Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Comparativa de herramientas libres de generación de tráfico de red

Estudiante: Manuel Vidal Vázquez
Dirección: Miguel González López

A Coruña, novembro de 2019.

Dedicado a quienes siempre apoyaron y me dijeron que podía terminar la carrera.

Agradecimientos

Agradezco a mis padres, por darme la oportunidad de hacer esta carrera y soportarme en los momentos de nerviosismo y mal humor.

A mi hermano por aguantar parte de mis quejas.

A mi tutor Miguel, por mostrar tanta paciencia conmigo y atenderme cuando se lo pedía.

Y por último a mis familiares y amigos que se preocuparon por mí.

Resumen

El objetivo de este TFG es realizar un estudio comparativo de las herramientas de uso libre de generación de tráfico en red (IPerf, NetPerf, MGEN, D-ITG, TRex y WARP17). Para ello se probarán las capacidades de cada herramienta, primero mostrando un ejemplo de cada una de las pruebas más característicos de cada una y luego comprobando cómo se comportan ante posibles problemas que pueda tener la red.

Abstract

The objective of this TFG is to carry out a comparative study of the tools for the free use of network traffic generation mentioned above (IPerf, Netperf, MGEN, D-ITG, TRex and WARP17). To do this, the capabilities of each tool will be tested, first showing an example of each of the most characteristic tests of each one and then checking how they behave when faced with possible problems that the network may have.

Palabras clave:

- Análisis de rendimiento de red
- Herramientas de uso libre
- Flujo de datos de red
- Simulación de red
- Medición de red

Keywords:

- Network performance analysis
- Free use tools
- Network data stream
- Network simulation
- Network measurement

Índice general

1 Presentación y estado del arte **1**

1.1 IPerf . . . 1
1.2 NetPerf . . . 2
1.3 MGEN . . . 2
1.4 D-ITG . . . 3
1.5 TREX . . . 3
1.6 WARP17 . . . 4
1.7 CORE Network Emulator . . . 4

2 Planificación y metodología **5**

2.1 Metodología . . . 5
2.2 Planificación . . . 5

3 Pruebas básicas con las herramientas de generación de tráfico **9**

3.1 IPerf . . . 9
3.2 NetPerf . . . 13
3.3 MGEN . . . 16
3.4 D-ITG . . . 19
3.5 TRex . . . 22
3.5.1 Stateful . . . 22
3.5.2 Stateless . . . 27

4 Comportamiento de las herramientas de generación de tráfico ante anomalías en la red **31**

4.1 Comandos empleados durante las pruebas . . . 33
4.2 Escenario original . . . 35
4.3 Limitación del ancho de banda en un enlace . . . 40
4.4 Aumento del *delay* de un enlace . . . 45

4.5 Aumento del *jitter* de un enlace . . . 51
4.6 Aumento de la pérdida de paquetes de un enlace . . . 52
4.7 Aumento de la duplicidad de un enlace . . . 62
4.8 Conclusiones de las pruebas anteriores . . . 64
4.9 Pruebas multiflujo . . . 68
4.10 Pruebas múltiples equipos . . . 68

5 Comparación de características de las herramientas 73
5.1 IPerf . . . 73
5.2 NetPerf . . . 73
5.3 MGEN . . . 74
5.4 DITG . . . 74
5.5 TREX . . . 74

6 Conclusiones 77

A Instalación de las herramientas 81
A.1 IPerf . . . 81
A.2 NetPerf . . . 81
A.3 MGEN . . . 81
A.3.1 TRPR . . . 81
A.4 D-ITG . . . 81
A.5 TREX . . . 82
A.6 WARP17 . . . 82
A.7 CORE Network Emulator . . . 83

B Instalación de las interfaces gráficas 85
B.1 JPerf (Iperf) . . . 85
B.2 Flent (Iperf y NetPerf) . . . 85
B.3 Core-MGEN . . . 85
B.4 D-ITG GUI . . . 85
B.5 TRex . . . 86

C Aplicaciones de interfaz gráfica para el uso de las herramientas 87
C.1 JPerf (IPerf) . . . 87
C.2 Flent (IPerf y NetPerf) . . . 88
C.3 Core-MGEN . . . 88
C.4 D-ITG GUI . . . 88

D Anexo: comandos de las pruebas **97**
D.1 IPerf 97
D.2 NetPerf 99
D.3 MGEN 100
D.4 D-ITG 102
D.5 TREX 103

Lista de acrónimos **105**

Bibliografía **107**

Índice de figuras

2.1 Primera parte del diagrama de Gantt . . . 7
2.2 Segunda parte del diagrama de Gantt . . . 8
2.3 Tercera y última parte del diagrama de Gantt . . . 8

3.1 Captura del escenario creado en CORE para realizar el TFG. . . . 11
3.2 Prueba básica con variación de tamaño de ventana con IPerf. . . . 11
3.3 Prueba básica con variación de tamaño de mensaje con IPerf. . . . 12
3.4 Flujos paralelos con IPerf. . . . 12
3.5 Flujo UDP con intervalos en IPerf. . . . 13
3.6 Flujo básico con NetPerf usando TCP. . . . 14
3.7 Flujo básico con NetPerf usando UDP. . . . 14
3.8 Flujo invertido con NetPerf. . . . 14
3.9 Prueba de envío con NetPerf. . . . 15
3.10 Muestra de error de paquete demasiado pequeño por prueba de envío con NetPerf. . . . 15
3.11 Prueba Petición-Respuesta con NetPerf usando TCP. . . . 15
3.12 Prueba Petición-Respuesta con NetPerf usando UDP. . . . 18
3.13 Prueba Omni de NetPerf. . . . 18
3.14 Muestra de generación de flujo MGEN. . . . 18
3.15 Muestra de modificación de flujo MGEN. . . . 18
3.16 Muestra de análisis de log MGEN con TRPR. . . . 20
3.17 Prueba básica en el lado cliente con D-ITG. . . . 20
3.18 Prueba básica en el lado servidor con D-ITG. . . . 20
3.19 Prueba pasiva D-ITG. . . . 21
3.20 Muestra de comportamiento del servidor de Logs de D-ITG . . . 23
3.21 Pruebas simultáneas con *script* en el lado cliente con D-ITG parte 1. . . . 23
3.22 Prueba simultáneas con *script* en el lado cliente con D-ITG parte 2. . . . 24

3.23 Prueba simultáneas con *script* en el lado servidor con D-ITG parte 1. 24
3.24 Prueba simultáneas con *script* en el lado servidor con D-ITG parte 2. 25
3.25 Muestra de log del lado cliente a prueba DNS. 25
3.26 Muestra de log del lado servidor a prueba DNS. 25
3.27 Ejemplo de salida de una prueba de TREX Stateful. 27
3.28 Ejemplo de salida de una prueba de TREX Stateless parte 1. 28
3.29 Ejemplo de salida de una prueba de TREX Stateless parte 2. 28

4.1 Versión del escenario para las herramientas excepto TREX. 32
4.2 Versión del escenario para TREX. 32
4.3 IPerf con protocolo TCP en escenario original. 36
4.4 IPerf con protocolo UDP en escenario original. 36
4.5 NetPerf con protocolo TCP en escenario original. 37
4.6 NetPerf con protocolo UDP en escenario original. 37
4.7 D-ITG con protocolo TCP en escenario original. 38
4.8 D-ITG con protocolo UDP en escenario original. 38
4.9 D-ITG con protocolo VoIP en escenario original. 39
4.10 TREX con protocolo TCP en escenario original. 39
4.11 TREX con protocolo UDP en escenario original. 41
4.12 IPerf con protocolo TCP en escenario con ancho de banda modificado. 41
4.13 IPerf con protocolo UDP en escenario con ancho de banda modificado 42
4.14 NetPerf con protocolo TCP en escenario con ancho de banda modificado . . . 42
4.15 NetPerf con protocolo UDP en escenario con ancho de banda modificado . . . 43
4.16 D-ITG con protocolo TCP en escenario con ancho de banda modificado 43
4.17 D-ITG con protocolo UDP en escenario con ancho de banda modificado 44
4.18 TREX con protocolo TCP en escenario con ancho de banda modificado 44
4.19 TREX con protocolo UDP en escenario con ancho de banda modificado 46
4.20 IPerf con protocolo TCP en escenario con *delay* modificado 46
4.21 IPerf con protocolo UDP en escenario con *delay* modificado 47
4.22 NetPerf con protocolo TCP en escenario con *delay* modificado 47
4.23 NetPerf con protocolo UDP en escenario con *delay* modificado 48
4.24 D-ITG con protocolo TCP en escenario con *delay* modificado 48
4.25 D-ITG con protocolo UDP en escenario con *delay* modificado 49
4.26 D-ITG con protocolo VoIP en escenario con *delay* modificado 49
4.27 TREX con protocolo TCP en escenario con *delay* modificado 50
4.28 TREX con protocolo UDP en escenario con *delay* modificado 50
4.29 IPerf con protocolo TCP en escenario con *jitter* modificado. 51
4.30 IPerf con protocolo UDP en escenario con *jitter* modificado. 53

4.31 NetPerf con protocolo TCP en escenario con *jitter* modificado. 53
4.32 NetPerf con protocolo UDP en escenario con *jitter* modificado. 54
4.33 D-ITG con protocolo TCP en escenario con *jitter* modificado. 54
4.34 D-ITG con protocolo UDP en escenario con *jitter* modificado. 55
4.35 D-ITG con protocolo VoIP en escenario con *jitter* modificado. 55
4.36 TREX con protocolo TCP en escenario con *jitter* modificado. 56
4.37 TREX con protocolo UDP en escenario con *jitter* modificado. 56
4.38 IPerf con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado. 57
4.39 IPerf con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado. 57
4.40 NetPerf con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado. 58
4.41 NetPerf con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado. 58
4.42 D-ITG con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado. 59
4.43 D-ITG con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado. 60
4.44 D-ITG con protocolo VoIP en escenario con porcentaje de pérdida de paquetes modificado. 60
4.45 TREX con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado. 61
4.46 TREX con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado. 61
4.47 IPerf con protocolo TCP en escenario con porcentaje de duplicidad modificado. 62
4.48 IPerf con protocolo UDP en escenario con porcentaje de duplicidad modificado. 63
4.49 NetPerf con protocolo TCP en escenario con porcentaje de duplicidad modificado. 63
4.50 NetPerf con protocolo UDP en escenario con porcentaje de duplicidad modificado. 65
4.51 D-ITG con protocolo TCP en escenario con porcentaje de duplicidad modificado. 65
4.52 D-ITG con protocolo UDP en escenario con porcentaje de duplicidad modificado. 66
4.53 D-ITG con protocolo VoIP en escenario con porcentaje de duplicidad modificado. 66
4.54 TREX con protocolo TCP en escenario con porcentaje de duplicidad modificado. 67
4.55 TREX con protocolo UDP en escenario con porcentaje de duplicidad modificado. 67
4.56 Prueba en paralelo con IPerf. 69

4.57 Prueba en paralelo con D-ITG. 69
4.58 Empleando múltiples equipos con IPerf. 70
4.59 Empleando múltiples equipos con D-ITG. 70
4.60 Empleando múltiples equipos con D-ITG (continuación). 71

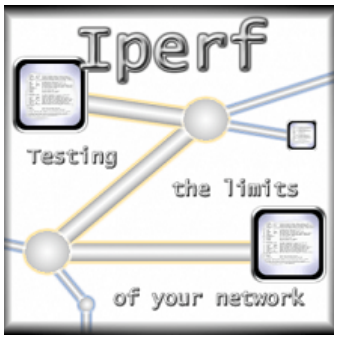
C.1 Interfaz de JPerf 89
C.2 Empezar una nueva prueba con Flent 89
C.3 En la pantalla de nuevo test apenas podemos escoger el test. 90
C.4 Lista de test a escoger. 90
C.5 Muestra de una gráfica resultante de un test. 91
C.6 Interfaz de CORE Network Emulator de manejo de flujos con MGEN 91
C.7 Interfaz principal de D-ITG GUI 92
C.8 Lista de plantillas de pruebas guardadas de D-ITG GUI 92
C.9 Interfaz de configuración de D-ITG GUI 94
C.10 Interfaz de pruebas multiflujo de D-ITG GUI 94
C.11 Interfaz del analizador de logs de D-ITG GUI 95
C.12 Muestra de un log analizado de una prueba anteriormente realizada con D-ITG GUI 95

Capítulo 1

Presentación y estado del arte

Comenzamos dando una presentación de cada herramienta empleada en este TFG. Describiremos el tipo de herramienta que es, daremos un breve resumen de las características principales de cada una y los usos propios que disponen.

1.1 IPerf



[1]

Internet Performance (IPerf). Se trata de una herramienta sencilla cliente – servidor que mide la velocidad que pueden alcanzar las conexiones entre distintos equipos comunicados en red. Permite generar tráfico tanto UDP como TCP añadiendo distintos factores, como el ancho de banda o el número de conexiones paralelas, con el fin de ajustar la prueba de la herramienta a la red testear. Muestra diferentes resultados según el tipo de tráfico a probar: en TCP indica el ancho de banda únicamente, mientras que en UDP muestra también el porcentaje de paquetes perdidos y el *jitter* además del ancho de banda.

1.2 NetPerf

[2, 3]

Networking Performance (NetPerf). Permite hacer pruebas de varios aspectos de una red. Por una parte, puede actuar como un generador de tráfico unidireccional, tanto UDP como TCP, para hacer pruebas sobre la velocidad y capacidad de la red, de forma similar a IPerf. Por otra parte, puede enviar tráfico de petición - respuesta, para comprobar y medir la capacidad de respuesta de los equipos de la red. Además, puede forzarse a la herramienta a que calcule un gran abanico de datos estadísticos entre los que puede escoger el usuario.

1.3 MGEN

[4, 5]

Multi-Generator (MGEN). Permite generar patrones de flujo de tráfico en tiempo real, con el fin de emular el funcionamiento de aplicaciones en los equipos de la red. Dichos patrones se generan mediante scripts, con los cuales pueden crearse pruebas programadas para actuar en momentos determinados. Genera tráfico tanto TCP como UDP, *unicast* y *multicast*, haciendo incluso que los equipos en pruebas puedan entrar y salir de grupos de difusión *multicast* durante las simulaciones.

Las estadísticas de rendimiento, pérdidas y retrasos en la conexión han de ser realizadas con otras herramientas ya que MGEN no las calcula directamente.

1.4 D-ITG

[6]

Es una herramienta capaz de producir tráfico de red empleando paquetes de capa de aplicaciones. Además, D-ITG es también una herramienta de medición de red capaz de obtener las estadísticas de rendimiento más comunes (*throughput*, *delay*, *jitter*, pérdida de paquetes) de forma precisa.

D-ITG soporta TCP, UDP, SCTP, y DCCP e ICMP. Además, la herramienta es capaz de replicar las estadísticas de tráfico de aplicaciones diferentes (p. ej. Telnet, VoIP, DNS, juegos en red...). Permite una alta capacidad de configuración al generar el tráfico y cuenta con opciones específicas para las diferentes aplicaciones que esta soporta.

D-ITG cuenta con un sistema de logs muy eficaz. Los logs se encriptan al momento de crearlos y pueden ser desencriptados por la propia herramienta de forma sencilla para poder leerlos. Además, cuenta con la capacidad de configurar un servidor de logs de forma rápida y sencilla en el momento que se necesite.

1.5 TREX

[7]

TRex es un generador de tráfico de código abierto y de bajo costo. Es capaz de generar tráfico siguiendo las directrices de un script Python, o bien replicar muestras obtenidas previamente con herramientas de captura de tráfico (archivos .pcap obtenidos, por ejemplo, con whireshark). Pueden realizarse forzarse a realizar las pruebas múltiples veces simultáneamente, multiplicando el resultado original para obtener un flujo de datos mayor.

Puede emplearse en cualquier equipo con *Data Plane Development Kit* (DKDP).

TRex divide su funcionalidad en dos partes: con estado (*stateful*) y sin estado (*stateless*).

La primera (*stateful*)[8] se basa en el empleo de plantillas que configuran cómo se generará el tráfico: direcciones de red, script o muestras a emplear, el tiempo de ejecución, etc.

La segunda (*stateless*)[9] ejecuta un script Python o una muestra de paquetería directamente, esto empleando una consola proporcionada por TRex para esta funcionalidad, permite monitorizar el tráfico que pasa por el equipo mientras se están realizando las pruebas.

1.6 WARP17

[10]

Es un generador de tráfico capaz de generar tráfico en todas las capas, aunque se centre en las aplicaciones (L5-L7) tanto TCP como UDP. Su principal característica es el poder generar flujos masivos de datos o generar un inmenso número de sesiones TCP.

Dispone de un sistema de plantillas con los que ejecutar pruebas configuradas previamente.

También permite la obtención de estadísticas desde los puertos en tiempo real y concretar en que capa o protocolo (por ejemplo, pueden comprobar las estadísticas del enrutamiento)

Puede emplearse en cualquier equipo con DKDP

1.7 CORE Network Emulator

[11, 12]

Common Open Research Emulator (CORE). Es una herramienta de simulación de redes que nos permite construir un escenario de red, emular el funcionamiento de este y emplear en los nodos las aplicaciones disponibles en nuestro equipo como si estuviesen instalados directamente en los nodos. Lo emplearemos para construir el entorno donde probaremos las herramientas anteriormente mencionadas para la realización del TFG.

A continuación, pasamos a ver algunos ejemplos de los tipos de pruebas más característicos de cada herramienta para ver y conocer su funcionamiento.

Capítulo 2

Planificación y metodología

2.1 Metodología

Para la realización de este estudio empleamos una metodología incremental, buscando formas en las que probar las herramientas con el fin de compararlas en más ámbitos y encontrar completar el propio análisis de las herramientas.

2.2 Planificación

Para el desarrollo de este estudio hemos seguido una planificación secuencial paso a paso, realizando una serie de pasos donde se trata cada herramienta y no procediendo al paso siguiente hasta haber tratado cada una.

Los pasos a que hemos seguidos fueron los siguientes:

1. Instalación y estudio de las herramientas.
2. Estudio y prueba de las interfaces gráficas disponibles para las herramientas.
3. Pruebas individuales de cada herramienta en el escenario creado en CORE.
4. Realización de pruebas con modificaciones en enlaces del escenario de CORE.
5. Estudio de los datos obtenidos, comparación y obtención de conclusiones.
6. Recopilación de datos y redacción de la documentación.

Cabe tener en cuenta, dado que una importante parte del trabajo consiste en el análisis de las pruebas realizadas, que se realizaba un trabajo de recopilación de datos a lo largo de todas las tareas, ya que era necesaria dicha recopilación para la posterior evaluación de los datos.

También debemos hacer mención de la asistencia por parte del director del TFG, con el cual realizábamos revisiones de forma periódica de aproximadamente una hora cada dos semanas en calidad de analista del proyecto.

Teniendo en cuenta las diferencias de la complejidad de las herramientas y el mantenimiento de estas, el tiempo de trabajo ha sido muy desigual. Originalmente esperábamos dedicar una cantidad similar días de trabajo a cada una, pero terminamos obteniendo tiempos muy desiguales.

A continuación, indicaremos el tiempo necesitado para la realización de cada paso de la realización del nuestro estudio.

1. Instalación y estudio de las herramientas.
   1.1. IPerf: tiempo estimado: 5 días, tiempo real: 3 días.
   1.2. NetPerf: tiempo estimado: 5 días, tiempo real: 5 días.
   1.3. MGEN: tiempo estimado: 5 días, tiempo real: 7 días.
   1.4. D-ITG: tiempo estimado: 5 días, tiempo real: 5 días.
   1.5. TRex: tiempo estimado: 5 días, tiempo real: 8 días.
   1.6. WARP17: tiempo estimado: 5 días, tiempo real: 7 días.
2. Estudio y prueba de las interfaces gráficas disponibles para las herramientas.
   2.1. JPERF: tiempo estimado: 5 días, tiempo real: 4 días.
   2.2. Flent: tiempo estimado: 5 días, tiempo real: 5 días.
   2.3. MGEN CORE: tiempo estimado: 5 días, tiempo real: 2 días.
   2.4. D-ITG GUI: tiempo estimado: 5 días, tiempo real: 8 días.
3. Pruebas individuales de cada herramienta en el escenario creado en CORE.
   3.1. IPerf: tiempo estimado: 10 días, tiempo real: 8 días.
   3.2. NetPerf: tiempo estimado: 10 días, tiempo real: 12 días.
   3.3. MGEN: tiempo estimado: 10 días, tiempo real: 13 días.
   3.4. D-ITG: tiempo estimado: 10 días, tiempo real: 10 días.
   3.5. TRex: tiempo estimado: 10 días, tiempo real: 15 días.
4. Realización de pruebas con modificaciones en enlaces del escenario de CORE.
   4.1. IPerf: tiempo estimado: 10 días, tiempo real: 10 días.
   4.2. NetPerf: tiempo estimado: 10 días, tiempo real: 14 días.
   4.3. D-ITG: tiempo estimado: 10 días, tiempo real: 11 días.
   4.4. TRex: tiempo estimado: 10 días, tiempo real: 12 días.
5. Estudio de los datos obtenidos, comparación y obtención de conclusiones: tiempo estimado: 25 días, tiempo real: 33 días.
6. Recopilación de datos y redacción de la documentación: tiempo estimado: 40 días, tiempo real: 75 días.
7. Revisiones periódicas: tiempo estimado: 14 horas, tiempo real: 21 horas.

Originalmente calculábamos un total de 1246 horas de trabajo (16 de las cuales son con la asistencia del director en las revisiones periódicas), lo correspondiente a una duración de 205 días, ya que habíamos establecido unas 6 horas de trabajo por día. Dado que habíamos

comenzado la realización del estudio el lunes 4 de febrero, calculábamos la finalización para el jueves 5 de septiembre. Estableciendo un coste de 8 euros por hora de trabajo y 15 euros por hora a la asistencia del director, estimábamos el coste 10.080€.

Sin embargo, terminamos necesitando aumentar el tiempo del desarrollo del estudio, necesitando un total de 1623 horas de trabajo (las horas de asistencia con el director ascendieron a 21), subiendo la duración del proyecto hasta 272 días, lo que supuso su finalización el viernes 15 de noviembre. Dado el aumento del número de horas, el coste terminó subiendo hasta los 13.131€.

A continuación, mostraremos un diagrama de Gantt con la distribución de las tareas a lo largo del desarrollo de nuestro proyecto. (Por su tamaño, hemos tenido que dividirlo en tres partes, las cuales son Figura 2.1, Figura 2.2 y Figura 2.3)

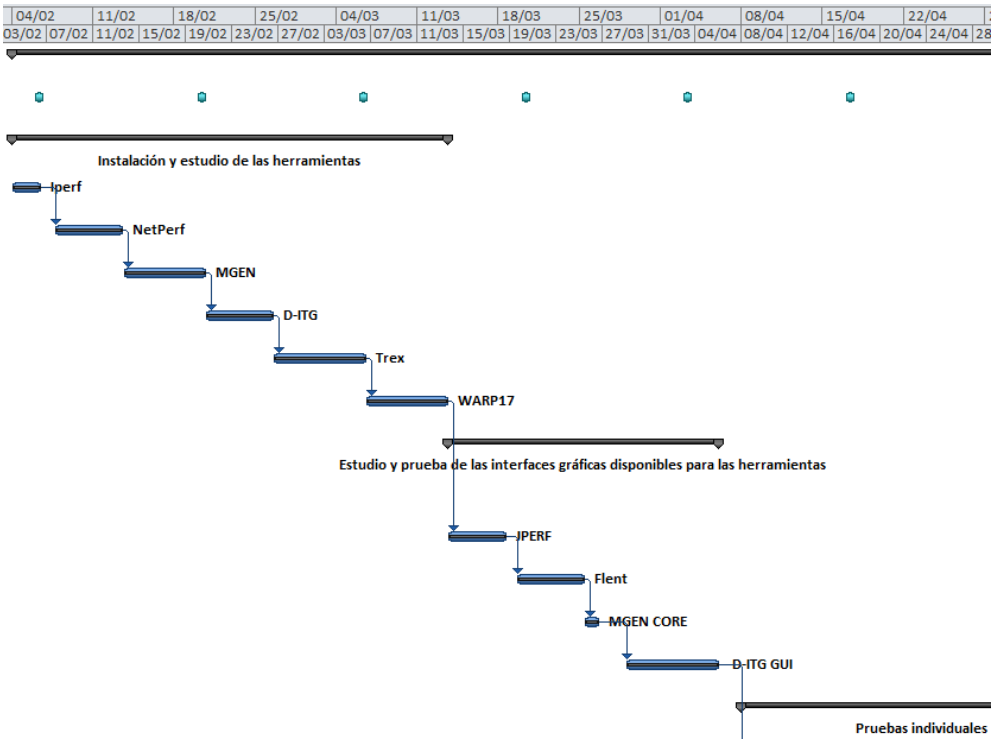


Figura 2.1: Primera parte del diagrama de Gantt

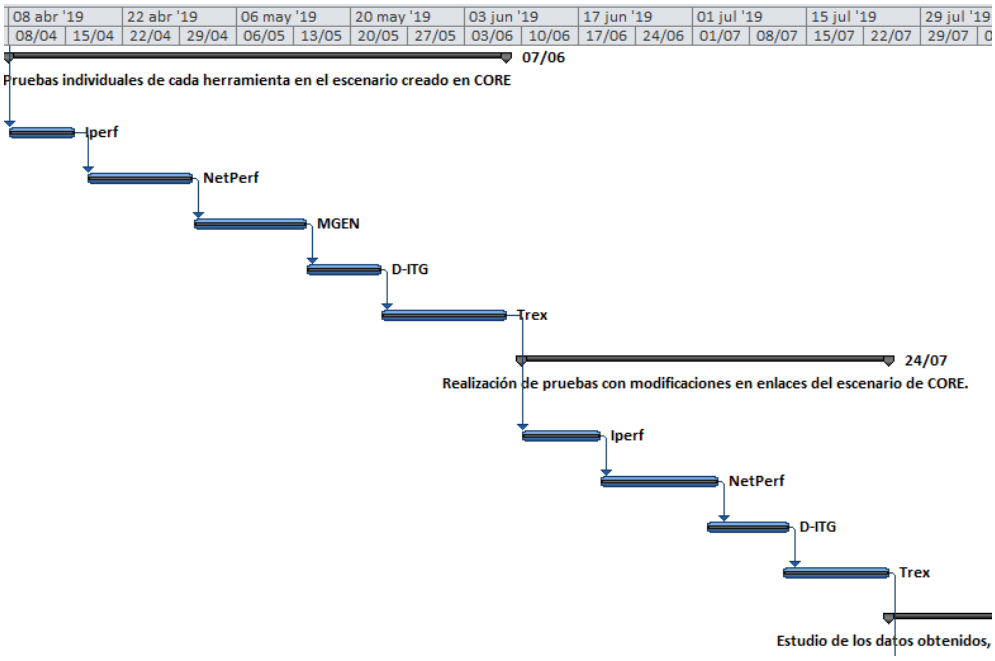


Figura 2.2: Segunda parte del diagrama de Gantt

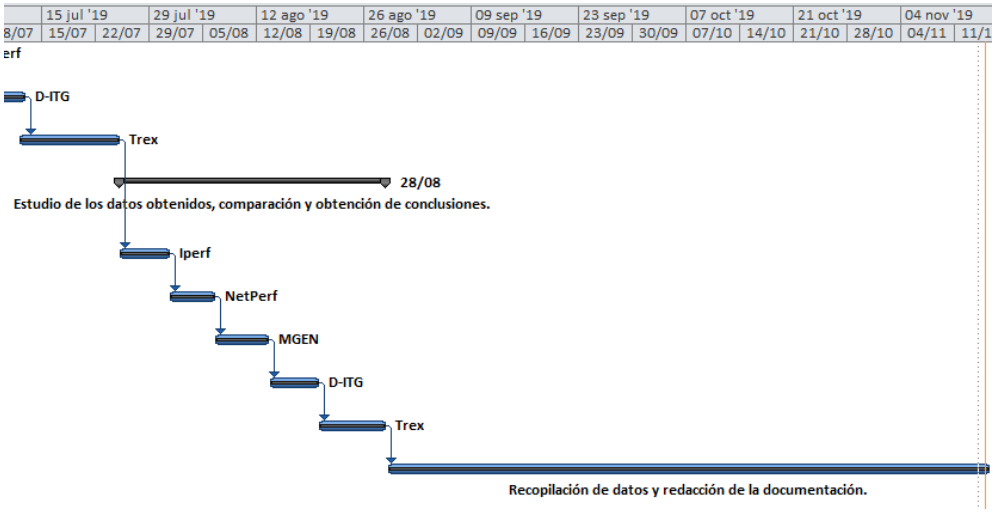


Figura 2.3: Tercera y última parte del diagrama de Gantt

Capítulo 3

Pruebas básicas con las herramientas de generación de tráfico

Comenzaremos las pruebas tras la generación de un escenario de red con estructura core – distribución – acceso emulado empleando CORE Network Emulator. Además, incluirá un *router* que conectará dos redes de nivel acceso por su cuenta con el fin de emplear dicho *router* para realizar las pruebas con TREX. El escenario que emplearemos es el siguiente (Figura 3.1).

Las siguientes herramientas hacen uso de un modelo Cliente-Servidor (necesitamos ejecutar la herramienta en dos equipos, uno en modo servidor y otro, o varios otros, en modo cliente), en la mayor parte de los casos emplearemos el nodo N15 como cliente y el nodo N19 como servidor.

3.1 IPerf

IPerf es una herramienta sencilla y con un abanico pequeño de posibilidades, pero eficiente a la hora de realizarlas.

Veremos un ejemplo de un flujo básico TCP, pudiendo mostrar alguna modificación en este. También veremos una muestra de ejecuciones en paralelo y bidireccionales y una muestra de un flujo UDP.

Recordemos que IPerf sigue el formato de cliente-servidor.

1. Pruebas con distintos tamaños de ventana y de mensaje:

   Son las pruebas más simples de IPerf, se genera un único flujo cuyas características cambian según los parámetros descritos a continuación. En caso de no indicarlos, IPerf usará sus valores por defecto.

- Servidor: `iperf -s -w 200K -m -M 1500`
- Cliente: `iperf -c 10.0.2.20 -w 100K -m -M 1500`
  - s: indica que la ejecución será en modo servidor.
  - c: indica que la ejecución será en modo cliente, seguida de la dirección ip del servidor.
  - m: en el resumen de la prueba se mostrarán los tamaños de los mensajes.
  - w: fuerza a que el tamaño de ventana de TCP (cantidad de datos que puede atender el buffer) sea del número indicado a continuación (16K por defecto).
  - M: fuerza a que el tamaño del segmento de mensaje (número de bytes del mensaje que puede enviar cada paquete) sea del número indicado a continuación (1500 por defecto).

Para ver con más claridad como afectan las modificaciones a las ejecuciones, las veremos por separado

Al emplear la modificación de tamaño de ventana (Figura 3.2) vemos como no permite el usar 200K por ser demasiado pequeño (indicando un warning que pide 195K), en vez de eso emplea 391K.

Al emplear la modificación de tamaño de mensaje (Figura 3.3) en nuestro caso, considera 1500 de tamaño de segmento de mensaje es demasiado grande y pasa a emplear 536.

2. Pruebas concurrentes y bidireccionales:

   Se caracterizan por realizar más de un flujo de datos, siendo las concurrentes en un único sentido y las bidireccionales en ambos.

   - Servidor: `iperf -s`
   - Cliente: `iperf -c 10.0.2.20 -P 5`
   - Cliente: `iperf -c 10.0.2.20 -r`
   - Cliente: `iperf -c 10.0.2.20 -d`
     - P: indica los flujos enviados en paralelo que envía el cliente al servidor.
     - r: mide el ancho de banda de la conexión.
     - d: al igual de –r pero realizando la prueba en ambos sentidos al mismo tiempo.

   Al ver los resultados (Figura 3.4), podemos ver en la imagen como se separan los envíos de cada proceso en paralelo y el sumatorio al final del tráfico de todos ellos.

3. Pruebas UDP con intervalos:

   Son las pruebas básicas con UDP, se caracteriza por el envío de paquetes por ráfagas con un intervalo, el cual podemos fijar. Una característica de las pruebas UDP, es que nos permiten ver el cálculo del *jitter* y de los paquetes perdidos (Figura 3.5).

   - Servidor: `iperf -s -u -i 5`
   - Cliente: `iperf -c 10.0.2.20 -u -b 50m`

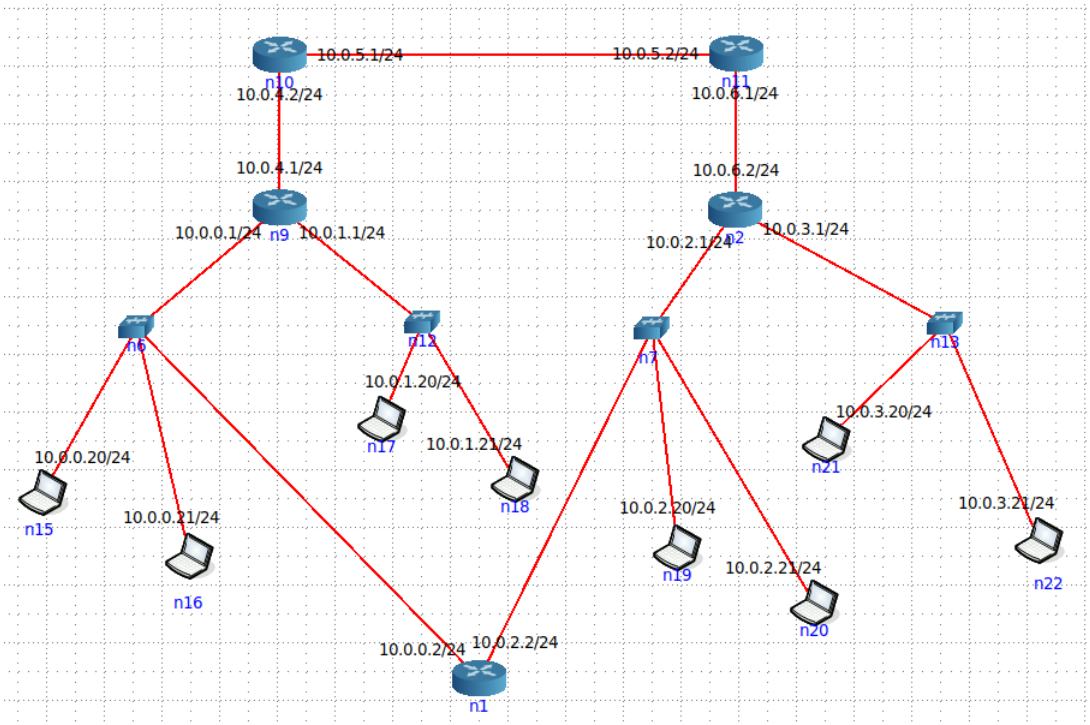


Figura 3.1: Captura del escenario creado en CORE para realizar el TFG.

```
^Croot@n19:/tmp/pycore.35169/n19.conf# iperf -s -w 200k
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  391 KByte (WARNING: requested  195 KByte)
------------------------------------------------------------
[  4] local 10.0.2.20 port 5001 connected with 10.0.0.20 port 42024
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  24.5 GBytes  21.0 Gbits/sec
```

```
Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@n15:/tmp/pycore.35169/n15.conf# iperf -c 10.0.2.20 -w 200k
------------------------------------------------------------
Client connecting to 10.0.2.20, TCP port 5001
TCP window size:  391 KByte (WARNING: requested  195 KByte)
------------------------------------------------------------
[  3] local 10.0.0.20 port 42024 connected with 10.0.2.20 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  24.5 GBytes  21.0 Gbits/sec
```

Figura 3.2: Prueba básica con variación de tamaño de ventana con IPerf.

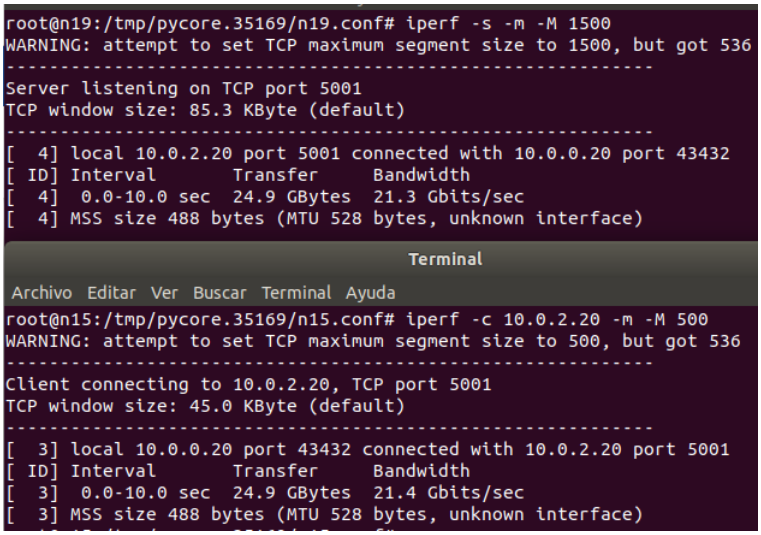


Figura 3.3: Prueba básica con variación de tamaño de mensaje con IPerf.

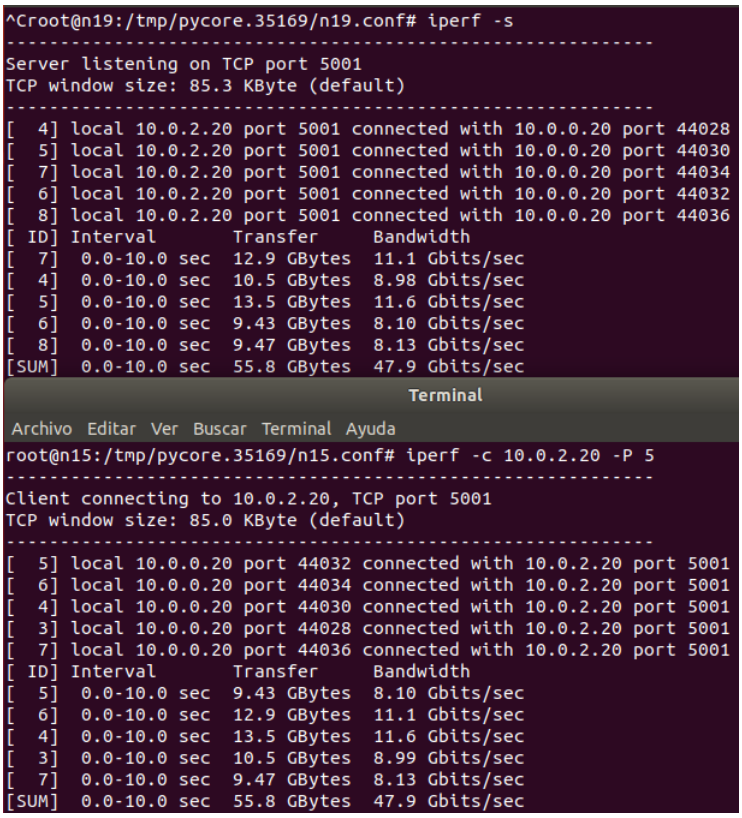


Figura 3.4: Flujos paralelos con IPerf.

- u: indica que la prueba se realizará con tráfico UDP, en su ausencia emplea TCP (ejemplos anteriores).
- i: indica el intervalo entre envíos en segundos (solo funciona con UDP).
- b: fuerza a emplear el ancho de banda deseado para pruebas con UDP.

Se puede ver como las pruebas con UDP calculan el *jitter* y la pérdida de datagramas tras la ejecución de cada intervalo.

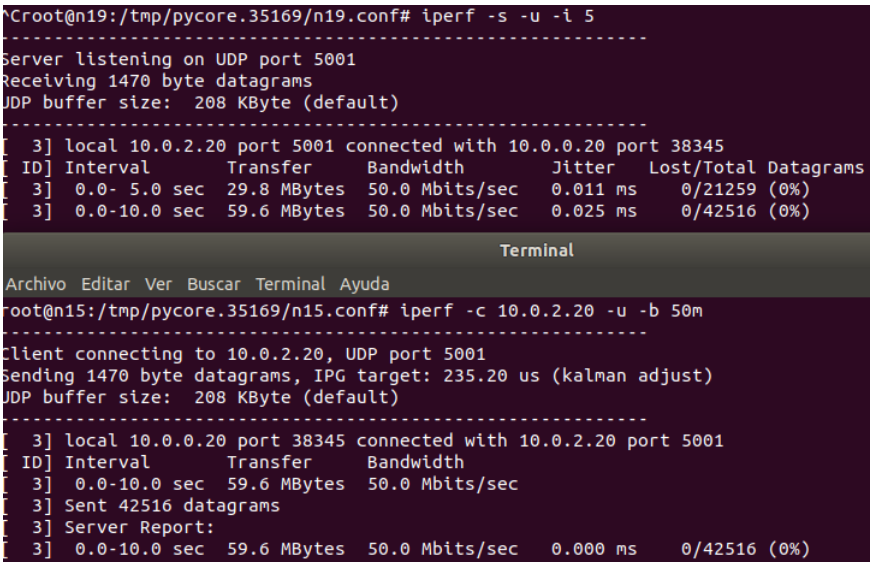


Figura 3.5: Flujo UDP con intervalos en IPerf.

Observamos que, para los parámetros indicados, la prueba se separa en dos intervalos, dado que indicamos intervalos de a cinco segundos y las pruebas por defecto son de a diez segundos.

3.2 NetPerf

NetPerf dispone de un amplio catálogo de pruebas que, en su mayoría, son derivadas de dos tipos de pruebas características, las pruebas de flujo y las pruebas de *request/response*. Además, dispone de la opción de forzar el cálculo de datos con las denominadas pruebas *omni*.

A continuación, procedemos a ver un ejemplo de los tipos de prueba principales.

Antes de comenzar, recordemos que, al igual que como IPerf, NetPerf sigue el formato de cliente-servidor.

1. Pruebas de flujos de datos: son las pruebas más básicas donde se ejecuta y analiza un único flujo de información.

   - Servidor: `netserver`
   - Cliente: `netperf -t TCP_STREAM -H 10.0.2.20 -- -m 16K`

- Cliente: `netperf -t UDP_STREAM -H 10.0.2.20 -- -R 1 -m 16K`
- Cliente: `netperf -H 10.0.2.20 -t TCP_MAERTS -- -s 128K -S 128K`
  - t <nombre_prueba>: indica cuál de los test disponibles por Netperf va a ser ejecutado.

TCP_STREAM: flujo TCP (Figura 3.6).

Es el test por defecto, por lo que es el que se realiza cuando no se especifica –t.

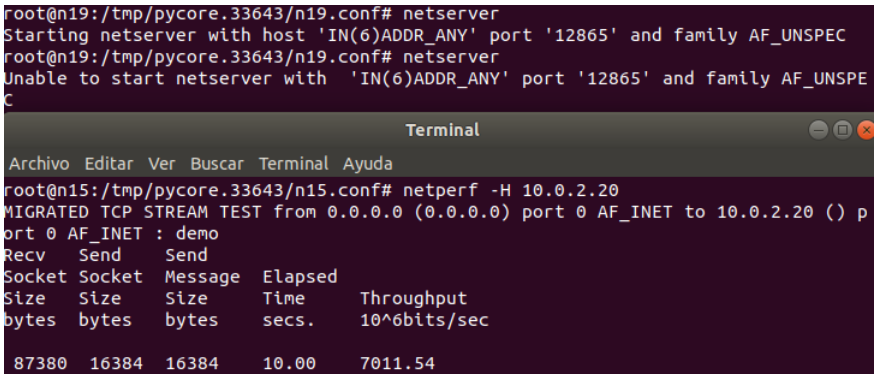


Figura 3.6: Flujo básico con NetPerf usando TCP.

UDP_STREAM: flujo UDP (Figura 3.7).

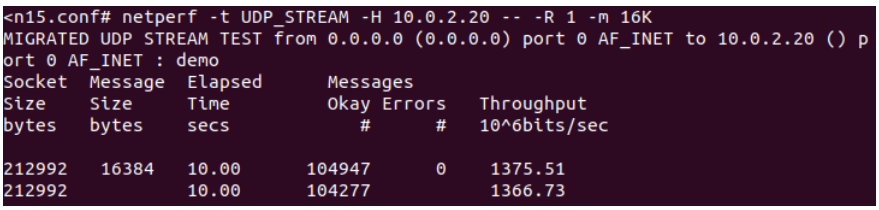


Figura 3.7: Flujo básico con NetPerf usando UDP.

TCP_MAERTS: flujo invertido TCP (Figura 3.8).

Envía un flujo de paquetes TCP de forma similar al TCP_STREAM, pero invirtiendo el sentido de los paquetes (nótese: MAERTS es STREAM al revés).

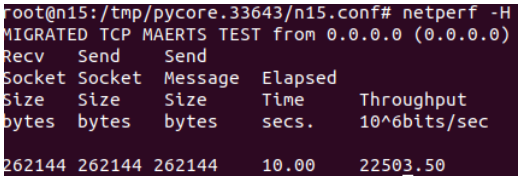


Figura 3.8: Flujo invertido con NetPerf.

2. Prueba de envío de archivos: se envía un archivo desde el cliente hasta el servidor y se analiza el comportamiento del envío (Figura 3.9).

   - Servidor: `netserver`
   - Cliente: `netperf -H 10.0.2.20 -F tsf -t TCP_SENDFILE -- -s 128K -S 128K`
     - F: indica el archivo que se usará en la prueba.

```
root@n15:/tmp/pycore.33643/n15.conf# netperf -H
TCP SENDFILE TEST from 0.0.0.0 (0.0.0.0) port 0
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

262144 262144 262144    10.00    16132.37
```

Figura 3.9: Prueba de envío con NetPerf.

   Si se trata de un archivo demasiado pequeño obtendremos el siguiente mensaje de error (Figura 3.10).

```
root@n15:/tmp/pycore.33643/n15.conf# netperf -H 10.0
TCP SENDFILE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_I
alloc_sendfile_buf_ring: specified file too small.
file must be larger than send_width * send_size
```

Figura 3.10: Muestra de error de paquete demasiado pequeño por prueba de envío con NetPerf.

3. Pruebas de petición - respuesta: NetPerf puede medir la relación entre las peticiones

   - Servidor: `netserver`
   - Cliente: `netperf -t TCP_RR -H 10.0.2.20`
   - Cliente: `netperf -t UDP_RR -H 10.0.2.20`

   Vemos el comportamiento empleando TCP (Figura 3.11) y empleando UDP (Figura 3.12)

```
root@n15:/tmp/pycore.33643/n15.conf# netperf -t TCP_RR
MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

16384  87380  1        1       10.00    11676.13
16384  87380
```

Figura 3.11: Prueba Petición-Respuesta con NetPerf usando TCP.

   Vemos como, igual que en casos anteriores, al hacer pruebas UDP muestra mayor número de estadísticas que el caso de TCP.

4. Pruebas omni: son un tipo de test que analizan los flujos emitidos con el fin de calcular las estadísticas especificadas dentro de las que dispone la herramienta.

- Servidor: `netserver`
- Cliente: `netperf -H 10.0.2.20 -t omni -- -d rr -k "THROUGHPUT,THROUGHPUT_UNITS"`
  - t omni: indica que se trata de un test omni
  - d: indica el tipo de prueba a realizar durante el test omni.
  - k: indica los parámetros que se quiere forzar a medir.

Con solo las estadísticas indicadas en este ejemplo obtendremos: Figura 3.13, pero disponemos de una muy amplia lista a escoger.

En este caso apreciamos el *throughput* medido en transmisiones por segundo, ya que se trata de un test tipo request/response (-d rr)

5. Pruebas concurrentes

   La herramienta no brinda una funcionalidad para hacer flujos de forma concurrente como hacía IPerf. En este caso sería necesario preparar un *script* que lanzase varias ejecuciones de Netperf desde el cliente para que las realizase simultáneamente.

3.3 MGEN

MGEN se trata de un generador de flujos que trabaja mediante eventos. Para ver como interactuar cómo la herramienta empezaremos probando a manejar un flujo instanciado con un nombre determinado y, después, un ejemplo de un *script* muy sencillo de como se suele emplear normalmente la herramienta.

Cabe destacar que MGEN, a diferencia de las demás herramientas, no muestra ningún resultado estadístico tras terminar los flujos, sino que necesita de otros programas para poder analizarlo (TRPR).

1. Pruebas de flujo básicas: también denominadas pruebas de instancia (o instanciadas). Generan un flujo de tráfico simple, al cual se le asigna un nombre al comienzo (nombre de instancia), pudiendo emplear dicho nombre para trabajar sobre el flujo al que hace referencia (de ahí que se denomine ”instanciadas”). Ejecutaremos la misma instancia en el equipo designado como servidor para ver los cambios en la consola.

   MGEN no emplea el formato cliente-servidor, pero ejecutaremos la misma instancia en el equipo que empleamos con las herramientas anteriores como servidor para poder ver cómo se muestran las acciones de la herramienta desde ambos equipos, tanto el designado como cliente como el designado como servidor.

   - Servidor: `mgen instance mgen1`

     Prueba básica con UDP:
   - Cliente: `mgen instance mgen1 event "on 1 udp dst 10.0.2.20/55000 periodic [1 1024]"`

- mgen1: nombre de la instancia.
- on 1: indica que se trata del lanzamiento de un flujo y el número que identifica el flujo
- dst <dirección_ip/puerto>: indica la dirección del flujo
- periodic [1 1024]: patrón periódico con periodo de envío de mensajes 1 segundo con tamaño de 1024 bytes.

Vemos como se muestra la creación de un flujo en la consola de la herramienta: Figura 3.14

2. Modificación del flujo: se provoca un cambio en un flujo que ya está siendo ejecutado.
   - Servidor: `mgen instance mgen1 event "mod 1 udp dst 10.0.2.20/5500 periodic [1 1024]"`
     - mod: indica que se trata de una modificación, la cual se ve en la consola (Figura 3.15)
3. Cierre del flujo: termina la ejecución de un flujo.
   - Cliente: `mgen instance mgen1 event "off 1".`
     – off: indica que se trata del fin de un flujo.
     – Estadísticas: recogemos una muestra de log empleando el argumento ”LOG <nombre>” y este lo empleamos para obtener una estimación de las estadísticas del flujo con la herramienta TRPR.
   - Servidor: `mgen instance mgen1`
   - Cliente: `mgen instance mgen1 event "on 1 udp dst 10.0.2.20/55000 periodic [1 1024]" LOG mgen.log`
     - LOG: permite sacar un archivo de log del flujo o conjunto de flujos empleados en la sesión anterior.
     - TRPR: programa de representación de datos que puede emplear los ficheros de log para calcular estadísticas y mostrarlas. Únicamente funciona la opción summary (mostrada en la Figura 3.16, que muestra las ratios de flujo de datos).

   Vemos una lectura de todos los movimientos registrados en el archivo de log y el cálculo de las estadísticas, las cuales se resumen en medir el tráfico en kb por segundo.
4. Prueba con Script: se emplea un *script* con comandos de generación o tratamiento de flujos, la herramienta permite poner en espera una prueba para que se ejecute en una hora determinada.

   Ejemplo de *script* sencillo que crea, modifica y elimina flujos:

```
0.0 ON 1 UDP SRC 5001 DST 127.0.0.1/5001 PERIODIC [1 1024]
0.0 ON 2 UDP SRC 5002 DST 224.225.1.2/5002 PERIODIC [1 512]
```

```
root@n15:/tmp/pycore.33643/n15.conf# netperf -T 1 -H 10.0.2.20 -t UDP_RR -c -C
MIGRATED UDP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10
 : cpu bind
Local /Remote
Socket Size   Request Resp.  Elapsed Trans.   CPU    CPU    S.dem   S.dem
Send   Recv   Size    Size   Time    Rate     local  remote local   remote
bytes  bytes  bytes   bytes  secs.   per sec  % S    % U    us/Tr   us/Tr

212992 212992 1       1      10.00   32803.69   26.77  -1.00  32.638  -1.000
212992 212992
```

Figura 3.12: Prueba Petición-Respuesta con NetPerf usando UDP.

```
root@n15:/tmp/pycore.33643/n15.conf# netperf -H 10.0.2.20 -t om
OMNI Send|Recv TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10
THROUGHPUT=11710.19
THROUGHPUT_UNITS=Trans/s
```

Figura 3.13: Prueba Omni de NetPerf.

```
root@n19:/tmp/pycore.33643/n19.conf# mgen instance mgen1
mgen: version 5.02b
mgen: starting now ...
13:46:46.944588 START Mgen Version 5.02b
13:46:50.239415 ON flow>1 srcPort>0 dst>10.0.2.20/55000
13:46:50.239453 DISCONNECT flow>1 srcPort>39927 dst>10.0.2.20/55000
Terminal
inal  Ayuda
n15.conf# mgen instance mgen1 event "on 1 tcp dst 10.0.2.20/55000 periodic [1 1024]"
```

Figura 3.14: Muestra de generación de flujo MGEN.

```
root@n19:/tmp/pycore.33643/n19.conf# mgen instance mgen1 event "mod
                                                                Termi
 Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@n15:/tmp/pycore.33643/n15.conf# mgen instance mgen1 event "on
mgen: version 5.02b
mgen: starting now ...
13:57:50.135505 START Mgen Version 5.02b
13:57:50.135566 ON flow>1 srcPort>0 dst>10.0.2.20/55000
mgen instance mgen1 event "0.0 MOD 1 DST 10.0.2.20/5500"
13:59:40.994982 OFF flow>1 srcPort>56509 dst>10.0.2.20/55000
14:07:04.814445 OFF flow>1 srcPort>56509 dst>10.0.2.20/5000
```

Figura 3.15: Muestra de modificación de flujo MGEN.

```
3 4.0 MOD 2 POISSON [10 1024]
4 10.0 OFF
5 110.0 OFF 2
```

- mgen input <script> START <hour:min:sec>[GMT]
- input: indica que se va a emplear un *script* para la ejecución.
- start: indica la hora en la que se va a ejecutar el *script*

GMT : si se incluye significa que se emplea la hora universal y no la del equipo.

3.4 D-ITG

D-ITG permite hacer pruebas de flujo convencionales empleando tanto TCP y UDP, así cómo datos de capa de aplicación como telnet o DNS. Además, pueden emplearse *scripts* para hacer envíos de flujos paralelos incluso de diferentes tipos de tráfico de forma simultánea. Una opción de D-ITG, que no disponen las demás herramientas, es el poder hacer pruebas de en las que el cliente pueda esperar a que el servidor esté disponible y ver la reacción.

1. Pruebas de flujo básicas: también denominadas pruebas activas con D-ITG. Inician un flujo de datos desde el cliente al servidor teniendo en cuenta las especificaciones indicadas como argumentos.
   - Servidor: `ITGRecv` # Ejecuta el servidor
   - Cliente: `ITGSend -T UDP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender1.log -x receiver1.log` # Ejecuta el cliente
   - Cliente: `ITGSend -T TCP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender4.log -x receiver4.log`
     - T: protocolo a emplear.
     - a: dirección IP destino del flujo.
     - c: tamaño constante de los paquetes.
     - C: indica tasa constante de envío de paquetes.
     - t: duración de la prueba en milisegundos.
     - l: nombre del archivo de log del envío.
     - x: nombre del archivo de log de la recepción.

   Para ver los datos, tomamos una captura del resultado mostrado tras emplear la opción de análisis de D-ITG con los archivos de log, ya que obtenemos un archivo de log tanto del cliente (Figura 3.17) como del Servidor (Figura 3.18)
2. Pruebas pasivas: indica al cliente que envíe tráfico al cliente en cuanto este esté al alcance, el cliente no espera como en la mayor parte de los casos, sino que solicita recibir el tráfico cuando este entra en contacto con el cliente especificado.
   - Cliente: `ITGSend -H -T UDP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender2.log -x receiver2.log`

```
manu@manu-VirtualBox:~/TRPR$ ./trpr input ../mgen11.log summary
TRPR Version 2.1b3
trpr: Adding default autoMatcher: *,*/*->*/*
trpr: At time 0.000000 - Adding flow: 0,d:0:47:0:0:0:47:0/0->0:f:0:0:d:0:47:0/0~0
trpr: At time 17.485238 - Adding flow: 0,0:0:d:0:47:0:0:f/0->0:0:d:0:47:0:0:f/0~0
trpr: At time 29.485138 - Adding flow: 0,0:0:d:0:47:0:f:0/0->d:0:47:0:0:0:47:0/0~0
trpr: At time 62.557411 - Adding flow: 0,0:0:d:0:48:0:0:f/0->0:0:d:0:48:0:0:f/0~0
trpr: At time 82.557520 - Adding flow: 0,48:0:0:f:0:0:d:0/0->48:0:0:f:0:0:d:0/0~0
trpr: At time 100.556957 - Adding flow: 0,0:0:d:0:48:0:f:0/0->d:0:48:0:0:0:48:0/0~0
trpr: At time 123.323593 - Adding flow: 0,0:0:d:0:49:0:0:f/0->0:0:d:0:49:0:0:f/0~0
trpr: At time 147.326319 - Adding flow: 0,0:0:d:0:49:0:f:0/0->d:0:49:0:0:0:49:0/0~0
trpr: At time 158.008484 - Adding flow: 0,0:0:d:0:49:0:49:0/0->0:0:5:49:0:f:0:d/0~0
trpr: At time 181.478317 - Adding flow: 0,0:0:d:0:49:0:0:f/0->0:0:d:0:50:0:0:f/0~0
trpr: At time 185.478764 - Adding flow: 0,0:0:d:0:50:0:0:f/0->0:0:d:0:50:0:0:f/0~0
trpr: At time 197.481682 - Adding flow: 0,d:0:50:0:0:0:50:0/0->f:0:d:0:50:0:0:f/0~0
#TRPR Summaries: range>0.000-204.070, windowSize>1.000
#flow>0,d:0:47:0:0:0:47:0/0->0:f:0:0:d:0:47:0/0~0, rate(kbps), ave>0.002146, min>0.000000, max>0.440000, dev>0.030656
#flow>0,0:0:d:0:47:0:0:f/0->0:0:d:0:47:0:0:f/0~0, rate(kbps), ave>0.017021, min>0.000000, max>0.320000, dev>0.071813
#flow>0,0:0:d:0:47:0:f:0/0->d:0:47:0:0:0:47:0/0~0, rate(kbps), ave>0.003636, min>0.000000, max>0.320000, dev>0.033918
#flow>0,0:0:d:0:48:0:0:f/0->0:0:d:0:48:0:0:f/0~0, rate(kbps), ave>0.029091, min>0.000000, max>0.320000, dev>0.091994
#flow>0,48:0:0:f:0:0:d:0/0->48:0:0:f:0:0:d:0/0~0, rate(kbps), ave>0.005203, min>0.000000, max>0.320000, dev>0.040472
#flow>0,0:0:d:0:48:0:f:0/0->d:0:48:0:0:0:48:0/0~0, rate(kbps), ave>0.003048, min>0.000000, max>0.320000, dev>0.031080
#flow>0,0:0:d:0:49:0:0:f/0->0:0:d:0:49:0:0:f/0~0, rate(kbps), ave>0.031220, min>0.000000, max>0.320000, dev>0.094950
#flow>0,0:0:d:0:49:0:f:0/0->d:0:49:0:0:0:49:0/0~0, rate(kbps), ave>0.005517, min>0.000000, max>0.320000, dev>0.041654
#flow>0,0:0:d:0:49:0:49:0/0->0:0:5:49:0:f:0:d/0~0, rate(kbps), ave>0.006809, min>0.000000, max>0.320000, dev>0.046178
#flow>0,0:0:d:0:49:0:0:f/0->0:0:d:0:50:0:0:f/0~0, rate(kbps), ave>0.013333, min>0.000000, max>0.320000, dev>0.063944
#flow>0,0:0:d:0:50:0:0:f/0->0:0:d:0:50:0:0:f/0~0, rate(kbps), ave>0.080000, min>0.000000, max>0.320000, dev>0.138564
#flow>0,d:0:50:0:0:0:50:0/0->f:0:d:0:50:0:0:f/0~0, rate(kbps), ave>0.055000, min>0.000000, max>0.440000, dev>0.145516
#flow>Summary, rate(kbps), ave>0.021002, min>0.000000, max>0.440000, dev>0.023366,
trpr: Done.
```

Figura 3.16: Muestra de análisis de log MGEN con TRPR.

```
****************  TOTAL RESULTS   ******************

Number of flows          =            1
Total time               =     14.906876 s
Total packets            =          149
Minimum delay            =      0.000000 s
Maximum delay            =      0.000000 s
Average delay            =      0.000000 s
Average jitter           =      0.000000 s
Delay standard deviation =      0.000000 s
Bytes received           =         14900
Average bitrate          =      7.996310 Kbit/s
Average packet rate      =      9.995387 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
```

Figura 3.17: Prueba básica en el lado cliente con D-ITG.

```
****************  TOTAL RESULTS   ******************

Number of flows          =            1
Total time               =     14.906874 s
Total packets            =          149
Minimum delay            =      0.000088 s
Maximum delay            =      0.000343 s
Average delay            =      0.000128 s
Average jitter           =      0.000026 s
Delay standard deviation =      0.000039 s
Bytes received           =         14900
Average bitrate          =      7.996311 Kbit/s
Average packet rate      =      9.995389 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
```

Figura 3.18: Prueba básica en el lado servidor con D-ITG.

- Servidor: `ITGRecv -H 10.0.0.20`
  - H: especifica que se está operando en modo pasivo.

Las pruebas pasivas no generan logs en el lado cliente, únicamente la recepción del servidor (Figura 3.19).

```
manu@manu-VirtualBox:~/DITG_LOGS$ ITGDec receiver2.log
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
Empty log file


****************  TOTAL RESULTS   ******************
Number of flows          =                0
Total time               = -90000.000000 s
Total packets            =                0
Minimum delay            =  90000.000000 s
Maximum delay            = -86400.000000 s
Average delay            =            -nan s
Average jitter           =        0.000000 s
Delay standard deviation =        0.000000 s
Bytes received           =                0
Average bitrate          =        0.000000 Kbit/s
Average packet rate      =        0.000000 pkt/s
Packets dropped          =                0 (0.00 %)
Average loss-burst size  =                0 pkt
Error lines              =                0
```

Figura 3.19: Prueba pasiva D-ITG.

La medición de los datos en este tipo de pruebas no cuenta con demasiado rigor, prácticamente todas las pruebas de este tipo dieron un resultado como el anteriormente indicado, el cual no tiene utilidad.

3. Prueba con servidor de Logs: D-ITG posee dentro de su funcionalidad el asignar un host como servidor de Log, enviando al equipo designado los resultados de las pruebas.

   - Servidor Log: ITGLog <- IP: 10.0.3.21 <- Servidor de logs
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -T UDP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender3.log -x receiver3.log -L 10.0.3.21 -X 10.0.3.21`
     - L y –X: Actúan igual que –l y –x, pero enviando los archivos a los servidores de Logs de las ips designadas. (Salida: Figura 3.20)

   Prueba con scripts

   - Servidor: `ITGRecv`
   - Cliente: `ITGSend Archivo_Script -l sender13.log -x receiver13.log`

   Se indica la ruta del archivo con el *script* en vez de usar los argumentos

4. Ejemplo de *script* para DITG:

```
-a 10.0.2.20 -rp 10001 -C 100 -c 100 -T UDP
-a 10.0.2.20 -rp 10002 -C 200 -c 200 -T UDP
-a 10.0.2.20 -rp 10003 -C 300 -c 300 -T UDP
```

Mostramos los resultados de una prueba en la que se emplea el *script* anterior, apreciamos cómo evalúa los flujos por separado y que muestra la suma total de todo el tráfico generado por los tres flujos. Este e caso muestra las estadísticas de tres flujos y la suma total de estas, debido a esto hemos necesitado emplear más figuras que en ejemplos anteriores:

Figura 3.21 y Figura 3.22 muestran el envío desde el cliente y Figura 3.23 y Figura 3.24 la recepción desde el servidor.

Podemos apreciar como el archivo de log muestra todos los flujos enviados de forma paralela y como muestra un resultado total donde calcula las estadísticas teniendo en cuenta todo el tráfico sumado durante los envíos de los 3 flujos.

5. Pruebas con tráfico de capa de aplicación: D-ITG es capaz de generar tráfico de capa de aplicaciones como tráfico DNS o de Voz IP.
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -a 10.0.2.20 -rp 10000 DNS -c 100 -C 10 -t 15000 -l sender7.log -x receiver7.log`
     - rp: indica el puerto y el tipo de tráfico asignado a dicho puerto

3.5 TRex

TRex destaca entre las demás herramientas porque no realiza un test y luego da los resultados, si no que genera un flujo de datos según se le especifique y monitoriza las entradas y salidas del equipo empleado. Para la realización de las pruebas de TRex, empleamos un *router* que está conectado directamente entre los *switches* de las redes 1.0.0.0 y 10.0.2.0.

Probaremos un ejemplo de cada tipo de prueba de TRex, una *stateful*, poniendo un ejemplo de una plantilla que emplee más de un tipo de tráfico, y otra *stateless*, en la que mostraremos como interactuar con la herramienta empleando su consola.

3.5.1 Stateful

Emplean plantillas *Yaml* para configurar las pruebas. Las plantillas disponen de 3 partes diferenciados:

1. Características de la prueba: indica los parámetros que afectan directamente a la ejecución de la prueba (en este caso solo hemos indicado la duración).

```
< -C 10 -t 15000 -l sender3.log -x receiver3.log -L 10.0.3.21 -X 10.0.3.21
ITGSend version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
Started sending packets of flow ID: 1
Finished sending packets of flow ID: 1

root@n15:/tmp/pycore.42673/n15.conf#
```

```
                                                    Terminal
 Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@n19:/tmp/pycore.42673/n19.conf# ITGRecv
ITGRecv version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 8999
Finish on UDP port : 8999
```

```
                                                    Terminal
 Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@n22:/tmp/pycore.42673/n22.conf# ITGLog
ITGLog version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
Press Ctrl-C to terminate!
Name Log File : receiver3.log
Protocol used : UDP
Receiving data on port: 9460
Name Log File : sender3.log
Protocol used : UDP
Receiving data on port: 9662
Data transmission ended on UDP channel
Data transmission ended on UDP channel
ls
^CFinish with CRTL-C!
root@n22:/tmp/pycore.42673/n22.conf# ls
defaultroute.sh  receiver3.log  sender3.log  var.log  var.run
root@n22:/tmp/pycore.42673/n22.conf#
```

Figura 3.20: Muestra de comportamiento del servidor de Logs de D-ITG

```
manu@manu-VirtualBox:~/DITG_LOGS$ ITGDec sender13.log
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
/------------------------------------------------------------
Flow number: 3
From 10.0.0.20:38782
To    10.0.2.20:10003
------------------------------------------------------------
Total time                =      9.999966 s
Total packets             =          2538
Minimum delay             =      0.000000 s
Maximum delay             =      0.000000 s
Average delay             =      0.000000 s
Average jitter            =      0.000000 s
Delay standard deviation  =      0.000000 s
Bytes received            =        761400
Average bitrate           =    609.122071 Kbit/s
Average packet rate       =    253.800863 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =      0.000000 pkt
------------------------------------------------------------
------------------------------------------------------------
Flow number: 2
From 10.0.0.20:45291
To    10.0.2.20:10002
------------------------------------------------------------
Total time                =      9.997141 s
Total packets             =          1796
Minimum delay             =      0.000000 s
Maximum delay             =      0.000000 s
Average delay             =      0.000000 s
Average jitter            =      0.000000 s
Delay standard deviation  =      0.000000 s
Bytes received            =        359200
Average bitrate           =    287.442180 Kbit/s
Average packet rate       =    179.651362 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =      0.000000 pkt
------------------------------------------------------------
```

Figura 3.21: Pruebas simultáneas con *script* en el lado cliente con D-ITG parte 1.

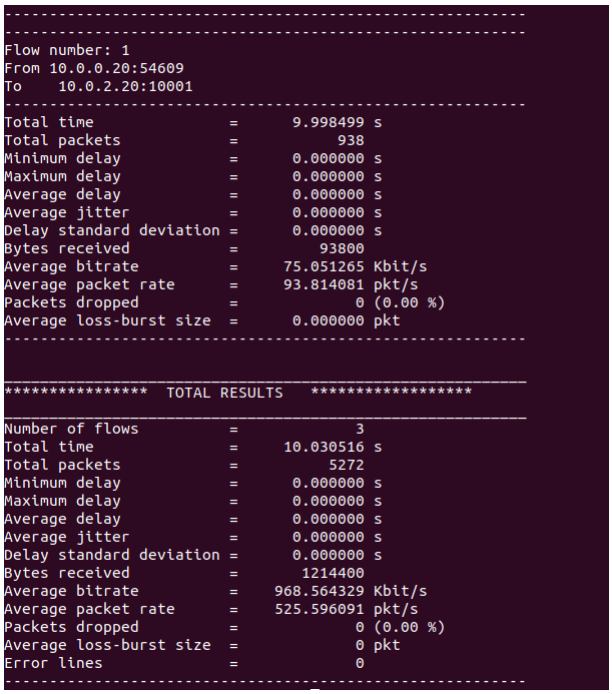


Figura 3.22: Prueba simultáneas con *script* en el lado cliente con D-ITG parte 2.

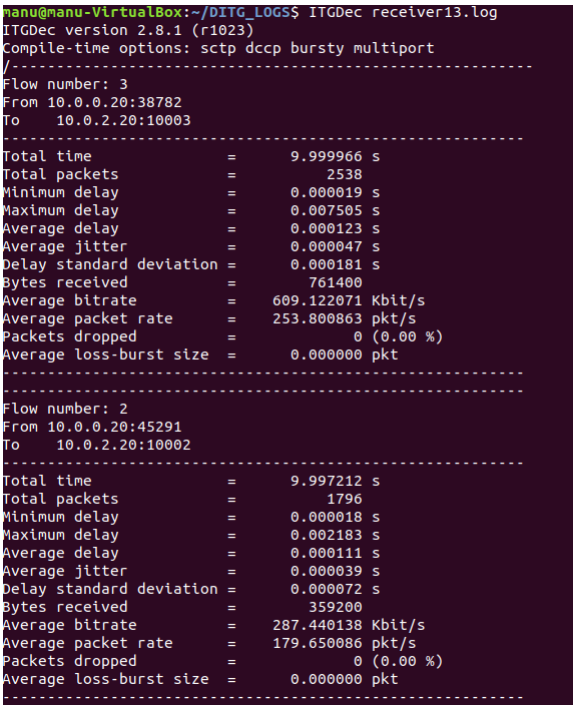


Figura 3.23: Prueba simultáneas con *script* en el lado servidor con D-ITG parte 1.

```
------------------------------------------------------------
------------------------------------------------------------
Flow number: 1
From 10.0.0.20:54609
To    10.0.2.20:10001
------------------------------------------------------------
Total time                =      9.998454 s
Total packets             =           938
Minimum delay             =      0.000019 s
Maximum delay             =      0.001211 s
Average delay             =      0.000106 s
Average jitter            =      0.000037 s
Delay standard deviation  =      0.000056 s
Bytes received            =         93800
Average bitrate           =     75.051603 Kbit/s
Average packet rate       =     93.814504 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =      0.000000 pkt
------------------------------------------------------------


****************  TOTAL RESULTS   *****************
------------------------------------------------------------
Number of flows           =             3
Total time                =     10.030532 s
Total packets             =          5272
Minimum delay             =      0.000018 s
Maximum delay             =      0.007505 s
Average delay             =      0.000116 s
Average jitter            =      0.000042 s
Delay standard deviation  =      0.000135 s
Bytes received            =       1214400
Average bitrate           =    968.562784 Kbit/s
Average packet rate       =    525.595253 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =             0 pkt
Error lines               =             0
------------------------------------------------------------
```

Figura 3.24: Prueba simultáneas con *script* en el lado servidor con D-ITG parte 2.

```
****************  TOTAL RESULTS   ******************
------------------------------------------------------------
Number of flows           =             1
Total time                =     14.968301 s
Total packets             =           150
Minimum delay             =      0.000000 s
Maximum delay             =      0.000000 s
Average delay             =      0.000000 s
Average jitter            =      0.000000 s
Delay standard deviation  =      0.000000 s
Bytes received            =         15000
Average bitrate           =      8.016942 Kbit/s
Average packet rate       =     10.021177 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =             0 pkt
Error lines               =             0
```

Figura 3.25: Muestra de log del lado cliente a prueba DNS.

```
****************  TOTAL RESULTS   ******************
------------------------------------------------------------
Number of flows           =             1
Total time                =     14.968410 s
Total packets             =           150
Minimum delay             =      0.000084 s
Maximum delay             =      0.000199 s
Average delay             =      0.000117 s
Average jitter            =      0.000015 s
Delay standard deviation  =      0.000024 s
Bytes received            =         15000
Average bitrate           =      8.016884 Kbit/s
Average packet rate       =     10.021104 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =             0 pkt
Error lines               =             0
```

Figura 3.26: Muestra de log del lado servidor a prueba DNS.

2. Características de la generación del flujo: indicamos parámetros como la distribución, el rango de equipos que actúan de clientes o servidores, etc.
3. Información de las capturas: se listan los archivos de capturas de tráfico a emplear junto con algunos parámetros que afectarán a cada uno por separado durante la ejecución de la prueba.

A continuación, vemos un ejemplo de una de estas plantillas.

```
-duration : 10.0
 Generator :
        distribution        : ""seq         Distribución que
    seguirá el flujo
        clients_start       : ""10.0.0.20   Dirección donde empieza
    el rango de clientes
        clients_end         : ""10.0.0.255  Dirección donde termina
    el rango de clientes
        servers_start       : ""10.0.0.20   Dirección donde empieza
    el rango de servidores
        servers_end         : ""10.0.0.255  Dirección donde termina
    el rango de servidores
        Clients_per_gb      : 201
        Min_clients         : 101
        Dual_port_mask      : ""1.0.0.0
        Tcp_aging           : 1
        Udp_aging           : 1

Cap_info
    -name : /home/manu/TRexTemplates/paquetes/dns.pcap
     cps     : 1.0          Conexiones por segundo
     Ipg     : 10000        Espacio entre paquetes en microsegundos.
     Rtt     : 10000        Debe indicarse el mismo valor que en Ipg
     W       : 1

    -name : /home/manu/TRexTemplates/paquetes/udp\_64B.pcap
     Cps     : 1.0
     Ipg     : 10000
     Rtt     : 10000
     W       : 1

    -name : /home/manu/TRexTemplates/paquetes/http.pcap
     Cps     : 1.0
     Ipg     : 10000
     Rtt     : 10000
     W       : 1
```

Desde el directorio donde está la herramienta:

- Comando: `./t-rex-64 -f /home/manu/TRexTemplates/mixto.yaml -c 1 -m 10 -d 10`
  - c: indica número de CPUs a emplear en la prueba.
  - m: multiplicador del tráfico.
  - d: duración de la prueba en segundos

Los datos se van viendo cómo se actualizan durante el transcurso de la prueba, al terminar esta se nos muestra un resumen como el siguiente mostrado aquí: Figura 3.27

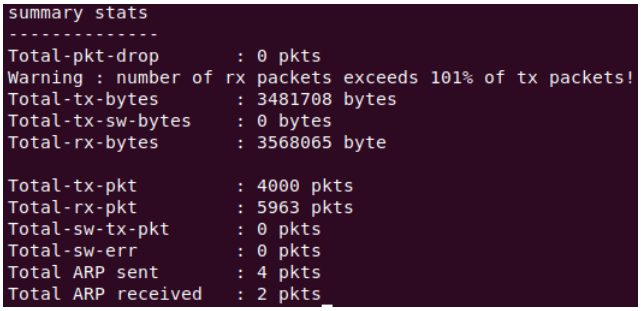


Figura 3.27: Ejemplo de salida de una prueba de TREX Stateful.

En estas pruebas, el Total-tx-bytes (transmisiones) y el Total-rx-bytes (recepciones) deberían ser iguales, pero se crean paquetes de más durante el flujo por cómo está dispuesto el escenario de las pruebas, por lo que puede dar el warning visto en la captura.

3.5.2 Stateless

Son pruebas que se realizan mediante comandos sin el empleo de plantillas de configuración como los casos anteriores. El tráfico se genera tanto a partir de muestras ".pcap" como las usadas por las plantillas como a partir de scripts Python. Requiere de la ejecución del daemon del que dispone TRex antes de ejecutar la consola.

Previo:

1. Arrancar daemon: `./t-rex-64 -i`
2. Abrir consola: `./trex-console`

Pruebas:

- Arrancar flujo con *script* de Python: `start -f <ruta script.py> -m 10mbps -a`
  - f: indica el fichero a usar
  - m: indica la velocidad de los mensajes del flujo.
  - a: indica que es para todos los puertos, puede usarse -p <nº puerto> para ser específicos.

- Arrancar flujo con muestra .pcap: push –f <ruta captura .pcap>
- Ver el estado del flujo: stats Podemos ver en la parte superior de la consola un resumen del estado de la máquina (Figura 3.28) y en la parte inferior vemos los datos de cada puerto configurado en la herramienta (Figura 3.29).

```
trex>stats
Global Statistitcs

connection   : localhost, Port 4501                total_tx_L2  : 2.83 b/sec

version      : STL @ v2.54                         total_tx_L1  : 4.04 b/sec

cpu_util.    : 0.06% @ 1 cores (1 per dual port)   total_rx     : 53.39 b/sec

rx_cpu_util. : 0.0% / 0.09 pkt/sec                 total_pps    : 0.01 pkt/sec

async_util.  : 0.04% / 1.26 KB/sec                 drop_rate    : 0 b/sec

                                                   queue_full   : 0 pkts


Port Statistics
```

Figura 3.28: Ejemplo de salida de una prueba de TREX Stateless parte 1.

```
Port Statistics

   port    |         0         |         1         |       total
-----------+-------------------+-------------------+------------------
owner      |              root |              root |
link       |                UP |                UP |
state      |      TRANSMITTING |      TRANSMITTING |
speed      |           10 Gb/s |           10 Gb/s |
CPU util.  |             0.06% |             0.06% |
--         |                   |                   |
Tx bps L2  |          1.41 bps |          1.41 bps |          2.83 bps
Tx bps L1  |          2.02 bps |          2.02 bps |          4.04 bps
Tx pps     |             0 pps |             0 pps |          0.01 pps
Line Util. |               0 % |               0 % |
---        |                   |                   |
Rx bps     |          25.2 bps |         28.19 bps |         53.39 bps
Rx pps     |          0.04 pps |          0.05 pps |          0.09 pps
----       |                   |                   |
opackets   |                 2 |                 2 |                 4
ipackets   |                17 |                24 |                41
obytes     |               110 |               110 |               220
ibytes     |              1324 |              1732 |              3056
tx-pkts    |            2 pkts |            2 pkts |            4 pkts
rx-pkts    |           17 pkts |           24 pkts |           41 pkts
tx-bytes   |             110 B |             110 B |             220 B
rx-bytes   |          1.32 KB  |          1.73 KB  |          3.06 KB
-----      |                   |                   |
oerrors    |                 0 |                 0 |                 0
ierrors    |                 0 |                 0 |                 0
```

Figura 3.29: Ejemplo de salida de una prueba de TREX Stateless parte 2.

- tx: transmisiones
- rx: recepciones
- L1 y L2: capas 1 y 2 del modelo OSI, no muestra capa 3. aunque si se comporta diferente si ejecutas TCP, UDP y otro tipo de protocolo de capa 3 o superior.

Otros comandos:

- Pause: pausa uno o varios flujos
- Resume: reanuda uno o varios flujos pausados.
- Stop: cierra uno o varios flujos
- Uptade: modifica la velocidad de los puertos
- Streams: da información básica de los flujos en ejecución.

A continuación, pasaremos a ver algunas de las pruebas descritas en este capítulo en un escenario donde se modifica alguno de los enlaces con el fin de ver cómo influye dichas diferencias en el funcionamiento de las herramientas.

Capítulo 4

Comportamiento de las herramientas de generación de tráfico ante anomalías en la red

En este apartado realizamos una serie de pruebas de cada herramienta tras hacer modificar valores de algunas de las conexiones dadas en el escenario. La razón será ver el comportamiento de estas según las condiciones y comprobar si su funcionamiento es adecuado o, por el contrario, no funciona correctamente con alguna de las limitaciones simuladas.

Para la realización de estas pruebas emplearemos dos escenarios (Figura 4.1 y Figura 4.2) , prácticamente iguales, pero uno dispone de un equipo que el otro no. Esto es debido a que el equipo con el que se probaba TRex representaba un obstáculo a la hora de probar las demás herramientas, ya que presentaba una ruta alternativa por donde circulaban los datos. Además, al realizar las pruebas con TRex, realizaremos las modificaciones en uno de los interfaces directamente conectados a la maquina empleada, ya que usar mismo enlace que las demás herramientas repercutía en menor grado sobre los resultados que ofrecía TRex.

Por simplicidad, modificamos únicamente un parámetro de los que el emulador de red CORE permite modificar en una conexión de red: ancho de banda, *delay*, *jitter*, pérdida de paquetes y duplicidad.

Los valores empleados durante estas pruebas son los siguientes.

- Ancho de banda: 1, 2, 5 y 10 Mb/s.
- *Delay*: 0.1, 0.2, 0.5 y 1 segundos.
- *Jitter*: 2.5, 3, 5 y 10 milisegundos.
- Pérdida de paquetes: 3, 5, 10 y 20 por ciento.
- Duplicidad: 3, 5, 10 y 20 por ciento.

Realizaremos las pruebas empleando tanto el protocolo TCP como UDP para contemplar las diferencias del funcionamiento de la herramienta con cada protocolo en cada caso. Además,

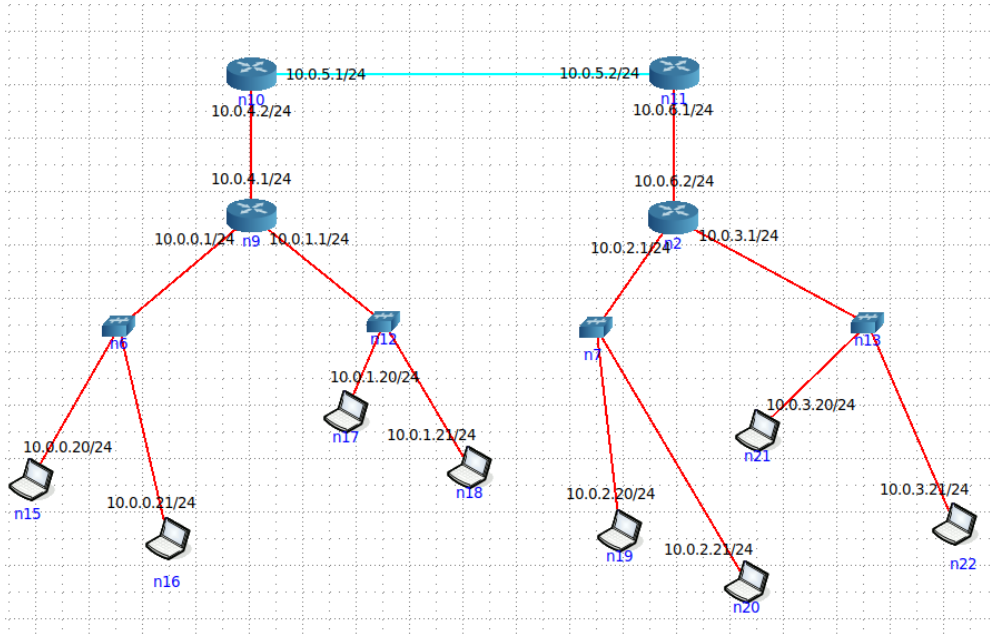


Figura 4.1: Versión del escenario para las herramientas excepto TREX.

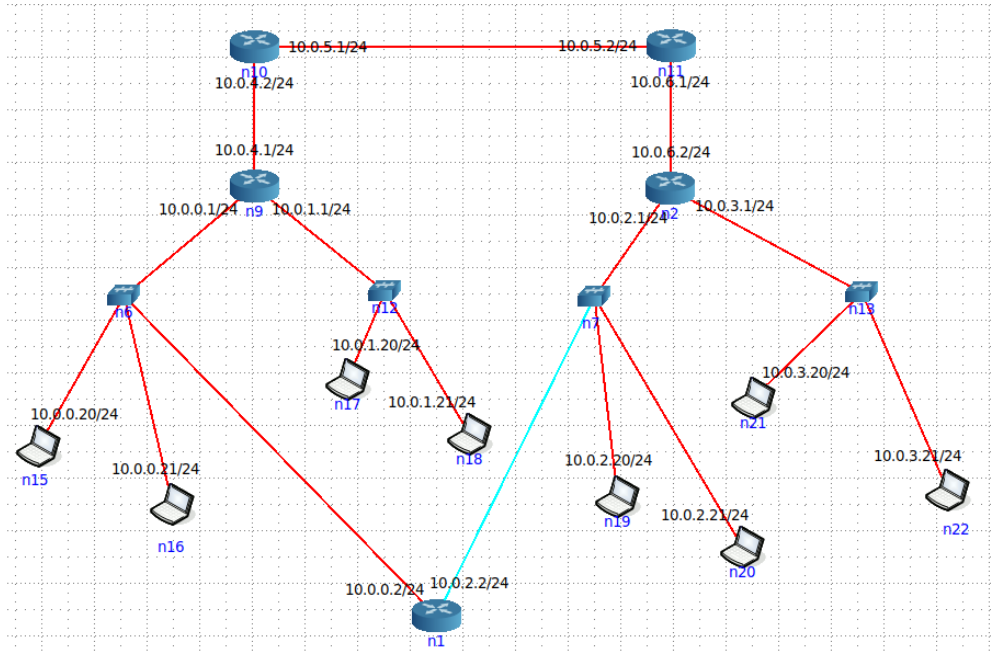


Figura 4.2: Versión del escenario para TREX.

en el caso de D-ITG, realizamos a mayores una serie de pruebas empleando VoIP ya que consideramos interesante el probar la funcionalidad de emplear tráfico de capa de aplicaciones que caracteriza a esta herramienta.

4.1 Comandos empleados durante las pruebas

Tratamos de realizar las simulaciones sin emplear ninguna variable a mayores en los comandos que pudiera entorpecer la toma de datos de las pruebas, pero nos encontramos con el problema que al trabajar con UDP las herramientas únicamente envían una trama y continúan la simulación sin mayor movimiento de tráfico, por este motivo tratamos de forzar el tráfico para las pruebas a unos 50Mb y emplearemos dicho volumen de tráfico tanto con TCP como con UDP con el fin de valorar las diferencias a en el comportamiento de la herramienta con ambos protocolos.

- IPerf
  - Caso básico TCP
    - Servidor: `iperf -s`
    - Cliente: `iperf -c 10.0.2.20 -b 50M`
  - Caso básico UDP
    - Servidor: `iperf -s -u`
    - Cliente: `iperf -c 10.0.2.20 -u -b 50M`
  - B: fuerza a que el envío sea con un ancho de banda de 50Mb/s.
- NetPerf
  - Test Omni exhibiendo el *throughput* y el *delay*.
    - Servidor: `netserver`
    - Cliente: `netperf -H 10.0.2.20 -j -t omni -T TCP -- -s 50M -S 50M -m 50M -d stream -k "THROUGHPUT, THROUGHPUT_UNITS, MIN_LATENCY, MAX_LATENCY"`
    - Cliente: `netperf -H 10.0.2.20 -j -t omni -T UDP -- -s 50M -S 50M -m 50M -d stream -k "THROUGHPUT, THROUGHPUT_UNITS, MIN_LATENCY, MAX_LATENCY"`
  - H: Dirección del servidor.
  - j: permite hacer mediciones de estadísticas adicionales como el *MAX_LATENCY*.
  - t: tipo de test, señalamos que es un test omni.
  - T: protocolo empleado para la prueba.
  - s: tamaño del buffer local.
  - S: tamaño del buffer remoto (servidor).
  - m: tamaño del mensaje a enviar.

  - d: indica el tipo de prueba a realizar durante el test omni
  - k: indica los parámetros que se quiere forzar a medir.

- MGEN
  - Debido a que MGEN no muestra la información estadística correctamente, se decidió no mostrar la información de las pruebas de esta herramienta en estos casos. Tanto en TCP como en UDP siempre muestra exactamente los mismos valores a pesar de las modificaciones en el comando para lanzar la prueba.

- D-ITG
  - Prueba con TCP
    * Servidor: `ITGRecv`
    * Cliente: `ITGSend -T TCP -a 10.0.2.20 -C 50000 -t 30000 -l sender.log -x receiver.log`
  - Prueba con UDP
    * Servidor: `ITGRecv`
    * Cliente: `ITGSend -T UDP -a 10.0.2.20 -C 50000 -t 30000 -l sender.log -x receiver.log`
  - Pruebas con VoIP
    * Servidor: `ITGRecv -l logfile`
    * Cliente: `ITGSend -a 10.0.2.20 VoIP -x G.711.2 -h RTP -VAD`

  Para TCP y UDP:

  - T: indica que protocolo se empleará.
  - a: indica la dirección a la que se dirige el flujo.
  - C: indica una ratio constante de paquetes por segundo.
  - t: establece el tiempo de prueba.
  - l: saca y nombra el paquete log correspondiente al cliente.
  - x: saca y nombra el paquete log correspondiente al servidor.

  Para VoIP:

  - a: indica la dirección a donde se envía el flujo de datos.
  - x: indica el codificador para la información.
  - h: indica que protocolo de VoIP usa.
  - VAD : activa la detección de actividad de voz.

- T-Rex
  - Realizamos la prueba con modo Stateless empleando dos scripts Python, uno para TCP y otro para UDP, para generar los flujos de datos. Dadas las características de T-Rex, al hacer los cambios sobre el enlace que modificamos durante las pruebas se veían en tiempo real como cambian los paquetes en la interfaz de dicho enlace.

4.2 Escenario original

Mostraremos un ejemplo de cada caso sin alterar el escenario para emplear como punto de referencia. También aprovecharemos para explicar que parámetros se mostrarán en las gráficas para facilitar la comprensión de los ejemplos posteriores.

1. *Throughput*: Datos recibidos por el servidor por unidad de tiempo.
2. Paquetes enviados: Datos que salen del cliente rumbo al servidor durante la ejecución de la prueba.
3. Paquetes recibidos: Datos que logran llegar al servidor durante la prueba.
4. *Delay*: retardo medio registrado durante el envío de los datos.
5. *Delay* máximo: el mayor retardo registrado (empleado en NetPerf).
6. *Jitter*: fluctuación del retardo.
7. Paquetes perdidos (%): porcentaje de paquetes que no alcanzan el servidor: (1 = 100%, 0.5 = 50%)
8. Transmisiones L2 Puerto 1: Datos de capa dos salidos por el puerto de red asignado uno por la herramienta.
9. Transmisiones L1 Puerto 1: Datos de capa uno salidos por el puerto de red asignado uno por la herramienta.
10. Recepciones Puerto 1: Datos totales recibidos por el puerto de red asignado uno por la herramienta.

IPerf muestra únicamente el *throughput* y los paquetes recibidos al emplear el protocolo TCP en las pruebas (Figura 4.3). A la hora de emplear UDP obtenemos además de los anteriores el *jitter* y el porcentaje de paquetes perdidos (Figura 4.4). IPerf es la única herramienta que muestra esta distinción a la hora de calcular sus estadísticas entre ambos protocolos.

NetPerf indica el *throughput*, los paquetes enviados y recibidos, el *delay* máximo y el porcentaje de paquetes perdidos (Figura 4.5 y Figura 4.6).

D-ITG coincide en los datos con NetPerf salvo el *delay*, el cual calcula el *delay* medio y no el máximo, y que D-ITG es capaz de calcular el *jitter* (Figura 4.7 y Figura 4.8). Sin embargo, no muestra los paquetes enviados al emplear el protocolo VoIP (Figura 4.9). TRex muestra las trasmisiones en capa 1 y 2 (transmisiones L2 y L1) y el total de las recepciones del puerto conectado al enlace modificado para las pruebas (Figura 4.10 y Figura 4.11).

El *throughput* lo medimos en Mb/s, los paquetes enviados y recibidos en MB, el *delay* y *delay* máximo en segundos, el *jitter* en milisegundos, y el porcentaje de paquetes perdidos se mide en siempre sobre 1, siendo 1 el 100Las transmisiones y recepciones de TRex se miden todas en Mb/s.

Solo el caso de D-ITG con VoIP emplea una escala menor para el *throughput* y los paquetes recibidos, siendo el cálculo en Kb/s y KB respectivamente.

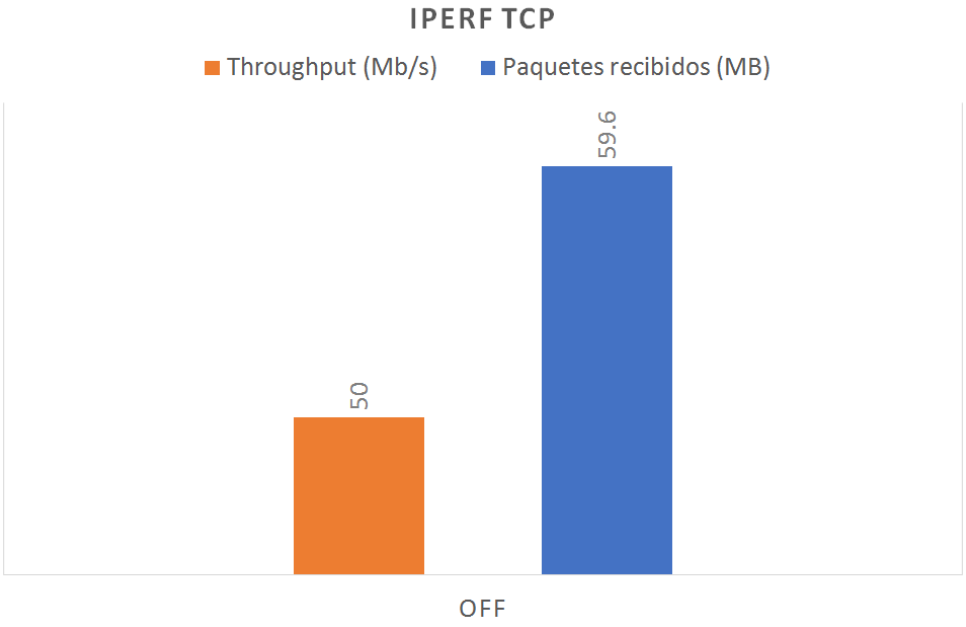


Figura 4.3: IPerf con protocolo TCP en escenario original.

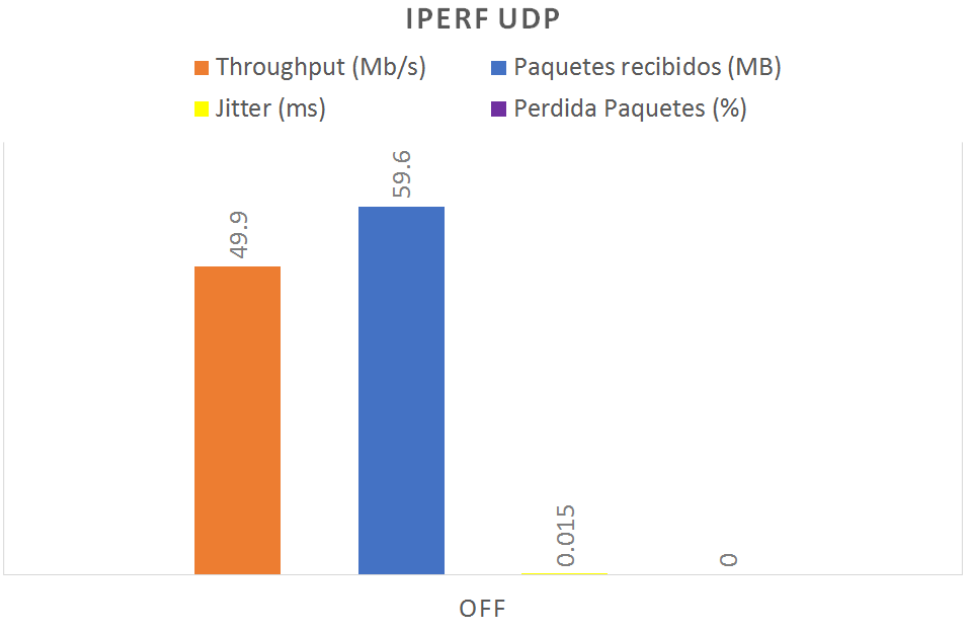


Figura 4.4: IPerf con protocolo UDP en escenario original.

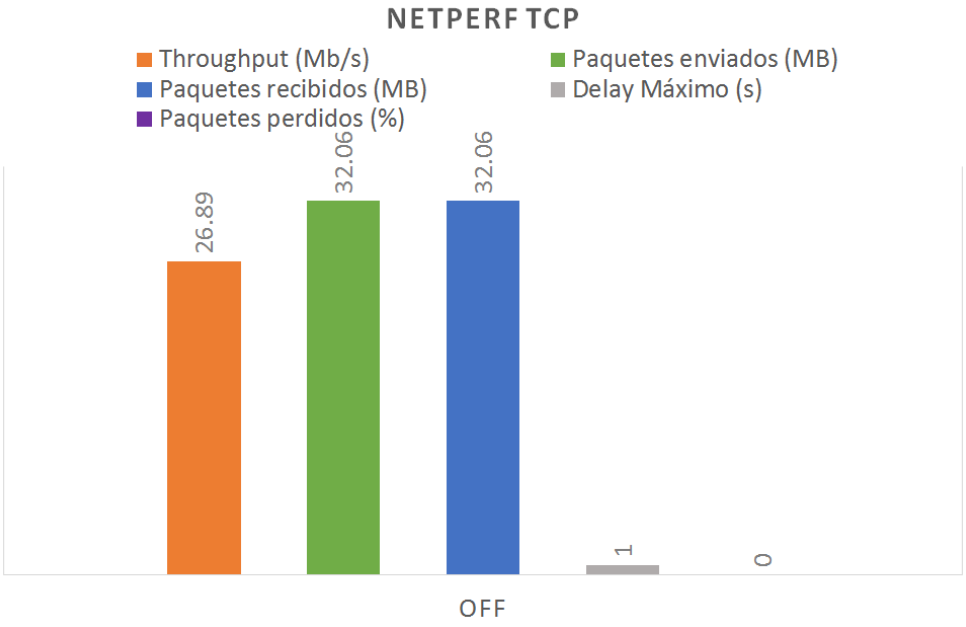


Figura 4.5: NetPerf con protocolo TCP en escenario original.

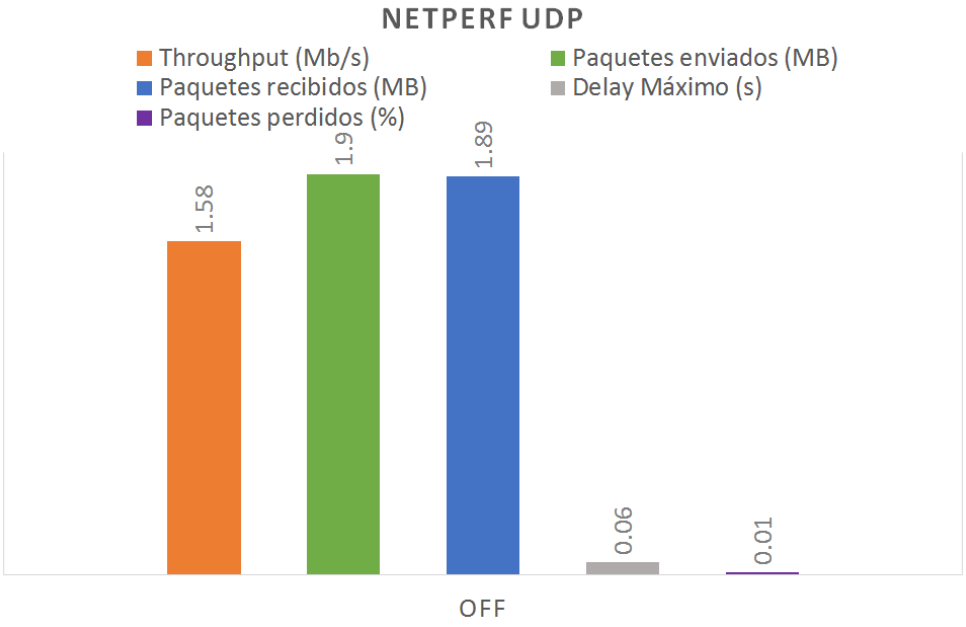


Figura 4.6: NetPerf con protocolo UDP en escenario original.

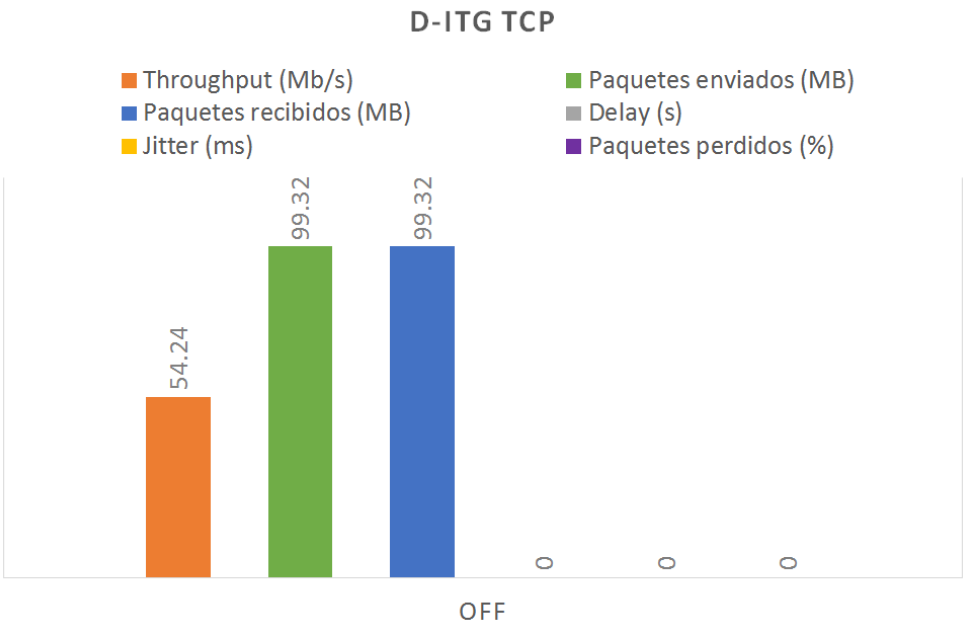


Figura 4.7: D-ITG con protocolo TCP en escenario original.

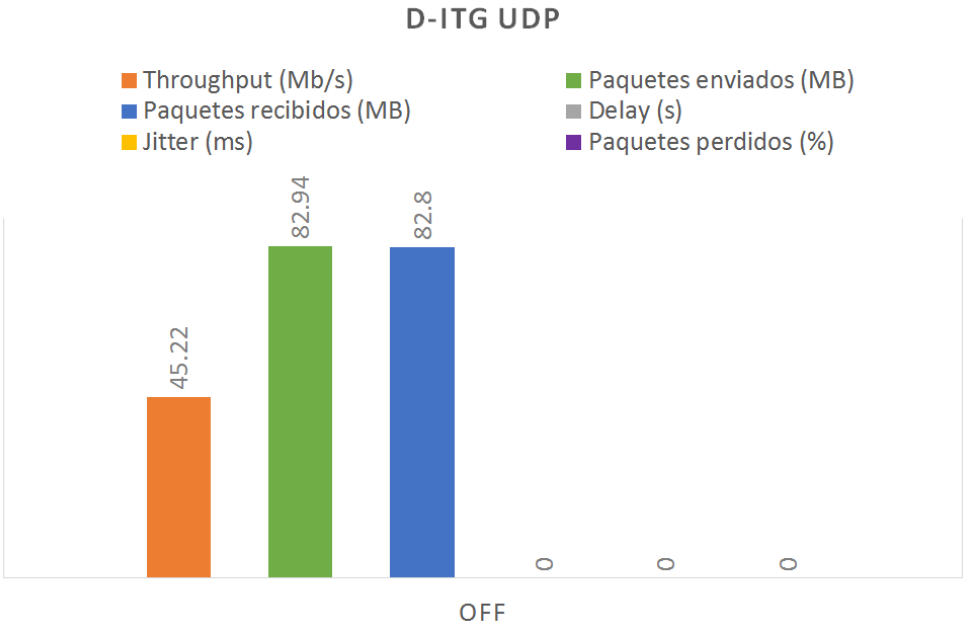


Figura 4.8: D-ITG con protocolo UDP en escenario original.

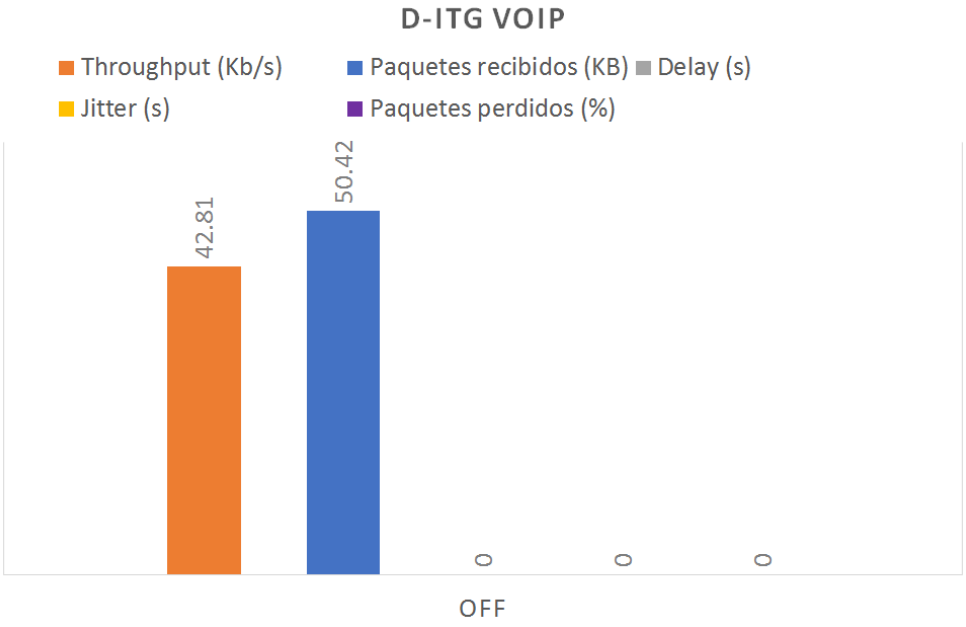


Figura 4.9: D-ITG con protocolo VoIP en escenario original.

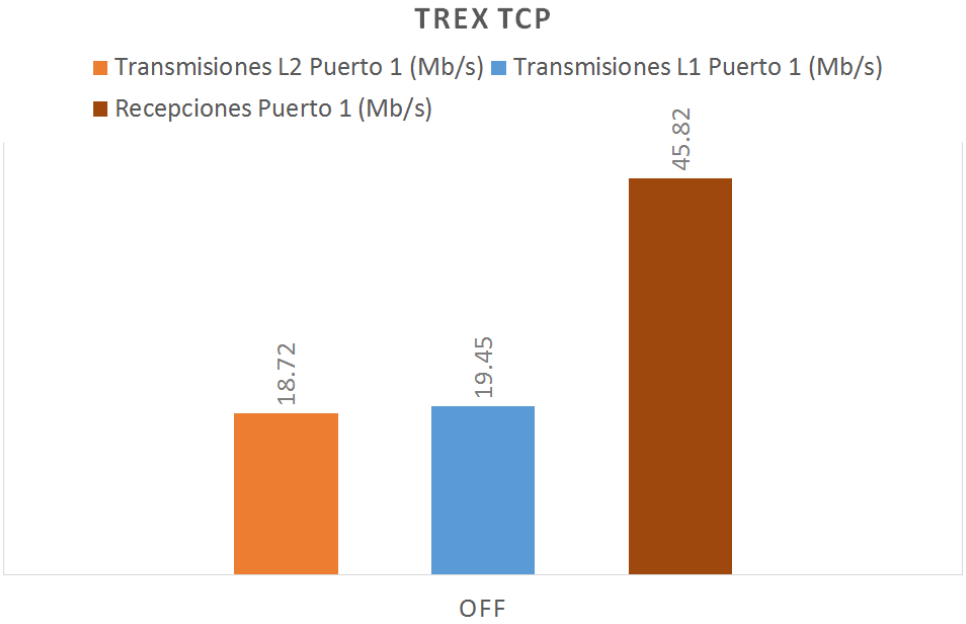


Figura 4.10: TREX con protocolo TCP en escenario original.

4.3 Limitación del ancho de banda en un enlace

1. IPerf: (TCP: Figura 4.12 y UDP: Figura 4.13)

   - TCP: los valores del *throughput* se mueven cercanos a los márgenes establecidos (1 Mb/s, 2 Mb/s, 5 Mb/s, 10 Mb/s), observamos que en dos casos más restrictivos el *throughput* está ligeramente por encima del nivel indicado.
   - UDP: los valores del *throughput* son significativamente más bajos que en caso homólogo con TCP, lo cual concuerda con las diferencias entre ambos protocolos (TCP y UDP). Vemos como la herramienta indica el valor de paquetes perdidos y observamos que el resultado de "Paquetes recibidos (MB)" y *"Throughput"* corresponden con la diferencia al total de la información enviada (la información enviada la veíamos en el caso inicial: Figura 4.4).

2. NetPerf: (TCP: Figura 4.14 y UDP: Figura 4.15)

   - TCP: vemos como el *throughput* se adapta al límite de ancho de banda en cada caso (1 Mb/s, 2 Mb/s, 5 Mb/s, 10 Mb/s) de forma correcta sin superar el límite. No disponemos de una medida del *delay* medio al emplear esta herramienta, registramos el caso máximo visto durante la ejecución de cada muestra y por ello podemos ver dicho valor tan elevado en algunos casos.
   - UDP: contemplamos que se genera una cantidad ínfima de tráfico el cual se pierde prácticamente en su totalidad en todos los casos, no resulta útil.

3. D-ITG: (TCP: Figura 4.16 y UDP: Figura 4.17)

   - TCP: vemos como en el caso más restrictivo (1Mb/s) la herramienta no da un valor adecuado, el *throughput* es una quinta parte del esperado para este caso, en los demás se mantiene más cerca del límite, podemos ver como apenas hay pérdida de paquetes y como desciende el *jitter* a medida que se baja la restricción.
   - UDP: vemos como en este caso el throughput tiene valores similares a su homologo con TCP excepto en el caso más restrictivo (1Mb/s), donde en este caso muestra un valor más adecuado. Podemos ver fácilmente la diferencia entre ambos protocolos atendiendo al porcentaje de paquetes de perdidos y a la relación entre paquetes Enviados (se envían más que en el mismo caso con TCP) y los Transmitidos con éxito al servidor, que son valores cercanos a los mostrados en TCP.
   - VoIP: en este caso no ponemos datos de VoIP ya que la herramienta no permite alterar las dimensiones del tráfico y la prueba genera demasiado poco tráfico cómo para que influya la modificación del enlace en este caso.

4. TREX: (TCP: Figura 4.18 y UDP: Figura 4.19)

   - TCP: Vemos que, aunque las recepciones sean superiores a lo que se esperaría

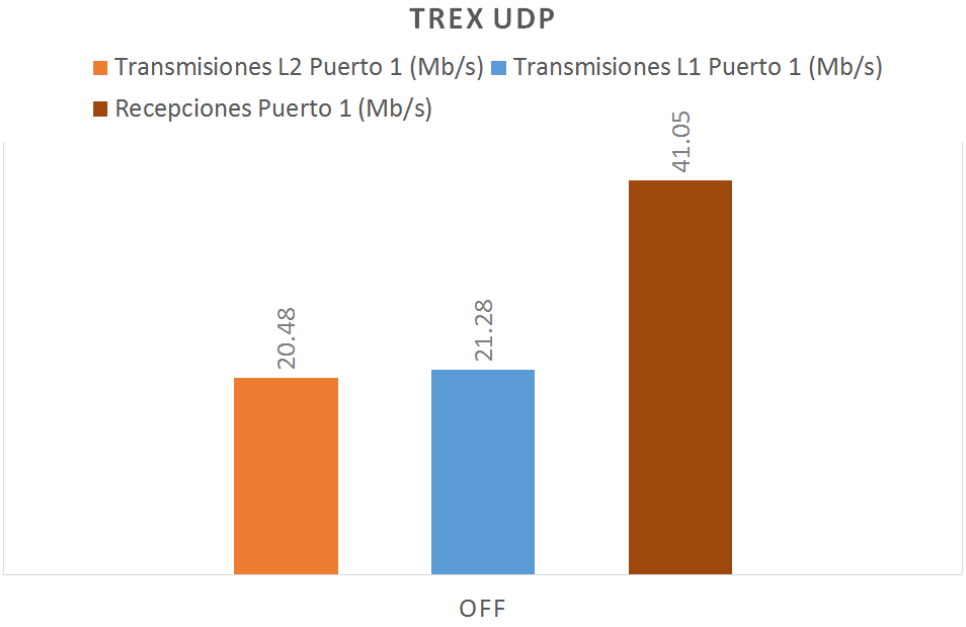


Figura 4.11: TREX con protocolo UDP en escenario original.

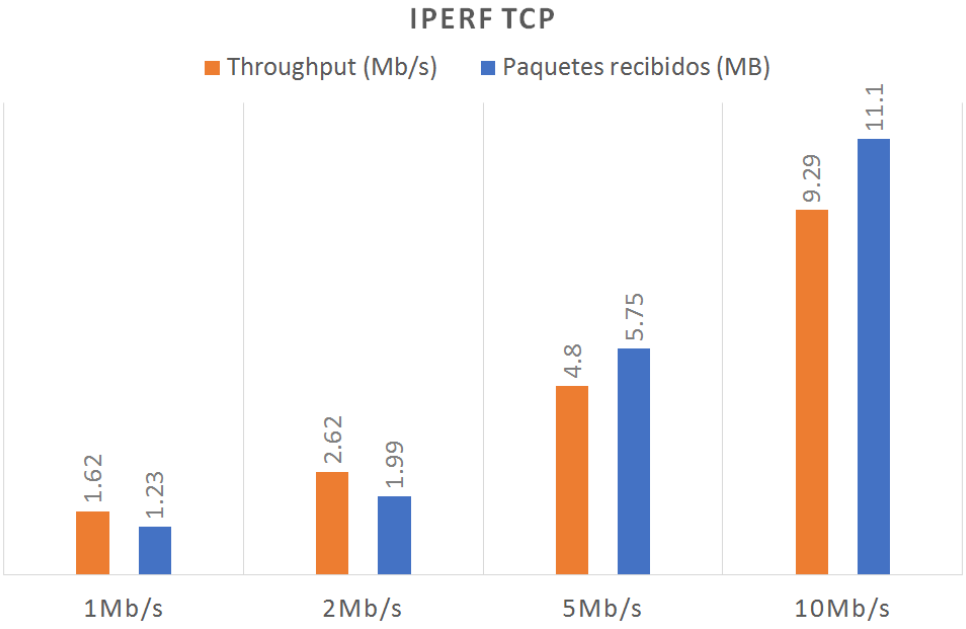


Figura 4.12: IPerf con protocolo TCP en escenario con ancho de banda modificado.

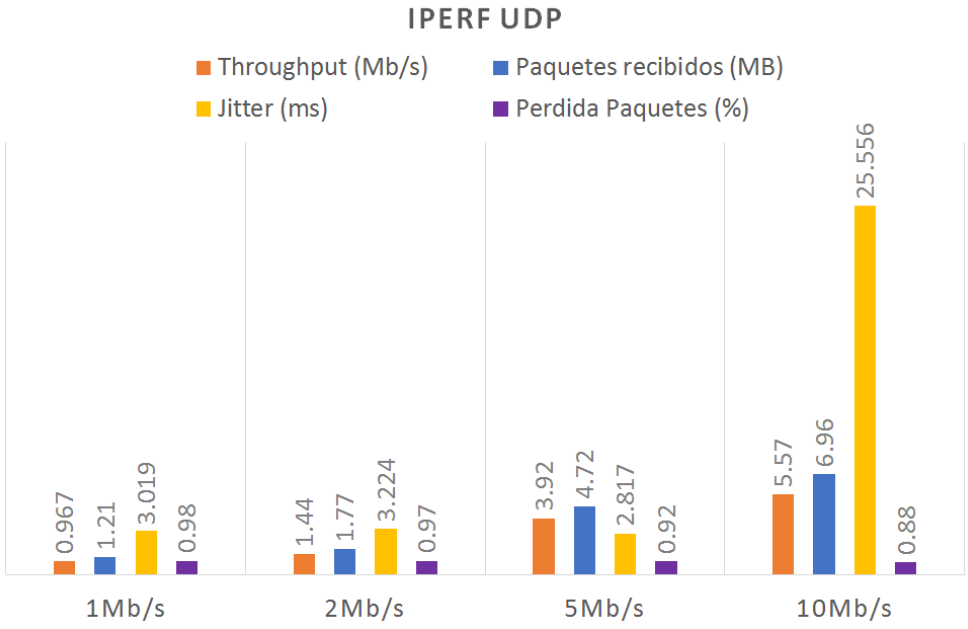


Figura 4.13: IPerf con protocolo UDP en escenario con ancho de banda modificado

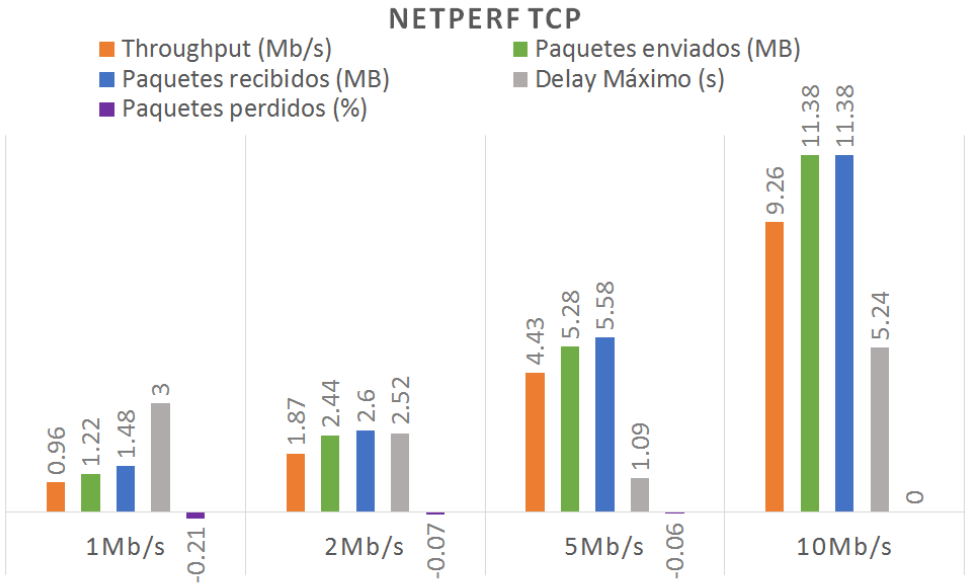


Figura 4.14: NetPerf con protocolo TCP en escenario con ancho de banda modificado

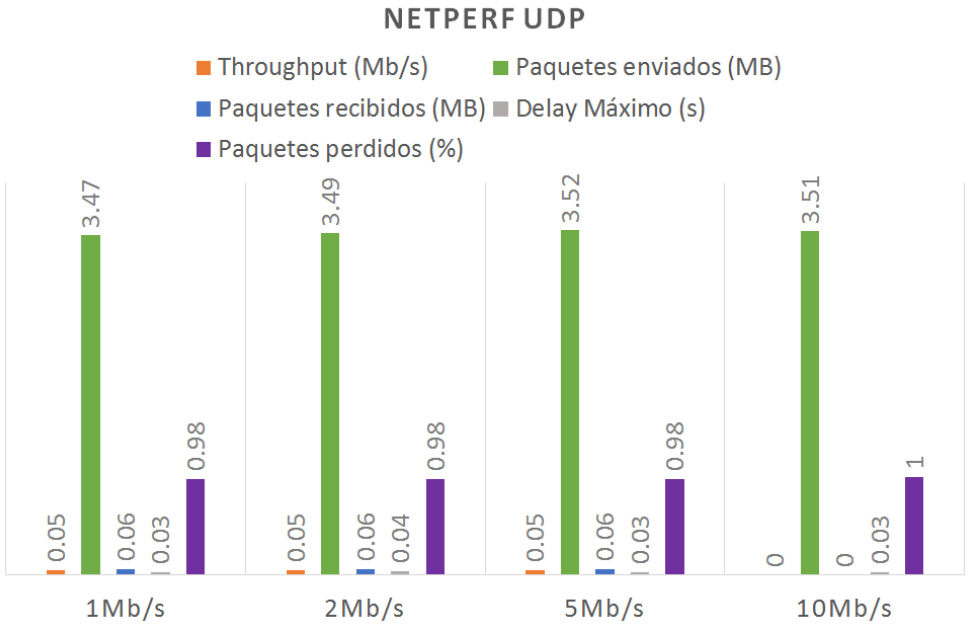


Figura 4.15: NetPerf con protocolo UDP en escenario con ancho de banda modificado

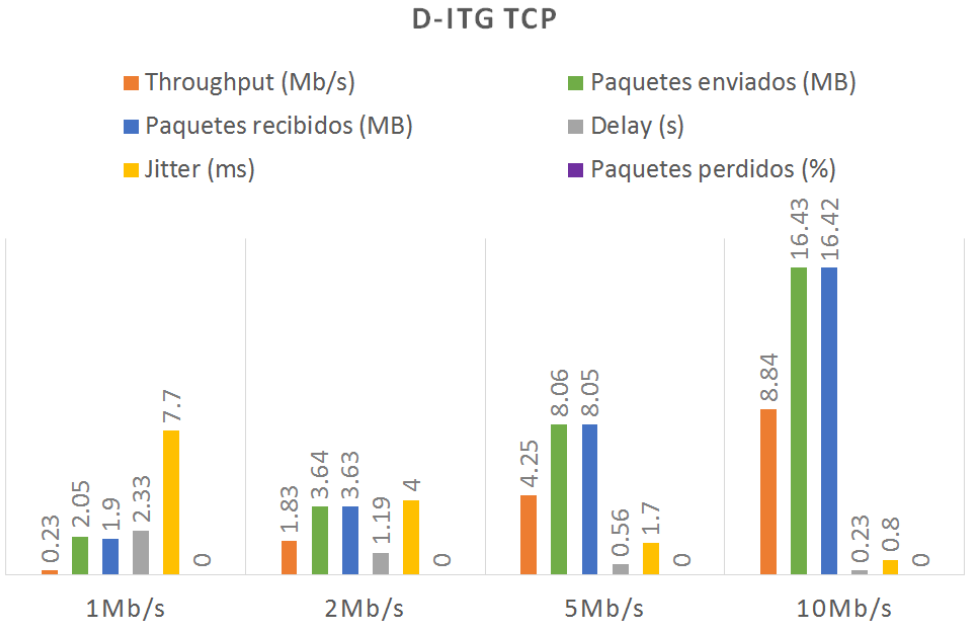


Figura 4.16: D-ITG con protocolo TCP en escenario con ancho de banda modificado

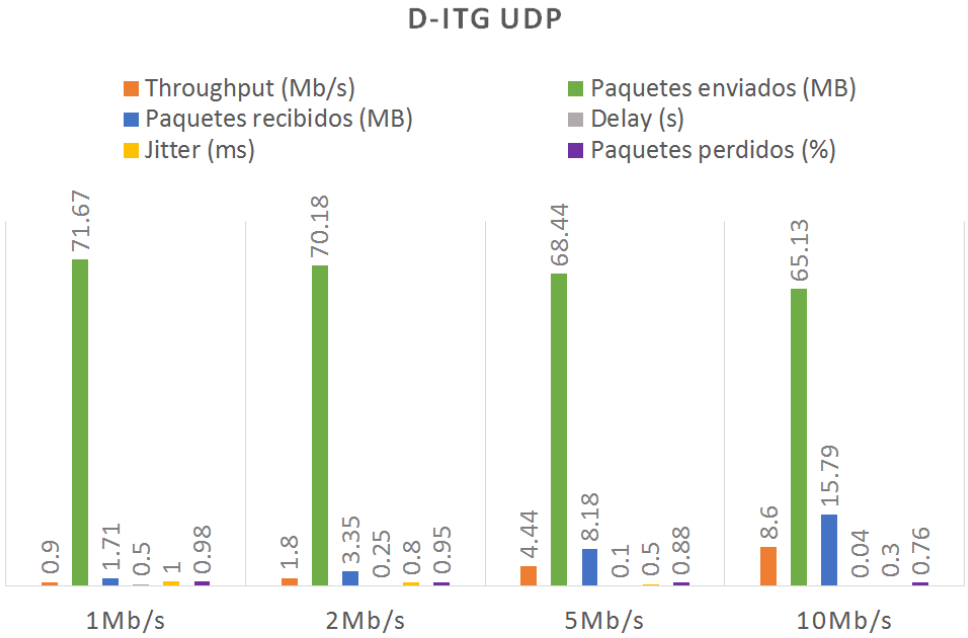


Figura 4.17: D-ITG con protocolo UDP en escenario con ancho de banda modificado

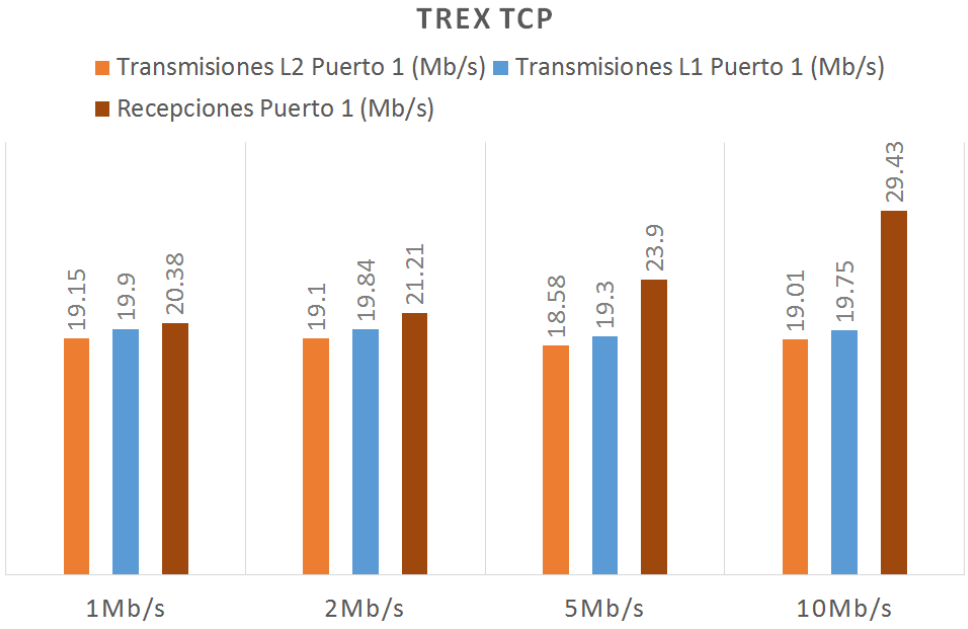


Figura 4.18: TREX con protocolo TCP en escenario con ancho de banda modificado

dadas las restricciones puestas, si restamos las recepciones a las ”Transmisiones L2” obtendremos valores semejantes a dichas restricciones.
- UDP: Obtenemos un resultado idéntico al caso homologo con TCP.

4.4 Aumento del *delay* de un enlace

1. IPerf: (TCP: Figura 4.20 y UDP: Figura 4.21)
   - TCP: vemos como a medida que se aumenta el *delay*, en cada caso llegan menos paquetes al ser pospuesto hasta lograr ser enviado, dado el funcionamiento de TCP.
   - UDP: en comparación con TCP, vemos como llega mayor cantidad d tráfico al servidor ya que, dado el funcionamiento de TCP, no se posponen o controlan los paquetes si no que se descartan en gran medida, podemos ver el aumento del porcentaje de paquetes perdidos en cada caso más restrictivo.
2. NetPerf: (TCP: Figura 4.22 y UDP: Figura 4.23)
   - TCP: vemos como se tiende a la misma curva de los niveles de *throughput* en cada caso, observamos como se reduce en la misma medida la cantidad de datos enviados y transmitidos y que, a pesar de esto, no se indican paquetes perdidos, ya que el descenso de flujo no pasa por descartar los paquetes. Funciona adecuadamente siguiendo lo esperado al protocolo TCP.
   - UDP: las pruebas con UDP no muestran valores significativos para analizar.
3. D-ITG: (TCP: Figura 4.24, UDP: Figura 4.25 y VoIP: Figura 4.26)
   - TCP: apreciamos como, siguiendo el esquema de las herramientas anteriores, el *throughput* y la cantidad de paquetes, tanto enviados como transmitidos, se reducen conforme se aumenta la restricción en el enlace. También apreciamos una pérdida de paquetes muy pequeña, señal de que se han producido algunos descartes durante la prueba, pero no demasiados en proporción.
   - UDP: observamos como el descenso del envío de información en el flujo de datos vuelve a mantenerse en un alto número de ”Paquetes Enviados” con una bajada significativa de los enviados con éxito al servidor, dicha relación mostrada en el porcentaje de paquetes perdidos. Observemos a mayores que en este caso el *delay* que calcula la herramienta corresponde perfectamente en cada caso con el impuesto al enlace durante la realización de las pruebas, cosa que hasta ahora no se dio.
   - VoIP: dado que únicamente se envía una trama (tal y como ha permitido la herramienta), no ha habido diferencias significativas en la prueba.

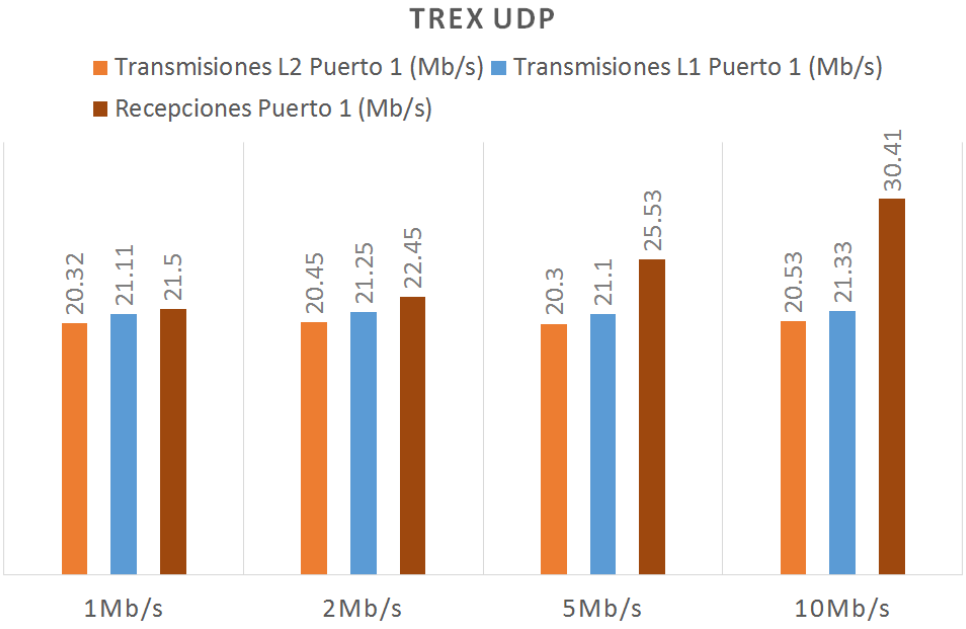


Figura 4.19: TREX con protocolo UDP en escenario con ancho de banda modificado

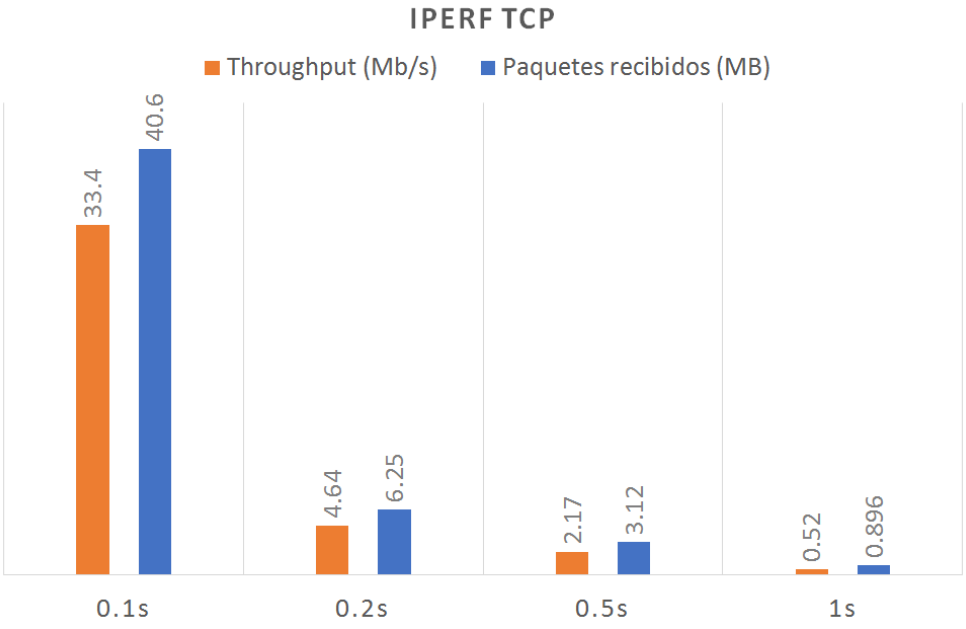


Figura 4.20: IPerf con protocolo TCP en escenario con *delay* modificado

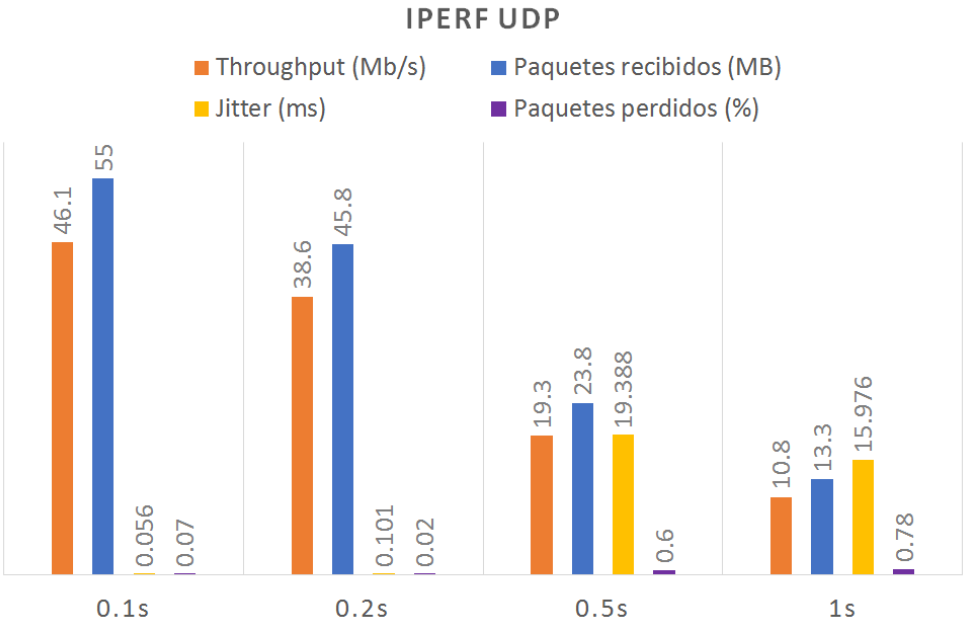


Figura 4.21: IPerf con protocolo UDP en escenario con *delay* modificado

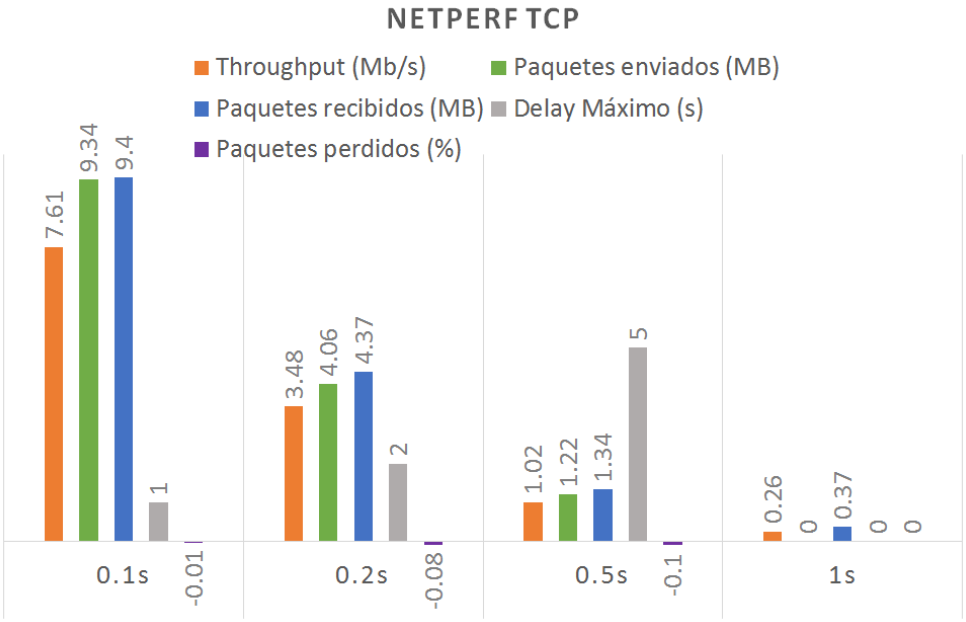


Figura 4.22: NetPerf con protocolo TCP en escenario con *delay* modificado

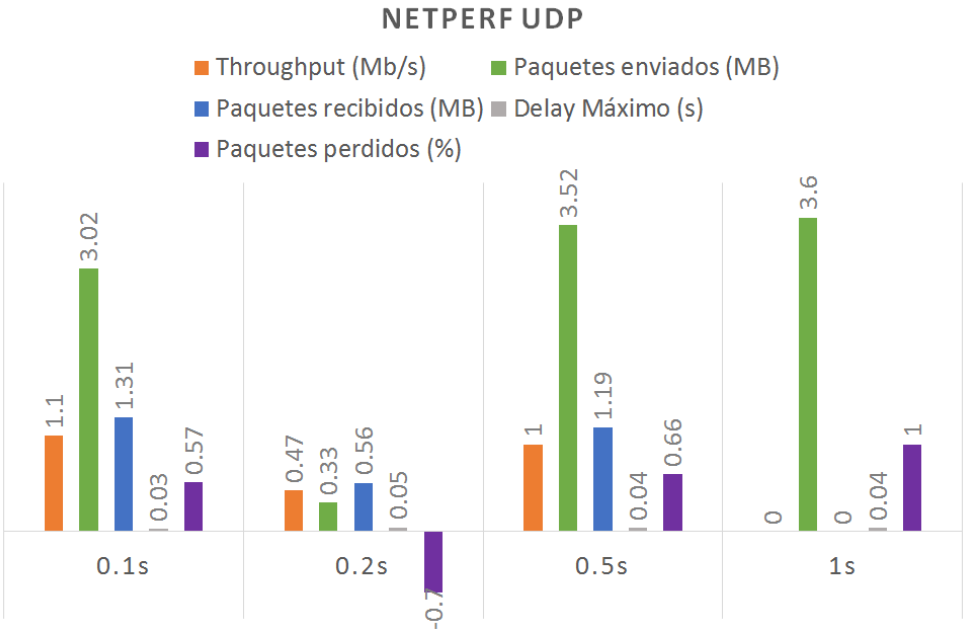


Figura 4.23: NetPerf con protocolo UDP en escenario con *delay* modificado

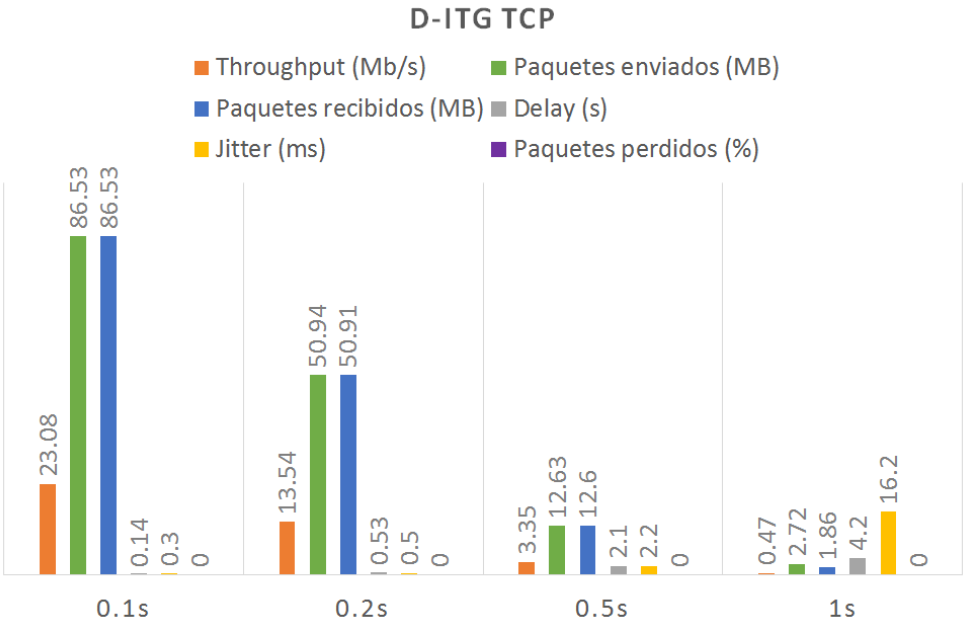


Figura 4.24: D-ITG con protocolo TCP en escenario con *delay* modificado

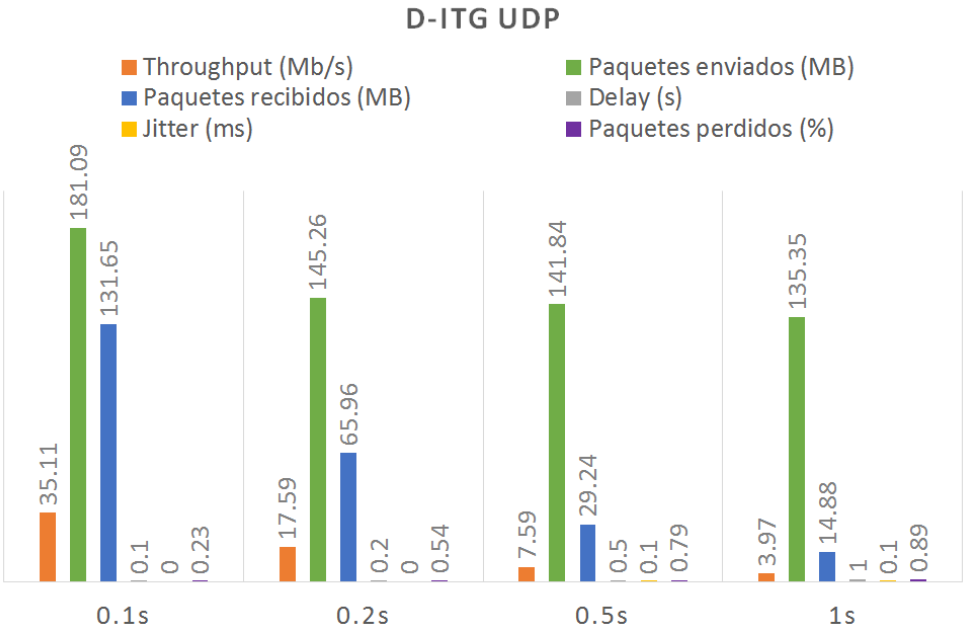


Figura 4.25: D-ITG con protocolo UDP en escenario con *delay* modificado

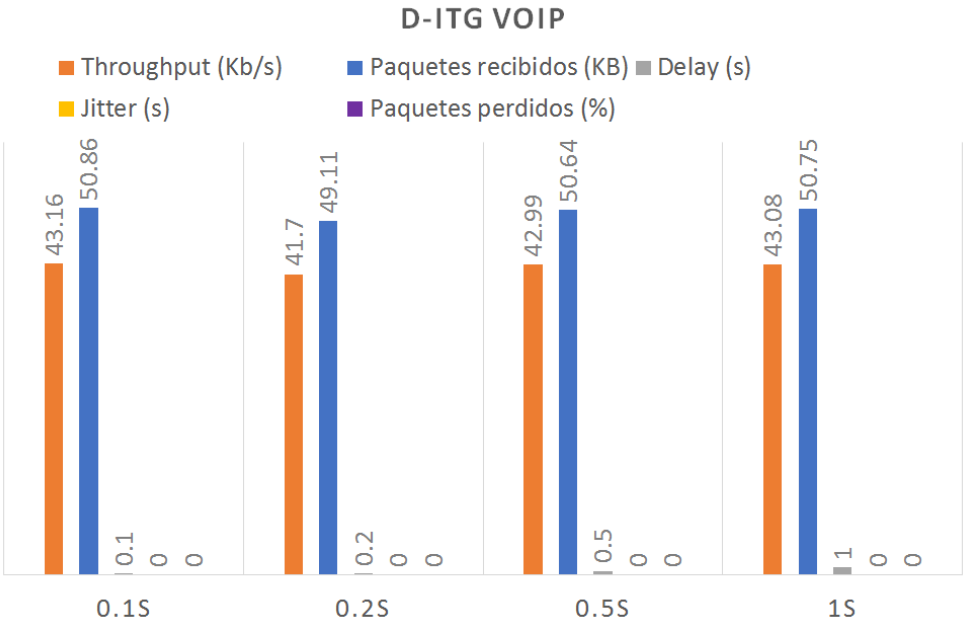


Figura 4.26: D-ITG con protocolo VoIP en escenario con *delay* modificado

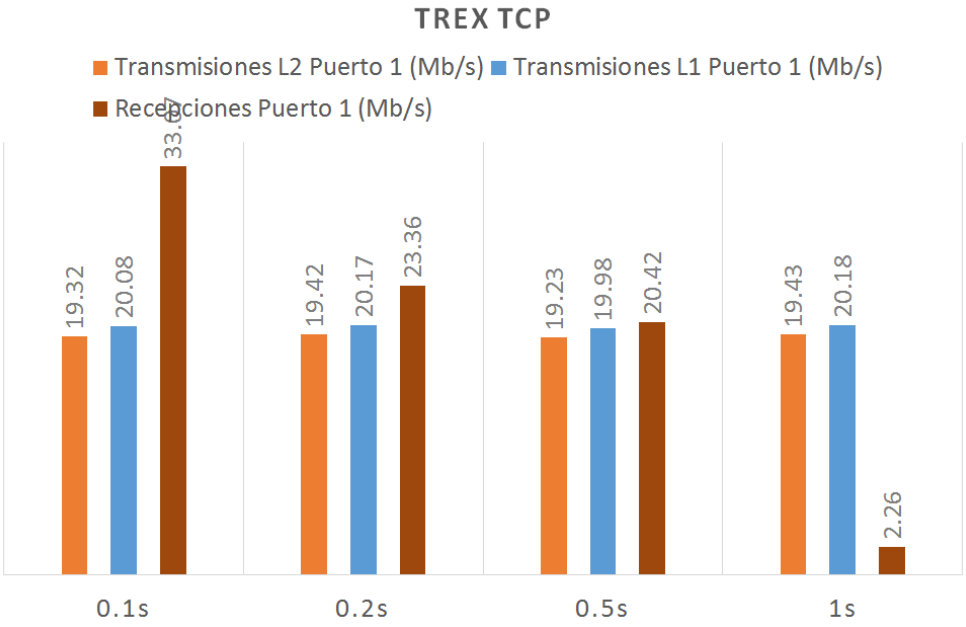


Figura 4.27: TREX con protocolo TCP en escenario con *delay* modificado

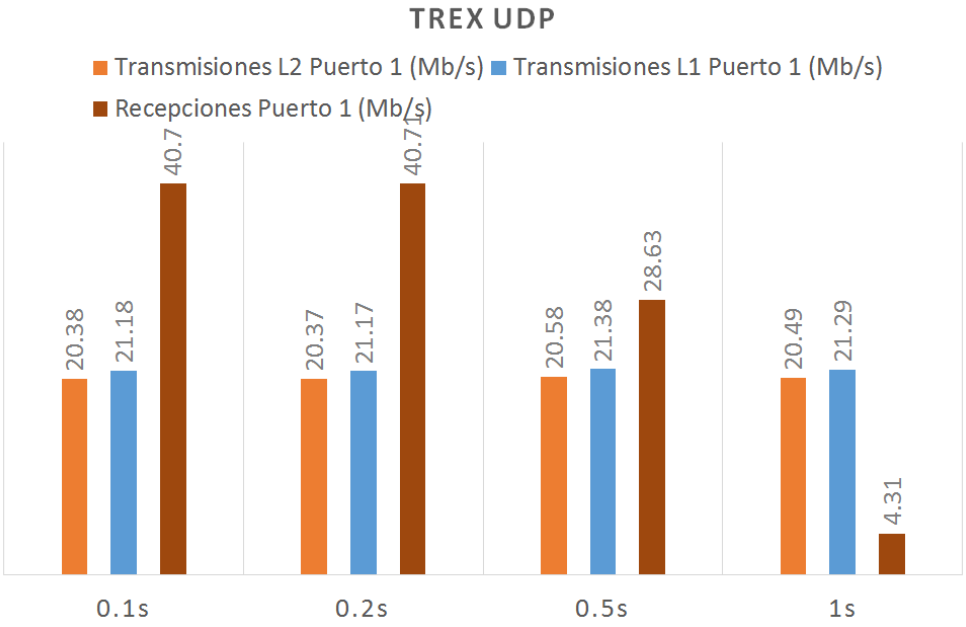


Figura 4.28: TREX con protocolo UDP en escenario con *delay* modificado

4. TREX: (TCP: Figura 4.27 y UDP: Figura 4.28)

   - TCP: Observamos una bajada en las recepciones conforme aumenta el tiempo de *delay* en cada prueba (0.1s, 0.2s, 0.5, 1s). Para el último caso casi no se reciben paquetes.
   - UDP: Aunque a primera vista pudiera parecer que el resultado es igual al de TCP, podemos ver que, primero, los valores de las recepciones son mayores en todos los casos y, segundo, casi no nota diferencia en los primeros casos. Dicho comportamiento coincide lo esperado teniendo en cuenta las diferencias entre TCP y UDP.

4.5 Aumento del *jitter* de un enlace

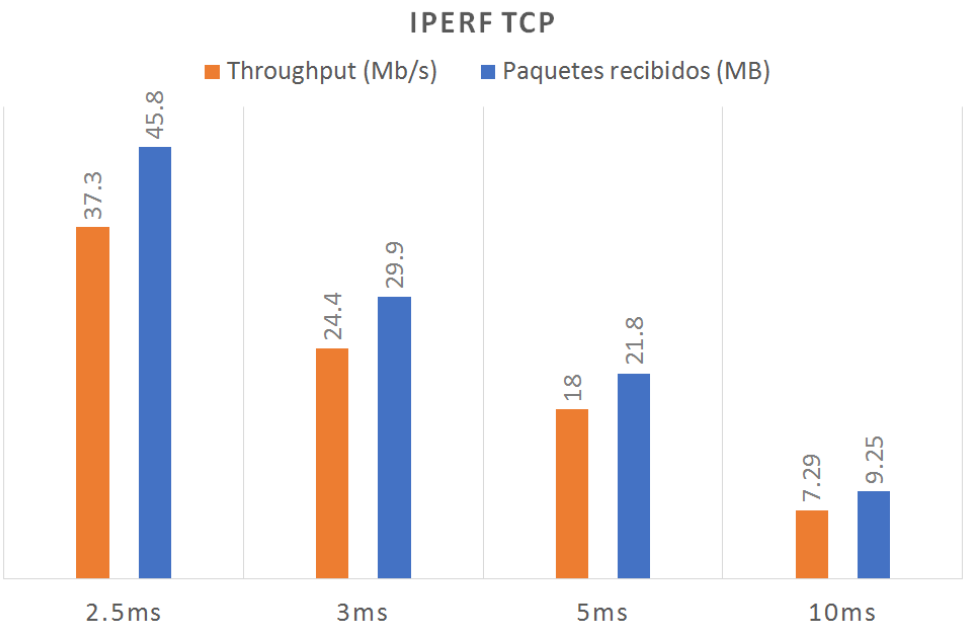


Figura 4.29: IPerf con protocolo TCP en escenario con *jitter* modificado.

1. IPerf: (TCP: Figura 4.29 y UDP: Figura 4.30)

   - TCP: podemos ver un descenso relativamente suave conforme se agrava restricción en el enlace. El retardo que puede alcanzar provoca pérdidas ya hay paquetes que pueden llegar más tarde de lo que permite el protocolo y dando lugar a su pérdida. Nota: hemos escogido las medidas para estas pruebas observando el punto donde empezaba a verse los efectos del *jitter* en IPerf y que ha resultado ser en aquella que se veía menos pronunciado el efecto de esta.

- UDP: podemos ver que no se presentan variaciones en ninguna de las pruebas, salvo en el aumento del *jitter* calculado por la herramienta, esto se debe al escaso control que tiene UDP sobre los paquetes del flujo.

2. NetPerf: (TCP: Figura 4.31 y UDP: Figura 4.32)
   - TCP: vemos un descenso no demasiado pronunciado, pero apreciamos que ya en el primer caso se ha visto muy reducido el flujo de datos respecto al caso inicial, señal de que netperf empieza a verse afectado a la hora de hacer pruebas con un *jitter* menor al empleado inicialmente.
   - UDP: no muestra información relevante para analizar.
3. D-ITG: (TCP: Figura 4.33, UDP: Figura 4.34 y VoIP: Figura 4.35)
   - TCP: vemos un descenso muy suave, como explicamos con NetPerf, D-ITG empieza a verse afectado antes del margen empleado. Podemos apreciar como en el caso de 3ms el flujo fue ligeramente mayor que su predecesor de 2.5ms, señal de que en esta prueba no hubo tantos paquetes con retardos demasiado grandes.
   - UDP: como en las herramientas anteriores, no se aprecian diferencias significativas para este caso, vemos como en hay casos con mayor restricción que el tráfico se ha transmitido de mejor forma.
   - VoIP: Otra vez, dado que solo envía una trama, no se ven diferencias significativas.
4. TREX: (TCP: Figura 4.36 y UDP: Figura 4.37)
   - TCP: No vemos que le afecte prácticamente en nada el aumento de *jitter* a las pruebas con TRex. (en este caso hasta pareciera que aumenta el flujo de datos conforme aumenta el *jitter*, aunque no se debe a eso).
   - UDP: Los valores obtenidos prácticamente no sufren ninguna diferencia en ninguna de las pruebas.

4.6 Aumento de la pérdida de paquetes de un enlace

1. IPerf: (TCP: Figura 4.38 y UDP: Figura 4.39)
   - TCP: vemos un descenso enorme en cada caso respecto del anterior, la pérdida de paquetes dificulta la continuidad de la comunicación, lo que deriva en mayores pérdidas, por lo que tenemos un descenso exponencial del tráfico en el flujo de datos.
   - UDP: si lo comparamos con TCP, vemos que el descenso es extremadamente ligero en comparación, y podemos apreciar que, de hecho, el descenso es exacto al porcentaje de pérdida indicado.
2. NetPerf: (TCP: Figura 4.40 y UDP: Figura 4.41)

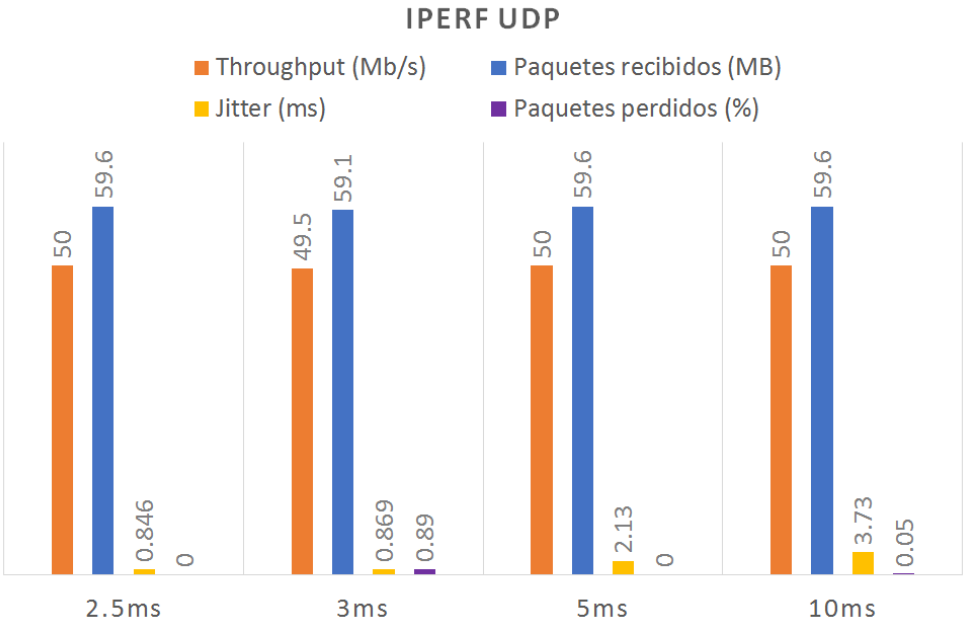


Figura 4.30: IPerf con protocolo UDP en escenario con *jitter* modificado.

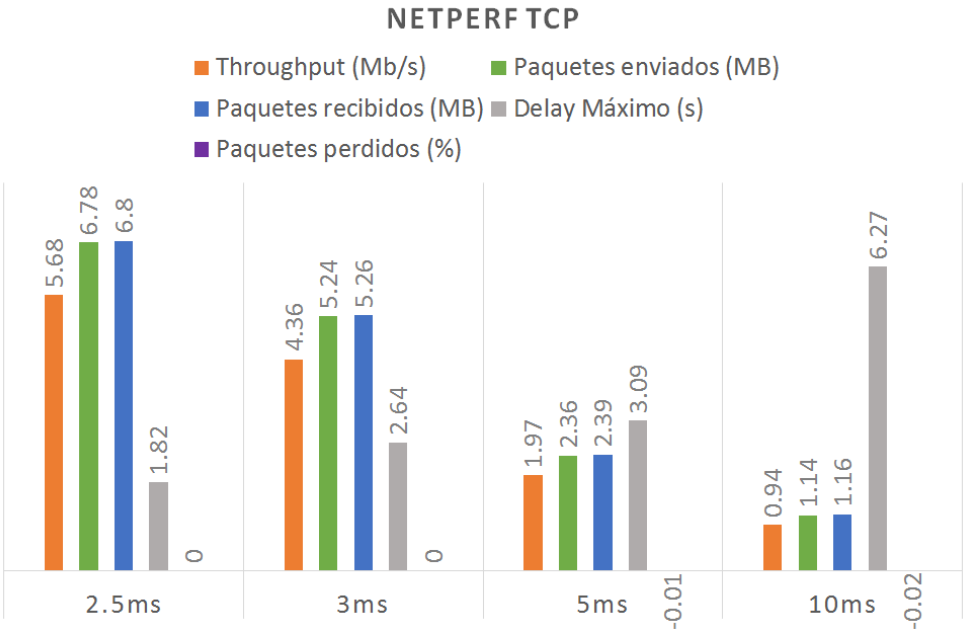


Figura 4.31: NetPerf con protocolo TCP en escenario con *jitter* modificado.

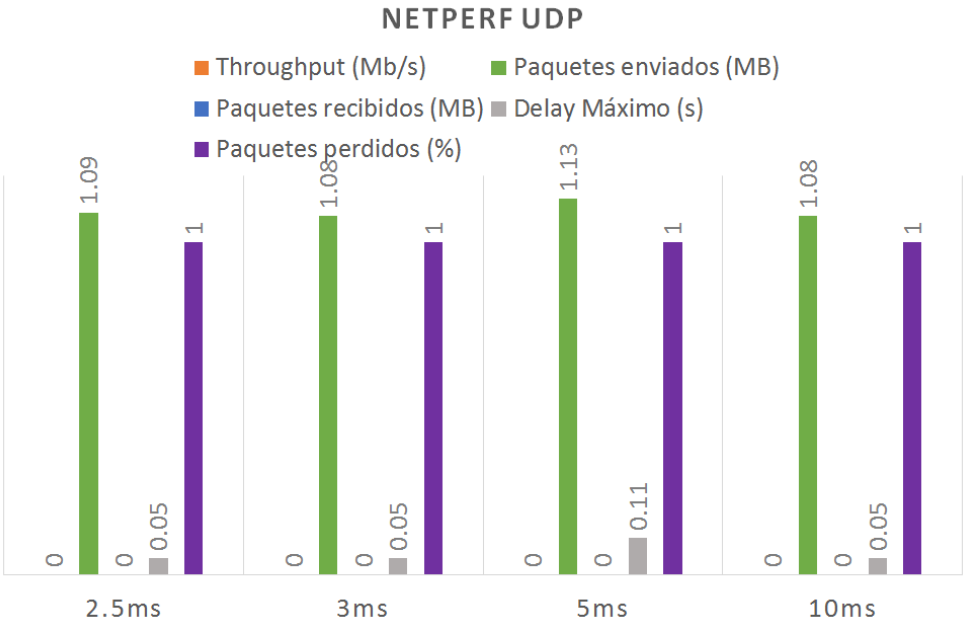


Figura 4.32: NetPerf con protocolo UDP en escenario con *jitter* modificado.

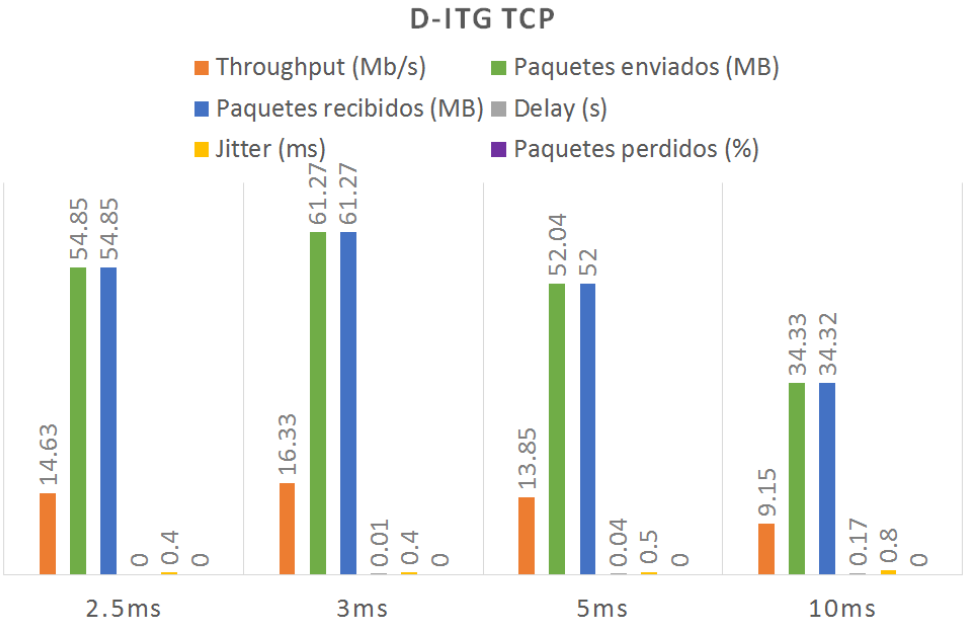


Figura 4.33: D-ITG con protocolo TCP en escenario con *jitter* modificado.

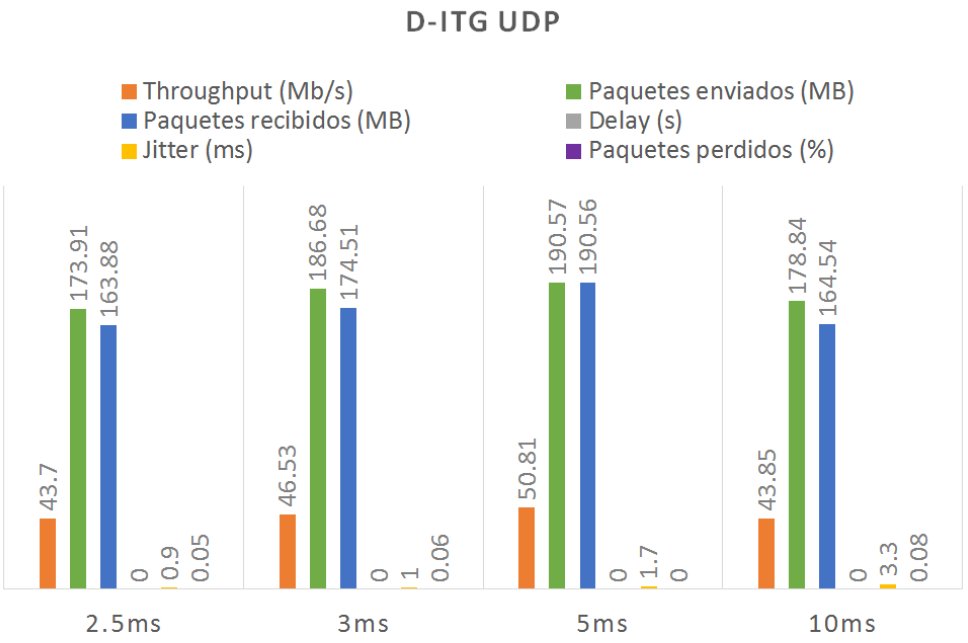


Figura 4.34: D-ITG con protocolo UDP en escenario con *jitter* modificado.

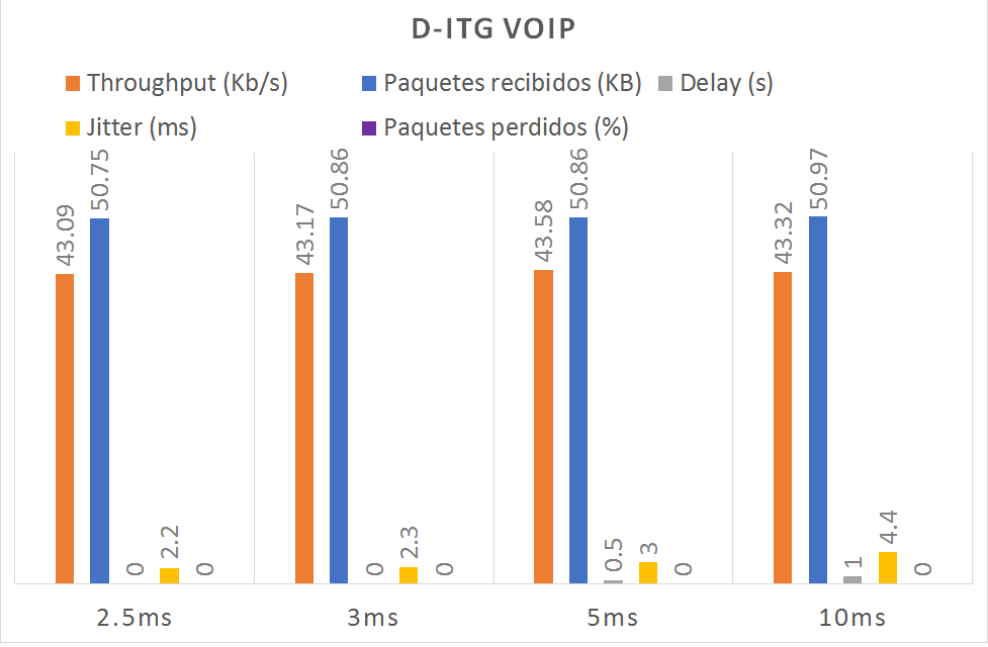


Figura 4.35: D-ITG con protocolo VoIP en escenario con *jitter* modificado.

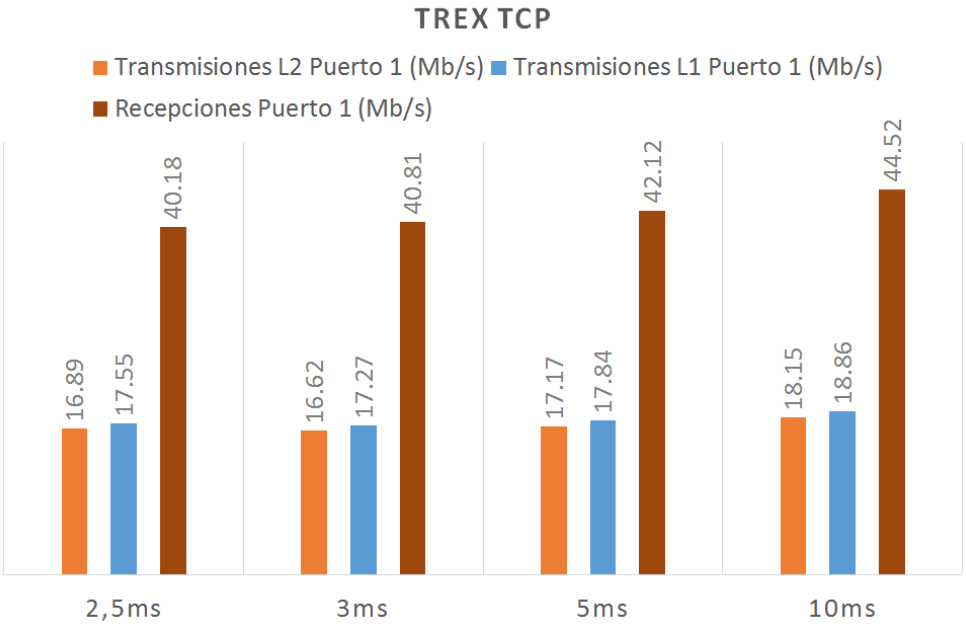


Figura 4.36: TREX con protocolo TCP en escenario con *jitter* modificado.

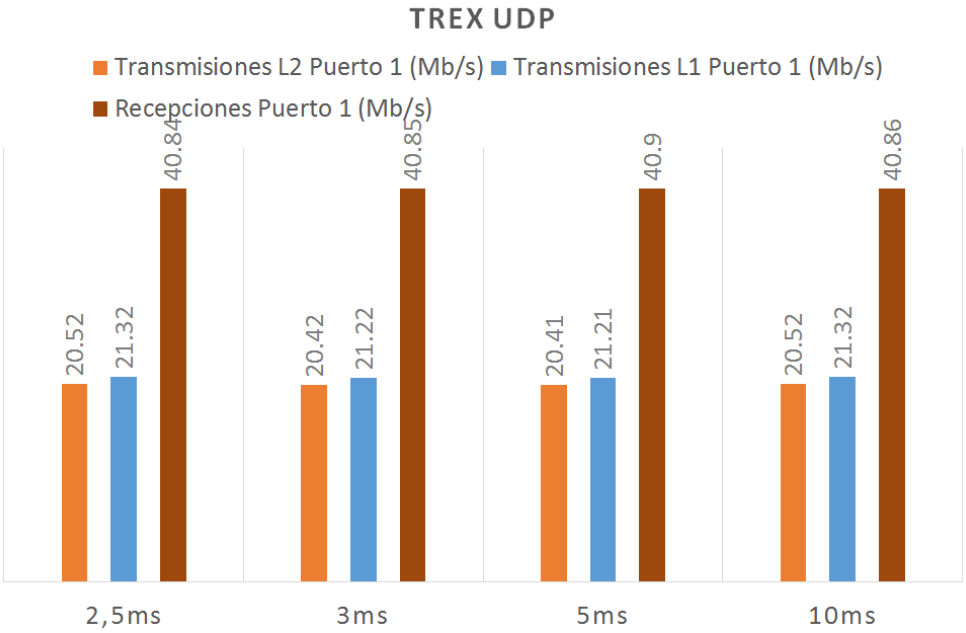


Figura 4.37: TREX con protocolo UDP en escenario con *jitter* modificado.

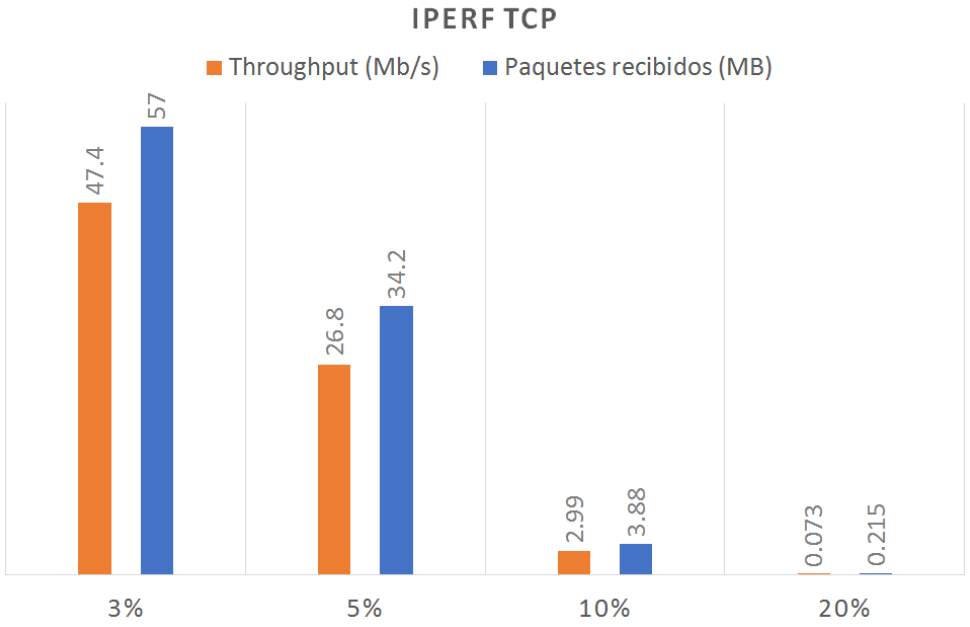


Figura 4.38: IPerf con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado.

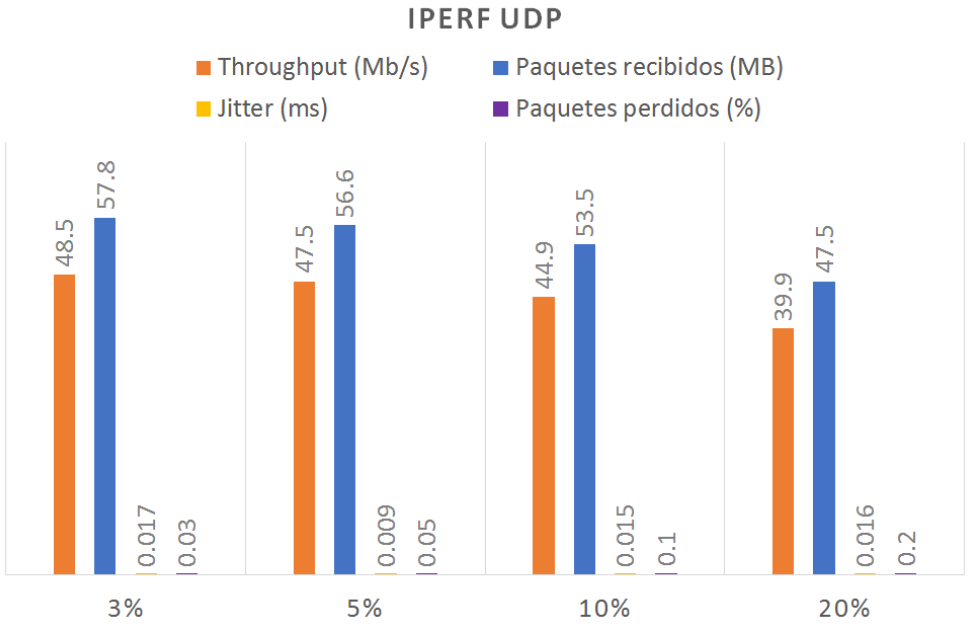


Figura 4.39: IPerf con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado.

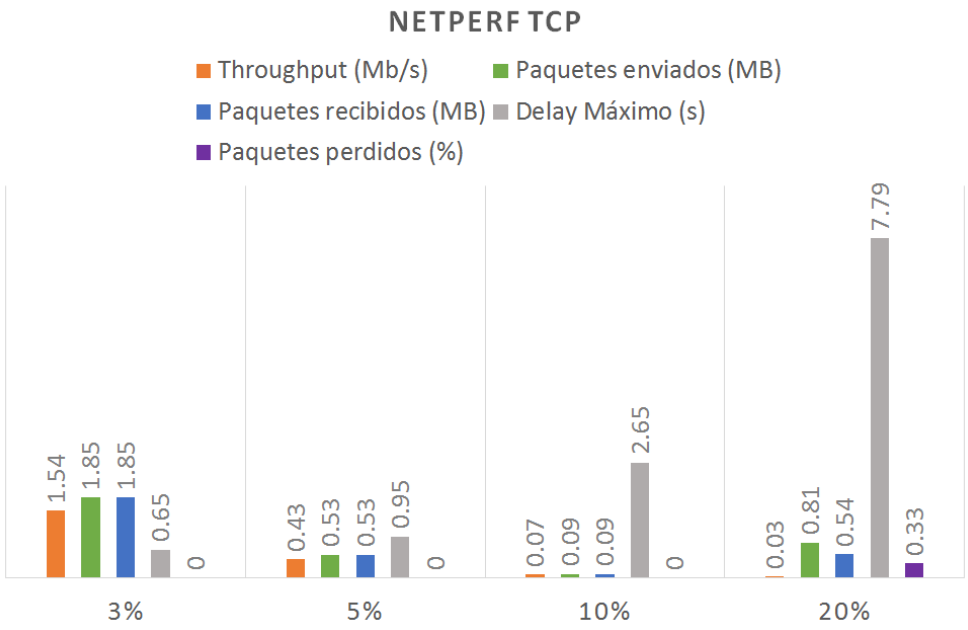


Figura 4.40: NetPerf con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado.

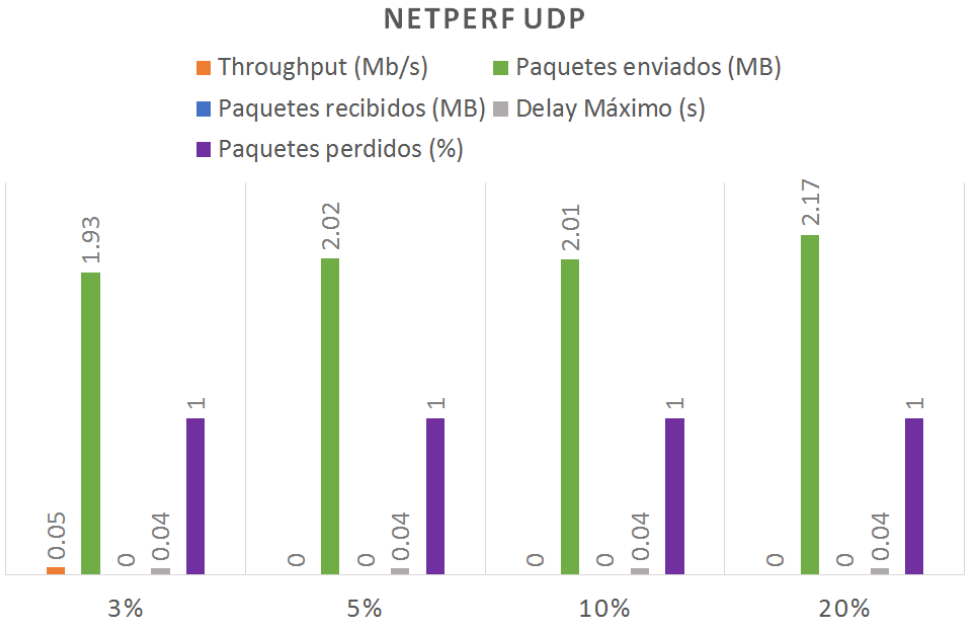


Figura 4.41: NetPerf con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado.

- TCP: NetPerf presenta una sensibilidad extrema ante esta situación, aunque se aprecia el descenso al final de la prueba llega a impedir casi el total del envío de datos.
- UDP: otra vez, NetPerf con UDP no muestra datos significativos.

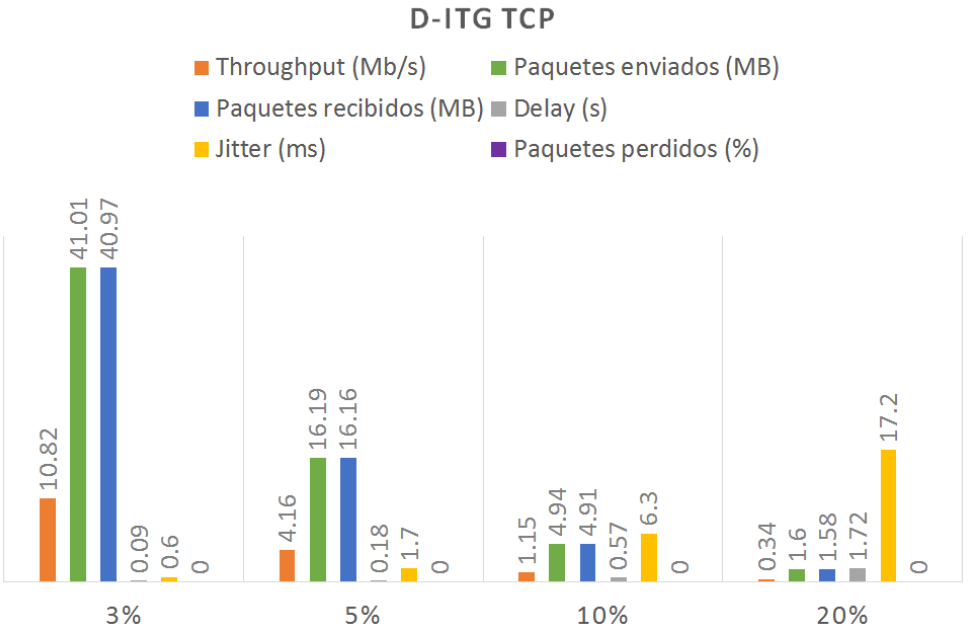


Figura 4.42: D-ITG con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado.

3. D-ITG: (TCP: Figura 4.36, UDP: Figura 4.43 y VoIP: Figura 4.44)
   - TCP: D-ITG resulta presentar una alta sensibilidad a la prueba. Podemos apreciar que ya con solo el primer caso se reduce de forma considerable el tráfico del flujo. Vemos como se mantiene la relación entre los paquetes enviados y transmitidos a pesar de la bajada que refleja.
   - UDP: con UDP se nos presenta un caso relativamente extraño, no se ve una curva como en el caso de IPerf, sino que los valores se ven Paquetes Enviados y Transmitidos varían de forma irregular en cada apartado, aunque mantienen la relación de paquetes perdidos indicado en el enlace modificado.
   - VoIP: se aprecia cómo se reducen los paquetes en cada caso de forma aproximada a lo establecido (el error se debe al pequeño número de paquetes que dispone la prueba).
4. TREX: (TCP: Figura 4.45 y UDP: Figura 4.46)
   - TCP: Vemos un descenso ligero de las recepciones conforme aumenta el porcentaje de paquetes perdidos. Al igual que el caso con el ancho de banda, podemos

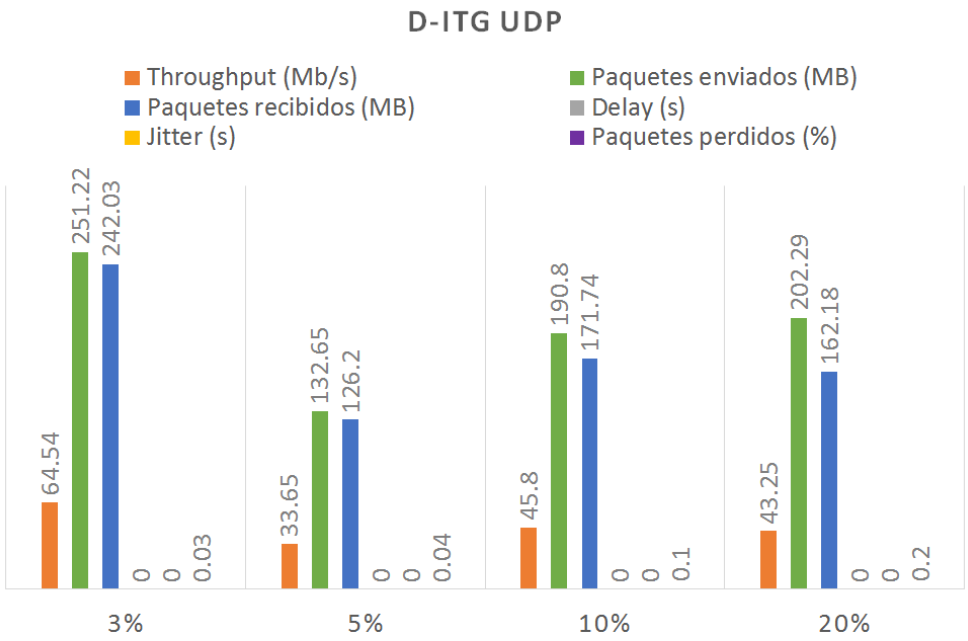


Figura 4.43: D-ITG con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado.

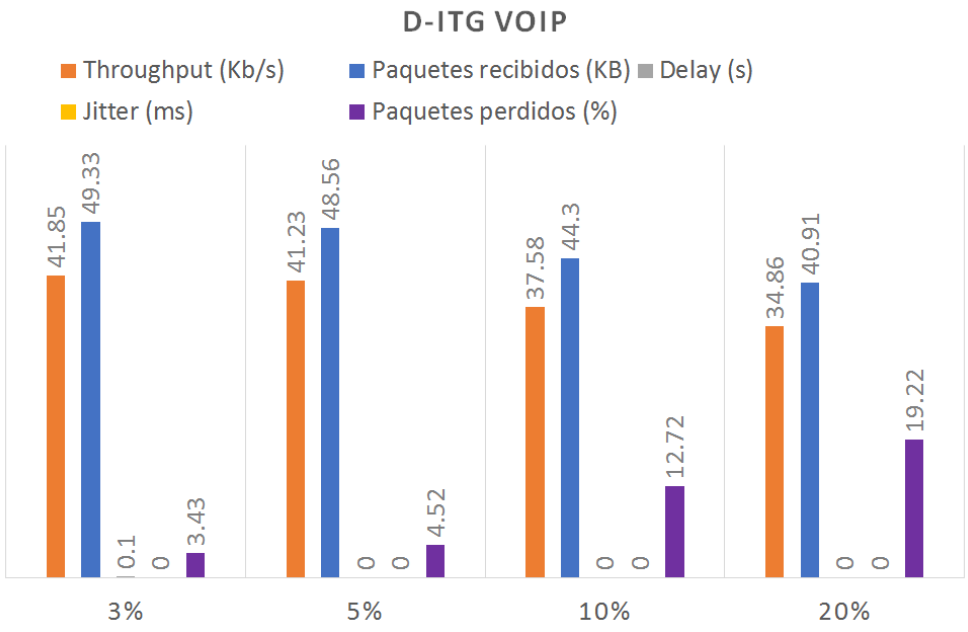


Figura 4.44: D-ITG con protocolo VoIP en escenario con porcentaje de pérdida de paquetes modificado.

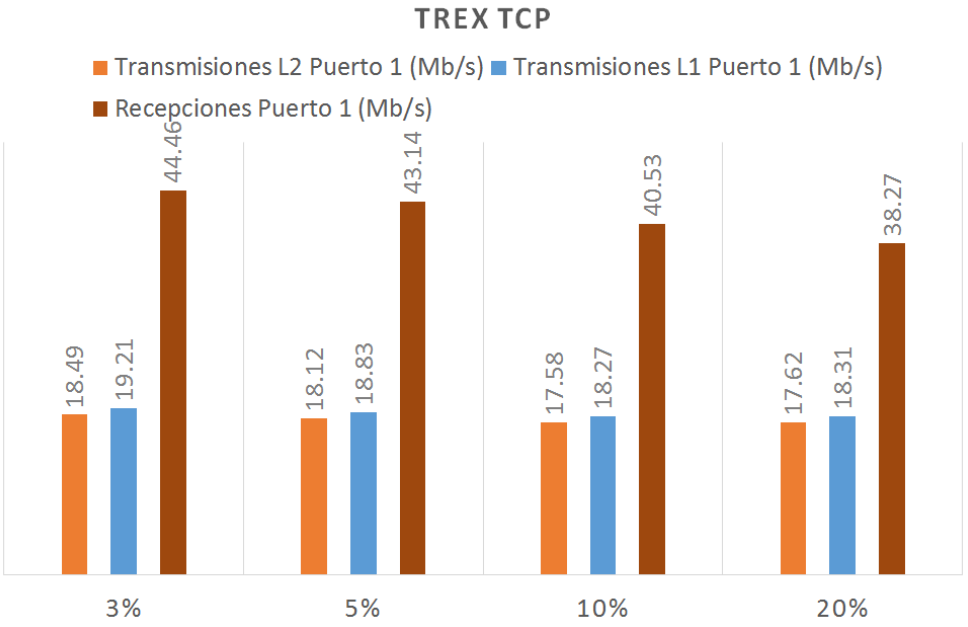


Figura 4.45: TREX con protocolo TCP en escenario con porcentaje de pérdida de paquetes modificado.

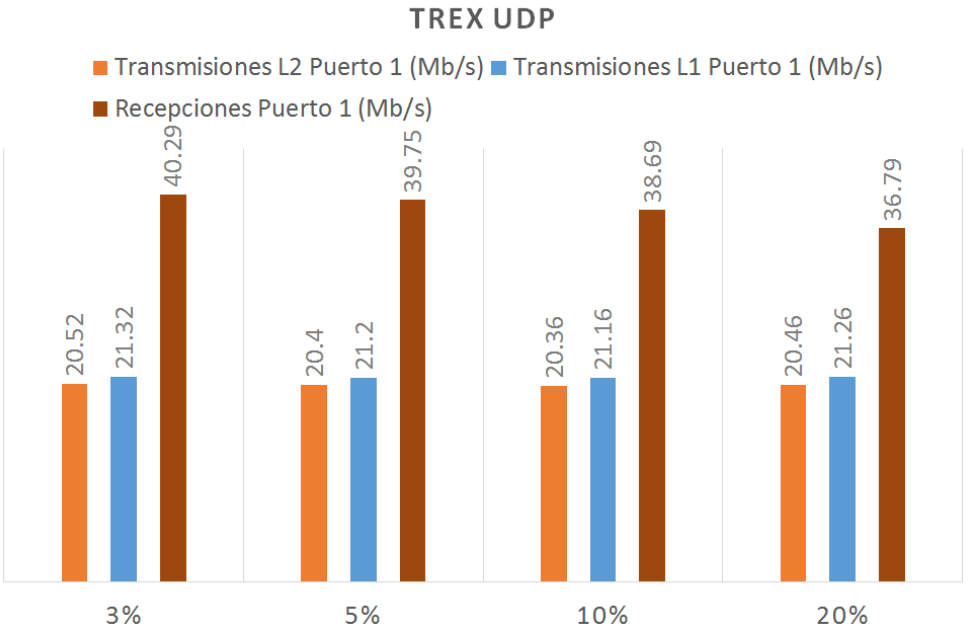


Figura 4.46: TREX con protocolo UDP en escenario con porcentaje de pérdida de paquetes modificado.

ver que las diferencias se corresponden mejor con los porcentajes (3%, 5%, 10%, 20%), emplear las diferencias entre las recepciones y las transmisiones L2, y estas compararlas contra el valor de recepciones del caso inicial (45.82 Mb/s).
- UDP: El comportamiento resulta gemelo a los casos con UDP, siendo la única diferencia el que las recepciones iniciales con UDP eran de 41.05Mb/s.

Nota: durante las primeras pruebas, previas a las reflejadas en las gráficas anteriores, se probó el comportamiento de las herramientas al 50% de pérdidas de paquetes, dando un caso extraño donde la recepción de tráfico se disparaba en vez de reducirse como debería pasar y si ocurre en los resultados mostrados.

4.7 Aumento de la duplicidad de un enlace

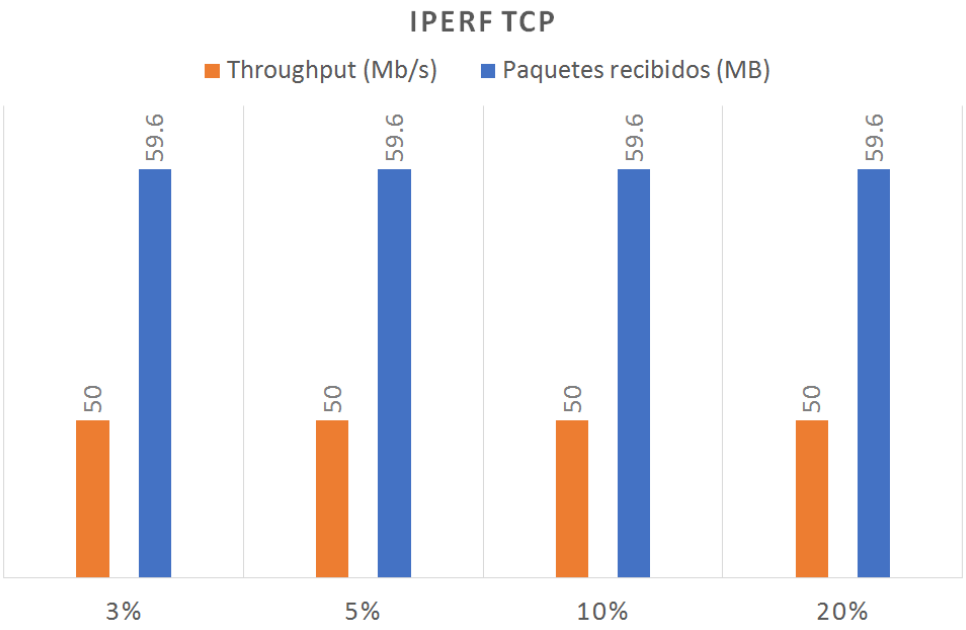


Figura 4.47: IPerf con protocolo TCP en escenario con porcentaje de duplicidad modificado.

1. IPerf: (TCP: Figura 4.47 y UDP: Figura 4.48)
   - TCP: no se muestran diferencias ante la duplicidad, tal como debería ocurrir con el protocolo TCP.
   - UDP: vemos un aumento de los Paquetes recibidos (MB) en proporción al porcentaje de duplicidad añadido.
2. NetPerf: (TCP: Figura 4.49 y UDP: Figura 4.50)
   - TCP: vemos un descenso del flujo de datos según subimos el porcentaje de duplicidad, a pesar de que no presenta pérdida de paquetes podemos ver como desciende

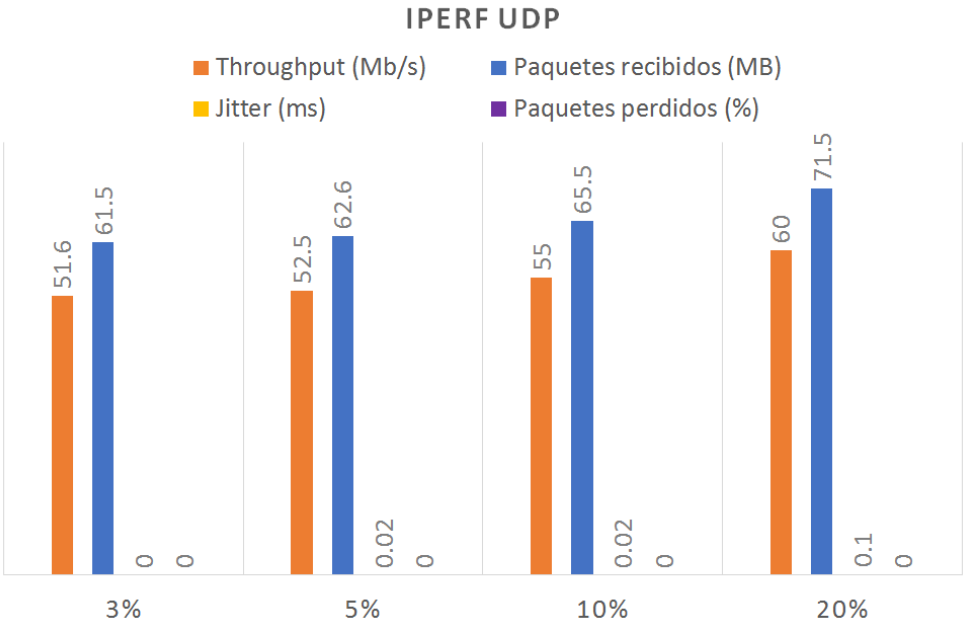


Figura 4.48: IPerf con protocolo UDP en escenario con porcentaje de duplicidad modificado.

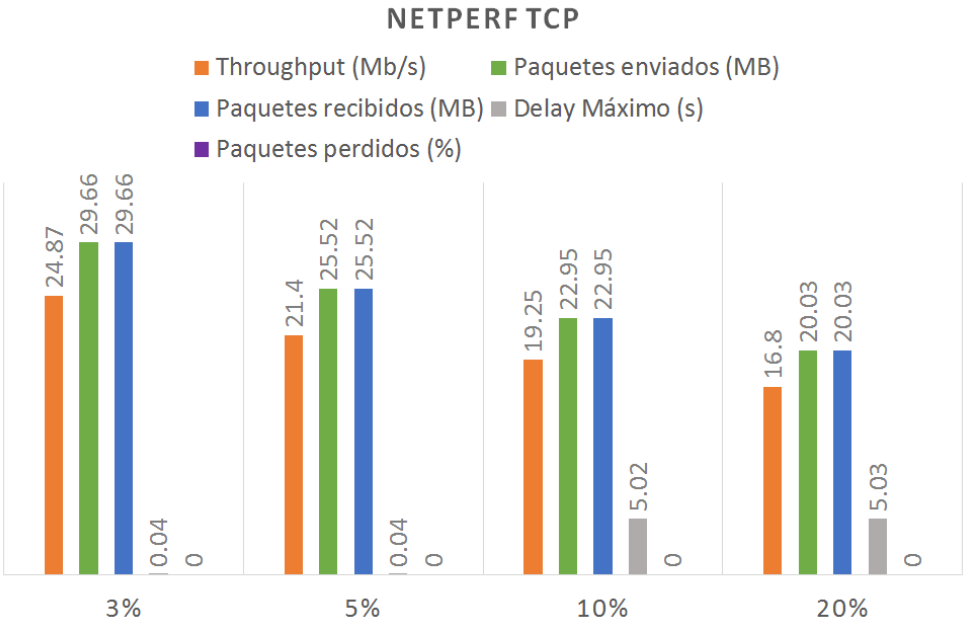


Figura 4.49: NetPerf con protocolo TCP en escenario con porcentaje de duplicidad modificado.

directamente la información a enviar. Concluimos que se debe a la dificultad del control de flujo de TCP empleando esta herramienta.
- UDP: podemos ver que el descenso del tráfico en el flujo según aumentamos el índice de duplicidad, a pesar del reducido tamaño del tráfico.

3. D-ITG: (TCP: Figura 4.51, UDP: Figura 4.52 y VoIP: Figura 4.53)
   - TCP: vemos un aumento significativo conforme aumentamos el porcentaje de duplicidad, dichos aumentos no corresponden con el porcentaje fijado.
   - UDP: las pruebas en este caso no presentan una relación clara, tenemos un aumento claro en el flujo de datos en los primeros incrementos del índice de duplicidad con un descenso en el caso final, dichos aumentos no corresponden de ninguna manera con dicho índice de duplicidad.
   - VoIP: aumenta el número de paquetes en relación con el porcentaje de duplicidad indicado.
4. TREX: (TCP: Figura 4.54 y UDP: Figura 4.55)
   - TCP: Podemos ver un comportamiento anormal en este caso, aunque disminuyan las recepciones con los tres primeros porcentajes (3%,5%,10%), vemos un aumento muy grande en el cuarto (20%), el aumento del cuarto caso corresponde con su porcentaje teniendo en cuenta a la resta comentada con el ancho de banda y las pérdidas. Concluimos que TRex resiste la duplicidad hasta un umbral donde si le comienza a afectar.
   - UDP: Podemos ver que los valores de recepción aumentan conforme lo hace el porcentaje de duplicidad, pero se tratan de aumentos demasiado pequeños, no creemos que sea debido a la influencia de la duplicidad.

4.8 Conclusiones de las pruebas anteriores

1. IPerf: es sorprendentemente robusto en comparación a las demás herramientas, en especial al emplear UDP durante las pruebas, donde presentan mayor claridad que las otras (aunque recordemos que al emplear TCP únicamente muestra dos estadísticas), los resultados son sencillos de entender.
2. NetPerf: obtiene resultados similares a IPerf en cuanto a pruebas con TCP se trata, incluso dando en ocasiones mejor resultado, pero prácticamente no funciona a la hora de realizar pruebas con UDP.
3. D-ITG: es casi tan robusto como IPerf, apenas se ve algún caso donde de resultados extraños (como por ejemplo el caso de la duplicidad, donde muestra valores demasiado aumentados) y con la información que ofrece se puede ver con mucha claridad las

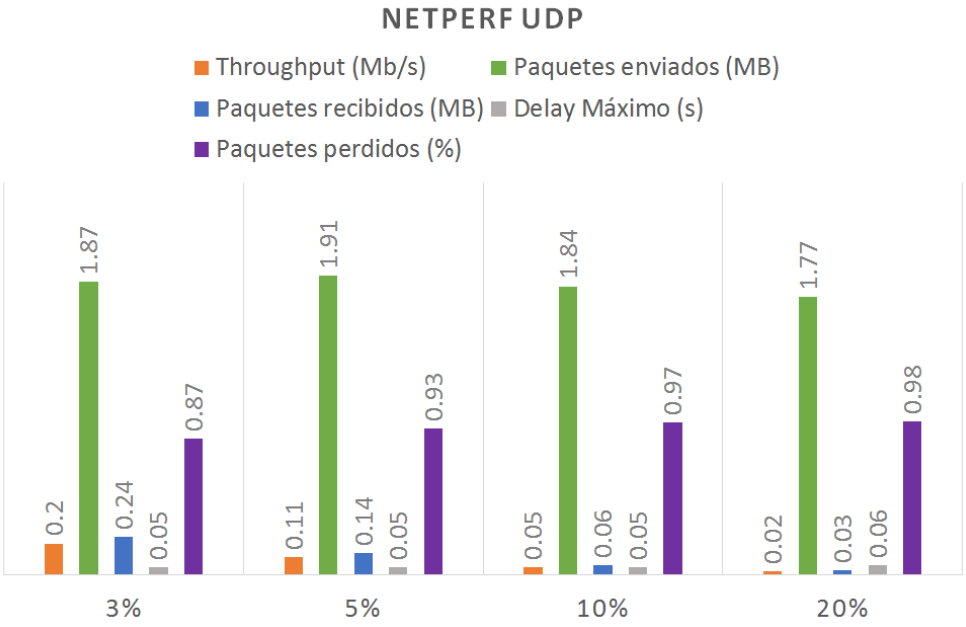


Figura 4.50: NetPerf con protocolo UDP en escenario con porcentaje de duplicidad modificado.

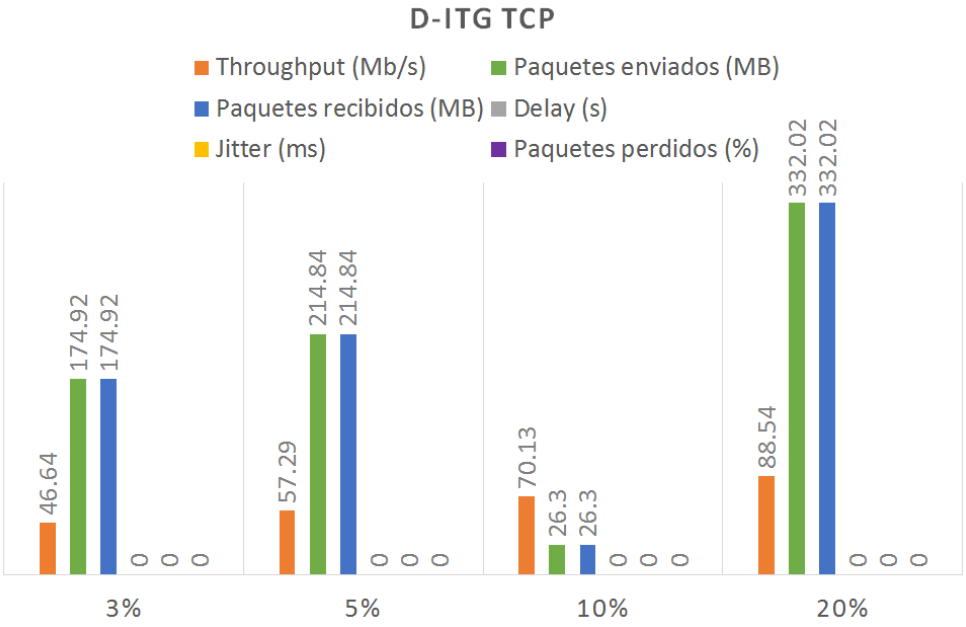


Figura 4.51: D-ITG con protocolo TCP en escenario con porcentaje de duplicidad modificado.

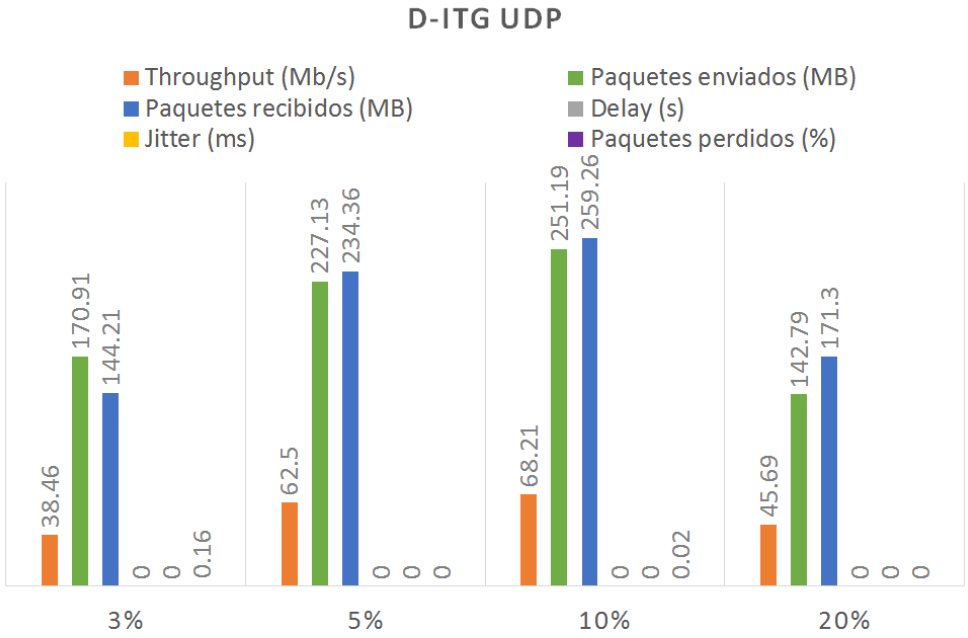


Figura 4.52: D-ITG con protocolo UDP en escenario con porcentaje de duplicidad modificado.

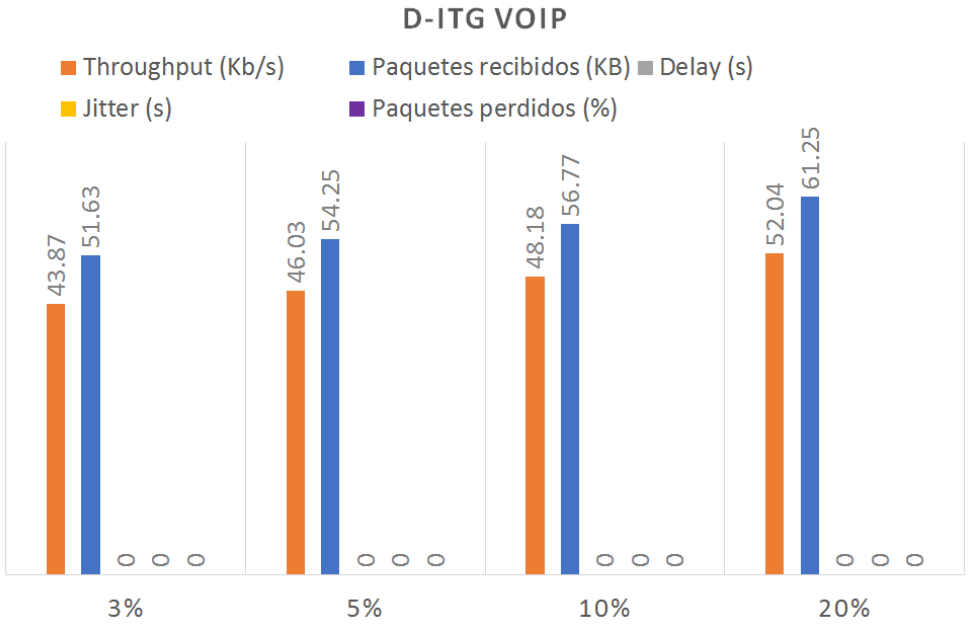


Figura 4.53: D-ITG con protocolo VoIP en escenario con porcentaje de duplicidad modificado.

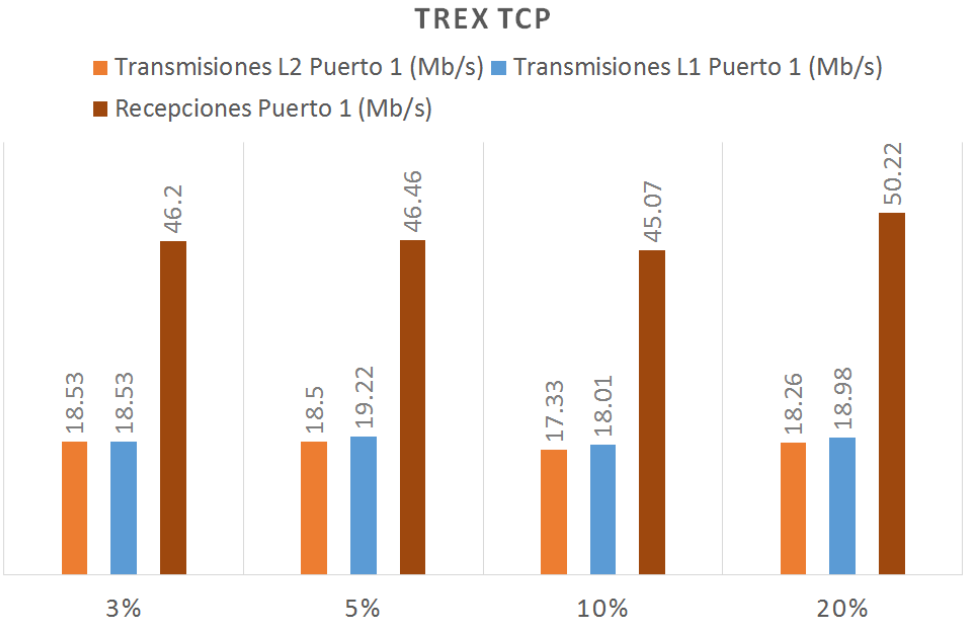


Figura 4.54: TREX con protocolo TCP en escenario con porcentaje de duplicidad modificado.

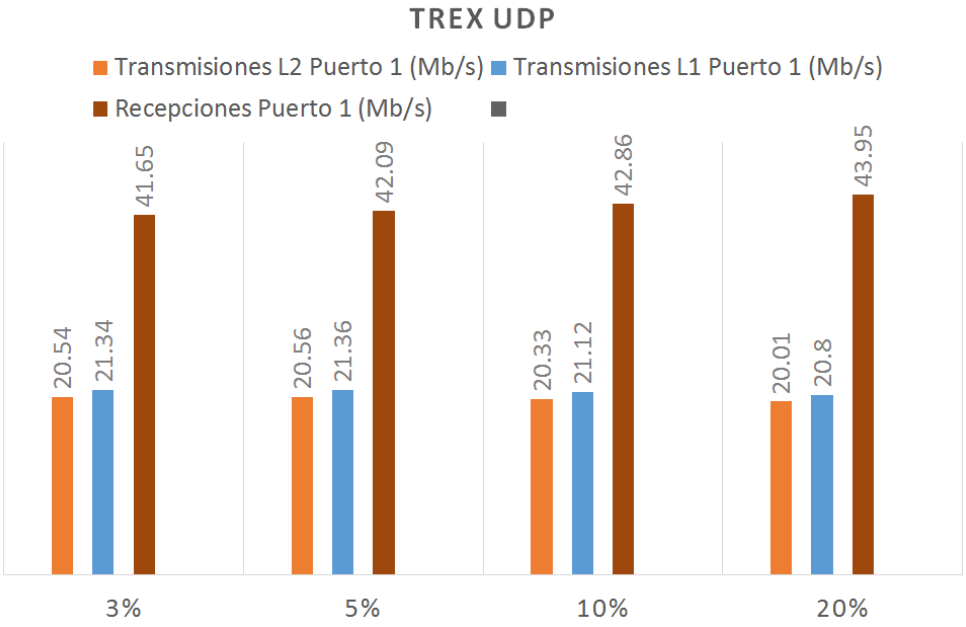


Figura 4.55: TREX con protocolo UDP en escenario con porcentaje de duplicidad modificado.

diferencias entre ambos protocolos al operar en la red. En el caso de las pruebas realizadas con VoIP, vemos como es sensible a los cambios excepto al *jitter*, al cual parece no afectarle.

4. TREX: por las características del funcionamiento de esta herramienta vemos que no se ven tan afectado como otras durante los envíos de datos. Pero no sacamos más datos que las ratios de los flujos de datos, no pudiendo comparar otros valores.

4.9 Pruebas multiflujo

Se repitieron las pruebas con IPerf (Figura 4.56) y D-ITG (Figura 4.57) pero esta vez haciendo que las herramientas ejecutasen 5 flujos de forma simultánea.

Vemos que en ambos casos se da un comportamiento similar respecto a las pruebas anteriores donde se hacían con un solo flujo por prueba. En el caso de las pruebas alterando el ancho de banda de la conexión, se ve como cada flujo reparte el ancho de banda de forma prácticamente proporcional entre los 5 flujos, haciendo que, al mostrar el total, el cual es un sumatorio de los datos de cada flujo, se ven los límites de ancho de banda establecidos en la conexión. Las demás pruebas ven una alteración en cada flujo similar a los casos anteriores mostrando en el total las sumas de las diferencias.

En el caso de TREX, durante las pruebas anteriores ya se hizo ejecutando varios flujos en paralelo, ya que las pruebas con esta herramienta son en base diferentes a las demás y para generar un tráfico de información que pudiese analizarse adecuadamente teniendo en cuenta que debía poder verse las diferencias en cada caso de los probados, necesitábamos un mayor número de flujos para multiplicar el tráfico de los casos de prueba a un nivel adecuado.

4.10 Pruebas múltiples equipos

Se repitieron las pruebas con IPerf (Figura 4.58) y (Figura 4.59 y (Figura 4.60) D-ITG pero esta vez haciendo que las herramientas en los 4 equipos de la red cliente, se alargó el tiempo de prueba para poder poner en ejecución los 4 clientes al mismo tiempo, dando como resultado comportamientos prácticamente idénticos a los descritos en el apartado anterior de pruebas multiflujo, con la diferencia de que, al ser varias pruebas y no una única, los datos se muestran de forma separada y sin mostrar un sumatorio total.

Tras probar y ver cómo reaccionan las herramientas en diferentes casos, continuaremos haciendo una comparativa entre estas.

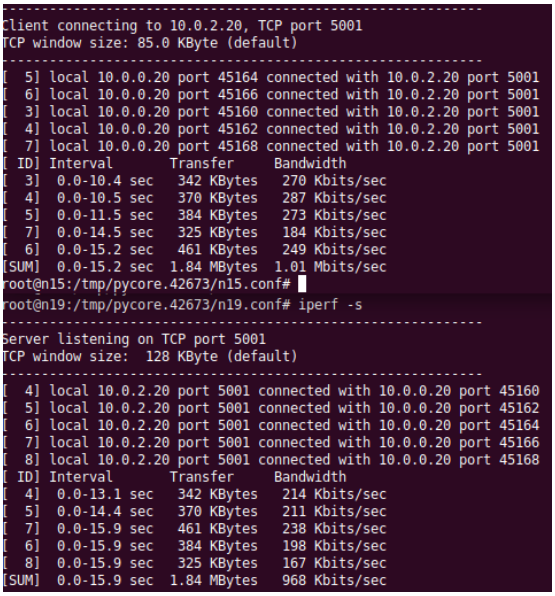


Figura 4.56: Prueba en paralelo con IPerf.

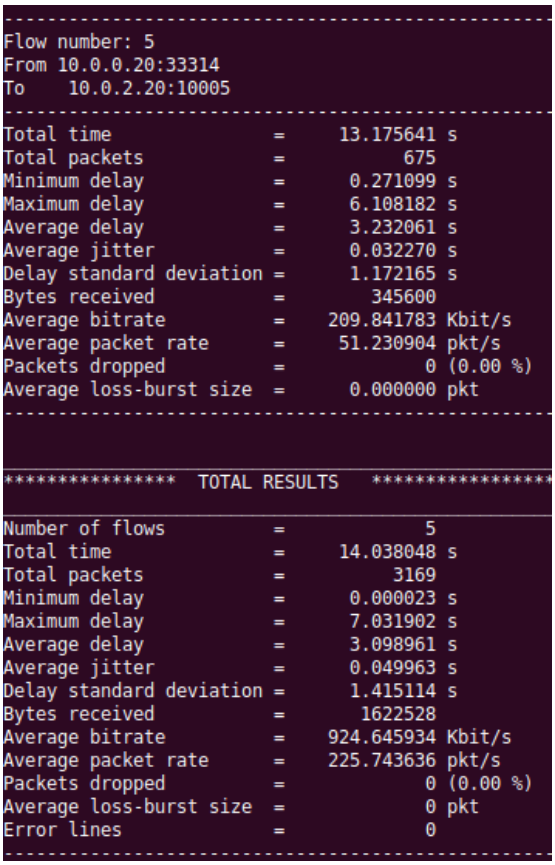


Figura 4.57: Prueba en paralelo con D-ITG.

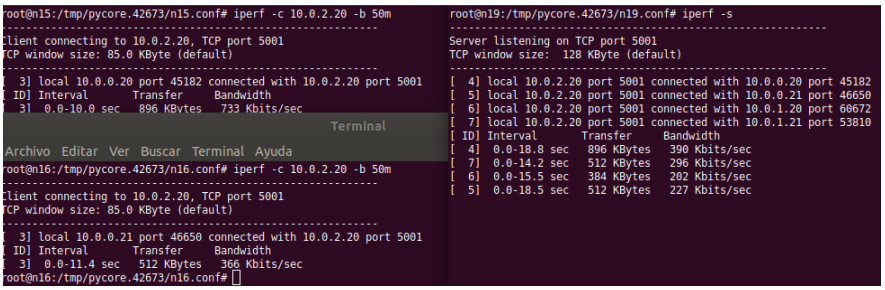


Figura 4.58: Empleando múltiples equipos con IPerf.

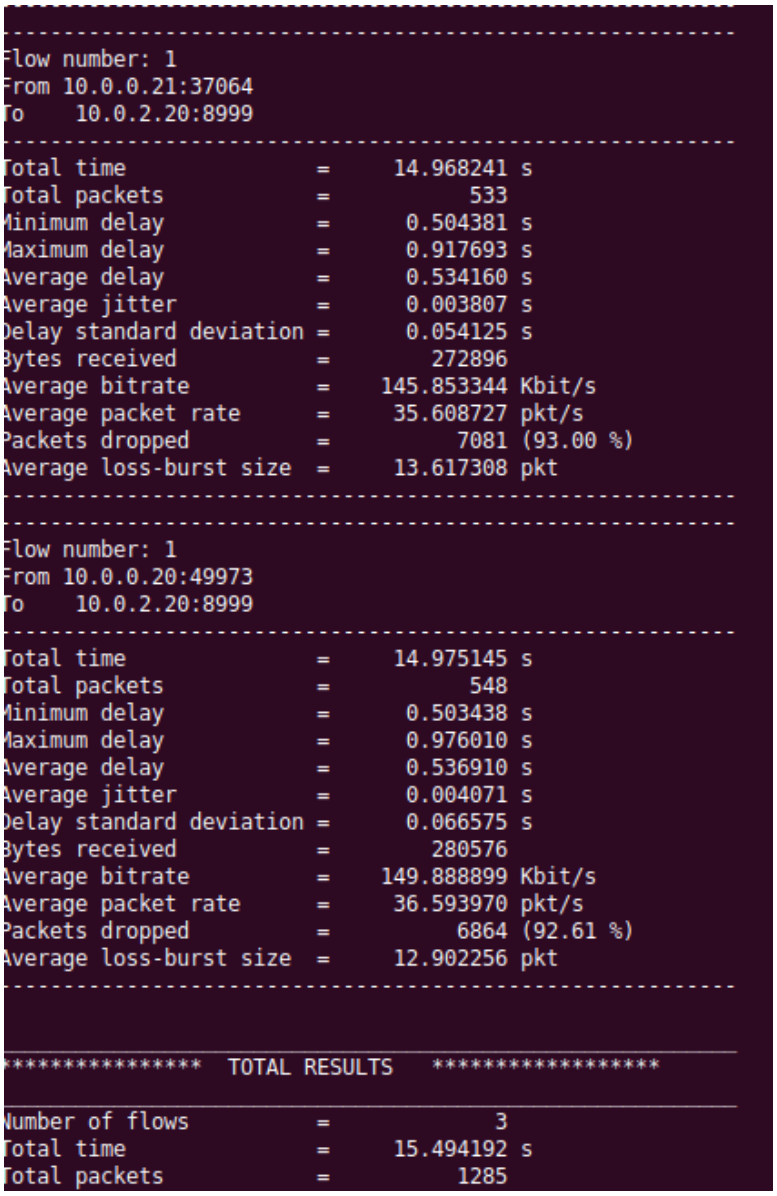


Figura 4.59: Empleando múltiples equipos con D-ITG.

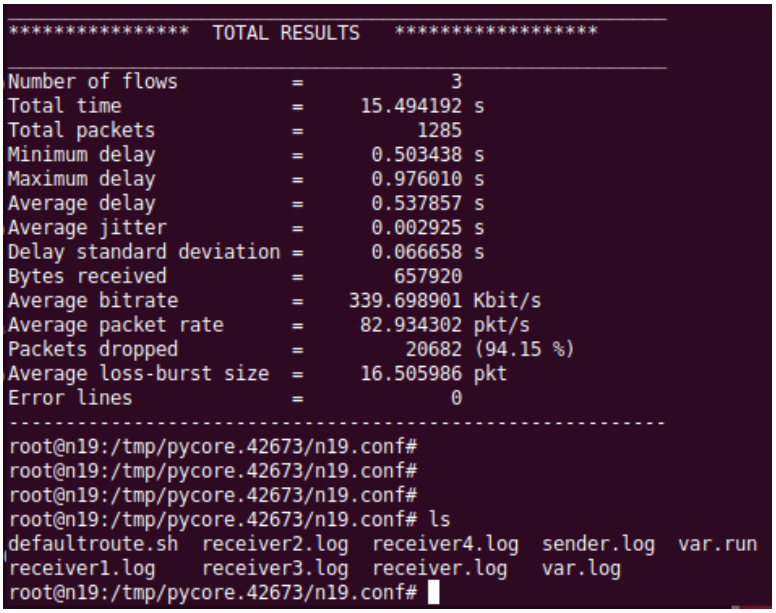


Figura 4.60: Empleando múltiples equipos con D-ITG (continuación).

Capítulo 5

Comparación de características de las herramientas

Procedemos a comparar las características que distinguen cada una, así como los pros y contras observados durante su prueba.

5.1 IPerf

- Características:
  - Puede crear múltiples flujos en un comando.
- Pros:
  - Simple
  - Muy intuitivo
  - Sorprendentemente robusto
- Contras:
  - Poca capacidad para calcular y mostrar información.

5.2 NetPerf

- Características:
  - Permite implementar test predefinidos por la herramienta.
- Pros:
  - Simple de usar
  - Puede exponer gran multitud de tipos de datos.
- Contras:
  - Puede darse que algún test no esté disponible y haya que obtenerlo externamente.
  - Es muy fácil que con funciona correctamente si se dan problemas en la red.

5.3 MGEN

- Características:
  - Diseñado principalmente para operar con scripts.
  - Permite automatizar pruebas.
- Pros:
  - Da pie a realizar pruebas complejas.
- Contras:
  - Complejo de usar.
  - Complicado de entender y aprender
  - No indica los datos directamente
  - Requiere de herramienta externas para poder ver los datos.
  - Demasiado tiempo sin actualizar

5.4 DITG

- Características:
  - Guarda los datos en archivos de log encriptados.
  - Facilita el uso de un servidor de logs
  - Permite el uso de scripts para facilitar pruebas más complejas.
- Pros:
  - Simple de usar
  - Intuitivo
  - Da la información muy completa y de forma clara.
- Contras:
  - No muestra los datos durante la ejecución o tras terminar, solo se puede ver en los archivos de log.

5.5 TREX

- Características:
  - Trabaja sobre las interfaces de red de un equipo directamente.
  - Se pueden usar plantillas personalizadas para pruebas complejas.
  - Indica los datos mientras se ejecuta la simulación para ver lo que ocurre durante las misas en tiempo real.
- Pros:

  - Permite trabajar por conjuntos de equipos.
  - Permite pruebas complejas a partir de casos reales
  - Puede trabajar por rangos de IPs
  - Muy potente, consigues mucha información con solo un comando.
- Contras:
  - Muy complejo y poco intuitivo
  - Difícil de aprender
  - No permite usar toda su potencia en entorno virtual (Solo hemos podido emplear 1UC durante las pruebas de esta exposición)

Finalizaremos comentando a las que hemos llegado durante la realización de este TFG.

Capítulo 6

Conclusiones

Tras haber estudiado las herramientas, haber probado las capacidades principales de cada una y haber puesto a prueba cada una obstaculizando el tráfico con diferentes restricciones, procederemos a continuación a explicar que conclusiones obtenemos acerca de las herramientas, cuales recomendaríamos para su uso y el motivo de su recomendación.

Disponemos tanto de IPerf como de NetPerf para hacer pruebas puntuales entre equipos concretos, de forma que pueda testar la conectividad del equipo para comprobar problemas.

Debido a que NetPerf tiene puede realizar las mismas pruebas que IPerf, además de disponer de mayor número de funcionalidades y que puede realizar pruebas más específicas, se aconsejaría el uso de Netperf por encima de IPerf. Sin embargo, IPerf resulta ser menos propensa a problemas (netperf funciona mal al realizar pruebas con UDP en entornos donde los enlaces presenten algún tipo de problema) y mucho más fácil de emplear, por lo que recomendamos IPerf entre estas dos, siempre que la prueba a realizar entre dentro de lo que IPerf puede mostrar.

MGEN, D-ITG y TREX serían más recomendables a la hora de realizar pruebas de rutina o de mantenimiento, ya que las tres herramientas permiten el uso de scripts o plantillas para realizar pruebas más complejas.

MGEN destaca en el uso de los scripts, tanto por cómo pueden trabajar con los flujos, pudiendo incluso hacerles modificaciones en medio de la prueba, como modificar partes del comportamiento de dichas pruebas y pudiendo programar cuando se ejecutan. Debido a la falta de mantenimiento y el hecho de que depende de programas complementarios para completar su funcionalidad, no se aconseja su uso.

D-ITG permite realizar pruebas para comprobar el estado y velocidad de la red con distintos tipos de tráfico, incluido tráfico de capa de aplicación. Puede realizar varias pruebas al mismo tiempo con diferentes tipos de tráfico y hacia diferentes equipos con comandos relativamente sencillos. Esta herramienta destaca por la muestra de los datos en sus archivos de log y la importancia de su tratamiento, mostrando de forma muy clara los resultados de las

pruebas.

TREX permite recrear flujos de tráfico de datos en base a muestras capturadas anteriormente (archivos PCAP) y, en vez de realizar un análisis tras la ejecución de la prueba, muestra la información del flujo a tiempo real mientras está siendo ejecutado, por lo que sirve para ver el comportamiento del equipo mientras durante el test. El hecho de que pueda generar más de un flujo de datos al mismo tiempo y contra varios equipos permite el realizar pruebas muy realistas y detalladas para comprobar el comportamiento de la red ante distintas aplicaciones y situaciones.

Tras lo explicado anteriormente, a criterio personal, emplearíamos D-ITG para comprobar el estado de la red y sus equipos y emplearíamos TREX para simular la ejecución de programas complejos o situaciones concretas.

Apéndices

Apéndice A

Instalación de las herramientas

Asumimos una distribución de Linux Ubuntu 18.04.2

A.1 IPerf

Comando: `sudo apt-get install iperf`

A.2 NetPerf

Comando: `sudo apt-get install netperf`

A.3 MGEN

Comando: `sudo apt-get install mgen`

A.3.1 TRPR

1. Ir a: https://downloads.pf.itd.nrl.navy.mil/proteantools/
2. Descargar src-trpr-2.1b2.tgz
3. Descomprimimos el archivo, entramos en el directorio resultante
4. hacemos: `make –f Makefile.linux`

A.4 D-ITG

Comando: `sudo apt-get install d-itg`

A.5 TREX

1. Descarga de la herramienta:

   - Comandos:
     (a) `TREX_WEB_URL= http://trex-tgn.cisco.com/trex`
     (b) `mkdir -p /opt/trex`
     (c) `cd /opt/trex`
     (d) `wget --no-cache $TREX_WEB_URL/release/latest`
     (e) `tar -xzvf latest`

2. Configuración básica:

   - Comandos:
     (a) CD <Directorio TREX>
     (b) sudo ./dpdk_setup_ports.py -s
     (c) cp cfg/simple_cfg.yaml /etc/trex_cfg.yaml
     (d) Sudo nano /etc/trex_cfg.yaml

   - Archivo: trex_cfg.yaml:

```
    <none>
  - port_limit      : 2
    version         : 2
  #List of interfaces. Change according to your setup. Use
  ./dpdk_setup_ports.py -s to see available options.

  interfaces    : [<nombre_intefaz1>, <nombre_intefaz2>]  #
  port_info     :  #Port IPs. Change according to your needs. In
  case of loopback, you can leave as is.
      - ip         : <dir_Interfaz1>
        default_gw : <dir_gw_interfaz1>
      - ip         : <dir_Interfaz2>
        default_gw : <dir_gw_interfaz2>

```

A.6 WARP17

Debido a problemas con el entorno virtual, no ha sido posible realizar con éxito la instalación de esta herramienta.

A.7 CORE Network Emulator

Herramienta que emula un entorno de red en tiempo real y que emplearemos para crear el escenario sobre el cual probar las herramientas de generación de tráfico.

Comando: `sudo apt-get install CORE Network`

Apéndice B

Instalación de las interfaces gráficas

B.1 JPerf (Iperf)

1. Requiere Java.
2. Descargar en: https://code.google.com/archive/p/xjperf/downloads (seleccionar ZIP más reciente)
3. Extraer el contenido del ZIP
4. Entrar en el directorio resultante
5. Añadir permisos de ejecución al archivo SH
6. Ejecutar dicho archivo SH

B.2 Flent (Iperf y NetPerf)

1. Comando: `apt-get install flent`

B.3 Core-MGEN

Viene incluido en CORE

B.4 D-ITG GUI

1. Descargar en: http://www.semken.com/downloads/downloads.php?get=itggui-092.zip
2. Descomprimir
3. Ejecutar con: `java -jar ITGGUI.jar`
4. Cambiar rutas en:
   (a) settings>"Binary Directory" (/usr/bin: ubicación de los ejecutables de D-ITG)
   (b) settings>"Logging Directory" (Ubicación donde deseamos guardar los logs)

B.5 TRex

1. TREX Stateful GUI
2. TREX Stateless GUI

Ambas están disponibles para Mac y Windows, no ha sido posible el prepararlas para probar en nuestra máquina virtual de Ubuntu donde realizamos el trabajo.

Apéndice C

Aplicaciones de interfaz gráfica para el uso de las herramientas

En este anexo mostramos algunas herramientas o interfaces gráficas que trabajan con las ya mostradas durante la práctica. Deseamos conocer si resultan de utilidad y en qué medida pueden facilitar el uso de las herramientas. La instalación de estas aplicaciones de interfaz gráfica viene indicada en el anexo anterior a este.

Las herramientas gráficas que analizaremos son JPerf (IPerf), Flent (Iperf y NetPerf), CoreMGEN y D-ITG GUI. Intentamos probar las GUI de TRex, pero no hemos podido instalarlas en la máquina empleada para la realización del TFG.

C.1 JPerf (IPerf)

JPerf dispone únicamente de una pantalla donde se reúnen todas las opciones disponibles (Figura C.1). Remarcamos con diferentes colores las zonas cuya que controlan las diferentes funciones de la herramienta.

- Zona amarilla: muestra el comando tal y como se corresponde a las opciones escogidas
- Zona roja: permite escoger si el equipo actúa de cliente o de servidor y configurar puerto y conexiones.
- Zona azul: permite guardar la configuración actual (icono disquete) o cargar una configuración anterior.
- Zona verde: parámetros de configuración del flujo (protocolo, salida formato, intervalos…)
- Zona naranja: gráfica que muestra la ejecución de la prueba en tiempo real.

La herramienta permite realizar todas las opciones que se pueden realizar por comandos en una única pantalla y se configuran de forma más intuitiva y sencilla, además permite ver la

ejecución de la prueba mientras esta se realiza y permite guardar y reutilizar configuraciones empleadas anteriormente, cosa que la herramienta original no permite de ninguna manera.

Es definitivamente mejor emplear esta interfaz que la herramienta en línea de comando.

C.2 Flent (IPerf y NetPerf)

Pantallas de la herramienta:

- Nueva pantalla: La primera pantalla no muestra ningún parámetro, para poder hacer algo debemos entrar al apartado "File". Figura C.2
- Nuevo test: Figura C.3
- Lista de test disponibles: Figura C.4
- Resultados del test: Muestra una gráfica a la izquierda que cambia según escoges las opciones de la derecha para ver una u otra estadística. Figura C.5

Aunque teóricamente esta herramienta sirve para emplear test que podrían hacer IPerf y Netperf, esta no dispone de tales opciones. Dispone de una decente lista de test a realizar, pero no se nos permite configurar nada para adaptarlo a nuestros requerimientos, además de que muestra la información de forma difícil de comprender y sin aportar claridad al usuario.

C.3 Core-MGEN

Como podemos observar en la interfaz (Figura C.6), la mayor parte de los campos son sencillos de entender, pero dan información acerca de otros como los "`additional MGEN parameters`". Es una interfaz simple, pero requiere un cierto conocimiento sobre la herramienta de antemano.

Además, no funciona a pesar de ser parte directamente de la herramienta de emulación de red.

C.4 D-ITG GUI

- Pantalla inicial (Figura C.7): Dispone de las opciones de configuración de los flujos.
- En la parte superior disponemos de las opciones para ejecutar D-ITG como cliente, servidor o servidor de logs, emplear más de un flujo al mismo tiempo o el uso de plantillas de configuración.
- También podemos movernos entre las pestañas para ver apartados no tan directamente relacionados con la ejecución de los flujos.
- Figura C.8: Lista de plantillas guardadas que permiten cargar una configuración previa al instante.

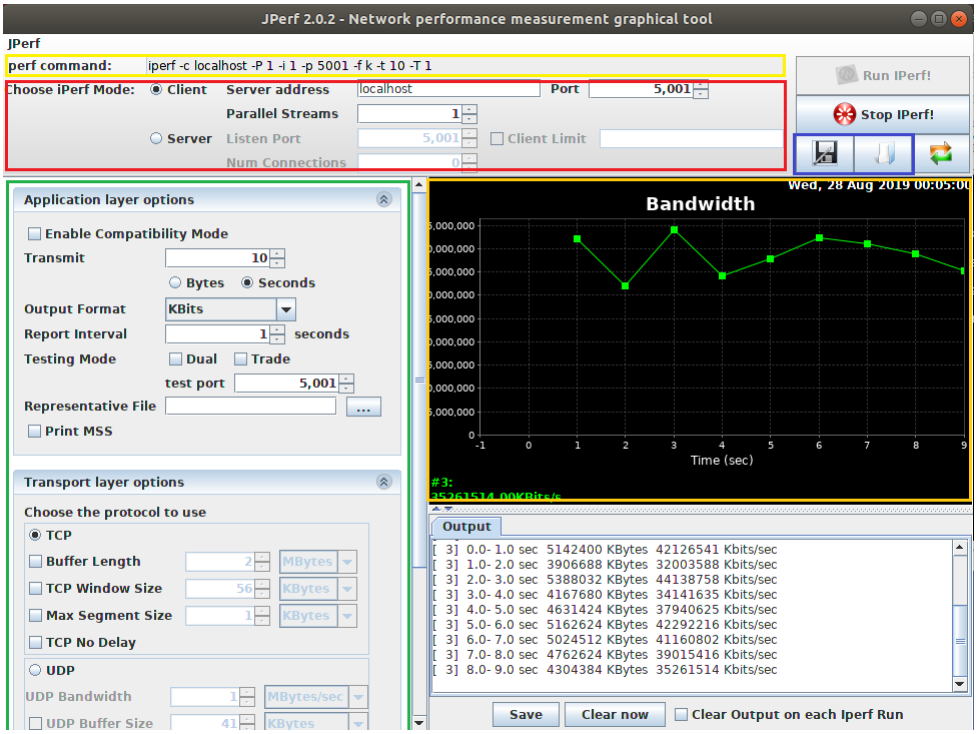


Figura C.1: Interfaz de JPerf

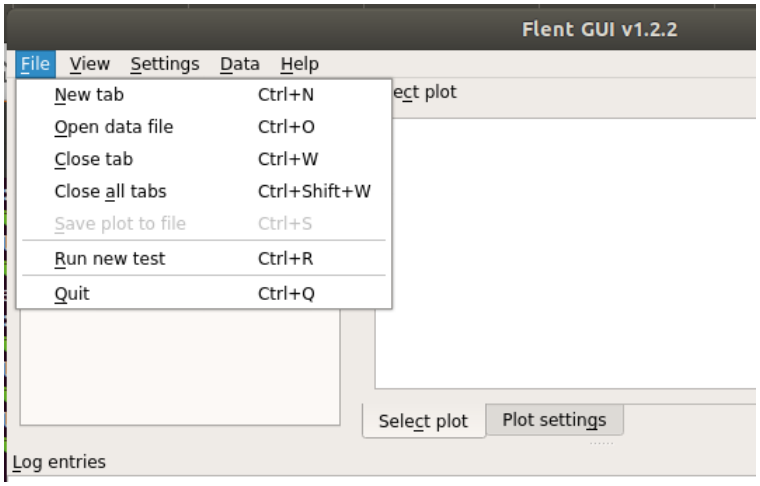


Figura C.2: Empezar una nueva prueba con Flent

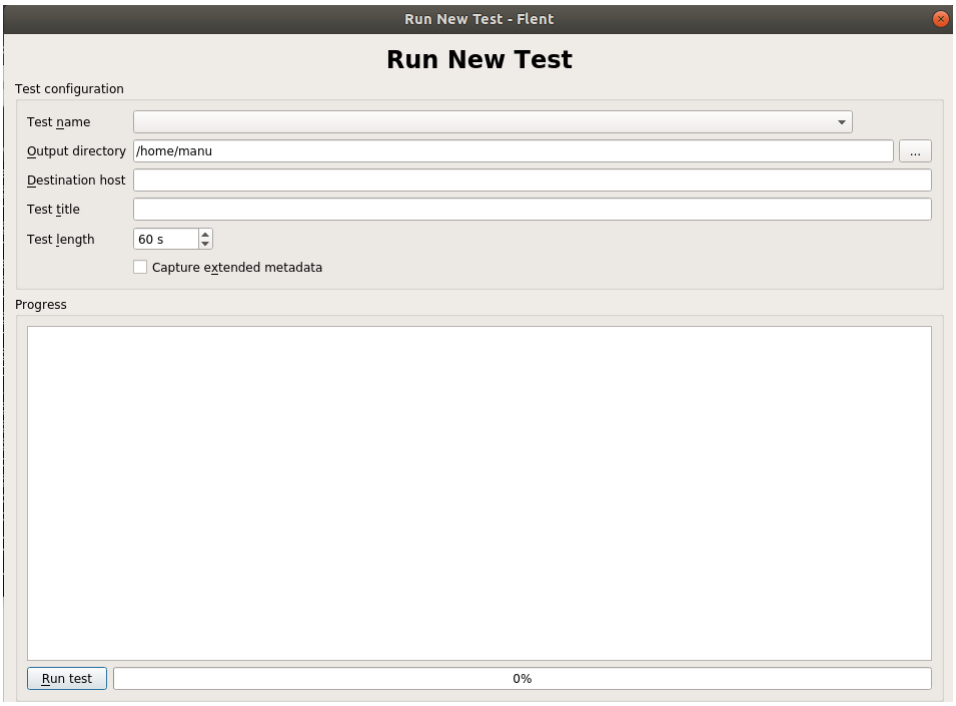


Figura C.3: En la pantalla de nuevo test apenas podemos escoger el test.

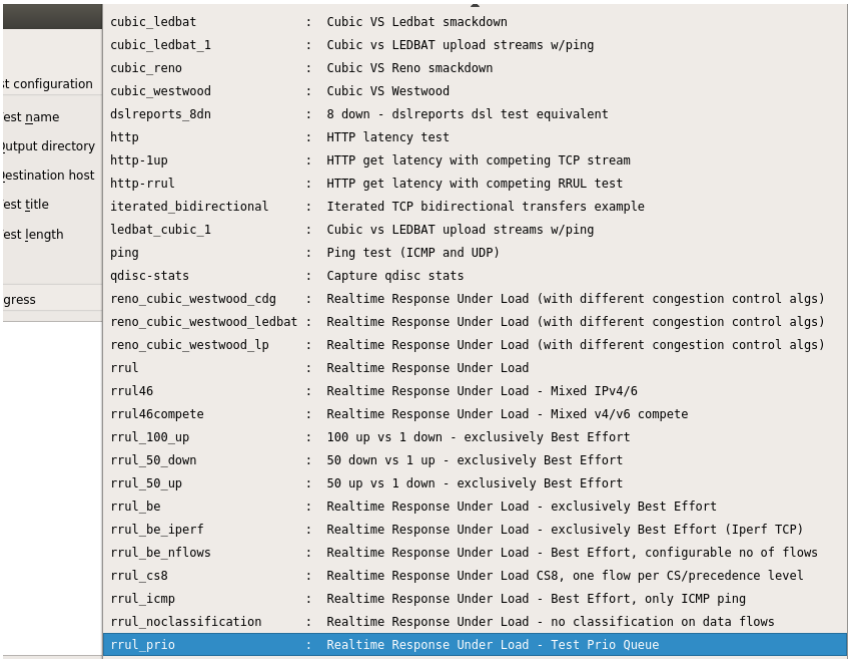


Figura C.4: Lista de test a escoger.

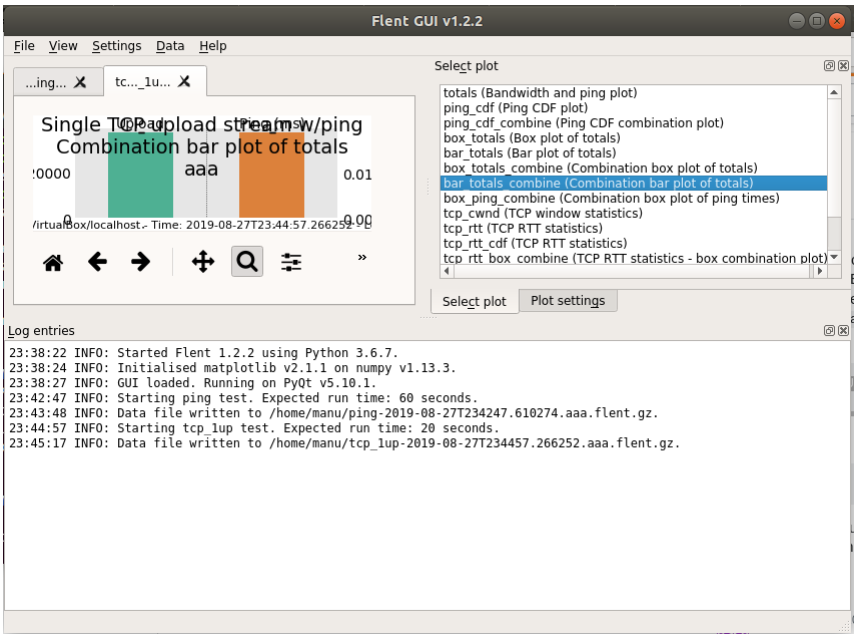


Figura C.5: Muestra de una gráfica resultante de un test.

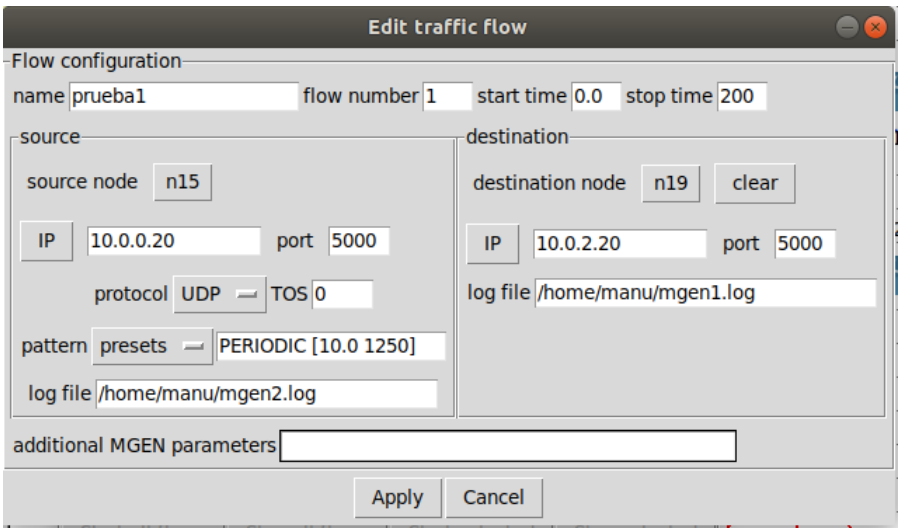


Figura C.6: Interfaz de CORE Network Emulator de manejo de flujos con MGEN

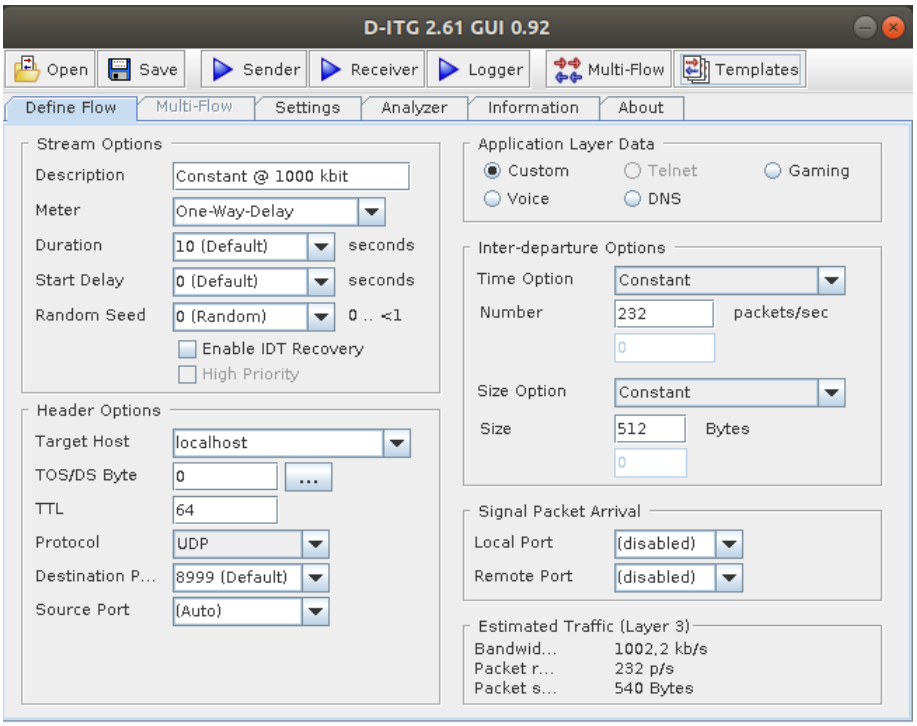


Figura C.7: Interfaz principal de D-ITG GUI

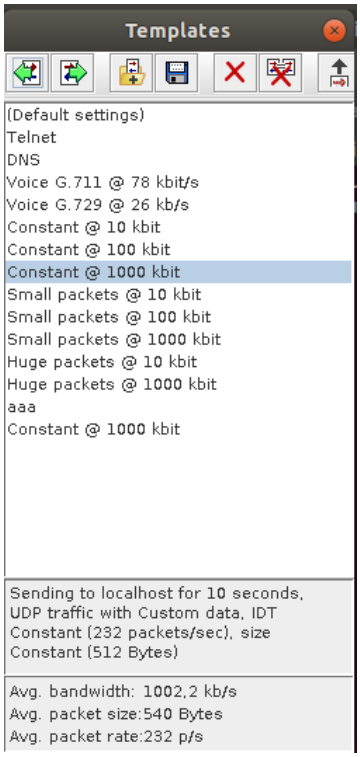


Figura C.8: Lista de plantillas de pruebas guardadas de D-ITG GUI

- Pantalla de configuración (Figura C.9): Permite configurar la ejecución de los comandos de D-ITG, así como parámetros referentes a los logs.
- Pantalla de multiflujo (Figura C.10): Permite configurar más de un flujo al mismo tiempo (sustituye a los scripts de ejecución que se pueden realizar con D-ITG) Observamos que en cuanto esta acción se activa, la pestaña "Define Flow" queda inhabilitada mientras que podemos entrar y salir de la pestaña "Multi-Flow", cuando se desactiva la opción Multi-Flow se restituye la situación de dichas pestañas.
- Pantalla de analizador de logs (Figura C.11): Permite desencriptar los archivos de log de forma que sean legibles o generar archivos de otro formato (como gráficas) con los datos de dichos logs.
- Log desencriptado (Figura C.12): Ejemplo de un resultado de un log desencriptado (mismo formato que en las demás muestras).

La interfaz contempla el funcionamiento completo de la herramienta y está perfectamente adaptada a esta. Actúa como un intermediario entre la herramienta original y permite incluso configurar el directorio de donde recoge los comandos las ejecuciones, de forma que puede emplearse la misma interfaz con diferentes versiones de D-ITG que se pudiese disponer en el equipo.

Esta interfaz cuenta con opciones para facilitar el trabajo del usuario, como el uso de las plantillas o la opción Multiflow que sustituye los scripts (las plantillas pueden guardar configuraciones multiflow también).

Se recomienda el uso de la interfaz para pruebas complejas, ya que para casos sencillos puede ser más cómoda la opción por comandos.

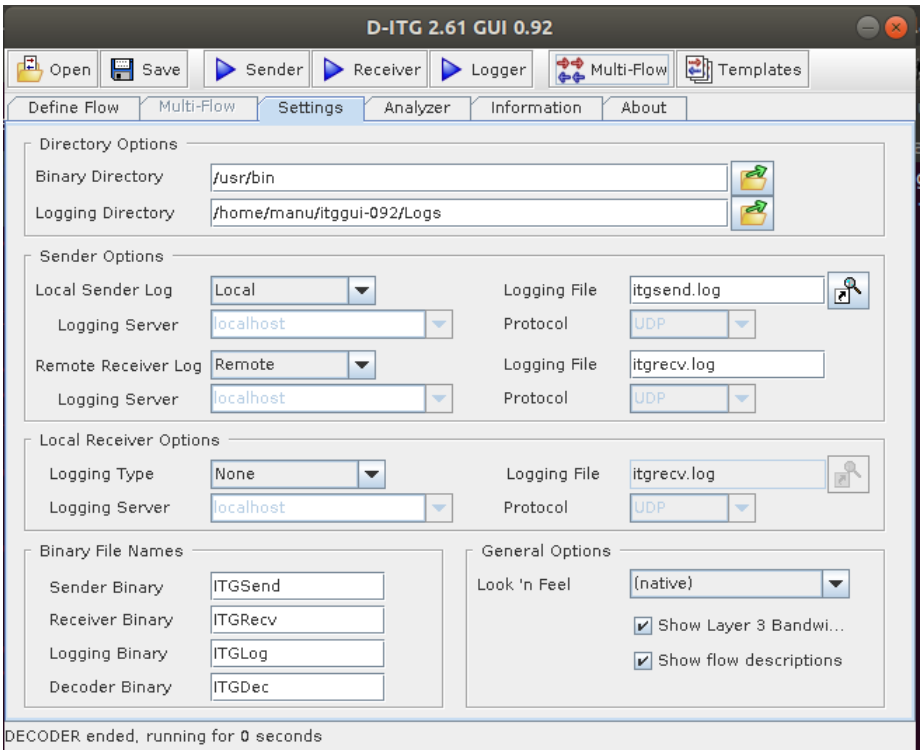


Figura C.9: Interfaz de configuración de D-ITG GUI

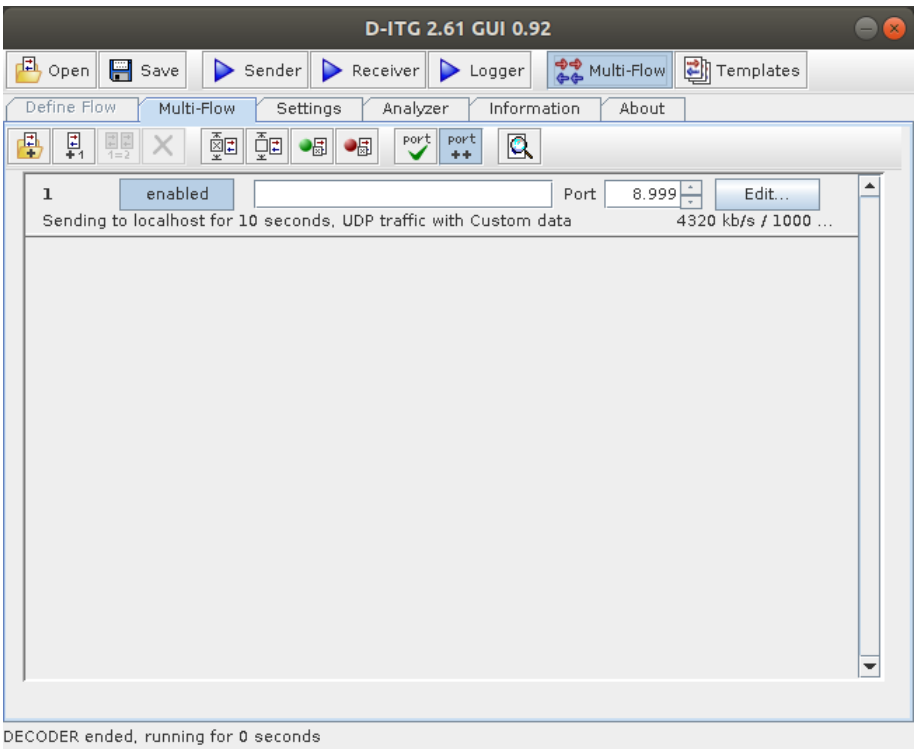


Figura C.10: Interfaz de pruebas multiflujo de D-ITG GUI

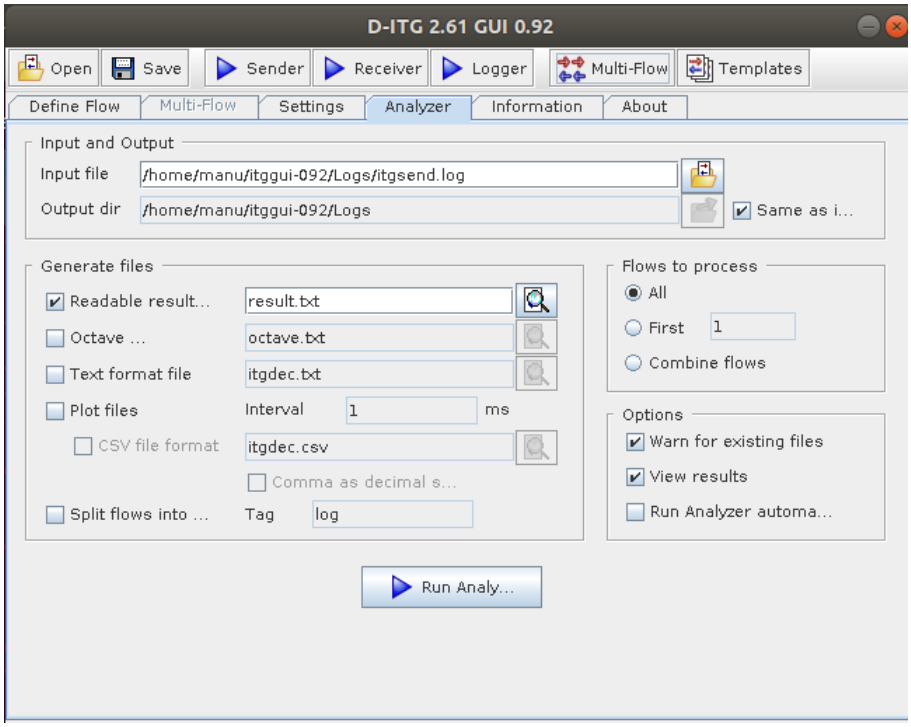


Figura C.11: Interfaz del analizador de logs de D-ITG GUI

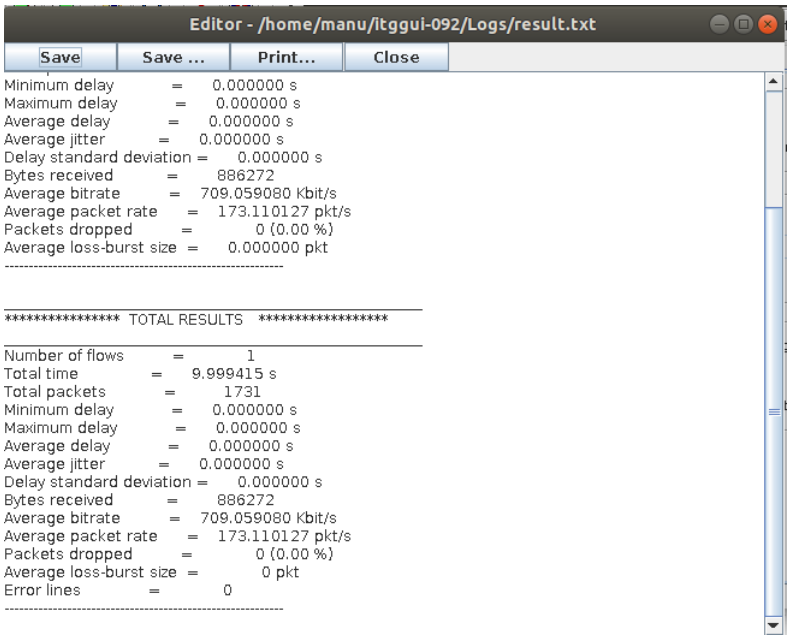


Figura C.12: Muestra de un log analizado de una prueba anteriormente realizada con D-ITG GUI

Apéndice D

Anexo: comandos de las pruebas

D.1 IPerf

1. Prueba TCP Básica: servidor <- Servidor; Cliente <- Cliente; TCP (default)
   - Servidor: `iperf -s`
   - Cliente: `iperf -c 10.0.2.20`
2. Prueba TCP window size iguales: servidor WS = 200K; Cliente WS = 200K
   - Servidor: `iperf -s -w 200K`
   - Cliente: `iperf -c 10.0.2.20 -w 200K`
3. Prueba TCP window size mayor lado servidor: servidor WS = 200K; Cliente WS = 100K
   - Servidor: `iperf -s -w 200K`
   - Cliente: `iperf -c 10.0.2.20 -w 100K`
4. Prueba TCP window size mayor lado cliente; Servidor WS = 100K; Cliente WS = 200K
   - Servidor: `iperf -s -w 100K`
   - Cliente: `iperf -c 10.0.2.20 -w 200K`
5. Prueba TCP viendo el tamaño máximo de segmento: solo vista.
   - Servidor: `iperf -s -m`
   - Cliente: `iperf -c 10.0.2.20 -m`
6. Prueba TCP viendo el tamaño máximo de segmento: mayor en servidor:
   - Servidor: `iperf -s -m -M 1500`
   - Cliente: `iperf -c 10.0.2.20 -m -M 500`
7. Prueba TCP viendo el tamaño máximo de segmento: mayor en cliente:
   - Servidor: `iperf -s -m -M 1500`
   - Cliente: `iperf -c 10.0.2.20 -m -M 5000`

8. Pruebas TCP cliente con paralelismo: Lanza 2 (p=2) peticiones a la vez:

   - Servidor: `iperf -s -m`
   - Cliente: `iperf -c 10.0.2.20 -m -P 2`

9. Pruebas TCP cliente con paralelismo (p=2) y window size mayor en servidor:

   - Servidor: `iperf -s -m -w 200k`
   - Cliente: `iperf -c 10.0.2.20 -m -w 100k -P 2`

10. Pruebas TCP cliente con paralelismo (p=2) y window size mayor en cliente:

    - Servidor: `iperf -s -m -w 100k`
    - Cliente: `iperf -c 10.0.2.20 -m -w 200k -P 2`

11. Pruebas TCP cliente con paralelismo (p = 5):

    - Servidor: `iperf -s`
    - Cliente: `iperf -c 10.0.2.20 -P 5`

12. Prueba TCP test ancho de banda individual:

    - Servidor: `iperf -s`
    - Cliente: `iperf -c 10.0.2.20 -r`

13. Prueba TCP test bidireccional simultaneo:

    - Servidor: `iperf -s`
    - Cliente: `iperf -c 10.0.2.20 -d`

14. Prueba TCP test bidireccional simultaneo:

    - Servidor: `iperf -s`
    - Cliente: `iperf -c 10.0.2.20 -d -P 5`

15. Prueba UDP básica

    - Servidor: `iperf -s -u`
    - Cliente: `iperf -c 10.0.2.20 -u`

16. Prueba UDP intervalos 1 segundo y ancho de banda 50 megas:

    - Servidor: `iperf -s -u -i 1`
    - Cliente: `iperf -c 10.0.2.20 -u -b 50m`

17. Prueba UDP intervalos 5 segundo y ancho de banda 50 megas:

    - Servidor: `iperf -s -u -i 5`
    - Cliente: `iperf -c 10.0.2.20 -u -b 50m`

18. Prueba UDP intervalos 1 segundo y ancho de banda 200 megas:

    - Servidor: `iperf -s -u -i 1`
    - Cliente: `iperf -c 10.0.2.20 -u -b 50m`

19. Prueba UDP intervalos 5 segundo y ancho de banda 200 megas:

    - Servidor: `iperf –s –u –i 5`
    - Cliente: `iperf –c 10.0.2.20 -u –b 50m`

D.2 NetPerf

1. Prueba básica (equivale a Prueba de transferencia TCP)

    - Servidor: `netserver`
    - Cliente: `netperf –H 10.0.2.20`

2. Prueba correos TCP (TCP_MAERTS): sockets de buffers grandes para hacer el envío de una vez:

    - Servidor: `netserver`
    - Cliente: `netperf -H 10.0.2.20 -t TCP_MAERTS -- -s 128K -S 128K`

3. Prueba archivos TCP (TCP_SENDFILE)

    - Servidor: `netserver`
    - Cliente: `netperf -H 10.0.2.20 -F tsf -t TCP_SENDFILE -- -s 128K -S 128K`

4. Prueba tráfico UDP (UDP_STREAM) con tamaño de envío fijado

   Nota: añadida opción -R por error de dirección de retorno no alcanzable

    - Servidor: `netserver`
    - Cliente: `netperf -t UDP_STREAM -H 10.0.2.20 -- -R 1 -m 16K`

5. Prueba tráfico UDP (UDP_STREAM) sin tamaño de envío fijado (se iguala al buffer)

    - Cliente: `netperf -t UDP_STREAM -H 10.0.2.20 -- -R 1`

6. Prueba tráfico UDP (UDP_STREAM) con tamaño de envío fijado y mayor al buffer

    - Cliente: `netperf -t UDP_STREAM -H 10.0.2.20 -- -R 1 –m 16k –S 8K`

7. Prueba tráfico UDP (UDP_STREAM) con tamaño de envío fijado y menor al buffer

    - Cliente: `netperf -t UDP_STREAM -H 10.0.2.20 -- -R 1 –m 8k –S 16K`

8. Prueba Petición Respuesta TCP (TCP_RR)

    - Servidor: `netserver`
    - Cliente: `netperf -t TCP_RR -H 10.0.2.20`

9. Prueba Conexión Cierre TCP (TCP_CC)
   - Servidor: `netserver`
   - Cliente: `netperf -T 1 -H 10.0.2.20 -t TCP_CC -c –C (con –c y –C vemos la carga de cpu local y remota durante la prueba)`
10. Prueba Petición Respuesta UDP (UDP_RR)
    - Servidor: `netserver`
    - Cliente: `netperf -t UDP_RR -H 10.0.2.20`
11. Pruebas Omni (Comprueba los protocolos que encuentre y múltiples valores especificados)
    - Servidor: `netserver`
    - Cliente: `netperf -H 10.0.2.20 -t omni -- -d rr -k "THROUGHPUT,THROUGHPUT_UNITS"`
    - Cliente: `netperf -H 10.0.2.20 -l 10 -j -c -C -t omni -- -d stream –k "THROUGHPUT,THROUGHPUT_UNITS,MIN_LATENCY,MAX_LATENCY,RT_LATENCY,STDDEV_LATENCY,P50_LATENCY,P90_LATENCY,P99_LATENCY,LOCAL_TRANSPORT_RETRANS,REMOTE_TRANSPORT_RETRANS,LOCAL_CPU_UTIL,REMOTE_CPU_UTIL"`
12. Pruebas concurrentes:
    - Se crea un script que ejecute un bucle for para hacer ejecutar varios comandos de testeo al mismo tiempo.

D.3 MGEN

1. Prueba Básica con UDP:
   - Servidor: `mgen instance mgen1`
   - Cliente: `mgen instance mgen1 event "on 1 udp dst 10.0.2.20/55000 periodic [1 1024]"`
2. Prueba Básica con TCP:
   - Servidor: `mgen instance mgen1`
   - Cliente: `mgen instance mgen1 event "on 1 tcp dst 10.0.2.20/55000 periodic [1 1024]"`
3. Prueba Básica con SINK:
   - Servidor: `mgen instance mgen1`
   - Cliente: `mgen instance mgen1 event "on 1 sink dst 10.0.2.20/55000 periodic [1 1024]"`

4. Prueba Modificación flujo:
   - Cliente: `mgen instance mgen1 event "on 1 udp dst 10.0.2.20/55000 periodic [1 1024]"`
   - Servidor: `mgen instance mgen1 event "mod 1 udp dst 10.0.2.20/5500 periodic [1 1024]"`

   Podemos ver la modificación en la consola.
5. Cierre de flujo
   - Cliente: `mgen instance mgen1 event "off 1"`
6. Prueba con Contador de mensajes (COUNT)
   - Servidor: `mgen instance mgen1`
   - Cliente: `mgen instance mgen1 event "on 1 udp dst 10.0.2.20/55000 periodic [1 1024] COUNT 10"`
7. Prueba Patrón periódico:
   - Cliente: `mgen instance mgen1 event "on 1 udp dst 10.0.2.20/55000 periodic [1 1024]"`
   - Servidor: `mgen instance mgen1 event "mod 1 udp dst 10.0.2.20/5500 periodic [0.5 512]"`
8. Prueba Poisson:
   - Cliente: `mgen instance mgen1 event "on 1 udp dst 10.0.2.20/55000 poisson[1 1024]"`
   - Servidor: `mgen instance mgen1 event "mod 1 udp dst 10.0.2.20/5500 poisson [0.5 512]"`
9. Prueba Burst:
   - Cliente: `mgen instance mgen1 event "0.0 ON 1 UDP DST 10.0.2.20/5000 BURST [RANDOM 10.0 PERIODIC [10.0 1024] EXP 5.0]"`
10. Prueba Jitter:
    - Cliente: mgen instance mgen1 event ”0.0 ON 1 UDP DST 10.0.2.20/5000 JITTER [1.0 1024 .5]”
11. Prueba Script

    0.0 ON 1 UDP SRC 5001 DST 127.0.0.1/5001 PERIODIC [1 1024]

    0.0 ON 2 UDP SRC 5002 DST 224.225.1.2/5002 PERIODIC [1 512]

    4.0 MOD 2 POISSON [10 1024]

    1 0.0 OFF

    1 10.0 OFF 2
    - Cliente: mgen input script START 18:34:00

D.4 D-ITG

1. Prueba básica UDP:
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -T UDP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender1.log -x receiver1.log`
2. Prueba pasiva UDP:
   - Cliente: `ITGSend -H -T UDP -a 10.0.2.20 -c 100 -C 10 -t` 15000 -l sender2.log -x receiver2.log
   - Servidor: `ITGRecv -H 10.0.0.20`
3. Prueba usando servidor de Logs:
   - Servidor Log: `ITGLog <- IP: 10.0.3.21`
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -T UDP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender3.log -x receiver3.log -L 10.0.3.21 -X 10.0.3.21`
4. Prueba básica usando TCP:
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -T TCP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender4.log -x receiver4.log -B E 100 W 10 100`
5. Prueba pasiva usando TCP
   - Cliente: `ITGSend -H -T TCP -a 10.0.2.20 -c 100 -C 10 -t 15000 -l sender5.log -x receiver5.log`
   - Servidor: `ITGRecv -H 10.0.0.20`
6. Prueba VoIP
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -a 10.0.2.20 -rp 10000 VoIP -x G.711.2 -h RTP -VAD -c 100 -C 10 -t 15000 -l sender6.log -x receiver6.log`
7. Prueba DNS
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -a 10.0.2.20 -rp 10000 DNS -c 100 -C 10 -t 15000 -l sender7.log -x receiver7.log`
8. Prueba de flujo SCTP
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -a 10.0.2.20 -m RTTM –T SCTP 3 1 –rp 9030 -c 100 -C 10 -t 15000 -l sender8.log -x receiver8.log`

9. Prueba de envío en ráfagas
   - Servidor: `ITGRecv`
   - Cliente: `ITGSend -T UDP -a 10.0.2.20 -l sender9.log -x receiver9.log -B 100 W 10 100`
10. Prueba de envío uniforme
    - Servidor: `ITGRecv`
    - Cliente: `ITGSend -T UDP -a 10.0.2.20 -l sender10.log -x receiver10.log -U 10 100`
11. Prueba de envío Exponencial
    - Servidor: `ITGRecv`
    - Cliente: `ITGSend -T UDP -a 10.0.2.20 -l sender11.log -x receiver11.log -E 100`
12. Prueba de envío Poisson
    - Servidor: `ITGRecv`
    - Cliente: `ITGSend -T UDP -a 10.0.2.20 -l sender12.log -x receiver12.log -O 100`
13. Pruebas con script básica
    - Servidor: `ITGRecv`
    - Cliente: `ITGSend Archivo_Script -l sender13.log -x receiver13.log`
14. Pruebas con script con varios tipos de paquetes
    - Servidor: `ITGRecv`
    - Cliente: `ITGSend Archivo_Script -l sender14.log -x receiver14.log`

D.5 TREX

1. Pruebas stateful:
   - `./t-rex-64 –f <ruta archivo .yaml> -m <multiplicador de la muestra> -c <Cores empleados> -d < segundos duración test>`
2. Pruebas stateless:

   Para empezar:
   - lanzar daemon: `./t-rex-64 –i`

- Abrir consola: `./trex-console`
- Pruebas:
- Abrir flujo: `start –f <dirección .py que genera el l flujo> -m <velocidad del flujo> -a` (para que vaya por todos los puertos, puede usarse –p <número puerto>):

Lista de acrónimos

DPDK *Data Plane Development Kit*

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

DNS *Domain Name System*

YAML *YAML Ain't Markup Language (recursive acronym)*

PCAP *Packet Capture*

TX *Transmit*

RX *Receive*

MB *Megabyte*

Mb/s *Megabits por segundo*

KB *Kilobyte*

Kb/s *Kilobits por segundo*

s *segundos*

ms *milisegundos*

RTP *Real Time Transport Protocol*

IP *Internet Protocol*

GUI *Graphical User Interface*

VoIP *Voice over IP*

CPU *Central Processing Unit*

UC *control unit*

CORE *Common Open Research Emulator*

IPerf *Internet Performance*

NetPerf *Networking Performance*

D-ITG *Distributed Internet Traffic Generator*

MGEN *Multi-Generator*

TRex *Realistic traffic generator*

JPerf *Java Perf*

Flent *FLExible Network Tester*

Bibliografía

[1] "Manual de iperf." [En línea]. Disponible en: https://openmaniak.com/iperf.php

[2] "Documentación de netperf." [En línea]. Disponible en: https://hewlettpackard.github.io/netperf/

[3] *Manual de NetPerf.* [En línea]. Disponible en: http://www.cs.kent.edu/~farrell/dist/ref/Netperf.html

[4] "Documentación de mgen." [En línea]. Disponible en: https://www.nrl.navy.mil/itd/ncs/products/mgen

[5] *Manual de MGEN.* [En línea]. Disponible en: http://cpham.perso.univ-pau.fr/ENSEIGNEMENT/QOS/mgen.html

[6] *Manual D-ITG.* [En línea]. Disponible en: http://ditg.comics.unina.it/manual/D-ITG-2.8.1-manual.pdf

[7] "Documentación trex." [En línea]. Disponible en: https://trex-tgn.cisco.com/

[8] *Manual general de TRex.* [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_book.pdf

[9] *Manual específico de TRex stateless.* [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_stateless.pdf

[10] "Proyecto de github de warp17." [En línea]. Disponible en: https://github.com/Juniper/warp17

[11] "Proyecto de github core network emulator." [En línea]. Disponible en: https://www.nrl.navy.mil/itd/ncs/products/core

[12] "Documentación de core network emulator." [En línea]. Disponible en: http://coreemu.github.io/core/