

Sistemas Operativos y Redes II

Trabajo Práctico 2. Análisis de Redes.

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

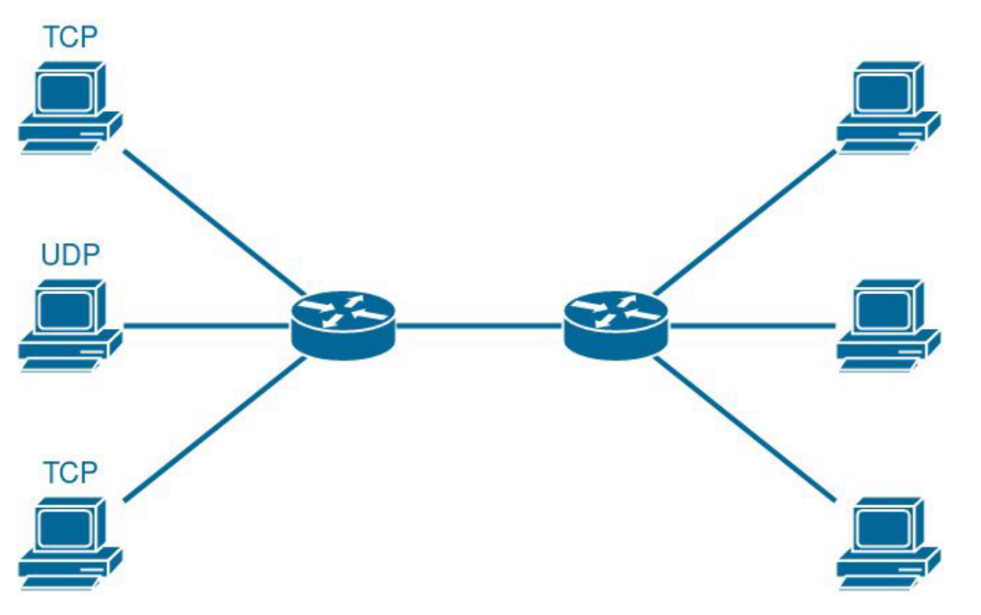
**Profesor: Alexis Tcach**

**Comisión: 1**

Primer Semestre 2022

# Introducción

En 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, utilizamos 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

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

En este apartado vamos a explicar las distintas etapas del protocolo TCP 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 enrutador puede procesar. Si el emisor percibe que hay poca congestión entre él y el receptor va a incrementar su tasa de emisión. Por otro lado, si percibe lo contrario reducirá la misma. Esto lo logrará mediante la variación del tamaño de la ventana SND.WND. El tamaño de la ventana SND.WND 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 un triple ACK duplicado o la expiración del tiempo de espera (timeout). El algoritmo de control de congestión se compone de cuatro mecanismos esenciales: arranque lento (slow start), evitación de la congestión (congestion avoidance), retransmisión rápida (fast retransmit) y la recuperación rápida (fast recovery). TCP Reno utiliza estos mecanismos y es considerado el algoritmo estándar de control de congestión. Además del valor cwnd, el algoritmo suma otra variable llamada umbral (ssthresh). Esta variable va a determinar si se va a utilizar el mecanismo de arranque lento o el de evitación de la congestión. Cuando la ventana de congestión se encuentre por debajo del umbral se utilizará arranque lento y cuando se encuentre por encima se utilizará evitación de la congestión. 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)**

Como se explicó antes, este algoritmo se utiliza cuando se está por debajo del umbral, por lo tanto, siempre lo utilizaremos al inicio de la transmisión. Inicialmente el valor de la ventana de congestión se establece en 1 MSS (también puede tener un valor distinto) y se incrementará en un segmento por cada ACK recibido. Por lo tanto, como por cada ACK, cwnd = cwnd + 1, durante esta etapa la ventana de congestión crecerá exponencialmente por cada RTT. No es exactamente exponencial ya que el receptor puede retrasar sus ACK, generalmente enviando un ACK por cada dos segmentos que recibe.

El emisor puede saturar al enrutador intermedio y este comenzará a descartar segmentos provocando una comunicación indirecta hacia el emisor para realizar los ajustes pertinentes al cwnd (ventana de congestión).

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

Este algoritmo entra en funcionamiento al momento que el valor de la ventana de congestión supera el valor del umbral. A diferencia del arranque lento, en esta etapa, por cada ACK que se recibe, cwnd = cwnd + 1/cwnd. Por lo tanto, la ventana de congestión se incrementará en 1 MSS por cada RTT, o sea, crecerá de forma lineal.

**Fast Retransmit (Retransmisión rápida)**

El emisor, controlando los ACKs duplicados, puede detectar una pérdida de segmento antes que expire el tiempo de espera (timeout). El ACK duplicado tendrá el número que correspondía a la secuencia que se esperaba recibir.

Si se reciben tres ACKs duplicados el emisor considera a un segmento como perdido. Aquí es donde el mecanismo de retransmisión rápida se ejecuta y realiza la retransmisión del segmento perdido antes de que el tiempo de espera termine.

El emisor asume que si existió un re-orden en la entrega de segmentos pueden existir uno o dos ACKs duplicados. Por lo tanto, al desconocer si los primeros dos ACKs duplicados son por un reordenamiento o por una pérdida, se debe esperar a recibir tres ACKs duplicados para dar por perdido un segmento.

**Fast Recovery (Recuperación rápida)**

Este algoritmo se ejecuta cuando el emisor recibe tres ACKs duplicados en la etapa slow start y de prevención de la congestión, aquí se asume que el segmento se perdió y lo retransmite inmediatamente.

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 la etapa de arranque lento y se mantiene en la de evitación de la congestión. Luego establece el valor del umbral a la mitad del valor de la ventana de congestión (ssthresh = cwnd / 2) y el valor de la ventana lo establece en el valor del umbral + 3 (cwnd = ssthresh + 3), esto lo hace por cada ACK duplicado que recibió de los tres segmentos que ya habían sido transmitidos luego del segmento perdido. También, si llega otro ACK duplicado se aumenta el cwnd por el segmento adicional recibido, si es posible, se transmitirá un nuevo segmento.

Esta etapa finalizará una vez que se recibió el ACK del segmento retransmitido, luego vuelve a la fase de evitación de la congestión. Si el evento de pérdida fue un timeout se volverá a la fase de arranque lento entonces el cwnd se “desinfla” sin mantener el valor.

Esta mejora y produce un alto rendimiento cuando se tiene una congestión moderada y especialmente para ventanas grandes.

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