UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Sistema de medida de calidad de servicio de red basado en trenes de paquetes usando mensajes estándar de control

Víctor Matesanz Cotillas
Tutor: Jorge Enrique López de Vergara Méndez

Mayo 2021

Sistema de medida de calidad de servicio de red basado en trenes de paquetes usando mensajes estándar de control

AUTOR: Víctor Matesanz Cotillas
TUTOR: Jorge Enrique López de Vergara Méndez

Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo 2021

Resumen

Hoy en día, debido a la evolución de las tecnologías y a la mejora en la calidad de servicio, está teniendo lugar un proceso de digitalización en el ámbito empresarial sin precedentes. De hecho, durante este último año este proceso se ha acentuado aún más por la pandemia COVID-19 que impide viajes, reuniones… etc. En este contexto, las empresas demandan cada vez más recursos y calidad en el servicio, puesto que de ello depende el correcto funcionamiento de la compañía. Ante el aumento de esta demanda, las operadoras se han visto obligadas a ofrecer herramientas a sus clientes para que puedan comprobar que se está cumpliendo con el acuerdo de nivel de servicio. Sin embargo, estas herramientas utilizan habitualmente técnicas basadas en el protocolo TCP (por ejemplo, iperf) que han resultado ser poco eficientes en determinadas ocasiones, puesto que es necesario parar toda la actividad de la red para poder realizar las medidas. Esto se debe a que estas herramientas generalmente funcionan por inundación. Además, es necesario tener la aplicación tanto en el cliente como en el servidor para poder hacer las medidas.

Consecuentemente, este Trabajo de Fin de Grado busca encontrar un método capaz de monitorizar la red de una forma más simple y eficiente. Para ello, se ha desarrollado un sistema de medida de calidad de servicio de red basado en trenes de paquetes usando mensajes estándar de control. Este sistema mide los parámetros habituales de calidad de servicio: pérdida de paquetes, retardo, variación del retardo(*jitter*), tasa de transferencia(*throughput*) y ancho de banda. Se han estudiado e implementado dos técnicas para realizar las estimaciones: una consiste en enviar y recibir paquetes ICMP de tipo eco, y la otra en transmitir paquetes UDP para generar mensajes de respuesta ICMP de puerto inalcanzable. Posteriormente, se ha desarrollado un programa en C capaz de enviar las ráfagas de paquetes y medir los parámetros de calidad de servicio. Gracias a este programa, se han puesto a prueba ambas técnicas, tanto en entornos reales como virtuales. Finalmente, se han analizado los resultados y se ha comprobado que la técnica basada en trenes de paquetes ICMP cumple con los requisitos mencionados anteriormente.

Palabras clave

ICMP, UDP, monitorización de redes, trenes de paquetes, parámetros de calidad de servicio, QoS, ancho de banda, medidas activas

Abstract

Nowadays, the evolution of technologies and the improvement in the quality of service (QoS) has caused an unprecedented business digitalization process. In fact, during this year this tendency has increased due to the pandemic situation of covid-19, which prevents meetings, travels, etc. In this context, companies require more resources and quality in the service, because the correct operation of the business depends on it. In order to attend to this demand, operators have been forced to offer tools for their clients. Thus, they can verify that the service level agreement is being fulfilled. However, these tools are based on the protocol TCP which has turned out to be inefficient on some occasions, since it is required to stop all network activity for carrying out the measurements. The main reason is because these tools generally work by flood. Furthermore, it is necessary to run the application on both the client and the server.

Thus, the aim of this Bachelor Thesis is to find a method for monitoring the network in a simple and efficient way. For achieving this goal, a system for measuring the network quality service based on packet trains using standard control messages has been developed. This system measures the usual parameters of quality of service: packet loss, delay, jitter, throughput and bandwidth. Thereby, two different techniques have been studied and implemented. The first one consists in sending and receiving ICMP echo packets, while the second one in transmitting UDP packets in order to generate ICMP port unreachable messages. Then, a program in C has been developed for sending bursts of packets and measuring the QoS parameters. Thanks to this program, both techniques have been tested in real and virtual environments. Finally, the results have been analysed and it has been proved that the technique based on ICMP packet trains fulfils the requirements mentioned above.

Keywords

ICMP, UDP, network monitoring, packet trains, quality of service parameters, QoS, bandwidth, active measurements.

Agradecimientos

Me gustaría comenzar agradeciendo a mi tutor Jorge E. López de Vergara, sin duda uno de los mejores profesores que he tenido. Quiero dar las gracias por la implicación que has mostrado. Durante todo el desarrollo del proyecto hemos estado reuniéndonos cada dos semanas, discutiendo sobre mis avances, incluso cuando he tenido algún contratiempo durante esas dos semanas no ha dudado en dedicar tiempo a ayudarme. Además, me aconsejó a la hora de elegir TFG y, finalmente me propuso este proyecto. He de decir que este trabajo no era mi primera opción, de hecho, tenía hablado otro TFG. Sin embargo, me ha sorprendido gratamente y me ha acabado gustando. Gracias a mi tutor y a este TFG he aprendido muchísimo, y creo que eso al final es lo más importante. Por todo ello, gracias Jorge.

También quiero agradecer a mis padres, que siempre estuvieran ahí. Al principio del proyecto tuve muchos problemas con el entorno de trabajo. Mi ordenador no cumplía con los requisitos para realizar las pruebas. Ellos se involucraron ayudándome a buscar formas de solucionar estos problemas. Hay que recordar que debido a la situación que vivimos no se podía acceder al laboratorio.

Por último, me gustaría dar las gracias a todas aquellas personas que, aunque no me han ayudado directamente con el proyecto han estado ahí en mi día a día dándome apoyo y cariño. Por eso, quiero dar las gracias a mi hermano Carlos, a mis amigos y a mis compañeros de carrera.

¡Muchas gracias a todos!

Víctor Matesanz Cotillas

Mayo 2021

ÍNDICE

ÍNDICE DE FIGURAS

GLOSARIO DE ACRÓNIMOS

- **ACK:** Acknowledgment. Mensaje de reconocimiento.
- **CPU:** Central Processing Unit. Unidad central de procesamiento
- **CRC:** Cyclic Redundancy Check. Verificación de redundancia cíclica
- **ETSI:** European Telecommunications Standards Institute. Instituto Europeo de Normas de Telecomunicaciones.
- **FTTH**: Fiber to the home, Fibra óptica hasta el hogar
- **HTTP**: Hypertext Transfer Protocol. Protocolo de transferencia de hipertexto.
- **ICMP:** Internet Control Message Protocol. Protocolo de mensajes de control de internet.
- **IP:** Internet Protocol. Protocolo de internet
- **IT:** Information Technology. Tecnologías de la información.
- **LAN:** Local Area Network. Red de área local
- **MAC:** Media Access Control Address. Dirección de control de acceso a los medios.
- **MTU:** Maximum Transfer Unit. Unidad máxima de transferencia.
- **OWD:** One Way Delay. Retraso en un sentido.
- **PCAP:** Packet Capture. Captura de paquetes.
- **QOE:** Quality Of Experience. Calidad de experiencia.
- **QOS:** Quality Of Service. Calidad de servicio.
- **RTT:** Round Trip Time. Tiempo de ida y vuelta.
- **SLA:** Service Level Agreement. Acuerdo de nivel de servicio.
- **TCP:** Transmission Control Protocol. Protocolo de control de transmisión.
- **WAN:** Wide Area Network. Red de área amplia.

1 Introducción

1.1 Motivación

Durante estos últimos años, debido a la evolución de la tecnología y a la mejora en la calidad del servicio, se han producido cambios en los hábitos de consumo de internet. Aplicaciones como los juegos en tiempo real o la televisión en *streaming*, requieren cada vez más recursos y calidad en el servicio (*Quality of Service*, QoS).

Esta necesidad de servicios de calidad adquiere todavía más importancia cuando se habla de redes empresariales como puede ser la red de una sucursal con una red central. Hoy en día, es complicado ser competitivo si la red de la compañía no funciona correctamente. Por ello, se hace indispensable poder comprobar que se está cumpliendo con el acuerdo de nivel de servicio[1] o SLA (service level agreement) a través de alguna herramienta o aplicación que realice las medidas de los parámetros QoS. Según el informe de 2019 publicado por la Oficina Española de Usuarios de las Telecomunicaciones [1], solo en ese año se registraron más de 25.000 reclamaciones, muchas de ellas relacionadas con el incumplimiento de este acuerdo.

Cabe destacar también que en España existe una orden ministerial (IET/1090/2014, de 16 de junio) recogida en el BOE [2] que regula los procedimientos para medir los parámetros de calidad de servicio, y es de obligado cumplimiento para las operadoras. Esta orden se basa en la guía ETSI (*European Telecommunications Standards Institute*), ETSI EG 202 057-4 [15].

1.2 Descripción del problema

Para conseguir la monitorización de las redes los proveedores de IT (*Internet Technology*) ponen a disposición del cliente herramientas para medir los parámetros de calidad de la red. Hoy en día, el servicio de enlaces de una red suele tener disponible una aplicación cliente y otra servidora para poder hacer las medidas, como es el caso de la herramienta Iperf [3]. Esta aplicación, utiliza el protocolo TCP para estimar la capacidad de la red [4]. Este proceso limita en muchos casos las medidas de calidad (sobre todo de ancho de banda) ya que pueden verse afectadas por el tráfico cruzado o porque haya aplicaciones ocupando la CPU, [5]. En otras palabras, para realizar medidas de forma precisa sería necesario parar toda la actividad que haya en la red durante el tiempo que dure la prueba. No hace falta mencionar el inconveniente que puede suponer para una compañía detenerse durante 10 segundos cada 5 minutos para poder realizar las medidas. Este es el tiempo que suele durar un test de iperf [4]. Además, como se ha mencionado antes, se requiere que tener el software tanto en el cliente como en el servidor y que este funcione de manera simultanea.

[1] Acuerdo entre el proveedor TI y el cliente por el cual se define la calidad del servicio.

Por todo ello, este TFG plantea implementar un sistema de medidas de red que sea simple, rápido y preciso. Utilizando técnicas basadas en trenes de paquetes se propone realizar las medidas de calidad de servicio mediante el envío de ICMP (*Internet Control Message Protocol*) de tipo eco, o bien mediante datagramas UDP (*User Datagram Protocol*) y respuestas ICMP de puerto inalcanzable. De esta forma, el otro extremo podrá responder sin tener que estar escuchando, puesto que son mensajes de control ya implementados por la propia pila de comunicaciones. Además de las medidas habituales de retardo o perdidas de paquetes, este trabajo se centrará sobre todo en obtener estimaciones del ancho de banda utilizando estas técnicas, puesto que para medir el resto de parámetros QoS ya existen herramientas que lo hacen de forma eficiente.

*1.3 Obj*etivos

El objetivo principal de este trabajo es comprobar si estas técnicas basadas en mensajes estándar de control pueden mejorar las prestaciones de las herramientas actuales para medir la calidad del servicio.

Por ello, el primer objetivo de este trabajo será comprender el funcionamiento tanto de los protocolos ICMP de eco, UDP e ICMP de puerto inalcanzable, así como de las técnicas de trenes de paquetes que se utilizarán para tomar las medidas.

Para poner en práctica lo anterior, se desarrollará un programa capaz de enviar las ráfagas de ICMP de eco y de UDP de *traceroute* para poder realizar las medidas. El propio programa deberá calcular y mostrar el ancho de banda de la red, además del resto de parámetros de calidad de servicio habituales.

También, utilizando el programa anterior, se pondrán a prueba ambas técnicas en entornos controlados tanto virtuales como reales. Finalmente, se validarán y comprobarán los resultados, pudiendo así determinar los puntos fuertes y las limitaciones de estos métodos en lugar de usar los convencionales.

1.4 Fases de realización

En este apartado se explicarán las etapas en que se ha dividido la realización de este trabajo, tal y como se muestra en la figura 1.1. Estas han sido las fases:

Estado del arte. En la primera etapa se recopiló información esencial para poder desarrollar el proyecto. Se consultaron artículos sobre estudios relacionados con la materia. Se buscó información acerca de los protocolos ICMP eco, UDP e ICMP de Puerto inalcanzable. También, sobre la librería PCAP. Además, se revisó el trabajo realizado en asignaturas realizadas durante la carrera como Arquitectura de Redes, Redes Multimedia o Sistemas Informáticos, entre otras.

Viabilidad del proyecto El siguiente paso fue comprobar si era posible utilizar estas técnicas para medir los parámetros de calidad. Para ello, se realizaron pruebas en un entorno real utilizando los programas burstPing [6] y nping [7] para enviar los paquetes, y la herramienta Wireshark para analizar los resultados. De esta manera, se verificó que era posible utilizar mensajes de control para medir los parámetros QoS, y sobre todo la capacidad del enlace. Por último, se probó la herramienta Token bucket [8] para limitar el ancho de banda y establecer un entorno controlado de medida.

Implementación. Durante esta etapa se realizó el programa denominado "QoStool". Se consultaron tanto practicas anteriores como código externo para entender cómo implementar los mensajes a nivel de enlace. También, se utilizó la librería PCAP para el envío y la recepción de paquetes.

Pruebas y resultados. Para ello, se establecieron los entornos adecuados para examinar estas técnicas. Se probó el programa tanto en entornos reales como virtuales controlados. Finalmente, se analizaron los resultados obtenidos.

Escritura de la memoria del TFG. Por último, se redactó este documento. En él se describe todo lo aprendido sobre ambas técnicas y se da información relevante para trabajos similares que se desarrollen en el futuro.

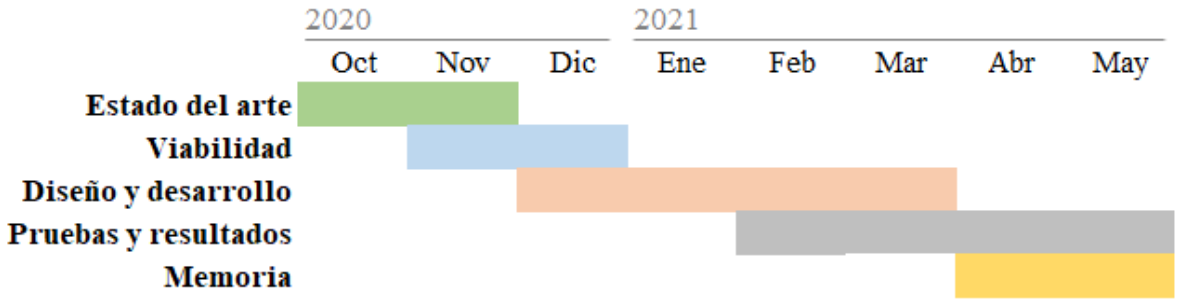

Figura 1.1: Diagrama de Gantt. El trabajo se ha realizado entre octubre 2020 y mayo 2021.

1.5 Estructura del documento

El resto del documento se estructura de la siguiente manera:

Estado del arte. En esta parte se expondrá de forma general toda la información necesaria para poder entender este trabajo. Se explicarán los diferentes parámetros de calidad y su relevancia. También, se hará un inciso en las técnicas actuales de medición. Finalmente, se explicarán las técnicas que se quieren implementar para medir los parámetros QoS, así como los mensajes ICMP eco, UDP e ICMP de puerto inalcanzable.

Diseño y desarrollo. En este capítulo se enunciarán las distintas pruebas realizadas con burstPing, Nping y el Token bucket para comprobar si es viable utilizar estas técnicas. También, se mostrará cómo se ha hecho la implementación del programa, explicando todo lo que se ha tenido en cuenta. Se mostrarán partes del código para facilitar la comprensión.

Resultados y validaciones. En este apartado, además de mostrar los resultados, se expondrá de forma detallada el proceso de obtención de los mismos. Para ello, se discutirán los parámetros (tamaño del paquete, número de paquetes…) que se han establecido para hacer las medidas, sobre todo para medir de la manera más precisa posible el ancho de banda. Además, se explicará cómo son los entornos controlados en los que se han realizado las pruebas.

Conclusiones y trabajo futuro En la última parte del documento se valorarán los resultados obtenidos, así como las pruebas realizadas. Se reflexionará sobre las ventajas y las limitaciones que se han observado a la hora de poner a prueba este sistema de medida. Asimismo, se compararán las técnicas implementadas en este trabajo con las que se utilizan en la actualidad. Por último, se comentará el trabajo que se puede hacer en el futuro como continuación del realizado en este TFG.

2 Estado del arte

2.1 Introducción

En esta parte se expondrá toda la información que fue necesaria para la realización del trabajo. Primero, se hablará sobre las medidas de calidad existentes y de su importancia en la calidad de la experiencia[2] (*Quality of Experience* ,QoE). A continuación, se explicarán las técnicas más utilizadas actualmente para monitorizar las redes, explicando sus puntos fuertes y sus limitaciones. Cabe destacar que en este documento solo se expondrán las medidas activas más extendidas, ya que son las que guardan más relación con las técnicas que se quieren implementar. Por último, se profundizará en los métodos que se van a utilizar para medir la calidad del servicio, hablando sobre la técnica de trenes de paquetes. También, durante esta última parte, se explicarán las características que debe tener el tren de paquetes para realizar las medidas de forma precisa, y los protocolos implicados, mostrando sus cabeceras.

2.2 Medidas de calidad

En este apartado se describirán las medidas de calidad que se realizan habitualmente en las redes de datos:

Pérdidas de paquetes: Se considera que se ha perdido un paquete cuando este no consigue llegar a su destino. Esto se puede deber a que el paquete no ha alcanzado su destino, ha llegado con errores, se ha descartado por exceder el tiempo de reensamblado, o que el buffer se encuentre lleno y tenga que ser descartado [9]. Es un factor clave para garantizar la calidad del servicio, la perdida de paquetes puede tener un gran impacto en las aplicaciones. Las aplicaciones en tiempo real como Skype suelen utilizar el protocolo UDP y pueden sufrir un deterioro en la calidad de las llamadas cuando las pérdidas son muy grandes. En la misma situación el protocolo TCP reduciría el *throughtput* para conseguir enviar los paquetes, pero si las pérdidas son excesivas se resentirá la funcionalidad de la aplicación [10].

Ancho de banda (*Bandwidth*): Es la cantidad datos que se pueden enviar o recibir por intervalo de tiempo de transmisión en una red o enlace. En una red se puede calcular de forma simple: Nºbits transmitidos entre unidad de tiempo. Hay que diferenciar el ancho de banda de un enlace con el ancho de banda extremo a extremo que está marcado por el cuello de botella (*botleneck*) de la red, y condiciona la velocidad de transmisión del camino [11]. Además, es necesario diferenciar dos conceptos: la capacidad máxima del enlace y la capacidad disponible del enlace que es la que no se está utilizando. En este trabajo únicamente se medirá la capacidad máxima extremo a extremo del enlace. El ancho de banda está limitado por las características físicas del medio por el que se transmiten los datos (fibra óptica, por ejemplo), así como por las características del hardware de

[2] Mide el nivel de molestia o deleite de un usuario con un servicio

transmisión o recepción [11]. El ancho de banda es un parametro crucial a la hora de analizar la calidad del servicio. En una compañía puede suponer que la red de la empresa sea capaz de mantener las reuniones por videollamada con una buena calidad de experiencia, permitiendo a los empleados trabajar desde casa. En general, la buena funcionalidad de la red de una empresa depende en gran medida de este parametro de calidad.

Transferencia de datos (*Thoughtput*): Se define como la tasa transferencia efectiva, es decir el volumen de información que circula por una red. Se relaciona con el ancho de banda y mide el rendimiento de las conexiones TCP. El inconveniente es su dependencia con factores como la congestión o el tráfico cruzado [12]. En este trabajo será la cantidad de datos que se envian a una red en un tiempo determinado.

Retardo en un sentido (*One-Way Delay*, OWD): Es el tiempo que transcurre desde que se envía un bit hasta que llega al receptor. El retardo es la suma de los tiempos de transmisión, de propagación y de procesado. Este parámetro está limitado por componentes físicos como la propagación de la señal a través de la fibra óptica. Pero también por elementos de hardware como puede ser el tiempo de procesamiento, de transmisión o de encolado.

Retardo ida y vuelta (*Round-Trip Time*, RTT): Se define RTT como el tiempo que tarda un paquete desde que sale del emisor hasta que vuelve al propio emisor habiendo pasado por el receptor [13]. Esta medida no es exactamente la mitad del OWD ya que RTT incluye el tiempo de procesamiento del paquete en el receptor. Además, el enlace no tiene por qué ser simétrico (ADSL, por ejemplo). RTT es una medida importante cuando se establece una conexión TCP, puesto que indica la cantidad de datos que se pueden enviar antes de que llegue el mensaje ACK [13]. Por la propia naturaleza de este trabajo se medirá el RTT en lugar del OWD, ya que se puede medir con facilidad utilizando únicamente un terminal.

Jitter: Se define como la variación del retardo de los paquetes de un tren de paquetes con respecto al retardo esperado. Un *jitter* alto puede perjudicar a aplicaciones que requieran una entrega regular de paquetes, como son las que funcionan en tiempo real, por ejemplo, las aplicaciones por voz. Generalmente, este problema se soluciona insertando un *buffer* en el receptor de manera que los paquetes se encolan. Para diseñar el *buffer* es necesario saber el *jitte*r que sufre la red, por lo tanto, hay que medirlo [14].

2.3 Otras técnicas de medición de los parámetros QoS

Descarga de archivos. Es un método muy extendido para medir de forma rápida el ancho de banda. Grandes proveedores de internet como Vodafone lo usan en sus pruebas de velocidad (www.vodafone.es/test). Esta técnica utiliza descargas de HTTP para estimar la capacidad del enlace. Esta se hallaría dividiendo el tamaño del archivo entre el tiempo que tarda en descargarse. Este método ha sido estandarizado por la ETSI en la EG 202 057-4 [15]. De acuerdo con la guía, el fichero debe ser 8 veces el tamaño nominal del link. Por ejemplo, en un enlace de 5 Mbps/s el fichero debe der de 40Mbit de tamaño. Para obtener estimaciones más precisas del ancho de banda, los test de

velocidad suelen realizar varias medidas de manera consecutiva. Una de las ventajas de esta técnica es que es sencilla de implementar. Sin embargo, el tiempo que se tarda en hallar la capacidad del enlace es grande, será de al menos 8 segundos ya que el fichero es 8 veces mayor que el cuello de botella. Además, tanto el tráfico cruzado como la ocupación de la CPU pueden influir en las medidas. Esto se debe a que el protocolo TCP tiene mecanismos de control de congestión y de flujo [5].

Pares de paquetes. Este método consiste en enviar pares de paquetes al mismo tiempo y de idéntico tamaño, a la velocidad máxima de transmisión que permita el emisor. Cuando pasan a través del cuello de botella habrá una distancia entre ellos proporcional a la capacidad del enlace. Se puede estimar el ancho de banda dividiendo el tamaño de los paquetes entre la diferencia del tiempo de llegada que hay entre ellos. Este método, a diferencia del de descarga de ficheros, puede utilizar el protocolo de transporte UDP, requiriendo sobre 4.8 ms en medir el enlace anterior de 5Mbps con paquetes de 1500bytes (2x1500x8/5Mbps). No obstante, los paquetes deben enviarse a la vez, por lo tanto, la precisión de este método depende en gran media de la resolución del reloj. Otra desventaja significativa, es que la técnica de pares de paquetes también es sensible al tráfico cruzado, los paquetes indeseados se pueden colar entre ambos paquetes empeorando las estimaciones [5].

2.4 Técnica de trenes de paquetes

Los métodos descritos en el apartado anterior son susceptibles al tráfico cruzado (figura 2.1). Por ello, en este trabajo se utilizará la técnica de trenes de paquetes. Esta técnica consiste en enviar un gran número de paquetes consecutivos a una tasa de transmisión mayor que la capacidad máxima del enlace. En una medida extremo a extremo la máxima tasa de transmisión está limitada por el cuello de botella. De esta manera, se puede hallar el ancho de banda del enlace midiendo el tiempo entre paquetes en el receptor [5].

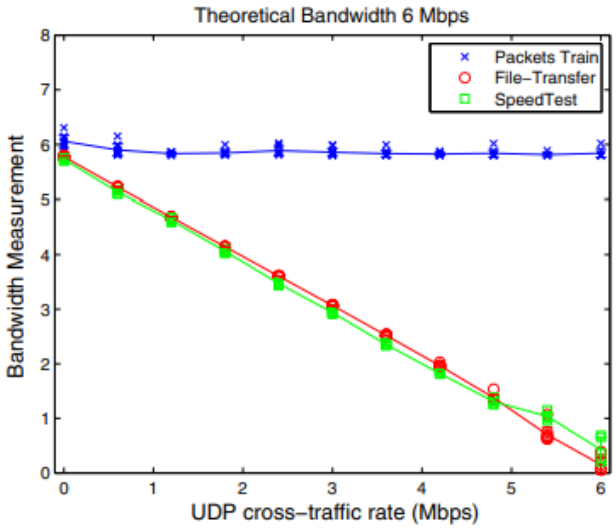

Figura 2.1: Respuesta de las tres técnicas ante el tráfico cruzado. La gráfica muestra como con mucho tráfico cruzado UDP, únicamente el método de trenes de paquetes realiza buenas estimaciones. *Figura extraída de [5].*

Como se puede observar en la figura 2.1 este método posee gran robustez ante el tráfico cruzado. Sin embargo, no es inmune a él debido a que entre paquete y paquete se puede colar uno no deseado que interfiera en las medidas, tal y como sucede en la técnica de pares de paquetes. Para que esto no suceda, el tiempo entre los paquetes debe ser el mínimo que permita la red (*minimum inter-packet gap*). También, aumentando el número de paquetes, disminuye la probabilidad de que haya tráfico cruzado en el espacio entre paquetes (figura 2.2). No obstante, un número muy grande de paquetes puede resultar intrusivo para el sistema provocando la perdida de los mismos [5]. Asimismo, no es posible en las redes actuales enviar un gran número de paquetes, ya que pueden ser considerados como un ataque [16]. Con todo ello, el número de paquetes tiene que ser un consenso entre la inmunidad al tráfico y la intrusión [5].

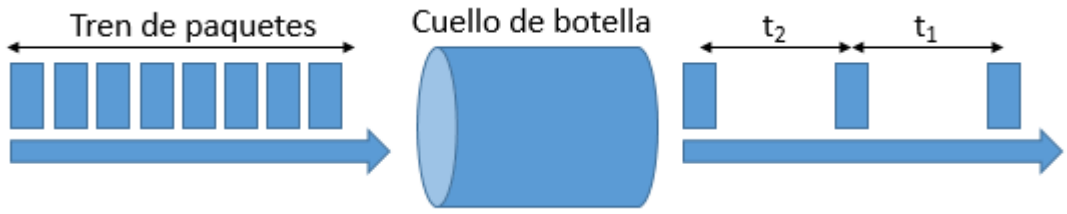

Figura 2.2: Esquema de la técnica de trenes de paquetes. Se envía un tren de paquetes a una velocidad mayor que la capacidad máxima del enlace que está marcada por el cuello de botella. Como resultado los paquetes se separarán una distancia proporcional al ancho de banda (t1 y t2).

También, hay que tener en cuenta el tamaño de los paquetes para que las estimaciones sean lo más precisas posibles. Si el paquete es menor que la MTU (*Maximum Transmissión Unit* o unidad máxima de transferencia) no se aprovecharía todo el ancho de banda (figura 2.3). Por otro lado, si el paquete es muy grande se fragmentará para adaptarse a la MTU de la interfaz. Lo óptimo para medir el ancho de banda sería enviar paquetes con el tamaño MTU.

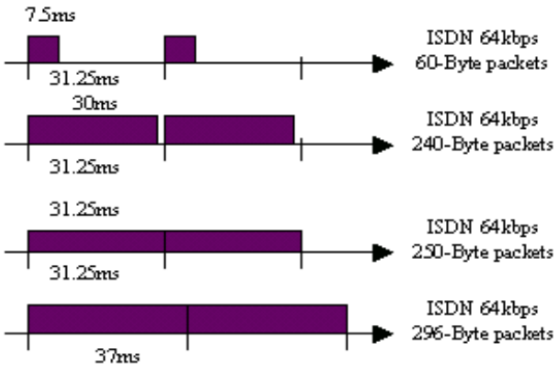

Figura 2.3: Envío de paquetes de distinto tamaño a la red. Se enviaron paquetes de diferentes tamaños por un enlace capaz de procesar 32 paquetes por segundo. *Figura extraída de [16]*

Por ejemplo, en la figura 2.3, se puede observar como el ancho de banda que se obtuvo al enviar paquetes de 60 Bytes fue de 15,36Kbps (60x8 bits/31.25ms) en lugar de los 64kbps de ancho de banda que tiene el enlace.

Por último, para medir el ancho de banda real en una red ethernet hay que considerar los bytes del preámbulo (8bytes), los del CRC (4 Bytes), y los 12 bytes de la pausa mínima entre paquetes (*interframe gap)*.En total serían 24 bytes adicionales(figura 2.4).

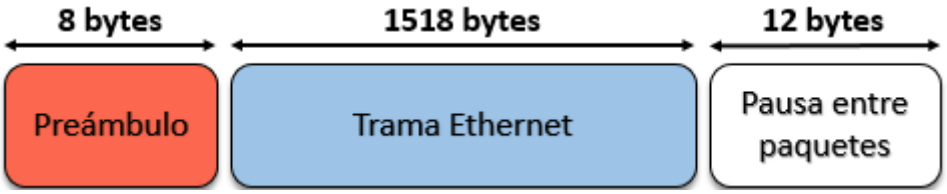

Figura 2.4: Trama ethernet con los bytes adicionales. En este caso la MTU sería 1514 bytes a los que hay que añadir los 24 bytes del CRC, el preámbulo y la pausa entre paquetes (*interframe gap).*

2.5 ICMP eco

ICMP es un subprotocolo de IP cuyo propósito es reportar errores e información operativa. Por ejemplo, que el puerto X no se encuentra activo (*port-unreachable*). El ICMP de tipo eco se utiliza para comprobar si dos *host* en la capa de red se pueden comunicar, reportando los fallos que puedan existir [17] . El funcionamiento sería el siguiente: el *host* 1 envía una petición *(request)* al *host* 2, este deberá contestar con un mensaje de respuesta *(reply)* que contenga exactamente los mismos datos que la petición (tamaño,número de secuencia, ect).

En este trabajo, se propone medir los parámetros de calidad de servicio a través de mensajes ICMP *request/reply* enviados en ráfaga, según la técnica de medida de los trenes de paquetes. Enviando un gran número de peticiones a una velocidad de transmisión tan alta como la capacidad del enlace y con el tamaño adecuado, se podría medir el ancho de banda (figura 2.5). Con el número de secuencia se puede saber que paquetes se han perdido.

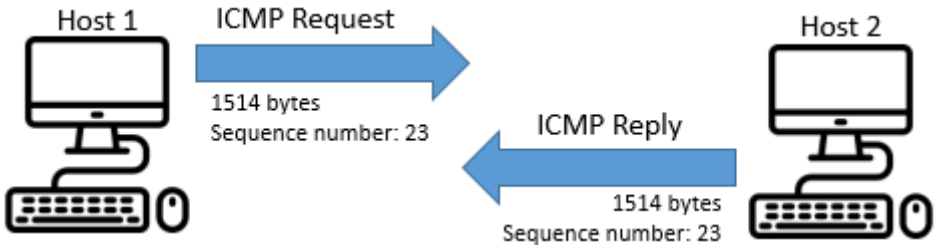

Figura 2.5: Ejemplo de intercambio de tramas ICMP de tipo eco. El *host 1* envía una petición ICMP (*request*) de tamaño 1514 bytes y número de secuencia 23. Cuando el *host 2* recibe el mensaje con la petición, este envía un mensaje ICMP de respuesta (*reply*) con las mismas características.

2.6 UDP e ICMP de puerto inalcanzable

Actualmente existe la tendencia de no permitir tráfico ICMP para evitar posibles ataques. Por este motivo, también se propone hacer las medidas de calidad de servicio a través de mensajes UDP/ICMP de puerto inalcanzable, que podrían estar disponible si solo se filtran los mensajes de eco. Por ejemplo, cuando se envía un datagrama a un puerto que no está activo, el *host* responderá enviando un mensaje ICMP de puerto inalcanzable [17]. De este modo, la técnica será muy similar a la de los trenesde paquetes ICMP (figura 2.6). En este caso el número de puerto se utilizará para identificar qué paquetes se han perdido.

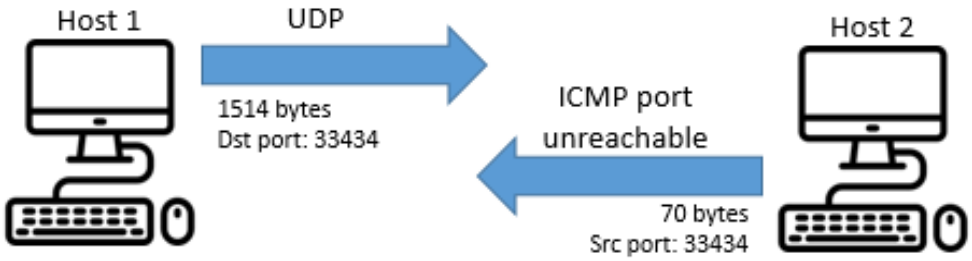

Figura 2.6: Ejemplo del envío de paquetes UDP. El *host 1* envía un paquete UDP de tamaño 1514 *bytes* hacia el puerto 33434. El *host 2* responde con un paquete ICMP de puerto inalcanzable (*unreachable*) cuyo puerto de origen es el 33434 y su tamaño 70 bytes (no tiene por qué ser ese tamaño)

En el artículo "*On the effect of concurrent applications in bandwidth measurement speedometers"* de J. Ramos, P.M. Santiago del Río, J.Aracil y J.E. López de Vergara [5] ya se utilizaron paquetes UDP para medir el ancho de banda con buenos resultados. Este protocolo no tiene mecanismos de control de congestión y de flujo como si los tienen TCP (utilizado por Iperf). En este caso, las medidas se obtuvieron utilizando una aplicación tanto en el cliente como en el servidor.

2.7 Conclusión

En este capítulo se han definido los distintos parámetros de calidad (QoS) y su importancia respecto de la calidad percibida por el usuario (QoE). También, se ha descrito el método de trenes de paquetes, que es un método más fiable ante tráfico cruzado. Además, se ha hablado sobre las características que debe tener el tren de paquetes para estimar la capacidad del enlace: el número de paquetes enviados debe ser un consenso entre la precisión de las medidas y el intrusismo, la velocidad de transmisión debe ser superior al cuello de botella, el tamaño debe ser la MTU y hay que considerar los 24 bytes adicionales. Por último, se han explicado las dos técnicas de medición que se van a poner a prueba en este Trabajo: con trenes de paquetes ICMP de tipo eco, enviando peticiones ICMP (*request*) y recibiendo las respuestas (ICMP *reply*); y transmitiendo trenes de paquetes UDP a determinados puertos reservados para generar mensajes de respuesta ICMP de puerto inalcanzable.

3 Diseño y desarrollo

3.1 Introducción

El objetivo de este capítulo es mostrar el proceso de implementación de las técnicas mencionadas en la sección anterior. Para ello, se explicarán las pruebas que se hicieron con Burstping [6], Nping [7] y Tbf de Linux [8] para comprobar que el trabajo era viable. A continuación, en el apartado 3.3 se hablará de todo lo relacionado con el programa que se ha creado para enviar los paquetes y realizar las medidas, desde cómo se han creado las cabeceras de los paquetes hasta cómo se han medido los parámetros QoS. Se mostrarán ejemplos del código con el fin de entender mejor el programa.

3.2 Viabilidad del proyecto

Para evaluar la viabilidad del proyecto, se utilizó burstPing y Nping para el envío de los paquetes, tratando de reproducir el envío de trenes de paquetes. A continuación, se hicieron pruebas con Token Bucket para limitar la capacidad del enlace. Los resultados se estudiaron con el analizador de red Wireshark, evaluando si era posible estimar la capacidad del enlace en estos casos. Estas pruebas se realizaron en un entorno doméstico con un ordenador conectado a un router a través de un cable RJ-45 CAT6. El router es el modelo cbn cg7486e, que alcanza velocidades de 1 Gbit/s. El ordenador es el modelo Toshiba Satellite C50-A-1GHz con una tarjeta de red Qualcomm Atheros QCA8172 Fast Ethernet. Es decir, el cuello de botella de esta red está marcado por la tarjeta de red cuya capacidad de enlace es de 100Mbit/s.

3.2.1 BurstPing

BurstPing es un programa escrito en C++ que ha sido implementado por Mirsky.Y, Kalbo.N, A.Vesper y Elovici.Y [6]. BurstPing envía ráfagas de paquetes ICMP echo *request* a una dirección IP configurable y captura los paquetes ICMP *reply.* De este modo, es capaz de medir el RTT y el *jitter*. Además, permite configurar el número de paquetes enviados, la velocidad de transmisión y el tamaño de los paquetes [6]. Hay que mencionar también, que a pesar de que envía ráfagas de paquetes su utilidad no esta pensada para medir la velocidad del enlace. El programa se ejecutó con la siguiente línea en el terminal:

./burstPing 192.168.1.1 -s 1472 -i 100 -c 50

Es decir, se enviaron 1514 bytes (1472+42 bytes de las cabeceras), 50 paquetes y 100 microsegundos entre paquetes (121Mbps). Sin embargo, el programa envía los paquetes fragmentados, enviando un paquete de 1514 bytes (MTU) y otro de 62 Bytes (Figura 3.1). El router responde transmitiendo dos paquetes de tamaño idéntico (Figura 3.1). De esta forma no se puede medir el ancho de banda de la red de manera precisa, ya que los paquetes de 62 bytes son menores

que el MTU, tal y como sucedía en el ejemplo de la figura 2.3. Se probó a utilizar un tamaño de paquete menor pero el resultado fue el mismo.

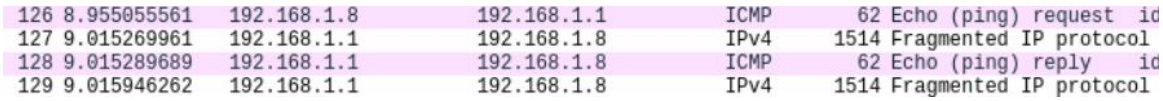

Figura 3.1: Captura de Wireshark de la prueba con burstPing. En la imagen se puede ver como el ordenador (IP 192.168.1.8) transmite los paquetes fragmentados y el router (IP 192.168.1.1) devuelve los paquetes con el mismo tamaño.

3.2.2 Nping

Al igual que burstping, Nping es una herramienta capaz de generar paquetes y analizar los tiempos de respuesta. Una de las ventajas de este programa es que permite generar una amplia variedad de protocolos, entre ellos UDP e ICMP de tipo eco. Además, es posible modificar la dirección IP de destino, el número de paquetes, el tamaño de los paquetes, el puerto de origen y el puerto de destino (esencial para UDP). [7] Para ejecutarlo se escribió la siguiente línea en el terminal: *time sudo nping --icmp --icmp-type echo --delay 0 -c 50 --data-length 1472 -H -N 192.168.1.1.* La figura 3.2 muestra cómo con Nping los paquetes se envían y se reciben correctamente.

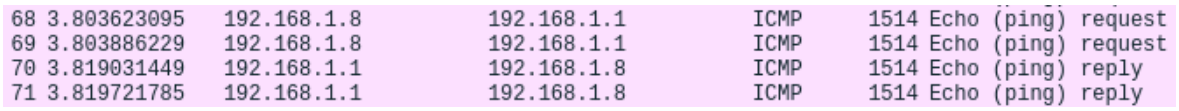

Figura 3.2: Captura de Wireshark de la prueba ICMP con Nping. En la imagen se puede ver como el ordenador (IP 192.168.1.8) transmite los paquetes ICMP *request* y el router (IP 192.168.1.1) devuelve los paquetes ICMP *reply*

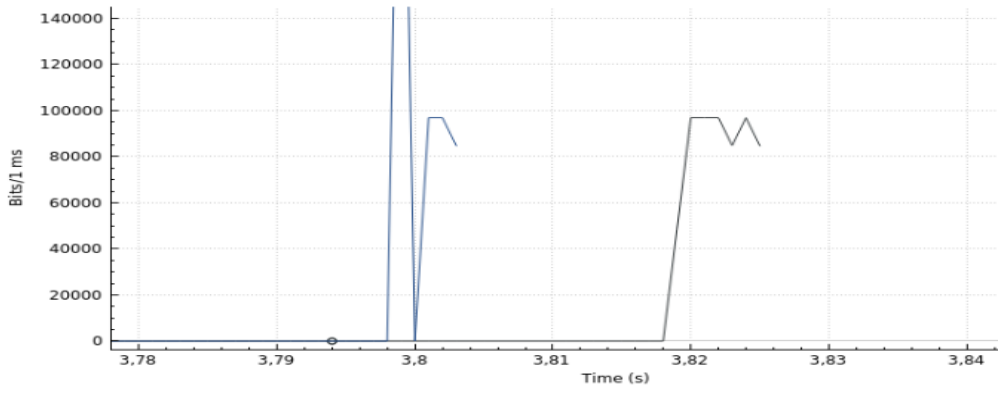

Figura 3.3: Gráfica del ancho de banda con Nping. En azul la velocidad de transmisión desde el ordenador y en negro la respuesta del router.

En la figura 3.3 se puede ver como al principio la tasa de envío es muy superior a los 100 Mbit/s del cuello de botella. Esto se debe a que la captura está realizada antes de que los paquetes se transmitan a la red. El ancho de banda medido con Wireshark (la línea negra) ha sido de 96.9 Mbit/s, que es bastante aproximado a la capacidad máxima del enlace de 100 Mbit/s.

Con Nping también se probó la técnica basada en paquetes UDP e ICMP de puerto inalcanzable. En la técnica de ICMP de tipo eco si se pierde un paquete se puede saber cuál es, ya que están identificados por el número de secuencia. Sin embargo, UDP no incluye este campo en su cabecera. Para solucionarlo, se pueden enviar los paquetes a puertos contiguos. Por ejemplo, si el paquete 10 se pierde no se recibirá su correspondiente mensaje de puerto inalcanzable. Para realizar las pruebas, se ha escogido el puerto cliente 12345 que es el puerto por defecto para hacer test [19]. Como puerto de destino, se utilizarán los puertos del 33434 al 33523 que se usan para hacer *traceroutes* con datagramas UDP [20]. Se escribió la siguiente línea en el terminal: *nping --udp -p 33434-33483 192.168.1.1 --delay 0 -c 1 -H -N -g 12345 --data-length 1472*.

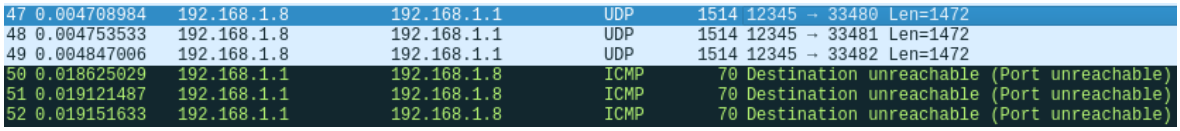

Figura 3.4: Captura de Wireshark de la prueba UDP con Nping. En la imagen muestra como el ordenador (IP 192.168.1.8) transmite los paquetes UDP desde el puerto 12345 y el router (IP 192.168.1.1) responde con paquetes ICMP de puerto inalcanzable de 70 bytes.

En este caso, no se puede medir el ancho de banda directamente con Wireshark ya que los paquetes ICMP de puerto inalcanzable solo tienen 70 bytes (figura 3.4). Para estimar el ancho de banda habría que utilizar el tamaño de los paquetes UDP enviados, 1514 bytes en este caso, y el tiempo que hay entre los paquetes ICMP de puerto inalcanzable:

Ancho de banda = Bytes_Totales_Enviados / [t_ICMP_final – t_ICMP_inicial]

3.2.3 Token bucket

En este trabajo se ha utilizado la herramienta Tbf de Linux [8]. Esta permite controlar el tráfico a través de un sistema basado en *tokens*. Los *tokens* se corresponden aproximadamente con bytes. Igualmente, a cada paquete se le asignan *tokens*, hay que tener en cuenta que existen paquetes de 0 bytes que ocupan el enlace por un tiempo. Como su propio nombre indica, esta herramienta emula un cubo que se va llenando de fichas (*tockens*). Un paquete “espera” hasta que lleguen suficientes fichas(*tockens*) para poderlo enviar a la red. Si no hay fichas disponibles, los paquetes se encolan [8]. Así, configurando la tasa de llegada de las fichas (*tokens*) se puede limitar el ancho de banda.

Para hacer las pruebas se utilizaron varios anchos de banda para comprobar si la herramienta era fiable restringiendo la capacidad del enlace. Por ejemplo, se escribió la siguiente línea en el terminal para fijar el ancho de banda a 50Mbit/s: *tc qdisc add dev enp9s0 root tbf rate 50mbit burst 12kbit latency 400ms (*figura 3.5).

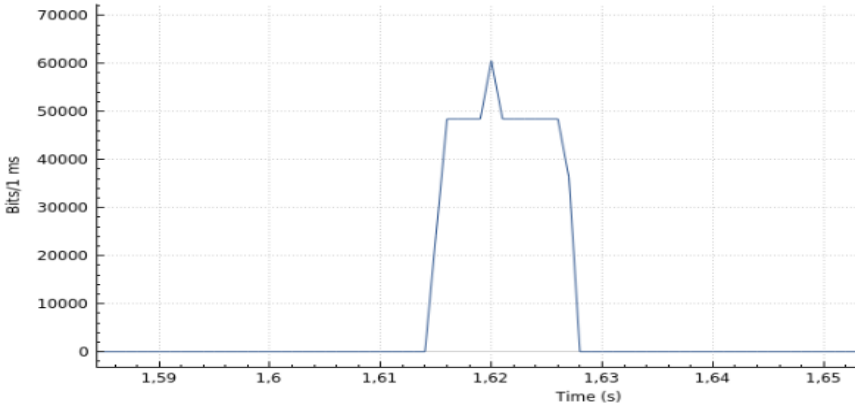

Figura 3.5: Gráfico del ancho de banda con Token bucket. El ancho de banda está limitado a aproximadamente 48Mbit/s.) Aunque no es perfecto sí que es una buena aproximación.

3.2.4 Libpcap y lenguaje de programación C

Libpcap es una librería de código abierto (*open source*) que ofrece una interfaz al programador para enviar, capturar, filtrar y modificar los paquetes de la capa de red [21].

También, hay que mencionar que el programa se ha escrito en C. Se contemplaron otras alternativas como utilizar un *wrapper* de la libpcap en Python, pero finalmente se optó por C debido a que su tiempo de ejecución es menor [22]. Esto es imprescindible a la hora de enviar los trenes de paquetes a la mayor tasa posible.

3.3 Implementación

En este apartado se explicará en detalle el programa que se ha creado para enviar las ráfagas de UDP e ICMP a una dirección IP, y para realizar las medidas de los parámetros QoS.

La figura 3.6 muestra la estructura que tiene el programa. Tras establecer los parámetros (protocolo, tamaño de los paquetes, etc.) e inicializarse, este utiliza dos hilos funcionando a la vez, uno para enviar y el otro para recibir los paquetes. De acuerdo con el esquema, el hilo 1 es el encargado de construir y enviar los mensajes. Por otro lado, el hilo 2 captura los paquetes y extrae y

guarda los datos. Para ello, el hilo 2 se encuentra escuchando constantemente la interfaz gracias a un bucle infinito. Tras esperar un tiempo de 1s ambos hilos finalizan. A continuación, una función calcula los parámetros de calidad de servicio obtenidos (RTT, paquetes perdidos, throughtput, ancho de banda y jitter).

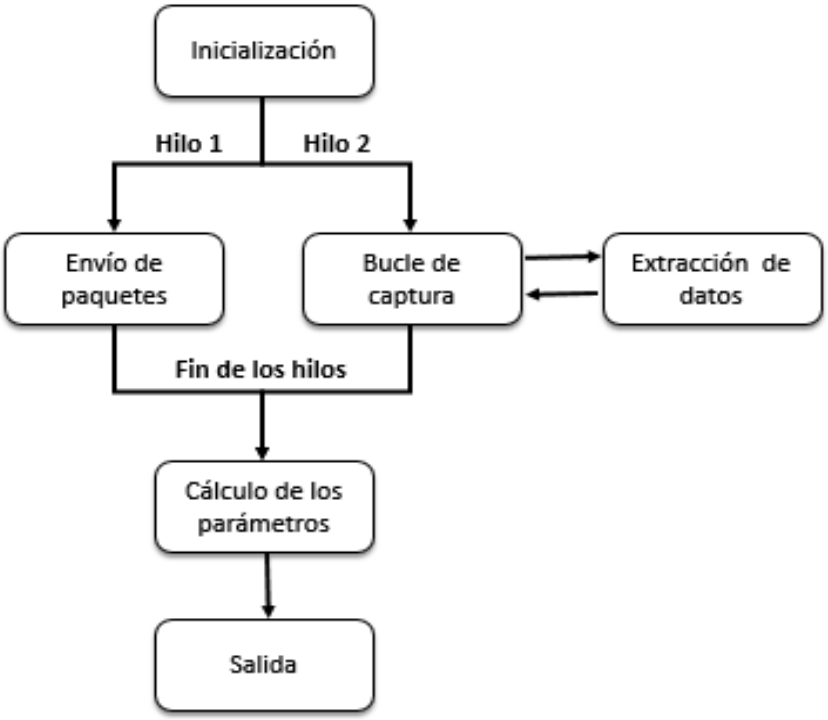

Figura 3.6: Esquema de la estructura del programa.

3.3.1 Inicialización

El programa permite configurar los parámetros antes de ser ejecutado. Se puede modificar la dirección IP de destino, el número de paquetes, el tamaño de los paquetes, la velocidad de transmisión, el protocolo (ICMP o UDP) y los 24 bytes del preámbulo, CRC e *inter-frame gap* mencionados en la sección 2.4. La velocidad de transmisión puede ser especificada por el usuario escribiendo la cifra en Mbit/s. Si se escribe 0 el programa envía los paquetes a la máxima velocidad posible. Un ejemplo de ejecución sería:

./QoStool 192.168.1.1 -l 10 -s 50 -p 1472 –i 24 —-icmp.

Es decir, se envían a la dirección IP 192.168.1.1 50 paquetes ICMP eco *request* con un tamaño de 1514(1472 bytes + las cabeceras) a una velocidad de 10 Mbit/s y teniendo en cuenta los 24 bytes del preámbulo, el CRC y el *interframe-gap* (figura 3.7).

La figura 3.7 muestra un ejemplo de ejecución del programa QoStool:

```
redes@lubuntu:~/Descargas/5-11-2021$ sudo ./QoStool 192.168.1.1 -l 50 -s 10 -p 1472 -i 24 --icmp
Target: 192.168.1.1
Payload: 1472 Bytes
packets: 50
Speed: 10.000000 Mbit/s
Type: ICMP
50 packets sent
********************************
          MEASURES
********************************
Throughput: 9.657013 Mbit/s
RTT min/average/max/time: 1.006000 1.345140 2.694000 67.257000 microseconds
Jitter: 0.374987 microseconds
Mean bandwidth: 9.439424 Mbit/s
General bandwidth: 9.481760 Mbit/s
Bandwidth desviation: 4.735729 Mbit/s
Bandwidth: min/average/max: 4.329346 9.439424 33.164420  Mbit/s
Received packets: 50
Lost packets: 0  0.00%
Program finished successfully
```

Figura 3.7: Ejemplo de ejecución del programa QoStool. Encima de *measures* los parámetros de la ráfaga. Debajo los parámetros QoS medidos.

Una vez se han establecido los parámetros y ejecutado el programa, hay que inicializarlo. Dentro de la libpcap la función pcap_open_live [21] permite abrir una interfaz para poder capturar los paquetes ICMP de respuesta y tener precisión en el tiempo de llegada.

```
if((per = pcap_open_live(interface,1514,1,1000,errbuf))==NULL)
    {
        printf("%s\n",errbuf);
        return 1;
    }
```

Figura 3.8: Función pcap_open_live. Dentro del paréntesis los argumentos de la función: interface, 1514,1 y errbuf.

El primer argumento es la cadena con el nombre de la interfaz que se quiere abrir(eth0,enp9s0..), obtenida con la función *pcap_lookupdev* [21]. El segundo argumento establece los bytes guardados por cada paquete. El 1 determina que la interfaz se va a abrir en modo promiscuo. El cuarto argumento especifica el tiempo de espera del buffer de paquetes, en este caso se ha escogido un tiempo grande para que no interfiera en las medidas. Por último, errbuf es un *array* que guardará el mensaje de error en el caso de que lo haya. La función devolverá el descriptor al archivo. pcap, o NULL en caso de error.

3.3.2 Construcción y envío de paquetes

Lo primero es construir los paquetes de acuerdo a los parámetros especificados. Para ello, se han creado dos funciones independientes: BuildICMP y BuildUDP, que se encargan de generar los paquetes ICMP y UDP respectivamente.

3.3.2.1 ICMP

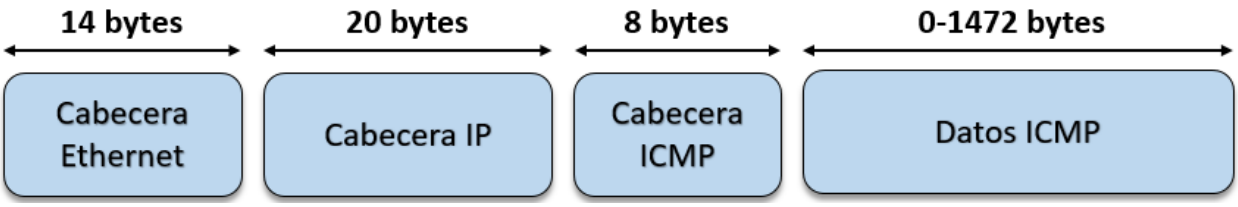

Figura 3.9: Estructura de un paquete ICMP de tipo eco.

Primero se crea un paquete vacío que tendrá de tamaño de la suma del ICMP *data* (parámetro especificado por el usuario) y de las cabeceras (42 bytes) (figura 3.9). A continuación, utilizando la estructura de la librería netinet/ether.h [23] se rellenan los campos de la cabecera ethernet: la MAC de origen, MAC de destino y el tipo (0x0800 o tipo IPv4). La MAC de destino se obtiene a través de mensajes ARP con la dirección IP de destino (parametro especificado). Hay que mencionar también que el programa puede enviar a direcciones Ip que se encuentren fuera de la LAN en cuyo caso la MAC de destino será la del router por defecto.

```
//ETHERNET_HEADER
eh->ether_shost[0] = Source_mac[0];
eh->ether_shost[1] = Source_mac[1];
eh->ether_shost[2] = Source_mac[2];
eh->ether_shost[3] = Source_mac[3];
eh->ether_shost[4] = Source_mac[4];
eh->ether_shost[5] = Source_mac[5];

eh->ether_dhost[0] = Dest_mac[0];
eh->ether_dhost[1] = Dest_mac[1];
eh->ether_dhost[2] = Dest_mac[2];
eh->ether_dhost[3] = Dest_mac[3];
eh->ether_dhost[4] = Dest_mac[4];
eh->ether_dhost[5] = Dest_mac[5];

eh->ether_type = htons(ETHERTYPE_IP);
```

Figura 3.10: Construcción de la cabecera ethernet. En la figura se puede ver como se rellenan los tres campos de la cabecera ethernet: el origen (Source_mac), el destino(Dest_mac) y el tipo (ETHERYPE_IP)

De forma similar se rellena la cabecera IP. En este caso los campos se rellenan usando una estructura recogida en la librería netinet/ip.h [24]. Para implementarlo se uso el siguiente codigo:

```
//IP HEADER
ip->version = 4;
ip->ihl = 5;
ip->tos = 0;
ip->tot_len = htons (packet_size - sizeof(struct ether_header));
ip->id = getpid();
ip->frag_off = 0;
ip->ttl = 255;
ip->protocol = IPPROTO_ICMP;
ip->check = in_cksum ((u16 *) ip, sizeof (struct iphdr));
ip->saddr = saddr;
ip->daddr = daddr;
```

Figura 3.11: Construcción de la cabecera IP. En la figura se puede ver como se rellenan los campos de la cabecera IP: versión, IHL, tipo de servicio, longitud, identificación, fragmentación, tiempo de vida, protocolo, checksum, dirección de origen y de destino

De arriba abajo según la figura 3.11: la versión de IP es IPv4 (4), el tamaño de la cabecera IP es de 5x32 bits o 20 bytes, el tipo de servicio(ToS) servicio rutinario (0x000), la longuitud se halla restando el tamaño total del paquete y la cabecera ethernet (14 bytes), para el identificador se utilizara el ID del proceso. La posición de fragmento es 0 ya que el paquete no se divide. El tiempo de vida es el máximo posible, 255. El tipo de protocolo es ICMP o tipo 1. El checksum se halla con la función *in_cksum*. La dirección IP de origen se obtiene al principo del programa. Por último, la dirección IP de destino es especificada por el usuario.

En el caso de la cabecera ICMP (figura 3.12) se utilizó la estructura de la librería netinet/ip_icmp.h [25].

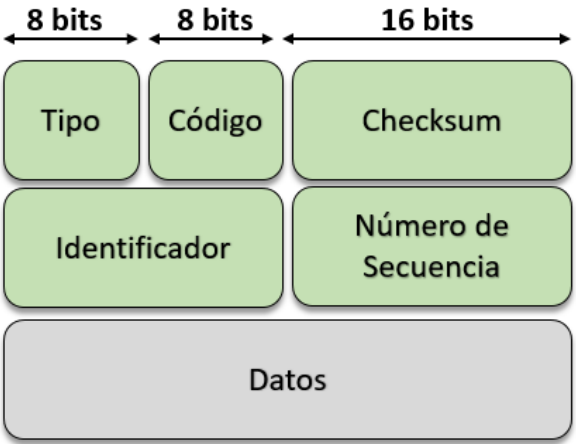

Figura 3.12. Esquema de la cabecera ICMP. Los campos son: el tipo, el código, el checksum, el identificador y el número de secuencia.

Para implementarlo se uso el siguiente codigo:

```
//ICMP HEADER
icmp->type = ICMP_ECHO;
icmp->code = 0;
icmp->un.echo.sequence = htons(sequence);
icmp->un.echo.id = getpid();
icmp->checksum = 0;
```

Figura 3.13: Construcción de la cabecera ICMP. Captura de pantalla del programa QoStool. En la captura se puede ver como se rellenan los campos de la cabecera ICMP representados en la figura 3.12.

De arriba abajo según la figura 3.13: el tipo es ICMP eco o tipo 8; el código debe ser 0; el número de secuencia es una variable (sequence) que va aumentando. Así, este número sería 1 para el primer paquete, 2 para el segundo, 3 para el tercero y así sucesivamente.De esta manera, en caso de que se pierda un paquete se puede saber exactamente cual es. El número de identificación al igual que para la cabecera IP, será el id del proceso. El *checksum* se inicializa a cero. Por último, se rellena el campo de datos con ceros de acuerdo con el tamaño especificado por el usuario (payload_size) y se calcula el checksum de la cabecera icmp (figura 3.14).

```
//DATA
memset(packet +sizeof(struct ether_header)+ sizeof(struct iphdr) + sizeof(struct icmphdr), 0x00, payload_size);
icmp->checksum = in_cksum((unsigned short *)icmp, sizeof(struct icmphdr) + payload_size);
```

Figura 3.14: Calculo del checksum. La función memset añade al paquete tantos como valor tenga *payload_size*. In_cksum es una función que calcula el *checksum*.

3.3.2.2 UDP

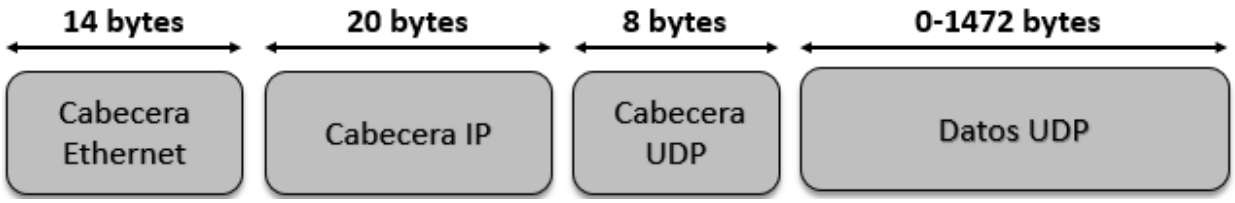

Figura 3.15: Estructura de un paquete UDP.

La cabecera ethernet e IP se implementan de la misma forma que en el apartado anterior, lo único que cambia es el protocolo de ip. En lugar de ICMP eco será UDP o tipo 17 (0x11 en hexadecimal) [26].

Para el caso de UDP no se utilizó ninguna biblioteca, sino que se creó la siguiente estructura:

```
struct udp_hdr
{
uint16_t src_port;
uint16_t dst_port;
uint16_t length;
uint16_t checksum;
};
```

Figura 3.16: Estructura de la cabecera UDP. Captura de pantalla del programa QoStool. Los campos son el puerto de origen, de destino, la longitud y el checksum

Para rellenar los campos se implemento el siguiente código:

```
//UDP HEADER
udp->src_port = htons(0x3039);
udp->dst_port = htons(0x829A + (uint8_t)j);
udp->length = htons(packet_size-sizeof(struct ether_header)-sizeof(struct iphdr));
udp->checksum = 0x0000;
```

Figura 3.17: Construcción de la cabecera UDP. Captura de pantalla del programa QoStool. En la captura se puede ver como se rellenan los campos de la figura 3.16.

De arriba abajo según la figura 3.17, el puerto de origen es 12345(0x3039 en hexadecimal), el puerto de destino es un puerto entre 33434(0x829A) y 33523 (se incrementa con la variable j). El motivo de la elección de los puertos viene explicado en la sección 3.2.2. El tamaño será el del paquete menos las cabeceras ethernet e Ip. Por ejemplo, si el usuario establece 1472 bytes de datos, la longuitud serían 1514 bytes (cabeceras + datos) – 34bytes (cabeceras ethernet e Ip), es decir, 1480 bytes. El *checksum* se inicializa a cero. Finalmente, se rellena el campo de datos con ceros de acuerdo con el tamaño especificado por el usuario (payload_size) y se calcula el *checksum* de la cabecera UDP con la función (tal y como se hacía para ICMP [figura 3.14])

3.3.2.3 Pcap_inject

Para enviar tráfico a la red se utiliza la función *pcap_inject* de la libpcap. Esta función permite enviar paquetes a través de la interfaz que se ha abierto en la fase de inicialización (figura 3.8).

```
if (pcap_inject(per,packet,packet_size) != packet_size)
{
    perror("send failed\n");
    return 1;
}
```

Figura 3.18: Función pcap_inject. Captura de pantalla del programa QoStool. Entre paréntesis, los argumentos de la función.

El primer argumento (per) es el descriptor *pcap* que se ha abierto en la fase de inicialización (sección 3.3.2) con *pcap_open_live*. *Pcap_inject* enviará los paquetes a la red a través de esta interfaz. El segundo argumento es la trama que se va a transmitir, en este caso sería el paquete ICMP o UDP que se ha construido en las secciones anteriores. Por último, el tercer argumento (packet_size) es un entero que representa el número de bytes que tiene la trama a enviar. El retorno de esta función es el número de bytes enviados. Por ello, se comprueba que el número de bytes enviados se corresponde al tamaño del paquete, en caso contrario se trataría de un error.

3.3.2.4 Bucle de envío

Además de construir los paquetes, es indispensable que los paquetes se manden de forma seguida, la separación entre paquetes debe especificarla el usuario. Para ello, en este programa se ha implementado un bucle encargado de enviar la ráfaga de paquetes (Figura 3.19).

```
while (sent < train_len)
 {

   gettimeofday(&tv, NULL);

   if(sendPacket(packet[sent])==1){
     printf("Error al enviar el paquete\n");
   }

   RTT[sent] = tv.tv_usec;
   ++sent;
   nanosleep(&interval,NULL);
 }
```

Figura 3.19: Bucle para enviar los paquetes. Captura de pantalla del programa QoS. El bucle se recorre tantas veces como paquetes se van a enviar, el número de paquetes es especificado por el usuario a través de la variable train_len.

En la figura 3.19 La función sendPacket contiene a *pcap_inject* tal y como muestra la figura 3.18, y es la encargada de enviar los paquetes. El argumento es el paquete que se va a enviar. Para establecer una separación desterminada entre los paquetes se ha utilizado la funcion *nanosleep*. Así, cuando el bucle pasa por esta función se detendra un tiempo en nanosegundos específico (*interval*) dejando un espacio entre los paquetes. La variable *interval* se halla a partir de la velocidad de transmisión especificada por el usuario.

Sin embargo, este método no es perfecto, ya que no se tiene en cuenta el tiempo de ejecución del resto de funciones del bucle, incluyendo el propi *nanosleep*. Para que sea lo más preciso posible se simplificó al máximo el bucle. Por ejemplo, los paquetes no se construyen en el bucle sino que se ha creado una lista (packet [figura 3.19]) que contiene en cada elemento un paquete. Además, se contempló el uso de otras funciones como *sleep o usleep*, pero *nanosleep* tiene una mejor resolución

y no interacciona con otras señales [27]. En este bucle también se mide y se guarda el tiempo de salida de los paquetes con *gettimeofday* y *RTT[sent]* respectivamente (figura 3.19), para posteriormente poder estimar el RTT.

Como se mencionó en apartados anteriores, el programa permite al usuario establecer la velocidad de transmisión (*speed* = N) o enviar a la velocidad máxima posible (*speed* = 0). Para el caso de velocidad máxima, se ha implementado un bucle alternativo. Este bucle es como el de la figura 3.19 pero sin la función nanosleep, de manera que este se recorre a la velocidad que permita la CPU.

3.3.3 Extracción de datos

Para capturar los mensajes el programa debe estar escuchando la interfaz a la vez que se envían los mismos. Para ello, se creó un segundo hilo con la función *pcap_loop* de la libpcap [21].

```
void * Capture(void *arg)
{
    pcap_loop(per,-1,attendPacket,NULL);
    return NULL;
}
```

Figura 3.20: Instrucción *Pcap_loop*. Captura del programa QoSloop. *Capture* es la función del hilo que se ha creado.

En la figura 3.20 el primer argumento de la función *pcap_loop* es el descriptor pcap que se ha abierto anteriormente con pcap_open_live. El segundo argumento es el número de paquetes que se van a capturar, -1 indica ilimitados. La función attendPacket se ejecutará cada vez que se lea un paquete. *Pcap_loop* devuelve -1 en caso de error, -2 si el bucle es interrumpido por otra función y otro valor si el paquete se lee correctamente. En el caso de que el número de paquetes se sobrepase puede retornar 0, pero es imposible en este caso [21].

La función attendPacket (figura 3.21) es la encargada de filtrar los paquetes deseados de la red, y de guardar los tiempos de llegada de los paquetes. El primer argumento son los datos auxiliares del usuario que no se utiliza en este programa. *Pcap_pkthdr* es una estructura que contiene datos del paquete como la cabecera, el *timestapmp* o la longitud. De todos ellos, solo es relevante el *timestamp*, el resto ya son conocidos. Por último, *packet* es un bytearray que contiene el paquete.

```
void attendPacket(u_char *user,const struct pcap_pkthdr *h,const u_char *packet)
```

Figura 3.21: Cabecera de la función attendPacket. Captura del programa QoStool. Esta función se encuentra dentro del bucle pcap_loop figura 3.20.

Antes de filtrar los paquetes, se extrae de ellos los campos necesarios para identificar si es el paquete deseado. Así, se obtiene la MAC de destino, el tipo ethernet, el tipo ip, el tipo ICMP, el puerto de destino y el número de secuencia.

```
if(memcmp(dest,Source_mac,6) ==0 && auxTipo == 0x0800 && IProto == 0x01 && ICMPtype == 0x00)

  TableTime[ntohs(Sequence_number)] = (h->ts.tv_usec)*1.0;
```

Figura 3.22: Filtro de los paquetes ICMP. Captura del programa QoStool.

En la figura 3.22 al principio se comprueba que la MAC de destino es igual a la dirección de origen del paquete ICMP de petición que se envío, es decir, al *host* desde el que se utiliza el programa. El tipo de ethernet es el IPv4 (0x0800). El 0x01 se corresponde al protocolo ICMP. Y debe ser un ICMP tipo eco de respuesta (0x00).

También, en la figura 3.22 se muestra cómo se han guardado los tiempos de llegada. Para ello, se ha creado una lista vacia *(TableTime)* donde cada paquete será guardado en su lugar correpondiente utilizando el número de secuencia (*Sequence_number*). De esta manera, si, por ejemplo, el paquete número 8 no llega, su lugar en la lista quedará vacio puediendo identificar los paquetes que falten. Además, los tiempos de llegada se guardan en microsegundos (*ts.tv_usec*).

```
if(memcmp(dest,Source_mac,6) ==0 && auxTipo == 0x0800 && IProto == 0x01 && ICMPtype == 0x03)

  TableTime[htons(Dest_port)-33434] = (h->ts.tv_usec)*1.0;
```

Figura 3.23: Filtro de los paquetes ICMP de puerto inalcanzable. Captura del programa QoStool.

La única diferencia respecto al filtro anterior es el tipo de ICMP, para el caso de los mensajes de puerto inalcanzable se correponde al tipo 3. Los datos también se guardan de manera diferente, hay que recordar que el protocolo UDP no tiene número de secuencia, por lo tanto, los tiempos de llegada se guardarán según el puerto de destino (figura 3.23).

3.3.4 Cálculo de los parámetros QoS

En este apartado se explicará cómo se han realizado los cálculos de los parámetros QoS. Hay que mencionar que el código que se va a mostrar a continuación está simplificado.

Perdida de paquetes. Tal y como se explica en la sección anterior, al principio del programa se crea una lista con todos los valores a cero y se va rellenando con el tiempo de llegada de los paquetes. Para ver cuántos paquetes se han perdido, se recorre la lista mediante un bucle y en caso de que el lugar de un paquete sea 0 se contará como perdido. (figura 3.24)

```
for(i=0;i<sent;i++){
    if(TableTime[i]== 0.0){
            packet_loss++;
    }
}
```

Figura 3.24: Bucle de perdida de paquetes. Captura del programa QoStool. Se recorre la lista *TableTime* mediante un bucle *for* y en el caso de que el tiempo sea cero se suma uno a la variable de paquetes perdidos(*packet_loss*).

RTT. En este caso, un bucle recorrerá la lista que contiene los tiempos de llegada descartando los paquetes que se hayan perdido. En caso contrario, se calcula el RTT restando el tiempo de llegada menos el de salida. El tiempo de salida de cada paquete se mide y se almacena en una lista justo antes de enviarlos (figura 3.19). Los valores se guardan en otra lista para posteriormente calcular el RTT máximo, el mínimo, el valor medio y el tiempo total.

```
for(i=0;i<sent;i++){

    if(TableTime[i]!= 0.0){

        TableRTT[i] = TableTime[i]-RTT[i];
    }
}
```

Figura 3.25. Cálculo del RTT. Captura del programa QoStool. Se recorre la lista de los tiempos de llegada de los paquetes *TableTime* mediante un bucle *for*. Si no se pierde, se calcula el RTT restando el tiempo de llegada (*TableTime*) menos el tiempo de salida (*RTT]*). El RTT se almacena en la lista *TableRTT.*

Jitter. Se calcula como la desviación típica del RTT (figura 3.26).

```
for(i=0;i<sent;i++){

    if(TableRTT[i] != 0.0){

        sumstadardRTT += pow(TableRTT[i]-RTTaverage, 2);
        N++;
    }
}

deviation_RTT = sqrt(sumstadardRTT/N);
```

Figura 3.26. Cálculo del Jitter. Un bucle recorre la lista *TableRTT* que contiene el RTT de cada paquete. En caso de que no se haya perdido (distinto de 0.0). Se aplicará la fórmula de la desviación típica.

Throughput. Se halla midiendo el tiempo de salida que hay entre los paquetes. El tiempo de salida se obtuvo en el bucle de envío. (figura 3.27)

```
for(i=0;i<sent-1;i++){
        TableThroughput[i] = RTT[i+1]- RTT[i];
}
```

Figura 3.27: Cálculo del *Throughput*. El bucle recorre la lista *RTT* que contiene los tiempos de salida. El tiempo entre paquetes se halla restando el tiempo de salida de un paquete al del anterior. (*RTT[i+1]-RTT[i]).* Los datos se guardan en la lista T*ablethorughput* para posteriormente calcular el throughput promedio.

Ancho de banda: Es el parámetro en el que más se ha enfocado este trabajo por la poca eficiencia de otros métodos (sección 2). Dada su importancia, se ha optado por hallarlo de dos formas diferentes. La primera, calculando la media de todos los tiempos entre paquetes. La segunda, dividiendo el número de bytes enviados entre el tiempo que hay entre el primer paquete y el último. Por ejemplo, si enviamos cuatro paquetes de 1514 bytes y los tiempos de llegada son 0ms, 2 ms, 5 ms y 7 ms. Con el primer método se haría la siguiente cuenta: (1514x8) / 2.33ms = 5.2 Mbit/s. Con la segunda forma sería: (1514x8x4) / 7ms = 6.9Mbit/s. El primer método calcula el ancho de banda medio (figura 3.28) y el segundo el ancho de banda total (figura 3.29).

```
for(i=0;i<(sent-1);i++){
    if(TableTime[i] != 0.0 && TableTime[i+1] != 0.0){
        TableBandwidth[i] =TableTime[i+1]-TableTime[i];
    }
}
```

Figura 3.28: Calculo del ancho de banda medio. Si un paquete no se ha perdido (*TableTime[i]= 0.0*) y el siguiente tampoco, se halla el tiempo entre paquetes (*Tabletime[i+1]-Tabletieme[i])* y se guarda en la lista *TableBandwidth.*

El problema del segundo método es cuando se pierden muchos paquetes, esto es aún peor cuando los que se pierden son el primero o el último paquete. Por ello, se ha implementado un algoritmo que busca el primer paquete (*first*) y el último (*last*) que no se hayan perdido (figura 3.29).

```
Total_BW = ((sent-packet_loss)*8.0*packet_size)/(TableTime[last] - TableTime[first]);
```

Figura 3.29: Calculo del ancho de banda total. Se aplica la formula Nºbits/tiempo: paquetes enviados (*sen t- packet_loss*) x tamaño de los paquetes (*packet_size*) / tiempo (*TableTime[last]-TableTime[first]*)

El resto de medidas como el ancho de banda máximo, mínimo y la desviación típica se calculan a partir de los datos de la lista *TableBandwidth* en la figura 3.28.

3.3.5 Salida

Para finalizar el programa se interrumpe el bucle de captura (*pcap_loop*) con la función de la libpcap *pcap_breakloo*p cuyo único argumento es el descriptor pcap que se abrió al princio del programa. Por último, se cierra el descriptor pcap con la función *pcap_close*.

3.4 Conclusión

En la primera parte de este capítulo se ha comprobado en un entorno real que ambas técnicas se pueden implantar. Corroborando con la herramienta Wireshark como un router doméstico recibe los mensajes y responde a los mensajes de control. Incluso, se han conseguido obtener estimaciones de la capacidad del enlace con la herramienta Nping. Además, se ha visto que la herramienta token bucket puede simular bastante bien la capacidad máxima de un enlace. Esto es esencial a la hora de hacer pruebas. Finalmente, se ha explicado cómo se ha estructurado e implementado el programa, así como se ha calculado cada parámetro QoS.

4 Pruebas y resultados

4.1 Introducción

En este capítulo se analizarán los parámetros de calidad de servicio obtenidos utilizando ambas técnicas. Sobre todo, se comprobará la precisión con la que cada método estima la capacidad máxima del enlace. Para ello, se realizaron pruebas tanto en entornos reales como virtuales. En este apartado se explicará en detalle cada uno de los test que se han llevado a cabo, desde el entorno en el que se han hecho hasta los parámetros que se han seleccionado. Estos parámetros se escogieron de acuerdo a los principios comentados en la sección 2.4, de manera que las estimaciones sean lo más precisas posibles. Por último, se analizarán los resultados obtenidos.

4.2 Pruebas en entorno Real

Estas pruebas se llevaron a cabo en un entorno doméstico. Se conectó un ordenador directamente a un router a través de un cable Ethernet. Así, el ancho de banda es el máximo posible para esta red. Estas pruebas se realizaron en el mismo entorno que la sección 3.2, donde el cuello de botella estaba marcado por la tarjeta de red del ordenador y era de 100 Mbit/s.

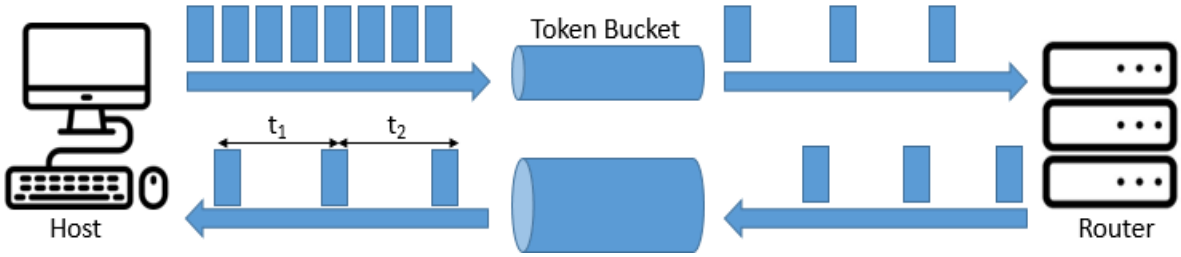

Figura 4.1: Esquema de las pruebas realizadas en entornos reales. El *host* contiene el programa QoStool y envía ráfagas de paquetes ICMP o UDP contra el router a una velocidad superior a la capacidad del enlace, que está limitada por Token bucket. Finalmente, el host mide el tiempo entre los paquetes que es proporcional al ancho de banda.

En este primer test se probaron ambas técnicas para distintos anchos de banda, con el fin de ver su precisión conforme va aumentando la capacidad del enlace. Como se mencionó anteriormente, se utilizó la herramienta Token bucket y se fue aumentando la capacidad del enlace en pasos de 5 Mbit/s. Se escribió esta línea en el terminal: *tc qdisc add dev enp9s0 root tbf rate 5mbit burst 12kbit latency 400ms.* El tamaño del cubo se limitó a 12 Kbit, de forma que solo pudiera entrar un paquete al mismo tiempo (12000/8 = 1500 bytes[La MTU]). Además, la latencia definida es relativamente grande (400ms) para que no se pierdan paquetes.

Se establecieron los siguientes parámetros: el tamaño de paquetes de 1514 bytes, la velocidad máxima de transmisión (marcada por la CPU) y el número de paquetes sería de 50. Se comprobó previamente que el número de paquetes no podía exceder de 80 ya que el router los tiraba para evitar posibles ataques de denegación de servicio. El código que se escribió en el terminal fue el siguiente: ./QoStool 192.168.1.1 -l 50 -s 0 -p 1472 –i 24 –icmp|udp. Por cada ancho de banda se realizaron tres medidas y se halló la media. Este procedimiento se hizo tanto para ICMP como para UDP.

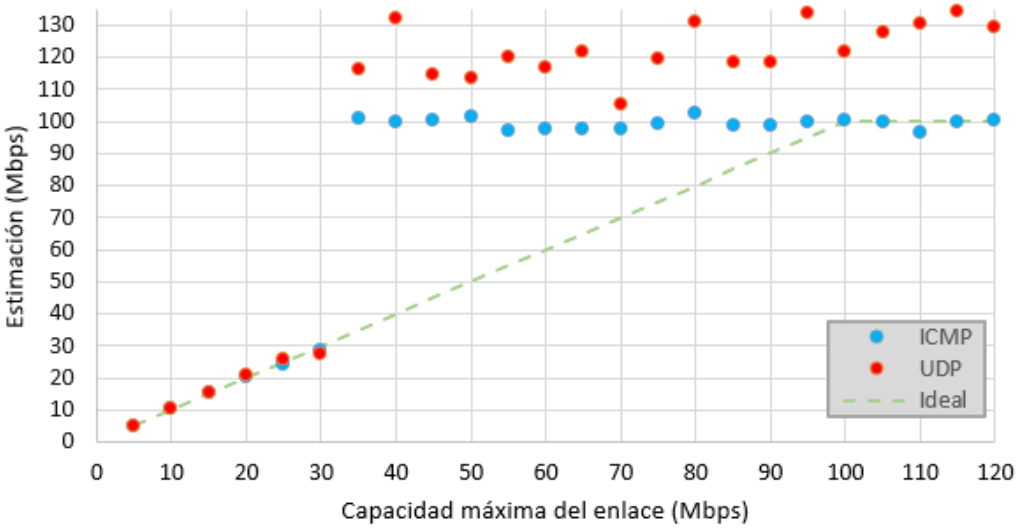

Figura 4.2. Gráfica del test 1. El eje Y representa las medidas que se han obtenido, y el eje X es la capacidad máxima del enlace que está limitada por Token bucket. A partir de 100Mbit/s el cuello de botella pasa a ser la tarjeta de red del ordenador. La línea discontinua verde representa las estimaciones que se deberían obtener.

En la figura 4.2 se puede ver como para tasas menores de 30Mbit/s ambas ténicas hacen estimaciones bastante aproximadas. El error medio de ICMP para capacidades menores de 30 Mbits es 2.64%, mientras que para UDP es 3.23%.

Mbps	5	10	15	20	25	30	35	40	45	50	55	60
ICMP	5.00	10.23	15.42	20.54	24.20	28.57	101.00	99.80	100.47	101.56	96.94	97.57
UDP	5.06	10.27	15.21	20.63	25.77	27.67	116.35	132.16	114.31	113.56	120.21	116.46
Mbps	65	70	75	80	85	90	95	100	105	110	115	120
ICMP	97.47	97.57	99.31	102.70	98.64	98.90	99.58	100.13	99.60	96.56	99.93	100.06
UDP	121.74	105.34	119.37	131.16	118.48	118.55	133.76	121.46	127.55	130.70	134.22	129.10

Figura 4.3: Resultados test 1. En gris la capacidad máxima del enlace limitada por el token bucket, en azul y rojo las estimaciones obtenidas.

Sin embargo, la figura 4.2 muestra como a partir de 30 Mbits las estimaciones son mucho mayores de lo que deberían. Esto se debe a que el router encola los paquetes y responde a varios a la vez a máxima velocidad (aproximadamente 100Mbit/s) en lugar de hacerlo de uno en uno, figura 4.4.

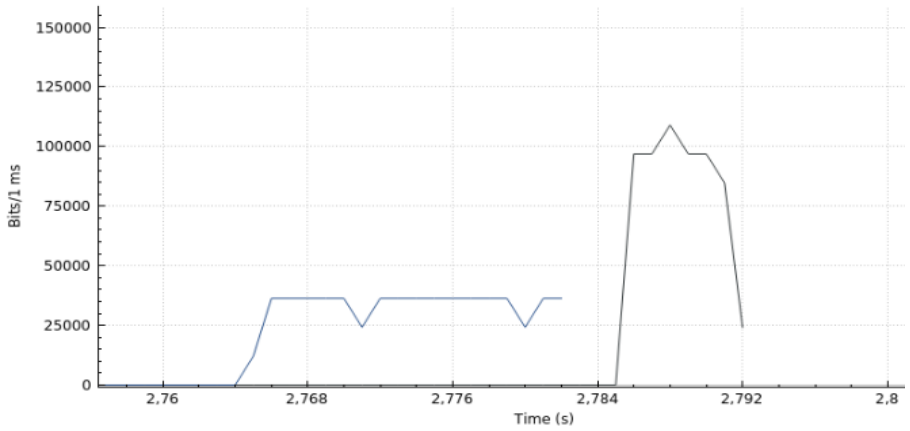

Figura 4.4. Captura del test 1 en Wireshark. en azul las peticiones ICMP y en negro las respuestas ICMP del router. El host envía a aproximadamente 40Mbps y el router responde a 100Mbps aproximadamente.

Para evitar este fenomeno se restringió la capacidad del enlace también para la entrada. No obstante, no es posible limitar el ancho de banda para la salida y la entrada al mismo tiempo. Por ello, se ha utilizado una técnica llamada espejo (*mirroring*) [28], que consiste en crear una interfaz virtual que hace de espejo de la interfaz física. Así, se puede restringir la entrada de la interfaz virtual con Token Bucket. Además, el programa fue modificado para que escuchara la interfaz virtual. Para este segundo test se han utilizado los mismos parametros que para el test anterior.

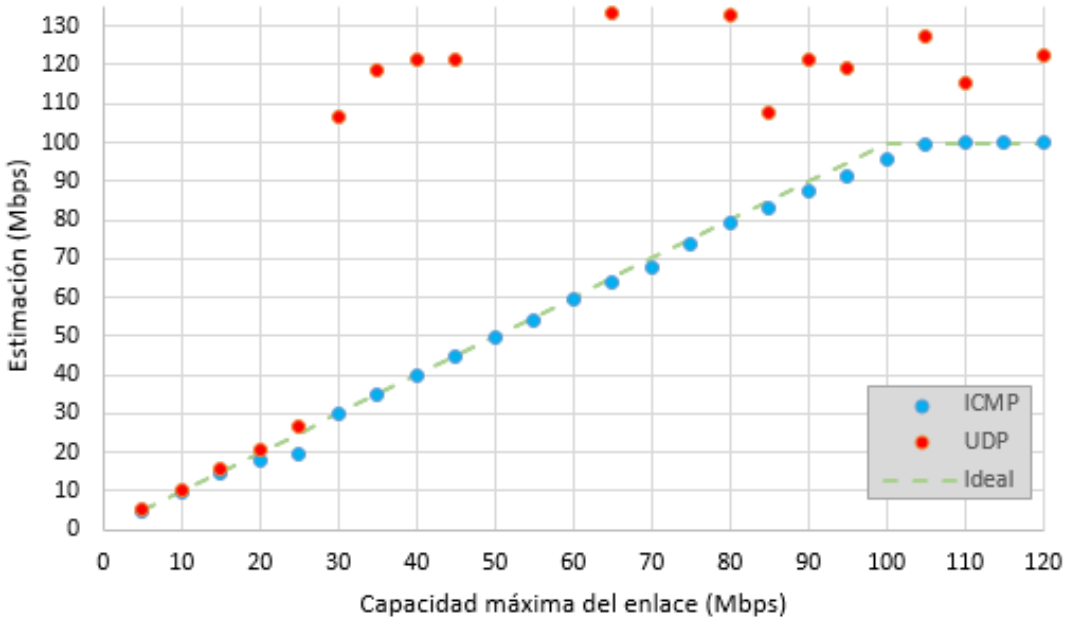

Figura 4.5: Gráfica del test 2. El eje Y representa las medidas que se han obtenido, y el eje X es la capacidad máxima del enlace que está limitada por Token bucket. A partir de 100Mbit/s el cuello de botella pasa a ser la tarjeta de red del ordenador. La línea discontinua verde representa las estimaciones que se deberían obtener

En la figura 4.5 se puede ver como la técnica de paquetes ICMP hace estimaciones bastante aproximadas, con un error medio de 3.16% hasta 100Mbit/s.También, se observa que cuando el token bucket supera los 100Mbit/s y es la propia red quien limita la velocidad de transmisión, el error medio es 0.21% (cuatro útlimos valores figura 4.6). Así, se puede ver que a pesar de ser una herramienta que restringe bastante bien el ancho de banda, no es perfecta y tiene influencia en los resultados.

Mbps	5	10	15	20	25	30	35	40	45	50	55	60
ICMP	5.00	9.96	14.59	17.85	19.70	29.84	34.83	39.75	44.67	49.48	54.15	59.61
UDP	5.07	10.33	15.92	20.41	26.78	106.67	118.46	121.37	121.10	155.61	148.74	149.87
Mbps	65	70	75	80	85	90	95	100	105	110	115	120
ICMP	63.64	67.55	73.75	79.09	82.99	87.57	91.29	95.77	99.60	99.76	99.93	99.90
UDP	133.28	150.07	135.75	133.04	107.73	121.49	119.01	151.25	127.55	115.26	135.22	122.24

Figura 4.6: resultados obtenidos en el test 2. en gris la capacidad máxima del enlace limitada por el token bucket, en azul y rojo las estimaciones obtenidas.

Por otro lado, el metodo basado en paquetes UDP presenta un comportamiento similar al test anterior (figura 4.2). A partir de 30 Mbit/s las medidas presentan un error muy grande. Por ejemplo, cuando Token bucket limita el ancho de banda a 40Mbit/s la estimación es de 121.368 Mbit/s, es decir, tiene un error de un 103,42%. Esto sucede por la coalescencia de los paquetes (*paket-coalescing*), [29] , [30]. El router agrupa los paquetes y los envía a la vez, limitando así el número de interrupciones. De esta forma se reduce la cantidad de procesamiento requerido. Sin embargo, a la hora de medir hace que las estimaciones no sean precisas. En la figura 4.8 se puede ver como se agrupan los paquetes, por ejemplo, los paquetes 36, 37, 38, 39, 40, 41 y 42 se transmitieron a la vez.

No.	Time delta from prev	Proto	Length	Info
35	0.000002422	ICMP	70	Destination unreachable (Port unreachable)
36	0.000543548	ICMP	70	Destination unreachable (Port unreachable)
37	0.000005763	ICMP	70	Destination unreachable (Port unreachable)
38	0.000002750	ICMP	70	Destination unreachable (Port unreachable)
39	0.000002609	ICMP	70	Destination unreachable (Port unreachable)
40	0.000002455	ICMP	70	Destination unreachable (Port unreachable)
41	0.000002510	ICMP	70	Destination unreachable (Port unreachable)
42	0.000002362	ICMP	70	Destination unreachable (Port unreachable)
43	0.000586936	ICMP	70	Destination unreachable (Port unreachable)
44	0.000005700	ICMP	70	Destination unreachable (Port unreachable)
45	0.000002766	ICMP	70	Destination unreachable (Port unreachable)

Figura 4.7: paquetes ICMP de puerto inalcanzable. Captura de Wireshark la segunda columna muestra los tiempos que hay entre los paquetes ICMP de puerto inalcanzable.

Este problema se intentó solventar con la herramienta ethtool [31] que permite configurar las opciones de coalescencia del ordenador. No obstante, no todos los ordenadores dejan cambiar estas opciones, y en este caso, no se pudo probar.

4.3 Pruebas en entornos Virtuales

Una vez se habían examinado ambas técnicas en un entorno doméstico, se decidió hacer más pruebas en entorno virtuales con el objetivo de ver lo que sucede con capacidades de enlace mayores de 100Mbit/s.

El primer test se realizó en *localhost*, para ello se modificó el programa para que fuera capaz de enviar y recibir paquetes, y de escuchar la interfaz virtual *loopback*. En este test los parámetros utilizados fueron los mismos que en los anteriores, la línea escrita en la terminal fue la siguiente: *./QoStool 127.0.0.1 -l 50 -s 0 -p 1472 –i 24 –icmp*. Tras tres mediciones, el ancho de banda medio obtenido fue de 31.581 Mbit/s. En la figura 4.9 se puede ver como la capacidad máxima del enlace es inferior a 40Mbit/s, siendo menor incluso que el ancho de banda máximo de los test anteriores (100Mbit/s). La figura 4.9 se ha obtenido Nping, escribiendo la siguiente línea en el terminal: *nping --icmp --icmp-type echo --delay 0 -c 45000 --data-length 1472 -H -N 127.0.0.1*

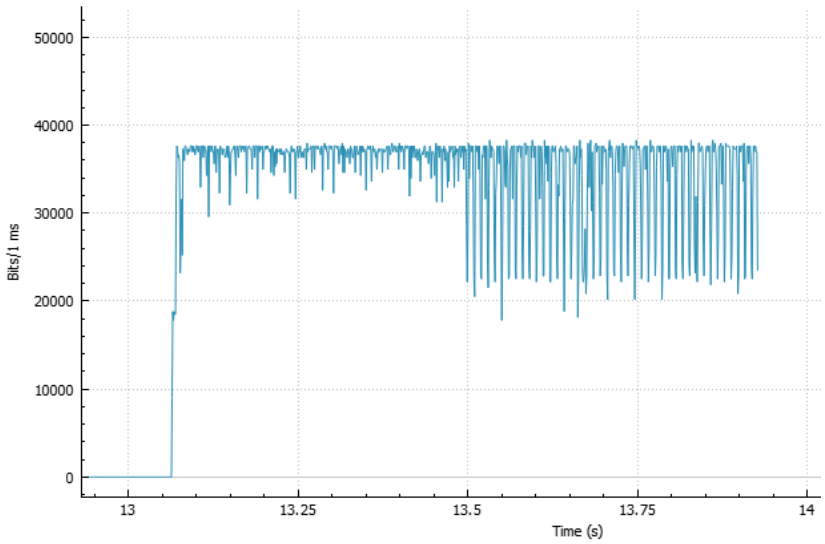

Figura 4.8: Gráfica de los paquetes ICMP de respueta en localhost. El eje X representa los bits por milisegundo y el eje Y el tiempo.

El segundo intento de test virtual, consistió en utilizar dos máquinas virtuales en el mismo ordenador con VMware (www.vmware.com/workstation). En una se ejecutó el programa que envió la ráfaga de paquetes contra la otra máquina. VMware tiene la opción de configurar la velocidad de transmisión de entrada y de salida de las máquinas virtuales. Sin embargo, los resultados obtenidos fueron dispares, desde 16Mbit/s hasta 272Mbit/s. Esto indica que simular la capacidad del enlace utilizando la herramienta VMware no es preciso. Estos resultados se comprobaron con Wireshark.

De esta manera, se optó finalmente por utilizar mininet [32], que es una herramienta que sirve para emular y prototipar redes. Tras arrancar mininet, se montarón dos pilas TCP/IP simulando dos equipos diferentes Al igual que para el VMware, las rafagas se transmitieron desde una máquina virtual hacia la otra. Y se realizaron las medidas de los parámetros QoS. Además, para restringir el

ancho de banda se utilizó la herramienta Token bucket, lo único que se hizo fue escribir la siguiente linea en el terminal h1(*host 1)* de mininet: *tc qdisc add dev enp9s0 root tbf rate 25mbit burst 12kbit latency 400ms* para 25Mbit/s.

El resto de parámetros fueron exactamente los mismos que para las pruebas en entornos reales: *./QoStool 10.0.0.2 -l 50 -s 0 -p 1472 –i 24 –icmp*. Donde 10.0.0.2 es la dirección IP del *host* contra el que se envían las ráfagas y 10.0.0.1 el *host* donde se ejecutó el programa. Hay que mencionar tambien, que para este test (figura 4.9) unicamente se probó la técnica basada en trenes de paquetes ICMP, ya que al enviar ráfagas de paquetes UDP solo recibieron paquetes de unos pocos puertos que ni si quiera eran contiguos, lo que hacía imposible realizar las medidas.

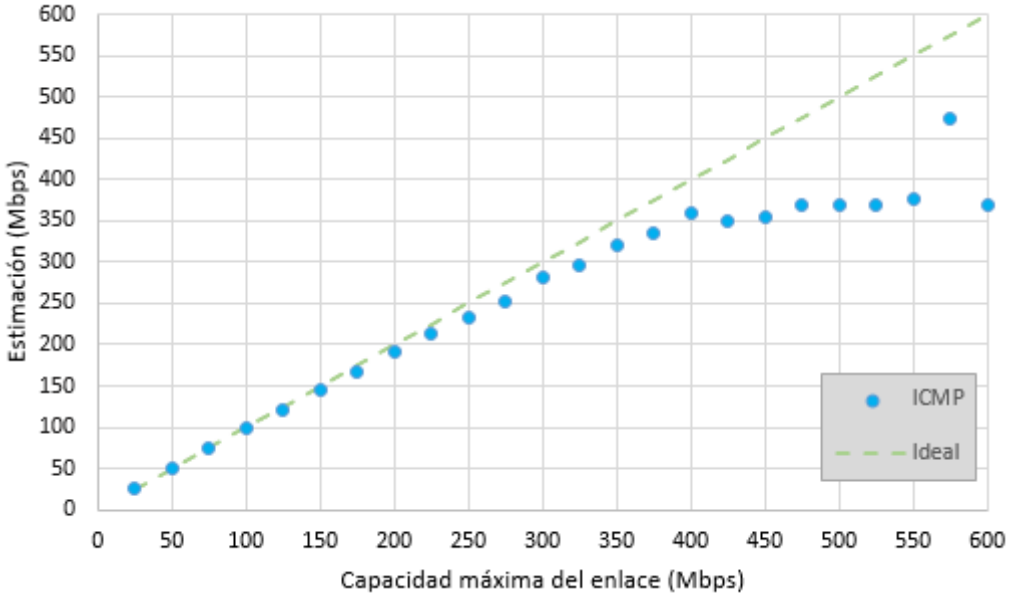

Figura 4.9: : Grafica del test 3.**.** El eje Y representa las medidas que se han obtenido, y el eje X es la capacidad máxima del enlace que está limitada por Token bucket. La línea discontinua verde representa las estimaciones que se deberían obtener.

La figura 4.9 muestra como a medida que la capacidad máxima del enlace aumenta, las medidas del ancho de banda son cada vez menos precisas. También, se puede ver como a partir de 400Mbit/s aproximadamente, las estimaciones parecen haber alcanzado su máximo que se encuentra entre 350 y 400 Mbit/s.

Mbps	25	50	75	100	125	150	175	200	225	250	275	300
ICMP	24.95	50.15	74.40	99.36	120.97	144.44	167.37	191.02	212.33	233.29	252.46	281.14
Mbps	325	350	375	400	425	450	475	500	525	550	575	600
ICMP	296.65	319.77	336.02	359.06	349.06	354.84	369.31	369.31	370.26	377.57	473.76	370.26

Figura 4.10: Resultados obtenidos en el test 3. en gris la capacidad máxima del enlace limitada por el token bucket, en azul las estimaciones obtenidas con ICMP.

En la figura 4.12 se muestra el error que se ha obtenido al realizar cada una de las medidas de ancho de banda. Viendo la gráfica, se puede corroborar lo mencionado anteriormente: a medida que aumenta la capacidad máxima del enlace las estimaciones son menos precisas. Además, se puede observar como a partir de 400 Mbit/s el error aumenta significativamente superando la barrera del 10 %. En otras palabras, a partir de 400 Mbps las medidas no pueden considerarse como validas ya que el error es alto. Para capacidades inferiores a 200 Mbps las estimaciones podrían considerarse como precisas, con un error de 2.22%.Y para capacidades inferiores a 400Mbit/s el error medio es 5.16%.

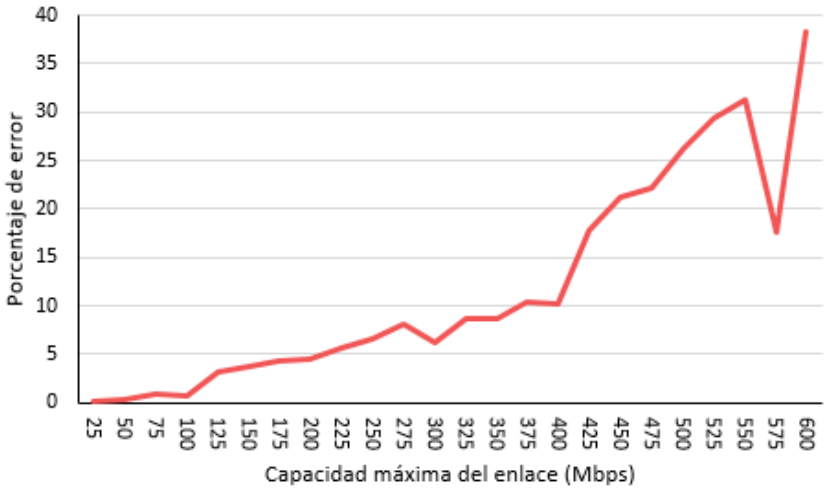

Figura 4.11: Gráfica del error en las estimaciones. El eje Y representa el porcentaje de error en las medidas, y el eje X es la capacidad máxima del enlace que está limitada por Token bucket.

De acuerdo con los datos de la figura 4.12. El error medio total es 11.94%. Y por tramos el error medio es 0.48% de 25 a 100 Mbps, 3.95% de 125 a 200 Mbps, 6.70% de 225 a 300 Mbit/s, 9.50% de 325-400 Mbit/s, 21.85% de 425 a 500Mbit/s y 29.18% de 525 a 600 Mbit/s.

Mbps	25	50	75	100	125	150	175	200	225	250	275	300
Error	0.20	0.30	0.80	0.64	3.22	3.71	4.36	4.49	5.63	6.68	8.20	6.29
Mbps	325	350	375	400	425	450	475	500	525	550	575	600
Error	8.72	8.64	10.40	10.23	17.87	21.15	22.25	26.14	29.48	31.35	17.61	38.29

Figura 4.12: Error obtenido en el test 3. En gris la capacidad máxima del enlace limitada por el token bucket, en rojo el porcenataje de error obtenido en las medidas.

A continuación, se hicieron estimaciones de la capacidad del enlace variando el número de paquetes enviados. El objetivo de este test es ver como mejora la precisión de las medidas cuando se aumenta el tamaño de la ráfaga. Esta prueba solo se ha podido hacer en un entorno virtual ya que por regla general los routers suelen rechazar ráfagas grandes de paquetes para evitar ataques.

La gráfica (figura 4.13) muestra como a medida que se incrementa el número de paquetes, la precisión de las medidas también aumenta. Así, el error medio para 50 paquetes es 14.12%, para 200 es 9.63%, para 500 es 6.14% y para 1000 es 4.87%.

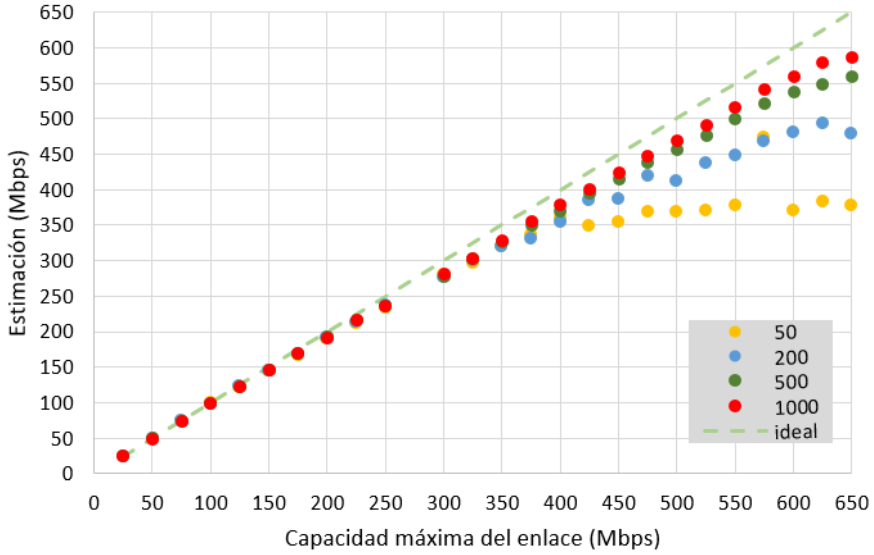

Figura 4.13: Gráfica del test 4. Se enviaron ráfagas de 50(amarillo), 200(azul),500(verde), 1000(rojo). La línea verde representa el ancho de banda que se debería medir.

La tabla de datos (figura 4.14) muestra cómo se obtuvieron resultados muy parecidos hasta 325 Mbps. Por ejemplo, con una capacidad máxima de enlace de 200 Mbps las cuatro ráfagas de paquetes han estimado que el ancho de banda es de 191 Mbps aproximadamente. Esto puede deberse a que la capacidad real del enlace sea 191 Mbps, ya que Token bucket no es perfecto. De hecho, en el manual de la herramienta en Linux se especifica que para tasas muy superiores a 1 Mbps se pierde precisión a la hora de restringir el ancho de banda [8].

	25	50	75	100	125	150	175	200	225	250	275	300	325
50	24.95	50.15	74.40	99.36	120.97	144.44	167.37	191.02	212.33	233.29	252.46	281.14	296.65
200	24.98	49.44	74.64	98.75	122.77	145.83	169.13	192.70	214.27	237.45	259.78	276.98	300.76
500	24.96	49.74	74.44	98.80	122.41	146.08	169.21	192.92	215.52	235.68	261.74	278.50	302.89
1000	24.90	49.42	74.57	98.80	122.78	146.17	169.58	191.00	215.51	236.90	258.99	281.54	303.08
	350	375	400	425	450	475	500	525	550	575	600	625	650
50	319.77	336.02	359.06	349.06	354.84	369.31	369.31	370.26	377.57	473.76	370.26	383.57	377.98
200	319.71	331.77	354.35	385.21	386.82	418.57	411.66	436.57	448.57	468.30	481.24	492.73	479.38
500	326.46	349.42	370.26	394.23	415.73	437.57	456.90	475.66	498.92	521.00	537.95	549.15	559.49
1000	327.57	355.48	378.41	401.31	423.49	446.92	468.48	491.46	515.22	541.66	558.36	579.19	587.02

Figura 4.14: Tabla con los datos del test 4. Las filas muestran el número de paquetes enviados y las columnas la capacidad del enlace

Por ello, se realizó otro test donde se medía la capacidad máxima del enlace sin restricciones. En este caso, el cuello de botella era la velocidad de procesamiento de la CPU, ya que dentro del propio ordenador se pueden alcanzar velocidades del orden de centenas de Gbit/s. El programa es capaz de medir la velocidad de transmisión de los paquetes, que esta restringida por la CPU. En otras palabras, el ancho de banda de la red lo marca la velocidad de transmisión de los paquetes (en la figura 4.15 es el Throughput) .

```
root@lubuntu:~/Descargas/5-11-2021# sudo ./QoStool 10.0.0.2 -l 1000 -s 0 -p 147
2 -i 24 --icmp
Target: 10.0.0.2
Payload: 1472 Bytes
packets: 1000
Speed: Max
Type: ICMP
1000 packets sent
*******************************
          MEASURES
*******************************
Throughput: 995.549802 Mbit/s
RTT min/average/max/time: 0.007000 0.008976 0.048000 8.976000 microseconds
Jitter: 0.001667 microseconds
Mean bandwidth: 997.783586 Mbit/s
General bandwidth: 983.196688 Mbit/s
Bandwidth desviation: 95.668399 Mbit/s
Bandwidth: min/average/max: 292.952381 997.783586 1230.400000  Mbit/s
Received packets: 1000
Lost packets: 0  0.00%
Program finished successfully
```

Figura 4.15: Ejemplo de ejecución del programa QoS para el test 5. Es la prueba número 6 de la figura 4.16. El throughput es la tasa de transferencia de los paquetes, y la medida que se ha tenido en cuenta para los resultados es el *Mean bandwidth*.

De esta manera, se enviaron 1000 paquetes de una maquina virtual a otra. El resto de parámetros fueron los mismos que para las pruebas anteriores. En la figura 4.16 se puede observar como las medidas han sido más precisas que en los test anteriores. El error medio de las 10 pruebas es de 0.35%, y el error máximo cometido es de 0.58%. Por ejemplo, en el test anterior se obtuvo que para 650 Mbit/s el error fue 9.69%.

Mbps	969.74	963.26	978.67	972.70	1003.24	995.55	960.06	865.39	982.80	964.25
ICMP	973.05	965.29	978.57	978.32	1007.23	997.78	965.06	869.42	986.13	968.38
Error	0.34	0.21	0.01	0.58	0.40	0.22	0.52	0.47	0.34	0.43

Figura 4.16: Resultados de las 10 pruebas realizadas sin restricciones. En gris la velocidad máxima de transmisión en Mbps, en azul las medidas y en rojo el error obtenido en porcentaje.

Además de las pruebas descritas anteriormente, se llevo a cabo un último test con el fin de ver qué sucede con el resto de los parámetros de calidad de servicio. Hay que recordar que el objetivo de este trabajo se buscar un método sencillo y preciso que utilice un único terminal para estimar los parámetros QoS.

En esta prueba se enviaron los paquetes contra un router de la red externa (WAN). Los parámetros utilizados fueron los mismos que para los test anteriores, excepto que se decidió aumentar el número de paquetes a 100. Tras varias pruebas se determinó que 100 era el número máximo de paquetes que se podían enviar sin que el router los rechazara. Los resultados se verificaron con Nping y Wireshark.

```
redes@lubuntu:~/Descargas/5-11-2021$ sudo ./QoStool 82.159.32.139 -l 100 -s 0 -p 1472 -i 24 --icmp
Target: 82.159.32.139
Payload: 1472 Bytes
packets: 100
Speed: Max
Type: ICMP
100 packets sent
********************************
          MEASURES
********************************
Throughput: 232.942067 Mbit/s
RTT min/average/max/time: 17.509000 42.448530 55.545000 4244.853000 microseconds
Jitter: 8.998214 microseconds
Mean bandwidth: 31.425004 Mbit/s
General bandwidth: 31.247098 Mbit/s
Bandwidth desviation: 37.925807 Mbit/s
Bandwidth: min/average/max: 1.823355 31.425004 138.247191  Mbit/s
Received packets: 100
Lost packets: 0  0.00%
Program finished successfully
```

Figura 4.17: Resultados del test 5. Es una captura del programa QoStool. Encima de *measures* los parámetros establecidos. *A* continuación, todos los parámetros QoS medidos.

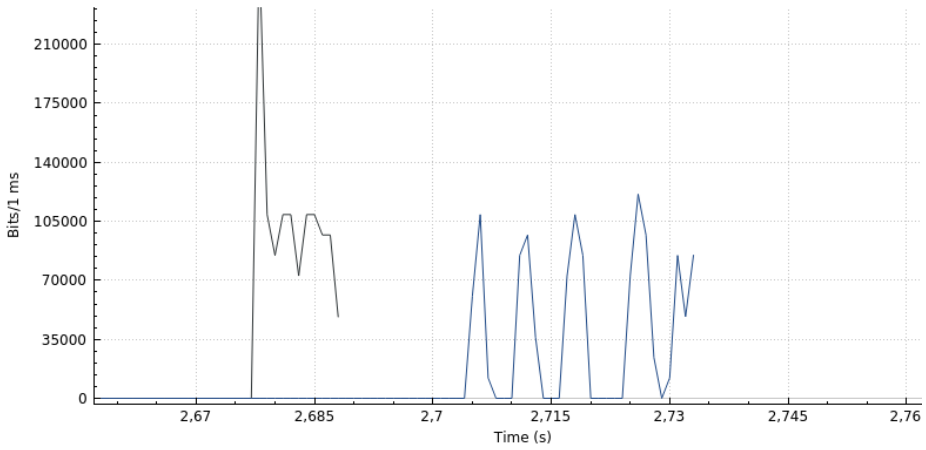

Figura 4.18: Gráfica del test 5.El eje Y representa los bits transmitidos por microsegundo y el eje X el tiempo en segundos. En negro los paquetes ICMP *request* y en azul los paquetes ICMP *reply*.

La figura 4.17 muestra los resultados y la figura 4.18 la gráfica. El throughput medido es 232,94 Mbps, en la gráfica se puede observar como al principio la red permite enviar a 232 Mbps, pero centésimas de segundo después este valor baja hasta 110Mbps de valor máximo (aproximadamente).

El RTT es 4244 microsegundos, que es mayor que el tiempo que se obtuvo en el entorno doméstico que fue de 67 microsegundos (figura 3.7). Esto es lógico puesto que hay mucha más distancia. Además, el jitter también ha aumentado respecto a la figura 3.7. Los valores de RTT se corroboraron con la herramienta Nping y se obtuvieron los mismos resultados. La gráfica muestra como los paquetes llegarón en cinco ráfagas distintas incrementándose el tiempo entre el primer paquete y el último. El ancho de banda medio de la red es 31.43 Mbps, sin embargo, en la gráfica se puede ver como hay picos de más de 100 Mbps, y momentos en los que prácticamente no se transmiten paquetes (1.82 Mbps). Por ello, la desviación típica del ancho de banda ha sido más elevada (37.92 Mbps) que en el ejemplo de la figura 3.7 (4.74 Mbps). Por último, no se ha perdido ningún paquete durante la transmisión. Con todo ello, se puede deducir que la calidad de experiencia percibida por el usuario en esta red es peor que la de la figura 3.17. Las aplicaciones en tiempo real como Skype se verán afectadas por el *jitter* y por las variaciones del ancho de banda.

Este mismo test también se realizó con paquetes UDP. No obstante, de 100 paquetes que se enviaron, solo se recibieron 6 paquetes de respuesta, lo que imposibilitó realizar las medidas. En la figura 4.21 se puede ver como solo algunos puertos contestan. En este caso únicamente se reciben mensajes de los seis primeros puertos: 33434, 33435, 33436, 33437, 33438 y 33439.

No.	Time	Source	Destination	Proto	Length	Destinatio	Info
3	1.002299714	192.168.1.8	82.159.32.139	UDP	1514	33434	12345 → 33434 Len=1472
4	1.002307821	192.168.1.8	82.159.32.139	UDP	1514	33435	12345 → 33435 Len=1472
5	1.002310410	192.168.1.8	82.159.32.139	UDP	1514	33436	12345 → 33436 Len=1472
6	1.002313128	192.168.1.8	82.159.32.139	UDP	1514	33437	12345 → 33437 Len=1472
7	1.002316873	192.168.1.8	82.159.32.139	UDP	1514	33438	12345 → 33438 Len=1472
8	1.002319581	192.168.1.8	82.159.32.139	UDP	1514	33439	12345 → 33439 Len=1472
9	1.002321973	192.168.1.8	82.159.32.139	UDP	1514	33440	12345 → 33440 Len=1472
10	1.002324534	192.168.1.8	82.159.32.139	UDP	1514	33441	12345 → 33441 Len=1472
11	1.002328336	192.168.1.8	82.159.32.139	UDP	1514	33442	12345 → 33442 Len=1472
12	1.002335843	192.168.1.8	82.159.32.139	UDP	1514	33443	12345 → 33443 Len=1472
13	1.002339733	192.168.1.8	82.159.32.139	UDP	1514	33444	12345 → 33444 Len=1472
14	1.002415937	192.168.1.8	82.159.32.139	UDP	1514	33445	12345 → 33445 Len=1472
15	1.002421918	192.168.1.8	82.159.32.139	UDP	1514	33446	12345 → 33446 Len=1472
16	1.002422902	192.168.1.8	82.159.32.139	UDP	1514	33447	12345 → 33447 Len=1472
17	1.002423828	192.168.1.8	82.159.32.139	UDP	1514	33448	12345 → 33448 Len=1472
18	1.002425141	192.168.1.8	82.159.32.139	UDP	1514	33449	12345 → 33449 Len=1472
19	1.002769459	192.168.1.8	82.159.32.139	UDP	1514	33450	12345 → 33450 Len=1472
20	1.002772775	192.168.1.8	82.159.32.139	UDP	1514	33451	12345 → 33451 Len=1472
21	1.002773625	192.168.1.8	82.159.32.139	UDP	1514	33452	12345 → 33452 Len=1472
22	1.002774942	192.168.1.8	82.159.32.139	UDP	1514	33453	12345 → 33453 Len=1472
23	1.022672833	82.159.32.139	192.168.1.8	ICMP	590	33434	Destination unreachable (Port unreachable)
24	1.023269287	82.159.32.139	192.168.1.8	ICMP	590	33435	Destination unreachable (Port unreachable)
25	1.023296787	82.159.32.139	192.168.1.8	ICMP	590	33436	Destination unreachable (Port unreachable)
26	1.023304181	82.159.32.139	192.168.1.8	ICMP	590	33437	Destination unreachable (Port unreachable)
27	1.028053219	82.159.32.139	192.168.1.8	ICMP	590	33438	Destination unreachable (Port unreachable)
28	1.028445111	82.159.32.139	192.168.1.8	ICMP	590	33439	Destination unreachable (Port unreachable)

Figura 4.19 : Respuesta del router a los paquetes UDP en el test 5. Captura de Wireshark del resultado de enviar 20 paquetes UDP. En azul los paquetes UDP y en Negro los paquetes ICMP de respuesta. La penúltima columna (*destination*) es el número de puerto.

4.4 Conclusiones

En este capítulo se han explicado y analizado las distintas pruebas realizadas. Tras llevar a cabo los distintos test se ha comprobado que la técnica basada en UDP no es tan fiable como podía parecer en un principio. De hecho, en ninguno de los test se han obtenido medidas cercanas a la capacidad del enlace (solo por debajo de 30 Mbps). Uno de los problemas a los que se enfrenta este método es la coalescencia de los paquetes, como los paquetes ICMP de puesto inalcanzable son pequeños (70 bytes en los test) el router los agrupa y los envía seguidos. Esto hace que el tiempo entre los paquetes pueda ser del orden de nanosegundos, alterando los resultados y disminuyendo así la precisión de las medidas. Aunque la coalescencia se puede eliminar con herramientas como ethtool no todo los routers permiten modificarla.

Otro problema con la técnica basada en paquetes UDP, es que no todos los routers contestan a este tipo de paquetes. Otros routers sí que responden con paquetes ICMP de puerto inalcanzable pero no de todos los puertos, lo que hace que las estimaciones no puedan ser consideradas como válidas o no se puedan realizar.

Por otro lado, el método basado en paquetes ICMP de tipo eco, ha dado resultados más precisos. Por ejemplo, en el segundo test llevado a cabo en un entorno doméstico se obtuvo un error medio del 3%. Además, en las pruebas realizadas en entornos virtuales se pudo observar como la precisión de esta técnica disminuye a medida que aumenta la capacidad máxima del enlace, obteniendo un error de 4,49% para 200 Mbps y de 10.23% para 400Mbps. Así, llega un punto que el error es tan grande que las medidas no se pueden considerar como válidas.

También, en esta sección se ha probado a aumentar el número de paquetes para ver cómo afecta a las medidas. De acuerdo con lo mencionado en el apartado 2.4, se ha obtenido que las estimaciones del ancho de banda son más precisas cuanto mayor es el número de paquetes, por ejemplo, para 50 paquetes el error medio es 14.12% y para 1000 paquetes es 4.87%. Así, se puede concluir que a la hora de medir anchos de banda lo mejor es enviar el máximo número de paquetes sin que el router lo considere un ataque. Además, con esta prueba se ha podido ver que la herramienta Token bucket también pierde precisión a medida que se le exigen capacidades de enlace más grandes. Esto es útil de cara a futuros test que se quieran realizar.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

El objetivo de este Trabajo de Fin de Grado era buscar una alternativa a las técnicas de medición de los parámetros de calidad de servicio que existen en la actualidad. Se buscaba encontrar un método rápido, simple y preciso, que permitiera a particulares y empresas monitorizar la red y, así, comprobar que se están cumpliendo los acuerdos de calidad de servicio (SLA). Más concretamente, este trabajo se ha centrado sobre todo en encontrar un método que reúna estas características para medir el ancho de banda, puesto que para el resto de parámetros ya existen herramientas que permiten monitorizar la red de manera precisa y eficiente.

Para ello, se llevó a cabo un estudio de las técnicas ICMP y UDP basadas en enviar trenes de paquetes. Investigaciones previas mostraban que se habían obtenido mejores resultados utilizando técnicas basadas en trenes de paquetes que usando otros métodos más extendidos, como descargar un archivo. También, se determinó cuáles debían ser las características del tren de paquetes para hacer las estimaciones más precisas. Así, se ha desarrollado un programa capaz de medir los parámetros QoS utilizando una de las dos técnicas, la que escoja el usuario. El código se encuentra disponible en la página web: https://github.com/matesanzvictor/QoStool.

A continuación, se realizaron varias pruebas para ver la precisión y la eficiencia de ambos métodos de medición. Tras los test, se puede concluir que la técnica basada en paquetes UDP no es tan precisa y fiable como se podía esperar en un principio. Como los paquetes ICMP de puerto inalcanzable son de menor tamaño están sujetos a la coalescencia. Además, no todos los routers permiten cambiar este parámetro. El otro problema que se ha observado es que no siempre se reciben los mensajes de respuesta de todos los puertos, lo que hace imposible hacer mediciones. No obstante, esta técnica tiene potencial para ser usada en determinadas ocasiones, como se explicará en el siguiente apartado.

Por otro lado, la técnica basada en trenes de paquetes ICMP de tipo eco, sí que ha demostrado ser precisa y eficiente. En las pruebas realizadas en un entorno doméstico se ha observado como las estimaciones del ancho de banda, así como del resto de parámetros QoS han sido bastante aproximados a la realidad, con un 3.16% de error medio. Hay que tener en cuenta que en un entorno doméstico hay tráfico cruzado y procesos que se están ejecutando a la vez que el programa.

En las pruebas en entornos virtuales se ha visto como esta técnica puede medir capacidades de enlace mayores con un error pequeño de 6.29% para 300Mbps. Sin embargo, este error de estimación podría ser mucho menor en la realidad puesto que la herramienta Token bucket limita peor el ancho de banda cuanto mayor sea este.

En resumen, La técnica basada en paquetes ICMP ha resultado cumplir con los requisitos de rapidez, simplicidad y precisión. Con este método se pueden obtener resultados del orden de

milisegundos. Por ejemplo, pasaron 0.080 segundos desde que se envió el primer paquete hasta que se recibió el último (figura 3.7). En comparación, con los 30 segundos que le ha llevado a este test obtener los resultados https://www.movistar.es/test-de-velocidad. Otra ventaja de esta técnica es que no es necesario parar la actividad de la red de una empresa para obtener medidas aproximadas ya que el tráfico cruzado no afecta a los resultados. Además, no hay que olvidar se requiere únicamente un *host* desde el que poder enviar los paquetes.

El principal inconveniente es a la hora de medir redes asimétricas, puesto que con esta técnica se mide el cuello de botella de toda la red, no de la dirección deseada. Sin embargo, en el ámbito empresarial muchas redes presentan una tipología simétrica donde el ancho de banda de bajada es similar al de subida por lo que sería efectivo utilizar esta técnica.

Otro inconveniente es que algunos routers pueden considerar los trenes de paquetes ICMP como un ataque y desecharlos. No obstante, viendo los resultados de las pruebas el número de paquetes de respuesta son suficientes para poder obtener medidas precisas.

Por último, hay que destacar que para el desarrollo de este trabajo se han requerido conocimientos de varias asignaturas de la carrera como Arquitectura de Redes I y II, Redes Multimedia, Sistemas informáticos y Programación I y II. Como consecuencia de la realización de este trabajo, han aumentado mis conocimientos en el ámbito de las redes, sobre todo, de los protocolos UDP e ICMP, y del nivel de enlace. Este proyecto también, ha requerido el desarrollo de un programa capaz de medir los parámetros QoS, por lo que también he profundizado mis conocimientos de programación en C.

5.2 Trabajo futuro

A pesar de que se han hecho varios test probando ambas técnicas, la herramienta token bucket ha demostrado no restringir de manera precisa la capacidad del enlace. Para futuros trabajos se podría buscar una alternativa que limite mejor el ancho de banda. También, sería interesante hacer pruebas en un entorno real que supere los 100Mbps de ancho de banda.

De esta manera, se podría probar cual es el límite del método basado en ICMP. En una de las pruebas (figura 4.17) se vio que ICMP podría llegar a medir capacidades del orden de Gigabit/s con bastante precisión. Por ejemplo, Esto podría interesar a la hora de medir el ancho de banda de una red ethernet empresarial o para el FTTH (*Fiber to the home*, Fibra óptica hasta el hogar), donde los operadores están ofreciendo hasta 1Gbps simétrico. De este modo, se podría saber cuál es la velocidad real que tenemos en nuestra red y ver cuánto se ajusta a lo acordado.

Por último, se podría continuar con la mejora del programa QoStool. Por ejemplo, se podría desarrollar una interfaz que grafique las medidas según se van realizando, tal y como hacen otras herramientas existentes.

6 Referencias

[1] Oficina de Atención al Ususario de Telecomunicaciones "Informe de la Oficina de Atención al Ususario de Telecomunicaciones Datos 2019", 2019, https://usuariosteleco.mineco.gob.es/quienes-somos/datos-informes-oficina/Paginas/datos-informes.aspx , último acceso: 14/04/2021.

[2] Ministerio de Industria, Energía y Turismo, "IET/1090/2014, por la que se regulan las condiciones relativas a la calidad de servicio en la prestacion de los servicios de comunicaciones electronicas", 2014", https://www.boe.es/buscar/doc.php?id=BOE-A-2014-6729, último acceso: 14/04/2021.

[3] Iperf.fr, "iPerf- The ultimate speed test tool for TCP, UDP and SCTP", https://iperf.fr/, último acceso: 30/04/2021.

[4] R. Oliveira, "Caveats of traditional network tools-Part II: Iperf", blog.thousandeyes.com, 6 de Agosto 2013, https://blog.thousandeyes.com/caveats-of-traditional-network-tools-iperf/ , último acceso: 17/04/2021.

[5] J. Aracil, J.E. López de Vergara, J. Ramos, P.M. Santiago del Río, "On the effect of concurrent applications in bandwidth", Computer Networks, Vol. 55, No. 6, abril 2011, pp. 1435-1453, Elsevier, ISSN: 1389-1286.

[6] Y.Mirsky,N.Kalbo,Y. Elovici,A.Shabtai,N.Vesper, "BurstPing", página web github.com, 23 de Noviembre 2018, https://github.com/ymirsky/burstPing, último acceso: 10/05/2021.

[7] L. Martin Garcia, "nping(1) - Linux man page ", página del manual de Linux, año 2006. https://man7.org/linux/man-pages/man1/nping.1.html, último acceso: 21/05/2021.

[8] N. Kuznetsov, "tc-tbf(8) — Linux manual page", página del manual de Linux, año 2001, https://man7.org/linux/man-pages/man8/tc-tbf.8.html , último acceso: 4/05/2021.

[9] D. Morato Osés, "QoS: Parámetros de red", departamento de Redes, Sistemas y Servicios Telemáticos, Universidad Pública de Navarra, 2020.

[10] M. S. Borella, D. Swider, S. Uludag y G. B. Brewster, "Internet Packet Loss: Measurement and Implications for End-to-End QoS", Conferencia: Architectural and OS Support for Multimedia Applications/Flexible Communication Systems/Wireless Networks and Mobile Computing ,Minneapolis, USA, septiembre 1998.

[11] R. Prasad, C. Dovrolis, M. Murray y k. claffy, "Bandwidth Estimation: Metrics,Measurement Techniques, and Tools", IEEE Network, Vol. 17, No. 6, diciembre 2003.

[12] M. Mathis, M. Allman, " A Framework for Defining Empirical Bulk Transfer Capacity Metrics A Framework for Defining", RFC 3148, julio 2001, https://tools.ietf.org/html/rfc3148 , último acceso: 21/05/2021.

[13] G. Forget, B.Canada, R. Geib, R. Schrage, B.Constantine, "Framework for TCP Throughput Testing,", RFC 6349, Agosto 2011, https://tools.ietf.org/html/rfc6349 ,ultimo acceso: 26/04/2021.

[14] C. Demichelis, P. Chimento,"IP Packet Delay Variation", RFC 3393, noviembre 2002. https://tools.ietf.org/html/rfc3393 , último acceso: 20/04/2021.

[15] European Telecommunications Standards Institute (ETSI), "Speech processing, transmission and qualityaspects (STQ); user related QoS parameter definitions and measurements; part 4: internet access", ETSI Guide, EG 202 057-4 V1.2.1, 2007-2008.

[16] I.García, J.Fernández, J.Lafuente, "QoS Estimators for Client-Side Dynamic Server Selection", Conferencia: Consumer Communications and Networking Conference, CNC 2006, Febrero 2006.

[17] J.Postel, "Internet Control Message Protocol", RFC 777, 1981, https://tools.ietf.org/html/rfc777, último acceso: 6/05/2021.

[18] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology", IEEE/ACM Transactions on Networking, Vol. 12, No. 6, diciembre 2004.

[19] Speedguide.net "Port 12345 Details" página web speedguide.net, https://www.speedguide.net/port.php?port=12345, último acceso: 23/04/2021.

[20] Speedguide.net, "Port 33434 Details"página web speedguide.net", https://www.speedguide.net/port.php?port=33434, último acceso: 23/04/2021.

[21] A. López Monge,"Aprendiendo a programar con Libpcap",20 de Febrero de 2005.

[22] P. Xie, "How Slow is Python Compared to C" página web medium.com, 2020. https://medium.com/codex/how-slow-is-python-compared-to-c-3795071ce82a#:~:text=It%20is%20450%20millions%20loops,mode%20for%20a%20better%20performance., último acceso: 11/04/2021.

[23] GNU C Library, “ethernet.h”, 2008, https://sites.uclouvain.be/SystInfo/usr/include/net/ethernet.h.html, último acceso: 24/04/2021.

[24] GNU C library, “Source to netinet/ip.h”, 1986, https://unix.superglobalmegacorp.com/Net2/newsrc/netinet/ip.h.html, último acceso: 26/04/2021.

[25] GNU C Library , “netinet/ip_icmp.h” , https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip_icmp.h.html, último acceso: 26/04/2021.

[26] J. Postel, “User Datagram Protocol”, RFC 768, 1980, https://tools.ietf.org/html/rfc768, último acceso : 23/04/2021.

[27] Man7.org, “Linux manual page, nanosleep(2)”, manual de Linux, , 2021, https://man7.org/linux/man-pages/man2/nanosleep.2.html , último acceso: 24/04/2021.

[28] Serverfault.com, “Tc: ingress policing and ifb mirroring”, blog serverfault, ,https://serverfault.com/questions/350023/tc-ingress-policing-and-ifb-mirroring, último acceso: 2/05/2021.

[29] Docs.microsoft.com, “Overview of Packet Coalescing”, 2017, https://docs.microsoft.com/en-us/windows-hardware/drivers/network/overview-of-packet-coalescing, último acceso: 5/05/2021.

[30] whatls.com, ”packet coalescing”, 2015, https://whatis.techtarget.com/definition/packet-coalescing, último acceso: 5/05/2021.

[31] D. Miller, “ethtool(8) - Linux man page.” ,https://linux.die.net/man/8/ethtool, último acceso: 5/05/2021.

[32] Mininet.org, “Mininet Overview”, http://mininet.org/overview/#:~:text=Mininet%20is%20a%20network%20emulator,switches%2C%20controllers%2C%20and%20links.&text=Enables%20complex%20topology%20testing%2C%20without,or%20running%20network%2Dwide%20tests, último acceso: 8/05/2021.

[33] Insituto Nacional de Estadística (INE), “El teletrabajo en España y la UE antes de la COVID-19”, Febrero 2020. https://www.ine.es/ss/Satellite?L=es_ES&c=INECifrasINE_DetalleCifrasINE.