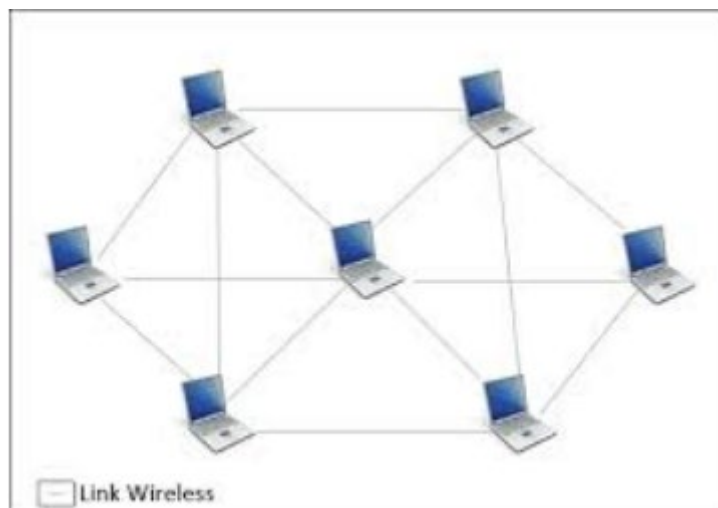


Sistemas Operativos y Redes II

Segundo Semestre 2023

Trabajo Práctico N°2: Análisis de redes



Alumnos:

- | | |
|----------------------|------------------------------|
| • Crevatin, Alan | <alan.univ@hotmail.com> |
| • Fandiño, Ma. Belén | <belen10.11.97bfo@gmail.com> |
| • Mendoza, Leonel | <leomendoza274@gmail.com> |
| • Palacio, Ezequiel | <nico.p22013@gmail.com> |

Docentes:

- Chuquimango, Benjamin
- Curto, Luis
- Echabarri, Alan

Contenido

Introducción.....	4
Análisis de red	4
TCP - Definición	4
Three Way Handshake	4
Control de congestión	5
• “Ventana de congestión”	5
• “Umbral de congestión”	5
• Slow Start:	5
• Congestion Avoidance:	5
• Fast Retransmission:	5
• Fast Recovery:	5
• TCP Tahoe:.....	6
• TCP Reno:	6
• TCP New Reno:	6
Fin de conexión - Four-way handshake	6
UDP	7
Herramientas	8
Ns-3:.....	8
Wireshark:	8
GNUplot:	8
Desarrollo	8
Diseño de la topología de red	8
Primera parte: TCP - Código	9
Análisis del código	10
Análisis de la red	12
TCP - Handshake	12
Congestión de la red	14
Ancho de Banda	21
Ancho de Banda - Definición:	21
Velocidad de Transferencia:.....	21
Ancho de banda promedio	21
Velocidad de transferencia	22
TCP - Fin de Conexión	24
Segunda Parte: TCP y UDP.....	25

Conclusión	28
Bibliografía	29

Introducción

Análisis de red

Para este trabajo vamos a analizar un escenario de red que se conoce como Dumbbell Topology, en donde contaremos con 3 emisores on/off application, 3 receptores y dos nodos intermedios. Uno de los emisores será UDP y los dos restantes TCP.

El objetivo es simular y entender cómo se comportan los diferentes protocolos de transmisión entre diferentes dispositivos. Para esto primero vamos a definir TCP y sus características, analizar el comportamiento en su conexión y transmisión de datos, y por último vamos a definir y compararlo con UDP.

TCP - Definición

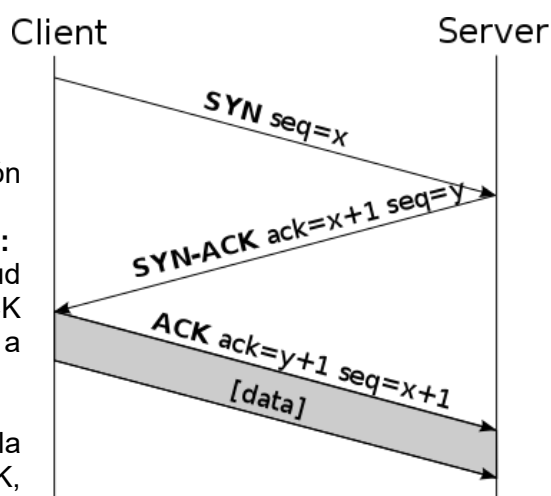
El Protocolo de Control de Transmisión (TCP) es un protocolo de comunicación de red que proporciona una transmisión confiable y ordenada de datos entre dispositivos en una red. Se encarga de dividir la información en paquetes, establecer conexiones, confirmar la recepción de datos y retransmitir la información si es necesario, asegurando así una comunicación eficiente y sin errores en entornos de red.

Three Way Handshake

Es un proceso fundamental en la comunicación mediante el Protocolo de Control de Transmisión (TCP). El mismo se basa en el intercambio de mensajes entre dos dispositivos con el fin de establecer una conexión antes de que realmente comience la transmisión de datos.

El proceso consta de tres pasos:

- **Solicitud de conexión (SYN):**
El dispositivo que inicia la conexión envía un mensaje SYN al otro.
- **Respuesta de conexión (SYN-ACK):**
El dispositivo que recibió la solicitud responde con un mensaje SYN-ACK para indicar que está dispuesto a establecer la conexión.
- **Confirmación de conexión (ACK):**
Finalmente, el dispositivo que inició la conexión envía un mensaje ACK, confirmando la conexión establecida.



Este proceso asegura que ambos extremos estén sincronizados y listos para intercambiar datos de manera fiable.¹

¹ más adelante podremos observar este proceso en la pantalla principal de WireShark

Control de congestión

El control de congestión en redes de comunicación hace referencia a las estrategias y mecanismos implementados por el protocolo TCP para evitar la saturación de la red. El mismo regula la cantidad de datos que se envían en un periodo de tiempo determinado, ajustándose dinámicamente según las condiciones de la red. Su objetivo es prevenir la pérdida de paquetes, optimizar el rendimiento y garantizar una transmisión eficiente, adaptándose a las variaciones en la carga de la red.

Para controlar dicha congestión utiliza las siguientes variables:

- **“Ventana de congestión”** (Congestion Window o CWND), la misma es utilizada por el emisor para limitar la cantidad de datos que puede tener en tránsito en un tiempo específico.
- **“Umbral de congestión”** (“congestion threshold” o SSTH) trata de ser una estimación del tamaño de la ventana del emisor a partir del cual existe riesgo de congestión. El valor inicial del SSTH debería establecerse en un valor arbitrariamente alto, pero tendría que reducirse como respuesta a la congestión.

Dentro del control de congestión, TCP hace uso de, principalmente, 4 algoritmos:

- **Slow Start:** el cual se utiliza para incrementar la Ventana de congestión de manera exponencial. Este algoritmo es utilizado mientras $CWND < SSTH$. La ventana de congestión se inicia con el valor de un segmento de tamaño máximo (MSS). Cada vez que se recibe un ACK, la ventana de congestión se incrementa en tantos bytes como hayan sido reconocidos en el ACK recibido. En la práctica, esto supone que el tamaño de la ventana de congestión será el doble por cada RTT, lo que da lugar a un crecimiento exponencial de la ventana. Una vez que se alcance, el SSTH, se entrará en la etapa de Congestion Avoidance.
- **Congestion Avoidance:** En este caso, el tamaño de la CWND crecerá de manera lineal incrementando un segmento por cada RTT. En el [RFC 5681](#), se menciona que una fórmula posible que TCP puede usar durante esta etapa es:
$$cwnd += \frac{SMSS^2}{cwnd}$$

²donde SMSS es el tamaño del segmento a transmitir

Nota: todos los incrementos durante esta etapa se ejecutan luego de cada ACK recibido.

- **Fast Retransmission:** este algoritmo está destinado a resolver una situación de pérdida de paquetes sin esperar a que se cumpla el timeout (RTO). Se basa en el hecho de que la recepción de tres o más ACKs idénticos (Dup ACKs) es un indicador de que un segmento se ha perdido, y al mismo tiempo la congestión de la red no es tan grave como para tomar medidas drásticas como en el caso de un timeout. Luego de la recepción de dichos ACKs, TCP realiza una retransmisión del segmento perdido.
- **Fast Recovery:** permite a TCP recuperarse más rápido de la pérdida de paquetes en comparación con Slow Start. Este proceso implica la retransmisión selectiva de los paquetes perdidos y la continuación de la transmisión de los paquetes que se han recibido correctamente. Este método ayuda a mantener un flujo de datos más constante y eficiente, reduciendo el impacto de la pérdida de paquetes en el rendimiento de la conexión TCP.

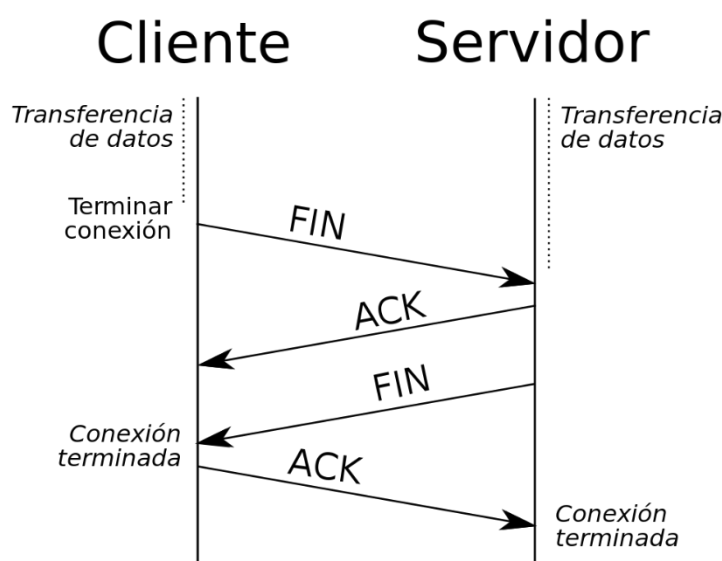
Por consiguiente, a lo largo del desarrollo del proyecto, hemos optado por utilizar el algoritmo New Reno. Antes de definirlo, debemos remontarnos a sus predecesores, Tahoe y Reno.

- **TCP Tahoe:** posee los algoritmos de slow start y congestion avoidance y, posteriormente, se le agregó el algoritmo de Fast Retransmit. La principal desventaja de esta implementación es que cada vez que se pierde un segmento, comienza nuevamente con el algoritmo de slow start, lo que demora demasiado el retorno al valor de las tasas anteriores.
- **TCP Reno:** cuando se detecta congestión, el TCP Tahoe reduce la ventana de congestión a un segmento, lo que dispara el algoritmo de Slow Start, mientras que TCP Reno dispara el algoritmo de Fast Recovery, reenviando el paquete perdido y reduce el umbral a la mitad, evitando disparar el algoritmo de slow start. Cuando recibe un nuevo ACK, sale de Fast Recovery y dispara el algoritmo de congestión avoidance. Esto permite que se recupere más rápido de la congestión.
- **TCP New Reno:** la optimización en esta versión de TCP mejora la pérdida individual y aislada de segmentos, pero no mejora las pérdidas en segmentos sucesivos. La diferencia con el TCP Reno es que cuando se está en la etapa de fast recovery, el arribo de uno nuevo ACK no hace que se salga de esta, hasta que todos los segmentos pendientes de confirmar antes del disparo del algoritmo tengan su acuse de recibo. De esta forma un ACK parcial implica que el segmento siguiente se perdió y debe ser retransmitido.

Fin de conexión - Four-way handshake

La fase de finalización de la conexión utiliza una negociación en cuatro pasos (four-way handshake), terminando la conexión desde cada lado independientemente. Sin embargo, es posible realizar la finalización de la conexión en 3 fases; enviando el segmento FIN y el ACK en uno solo. Cuando uno de los dos extremos de la conexión desea parar su "mitad" de conexión transmite un segmento con el flag FIN en 1, que el otro interlocutor confirmará con un ACK. Por tanto, una desconexión típica requiere un par de segmentos FIN y ACK desde cada lado de la conexión.

Una conexión puede estar "medio abierta" en el caso de que uno de los lados la finalice, pero el otro no. El lado que ha dado por finalizada la conexión no puede enviar más datos, pero la otra parte sí podrá.



UDP

El Protocolo de Datagramas de Usuario (UDP) es un protocolo de comunicación en redes que ofrece una transmisión de datos sin conexión y de naturaleza no orientada a la fiabilidad. En lugar de establecer una conexión antes de la transmisión, como en el caso de TCP, UDP simplemente envía paquetes de datos, conocidos como datagramas, de manera independiente. Aunque esto implica una menor garantía de entrega y orden, la simplicidad y la velocidad inherentes a UDP lo convierten en una elección valiosa para aplicaciones donde la velocidad y la eficiencia son prioritarias, como transmisiones en tiempo real y juegos en línea.

A continuación, se muestra un cuadro comparativo entre el protocolo UDP y el protocolo TCP.

Factor	TCP	UDP
Tipo de conexión	Requiere una conexión establecida antes de transmitir datos	No se necesita conexión para iniciar y finalizar una transferencia de datos
Secuencia de datos	Puede secuenciar datos (enviar en un orden específico)	No puede secuenciar u ordenar datos
Retransmisión de datos	Puede retransmitir datos si no llegan los paquetes	Sin retransmisión de datos. Los datos perdidos no se pueden recuperar
Entrega	La entrega está garantizada	La entrega no está garantizada
Comprobar si hay errores	Una exhaustiva comprobación de errores garantiza que los datos lleguen en buen estado	La comprobación de errores cubre los aspectos básicos, pero puede que no evite todos los errores
Emisiones	No es compatible	Sí es compatible
Velocidad	Lenta, pero entrega los datos completos	Rápida, pero existe el riesgo de que los datos se entreguen incompletos

Fuente: [TCP o UDP: comparación de protocolos TCP y UDP | Avast](#)

Herramientas

Ns-3:

Es un simulador de redes de código abierto basado en eventos discretos, diseñado principalmente para entornos educativos y de investigación. Este software brinda la capacidad de simular de manera efectiva tanto protocolos de comunicación unicast como multicast. Proporciona un ambiente virtual donde investigadores y estudiantes pueden modelar, analizar y evaluar el rendimiento de redes de comunicación, permitiendo un entendimiento más profundo de los protocolos y algoritmos en un contexto controlado. Es una herramienta poderosa para diseñar y probar redes virtuales antes de implementarlas en la realidad.

Wireshark:

Es una herramienta de análisis de protocolos de red que permite capturar y examinar el tráfico en una red, mostrando el flujo de datos y permitiendo a los usuarios inspeccionar y analizar la información para así poder comprender mejor qué es lo que está sucediendo en la red.

GNUplot:

Es una herramienta de trazado y visualización de datos. Permite generar gráficos de alta calidad a partir de conjuntos de datos, facilitando la interpretación de tendencias y patrones. Gnuplot es especialmente útil en entornos científicos y de ingeniería, donde la representación visual de datos es crucial para el análisis y la toma de decisiones.

Desarrollo

Diseño de la topología de red

Comenzamos el desarrollo diseñando la Dumbell Topology solicitada, la misma cuenta con:

- 3 nodos emisores (2 utilizando TCP 1 UDP),
- 3 nodos receptores (2 utilizando TCP 1 UDP),
- 2 routers intermedios.

Definimos que todos los enlaces tengan un data-rate de 50Mbps, exceptuando a las interfaces que conectan a los 2 routers ("Interfaz 0" en la Figura 1), para estas, decidimos que posean una velocidad de conexión de 50Kbps.

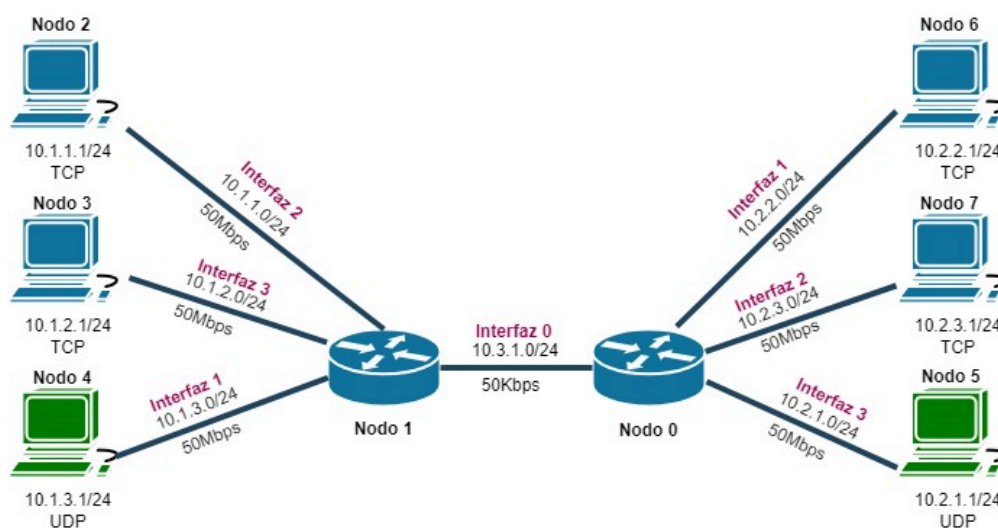


Figura 1: Topología implementada, con las IPs, Interfaces, protocolos y velocidades correspondientes a cada nodo y enlace

Al definir los data-rate mencionados, se busca generar un cuello de botella (bottleneck) en la red, y, por ende, provocar una congestión.

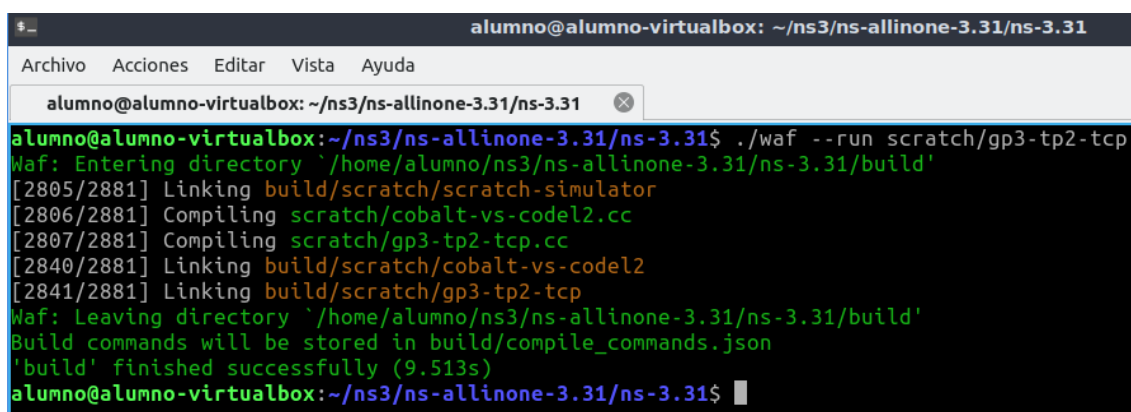
Primera parte: TCP - Código

Cómo ejecutar el código:

Para lograr correr de manera efectiva los archivos adjuntados “gp3-tp2-tcp.cc” y “gp3-tp2-tcp-udp.cc” que poseen código en C++ y que estos produzcan los outputs correspondientes, en primer lugar, se debe de copiar dichos archivos en la ruta “/home/alumno/ns3/ns3-allinone-3.31/ns3-3.31/scratch”.

En segundo lugar, en una terminal, debe situarse en una carpeta anterior a la de “scratch” (“/home/alumno/ns3/ns3-allinone-3.31/ns3-3.31/”) y por consiguiente utilizar el comando “./waf - -run scratch/<nombre_archivo.cc>”.

De haber conseguido un resultado exitoso, se obtendrán una serie de mensajes por terminal similar a los que se muestra en la Figura 2 y se crearán los archivos que se muestran en la Figura 2.1.



```
alumno@alumno-virtualbox: ~/ns3/ns-allinone-3.31/ns-3.31
Archivo Acciones Editar Vista Ayuda
alumno@alumno-virtualbox: ~/ns3/ns-allinone-3.31/ns-3.31
alumno@alumno-virtualbox:~/ns3/ns-allinone-3.31/ns-3.31$ ./waf --run scratch/gp3-tp2-tcp
Waf: Entering directory `/home/alumno/ns3/ns-allinone-3.31/ns-3.31/build'
[2805/2881] Linking build/scratch/scratch-simulator
[2806/2881] Compiling scratch/cobalt-vs-codel2.cc
[2807/2881] Compiling scratch/gp3-tp2-tcp.cc
[2840/2881] Linking build/scratch/cobalt-vs-codel2
[2841/2881] Linking build/scratch/gp3-tp2-tcp
Waf: Leaving directory `/home/alumno/ns3/ns-allinone-3.31/ns-3.31/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (9.513s)
alumno@alumno-virtualbox:~/ns3/ns-allinone-3.31/ns-3.31$
```

Figura 2: Ejecución del comando “waf”

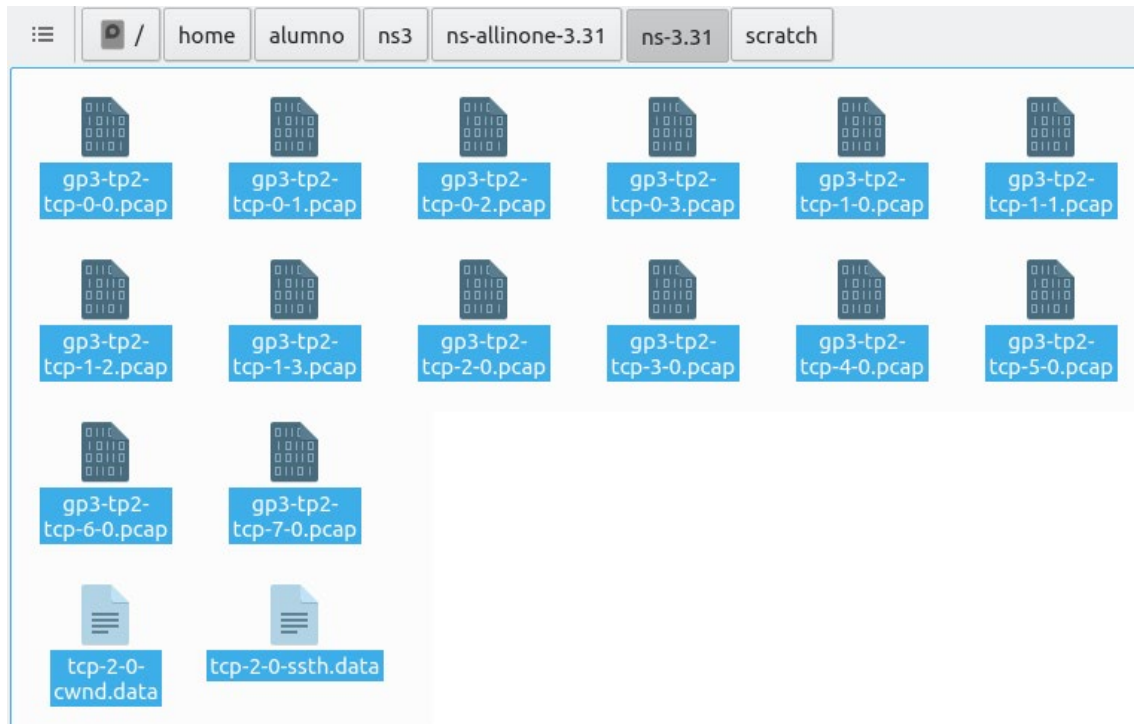


Figura 2.1: Creación de los archivos obtenidos al correr el código C++

Análisis del código

Luego de haber definido la topología mostrada, continuaremos realizando el código del script de NS3 para así comenzar la simulación de los 4 nodos TCP (los 2 emisores, y los 2 receptores).

```

190 //Define que el algoritmo a usar es TCP New Reno;
191 Config::SetDefault("ns3::TcpL4Protocol::SocketType",TypeIdValue(TcpNewReno::GetTypeId()));
192
193
194 //Genero las conexiones punto a punto
195 PointToPointHelper bottleneck;
196 bottleneck.SetDeviceAttribute("DataRate",StringValue("50KBps"));
197 bottleneck.SetChannelAttribute("Delay",StringValue("100ms"));
198 bottleneck.SetQueue("ns3::DropTailQueue","MaxSize",StringValue("10p"));
199
200
201 PointToPointHelper pointToPoint;
202 pointToPoint.SetDeviceAttribute("DataRate",StringValue("50MBps"));
203 pointToPoint.SetChannelAttribute("Delay",StringValue("1ms"));
204 pointToPoint.SetQueue("ns3::DropTailQueue","MaxSize",StringValue("10p"));
205
206 PointToPointDumbbellHelper dumbbellNetwork(3,pointToPoint,3,pointToPoint,bottleneck);
207

```

Figura 3: Configuración de los PointToPoint Helpers

Por consiguiente, en el main del programa, comenzamos definiendo el algoritmo a utilizar, que, en nuestro caso, es TCP New Reno. A su vez, definimos 2 PointToPointHelper³:

- El primero, denominado “bottleneck” (ya que es el canal compartido y en donde se generará la congestión) el cual establece la configuración de los 2 routers. A este lo configuramos con velocidad de conexión de 50KBps, un delay de 100ms y una Drop Tail Queue (una cola FIFO que descarta paquetes en caso de desbordamiento) que tiene un tamaño máximo de 10 paquetes.

- El segundo, “pointToPoint” serán los enlaces que se conectarán de los routers a los nodos emisores y receptores. La configuración es similar al PointToPointHelper anterior, con la salvedad de que, en este caso, el data-rate es de 50Mbps y el delay de 1ms.

³ clase que se utiliza para ayudar en la configuración y creación de enlaces punto a punto en la simulación de redes.

También, en esta instancia hacemos uso de un [PointToPointDumbbellHelper](#) que nos facilita la creación de la topología definida en el apartado anterior. Gracias a este Helper, solo debemos colocar: la cantidad de nodos a la izquierda, sus conexiones puntos a puntos, cantidad de nodos a la derecha, sus conexiones y, finalmente, la conexión de los 2 routers.

```

208 //Se instala la pila de protocolos TCP/IPv4
209 InternetStackHelper stack;
210 dumbbellNetwork.InstallStack(stack);
211
212 //Defino las IPs bases para los nodos
213 Ipv4AddressHelper leftIP("10.1.1.0", "255.255.255.0");
214 Ipv4AddressHelper rightIP("10.2.1.0", "255.255.255.0");
215 Ipv4AddressHelper routersIP("10.3.1.0", "255.255.255.0");
216
217 dumbbellNetwork.AssignIpv4Addresses(leftIP,rightIP,routersIP);
218
219 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

```

Figura 4: Instalación de protocolos TCP/IPv4 y asignación de direcciones IPs

En este fragmento, utilizamos un [InternetStackHelper](#) que nos ayuda a instalar las funcionalidades TCP/IP/UDP a nuestra topología dumbbell. Posteriormente, definimos las direcciones IPs que usarán tanto los nodos izquierdos, derechos y los routers centrales, y se las asignamos a la red. Por último, en la línea 219, se inicializa la tabla de ruteo de los nodos en la simulación.

```

182 //Se crean las aplicaciones OnOff TCP
183 //-----
184 OnOffHelper clientHelper("ns3::TcpSocketFactory",InetSocketAddress(dumbbellNetwork.GetRightIpv4Address(nodo1),port));
185 clientHelper.SetAttribute("OnTime",StringValue("ns3::ConstantRandomVariable[Constant=1000.0]"));
186 clientHelper.SetAttribute("OffTime",StringValue("ns3::ConstantRandomVariable[Constant=0]"));
187 clientHelper.SetAttribute("MaxBytes",UIntegerValue(data_bytes*10000));
188
189 clientApps.Add(clientHelper.Install(dumbbellNetwork.GetLeft(nodo0)));
190
191 PacketSinkHelper server("ns3::TcpSocketFactory",InetSocketAddress(dumbbellNetwork.GetRightIpv4Address(nodo1),port));
192 serverApps.Add(server.Install(dumbbellNetwork.GetRight(nodo1)));
193
194 //-----
195 OnOffHelper clientHelper2("ns3::TcpSocketFactory",InetSocketAddress(dumbbellNetwork.GetRightIpv4Address(nodo2),port));
196 clientHelper2.SetAttribute("OnTime",StringValue("ns3::ConstantRandomVariable[Constant=1000.0]"));
197 clientHelper2.SetAttribute("OffTime",StringValue("ns3::ConstantRandomVariable[Constant=0]"));
198 clientHelper2.SetAttribute("MaxBytes",UIntegerValue(data_bytes*10000));
199
200 clientApps.Add(clientHelper2.Install(dumbbellNetwork.GetLeft(1)));
201
202 PacketSinkHelper server2("ns3::TcpSocketFactory",InetSocketAddress(dumbbellNetwork.GetRightIpv4Address(nodo2),port));
203 serverApps.Add(server2.Install(dumbbellNetwork.GetRight(nodo2)));
204
205 //-----

```

Figura 5: Configuración de las aplicaciones OnOff TCP

Como se observa en la Figura 5, se utiliza la clase [OnOffHelper](#) para configurar y gestionar las aplicaciones de tráfico de tipo "On/Off" y, como se ve en la línea 184, se le asigna el protocolo TCP. Gracias a esta clase, se simplifica la configuración de las aplicaciones. En cada aplicación se setean los atributos:

- **"OnTime"**: el cual hace referencia al tiempo durante el cual la aplicación de tráfico On/Off está activa, es decir, cuando está generando tráfico, en nuestro caso el valor de la constante es mil.
- **"OffTime"**: que refiere al tiempo durante el cual la aplicación de tráfico On/Off está inactiva, es decir, cuando no está generando tráfico, en nuestro caso el valor de la constante es 0.
- **"MaxBytes"**: que significa el número total de bytes a enviar (en caso de establecerse en 0, significa que no habrá límite). En el código, lo definimos como 4 Megabytes.

Además, hacemos uso de la clase [PacketSinkHelper](#), la misma se utiliza para ayudar en la configuración y creación de un "receptor de paquetes"⁴ en la simulación de redes.

```

212 if(tracing){
213     pointToPoint.EnablePcapAll ("gp3-tp2-tcp",true);
214
215     std::ofstream ascii;
216     Ptr<OutputStreamWrapper> ascii_wrap;
217     ascii.open ((prefix_file_name + "-ascii").c_str ());
218     ascii_wrap = new OutputStreamWrapper ((prefix_file_name + "-ascii").c_str (),std::ios::out);
219     stack.EnableAsciiIpv4All (ascii_wrap);
220
221     //Definimos que el tracing comience a partir del segundo 2 ya que, si comenzaba antes, todos los .data se generaban vacios;
222     Simulator::Schedule (Seconds (2.000001), &TraceCwnd2,cwnd_name_node_2 + "-cwnd.data");
223     Simulator::Schedule (Seconds (2.000001), &TraceSsthresh2, cwnd_name_node_2 + "-sssth.data");
224 }

```

Figura 6: Configuración los diversos tipos de outputs que generar la simulación

⁴ dispositivo que se encarga de recibir y procesar los paquetes que son enviados a través de la red simulada.

En la Figura 6, se puede apreciar que, al correr la simulación, ésta producirá diversos archivos para estudiar la red. En nuestro caso generamos archivos ".pcap" (para analizar el tráfico de la red con Wireshark), "ascii" y ".data".

Los ".pcap" que da como resultado NS3, respetan el siguiente formato: "gp3-tp2-tcp-nodo-interfaz.pcap", por ejemplo, el archivo "gp3-tp2-tcp-1-2.pcap" hace referencia al tráfico del nodo 1 y la interfaz 2.

Los archivos ".data" que produce nuestra simulación son 2, "tcp-2-0-cwnd.data" y "tcp-2-0-sssth.data". Estos, junto con el archivo "config_cwnd.plt", se combinarán y darán como resultado un gráfico en el que mostrará la CWND y el SSTH a través del tiempo de la simulación.

Análisis de la red

En esta sección, haremos un análisis del tráfico de la red simulada con NS3 con solo los nodos TCP emitiendo. Utilizaremos como base el archivo "gp3-tp2-tcp-0-0.pcap", el cual posee todos los paquetes que se emiten por la interfaz 0.

TCP - Handshake

Como ya se describió, en esta etapa de TCP es donde se comienzan a establecer las conexiones entre los nodos emisores (IPs: 10.1.1.1 y 10.1.2.1) y los nodos receptores (IPs: 10.2.2.1 y 10.2.3.1), como se muestran en la Figura 7.

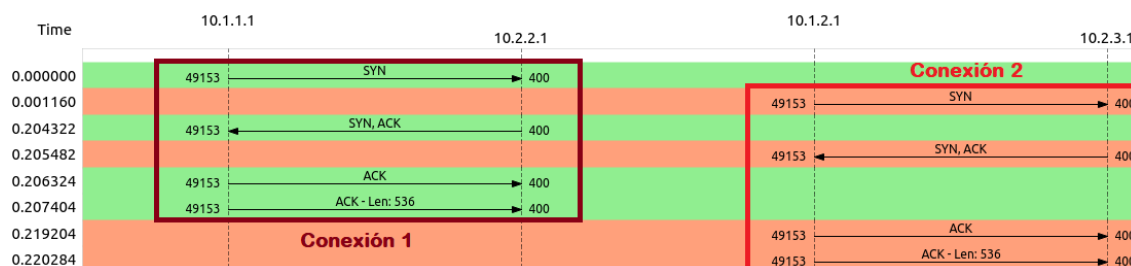


Figura 7: Etapa handshake del protocolo TCP

Dentro de esta etapa, además, se define la RWND⁵ (ventana de recepción) de los nodos en la red. Como muestra la Figura 8, el nodo emisor en el Three-Way Handshake establece su tamaño de ventana en 65535 bytes, pero, asimismo, hace uso del campo “Window scale” dentro del segmento TCP. Al utilizar este campo, lo que se quiere denotar es que: al valor que se encuentra en el campo “Window size value”, se lo debe multiplicar por el “Window scale” y, de esta forma, obtener el verdadero valor del tamaño de ventana de recepción.

⁵ buffer del lado del emisor y receptor que retiene temporalmente los datos (bytes) que llegan.

¿Por qué se usa este mecanismo para conseguir la RWND?

Porque el campo “Window size value” ocupa 2 bytes en el segmento, por lo que el mayor valor posible es 65535.

La ventana de recepción no es un valor fijo sino que puede ir cambiando a lo largo del tiempo, ya que es parte del control de flujo en TCP. Esto último se puede ver reflejado en la Figura 9, en donde el mismo nodo ajusta su RWND a 131072 bytes.

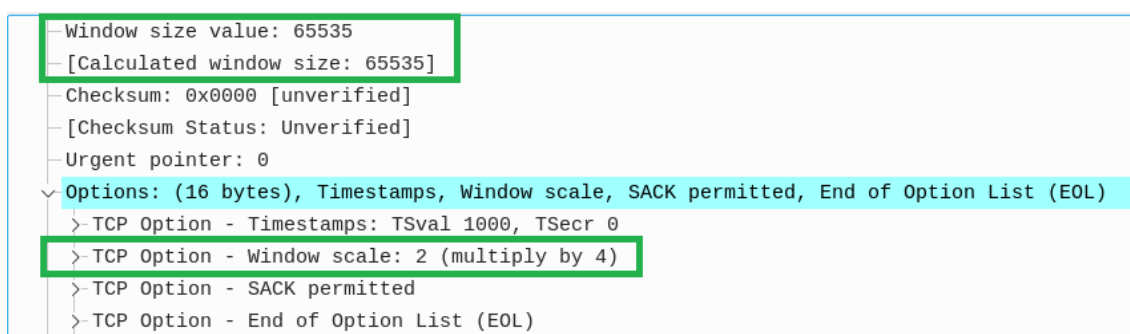


Figura 8: Ventana de recepción del nodo emisor N°2 (10.1.1.1) dentro del paquete SYN

```

> Flags: 0x010 (ACK)
- Window size value: 32768
- [Calculated window size: 131072]
- [Window size scaling factor: 4]
- Checksum: 0x0000 [unverified]
- [Checksum Status: Unverified]
- Urgent pointer: 0
> Options: (12 bytes), Timestamps, End of Option List (EOL)
> [SEQ/ACK analysis]
> [Timestamps]

```

Figura 9: Ventana de recepción del nodo emisor N°2 (10.1.1.1) dentro del primer paquete que emite

Congestión de la red

En este apartado, analizaremos la congestión de la red que provocamos al realizar el cuello de botella ya explicado. Analizaremos el archivo “gp3-tp2-tcp-0-0.pcap” en donde se reflejan todos los paquetes enviados y recibidos de los 4 nodos TCP.

Primeramente, vamos a analizar la Figura 11 que muestra el gráfico de la ventana de congestión (CWND) y el SSTH de la red a lo largo del tiempo. Además, indica cada etapa y algoritmo del Control de congestión.

Dicho gráfico lo generamos con la herramienta Gnuplot utilizando el comando “gnuplot config_cwnd.plt”, el cual utiliza un archivo “.plt” que se encarga de combinar los archivos “tcp-2-0-cwnd.data” y “tcp-2-0-ssth.data” para generar un “.jpg” con el gráfico.

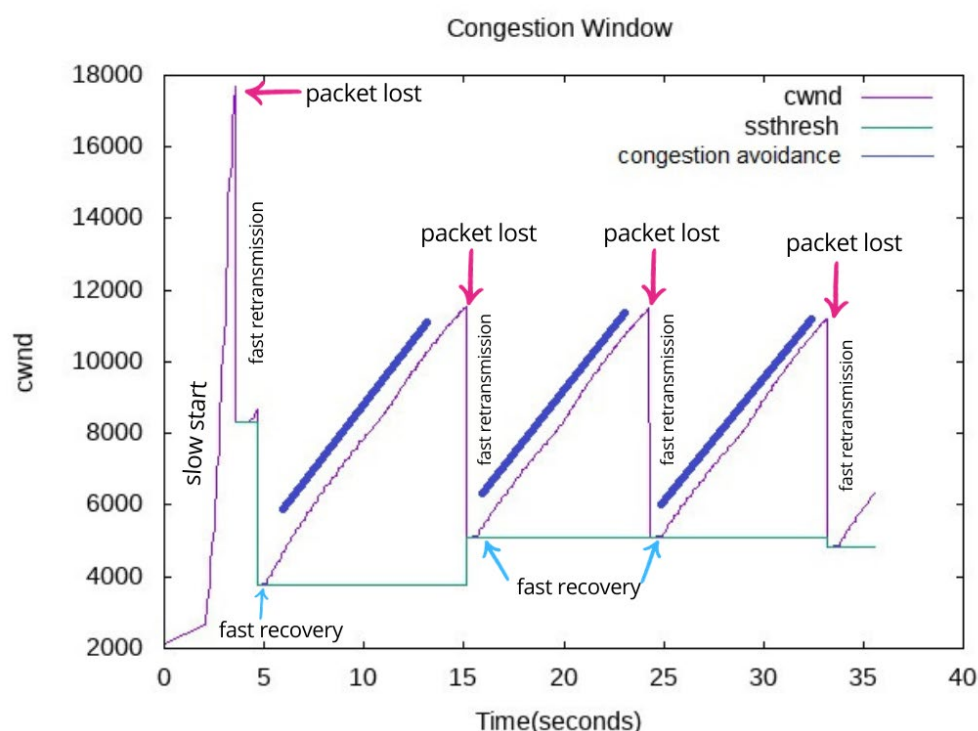


Figura 10: Gráfico de la ventana de congestión a través del tiempo del nodo 10.1.1.1 y 10.2.2.1, generado con gnuplot

Observando el gráfico y con ayuda de Wireshark, podemos observar que la primera congestión arranca alrededor del segundo 2.509, como se muestra en la Figura 11, ya que comienzan a enviarse paquetes del tipo “TCP Dup ACK”, los mismos indican que el receptor ha recibido un segmento TCP duplicado y del tipo “TCP Previous segment not captured”, lo que significa que Wireshark está observando un ACK de un paquete que no ha capturado.

No.	Time	Source	Destination	Protocol	TCP Segment Len	Bytes in flight	Calculated window size	Info
194	2.506733	10.1.1.1	10.2.2.1	TCP	536			131072 49153 → 400 [ACK] Seq=44489
195	2.509546	10.2.3.1	10.1.2.1	TCP	0			131072 [TCP Dup ACK 167#8]
196	2.518533	10.1.2.1	10.2.3.1	TCP	536	9648		131072 49153 → 400 [ACK] Seq=44489
197	2.521346	10.2.3.1	10.1.2.1	TCP	0			131072 [TCP Dup ACK 167#9]
198	2.530333	10.1.2.1	10.2.3.1	TCP	536			131072 [TCP Previous segment not captured]
199	2.532986	10.2.2.1	10.1.1.1	TCP	0			131072 400 → 49153 [ACK] Seq=44489
200	2.542133	10.1.2.1	10.2.3.1	TCP	536			131072 49153 → 400 [ACK] Seq=44489
201	2.544786	10.2.2.1	10.1.1.1	TCP	0			131072 [TCP Dup ACK 199#1]
202	2.553933	10.1.1.1	10.2.2.1	TCP	536			131072 [TCP Previous segment not captured]
203	2.556586	10.2.2.1	10.1.1.1	TCP	0			131072 [TCP Dup ACK 199#2]
204	2.565733	10.1.1.1	10.2.2.1	TCP	536	8040		131072 49153 → 400 [ACK] Seq=44489
205	2.568546	10.2.3.1	10.1.2.1	TCP	0			131072 [TCP Dup ACK 167#10]
206	2.577533	10.1.1.1	10.2.2.1	TCP	536			131072 [TCP Previous segment not captured]
207	2.580346	10.2.3.1	10.1.2.1	TCP	0			131072 [TCP Dup ACK 167#11]
208	2.589333	10.1.2.1	10.2.3.1	TCP	536	11792		131072 49153 → 400 [ACK] Seq=44489
209	2.591986	10.2.2.1	10.1.1.1	TCP	0			131072 [TCP Dup ACK 199#3]
210	2.601133	10.1.2.1	10.2.3.1	TCP	536	12328		131072 49153 → 400 [ACK] Seq=44489
211	2.603786	10.2.2.1	10.1.1.1	TCP	0			131072 [TCP Dup ACK 199#4]
212	2.612933	10.1.2.1	10.2.3.1	TCP	536	12864		131072 49153 → 400 [ACK] Seq=44489
213	2.615746	10.2.3.1	10.1.2.1	TCP	0			131072 [TCP Dup ACK 167#12]
214	2.624733	10.1.1.1	10.2.2.1	TCP	536			131072 [TCP Previous segment not captured]

Figura 11: Comienzo del packet loss de la red en el archivo gp3-tp2-tcp-0-0.pcap

Luego, en torno al segundo 2.7, que es cuando la CWN baja abruptamente, es el momento en el cual un emisor TCP (10.1.1.1) emite un TCP Fast Retransmission, luego de llegarle dos Dup ACKs. Consideramos que esto es una “anomalía” ya que en la teoría se habla de que el emisor envía el Fast Retransmission justo después del Dup ACK #3, pero aquí lo envía justo después de recibir el Dup ACK #2.

No.	Time	Source	Destination	Protocol	TCP Segment Len	Bytes in flight	Calculated window size	Info
2405039		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=44489 Ack=1 Win=400
2405050		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=45025 Ack=1 Win=400
2405062		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=45561 Ack=1 Win=400
2426626		10.1.1.1	10.2.2.1	TCP	536			400 → 49153 [ACK] Seq=1 Ack=31089 Win=400
2428639		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=46097 Ack=1 Win=400
2428650		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=46633 Ack=1 Win=400
2428662		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=47169 Ack=1 Win=400
2473826		10.1.1.1	10.2.2.1	TCP	536			400 → 49153 [ACK] Seq=1 Ack=32161 Win=400
2475839		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=47705 Ack=1 Win=400
2475850		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=48241 Ack=1 Win=400
2475862		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=48777 Ack=1 Win=400
2532986		10.1.1.1	10.2.2.1	TCP	536			400 → 49153 [ACK] Seq=1 Ack=32697 Win=400
2534999		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=49313 Ack=1 Win=400
2535011		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=49849 Ack=1 Win=400
2535023		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=50385 Ack=1 Win=400
2544786		10.1.1.1	10.2.2.1	TCP	536			[TCP Dup ACK 127#1] 400 → 49153 [ACK] Seq=44489
2546799		10.1.1.1	10.2.2.1	TCP	536			49153 → 400 [ACK] Seq=50921 Ack=1 Win=400
2556586		10.1.1.1	10.2.2.1	TCP	536			[TCP Dup ACK 127#2] 400 → 49153 [ACK] Seq=44489
2558599		10.1.1.1	10.2.2.1	TCP	536			49153 [TCP Fast Retransmission] 49153 → 400 [ACK] Seq=44489

Figura 12: Emisor 10.1.1.1 envía un segmento TCP Fast Retransmission después del Dup ACK #2 en el archivo gp3-tp2-0-1.pcap

Un caso en el que se cumple lo que menciona la teoría es en la Figura 13, que ocurre en el segundo 3.6913, momento en el que la ventana de congestión vuelve a bajar de manera abrupta.

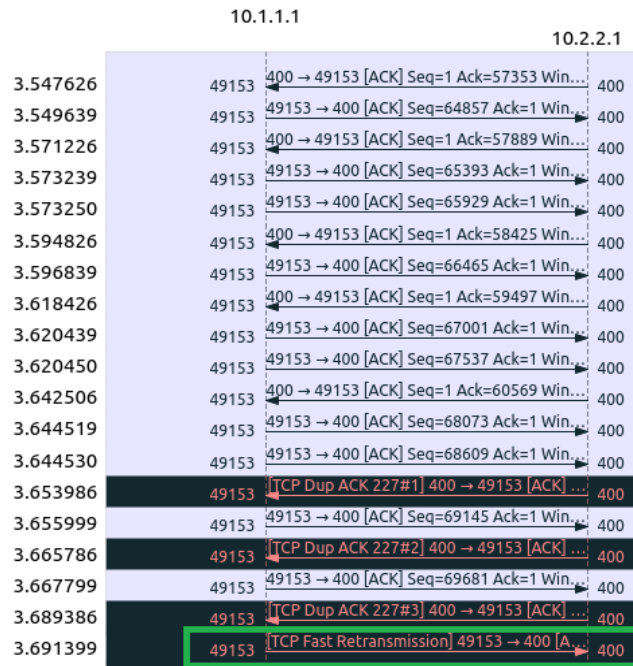


Figura 13: Emisor 10.1.1.1 envía un segmento TCP Fast Retransmission después del tercer Dup ACK recibido en el archivo gp3-tp2-0-1.pcap

En las Figuras 14,15 y 16 se puede ver los momentos de Fast Retransmission que muestra el gráfico de la Figura 10, los cuales ocurren en los segundos 14.124, 23.293 y 32.225, respectivamente.

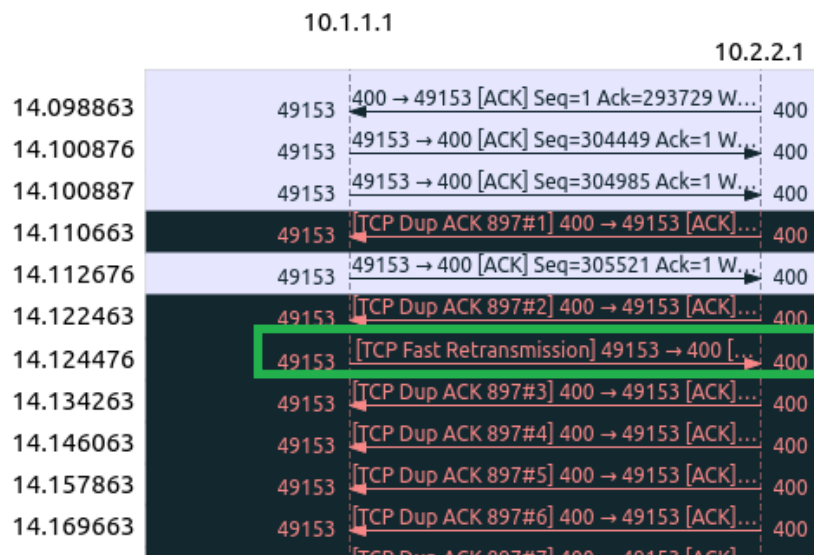


Figura 14: Emisor 10.1.1.1 envía un segmento TCP Fast Retransmission después del segundo Dup ACK recibido en el archivo gp3-tp2-0-1.pcap

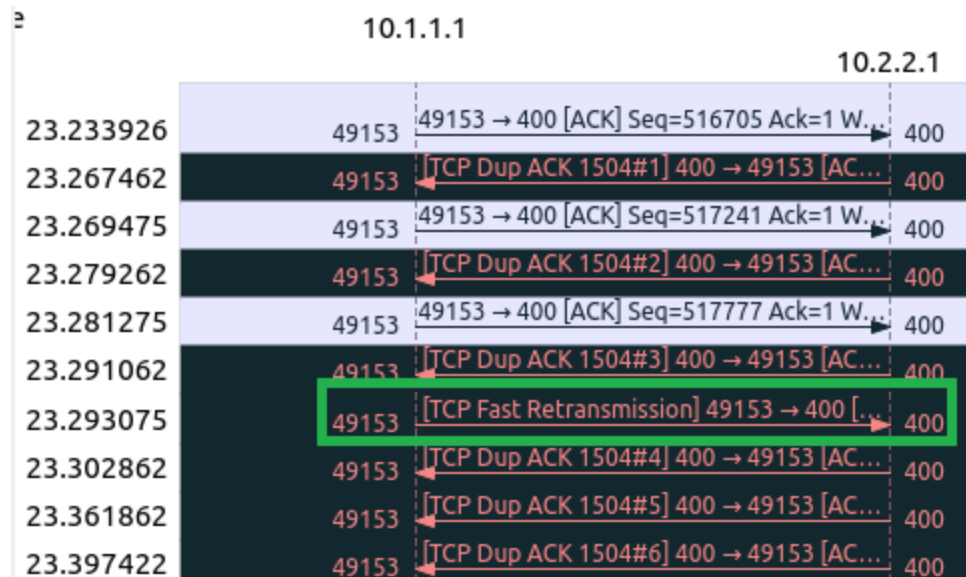


Figura 15: Emisor 10.1.1.1 envía un segmento TCP Fast Retransmission después del tercer Dup ACK recibido en el archivo gp3-tp2-0-1.pcap

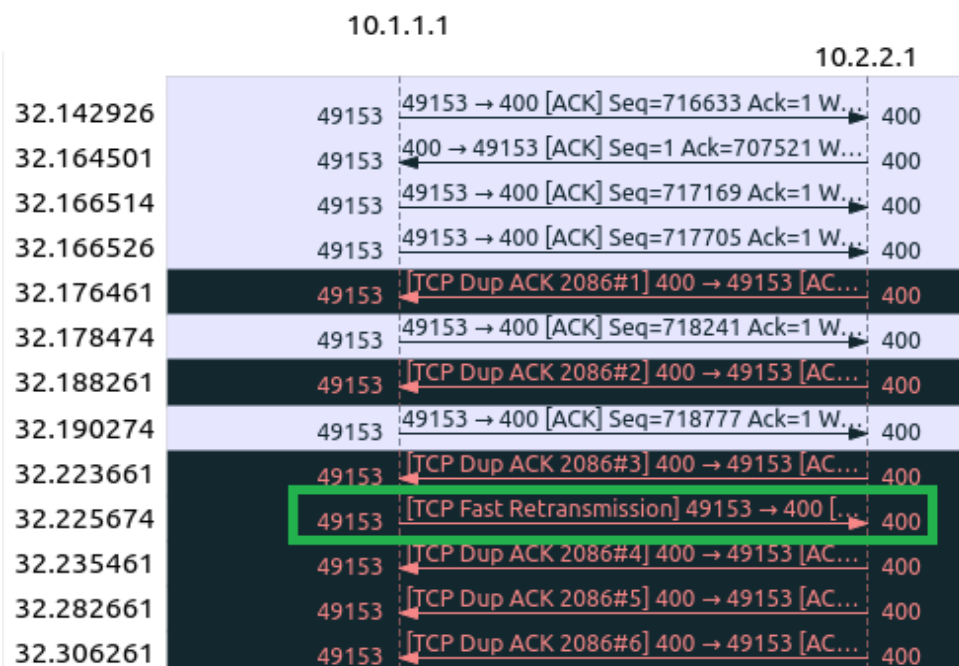


Figura 16: Emisor 10.1.1.1 envía un segmento TCP Fast Retransmission después del tercer Dup ACK recibido en el archivo gp3-tp2-0-1.pcap

El cuello de botella que planteamos también lo podemos ver reflejado en el gráfico que genera Wireshark llamado “Window scaling”. En este gráfico en el eje Y, se muestran los Bytes in flight⁶ a lo largo del tiempo, por lo que, si vemos la Figura 17 y 18, los picos que alcanza el tamaño de los Bytes in flight ocurren en momentos aledaños a las bajadas abruptas de la CWND.

⁶ cantidad de bytes que han sido enviados por el emisor, pero aún no han sido confirmados por el receptor

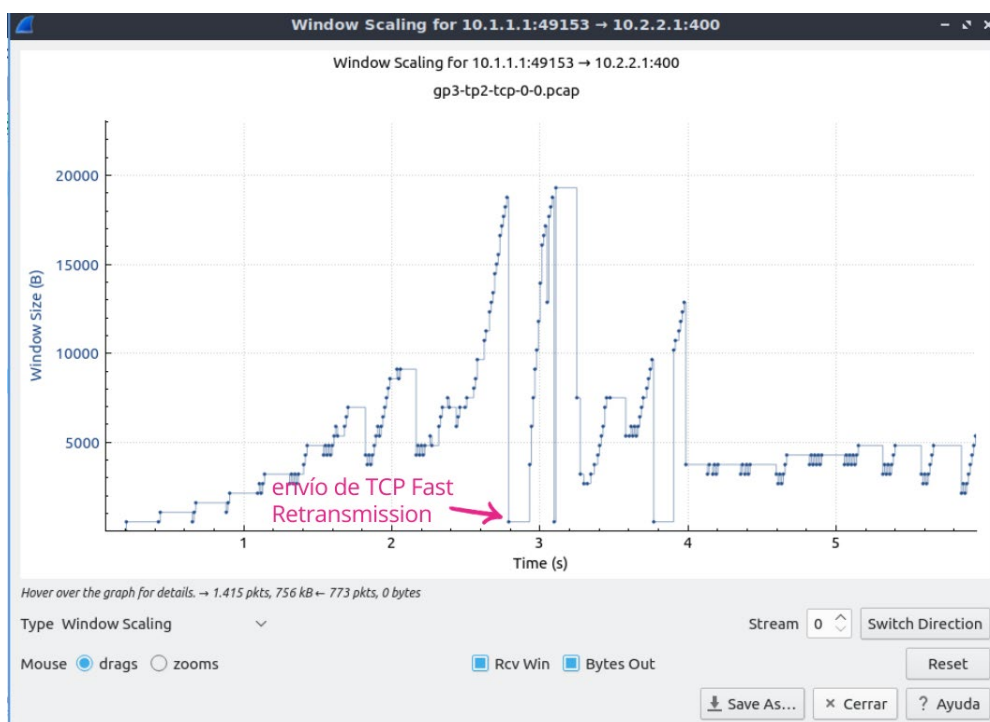


Figura 17: Gráfico Window scaling de Wireshark en el archivo gp3-tp2-0-0.pcap

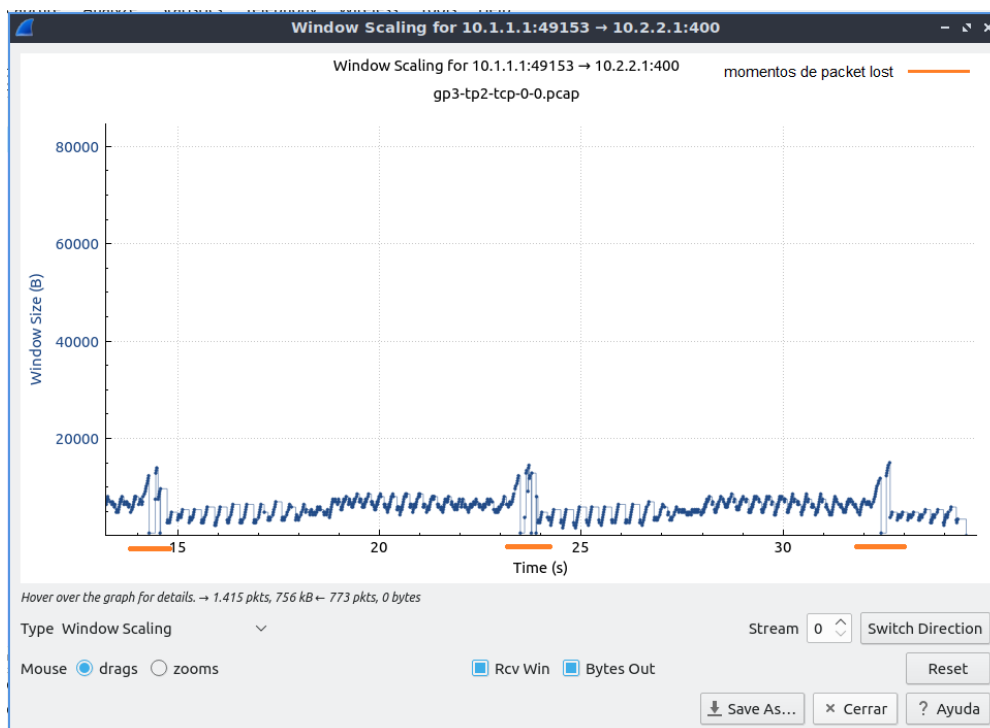


Figura 18: Figura 17: Gráfico Window scaling de Wireshark indicando las zonas de congestión, en el archivo gp3-tp2-0-0.pcap

Además, este gráfico nos sirve para ver que la cola de recepción (RWND), siempre se mantuvo constante a lo largo de la simulación, específicamente en 131070 bytes.



Figura 19: Figura 17: Gráfico Window scaling de Wireshark la ventana de recepción, en el archivo gp3-tp2-0-0.pcap

A pesar de los momentos de congestión mostrados, utilizamos el filtro “not tcp.analysis.flags” para recolectar los paquetes que fueron enviados y recepcionados correctamente. Por lo que podemos ver en la Figura 20, estos representan un 93.9%.

No.	Time	Source	Destination	Protocol	TCP Segment Len	Bytes in flight	Calculated window size	Info
1	0.000000	10.1.1.1	10.2.2.1	TCP	0			65535 49153 → 400 [SYN] Seq=
2	0.001160	10.1.2.1	10.2.3.1	TCP	0			65535 49153 → 400 [SYN] Seq=
3	0.204322	10.2.2.1	10.1.1.1	TCP	0			65535 400 → 49153 [SYN, AC
4	0.205482	10.2.3.1	10.1.2.1	TCP	0			65535 400 → 49153 [SYN, AC
5	0.206324	10.1.1.1	10.2.2.1	TCP	0			131072 49153 → 400 [ACK] Seq=
6	0.207404	10.1.1.1	10.2.2.1	TCP	536	536		131072 49153 → 400 [ACK] Seq=
7	0.219204	10.1.2.1	10.2.3.1	TCP	0			131072 49153 → 400 [ACK] Seq=
8	0.220284	10.1.2.1	10.2.3.1	TCP	536	536		131072 49153 → 400 [ACK] Seq=
9	0.422297	10.2.2.1	10.1.1.1	TCP	0			131072 400 → 49153 [ACK] Seq=
10	0.424310	10.1.1.1	10.2.2.1	TCP	536	536		131072 49153 → 400 [ACK] Seq=
11	0.435177	10.2.3.1	10.1.2.1	TCP	0			131072 400 → 49153 [ACK] Seq=
12	0.436110	10.1.1.1	10.2.2.1	TCP	536	1072		131072 49153 → 400 [ACK] Seq=
13	0.447910	10.1.2.1	10.2.3.1	TCP	536	536		131072 49153 → 400 [ACK] Seq=
14	0.459710	10.1.2.1	10.2.3.1	TCP	536	1072		131072 49153 → 400 [ACK] Seq=
15	0.651003	10.2.2.1	10.1.1.1	TCP	0			131072 400 → 49153 [ACK] Seq=
16	0.653016	10.1.1.1	10.2.2.1	TCP	536	536		131072 49153 → 400 [ACK] Seq=
17	0.664816	10.1.1.1	10.2.2.1	TCP	536	1072		131072 49153 → 400 [ACK] Seq=
18	0.674603	10.2.3.1	10.1.2.1	TCP	0			131072 400 → 49153 [ACK] Seq=
19	0.676616	10.1.1.1	10.2.2.1	TCP	536	1608		131072 49153 → 400 [ACK] Seq=
20	0.688416	10.1.2.1	10.2.3.1	TCP	536	536		131072 49153 → 400 [ACK] Seq=
21	0.700216	10.1.2.1	10.2.3.1	TCP	536	1072		131072 49153 → 400 [ACK] Seq=

> Frame 140: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
 > Point-to-Point Protocol
 > Internet Protocol Version 4, Src: 10.2.2.1, Dst: 10.1.1.1
 > Transmission Control Protocol, Src Port: 400, Dst Port: 49153, Seq: 1, Ack: 22513, Len: 0

0000 00 21 45 00 00 34 00 16 00 00 3f 06 00 00 0a 02 -!E-4--?--...
 TCP Analysis Flags: Label

Packets: 4388 - Displayed: 4122 (93.9%) Profile: Default

Figura 20: Muestra de paquetes enviados y correctamente aceptados

Podemos utilizar otro gráfico que brinda Wireshark llamado “I/O Graphs” en el que muestra la cantidad de paquetes enviados a lo largo del tiempo y, también, los diversos errores TCP ocurridos a lo largo de la simulación. En este caso, aplicamos los filtros “tcp.analysis.duplicate.ack”, “tcp.analysis.duplicate.out_of_order” y “tcp.analysis.lost_segment” para ver los momentos en el que llegaban estos tipos de paquetes. Por lo que se puede observar en la Figura 21, en el segundo 2 es en donde llega una gran cantidad de los paquetes nombrados, y por ende, provoca una pérdida de los mismos.

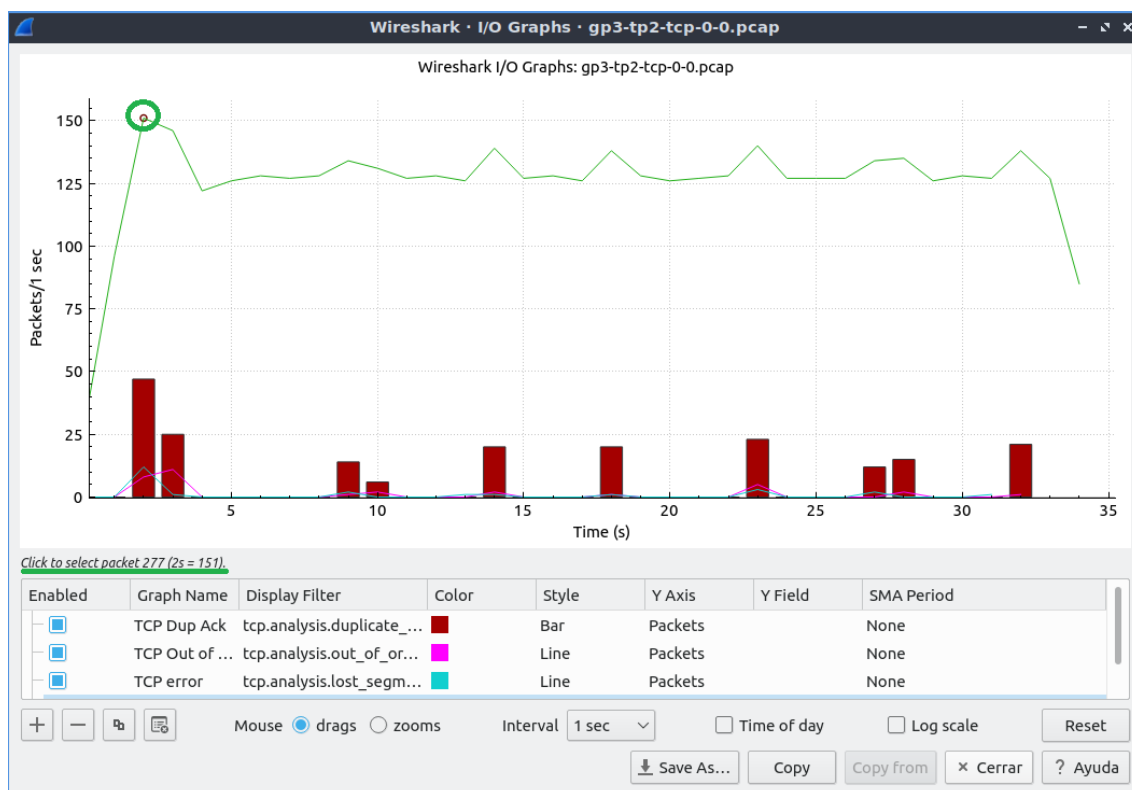


Figura 21: I/O Graph de Wireshark mostrando los paquetes por segundo y con filtros de TCP Error

Ancho de Banda

Para este trabajo también se propuso analizar qué sucedió con el ancho de banda durante una captura de transferencias de datos que hayamos realizado. En nuestro caso, vamos a tomar como ejemplo de muestra una captura de "conversaciones"⁷ realizadas entre dos nodos TCP con otros dos nodos TCP (es decir, dos conversaciones en simultáneo). Pero antes de entrar en análisis de gráficos y cálculos de velocidades, tenemos que definir y especificar algunos términos.

⁷Refiriendonos por "Conversación" a la conexión y comunicación establecida entre dos nodos.

Cuando hablamos de redes y análisis de tráfico, los términos "ancho de banda" y "velocidad de transferencia" a menudo se utilizan de manera intercambiable, pero en un sentido técnico, pueden tener connotaciones ligeramente diferentes. Veamos estas distinciones:

Ancho de Banda - Definición:

El término "ancho de banda" hace referencia a la capacidad máxima de transferencia de datos de un canal de comunicación. Es una medida teórica y se expresa generalmente en bits por segundo (bps) o en unidades más grandes como kilobits por segundo (Kbps), megabits por segundo (Mbps) o gigabits por segundo (Gbps). El ancho de banda indica la capacidad máxima de la conexión, pero no necesariamente la cantidad de datos que se están transfiriendo en un momento particular.

Velocidad de Transferencia:

La "velocidad de transferencia" o "tasa de transferencia" se refiere a la cantidad real de datos que se están transfiriendo en un momento particular. Se expresa generalmente en las mismas unidades que el ancho de banda (bps, Kbps, Mbps, etc.). La velocidad de transferencia puede variar en el tiempo y ser menor que el ancho de banda máximo debido a factores como la congestión de la red, la latencia, la pérdida de paquetes, etc.

Ya aclarada esta pequeña diferencia, ahora vamos a calcular tanto el ancho de banda promedio como la velocidad de transferencia de un momento en particular de la captura de tráfico de dos conversaciones TCP que habíamos especificado anteriormente. Si bien el que más nos interesa es este último, nos pareció interesante la idea de tener una medición que indique la velocidad de transferencia en un periodo de tiempo determinado.

Ancho de banda promedio

Para calcular esta medición vamos a tomar en cuenta los siguientes dos valores:

- Cantidad total de datos transmitidos (durante las conversaciones)
- Duración máxima de la transmisión

La idea es dividir esta cantidad de datos totales sobre la cantidad de tiempo que haya durado la transmisión y de esta forma obtener el promedio del ancho de banda utilizado durante la simulación.

Para obtener estos valores nos apoyamos en la información que obtenemos de la herramienta Wireshark en su ventana "Conversation" del menú de "Statistics".

Wireshark · Conversations · gp3-tp2-tcp-0-0.pcap

Ethernet	IPv4 - 2	IPv6	TCP - 2	UDP	Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.1.1.1	49153	10.2.2.1	400	2.188	876 k	1.415	832 k	773	43 k	0.000000	34.5443	192 k	10 k					
10.1.2.1	49153	10.2.3.1	400	2.200	884 k	1.430	841 k	770	42 k	0.001160	34.7646	193 k	9.826					

Figura 22: Ventana de conversaciones. Se indica: Bytes totales transmitidos y duración máxima de la transmisión en el canal.

Como se puede ver en la figura anterior, la cantidad de bytes totales de las conversaciones es 1760 KBytes (876 + 884), y la duración máxima es de 34,7646 seg.

Aplicando la fórmula mencionada anteriormente obtenemos:

$$\begin{aligned}
 \text{Ancho de banda} &= \text{Cantidad total datos transferidos} / \text{Duración de la conversación} \\
 &= 876 \text{ KB} + 884 \text{ KB} / 34,7646 \text{ seg} \\
 &= 1760 \text{ KB} / 34,7646 \text{ seg} \\
 &= 50,6262 \text{ KB/s}
 \end{aligned}$$

Finalmente, podemos concluir que el ancho de banda promedio de esta conversación es de aproximadamente 50 KB/s, lo mismo que se definió como cuello de botella teórico entre nodos que se había propuesto para esta simulación (Figura 1).

Velocidad de transferencia

Ahora nos interesa analizar un momento en particular. Y, para agregar valor a nuestra simulación, vamos a tomar el instante cuando se alcanza el pico máximo de transferencia justo antes de la bajada abrupta ocasionada por la primera pérdida de paquetes en una de las conversaciones.

Para esto nos vamos a apoyar en los datos obtenidos de uno de los gráficos que ofrece Wireshark. Los filtros que vamos a utilizar sobre las IPs nos permiten ver la cantidad de bits por segundo que se transfieren en cada conversación.

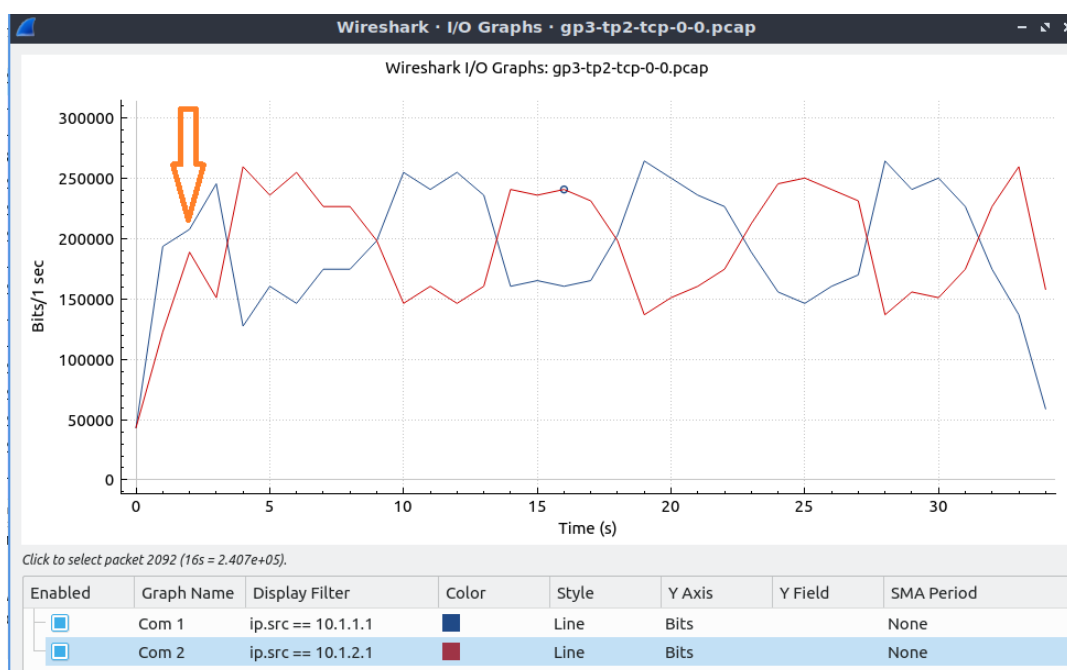


Figura 23: Gráfico que muestra la cantidad de bits transferidos por segundo de cada conversación. Se indica el momento que vamos a analizar.

Antes de seguir con el cálculo para medir la velocidad de transferencia analicemos un poco la figura 23.

Si bien vamos a medir sobre el instante indicado en la figura anterior, notemos que a partir de dicho momento todo el gráfico tiene un comportamiento simétricamente opuesto. Es decir, cuando una de las conversaciones “baja” en cantidad de bits por segundo, la otra “sube”. Esto se debe a que al límite de cuello de botella que se estableció para esta simulación y las diferentes etapas de control de congestión del protocolo TCP.

Notemos que el único momento donde ambas conversaciones no son simétricamente opuestas es al principio, cuando ambas están en etapa de “Slow Start”. Luego llegan a un punto lógico y esperable donde deberían alcanzar un límite de congestión.

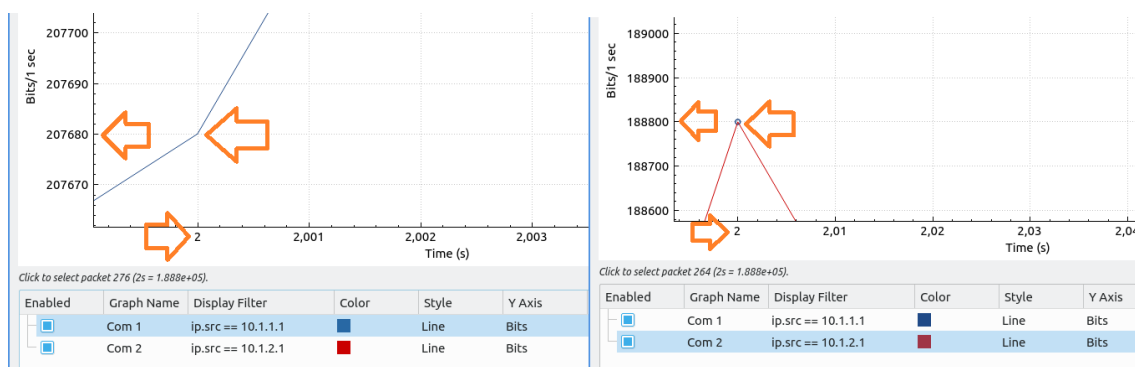


Figura 24: Zoom de la figura 23. Se indican los valores a tener en cuenta para analizar la velocidad de transferencia.

Como se puede ver en la figura 24, en el segundo 2 una de las dos conversaciones se ve afectada por pérdida de paquetes. Analicemos que sucede en este instante de tiempo.

- La conversación “Com 1” tiene una tasa de transferencia de 207080 bits por segundo.
- La conversación “Com 2” tiene una tasa de transferencia de 188800 bits por segundo.

Sumando ambas velocidades obtenemos una velocidad total de 395880 bits/s, que transformado a KBytes es igual a 48,3251 Bytes/s.

Como era de esperar, cuando se alcanza el punto de congestión establecido para la simulación, las velocidades de transferencia de las conversaciones se ven afectadas. En este caso, “Com 2” ve disminuida su velocidad y “Com 1” crece de manera simétricamente opuestas, es decir, mantiene el límite de 50KB/s establecido. Este comportamiento se ve reflejado durante los aproximadamente 34 segundos que duran las transmisiones, por lo que podemos decir que el límite de velocidad nunca es superado.

Luego, las velocidades se mantienen oscilantes entre los 150000 y los 250000 bits por segundo (un total de 48 KB/s aprox), en donde asumimos que cada “bajada” representa un cambio de estado a “fast retransmission” y cada punto de subida un cambio a “fast recovery” y “congestion avoidance”.

TCP - Fin de Conexión

Muy similar a cuando se inicia una conexión, TCP utiliza una serie de mensajes del tipo avisos y confirmaciones para finalizar una comunicación (Four-way handshake). Al momento de terminar la sesión, el protocolo utiliza un mecanismo de verificación de 4 pasos. A continuación, lo analizaremos en nuestra simulación.

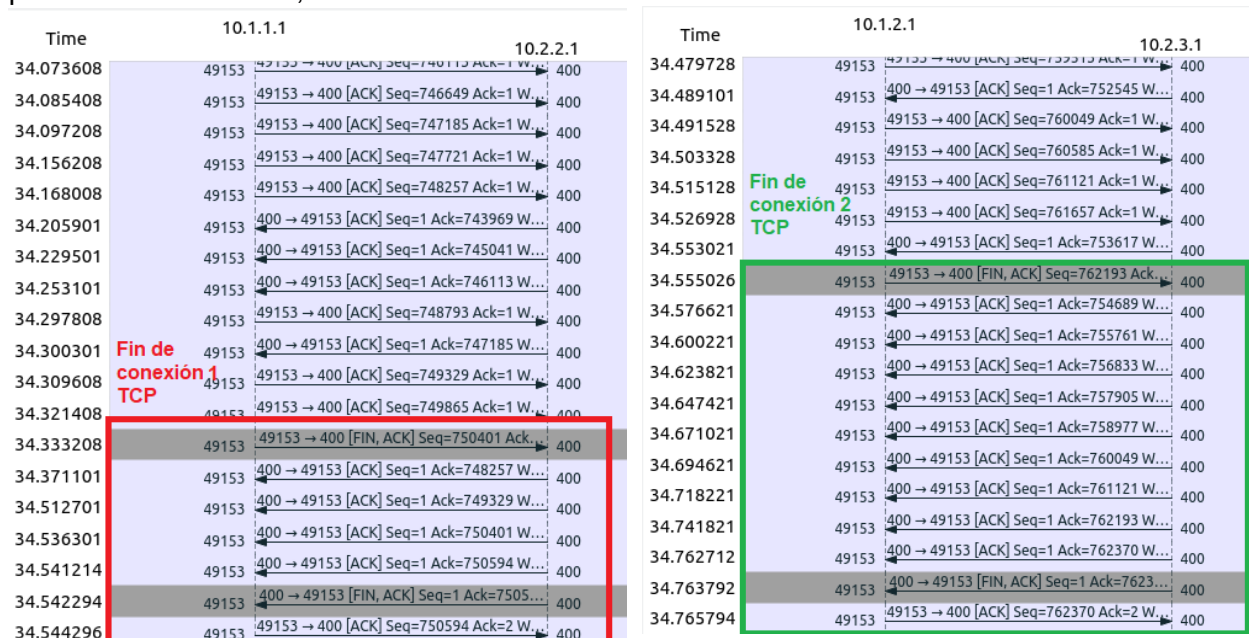


Figura 25: Flow Graph de Wireshark en el que se señala el momento del Fin de conexión entre los emisores y receptores.

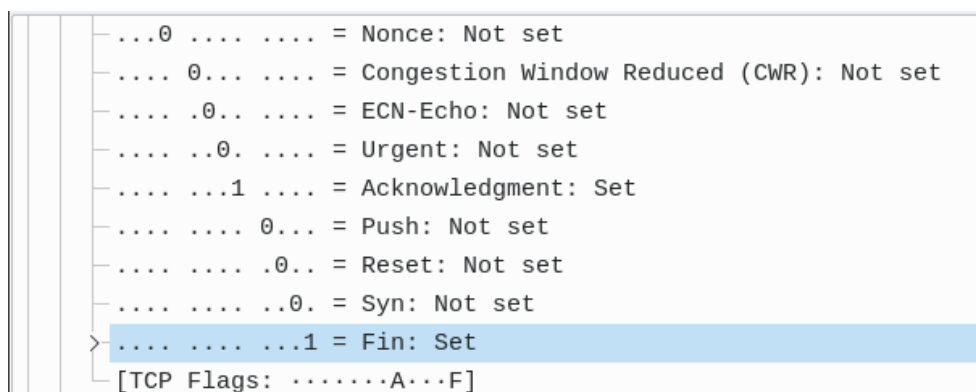


Figura 26: Segmento del nodo 10.1.1.1 (emisor TCP), el cual posee el bit FIN.

Como se observa en la Figura 25, el emisor envía un segmento el cual posee el bit FIN (Figura 26). Luego el receptor le responde con un ACK y, además, le envía su propio segmento con el bit FIN. Finalmente, el receptor culmina la conexión respondiendo con un ACK a este último segmento.

Segunda Parte: TCP y UDP

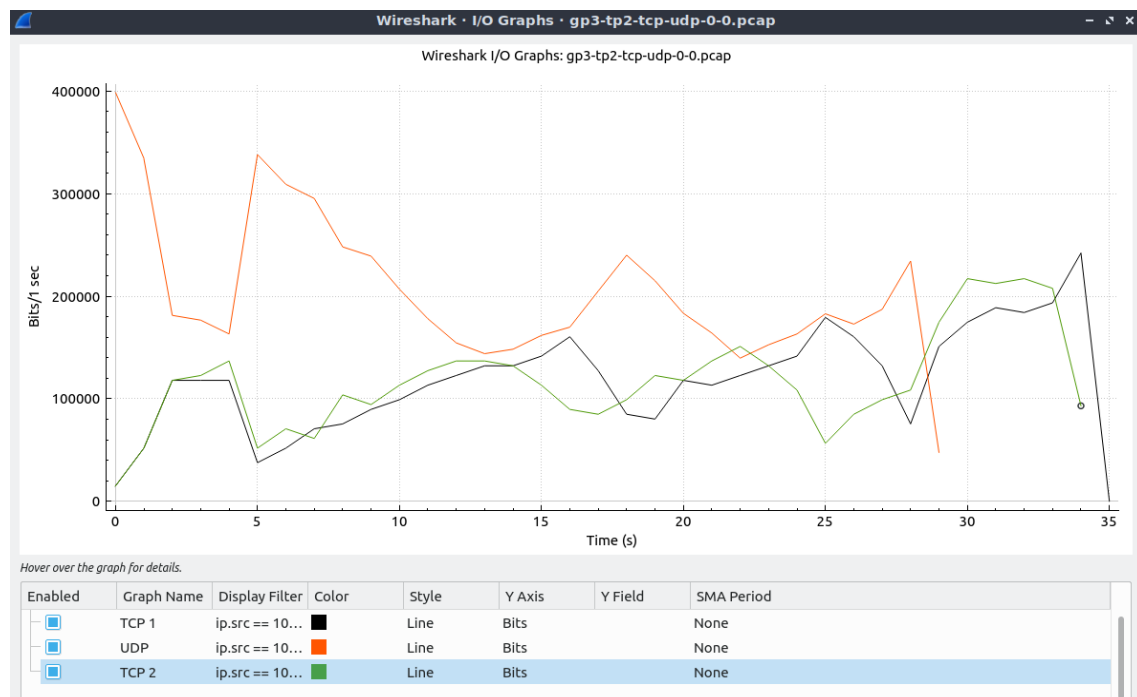


Figura 27: Gráfico que muestra la cantidad de bits transferidos por segundo tanto de los nodos TCP y el nodo UDP.

Ahora, además de los nodos TCP, el nodo UDP también está emitiendo paquetes. Como vimos al principio de este informe, el protocolo UDP tiene características distintas a TCP.

UDP al no estar orientado a conexión, comienza enviando una gran cantidad de paquetes a diferencia de TCP ya que primero necesita hacer Three Way Handshake antes de transmitir los datos.

De esta forma, en el segundo 5 podemos observar en el gráfico que cuando la ventana de congestión(cwnd) baja debido a la pérdida de paquetes, UDP aprovecha este momento para ocupar más ancho de banda y así enviar más paquetes. Sucede lo mismo en los segundos 17 y 28 aproximadamente, en donde UDP vuelve a aprovechar el ancho de banda para transmitir más paquetes.

Luego cuando la ventana de congestión sube, UDP baja el tráfico debido a que se reduce su ancho de banda, ya que los nodos TCP vuelven a transmitir más paquetes. Finalmente una vez que UDP termina de transmitir, los nodos TCP ya son "libres" para monopolizar el canal.

Ethernet	IPv4 · 3	IPv6	TCP · 2	UDP · 1					
Address A ▾	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
10.1.1.1	10.2.2.1	1.404	559 k	904	531 k	500	28 k	0.000000	35.2296
10.1.2.1	10.2.3.1	1.351	539 k	872	512 k	479	26 k	0.001160	34.6869
10.1.3.1	10.2.1.1	2.556	766 k	1.278	692 k	1.278	74 k	0.008201	29.4523

Figura 28: UDP envió un total 2556 paquetes, mientras que TCP envió 1404 y 1351 respectivamente

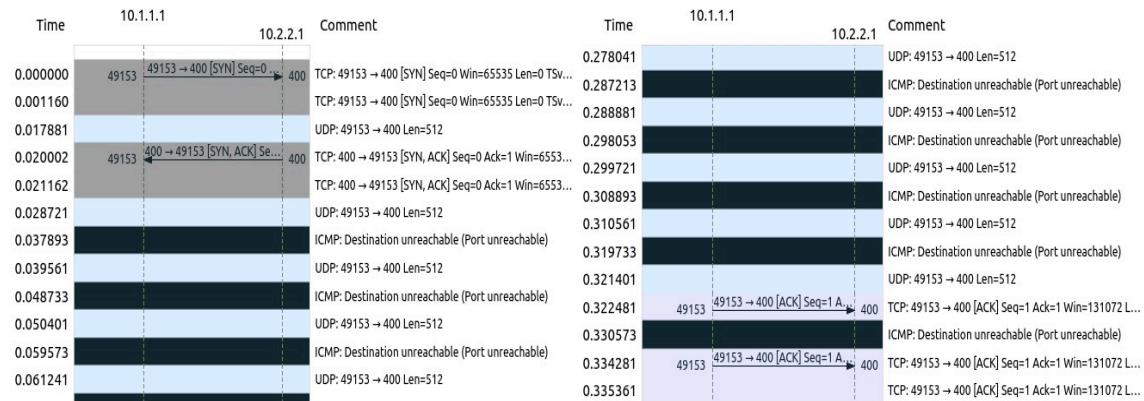


Figura 29: Three Way Handshake de TCP

En la figura 29 se puede observar el Three Way Handshake para iniciar la conversación TCP, mientras además se envían paquetes UDP, es por este motivo que en el segundo 0 los paquetes UDP aprovechan el máximo ancho de banda disponible antes que los paquetes TCP empiezan a transmitir.

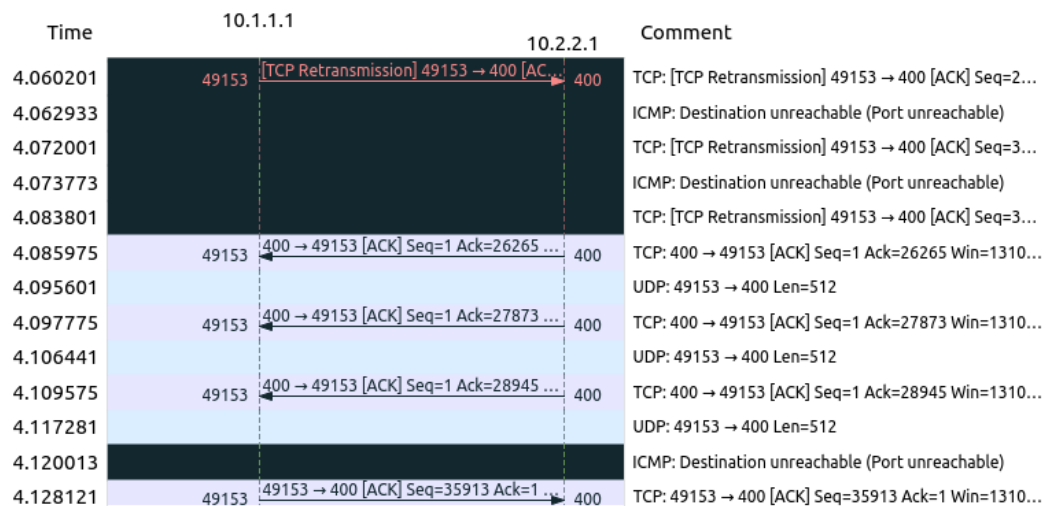


Figura 30: Emisor 10.1.1.1 envía un segmento TCP Fast Retransmission después del tercer Dup ACK recibido en el archivo gp3-tp2-tcp-udp-0-0.pcap

Los paquetes perdidos de UDP no se recuperan por características del protocolo. En cambio, como podemos observar en la figura los paquetes TCP se pueden recuperar por medio de la retransmisión, si el emisor no recibe un ACK para un segmento de datos dentro del tiempo de espera establecido, asume que el segmento se perdió y lo retransmite.

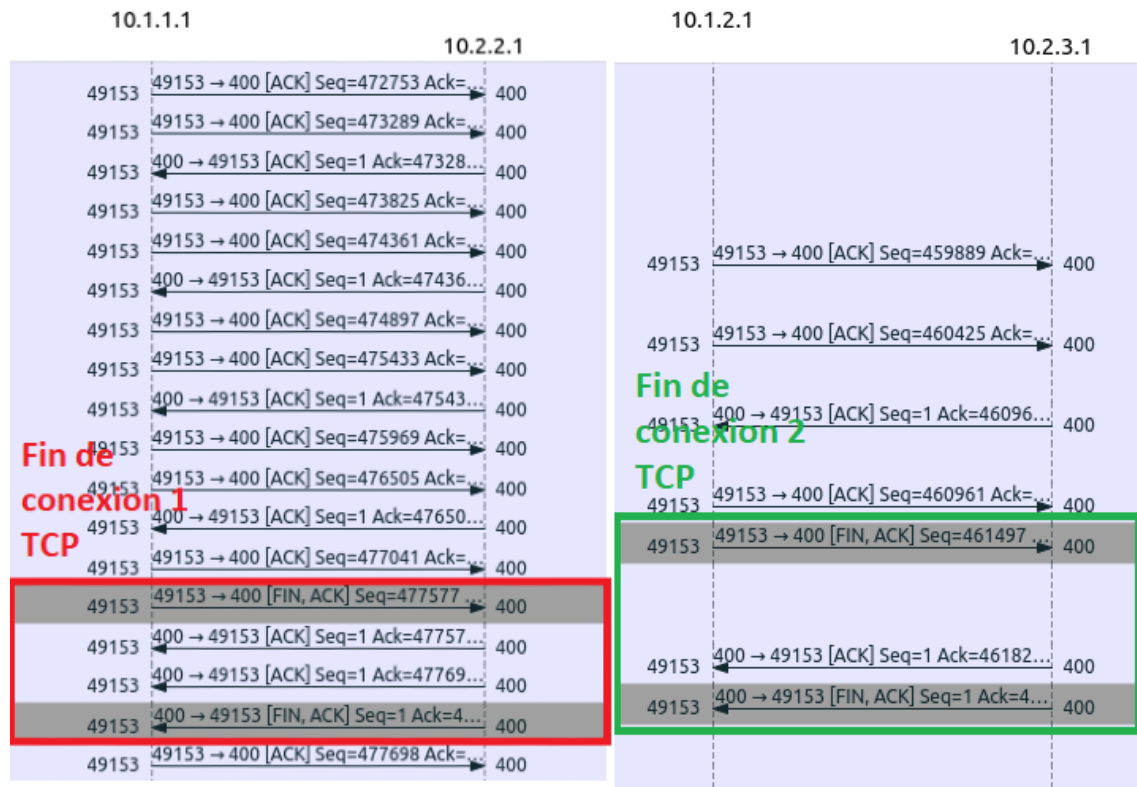


Figura 31: Flow Graph de Wireshark en el que se señala el momento del Fin de conexión entre los emisores y receptores TCP.

```

000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
> .... .... ...1 = Fin: Set
[TCP Flags: .....A...F]

```

Al igual como se explicó en la sección TCP, los nodos TCP terminan la conversación.

Conclusión

A modo de conclusión, en este trabajo pudimos no solo comprender mejor el funcionamiento de redes en general, sino que también ahondamos en la comprensión de la capa de transporte, específicamente en los protocolos TCP y UDP. Además, pudimos poner en práctica los conocimientos teóricos al analizar el tráfico que simulamos con NS3.

Así mismo, en el grupo coincidimos que, aunque por momentos resultó difícil, indagar e investigar en materia de protocolos resultó muy enriquecedor. Algunos temas, como “Etapas de la transmisión TCP”, “Ancho de banda”, “Algoritmos de control de congestión (Slow start, fast recovery, New Reno, etc)” y “comparativa entre TCP-UDP”, etc son en particular interesantes de analizar en contexto tráfico de datos.

Nos pareció particularmente interesante el hecho de poder observar la convivencia en conversaciones de diferentes protocolos al mismo tiempo de transmisión y el hecho de ver cómo en algunos casos, en la práctica no se refleja al 100% lo planteado en la teoría, por ejemplo, las anomalías descritas en la sección de congestión de red.

Finalmente, concluimos considerando una experiencia positiva el poder investigar simulando una topología Dumbbell junto con todos los escenarios realizados durante el desarrollo de este trabajo.

Bibliografía

[Descarga de NS3](#)

[TCP Congestion Avoidance RFC](#)

[Practical TCP Series - The TCP Window](#)

[NS3- Queues Models](#)

[El sistema de trazas de NS3](#)

[What is TCP New Reno?](#)

[Explorando posibles mejoras de protocolo TCP en redes móviles](#)

[How to do TCP Retransmission Analysis using Wireshark](#)

[Conceptos Avanzados del Protocolo TCP](#)

[TCP Congestion Control // Hands-On Deep Dive TCP Analysis with Wireshark](#)

[How TCP Works - Window Scaling Graph](#)

[SF19US - 07 How TCP congestion control algorithms work](#)

[TCP: Flow Control and Window Size](#)

[UDP RFC](#)