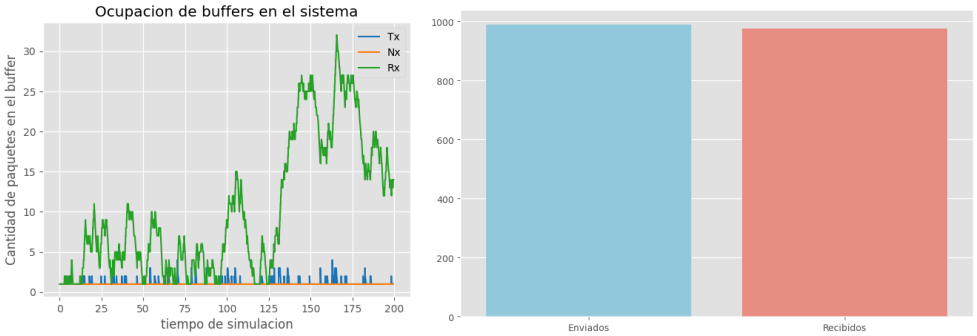
Utilizamos un buffer que permite almacenar como máximo 200 paquetes para la queue del NodeRx, modificamos la velocidad en la que los paquetes se generaban y obtuvimos los siguientes resultados:

generationInterval = exponential(1)



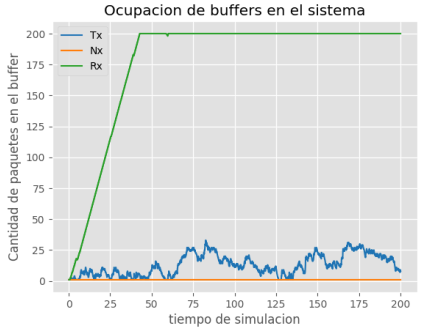
En este primer caso con un intervalo de generación de paquetes exponential(1), podemos ver como la pérdida de paquetes no es un problema, ya que el buffer no solo que nunca se llega a saturar, sino que la cantidad de paquetes en él es constantemente 1, es decir que le da tiempo a procesar el paquete y mandarlo, antes de que llegue otro. Por esta razón es que no es necesario en este caso controlar el flujo (por la baja velocidad de generación de paquetes).
En los siguientes casos probaremos bajando este número de intervalo de generación de paquetes, hasta ver si tenemos un problema de control de flujo. Primero probaremos con el caso de:

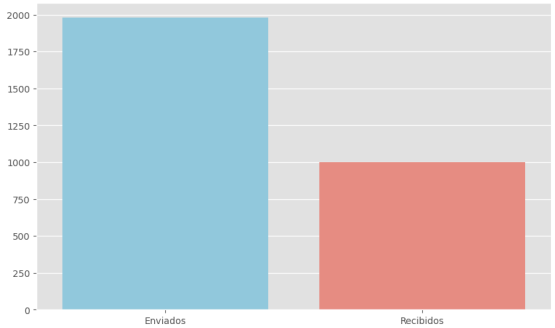
generationInterval = exponential(0.2)

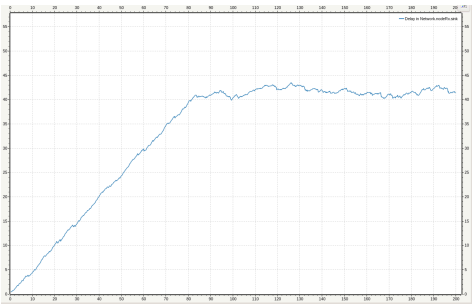


En este caso podemos ver que el buffer de la queue del nodo NodeRx no se llena tampoco (igual que en el caso anterior), por lo tanto no ocurre el hecho de saturarse y no se ve el problema de pérdida de paquetes por control de flujo. Busquemos aumentar un poco más la velocidad de generación de los paquetes (generationInterval) para que de esta manera logremos saturar el buffer:

generationInterval = exponential(0.1)







Disminuyendo el intervalo en el que se crean los paquetes a un número muy bajo, se presenta el problema de control de flujo. Podemos ver que el buffer de la queue en el nodo NodeRx se llena rápidamente por lo tanto muchos paquetes serán dropeados al no poder ser almacenados en el buffer.
Podemos ver cómo a partir de este intervalo de generación de paquetes, mientras más bajo sea el número, más paquetes serán perdidos, ya que más rápido se saturará el buffer.

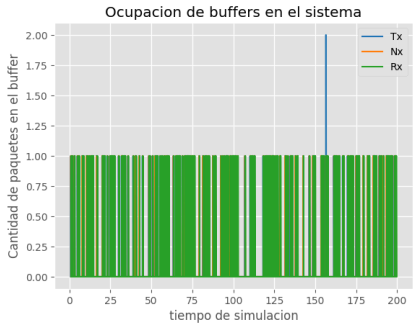
Y por el ultimo grafico, viendo el delay promedio, notamos que éste va aumentando hasta llegar a un máximo, de donde lo más importante a notar es que en esta gráfica no van a notar diferencias entre el caso 1 y caso 2, ya que el problema es el mismo, los dos tienen un cuello de botella en diferentes lugares pero es el mismo cuello de botella.

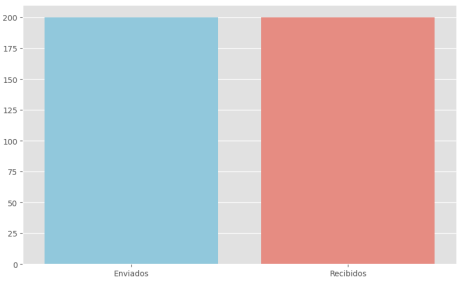
Caso de estudio 2

El segundo problema a analizar será el control de congestión(buffer en la queue intermedia lleno), que se ve reflejado en el caso de estudio 2. En esta situación ocurrirá lo mismo que en el caso de estudio 1, pero la queue a saturarse será la queue intermedia

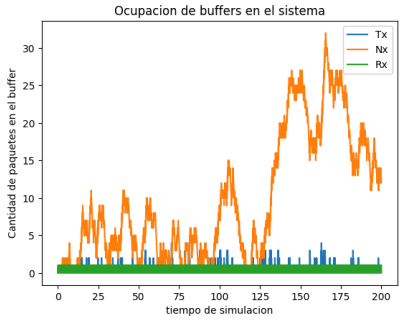
Hicimos las mismas pruebas que para el caso anterior:

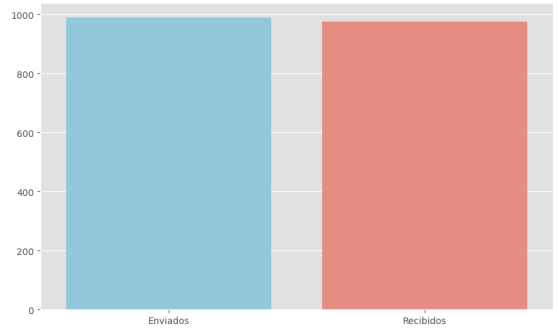
generationInterval = exponential(1)



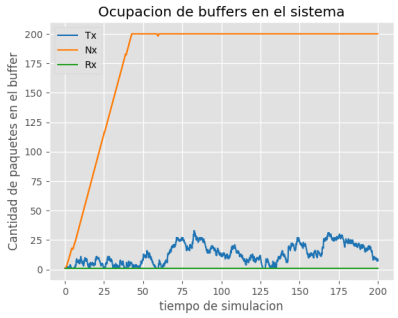


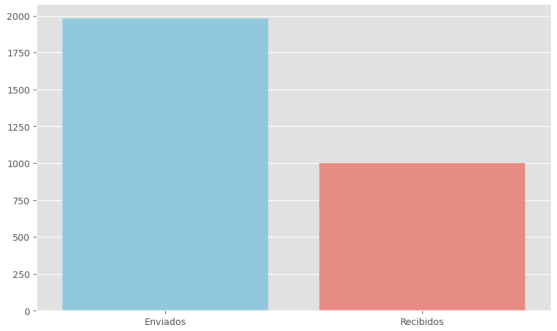
generationInterval = exponential(0.2)

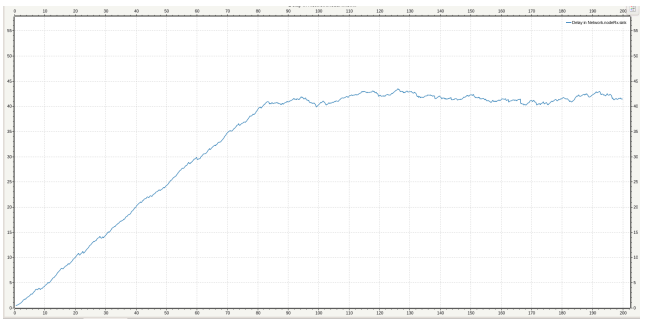




generationInterval = exponential(0.1)





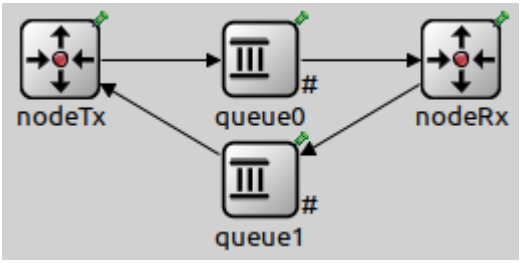


En este caso sucede lo mismo que en el caso 1 pero a diferencia de llenarse el buffer de la queue del NodeRx se llena el buffer de la queue intermedia , ocurrirá un problema de congestión cuando los tiempos de generación de paquetes sean muy pequeños produciendo así pérdidas de paquetes.

Métodos

Sistema de control de flujo y congestión

Para tratar estos problemas, se modificó el esquema del Network, agregando una conexión de “retorno” entre nodeTx y nodeRx, para que nodeRx le envíe mensajes de tipo "feedback" a nodeTx y de esta manera solucionar los problemas de congestión y flujo.



Algoritmo Implementado:
Lo primero que hacemos es establecer un “límite” a los buffer de las colas de un 80% de su capacidad, de tal forma que si alguno de estos alcanza este “límite” se crea y envía un paquete de tipo “FeedbackPkt”.
Entonces lo interesante es cómo tratan este tipo de paquetes los nodos Transmisor (NodeTx) y receptor (NodeRx):

Transmisor

La lógica de este nodo es identificar los paquetes según su tipo, “datapacket” o “feedback”, y tratarlos según el estado actual de la red, osea si está congestionada o no.
Si la red **no está congestionada**:
Si llega un paquete de tipo “datapacket”, simplemente lo envía.
Si llega un paquete de tipo “feedback”, el nodo se encarga de ralentizar el tiempo de envío de los paquetes con el objetivo de otorgarle tiempo al nodo receptor para liberar espacio en el buffer y así evitar la pérdida de paquetes.
Si la red **está congestionada**:
En nuestra implementación este caso ocurre durante 6 segundos luego de la llegada de un paquete “feedback”.
Si llega un paquete de tipo “datapacket”, se envía pero con un ligero “delay” para así darle tiempo al nodo receptor para liberar un poco el buffer.
Si llega un paquete de tipo “feedback”, simplemente se ignora.

Receptor

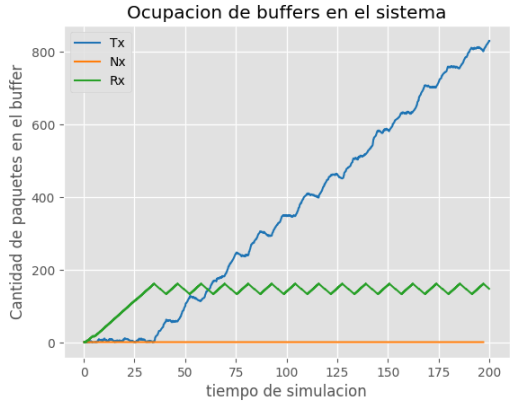
La idea de este nodo es crear cuando sea necesario un paquete de tipo “feedback” avisando así al transmisor que debe bajar la velocidad en la que envía los paquetes.
A diferencia del anterior, solamente identifica el tipo de paquetes que le llegan y los trata de forma que:
Si le llega un paquete de tipo “**datapacket**”, el nodo verifica si el espacio ocupado del buffer alcanza nuestro “límite” establecido, si es así entonces se envía un paquete de tipo “feedback” por la conexión de retorno hacia el Transmisor. Si no, simplemente lo envía.

Si le llega un paquete de tipo “**feedback**”, entonces estamos en el caso que el buffer en la cola intermedia alcanzó nuestro “límite” establecido, y envía un paquete de tipo “feedback” al Transmisor.

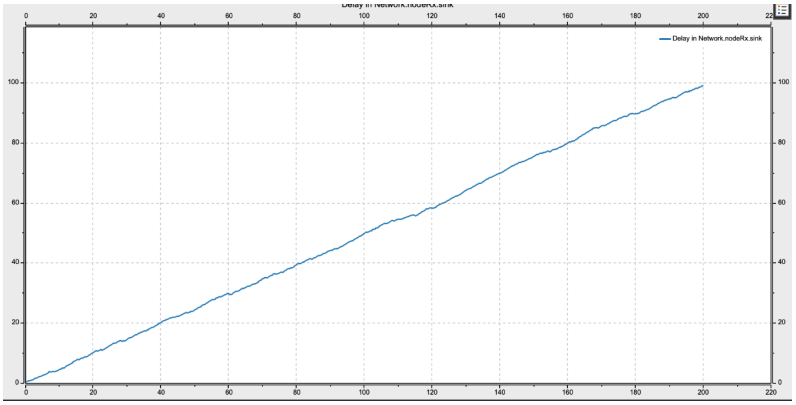
Resultado

Luego de aplicar nuestro diseño para el control de flujo y congestión . Volvemos a realizar el experimento con generación de paquetes en el intervalo que nos generaba problemas en ambos casos (generationInterval = exponential(0.1)), ya que era el caso que nos importaba solucionar, por la pérdida de paquetes que nos producía a causa de la saturación de buffers.

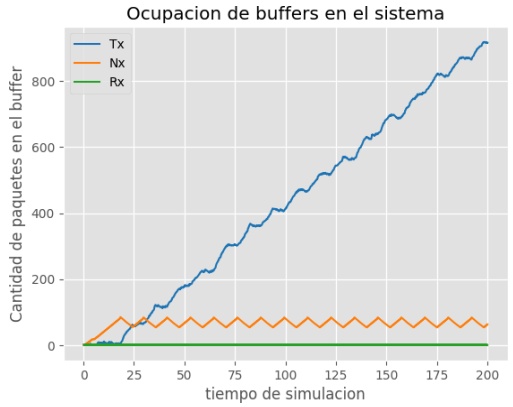
Caso de estudio 1



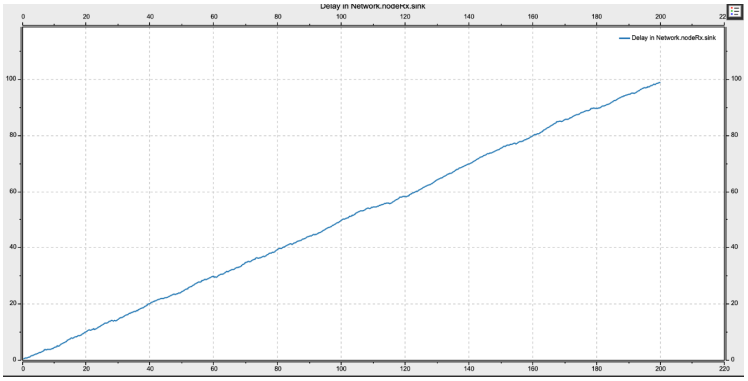
Podemos ver en el gráfico como, luego de aplicar la nuestra estrategia para controlar el flujo, el algoritmo impide que el buffer de la queue de NodeRx se llene (es decir, llegue a 200 paquetes), de manera que evitamos la pérdida de paquetes por la saturación del buffer.



Caso de estudio 2

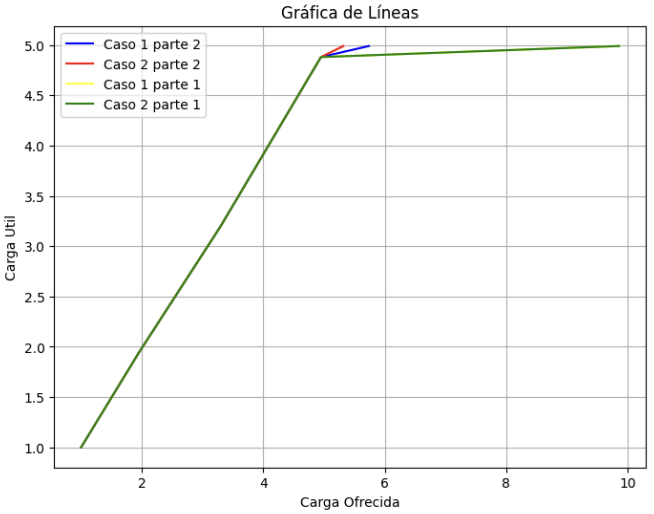


Analizando el caso de estudio 2, podemos ver cómo ocurre algo similar que con el caso de estudio 1, nada más que ahora lo que logramos controlar es el problema de la saturación del buffer de la queue intermedia (observación: el buffer de la queue intermedia tiene tamaño 100), solucionando el problema de congestión, y así evitando la pérdida de paquetes.



Carga Ofrecida vs Carga Útil

A través del siguiente gráfico podremos ver la efectividad de nuestro algoritmo, a través del gráfico de carga útil vs carga ofrecida, que compara la cantidad de paquetes enviados por segundo (carga ofrecida), con la cantidad de paquetes recibidos por segundo (carga útil). Podemos ver en el gráfico como se muestra lineal durante el comienzo, ya que tenemos un intervalo de generación de paquetes alto, lo cual hace que el consumidor reciba la misma cantidad de paquetes

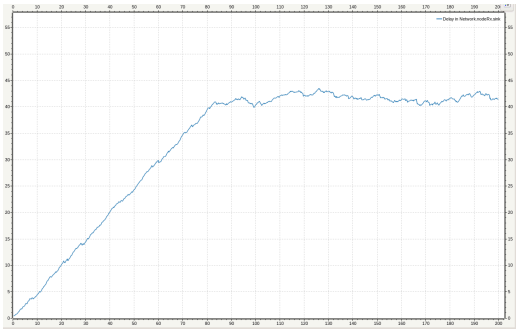


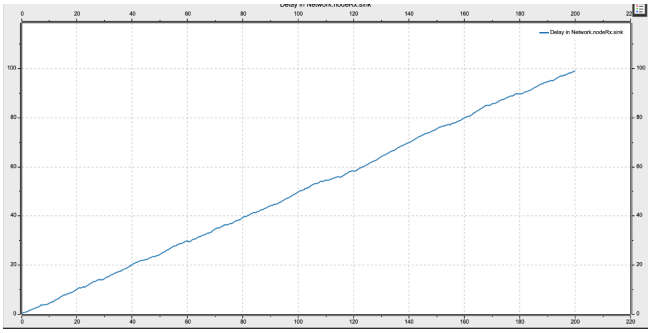
de las que el generador envía, y de esta manera no perdamos paquetes. Luego podemos ver como la gráfica empieza a cambiar a partir de cuando envía 5 paquetes por segundos aproximadamente. Comparando los distintos casos luego de la implementación de nuestro algoritmo, se reduce significativamente la diferencia entre los paquetes enviados y recibidos, tanto en el caso 1, como en el caso 2, lo que nos muestra que nuestro algoritmo está funcionando de manera adecuada ya que podemos ver como la carga útil toma valores más cercanos a los de la carga ofrecida, intentando de esta manera mantener un gráfico más lineal.

Discusión

Podemos concluir que nuestro algoritmo implementado soluciona el problema de flujo y el problema de congestión para la pérdida de paquetes en la red. Cabe aclarar que como la idea principal de éste es limitar (ralentizar) el tiempo en el que envía los paquetes el transmisor, entonces se produce un notable aumento en el retardo de la entrega de los paquetes.

Lo que se puede ver en los gráficos de delay de la red inicial (izquierda) y de la red modificada aplicando nuestro algoritmo (derecha).





En resumen, aunque logramos prevenir la pérdida de paquetes con nuestro algoritmo, reconocemos que esto conlleva un aumento en el tiempo que tardan los paquetes en ser entregados.

Referencias

Omnet++
Script de instalación de OMNeT++
Carga Ofrecida vs Carga Útil