

Sistemas Operativos y Redes II

Trabajo Práctico 2. Análisis de Redes.

**Alumnos: Alan Echabarri - alanechabarri@gmail.com Juan M. Corbera - jmcorbera@gmail.com**

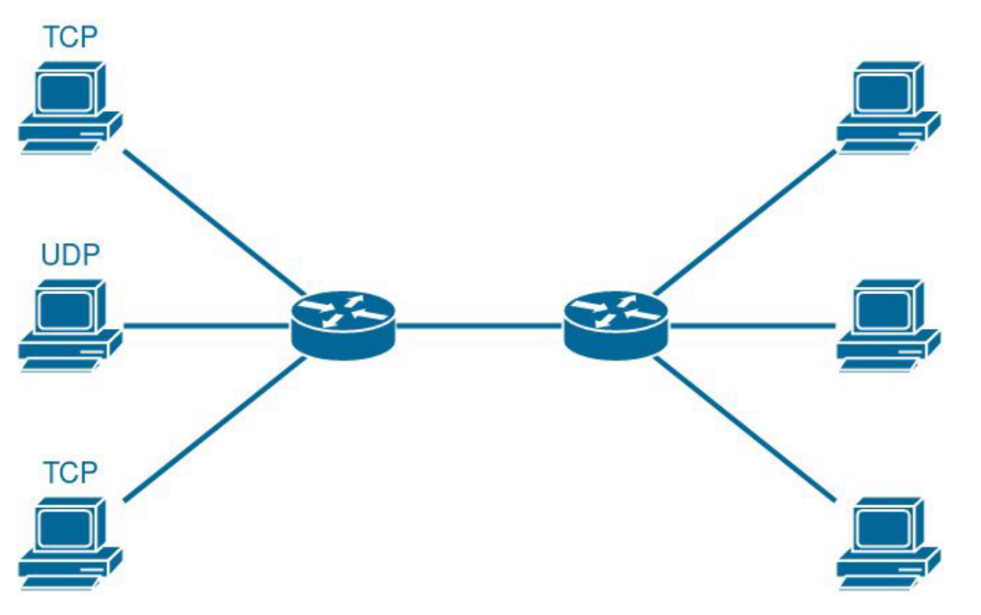
**Profesor: Alexis Tcach**

**Comisión: 1**

Primer Semestre 2022

# Introducción

En el siguiente informe se va a detallar el análisis de redes poniendo el foco en las consignas pedidas en el TP2 de Sistemas Operativos y Redes II.

En primer lugar, implementaremos el escenario sobre el cual realizaremos las diferentes prácticas. Se diseña para esto un escenario con 3 emisores on/off application, 3 receptores y dos nodos intermedios, utilizaremos lo que normalmente es conocido como Dumbbell topology. La red contará con 2 transmisores TCP y un transmisor UDP. Las conexiones serán cableadas.

Para las primeras prácticas, solo utilizaremos 2 transmisores TCP y se buscará saturar el canal de transmisión, medir la velocidad de transferencia. Se mostrará mediante gráficos tamaño de colas de recepción, ventana de TCP y se explicaran las distintas etapas del protocolo TCP. También se calculará el ancho de banda del canal.

Luego con la misma configuración se agregará un nodo UDP y se mostrará los cambios que suceden.

En una segunda etapa se agregará un algoritmo llamado Westwood, se explicará su funcionamiento. El mismo será implementado en el primer escenario probado en el punto anterior, utilizando 2 transmisores TCP y presentaran los datos obtenidos de las muestras realizadas.

# Configuración

Las distribuciones de Linux utilizadas para el desarrollo y las distintas pruebas fueron:

● Ubuntu 18.04 LTS

Para las diferentes prácticas se utilizó un software de simulación de redes con eventos discretos, desarrollado en C++, conocido como Network Simulator 3 (NS3). El mismo es un software libre de licencia GNU GPL v2 (General Public License GNU Versión 2).

NS3 cuenta con una documentación muy extensa de donde se puede obtener los prerrequisitos de instalación, como así también las diferentes configuraciones para poder utilizar la aplicación.

* <https://www.nsnam.org/docs/tutorial/html/index.html>

Para esta serie de prácticas se utilizó la release ns-3.30.1. Descargada desde el siguiente link:

* <https://www.nsnam.org/releases/ns-3-30/>

Instalación Simple:

1. Descomprimimos la carpeta del NS3, nos ubicamos dentro de la misma y ejecutamos:

* build.py: ./build.py –enable-examples –enable-tests
* bake: ./bake.py build

1. Nos ubicamos dentro de la carpeta ns-3.30.1 y ejecutamos:

Compilar waf

* ./waf clean
* ./waf -d optimized –enable-examples –enable-tests configure
* ./waf clean
* ./waf -d debug –enable-examples –enable-tests configure
* ./waf

1. Para verificar que todo haya quedado bien configurado, colocar algunos de los archivos.cc (ejemplo: ../ns-3.30.1/examples/tutorial/hello-simulator.cc) dentro de la carpeta scratch y ejecutar/.

* ./waf --run nombre\_archivo

Para graficar las pruebas se utilizó Gnuplot, que es una herramienta visual para graficar las trazas de las diferentes simulaciones realizadas. Este programa nos permite generar gráficas 2D y 3D. Puede producir resultados directamente en pantalla y en varios formatos de imagen, como PNG, JPEG, etc.

# Desarrollo

## Parte 1

Se diseñó un escenario con 3 emisores on/off application, 3 receptores y dos nodos intermedios. Se conectaron los 3 emisores a un nodo, luego éste a otro y finalmente éste a los 3 destinos finales. Uno de los emisores se definió como UDP y los otros 2 TCP. Con conexiones cableadas de 100Kb/s para todos los enlaces.

### Prueba con 2 emisores TCP

**Saturación de canal y velocidad de transferencia.**

* Explicar saturación de canal y velocidad de transferencia

Primero nos aseguramos de ubicarnos dentro de la carpeta ns-3.3x

luego corremos el programa con el comando

/waf --run scratch/dumbbell-tcp-udp

Tras hacer esto, el programa creara una serie de archivos pcap, y archivos .data

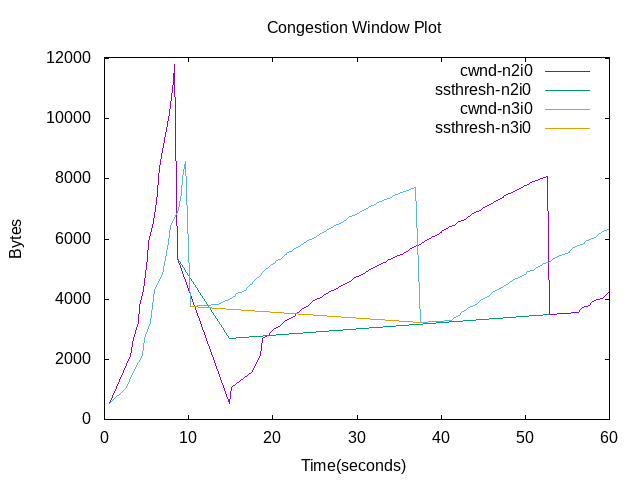
Con estos archivos revisaremos como se satura el canal y la velocidad de transferencia. Visualizaremos el grafico generado por el comando “gnuplot config\_cwnd\_ssthresh.plt” , nos dara como salida una imagen png, que usa los 4 archivos “.data” que aparecieron anteriormente. Estos archivos contienen la variacion en tiempo de la ventana de congestion y ssthresh de los nodos 2 y 3, via interfaz 0.

El umbral sstresh comienza con un valor arbitrariamente alto, para este trabajo, ese valor inicial lo omitimos.

**SLOW START**

--Creo que se esta utilizando tcp newReno en la version 3,30,1 ns3 por defecto---

Como observamos en el grafico las ventanas de congestion de ambos nodos emisores comienzan en cero, y van incrementandose exponencialmente los paquetes que envian por la red. Aproximadamente en el segundo 8,64 el nodo2 tras recibir 3 ack de paquetes Tcp, duplicados. Es decir recibe el mismo ack una y otra vez,Realiza Fast retrasmit y se entra en fast recovery.



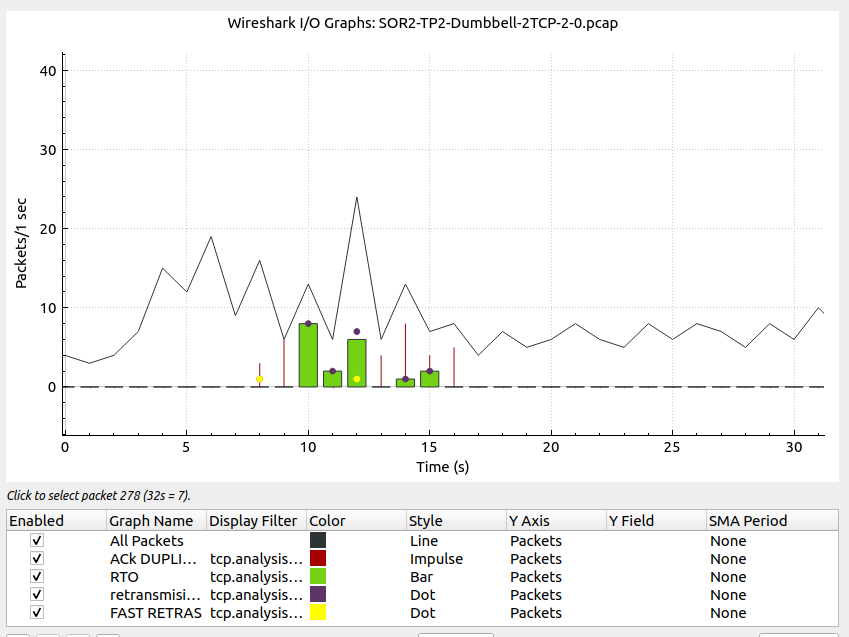
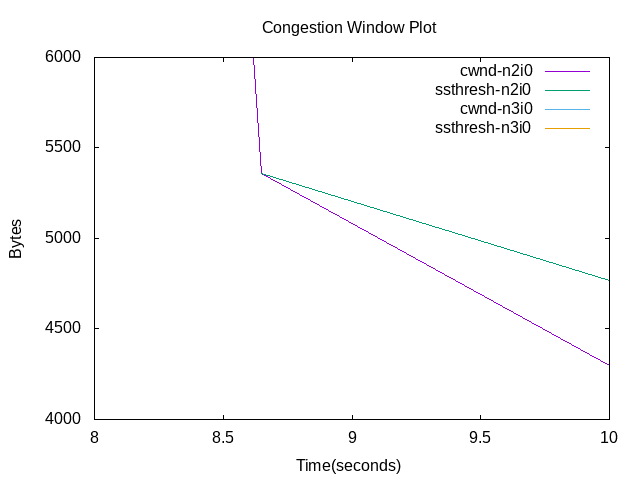
**FAST RECOVERY**

Como se puede observar en el grafico con zoom

Se setea el umbral de congestion en sstresh=CWND/2.

Del segundo 8 al 10 el rtt del nodo 2 es aproximadamente de 2segundos. Y Durante este tiempo recibe ack duplicados por lo que durante este tiempo el CWND solo baja. En el segundo 10 se recibe un RTO por lo que cwnd deberia setearse en 1mss=536bytes, pero al parecer no lo hace hasta el segundo 15,

desde el segundo 15 claramente entra en slow start y casi llegando al 20, al cruzar el ssthresh congestionAvoidance,



**Congestion Avoidance**

**Gráficos**

* Colocar captura de imágenes
  + Tamaño de colas de recepción
  + Ventana de TCP
  + Cualquier mecanismo que muestre lo que sucede

**Protocolo TCP**

* Introducción a las distintas etapas del protocolo TCP
* Explicar en los gráficos capturados

Explicaremos como el protocolo TCP utiliza distintas técnicas para evitar la congestión de la red, para esto el protocolo implementa un mecanismo de control entre el emisor y el receptor. La congestión ocurre cuando los emisores envían un volumen de paquetes mayor al que un receptor puede procesar. Si el emisor percibe que hay poca congestión entre él y el receptor va a incrementar su tasa de emisión. Por lo contrario, si percibe mucha congestión reducirá la misma. Esto lo logrará mediante la variación del tamaño de la ventana SND.WND. Este tamaño se determinará por el valor de la ventana de recepción y por el valor de la ventana de congestión (CWND) que dependerá del algoritmo de control de congestión. Este algoritmo percibe la congestión de la red a través de los eventos de pérdidas de segmentos que pueden ser: la recepción de 3 ACK´s duplicados o la expiración del TIMEOUT (tiempo de espera).

Se han implementado varias técnicas de control de congestión en TCP para limitar la velocidad de envío de datos que ingresan a Internet mediante la regulación del tamaño de la CWND. Las técnicas que utiliza, son los algoritmos **Slow Start**, **Congestion Avoidance**, **Fast Retransmit** y **Fast Recovery**. Las primeras dos fases del mecanismo de control de lacongestión, *Slow Start* y *Congestion Avoidance*, regulan la cantidad de paquetes inyectadosen la conexión y son responsables de la detección de la congestión, controlan las transmisiones. Mientras que los otros dos algoritmos, *Fast Retransmit* y *Fast**Recovery*, reaccionan para superar la congestión y proporcionar un mecanismo que acelera larecuperación de la conexión.

Además del valor CWND, el algoritmo suma otra variable llamada SSTHRESH (umbral *entre slow start* y *congestion avoidance*). Esta variable va a determinar si se va a utilizar el mecanismo de arranque lento o el de evitación de la congestión. Cuando CWND se encuentre por debajo del umbral se utilizará *Slow Start* y cuando se encuentre por encima se utilizará *Congestion Avoidance*. El valor de SSTHRESH se calculará cuando ocurra un evento de pérdida de segmentos. Se tomará el valor de CWND (con al menos dos segmentos) y se lo dividirá por 2, por lo tanto, SSTHRESH = CWND / 2.

**- Slow Start (Arranque lento)**

El algoritmo Slow Start sugiere que se incremente el tamaño de la CWND en el mismo número de segmentos con acuse de recibo, por cada ACK recibido con éxito. Al recibir un ACK, se puede enviar el doble de la cantidad de datos que fueron reconocidos por ese ACK (política de aumento multiplicativo). TCP empieza la comunicación probando la red, enviando un solo segmento, y fija el tamaño de la CWND a un segmento. Es decir, en el primer RTT la ventana aumenta el valor de CWND a dos, con lo que se envían dos segmentos, luego aumenta a cuatro, se envían cuatro segmentos y así sucesivamente utilizando múltiplos de dos. Por lo tanto, esto provoca que se duplique la tasa de envío por cada RTT y que aumente exponencialmente la CWND hasta que perdemos un paquete que es un signo de congestión.

A partir de esto, disminuimos nuestra tasa de envío y reducimos la CWND a uno para garantizar la liberación de recursos de red.

**- Congestion Avoidance (Evitación de la congestión)**

Cuando la CWND alcanza el valor de la variable umbral denominado SSTHRESH (umbral de Slow Start), TCP abandona la fase de Slow Start y entra en una fase Congestion Avoidance para encontrar la capacidad disponible de la red y no saturarla.

Por lo tanto, el valor de la ventana de transmisión en cada momento da una indicación de la tasa de transmisión de TCP, ya que, en ausencia de errores se va a transmitir durante un RTT como máximo tantos segmentos como indique la ventana de transmisión. Se abandona el estado de Congestion Avoidance cuando el emisor TCP detecta un evento de congestión por expiración de timeout o por la recepción consecutiva de tres ACK duplicados.

En el primer caso, se reduce el valor de CWND a 1 y se define el valor de SSTHRESH a la mitad del valor alcanzado por CWND o al menos dos segmentos. En caso contrario, se reconfigura la variable SSTHRESH al valor CWND/2, divide a la mitad el valor de CWND y le suma 3 de los ACK duplicados recibidos luego, establece el valor de SSTHRESH a la mitad del valor de CWND y se invoca a la estrategia Fast Retransmit

**- Fast Retransmit (Retransmisión rápida)**

La fase de Fast Retransmit permite que un emisor conozca que se ha perdido un segmento incluso antes de que venza el temporizador de retransmisión. Cuando se recibe un segmento fuera de orden, el receptor genera lo que se denomina un ACK duplicado, es decir, vuelve a asentir los mismos datos que ya asintió anteriormente. Lo que hace es retransmite el segmento, donde no se requiere tiempo de espera para que expire el temporizador de retransmisión, solo asume en que en cuanto llegan al emisor tres ACKs duplicados (cuatro ACKs idénticos) son un buen indicador de un segmento perdido. Entonces se entra en Fast Recovery

**- Fast Recovery (Recuperación rápida)**

Los segmentos recibidos luego del segmento perdido generaron los ACKs duplicados, esto quiere decir que todavía hay datos entre los dos extremos. TCP no quiere reducir el flujo bruscamente, por lo tanto, evita el ingreso a *Slow Start* y de *Congestion Avoidance*. Luego, el emisor establece SSTHRESH en la mitad de la CWND actual y ajusta el valor de la CWND igual a uno, esto lo hace por cada ACK duplicado que recibió de los tres segmentos que ya habían sido transmitidos luego del segmento perdido CWND = SSTHRESH + 3, reiniciando la fase Slow Start.

**Ancho de Banda del Canal**

**Calcular el ancho de banda del canal**

**Explicar que sucede (Anomalías? explicarlas)**

### Prueba con 2 emisores TCP y uno UDP

**Ancho de Banda de los diferentes Emisores**

**TCP distintas acciones del protocolo**

## Parte 2

Una implementación de TCP Westwood.

Westwood emplea el paradigma de control de congestión AIAD (Aumento Aditivo/Disminución Adaptativa). Cuando ocurre un episodio de congestión, en lugar de reducir a la mitad el cwnd, este protocolo intenta estimar el ancho de banda de la red y usa el valor estimado para ajustar el cwnd. Westwood realiza el muestreo del ancho de banda en cada recepción de ACK.

Los dos métodos principales en la implementación son CountAck (const TCPHeader&) y EstimateBW (int, const, Time). El método CountAck calcula el número de segmentos reconocidos al recibir un ACK. El valor de BudgetBW estima el ancho de banda en función del valor devuelto por CountAck y el intervalo de muestreo (tiempo entre llegadas del último ACK).

# Referencias

<https://www.nsnam.org/docs/release/3.30/tutorial/html/index.html/html/index.html>

https://www.nsnam.org/doxygen/dumbbell-animation\_8cc\_source.html

<https://www.ibm.com/docs/ko/psfa/1.6?topic=throughput-tcp-window-size-latency>

http://intronetworks.cs.luc.edu/current2/mobile/ns3.html