Proyectos CT y CR

Presentación de los proyectos realizados para el laboratorio 3 y 4

Grupo 31:

- Álvarez Miranda, Facundo (43342875)
- Bratti, Juan (44274011)
- Herrador, Emanuel Nicolás (44898601)

Laboratorio 3

Análisis del Flujo y Congestión Simulaciones en Omnet++

En este laboratorio nos adentramos en la simulación de modelos de red en Omnet++, y el análisis de tráfico de las respectivas redes bajo condiciones particulares.

Nos centramos en dos tareas principales: Análisis y Diseño, en las que analizamos la respuesta de una red ante la introducción de tasas de transmisión y demás parámetros, y diseñamos un control de flujo y congestión para administrar los datos transportados por la red de forma adecuada.

Introducción

En una red hay varios factores que pueden alterar a su desempeño de la misma a la hora de enviar y recibir datos. Entre algunos de esos factores, encontraremos los relacionados al flujo y a la congestión de la red.

Control de Flujo

Control del tráfico de datos entre un emisor y un receptor, para asegurar que el receptor no esté recibiendo una cantidad abrumadora de datos por parte del emisor

Control de Congestión

Control que se utiliza para regular la cantidad de datos que son enviados por la red en sí, debido a que puede producirse un envío masivo para una capacidad de transferencia de la red limitada. No se centra en la cantidad que puede recibir un receptor, si no en la cantidad que puede aguantar la red cuando transfiera esos datos.

Objetivo

En particular, en este proyecto, nos centraremos en esos dos conceptos y simularemos primero, el comportamiento de una red cuando distintos parámetros vinculados al tráfico de la red se manipulan, y luego veremos cómo cambia su comportamiento cuando se emplea un control de flujo y de congestión particular.

Análisis de Casos

Para el análisis de caso lo que hicimos fue guiarnos por dos casos/contextos distintos, midiendo en ambos el impacto que tienen sus características en la transmisión de los paquetes y la memoria que tienen los buffers de los respectivos nodos.

Así, nuestro análisis consistió en dos casos/contextos:

Primer Caso

Características

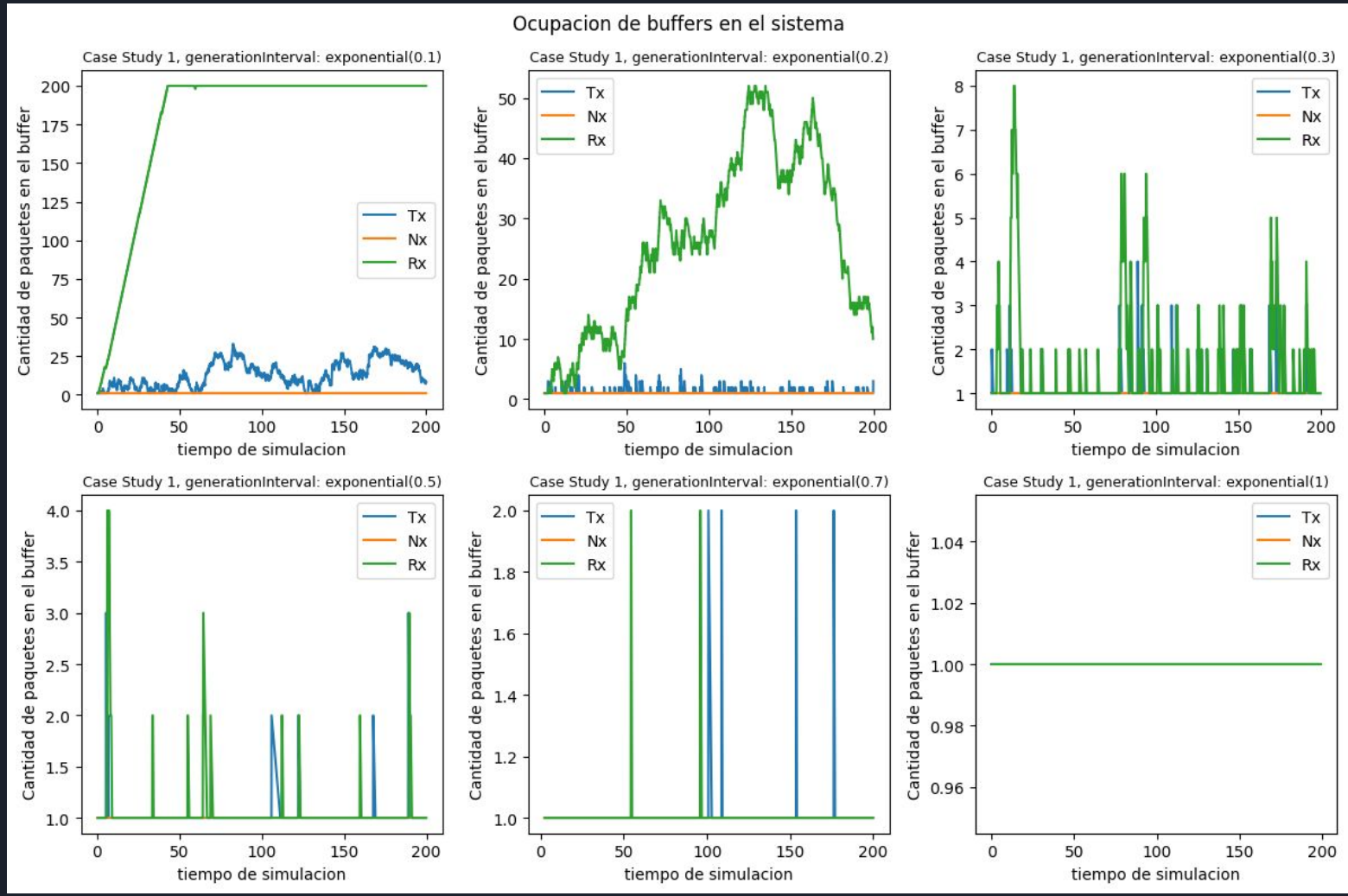
Las características de la red en este casos son las siguientes:

1. **NodeTx a Queue**: tasa de transferencia de 1 Mbps y demora de transmisión de 100 us (microsegundo).
2. **Queue a NodeRx**: tasa de transferencia de 1 Mbps y demora de transmisión de 100 us (microsegundo).
3. **Queue a Sink**: tasa de transferencia de 0.5 Mbps

Con estos cambios, medimos su impacto en los buffers de los nodos, y en la transmisión de paquetes

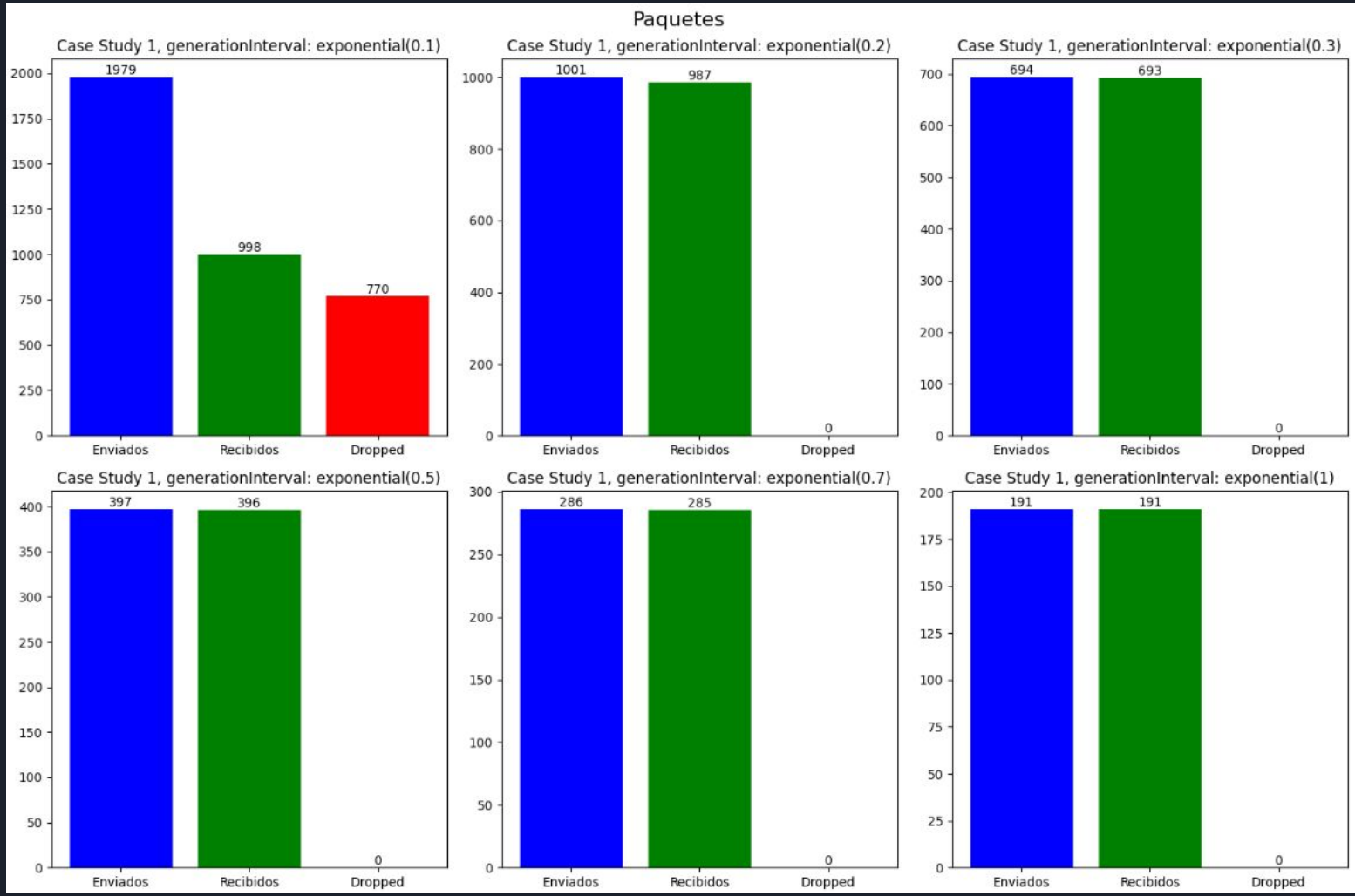
Primer Caso

Impacto en los buffers



Primer Caso

Impacto en transmisión paquetes



Segundo Caso

Características

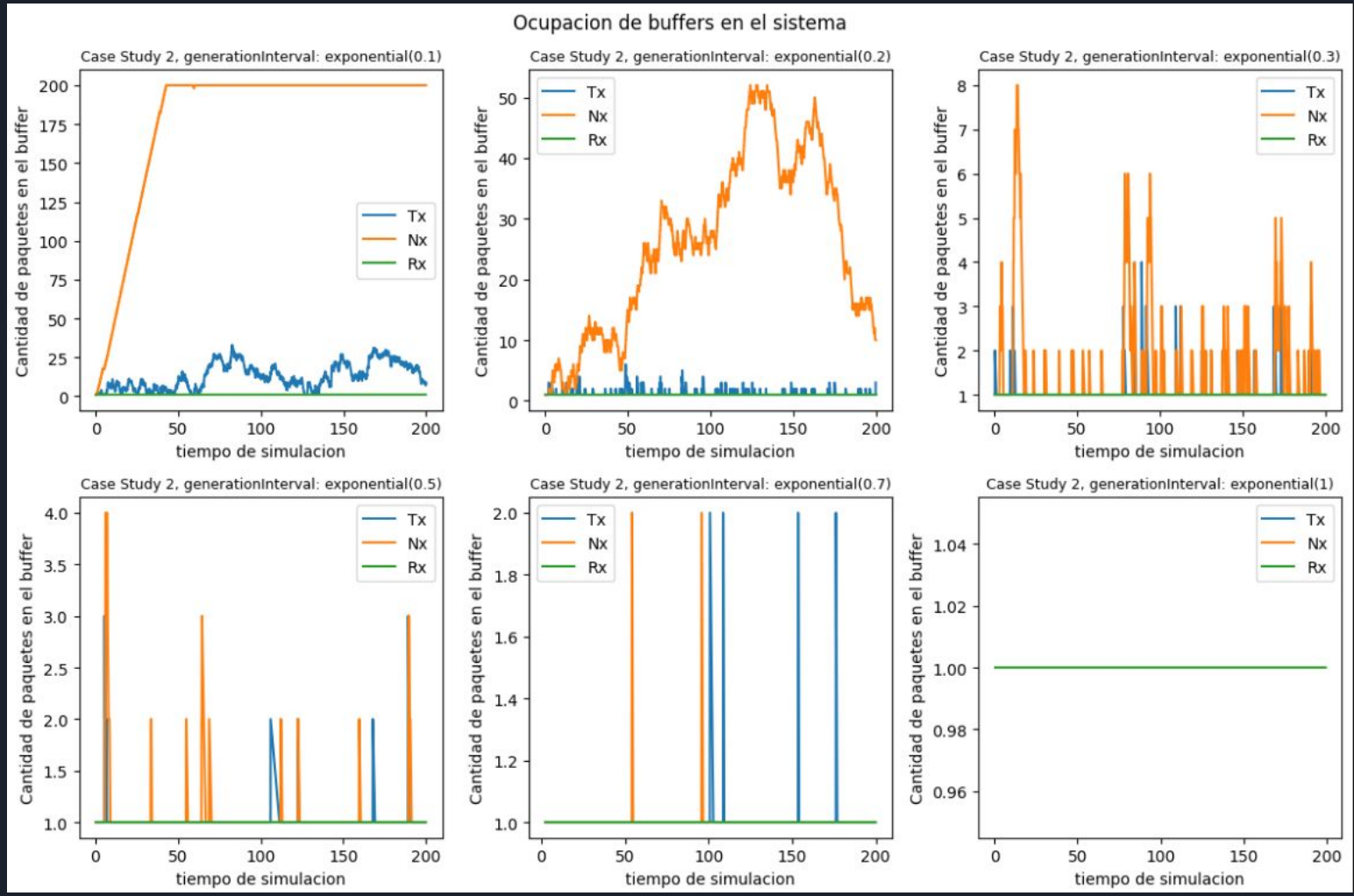
Las características de la red en este casos son las siguientes:

1. **NodeTx a Queue**: tasa de transferencia de 1 Mbps y demora de transmisión de 100 us.
2. **Queue a NodeRx**: tasa de transferencia de 0.5 Mbps y demora de transmisión de 100 us.
3. **Queue a Sink**: tasa de transferencia de 1 Mbps

Con estos cambios, medimos su impacto en los buffers de los nodos, y en la transmisión de paquetes

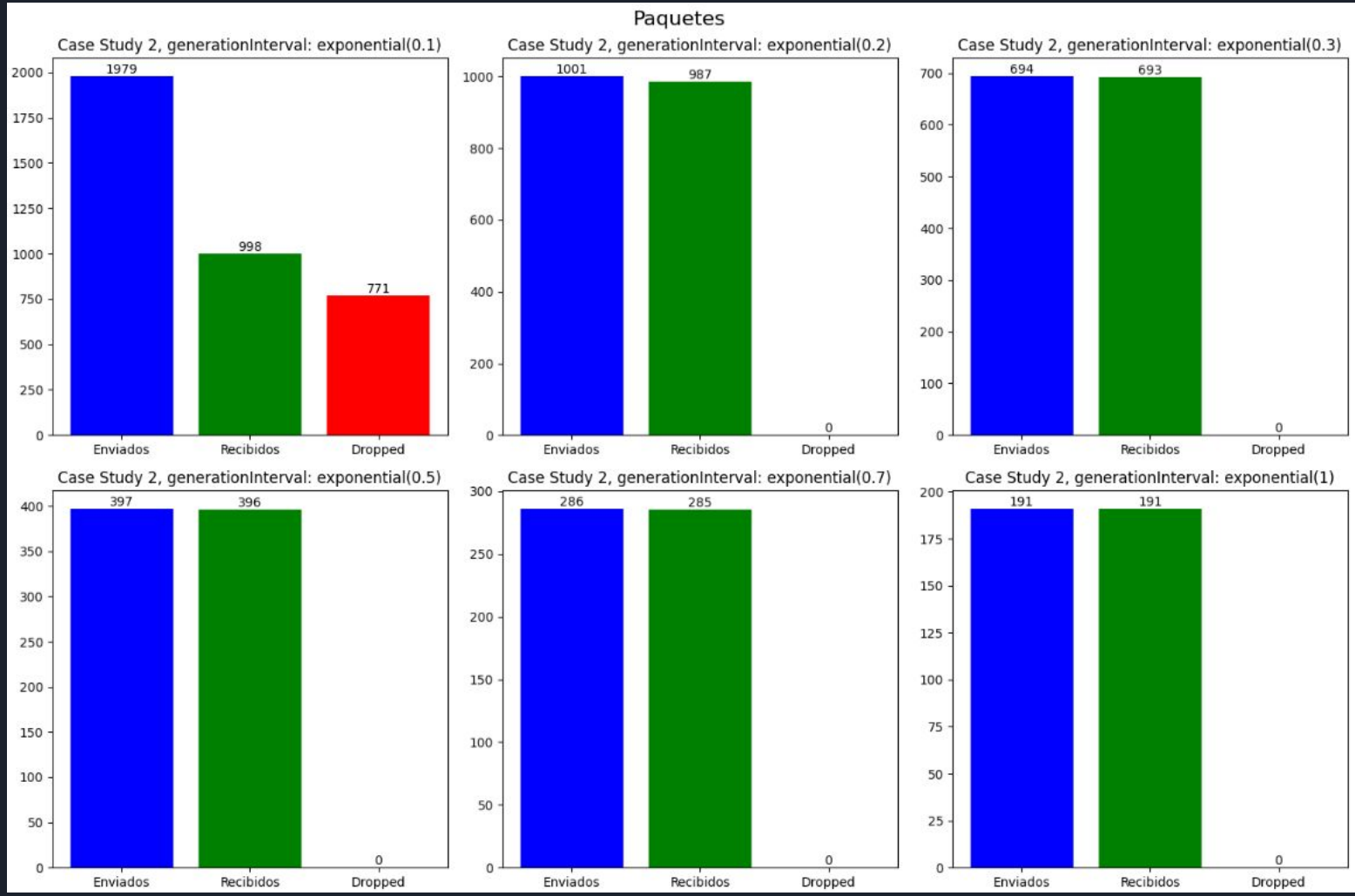
Segundo Caso

Impacto en los buffers



Segundo Caso

Impacto en transmisión paquetes



Diferencias entre el caso 1 y 2

En el caso 1, podemos observar que el control que se realiza es de flujo:

- Esto se debe a que la tasa de transf. hacia Sink es menor a las demás tasas. Es decir, la red es una red veloz que tiene un resumidero con “poca capacidad” (poca capacidad porque se transfieren poco paquetes debido a la disminución de la tasa de transferencia de la red en ese punto).

En el caso 2, podemos observar que el control que se realiza es de congestión:

- Aquí el “problema” no es puntualmente que la cantidad de paquetes que llegan a destino (Sink) es el factor limitante, si no que la capacidad interna de la red se ve afectada por la baja tasa de transferencia de Queue a NodeRx, resultando en una congestión general de la red en sí.

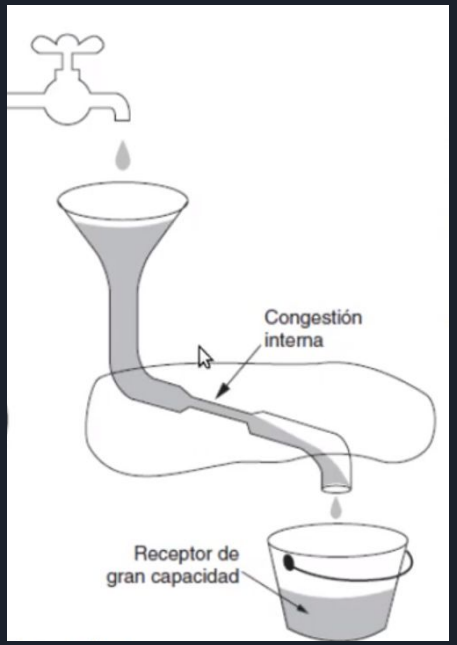
Fuentes limitantes en cada caso

Caso 1:

- La cantidad de paquetes que llegan a Sink.

Caso 2:

- La cantidad de paquetes que pueden ser transferidos a través de la red.

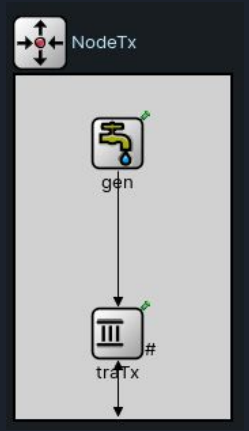


Tarea de Diseño

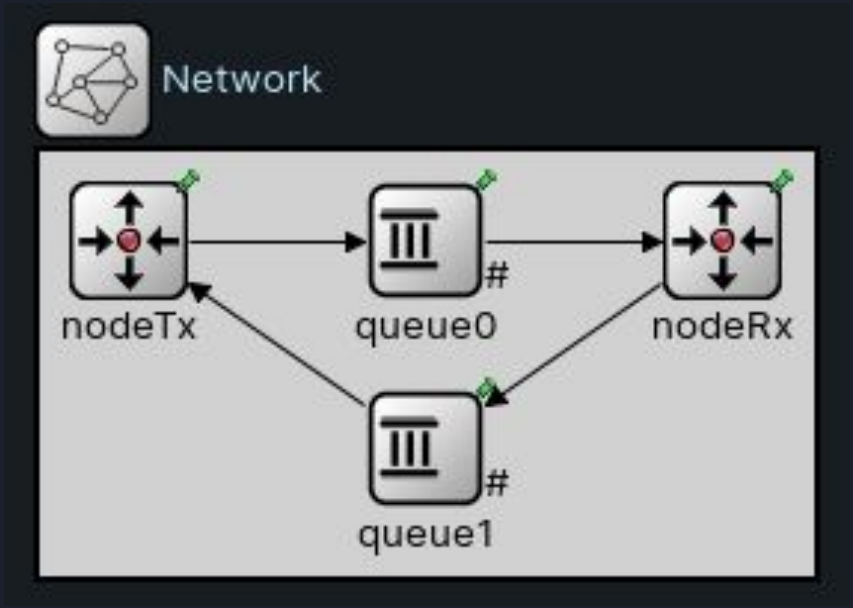
En la segunda parte de este proyecto, el objetivo que se tiene es la re-estructuración de la red para la implementación de un algoritmo de control de flujo y congestión que nos permita poder comparar mediante un análisis los resultados de *intentar resolver los problemas vs no hacer nada*.

Estructura de la Red

Buffers de input y output de los módulos de generación (**NodeTx**) y de recepción (**NodeRx**) llamados **TransportTx** y **TransportRx**



Nos permite poder crear en la red un canal de retorno desde el nodo de recepción al nodo de generación, brindando la posibilidad de que el receptor lo use para poder enviar mensajes de control al emisor



Nuevo tipo de mensaje

Nuevo tipo de paquete, el cual es de control y permite almacenar datos que el receptor llena para que sean leídos e interpretados por el emisor para realizar alguna acción.

Lo llamamos **ControlPacket** y considera:

- *totalBuffer*: tamaño total del buffer del receptor
- *remainingBuffer*: capacidad restante del buffer del receptor
- *timeElapsedToReceivePacket*: cantidad de tiempo que transcurrió entre el envío y la recepción del correspondiente paquete de datos

Idea de elementos de transporte

TransportTx

1. Recepción de paquetes de datos desde Gen y su envío a *Queue0*
2. Recepción de paquetes de control desde *Queue1* y gestión de acciones que modifiquen los tiempos de *(1.)* para evitar problemas de flujo y congestión (y, por ende, pérdida de paquetes)

TransportRx

1. Recepción de paquetes de datos desde *Queue0* y su envío a *Sink*
2. Creación de paquetes de control (uno por cada paquete de dato agregado al buffer) y su envío a *Queue1* con destino a *TransportTx*

Idea del algoritmo implementado

- Por cada paquete recibido en *TransportRx* y aceptado (es decir, que no se elimina), se crea su correspondiente mensaje de control
- En *TransportTx* se recibe la información de control y se considera:
  - El tiempo mínimo que tardó un mensaje en llegar al receptor (variable *minSend*)
    - Esto nos permite darnos una idea de la congestión de la red y actuar en base a eso
    - Se "resetea" cada 50 paquetes de control recibidos para mantener actualizada la información (porque capaz no está congestionada la red pero tiene más carga que antes)

- Vamos a tener una variable controlFactor que es la responsable del manejo del control de flujo y congestión (valor entre 0 y 1) generando el delay de envío de paquetes:

```
void TransportTx::scheduleSendPacketWithDelay(simtime_t delay) {
    scheduleAt(simTime() + delay + delay*controlFactor, endServiceEvent);
}
```

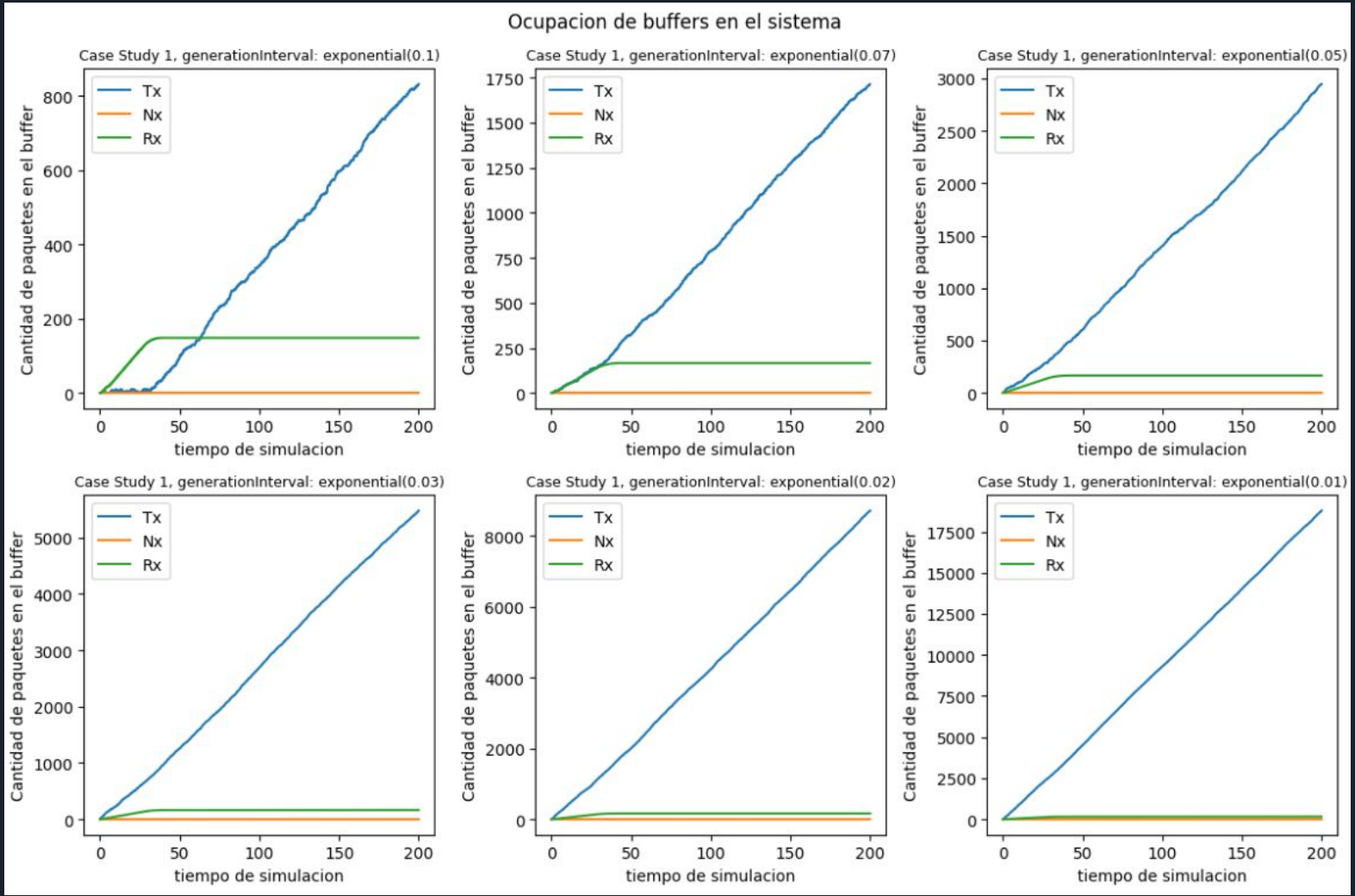
- Respecto al **flujo** se considera
  - Si remainingBuffer <= 0.30*totalBuffer, entonces se aumenta controlFactor en 1e-2 (receptor se queda sin espacio)
  - Si remainingBuffer >= 0.50*totalBuffer, entonces se disminuye controlFactor en 1e-2 (receptor tiene espacio)
- Respecto a la **congestión**
  - Si timeElapsedToReceivePacket >= 2*minSend, entonces se aumenta controlFactor en 1e-2 (hay más congestión que la registrada)
  - Si timeElapsedToReceivePacket <= minSend, ntonces se disminuye controlFactor en 1e-2 para acelerar la salida de paquetes (tiene menos carga que la registrada)

Resultados

Para poder saber cómo fue el comportamiento de la red con este nuevo enrutamiento, volvimos a analizar los casos anteriores con la diferencia de que el intervalo de generación de los paquetes cada vez disminuye más.

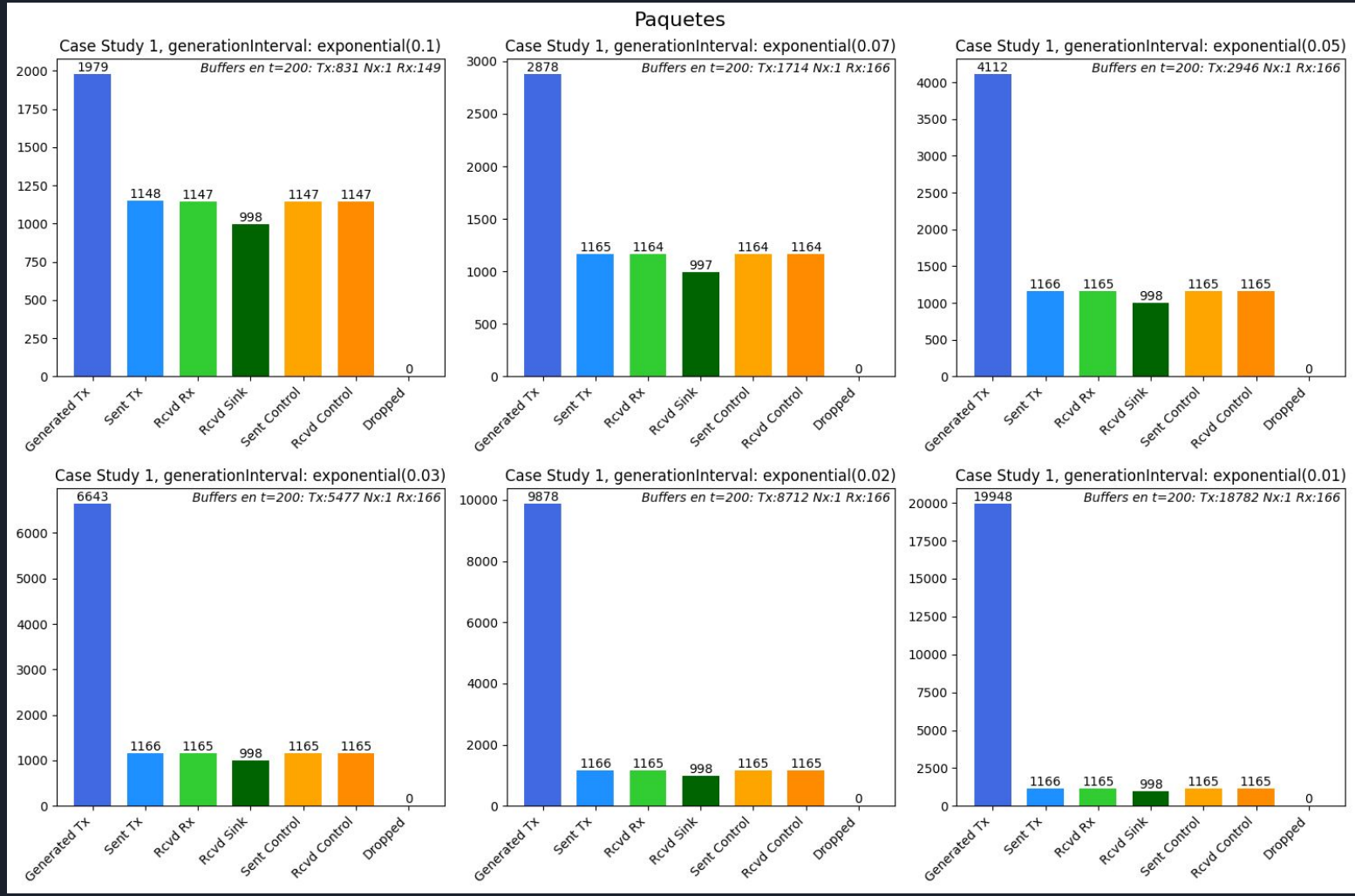
Primer Caso

Impacto en los buffers



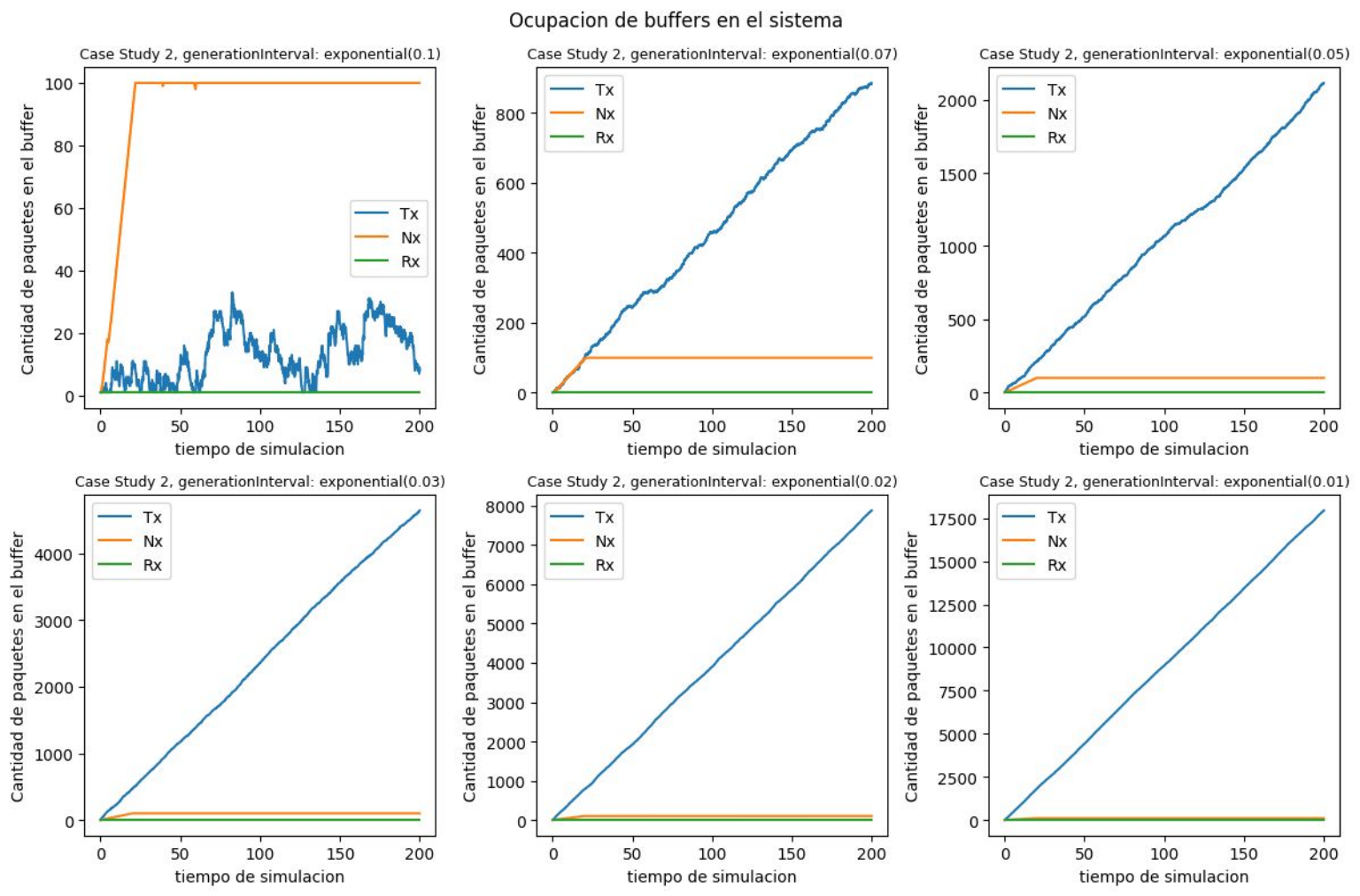
Primer Caso

Impacto en transmisión paquetes



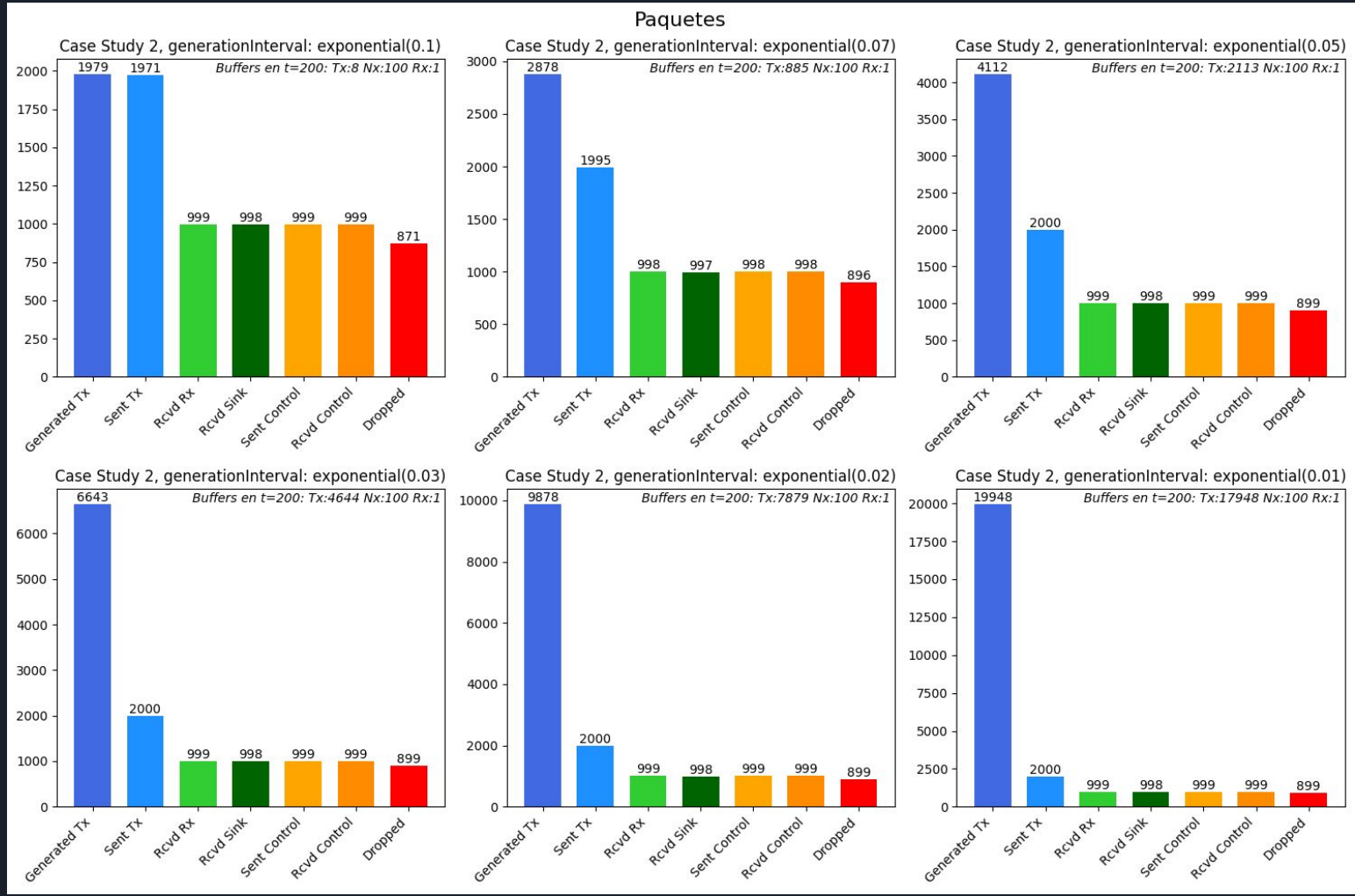
Segundo Caso

Impacto en los buffers



Segundo Caso

Impacto en transmisión paquetes



Comportamiento del algoritmo ante control de flujo y congestión

El algoritmo se comporta correctamente ante el control de flujo y congestión. Vimos en los gráficos que cuando se reciben muchos paquetes, se limita al emisor para evitar la pérdida de paquetes mediante los controles constantes entre TraTx y TraTx.

Para la congestión también, a medida que muchos paquetes se envían, la misma se detecta y se ralentiza a TraTx.

¿Funciona para el caso de estudio 1 y 2 por igual? ¿Por qué?

No. El funcionamiento no es igual en ambos casos.

En el caso 1 se presta atención a la cantidad de paquetes en el buffer de TraRx en ese momento, y se realiza un análisis del restante que le queda de tamaño al buffer y el 30 por ciento de la capacidad total para decidir si se debe ralentizar la emisión de paquetes por parte de TraTx.

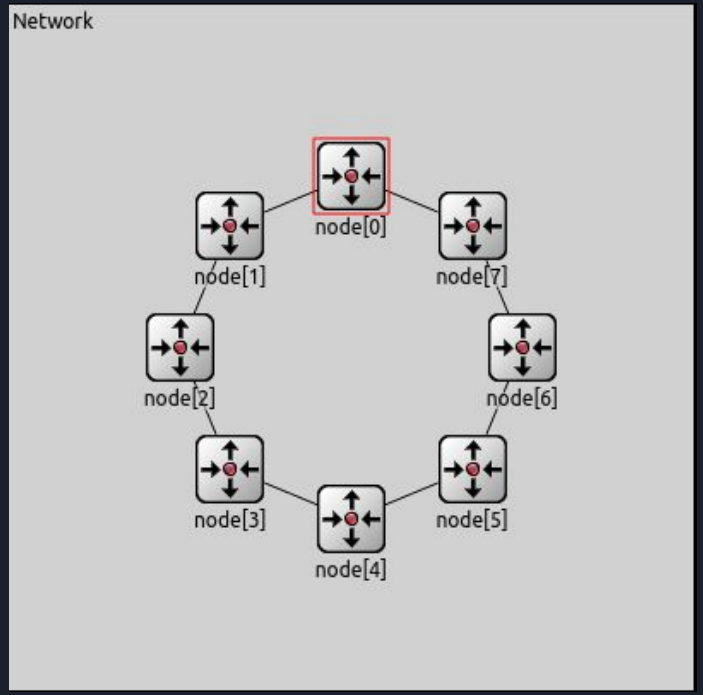
Por otra parte, en el caso 2, el algoritmo trabaja por otro lado: “analiza” el tiempo que tardan los paquetes en llegar a TraRx, no la cantidad.

Laboratorio 4

Análisis e Implementación de Algoritmos de Enrutamiento de la Capa de Red en Networks simulados en Omnet++

En este laboratorio estudiamos la topología anillo y elaboramos un enrutamiento eficiente para la misma.

Realizamos un análisis crudo del comportamiento de nuestra red en distintos casos particulares. Luego, mejoramos la estrategia de enrutamiento para evitar problemas de congestión y obtener una mayor eficiencia.



Introducción

En una red de topología anillo, en donde se tienen varios nodos como actores responsables del envío/recibo de paquetes, es importante aprovechar las múltiples salidas y entradas para la comunicación de mensajes.

Nuestro objetivo radica en eso, poder ingeniar un enrutamiento de paquetes de tal forma que se aprovechen al máximo las múltiples conexiones disponibles.

Objetivo

Previo al diseño del algoritmo de enrutamiento, analizamos el comportamiento de la red base, es decir, con el enrutamiento original dado por la cátedra.

Luego, implementamos el algoritmo nuevo y desarrollamos la tarea estrella.

Análisis de Casos

Para el análisis de caso lo que hicimos fue también guiarnos por dos casos/contextos distintos, midiendo 3 detalles importantes:

- Una medida para la demora de entrega de paquetes
- Una medida para la cantidad de saltos hechos por cada paquete
- Una medida para la utilización de los recursos de la red (buffers y enlaces)

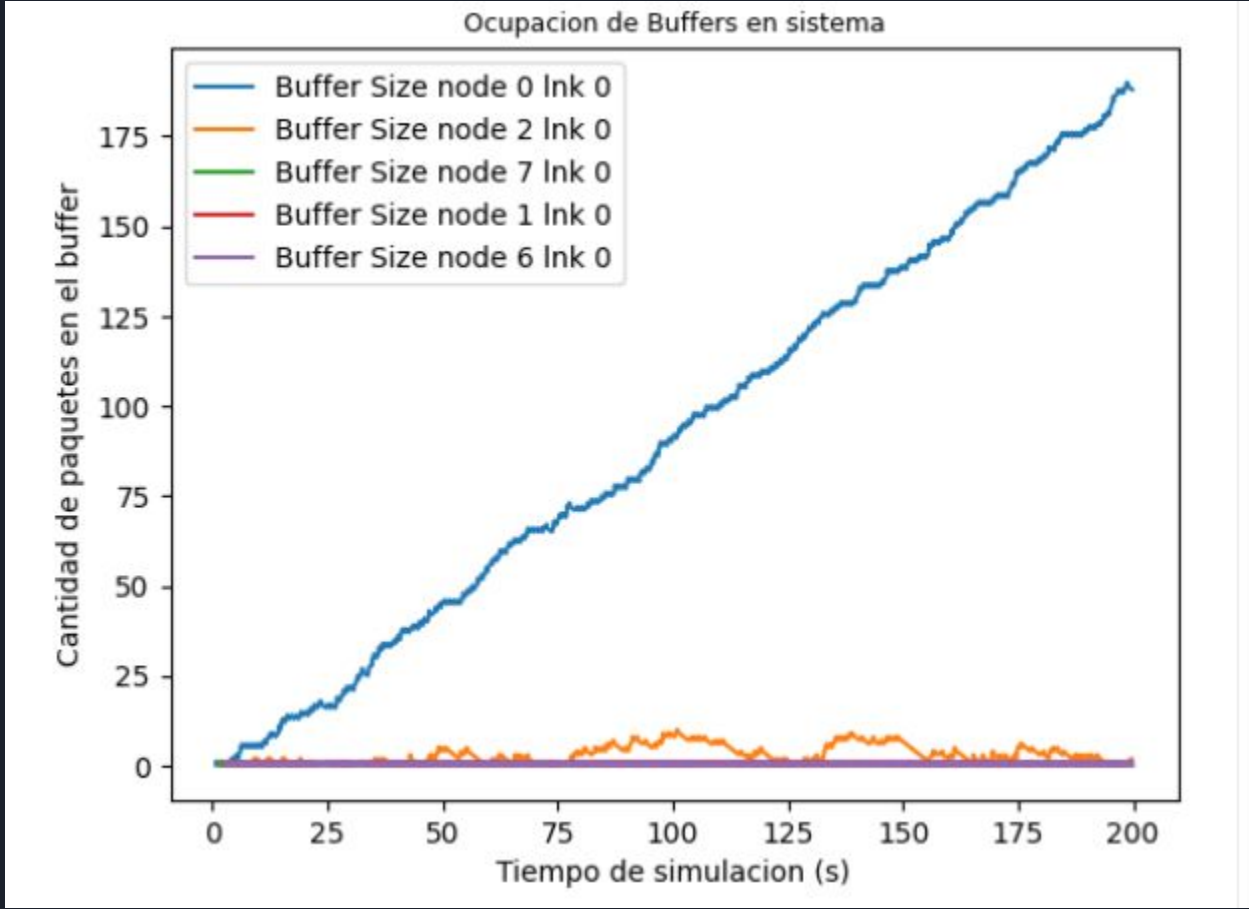
Entonces, veamos los dos casos propuestos.

Primer Caso

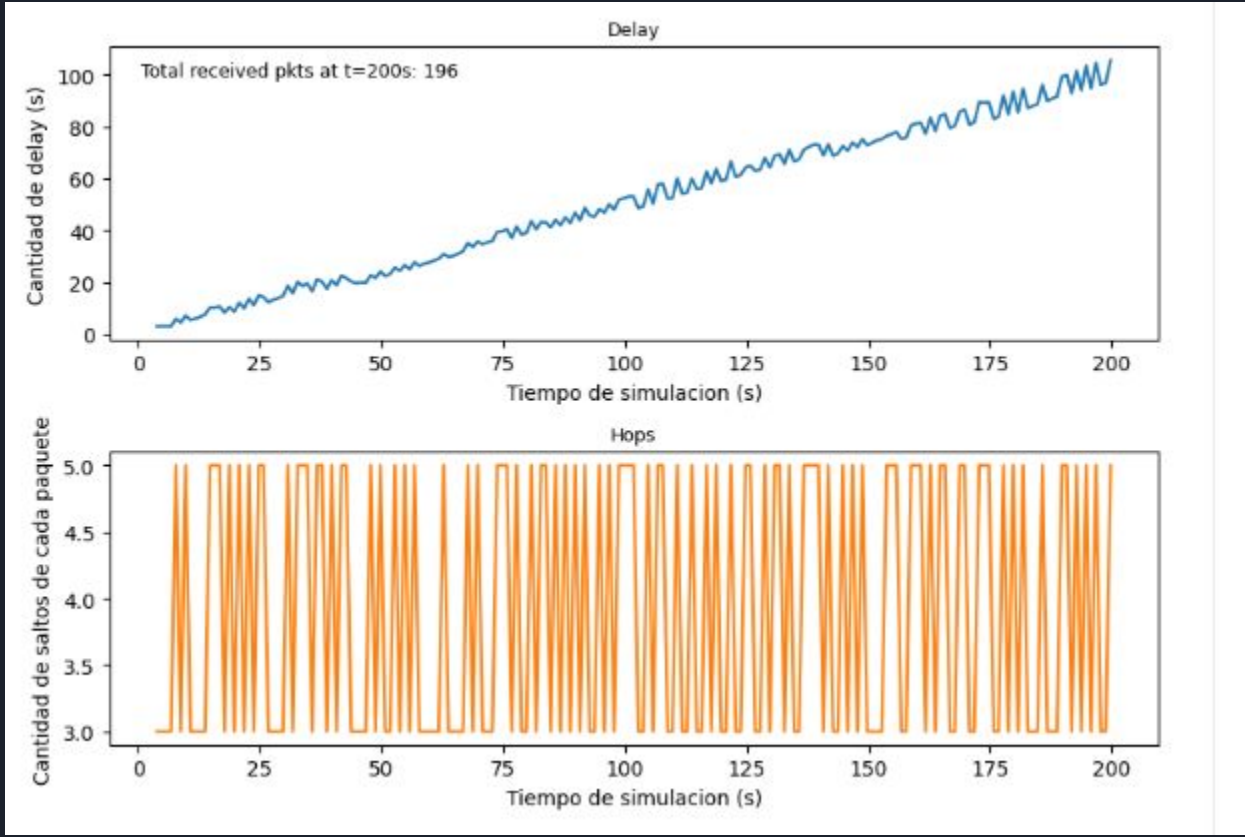
En este caso corrimos el modelo sin cambios en su configuración, es decir, los nodos 0 y 2 generan paquetes y lo transmiten al nodo 5.

Veremos que la utilización de los recursos de la red es muy ineficiente. Se produce congestión y tenemos la mitad de la red sin utilización.

Primer Caso: Impacto en los buffers



Primer Caso: Saltos y delays



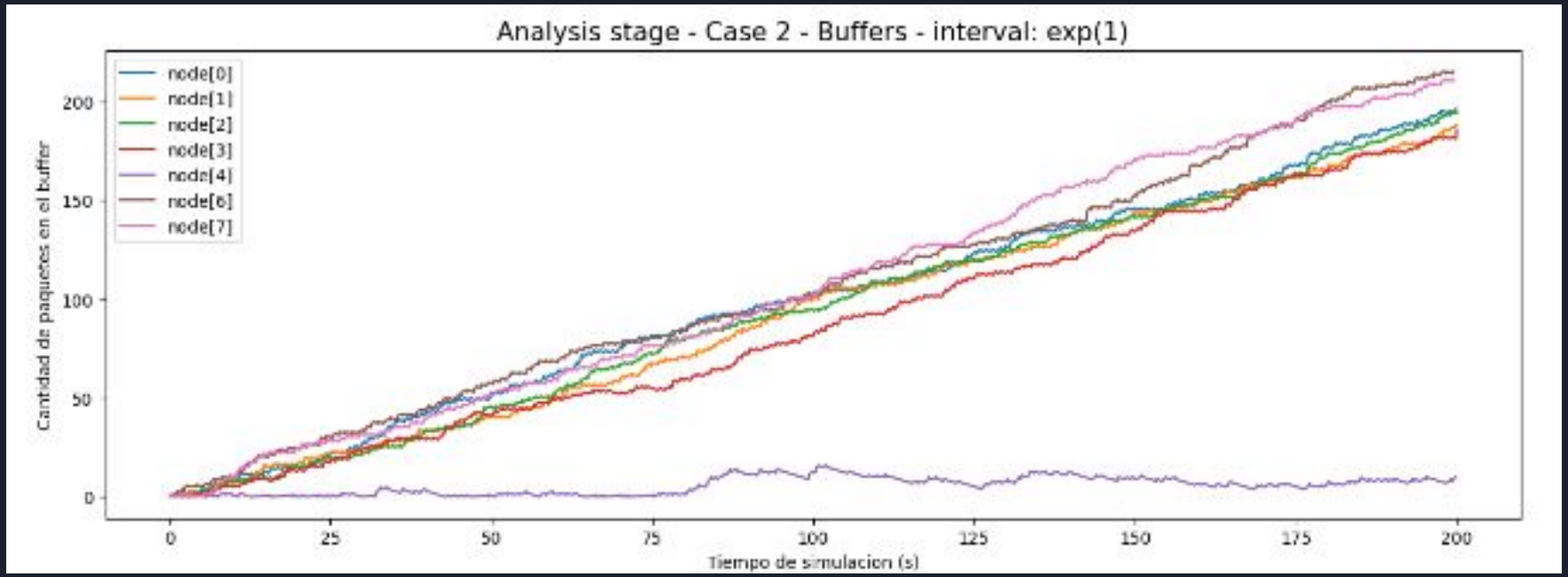
Segundo Caso

En este caso, todos los nodos generan paquetes hacia el nodo 5. PacketByteSize e InterArrivalTime son idénticos entre los nodos. Debemos además, buscar un tiempo de interArrivalTime en donde la red se estabilice.

Tendremos los gráficos para distintos interArrivalTimes.

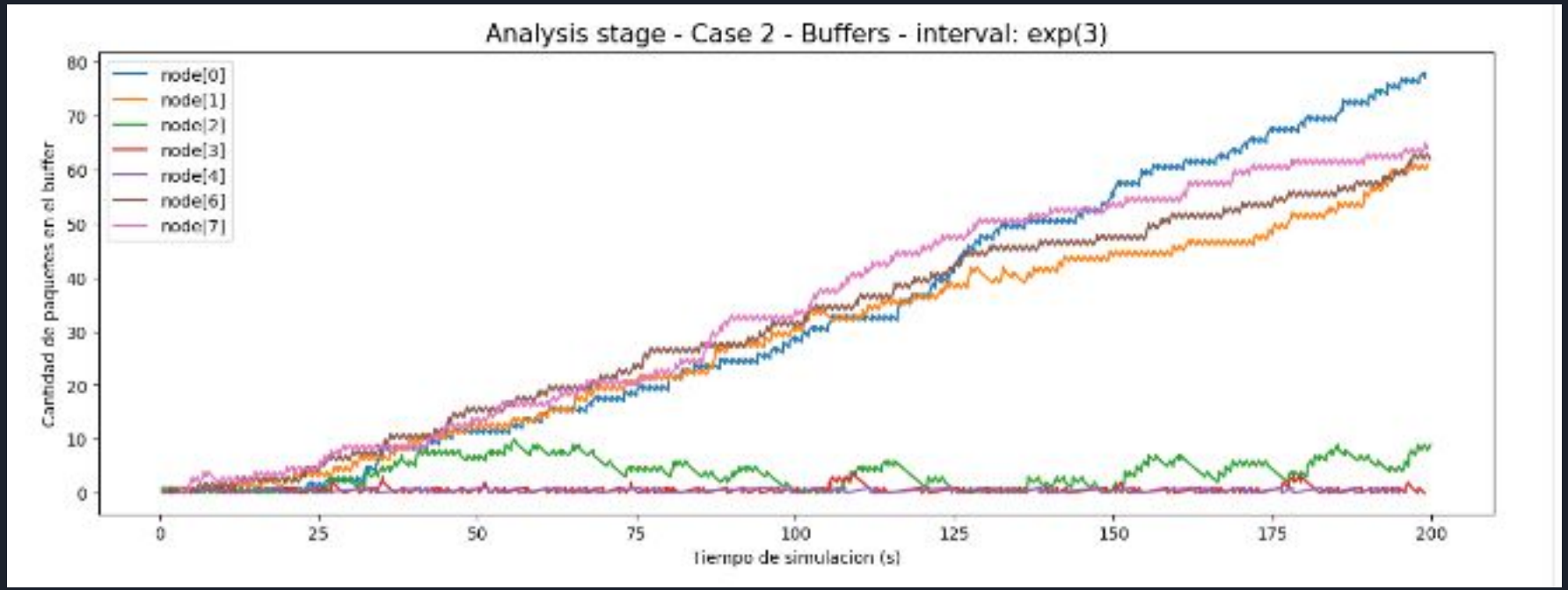
Segundo Caso: Impacto en los buffers

interArrivalTime = exp(1)



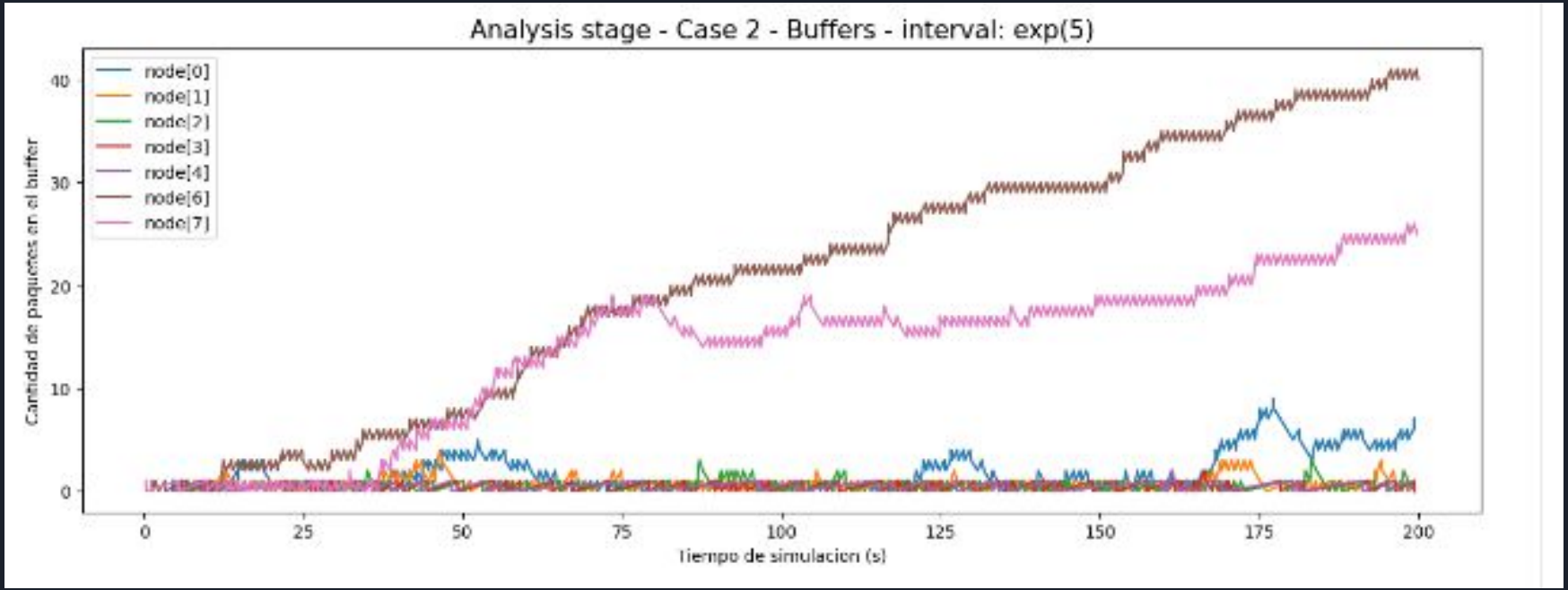
Segundo Caso: Impacto en los buffers

interArrivalTime = exp(3)

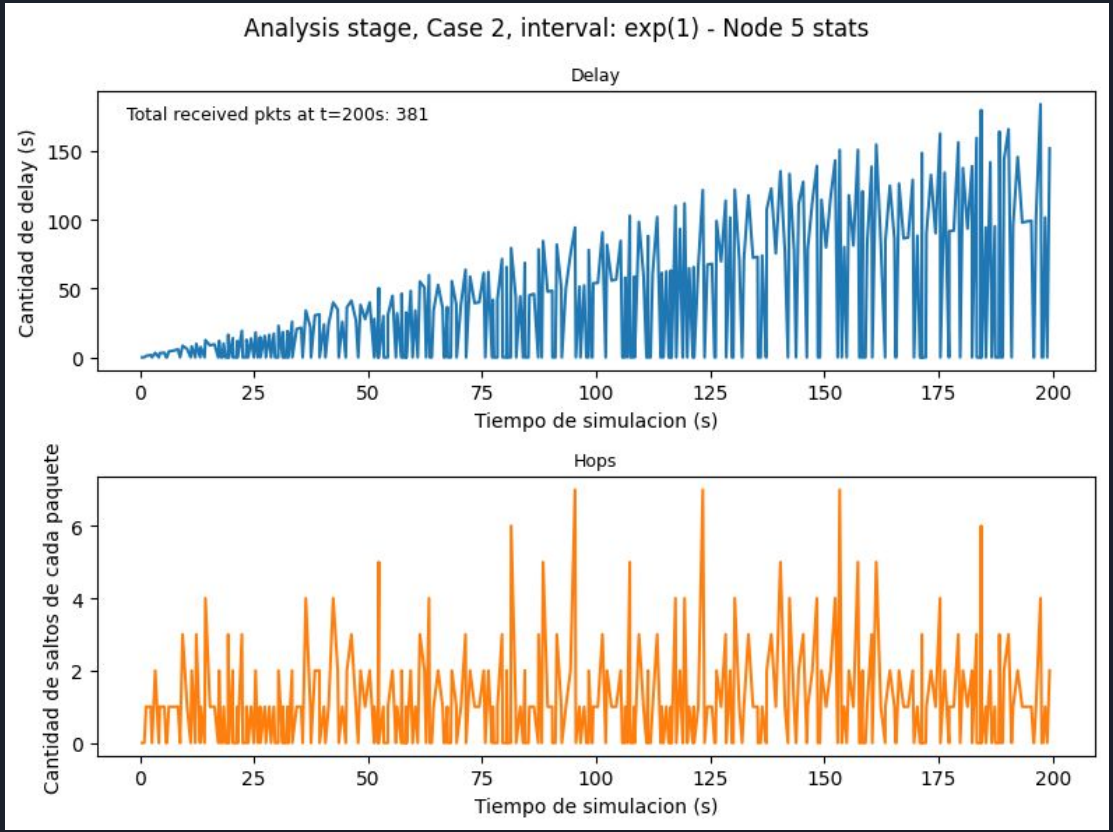


Segundo Caso: Impacto en los buffers

interArrivalTime = exp(5)

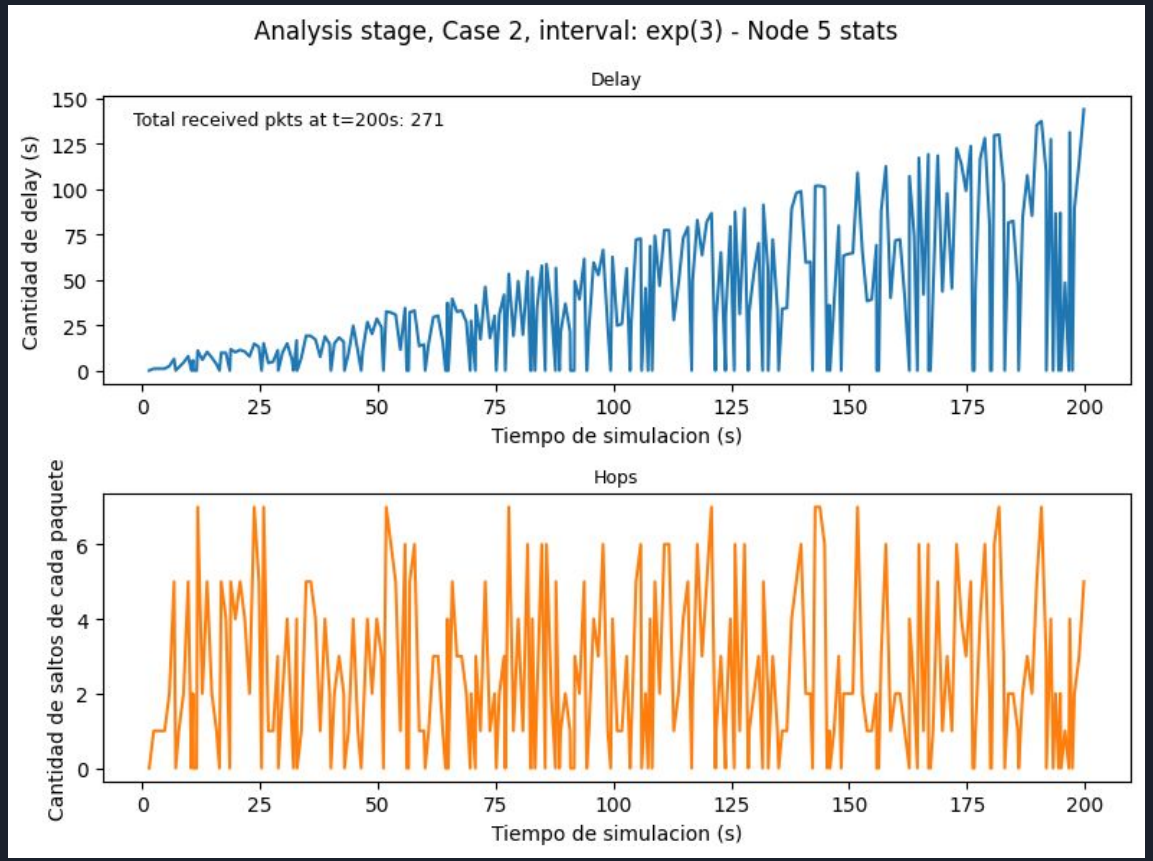


Segundo Caso: Saltos y delays



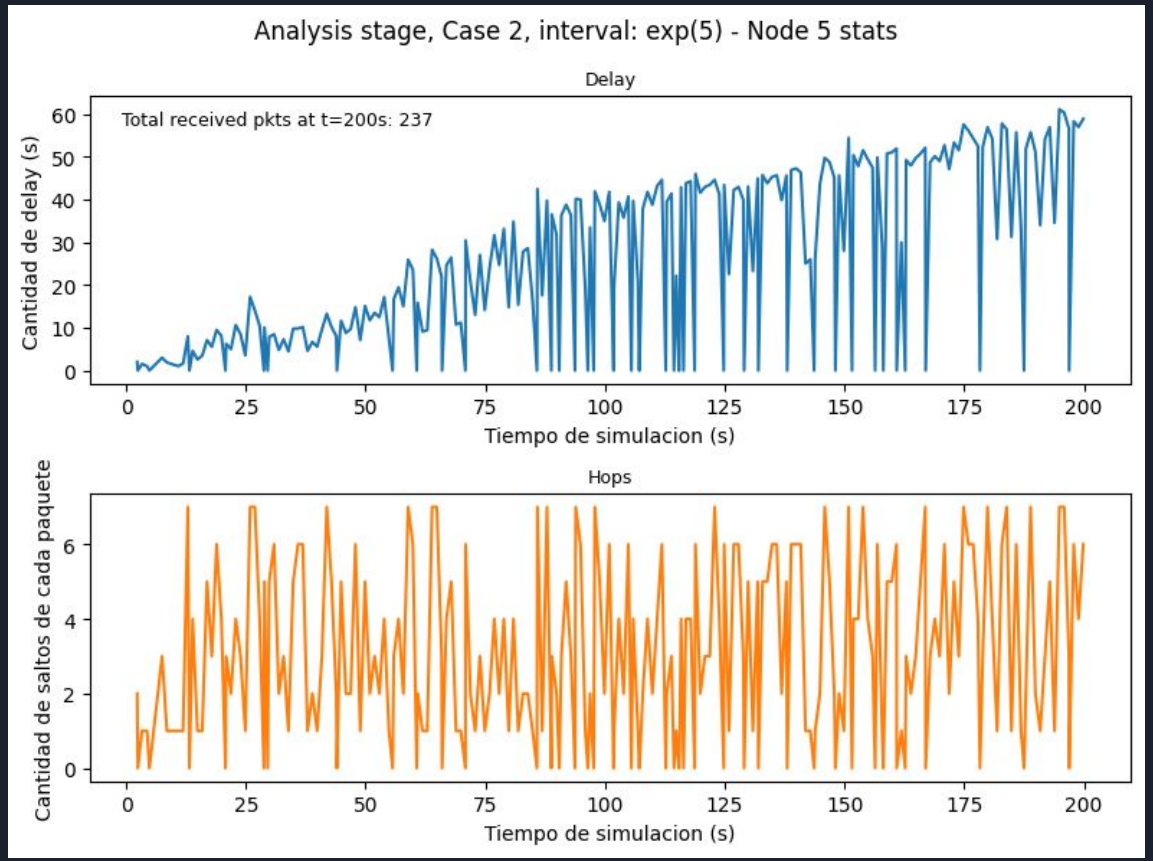
interArrivalTime = exp(1)

Segundo Caso: Saltos y delays



interArrivalTime = exp(3)

Segundo Caso: Saltos y delays



interArrivalTime = exp(5)

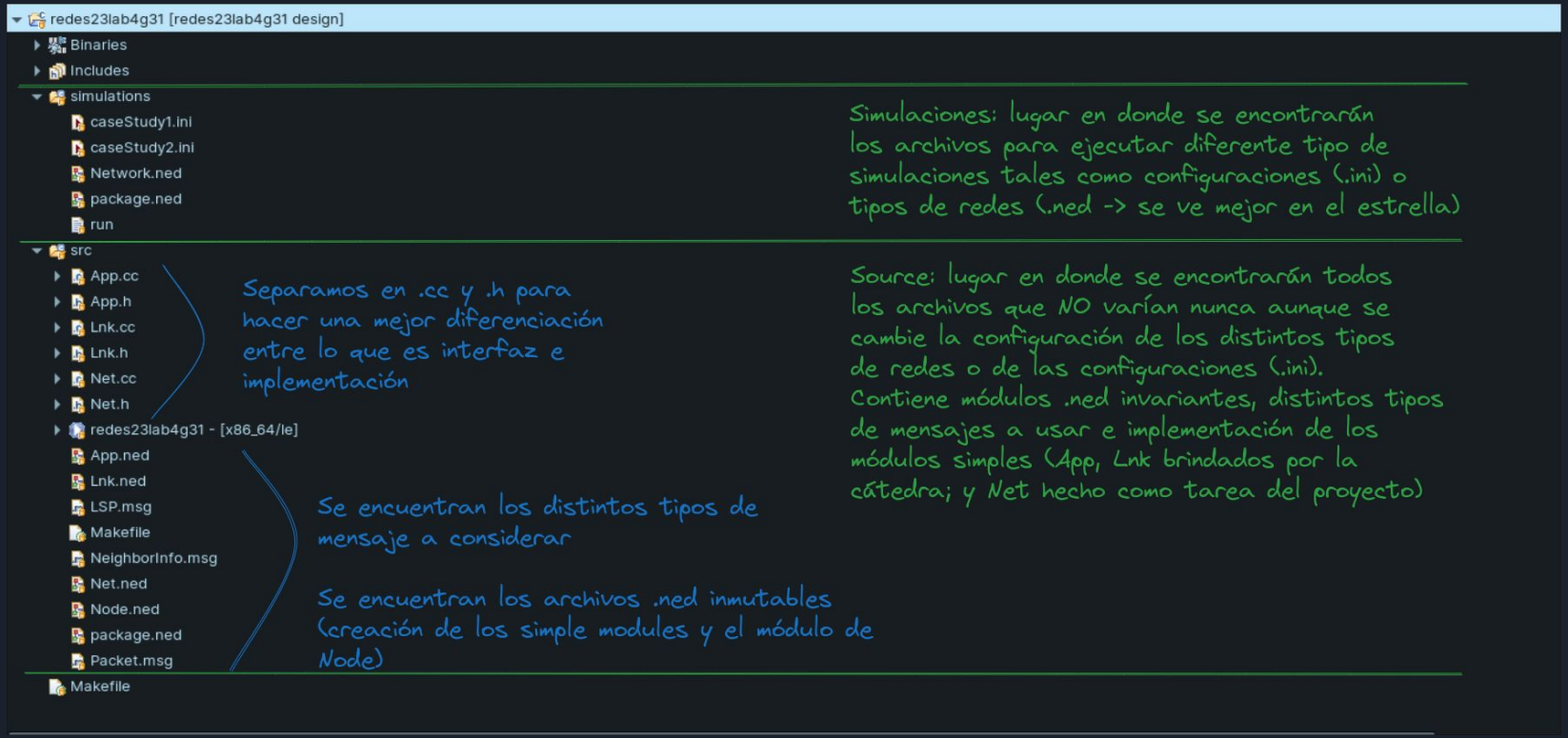
Tarea de Diseño

Para la misma topología de anillo, se implementa un algoritmo que minimice la *cantidad de saltos* realizados por los paquetes en su envío a un destino.

La idea de la implementación fue realizada con el objetivo de que sea una base de la cual poder desarrollar el punto estrella de forma más fácil (es decir, sin tanto trabajo teniéndolo que re-hacer completo).

La principal suposición que se tiene en cuenta para el desarrollo es que **en ningún momento se produce algún cambio en la topología de la red.**

Estructura a considerar



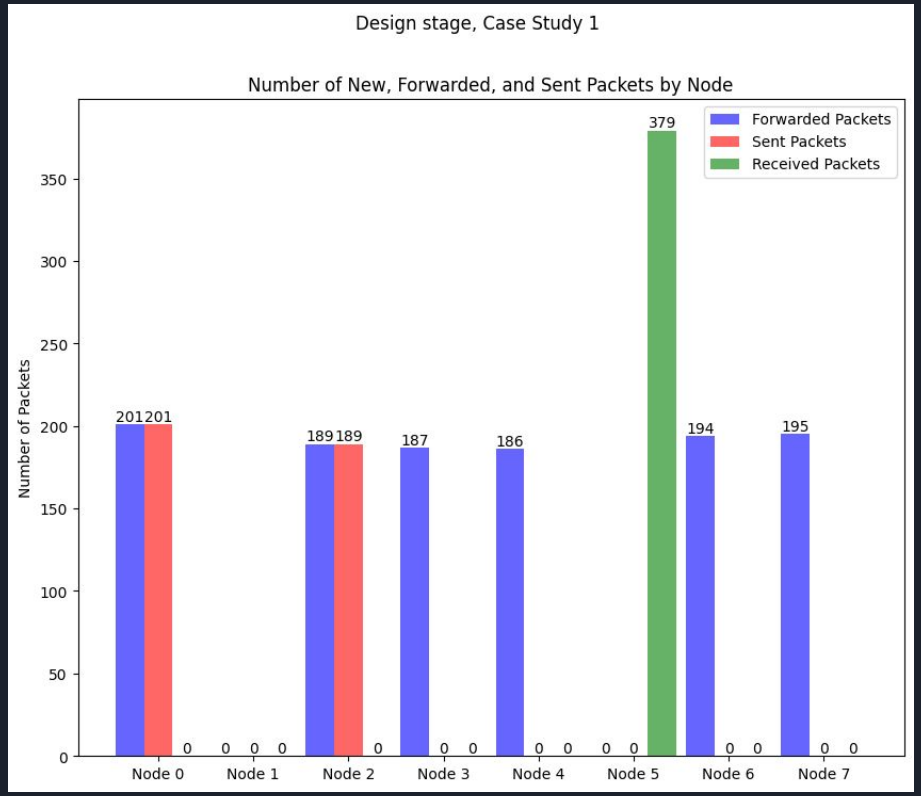
Puntos a tener en cuenta

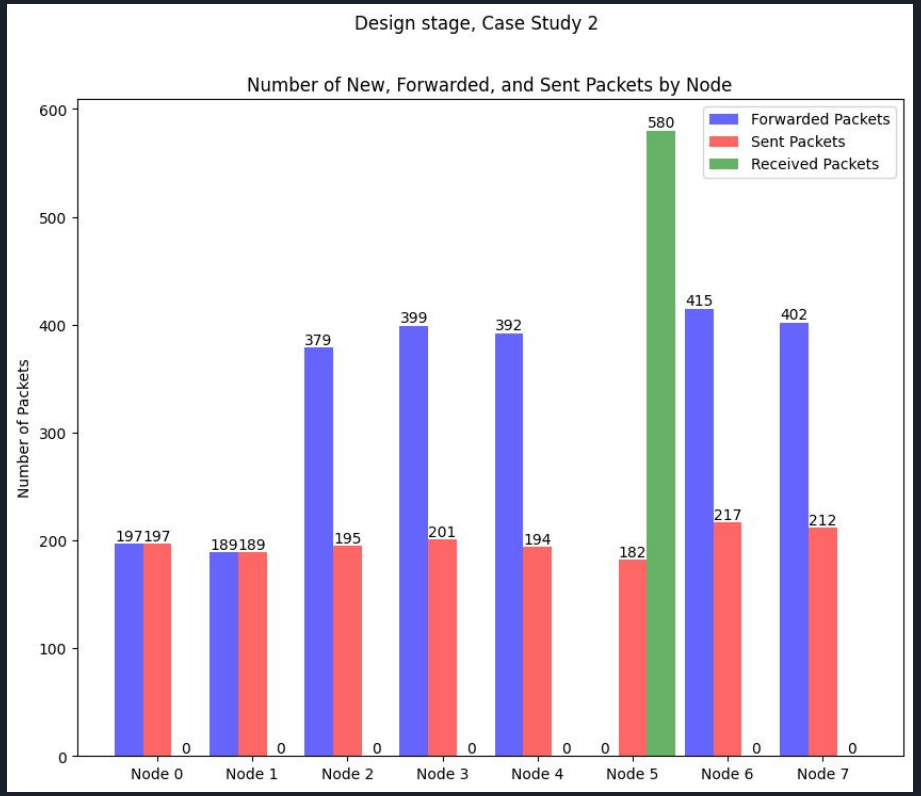
- No depende de número ni nombre de nodos
- Depende de
  - Que la red tiene topología de anillo
  - Que la red es inmutable en el tiempo
  - Idea de escalabilidad
    - Para facilitar el desarrollo del punto estrella
  - Posibilidad de DEBUGGING (flag DEBUG)

Idea de la estrategia de enrutamiento

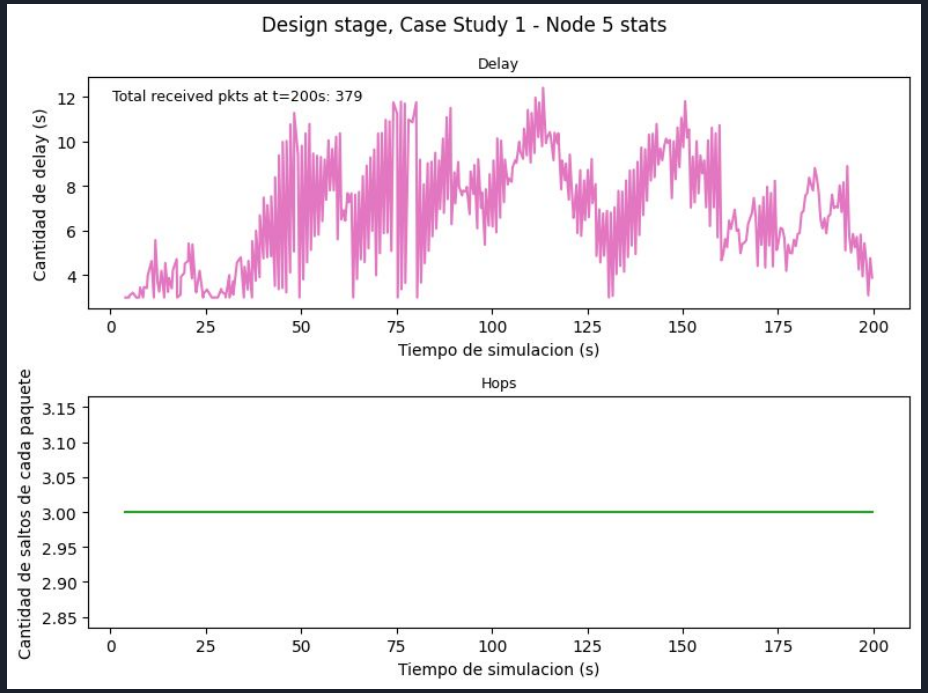
El proceso es bastante similar al protocolo de enrutamiento por estado de enlace.

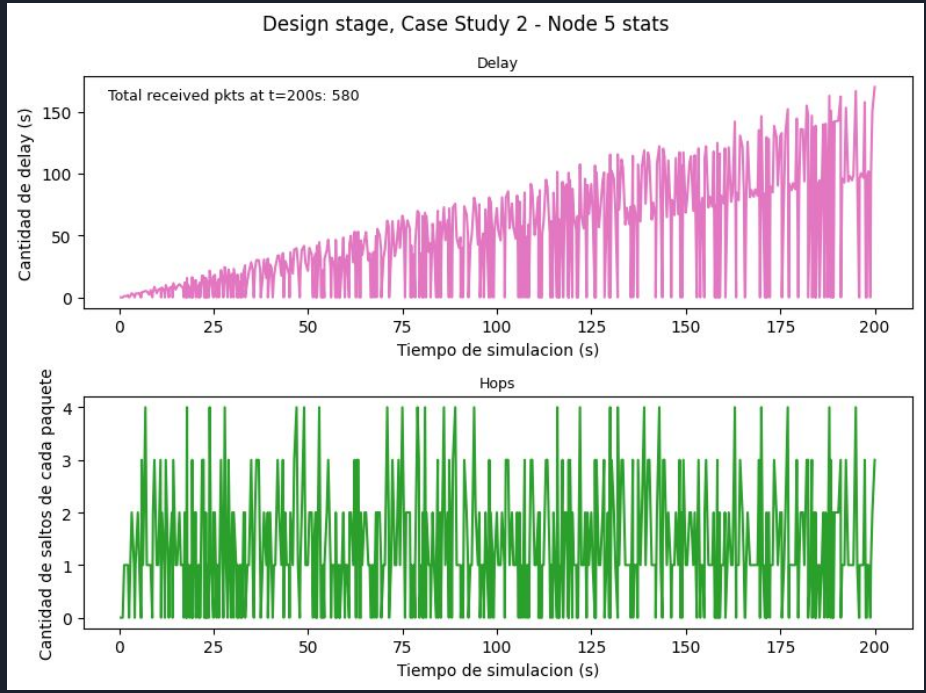
- Consulto por información a mis vecinos (NeighborInfo)
- Obtengo información de la red general (LSP)
  - Se usa inundación con registro
- Se crea el grafo representativo
- Se calcula el enrutamiento óptimo con BFS para saber si tengo que ir por izquierda o derecha (permite mayor simplicidad del BFS)
  - Como es inmutable, es parte del precómputo antes de comenzar a enviar paquetes





Análisis de llegada de paquetes en casos con algoritmo implementado





Gráficos de delay y saltos de cada paquete en casos con algoritmo implementado

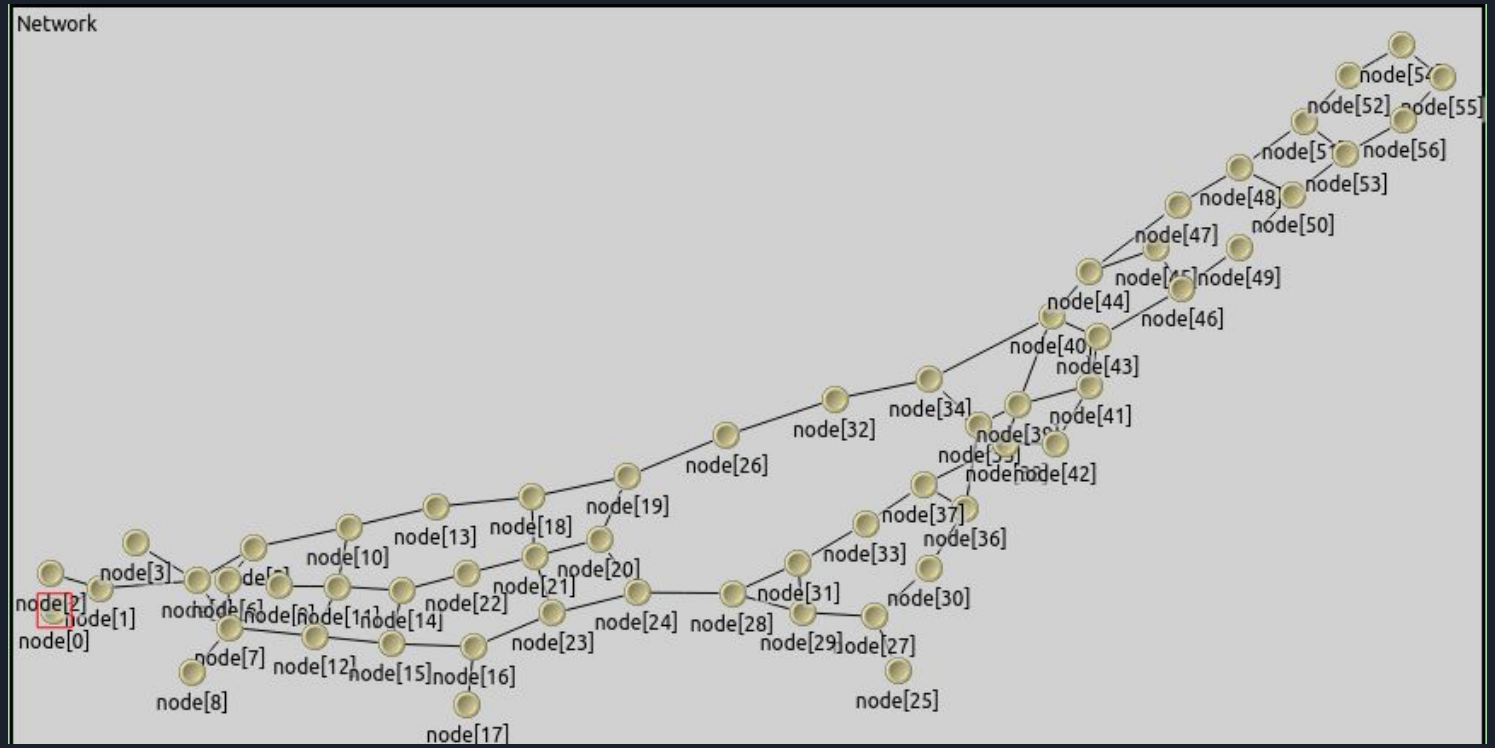
Tarea Estrella

Ahora, lo que se pretende es implementar una estrategia de enrutamiento que se adapte a *cualquier tipo de topología*.

En nuestro caso, hemos extendido la idea de la *Tarea de Diseño* agregando el paso de “reconocimiento” de la red (dado que puede tener cualquier topología) y considerando también que en **ningún momento se produce algún cambio**.

En esta parte, los nodos pueden tener varias salidas (gates) pero algunas pueden estar conectadas o no.

El Network particular que se considera (para correr el algoritmo y hacer el análisis de los casos) es el siguiente:



Puntos a tener en cuenta

- No depende de
  - Número o nombre de nodos
  - Cantidad de interfaces o que se usen todas
  - La topología
- Depende de que la *red es inmutable*
- Escalabilidad
  - Se planteó el algoritmo para que la mejora para **redes mutables** se tuviera que implementar solo en la función *actualizeNetworkInformation* en *Net*.
- Posibilidad de DEBUGGING (funciones de print en Net)

Idea de la estrategia de enrutamiento

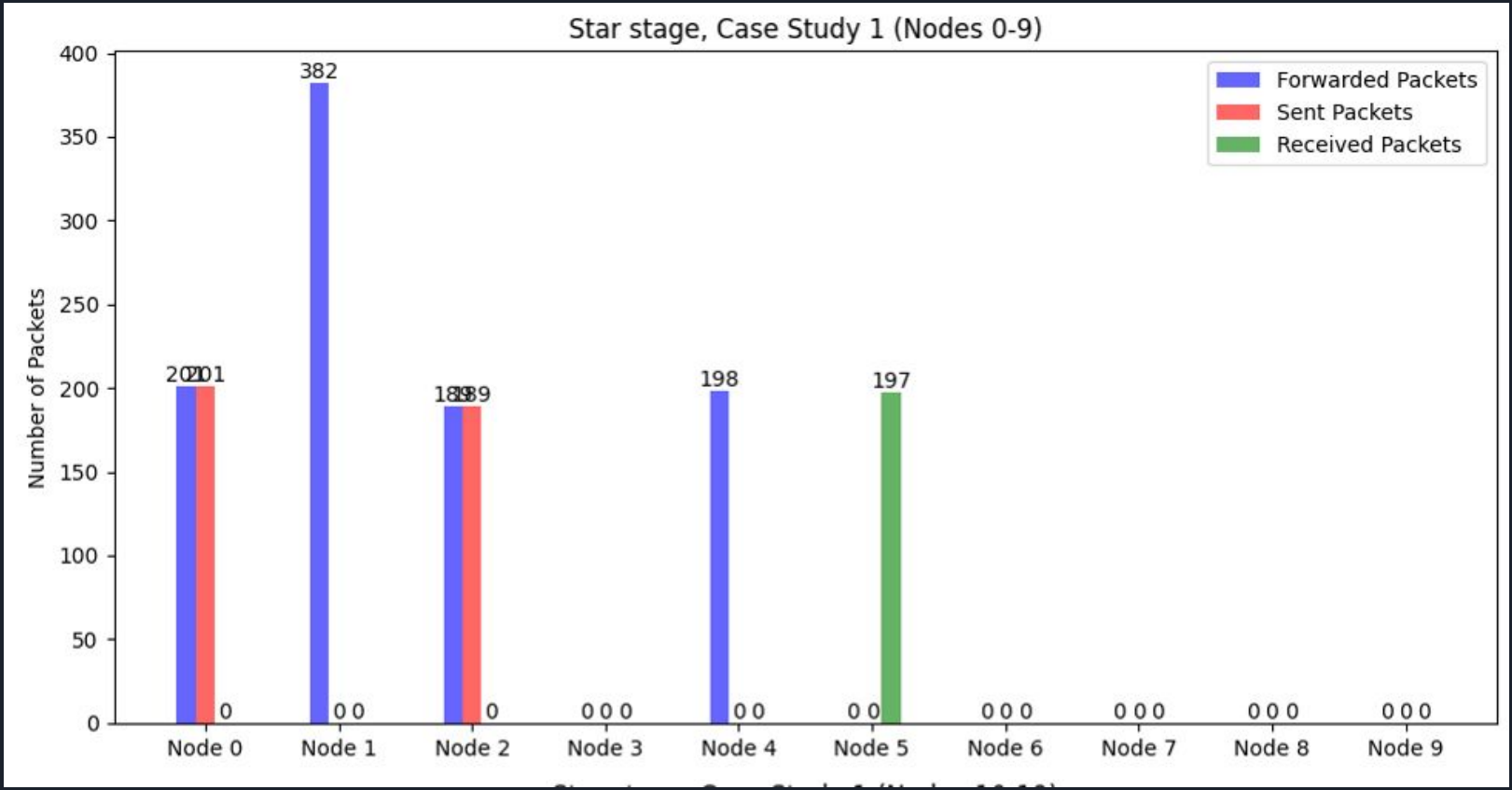
Similar al punto anterior y al protocolo de enrutamiento de estado de enlace:

- Obtengo la información de mis vecinos (NeighborInfo)
  - Para ello, tengo que chequear si la gate está conectada a algún nodo (uso funciones de gate)
- Obtengo información de la red en general (LSP)
- Creo el grafo representativo juntando todos los LSP
  - Inundación con registro
- Cálculo del enrutamiento óptimo con BFS luego de cada actualización ante llegada de LSP (precómputo)
- El envío de los paquetes se realiza en base a las líneas de salida obtenidas en el precómputo

Diseño vs Estrella

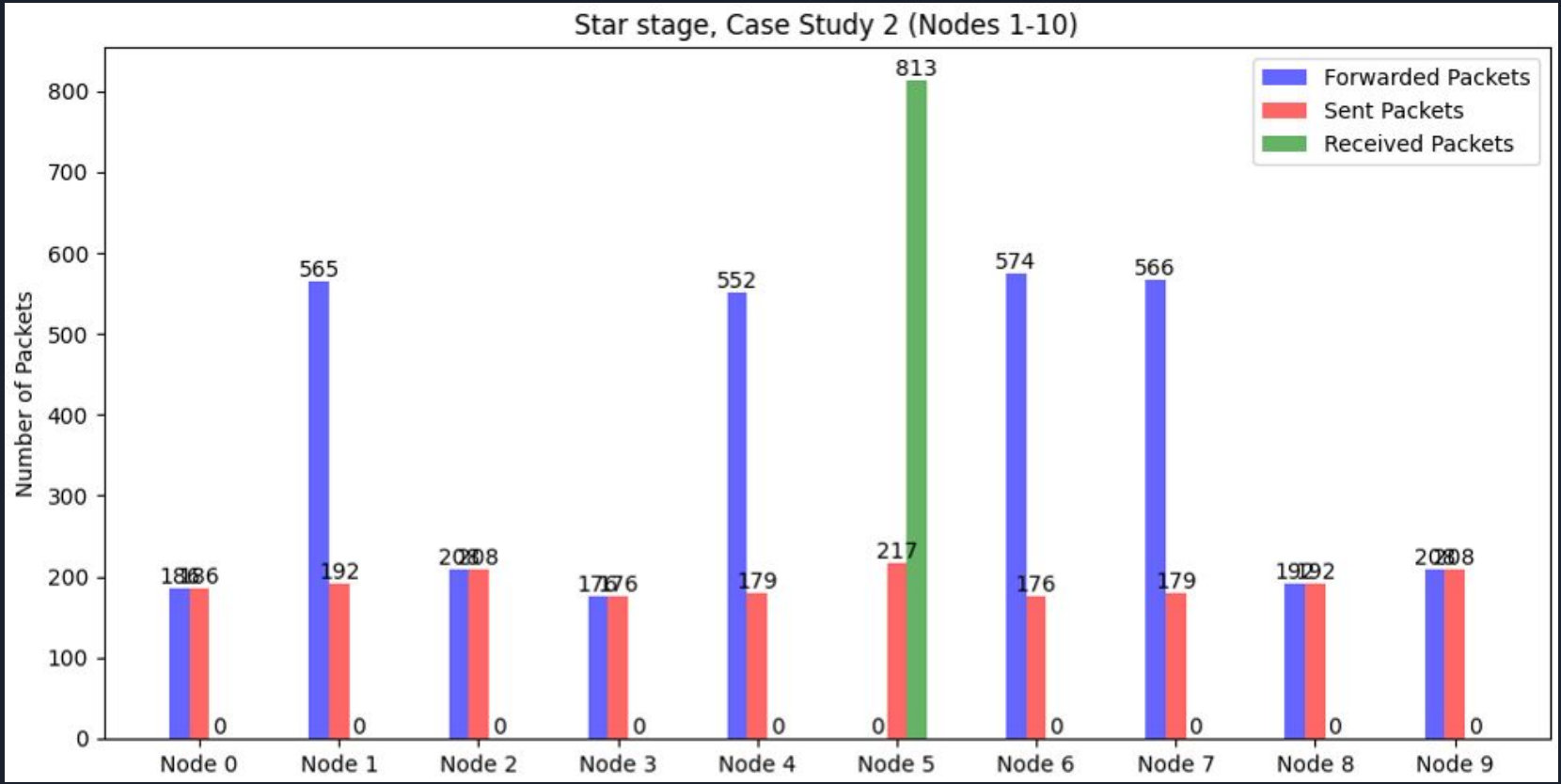
La *idea base* es similar entre las dos partes (solo que una aplicada a una topología en particular y la otra a cualquiera), salvo que se prefirió implementar el **estrella desde cero** por los siguientes motivos:

- Organizar mejor el código realizado
- La primera parte (actualización del grafo representativo) sale de forma rápida. Se tuvo que prestar especial atención en el BFS (precómputo) ya que se realiza de a partes a medida que llegan los LSP.



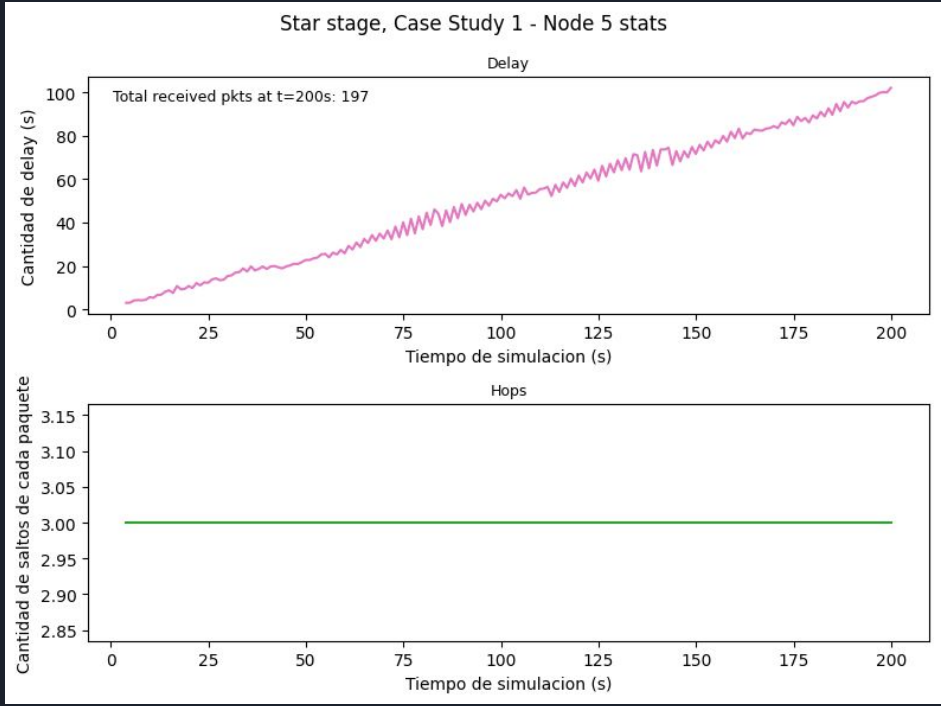
Análisis de llegada de paquetes en caso 1 con algoritmo implementado.

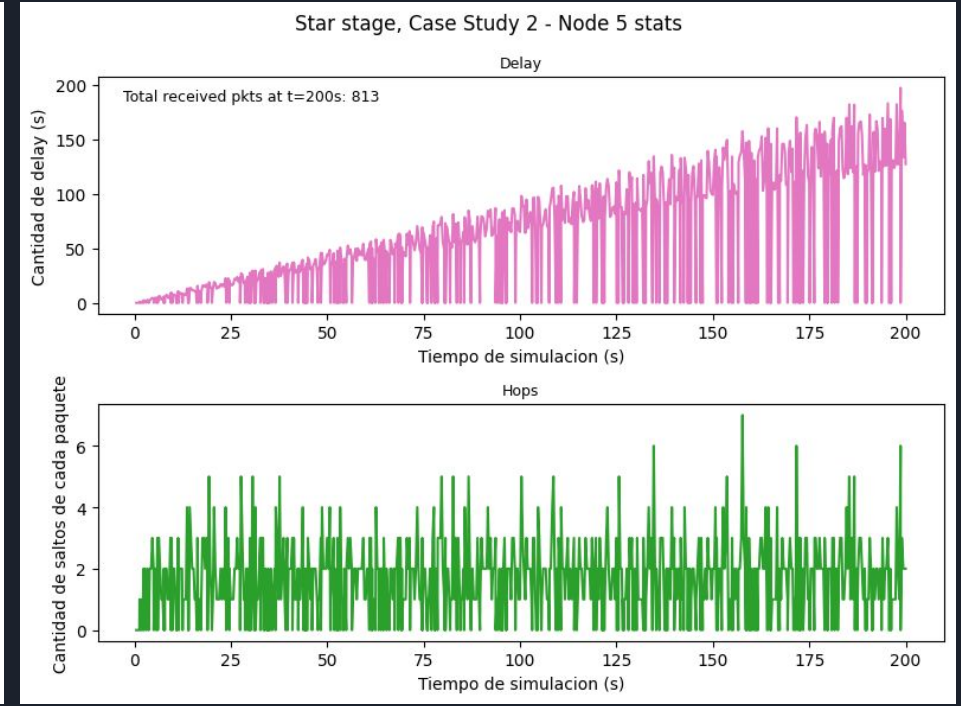
En el caso 1, desde el nodo 6 en adelante no hay actividad en la red.



Análisis de llegada de paquetes en caso 2 con algoritmo implementado.

Por conveniencias de tamaño de slide solo mostramos los gráficos para los nodos 0-10.





Gráficos de delay y saltos de cada paquete en casos con algoritmo implementado