UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

FACULTAD DE INGENIERÍA

INGENIERÍA ELECTRÓNICA

TRABAJO DE GRADO

MODALIDAD DE INVESTIGACIÓN

"Algoritmo de volcado del tráfico de datos para redes inalámbricas sobre una red definida por software"

Autores:

Jann Camilo Sánchez Huertas

Sebastian Romero Avendaño

Director:

Gustavo Adolfo Puerto Leguizamón

Bogotá, 2019

TABLA DE CONTENIDO

1. Generalidades 10
   1.1 Planteamiento del problema 10
   1.2 Justificación 11
   1.3 Objetivos 12
      1.3.1 Objetivo General 12
      1.3.2 Objetivos Específicos 12
   1.4 Alcances y limitaciones 13
2. Marco teórico 14
   2.1 Redes fijas 14
      2.1.1 CPE 14
      2.1.2 ADSL 14
      2.1.3 FTTH 14
      2.1.4 Cable 15
      2.1.5 WiFi 16
      2.1.6 WiMAX 16
   2.2 Redes móviles 16
      2.2.1 LTE 17
      2.2.2 4G 17
      2.2.3 5G 17
      2.2.4 Comparación de las generaciones de telefonía móvil 19
   2.3 Redes definidas por software (SDN) 19
      2.3.1 Arquitectura 20
      2.3.2 Protocolo OpenFlow 21
   2.4 Controlador ONOS 21
   2.5 Mininet 23
      2.5.1 Ventajas 23
      2.5.2 Limitaciones 24
   2.6 Parámetros en la calidad de servicio (QoS) 24

2.6.1 Rendimiento (throughput) 24
2.6.2 Retardo (delay) 25
2.6.3 Variabilidad (jitter) 25
2.6.4 Tasa de paquetes perdidos 25
2.7 Volcado del tráfico de datos en redes móviles 26
2.7.1 WiFi offload 26
3. Antecedentes 27
3.1 Configuración de redes basadas en software 27
3.2 Virtualización de redes 28
3.3 Aplicaciones SDN 29
3.3.1 SND e IoT 29
3.3.2 Ataques dinámicos y SDN 29
3.3.3 Calidad del servicio (QoS) y SDN 30
3.4 Desafíos de SDN y soluciones existentes 31
3.5 Volcado del tráfico de datos en SDN 31
3.5.1 Balance de carga 31
3.5.2 Descarga de datos con balanceo de carga 32
4. Mininet y ONOS 34
4.1 Emulador de redes Mininet 34
4.2 Controlador ONOS 36
4.3 Conectar mininet con ONOS 38
4.4 Generador de tráfico D-ITG 41
4.5 Generador de tráfico iperf 44
5. Desarrollo del algoritmo 46
5.1 Crear una nueva aplicación en ONOS 46
5.2 Algoritmo del volcado del tráfico de datos 50
5.3 Comandos preestablecidos en ONOS 54
5.4 Preparación del entorno de pruebas en ONOS 56
5.5 Escalado de datos 62
5.6 Topología 63

5.6.1 Implementación en mininet 64
6. Implementación física 65
6.1 OpenvSwitch 65
6.2 Topología red SDN 68
6.3 Tráfico generado 69
7. Resultados 70
7.1 Consideraciones previas 70
7.2 Tráfico UDP 72
7.3 Tráfico TCP 83
7.4 Implementación física 94
8. Trabajos futuros 96
9. Conclusiones 98
Referencias 99
Anexos 102

INDICE DE FIGURAS

Figura 2.1. Arquitectura general – SDN [14] 20
Figura 3.1. Cronología redes virtuales (Fuente: https://agema.deltaww.com/blog_con.php?id=87) 28
Figura 3.2. Simulación ataque DDos [28]. 29
Figura 3.3. Algoritmo (GSOA) [32]. 32
Figura 3.4. Algoritmo de descarga parcial de datos [20] 33
Figura 4.1. Comando de instalación de git 34
Figura 4.2. Comando para obtener copia del código fuente de mininet 34
Figura 4.3. Comando de instalación del emulador mininet 35
Figura 4.4. Comando para comprobar la correcta instalación de Mininet 35
Figura 4.5. Comandos de instalación de Java8 36
Figura 4.6. Comando de instalación de curl 36
Figura 4.7. Comando para descargar el archivo “onos-2.1.0.tar.gz” 36
Figura 4.8. Comandos para extraer el archivo “onos-2.1.0.tar.gz” y cambiar el nombre de la carpeta por “onos”. 37
Figura 4.9. Modificación del archivo “onos-service” 37
Figura 4.10. Inicio de ONOS en la CLI 37
Figura 4.11. Inicio de la GUI de ONOS 38
Figura 4.12. Sección de aplicaciones en la GUI de ONOS – OpenFlow desactivado 39
Figura 4.13. Sección de aplicaciones en la GUI de ONOS – OpenFlow activado 39
Figura 4.14. Prueba de conexión entre mininet y onos 40
Figura 4.15. Prueba de conexión entre mininet y onos – Con hosts visibles 40
Figura 4.16. Comando de instalación del generador de tráfico D-ITG 41
Figura 4.17. Comando para abrir emulador de terminal 42
Figura 4.18. Emulador de terminales para h1 y h4 42
Figura 4.19. Host h1 configurado como receptor 42

Figura 4.20. Host h4 configurado como emisor 42
Figura 4.21. Flujo de bytes en la GUI de ONOS 43
Figura 4.22. Registro almacenado en el archivo “receiver.log” 43
Figura 4.23. Comando de instalación del generador de tráfico iperf 44
Figura 4.24. Host h1 configurado como receptor 45
Figura 4.25. Host h4 configurado como emisor 45
Figura 4.26. Reporte en el receptor 45
Figura 5.1. Comando de instalación de Apache Ant 46
Figura 5.2. Comando de instalación de maven 46
Figura 5.3. Comando para crear una nueva aplicación en ONOS 47
Figura 5.4. Resultado luego de la correcta construcción de la aplicación 47
Figura 5.5. Modificación del archivo “pom.xml” 48
Figura 5.6. Comando para compilar el código fuente de la aplicación 48
Figura 5.7. Comando para instalar la aplicación en ONOS 48
Figura 5.8. Comandos que se deben ejecutar si no reconoce el comando onos-app 49
Figura 5.9. Resultado del comando “app -s” en la CLI de ONOS 49
Figura 5.10. Creación de un nuevo comando con la aplicación foo-app 49
Figura 5.11. Comando para reinstalar la aplicación en ONOS 50
Figura 5.12. Comando para activar la aplicación en ONOS 50
Figura 5.13. Funcionamiento del algoritmo en el tiempo – Parte 1 52
Figura 5.14. Funcionamiento del algoritmo en el tiempo – Parte 2 52
Figura 5.15. Algoritmo del volcado de tráfico de datos 53
Figura 5.16. Mensajes entre controlador y red 54
Figura 5.17. Tasa de transmisión mensajes entre el controlador y la red 54
Figura 5.18. Comando “summary” en la CLI de ONOS 55
Figura 5.19. Comando “portstats” en la CLI de ONOS 55
Figura 5.20. Comando “links” en la CLI de ONOS 55
Figura 5.21. Comando “edge-ports” en la CLI de ONOS 55
Figura 5.22. Comando “portstate” en la CLI de ONOS 56

Figura 5.23. GUI de ONOS después de ejecutar el comando “portstate” 56
Figura 5.24. Modificación en el archivo “PortStatsCollector.java” 57
Figura 5.25. Modificación en el archivo “pom.xml” 58
Figura 5.26. Contenido del archivo “foo-app-1.0-SNAPSHOT.jar” 58
Figura 5.27. Sección de aplicaciones en ONOS – Descarga de la aplicación “OpenFlow Base Provider” 59
Figura 5.28. Sección de aplicaciones en ONOS – Desinstalación de la aplicación “OpenFlow Base Provider” 59
Figura 5.29. Contenido del archivo “onos-providers-openflow-device-2.0.0.jar” 60
Figura 5.30. Sección de aplicaciones en ONOS – Instalación de la aplicación “OpenFlow Base Provider” 61
Figura 5.31. Sección de aplicaciones en ONOS – Activación de la aplicación “OpenFlow Base Provider” 61
Figura 5.32. Comando “portstats -d” en la CLI de ONOS 62
Figura 5.33. Topología 63
Figura 5.34. Agregar controlador ONOS en la red emulada 64
Figura 5.35. Definición de los enlaces - anchos de banda 64
Figura 6.1. Interfaces de red 65
Figura 6.2. Creación de un nuevo enlace o puente 66
Figura 6.3. Interfaz Ethernet agregada 66
Figura 6.4. Editor de redes inalámbricas 66
Figura 6.5. Interfaz de red inalámbrica agregada 67
Figura 6.6. Interfaz de red inalámbrica agregada 67
Figura 6.7. Resultado final al crear el puente o enlace 67
Figura 6.8. Topología de la red SDN implementada 68
Figura 6.9. Raspberry Pi 3 – OpenvSwitch 69
Figura 7.1. Host 1 – Throughput – Reporte de Wireshark 71
Figura 7.2. Host 1 – Throughput – Reporte de ONOS 71
Figura 7.3. Prueba 1 con $Kp = 1$ y $Kc = 4$ 72

Figura 7.4. Prueba 2 con $Kp = 1$ y $Kc = 4$ 73
Figura 7.5. Prueba 3 con $Kp = 1$ y $Kc = 4$ 73
Figura 7.6. Prueba 4 con $Kp = 1$ y $Kc = 4$ 74
Figura 7.7. Prueba 5 con $Kp = 1$ y $Kc = 4$ 74
Figura 7.8. Prueba 6 con $Kp = 1$ y $Kc = 4$ 75
Figura 7.9. Prueba 1 con $Kp = 5$ y $Kc = 4$ 76
Figura 7.10. Prueba 2 con $Kp = 5$ y $Kc = 4$ 76
Figura 7.11. Prueba 3 con $Kp = 5$ y $Kc = 4$ 77
Figura 7.12. Prueba 4 con $Kp = 5$ y $Kc = 4$ 77
Figura 7.13. Prueba 5 con $Kp = 5$ y $Kc = 4$ 78
Figura 7.14. Prueba 6 con $Kp = 5$ y $Kc = 4$ 78
Figura 7.15. Prueba 1 con $Kp = 5$ y $Kc = 10$ 79
Figura 7.16. Prueba 2 con $Kp = 5$ y $Kc = 10$ 80
Figura 7.17. Prueba 3 con $Kp = 5$ y $Kc = 10$ 80
Figura 7.18. Prueba 4 con $Kp = 5$ y $Kc = 10$ 81
Figura 7.19. Prueba 5 con $Kp = 5$ y $Kc = 10$ 82
Figura 7.20. Prueba 6 con $Kp = 5$ y $Kc = 10$ 82
Figura 7.21. Prueba 1 con $Kp = 1$ y $Kc = 4$ 83
Figura 7.22. Prueba 2 con $Kp = 1$ y $Kc = 4$ 84
Figura 7.23. Prueba 3 con $Kp = 1$ y $Kc = 4$ 84
Figura 7.24. Prueba 4 con $Kp = 1$ y $Kc = 4$ 85
Figura 7.25. Prueba 5 con $Kp = 1$ y $Kc = 4$ 85
Figura 7.26. Prueba 6 con $Kp = 1$ y $Kc = 4$ 86
Figura 7.27. Prueba 1 con $Kp = 5$ y $Kc = 4$ 87
Figura 7.28. Prueba 2 con $Kp = 5$ y $Kc = 4$ 87
Figura 7.29. Prueba 3 con $Kp = 5$ y $Kc = 4$ 88
Figura 7.30. Prueba 4 con $Kp = 5$ y $Kc = 4$ 88
Figura 7.31. Prueba 5 con $Kp = 5$ y $Kc = 4$ 89

Figura 7.32. Prueba 6 con $Kp = 5$ y $Kc = 4$ 89
Figura 7.33. Prueba 1 con $Kp = 5$ y $Kc = 10$ 90
Figura 7.34. Prueba 2 con $Kp = 5$ y $Kc = 10$ 91
Figura 7.35. Prueba 3 con $Kp = 5$ y $Kc = 10$ 91
Figura 7.36. Prueba 4 con $Kp = 5$ y $Kc = 10$ 92
Figura 7.37. Prueba 5 con $Kp = 5$ y $Kc = 10$ 92
Figura 7.38. Prueba 6 con $Kp = 5$ y $Kc = 10$ 93
Figura 7.39. Reporte generador de tráfico 94
Figura 7.40. Prueba con $Kp = 1$ y $Kc = 4$ 95
Figura 7.41. Prueba con $Kp = 5$ y $Kc = 4$ 95
Figura 7.42. Prueba con $Kp = 5$ y $Kc = 10$ 95
Figura 8.1. Propuesta para mejorar el funcionamiento del algoritmo 96

1. Generalidades

1.1 Planteamiento del problema

Se prevé que las redes definidas por software (Software Define Networking, SDN por sus siglas en inglés) serán las responsables de proveer una implementación en los servicios de red de una manera dinámica y escalable, siendo esta una solución al crecimiento exponencial que se viene presentando en el número de dispositivos conectados a la red ya que el número de usuarios de teléfonos móviles en 2018 es de más de 5.000 millones [1]. Se prevé que en 2025 habrán 40.000 millones de dispositivos inteligentes, además a nivel empresarial y social, se estima que habrán 100.000 millones de conexiones a nivel global, que serán utilizadas en servicios públicos, transporte, industria manufacturera, cuidado de la salud, agricultura, finanzas, entre otras [2].

En este proyecto se hará uso de las redes SDN con el fin garantizar que un nodo o host que esté conectado a una red mantenga una alta disponibilidad frente a cualquier tipo de imprevisto, mejorando de esta forma la resiliencia de la red, además de proporcionar una alternativa para el transporte de grandes volúmenes de datos (Big Data), donde se hará uso del concepto de volcado del tráfico de datos.

Del mismo modo, se espera que el tráfico mundial de datos móviles se multiplique por siete entre 2016 y 2021, además, el tráfico de datos móviles crecerá a una tasa de crecimiento anual compuesta (Compound Annual Growth Rate, CAGR por sus siglas en inglés) de 47 por ciento en este mismo periodo, alcanzando 49.0 exabytes por mes para el año 2021 [3].

De acuerdo con estos pronósticos, hay una necesidad de tener nuevas alternativas para poder manejar el crecimiento en el tráfico de datos, para esto se propone hacer uso del paradigma del balanceo de cargas, que permitiría volcar el flujo de datos de una red inalámbrica hacia una red fija, beneficiando a la red inalámbrica ya que los recursos de esta son más escasos y costosos, en ese orden de ideas ¿puede este paradigma aportar para que las redes inalámbricas tengan una mayor capacidad en el manejo del tráfico de datos y además mejorar la resiliencia de la red?

1.2 Justificación

El dramático aumento que se está presentando en el número de dispositivos conectados a las redes fijas y móviles, conlleva a un problema en la administración de los recursos limitados que poseen las redes, las conexiones 4G móviles globales crecerán de 2.1 billones en 2016 a 6.1 billones para el 2021 a una tasa compuesta anual del 24 por ciento, además, las conexiones 5G aparecerán en la escena en 2020 y crecerán más de un mil por ciento de 2.3 millones en 2020 a más de 25 millones en 2021 [3], esto implica por supuesto un aumento en el tráfico de datos, que necesitara de un manejo adecuado para evitar sobrecargas, es aquí donde surge el concepto de volcado del tráfico de datos, que puede ser una solución directa y eficiente a este problema.

Estos factores sumados a la necesidad de dar una solución que no afecte de manera dramática la infraestructura física existente, permiten proponer que el concepto de volcado del tráfico de datos sea implementado sobre un modelo de red basado en el paradigma SDN, ya que este tiene grandes ventajas en cuanto a la administración y gestión de una red.

Además, la implementación de una red SDN no implica cambios drásticos en la infraestructura física de una red, lo que permitiría mitigar el impacto ambiental en términos de residuos electrónicos, ya que a nuestro ritmo actual de producir y desechar, la Organización de las Naciones Unidas advierte: en 2050, la cantidad de basura electrónica acumulada habrá alcanzado las 120 millones de toneladas [4].

A nivel académico se pretende desarrollar un algoritmo que permita manejar el concepto de volcado del tráfico de datos sobre una red definida por software, lo que implica un enorme aporte ya que sirve de base para desarrollos futuros de mayor magnitud y debido a la novedad del desarrollo se abre la posibilidad de generar nuevas líneas de investigación hacia áreas como las telecomunicaciones y la telemática.

A nivel socioeconómico, esto permitirá a los proveedores de servicios de telefonía móvil mejorar la calidad del servicio ofrecido, al incrementar el ancho de banda y evitar sobrecargas en sus redes, lo que los beneficia económicamente al poder soportar una mayor cantidad de dispositivos conectados a sus redes, además, el paradigma SDN puede dar una reducción en costos de operación y mantenimiento de la red. Por otro lado, el mejorar la calidad del servicio representa un gran potencial para los usuarios.

La motivación para la realización de este trabajo es el deseo de conocer más a fondo acerca del paradigma SDN, que se prevé tendrá una implementación generalizada en un futuro no muy lejano, y también acerca de las redes inalámbricas con las que convivimos e interactuamos todos los días. Para a partir de esto, poder dar una solución a las redes inalámbricas respecto al creciente aumento en el tráfico de datos que se está presentando y que como se mencionó antes se intensificará en los próximos años.

1.3 Objetivos

1.3.1 Objetivo General

- Evaluar la aplicabilidad del concepto de volcado del tráfico de datos para redes inalámbricas haciendo uso del paradigma de redes definidas por software.

1.3.2 Objetivos Específicos

- Reconocer características de redes móviles, fijas y redes definidas por software, así como los parámetros que intervienen en estas, en términos de desempeño y eficiencia.

- Identificar los mecanismos que existen dentro de las redes definidas por software en aspectos como calidad de servicio y de volcado del tráfico de datos.

- Desarrollar un algoritmo haciendo uso de los parámetros de red identificados, que permita manejar el concepto de volcado del tráfico de datos sobre una red definida por software.

- Evaluar las capacidades del algoritmo propuesto en términos del desempeño de los parámetros seleccionados mediante procesos de emulación.

1.4 Alcances y limitaciones

Se desarrollará un algoritmo que permitirá manejar el concepto de volcado del tráfico de datos sobre una red SDN, además sobre la red se van a monitorear parámetros de desempeño en términos de QoS. En ese orden de ideas se realizará un desarrollo sobre cada una de las capas de la arquitectura de una red SDN:

- *Capa de aplicación:* Desarrollo del algoritmo e implementación dentro del controlador ONOS.
- *Capa de control:* Implementación del controlador ONOS dentro de la red.
- *Capa de infraestructura:* Despliegue de una red básica, conectada con el controlador ONOS a través del protocolo OpenFlow.

Sin embargo, debido a las limitaciones económicas y de recursos físicos (hardware) solo se hará el modelamiento de una red SDN que haga uso del concepto de volcado del tráfico de datos con una topología de red sin mucha complejidad.

2. Marco teórico

2.1 Redes fijas

Una red fija es aquella en la que sus usuarios tienen una movilidad reducida o nula. El acceso se realiza siempre en el mismo edificio o zona. Cuando el acceso se realiza mediante un cable la movilidad esta reducida al espacio que permite dicho cable. Para las redes WiFi o Wimax la movilidad es algo mayor pero no permiten que el usuario esté en movimiento. En el caso de una red WiFi que cubra un edificio completo el usuario puede moverse libremente, pero lo normal es que la conexión de datos se corte al pasar de una estancia a otra. Por lo tanto, la característica fundamental es que la movilidad permitida sea reducida, aunque la conexión se realice de forma inalámbrica [5].

2.1.1 CPE

El equipo local del cliente o "Customer Premise Equipment" por sus siglas en inglés, es el router ADSL (Asymmetric Digital Subscriber Line) o FTTH (Fiber To The Home) instalado en el domicilio o en las instalaciones de una empresa. La responsabilidad del operador termina en el CPE y, a partir de este, se considera la red privada del cliente. El CPE normalmente realiza la función NAT (traducción de direcciones de red) para adaptar las direcciones IP privadas a la dirección IP que se utiliza para la conexión a internet. Esta función NAT, además, protege la red privada de ataques desde internet [5].

2.1.2 ADSL

La línea de abonado digital asimétrica, es la tecnología que permite la conexión a internet utilizando la línea telefónica clásica. Esta permite una velocidad de hasta 20 millones de bit por segundo (Mbit/s), aunque esto solo es posible en condiciones muy favorables. La velocidad máxima es muy dependiente de la longitud del cable que conecta la central con nuestra casa. A más distancia menos velocidad. Incluso en las grandes capitales es difícil pasar de 10 Mbit/s y en las afueras se puede tener 3 Mbit/s o menos, y esta velocidad es de la central a nuestra casa, porque la velocidad de nuestra casa a la central puede ser 10 veces menor. La conexión ADSL es punto a punto por lo que tenemos que poner unos filtros a nuestros teléfonos para que no afecten a la comunicación. Esta comunicación se establece entre un equipo en la central que se llama DSLAM (multiplexor de acceso de línea de abonado digital) y el router ADSL [5].

2.1.3 FTTH

La fibra hasta la casa, utiliza habitualmente la tecnología GPON (Gigabit-capable Passive Optical Network o Red Óptica Pasiva con capacidad de Gigabits). Esta tecnología utiliza la fibra óptica para la comunicación entre la central y la casa. Al ser fibra óptica su capacidad es elevada incluso con largas distancias entre la central y el usuario final. Las fibras ópticas de varios puntos terminales se agrupan en una para ir reduciendo el número de fibras ópticas que se deben desplegar. A su vez las fibras ópticas de varias zonas se agrupan también en 1 con el mismo objetivo, reducir el número

de fibras ópticas a desplegar. Esto equipos que agrupan fibras ópticas se denominan acopladores ópticos los cuales no necesitan alimentación con lo que pueden estar en cualquier lado. Por supuesto, la capacidad máxima es la de la fibra más cercana a la central y esta capacidad se reparte entre todos los abonados conectados. La velocidad habitual suele ser de 100 Mbit/s (de la central a nuestra casa) y 5 Mbit/s en el sentido opuesto aunque, como se ha indicado anteriormente, se podría llegar a 1 Gbit/s [5].

En los hogares la fibra termina en un modem llamado ONT (terminal de red óptica) que permite la conexión de cables de cobre menos delicados que el cable de fibra óptica. A este modem se conecta un router que, por un lado, a través de la red de fibra, se conecta con el equipo BRAS recibiendo de este la dirección IP pública que se utilizará en internet. Por otro lado, el router permite la conexión de distintos dispositivos conectados tanto por cable de cobre directamente al router como conexiones inalámbricas mediante una conexión Wifi. Otra conexión que permite el modem es la de teléfonos clásicos. El modem utiliza la tecnología de VoIP (Voz sobre IP) para conectarse con la central a través de la fibra simulando una línea telefónica clásica permitiendo la conexión de nuestros teléfonos de toda la vida o de un fax [5].

2.1.4 Cable

Esta tecnología está a medio camino entre el ADSL y el FTTH, y se deriva de la tecnología de televisión por cable o CATV (Cable TV). El funcionamiento de la tecnología CATV es muy similar a una antena colectiva. Un equipo denominado CMTS (sistema de terminación de cable módems) transmite la señal con mucha potencia a través de un cable coaxial que va de abonado en abonado. Cada abonado extrae una pequeña potencia de manera que no afecta a la señal que viaja por el cable. Todos los abonados escuchan todos los canales. Sobre la tecnología CATV se desarrolló la tecnología DOCSIS (especificación de interfaz para servicios de datos por cable). En ella se utilizaban los canales no utilizados para la televisión para transmitir datos. A cada abonado le correspondía uno de estos canales. También se establecía un mecanismo para el sentido contrario (del modem a la red) donde cada modem respondía en su canal correspondiente [5].

La tecnología DOCSIS es similar a FTTH porque termina en un modem denominado CM (cable modem) y necesita un router para establecer la conexión con el BRAS y para conectar nuestros dispositivos y ordenadores. A diferencia de FTTH el último tramo que llega al domicilio es un cable de cobre coaxial. Su capacidad es mayor que la tecnología ADSL sobre cable telefónico, pero no llega a la capacidad de la fibra óptica, aunque sigue compitiendo con ella. Con la última tecnología denominada DOCSIS 3.1 se alcanzan los 10 Gbit/s, velocidad totalmente comparable a la velocidad de la fibra, aunque la tecnología FTTH puede llegar a esta capacidad a distancias muy superiores gracias a la baja atenuación de la fibra [5].

2.1.5 WiFi

Es una tecnología inalámbrica que permite una movilidad en un ámbito de unas decenas de metros entorno a la antena. En el caso de grandes espacios se puede aumentar la cobertura con varias antenas, aunque el usuario debe estar conectado a una de ellas. Si el equipo necesita cambiar de antena por el movimiento del usuario la conexión normalmente se corta. La característica más importante de un equipo WiFi es la norma que cumple, todas se caracterizan por que empiezan por 802.11, la más lenta es la 802.11b que llega hasta 10 Mbit/s, la norma 802.11n podría llegar hasta 600 Mbit/s y la más moderna 802.11ac que alcanza hasta 1.3 Gbps. El organismo IEEE que es el encargado de estandarizar esta tecnología, y ha especificado desde la norma 802.11a hasta la 802.11af, aunque todavía no hay equipos comerciales de las últimas normas. Las bandas utilizadas son dos: la más antigua es la banda de 2.4 GHz y la más moderna de 5 GHz utilizada por las normas de mayor velocidad. Estas bandas son de libre disposición por lo que nadie la tiene reservada. Existen varios operadores que dan acceso a internet utilizando esta tecnología, pero, al no tener reservada la banda, no pueden garantizar ninguna calidad de servicio y están restringidos a edificios públicos y privados como aeropuertos, estaciones, hoteles, etc. Las razones para su uso a pesar de la mejor calidad de las redes móviles, suelen ser económicas (gratuita o precio inferior al roaming) [5].

2.1.6 WiMAX

Interoperabilidad mundial para acceso por microondas o "Worldwide Interoperability for Microwave Access" por sus siglas en inglés. Es una tecnología estandarizada por el organismo IEEE bajo la norma 802.16. A diferencia de la tecnología WiFi está pensada para una red pública pudiendo dar acceso a internet en una zona de 60 km de radio alrededor de la antena. Por lo tanto, debe utilizar frecuencias que hayan sido otorgadas por el gobierno a un operador, normalmente en la banda de 2,3 GHz o la banda de 3.5 GHz. Aunque las últimas versiones de la norma (802.16e) permiten la movilidad completa en la conexión; el uso más habitual es para proveer acceso internet en zonas donde es complicado utilizar otras tecnologías. Normalmente el abonado es un equipo con una antena situada en el tejado del edificio que distribuye la conexión a internet en el edificio mediante un cableado normal. Con esta configuración se puede dar una conexión de internet de hasta 20 Mbit/s en áreas de hasta 6 km alrededor de la antena. Al ser una tecnología usada donde no llegaban las demás no ha tenido el éxito que se esperaba. Debido a esto los operadores solo suelen cubrir las ciudades más importantes que son las únicas en las que se consigue suficiente rentabilidad [5].

2.2 Redes móviles

Son aquellas redes pensadas para que el teléfono o equipo del usuario pueda moverse con libertad en la zona cubierta por dicha red incluso mientras mantiene una conversación o una conexión de datos. Una red móvil debe permitir el movimiento

incluso a la velocidad de un coche sin que exista una pérdida de la conexión. Las redes móviles actuales permiten mantener esta conexión incluso a la velocidad de un tren de alta velocidad, con velocidades superiores a 300 km/h [6].

2.2.1 LTE

LTE es una tecnología inalámbrica de banda ancha, con la que se pueden transmitir datos a dispositivos móviles. Destaca sobre todo por tener una capacidad de subida y bajada de datos muy rápida. Según los cálculos, la rata de trasferencia puede alcanzar los 300 Mbps, que en la práctica supone poder descargar un archivo en apenas unos segundos [7].

La velocidad fue precisamente lo que motivó a desarrollar una nueva tecnología de comunicación, además, de lograr una reducción de costos. Por tanto, la necesidad de desarrollar una tecnología mejor que el 3G se debía a un aspecto, principalmente la competitividad, tanto en precio como en prestaciones. Así es como evolucionó el estándar 3G hacia esta tecnología [7].

2.2.2 4G

La tecnología 4G representa la cuarta generación de velocidades de descarga de datos, 4G es la cuarta generación inalámbrica móvil moderna, con alta velocidad de datos, en un principio podía alcanzar 100Mbps, 150Mbps y 300Mbps, pero actualmente es posible alcanzar 1000Mbps. Las redes móviles ofrecen una velocidad y eficiencia asombrosas y se apoderaron del mercado móvil. La red móvil 4G ofrece una alta velocidad, además tiene un mayor ancho de banda lo que se traduce en una velocidad de transferencia de datos mucho más rápida, lo que es especialmente ventajoso para los dispositivos móviles, los usuarios de la red 4G obtienen el beneficio de una conectividad superior e ininterrumpida, especialmente para las tareas avanzadas como video chats y conferencias [8].

Las redes 4G ofrecen una cobertura de 48 km y más, además permite la superposición de rangos de la red, de esta manera los usuarios pueden tener la seguridad de una conectividad completa en todo momento [8].

2.2.3 5G

La tecnología inalámbrica de quinta generación, o 5G, es la última versión de la tecnología celular, diseñada para aumentar considerablemente la velocidad y la capacidad de respuesta de las redes inalámbricas. Con 5G, los datos transmitidos a través de conexiones inalámbricas de banda ancha podrían viajar a velocidades de hasta 20 Gbps según algunas estimaciones, superando las velocidades de la red de telefonía fija, así como ofrecer una latencia de 1 ms o menos para usos que requieren retroalimentación en tiempo real. 5G también permitirá un fuerte aumento en la cantidad de datos transmitidos a través de sistemas inalámbricos debido al mayor ancho de banda disponible y la tecnología avanzada de antena. Además de las mejoras en la velocidad, la capacidad y la latencia, 5G ofrece funciones de administración de red, entre ellas la

división de la red, que permite a los operadores móviles crear múltiples redes virtuales dentro de una sola red física 5G [9].

Las redes y servicios 5G se implementarán en etapas durante los próximos años, para adaptarse a la creciente dependencia de los dispositivos móviles e Internet. En general, se espera que 5G genere una variedad de nuevas aplicaciones, usos y casos de negocios a medida que se despliegue la tecnología. Se espera que los servicios de banda ancha fijos hagan que a los operadores les resulte menos costoso entregar servicios de banda ancha a hogares y empresas porque este enfoque elimina la necesidad de desplegar líneas de fibra óptica en cada residencia. En su lugar, los operadores solo necesitan instalar fibra óptica en los sitios celulares, y los clientes reciben servicios de banda ancha a través de módems inalámbricos ubicados en sus residencias o negocios [9].

Tabla 2.1. Comparación de las generaciones de telefonía móvil [10]

	Tecnología				
Característica	**1G**	**2G**	**3G**	**4G**	**5G**
Comienzo/ Despliegue	1970-1980	1990-2004	2004-2010	Ahora	Pronto (Se espera para 2020)
Ancho de banda de los datos	2kbps	64kbps	2Mbps	1Gbps	> 1Gbps
Tecnología	Tecnología celular analógica	Tecnología celular digital	CDMA 2000 (1xRTT, EVDO) UMTS, EDGE	WiMax LTE WiFi	WWWW (llegando pronto)
Servicio	Telefonía móvil (voz)	Voz digital, SMS, datos empaquetados de mayor capacidad	Audio, video y datos integrados de alta calidad	Acceso dinámico a la información, dispositivos portátiles	Acceso dinámico a la información, Dispositivos portátiles con capacidades de IA
Multiplexación	FDMA	TDMA, CDMA	CDMA	CDMA	CDMA
Conmutación	Circuitos	Circuitos, paquetes	paquetes	Todo paquetes	Todo paquetes
Red de núcleo	PSTN	PSTN	Packet N/W	Internet	Internet

FDMA: Acceso múltiple por división de frecuencia

TDMA: Acceso múltiple por división de tiempo

CDMA: Acceso múltiple por división de código

PSTN: Red telefónica publica conmutada

2.2.4 Comparación de las generaciones de telefonía móvil

En la tabla 4.1 se puede observar una comparación de algunas de las características de las generaciones de telefonía móvil (1G, 2G, 3G, 4G y 5G).

2.3 Redes definidas por software (SDN)

Las redes definidas por software (SDN) son un paradigma emergente que permite romper la integración de los planos de control y de datos [11].

SDN ofrece una vista centralizada de la red, que le da a un controlador SDN la capacidad de actuar como el "cerebro" de la red. El controlador SDN transmite información a los conmutadores y enrutadores a través de las API hacia el sur, y a las aplicaciones con las API hacia el norte. Uno de los protocolos más conocidos utilizados por los controladores SDN es OpenFlow , sin embargo, no es el único estándar SDN [12].

Los entornos SDN centralizados y programables pueden adaptarse fácilmente a las necesidades cambiantes de las empresas. SDN puede reducir los costos y limitar el aprovisionamiento inútil, así como proporcionar flexibilidad e innovación para las redes [12].

Características de la arquitectura de una red SDN [13]:

- *Directamente programable:* El control de la red es directamente programable porque está desacoplado de las funciones de reenvío.
- *Agilidad:* Al abstraer el control del reenvío, los administradores pueden ajustar dinámicamente el flujo de tráfico de la red para satisfacer las necesidades cambiantes.
- *Gestión centralizada:* La inteligencia de la red está (lógicamente) centralizada en controladores SDN basados en software que mantienen una visión global de la red, que parece ser un conmutador lógico para las aplicaciones y los motores de políticas.
- *Configuración programable:* SDN permite a los administradores de red configurar, administrar, asegurar y optimizar los recursos de red muy rápidamente a través de programas SDN dinámicos y automatizados, que pueden ser escritos por los mismos administradores ya que los programas no dependen de software propietario.
- *Basado en estándares abiertos y proveedor neutral:* Cuando se implementa a través de estándares abiertos, SDN simplifica el diseño y la operación de la red porque las

instrucciones son proporcionadas por los controladores SDN en lugar de múltiples dispositivos y protocolos específicos del proveedor.

2.3.1 Arquitectura

En la

Figura 2.1 se muestra la arquitectura general de una red definida por software.

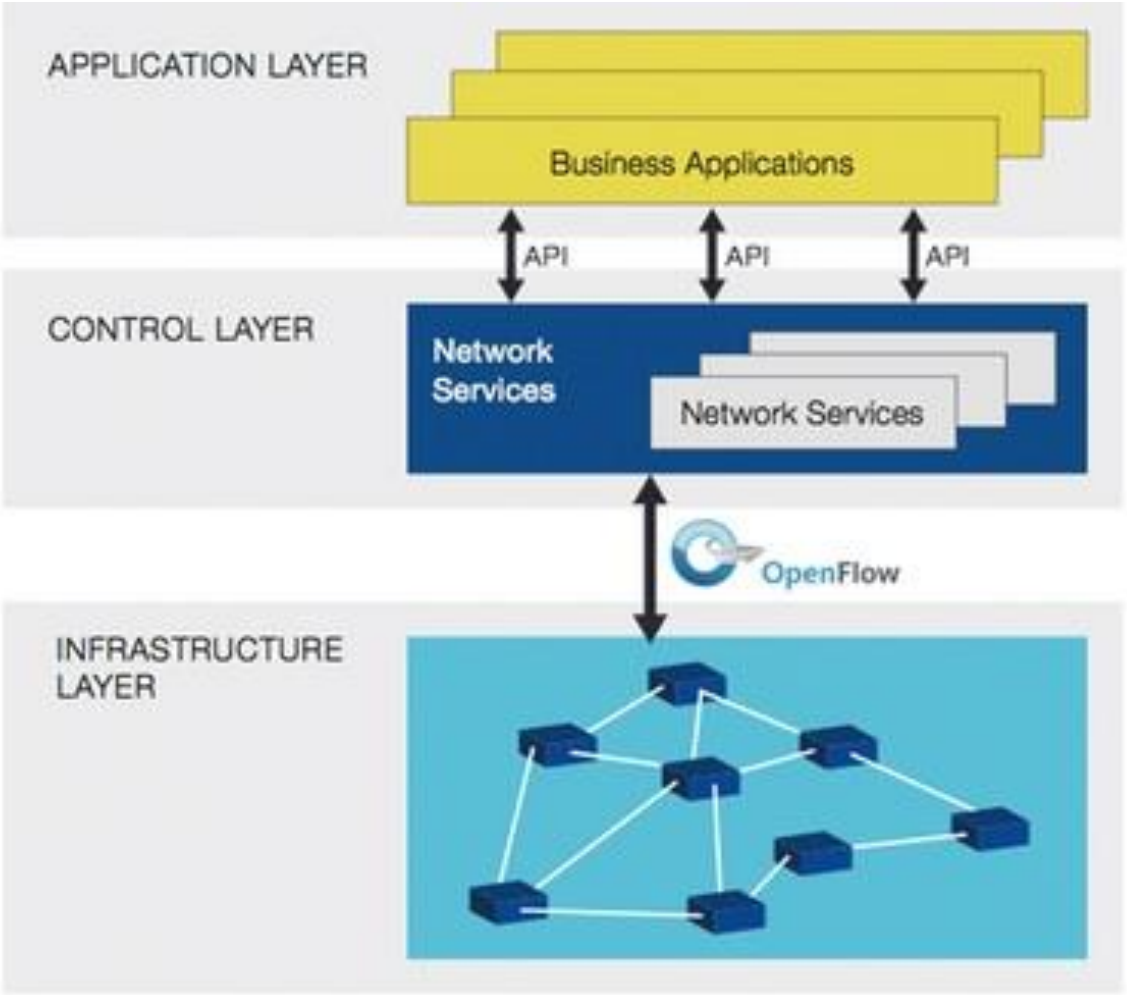


Figura 2.1. Arquitectura general – SDN [14]

Todas las redes definidas por software tienen alguna versión de un controlador SDN, así como las API hacia el sur y las API hacia el norte [14]:

- *Controladores:* Estos son los "cerebros" de la red, los controladores SDN ofrecen una vista centralizada de la red general y permiten a los administradores de red dictar a los sistemas subyacentes (como conmutadores y enrutadores), cómo el plano de reenvío debe manejar el tráfico de la red.
- *API hacia el sur:* Las redes definidas por software utilizan API's hacia el sur para transmitir información a los conmutadores y enrutadores "abajo".

OpenFlow, considerado el primer estándar en SDN, fue la API original hacia el sur y sigue siendo uno de los protocolos más comunes.

- *API hacia el norte:* Las redes definidas por software usan API's hacia el norte para comunicarse con las aplicaciones y la lógica de negocios "arriba". Estas ayudan a los administradores de red a configurar el tráfico y desplegar servicios mediante programación.

2.3.2 Protocolo OpenFlow

OpenFlow (OF) se considera uno de los primeros estándares de las redes definidas por software (SDN). Originalmente, definió el protocolo de comunicación en entornos SDN, que permite que el controlador SDN interactúe directamente con el plano de reenvío de los dispositivos de red, como conmutadores y enrutadores, tanto físicos como virtuales, para que pueda adaptarse mejor a los requisitos comerciales cambiantes [13].

Beneficios de OpenFlow [13]:

- Programabilidad
  - Habilitar la innovación/diferenciación
  - Acelera la introducción de nuevas funciones y servicios
- Inteligencia centralizada
  - Simplificar el aprovisionamiento
  - Optimizar el rendimiento
  - Gestión de políticas granulares
- Abstracción
  - Desacoplamiento de hardware y software, plano de control y reenvío, y configuración física y lógica

2.4 Controlador ONOS

Open Network Operating System (ONOS) o en español sistema operativo de red abierta es el controlador SDN de código abierto líder para la creación de soluciones SDN/NFV de próxima generación [15].

ONOS fue diseñado para satisfacer las necesidades de los operadores que desean construir soluciones a nivel de operador, al tiempo que ofrece la flexibilidad para crear e implementar nuevos servicios de red dinámicos con interfaces programáticas simplificadas. ONOS admite tanto la configuración como el control en tiempo real de la red, eliminando la necesidad de ejecutar protocolos de control de enrutamiento y conmutación dentro de la estructura de la red. Al trasladar la inteligencia al controlador en la nube de ONOS, se habilita la innovación y los usuarios finales pueden crear fácilmente nuevas aplicaciones de red sin la necesidad de alterar los sistemas de planos de datos [15].

Las características de ONOS se pueden sintetizar como [16]:

- *Alta disponibilidad y resiliencia:* Los proveedores de servicios requieren una alta disponibilidad para que los clientes no experimenten tiempo de inactividad de la red. ONOS fue diseñado desde el principio para soportar las redes de operadores más exigentes y tiene muchos mecanismos para garantizar que la red y sus conexiones sean confiables.
- *Rendimiento a escala:* ONOS ha sido diseñado y construido para proporcionar el mayor rendimiento posible para operaciones de red escaladas. Todos los lanzamientos están sujetos a este rendimiento, incluso al agregar muchas características nuevas. Admite millones de solicitudes de intentos de aplicaciones en su interfaz hacia el norte, mientras mantiene un tiempo de respuesta de menos de 50 ms (o mejor) para eventos de red, y ONOS escala según sea necesario, al agregar nuevas instancias cuando se necesita más capacidad del plano de control.
- *Software modular:* La modularidad del software en ONOS significa que la comunidad ha sido diligente en mantener las funciones del software bien definidas y localizadas al definir las abstracciones e interfaces correctas. Esto tiene muchos beneficios importantes: un software que es más fácil de leer, probar y mantener. Lo más importante es que permite a los socios personalizar más fácilmente el software. ONOS viene con más de 135 extensiones de plataforma, que incluyen aplicaciones de control de tráfico, aplicaciones de superposición de red, proveedores hacia el sur, modelos de YANG precompilados (incluyendo OpenConfig, Open ROADM), controladores de dispositivos y varias utilidades. Esta lista de extensiones disponibles sigue creciendo con cada lanzamiento de la plataforma.
- *Abstracciones hacia el norte:* Facilidad de programación de la red para automatización y control. ONOS proporciona abstracciones hacia el norte innovadoras que simplifican la creación, implementación y operación de las aplicaciones de configuración, administración y control. La vista de red global y el marco de intención de la aplicación son dos ejemplos. Las aplicaciones se pueden agregar fácilmente para ejecutar "en la caja" usando interfaces nativas, o "fuera de la caja" usando las interfaces REST y/o gRPC.
- *Abstracciones hacia el sur:* Fácil adaptación a dispositivos nuevos o heredados (arquitectura de plug-in). ONOS resume las características del dispositivo para que el sistema operativo central no tenga que estar al tanto del protocolo particular que se utiliza para controlar o configurar un dispositivo. ONOS tiene una lista extensa y creciente de soporte hacia el sur que incluye P4, OpenFlow, NETCONF, TL1, SNMP, CLI, BGP, RESTCONF y más.
- *Interfaz gráfica de usuario (GUI):* La interfaz gráfica de usuario de ONOS proporciona la vista de la red de múltiples capas y permite que el usuario explore elementos de la red, conectividad, estado de la red, errores de red y más.
- *Cadena de herramientas YANG:* La cadena de herramientas YANG de ONOS proporciona un compilador capaz de analizar archivos fuente de YANG y generar artefactos Java, que pueden usarse para escribir aplicaciones contra las

abstracciones definidas por los modelos de YANG. También proporciona un tiempo de ejecución capaz de codificar y decodificar entre dichos modelos internos y sus representaciones de datos JSON o XML externos. Juntos, el compilador y el tiempo de ejecución pueden ser utilizados por las aplicaciones de ONOS para interactuar con varios elementos de red que admiten interacciones de configuración modeladas a través de YANG. La cadena de herramientas también admite la compilación sobre la marcha de modelos YANG, lo que le permite a la plataforma ampliar dinámicamente sus capacidades de configuración.

2.5 Mininet

Mininet es un emulador de red que crea una red de hosts virtuales, conmutadores, controladores y enlaces. Los hosts de Mininet ejecutan el software de red estándar de Linux, y sus conmutadores son compatibles con OpenFlow para un enrutamiento personalizado altamente flexible y con redes definidas por software [17].

Dentro de las principales características de Mininet se tiene [17]:

- Proporciona un banco de pruebas de red simple y económica para desarrollar aplicaciones OpenFlow.
- Permite que múltiples desarrolladores concurrentes trabajen independientemente en la misma topología.
- Admite pruebas de regresión a nivel del sistema, que son repetibles y se empaquetan fácilmente.
- Permite realizar pruebas de topología complejas, sin la necesidad de conectar una red física.
- Incluye un CLI que es compatible con topología y OpenFlow, para depurar o ejecutar pruebas en toda la red.
- Admite topologías personalizadas arbitrarias e incluye un conjunto básico de topologías parametrizadas.
- También ofrece una API de Python directa y extensible para la creación y experimentación de redes.
- Proporciona una manera fácil de obtener el comportamiento correcto del sistema (y, en la medida en que lo admita su hardware, rendimiento) y experimentar con topologías.

2.5.1 Ventajas

Mininet combina muchas de las mejores características de emuladores, bancos de pruebas de hardware y simuladores.

En comparación con los enfoques basados en la virtualización de sistemas completos, Mininet [17]:

- *Boots (arranques) más rápidos:* Segundos en lugar de minutos.
- *Escalas más grandes:* cientos de hosts y switches frente a dígitos individuales.
- *Proporciona más ancho de banda:* Normalmente un ancho de banda total de 2 Gbps en hardware Modesto.
- *Se instala fácilmente:* Hay una VM preempaquetada disponible que se ejecuta en VMware o VirtualBox para Mac/Win/Linux con las herramientas OpenFlow v1.0 ya instaladas.

Comparado con los bancos de pruebas de hardware, Mininet [17]:

- Es barato y siempre está disponible
- Es rápidamente reconfigurable y reiniciable

Comparado a los simuladores, Mininet [17]:

- Ejecuta código real, sin modificar, incluido el código de la aplicación, el código de kernel del sistema operativo y el código del plano de control (tanto el código del controlador OpenFlow como el código Open vSwitch)
- Se conecta fácilmente a redes reales

2.5.2 Limitaciones

Las redes basadas en Mininet no pueden (actualmente) exceder la CPU o el ancho de banda disponible en un solo servidor. Mininet no puede (actualmente) ejecutar switches o aplicaciones OpenFlow no compatibles con Linux; Este no ha sido un problema importante en la práctica [17].

2.6 Parámetros en la calidad de servicio (QoS)

La calidad de servicio (QoS), se puede entender como la medida del comportamiento de la capacidad de la red con respecto a ciertas características de los servicios definidos, o como la capacidad de una red para proveer mejor servicio para un determinado tipo de tráfico. Los parámetros relacionados con QoS son: ancho de banda, nivel de retardo, variación del retardo (jitter), rendimiento (throughput) y pérdida de paquetes. Basándose en las políticas de QoS, la red debe garantizar que tiene la capacidad de ofrecer un nivel de calidad de servicio para cierto tráfico con un conjunto especifico de parámetros [18].

2.6.1 Rendimiento (throughput)

Este parámetro especifica cuantos datos (máximo o media) son transferidos a través de la red. Esta medida se da en referencia a un tiempo determinado con unidades de bit/s o bps. El throughput es medido después de la transmisión de datos porque un sistema añade retardo causado por limitaciones del procesador, congestión de la red, ineficiencias del proceso de almacenamiento de datos en los buffers, transmisión de bits errados, carga de tráfico o hardware inadecuado. El throughput varía con el tiempo durante la transmisión de datos debido al tráfico y la congestión [18].

2.6.2 Retardo (delay)

El retardo en transmisión o delay es usualmente definido como el tiempo que transcurre entre la emisión de los datos, hasta el momento en que llegan al receptor. El retardo es una medida que expresa el tiempo gastado en el subsistema de comunicación. Este parámetro es también conocido en el ámbito de las telecomunicaciones como latencia (latency). La causa de este retardo es que cuando la data es procesada esta fluye a través de una gran cantidad de componentes y subsistemas de comunicación, situados en el sistema receptor, así como en la red. Cada uno de esos componentes pueden ser caracterizados por su velocidad de procesamiento y por la capacidad de almacenamiento (buffers) donde los datos esperan para ser procesados. La suma de todas las contribuciones de retardos individuales vista como un todo es lo que genera el parámetro reconocido como retardo. El máximo retardo que es el que ocurre de extremo a extremo conocido como mouth-to-ear (de boca a oído) recomendado para conversaciones en tiempo real no debe exceder los 150 ms. Este retardo se calcula como el promedio de los retardos paquete a paquete [18].

$$\frac{\sum_{i=1}^{n} x_i}{n} \tag{2.1}$$

x_i: Retardo del i-ésimo paquete
n: Cantidad de paquetes

2.6.3 Variabilidad (jitter)

El retardo por jitter o simplemente jitter es usualmente definida como la variación en el tiempo de llegada al punto receptor, sufrida entre paquetes de datos sucesivos [18].

$$\frac{\sqrt{\sum_{i=1}^{n} {x_i}^2 - n * D^2}}{n-1} \tag{2.2}$$

x_i: Retardo del i-ésimo paquete
n: Cantidad de paquetes
D: Delay

2.6.4 Tasa de paquetes perdidos

La pérdida de paquetes ocurre cuando uno o más paquetes de datos que viajan en una red IP fallan en alcanzar su destino. La causa de la pérdida de paquetes es debido a varios factores, entre los que se puede nombrar: degradación de la señal al viajar por el medio, interconexiones de la red sobresaturadas, paquetes con error rechazados en el tránsito, falla en el hardware de la red o rutinas normales de enrutamiento. Cuando la pérdida de paquetes es causada por problemas en la red, los paquetes perdidos pueden resultar en problemas de desempeño que causen fallas notables en el desempeño de esta, sin embargo, la pérdida de paquetes no siempre es tan dañina, como por ejemplo cuando es usada para contrarrestar la latencia [18].

2.7 Volcado del tráfico de datos en redes móviles

Para admitir un servicio de datos adecuado, se necesitan muchas más estaciones base celulares (puntos de acceso). El desafío es obviamente aún mayor en interiores, ya que las señales de radio deben penetrar en los edificios para proporcionar cobertura de datos en interiores. Dado que el costo para el negocio de los operadores está linealmente relacionado con la densidad de despliegue de la estación base, los servicios de datos representan un gran desafío financiero para los operadores, además de los obstáculos logísticos [19].

2.7.1 WiFi offload

WiFi offload o descarga WiFi se refiere al uso de redes complementarias para entregar los datos originalmente destinados a las redes celulares [20]. La descarga de WiFi es una técnica comúnmente propuesta para reducir la carga de tráfico en redes celulares 3G/4G/5G. El proceso de descarga normalmente se basa en la infraestructura, es decir, el tráfico descargado se enruta a través de un punto de acceso WiFi [21].

La inmensa mayoría de los teléfonos móviles permite la conexión a redes WiFi. Para descargar la red móvil, el operador instalaría una serie de puntos WiFi en lugares especialmente sobrecargados como aeropuertos o estaciones [6].

El teléfono cambiaría automáticamente de la red móvil a la red WiFi descargando la red móvil e incluso aumentando la velocidad de conexión ya que las redes WiFi tienen mayor velocidad [6].

3. Antecedentes

El constante aumento en la complejidad de las redes en todo el mundo, sumado a la necesidad de administrar y/o controlar estas redes de manera sencilla y eficiente, lleva a plantear nuevos mecanismos para proveer una implementación de manera dinámica y escalable.

Las redes heredadas se han vuelto difíciles de automatizar, estas dependen de las direcciones IP para identificar y localizar servidores y aplicaciones. Este enfoque funciona bien para redes estáticas donde cada dispositivo físico es reconocible por una dirección IP, pero es extremadamente laborioso para redes virtuales grandes. La gestión de estos entornos complejos con redes tradicionales requiere mucho tiempo y es costosa, especialmente en el caso de la migración de máquinas virtuales (VM) y la configuración de la red. Para simplificar la tarea de administrar grandes redes virtualizadas, los administradores deben resolver los problemas de infraestructura física que aumentan la complejidad de la administración [22].

Las redes definidas por software (SDN) están cambiando la forma en que se diseñan y gestionan las redes. SDN tiene dos definiciones características, primero una SDN separa el control (que decide cómo manejar el tráfico) del plano de datos (que reenvía el tráfico de acuerdo con las decisiones que hace el plano de control), en segundo lugar, una SDN consolida el plano de control, de modo que un solo programa de software controle múltiples elementos del plano de datos [23].

3.1 Configuración de redes basadas en software

En 2007, un grupo de trabajo de la Universidad de Standford formado por los profesores Nick McKeown, Scott Shenker y el estudiante de doctorado Martín Casado desarrollaron OpenFlow y fundaron Nicira, una compañía de virtualización de redes. Es a partir de este momento donde se sitúa el primer hecho relevante para el nacimiento de SDN, muchas implementaciones de SDN se basan en un estándar abierto OpenFlow. Es OpenFlow el que le brinda al administrador de la red los medios para controlar remotamente las tablas de enrutamiento y conmutación [24].

SDN a través de OpenFlow revoluciona y amplía las capacidades de la arquitectura de red y proporciona beneficios clave para el centro de datos en constante cambio. Con la rápida innovación y experimentación ahora posibles a través del control de software, OpenFlow ofrece la flexibilidad para combatir los problemas actuales y futuros de la red. No solo hay más opciones con respecto a las nuevas aplicaciones, sino que también hay más mercados de proveedores disponibles. La evolución de una arquitectura de red basada en hardware a una basada en software abre opciones para que los operadores de red seleccionen diferentes proveedores para el control y los planos de datos [24].

3.2 Virtualización de redes

El desarrollo o concepción de redes virtuales se ha venido dando desde antes de OpenFlow, sus inicios se remontan a poco antes de 1995, pero con avances poco significativos; solo hasta finales del 2009 se tuvo un avance relevante, con la implementación del puente de red para proporcionar conectividad a las máquinas virtuales. Este componente de red se llama Virtual Ethernet Bridge (VEB) o conmutador virtual (vSwitch) [25]; En el 2011, Scott Shenker y Nick McKeown fundaron la Open Networking Foundation (ONF), organización que buscaba fomentar el uso de OpenFlow, la creación de estándares y la implantación de SDN más allá de las universidades, en la Figura 3.1 podemos apreciar una línea de tiempo con los avances que ha tenido la virtualización de redes.

La virtualización es dar a un programa la ilusión de que este tiene su propio hardware. La máquina virtual depende de esta tecnología para ejecutar programas de aplicación, como una máquina real. La tecnología clave para el surgimiento de la computación en la nube es la virtualización. Actualmente, la mayoría de los entornos de virtualización en una nube IDC (International Data Corporation) utilizan conmutadores virtuales basados en software o conmutadores virtuales basados en hardware. Por otro lado, debido a la aparición del servicio de computación en la nube, la cantidad de conmutadores virtuales comienza a expandirse de manera dramática [26].

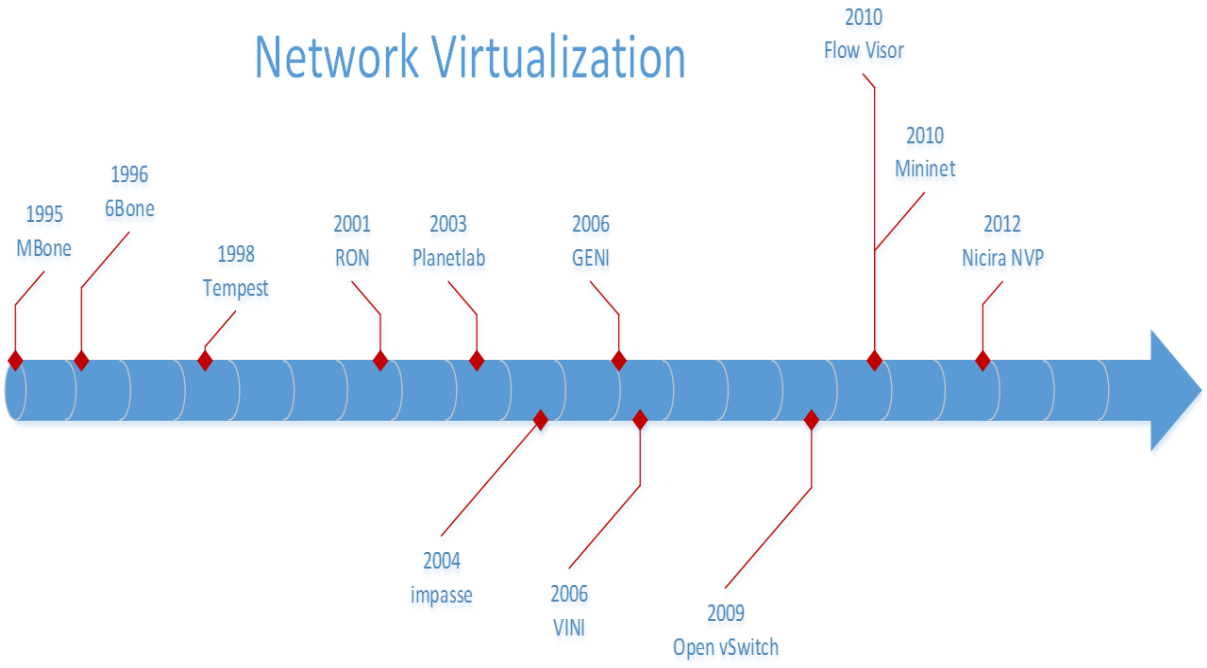


Figura 3.1. Cronología redes virtuales
(Fuente: https://agema.deltaww.com/blog_con.php?id=87)

3.3 Aplicaciones SDN

3.3.1 SND e IoT

Lo interesante de SDN es la división del plano de control del plano de datos, ya que facilita la modificación de los protocolos de red sin tener que ir hasta los dispositivos en el sitio. Por otro lado, el concepto de Smart Cities (ciudades inteligentes) se ha acuñado recientemente, donde se conectará una gran cantidad de dispositivos inteligentes y se brindarán toneladas de servicios a los ciudadanos, funcionarios y departamentos gubernamentales. Aquí el Internet de las cosas (IoT) desempeñará un papel vital en la garantía de dichos servicios. Se han realizado pocos esfuerzos para fusionar SDN e IoT, estos únicamente se han centrado en la recuperación de datos de una manera eficiente y en lograr redes configurables de forma remota [27].

Para realizar dicha fusión entre SDN e IoT se desarrolló una arquitectura que consiste en la recopilación y la gestión de datos. Los datos de varios dispositivos integrados habilitados para IoT se recopilan y envían al nivel de procesamiento de datos utilizando la red habilitada para SDN. Además, los datos se pasan a los usuarios respectivos después de tratar el nivel de procesamiento de datos [27].

3.3.2 Ataques dinámicos y SDN

Con la masiva llegada de los dispositivos inteligentes y la reducción en los precios de los dispositivos de detección (sensores), la adopción de IoT está ganando impulso. Pero estos dispositivos IoT, tienen una mayor amenaza de ser atacados o comprometidos, lo que podría llevar a la Denegación de Servicio (DoS) y la Denegación de Servicio Distribuida (DDoS). Además el alto volumen de dispositivos IoT con un alto nivel de heterogeneidad, aumenta la posibilidad de amenazas a la seguridad [28].

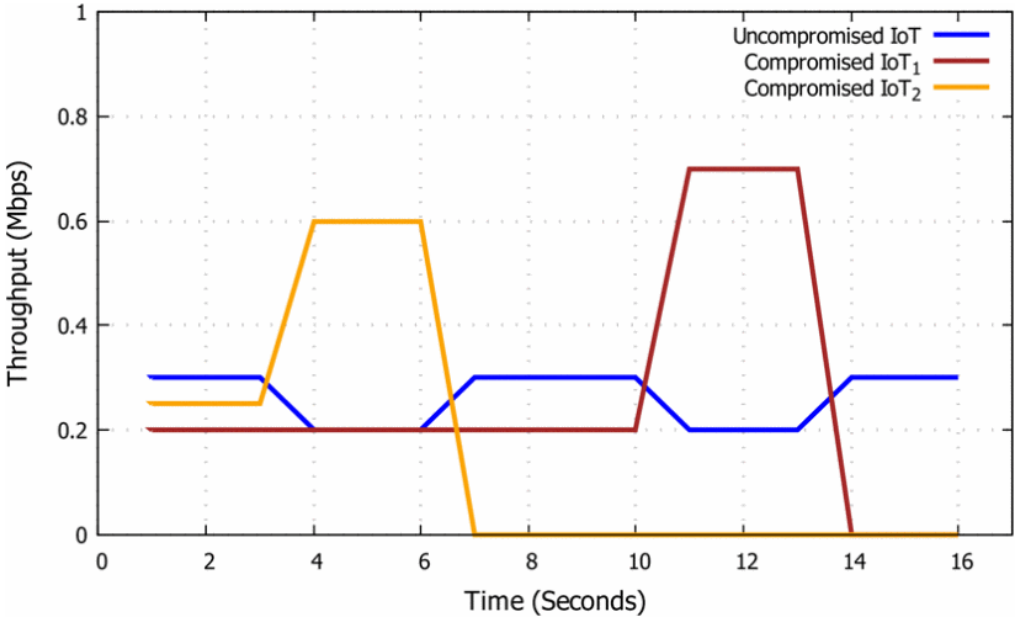


Figura 3.2. Simulación ataque DDos [28].

Estos desafíos pueden abordarse con la ayuda de las redes definidas por software, ya que estas pueden manejar efectivamente las amenazas de seguridad para los dispositivos de IoT de manera dinámica y adaptable sin ninguna carga para los dispositivos de IoT. En [28], se propone un marco de IoT seguro basado en SDN, llamado SoftThings para detectar comportamientos anormales y ataques tan pronto como sea posible y mitigarlos según corresponda; Machine Learning se utiliza en el controlador SDN para monitorear y conocer el comportamiento de los dispositivos IoT a lo largo del tiempo, en la Figura 3.2 podemos apreciar los resultados obtenidos al simular una ataque DDoS y los tiempos de recuperación del sistema, el parámetro monitoreado fue la capacidad de canal en Mbps, se observa que aproximadamente en 2 segundos el sistema logró recuperarse.

3.3.3 Calidad del servicio (QoS) y SDN

La mayoría de las aplicaciones que se desarrollan en las redes actuales tienen que ver con IoT, y estas sirven para mejorar la calidad de vida en diferentes aspectos de la sociedad. Estas aplicaciones requieren garantías en términos de calidad de servicio (QoS) dentro de la red, para realizar sus tareas de forma eficiente y satisfactoria, cuando se producen eventos que implican la intervención de un controlador, la dificultad en el desarrollo de dichas aplicaciones reside en el hecho, de que cada dispositivo de red en Internet tiene su propio software de control para tomar las decisiones de reenvío, careciendo de una perspectiva más amplia de los recursos disponibles en su red. Es por esto que las arquitecturas actuales de QoS no son realmente exitosas y no se implementan globalmente a través de Internet [29].

Con la tecnología de redes definidas por software, es posible resolver los problemas anteriores, ya que sustituyen el software de todos los dispositivos en una red y lo integra en uno o más servidores externos (controladores), para de esta manera proporcionar una estandarización en las interfaces, dotando a los controladores de una perspectiva amplia de la red y su desarrollo en el tiempo [29].

Los últimos avances en QoS se han implementado en la capa de control, dados los hechos mencionados anteriormente, es esta la responsable de administrar el plano de datos y de proporcionar aplicaciones de red externa con una API para interactuar con la red. Esta funcionalidad es implementada por el controlador SDN. La capa de administración consta del componente de aplicación IoT-QoS, que: (1) detecta nuevos dispositivos que se unen a la red, (2) permite a los usuarios especificar tipos para sus dispositivos, que determinan los perfiles de QoS asignables, (3) calcula las rutas de QoS entre pares de dispositivos, (4) reconfigura la red dinámicamente para garantizar la QoS y, (5) permite la especificación de reglas dinámicas de QoS, aplicando niveles de QoS específicos solo cuando se desencadenan eventos. En la figura 5.3 podemos apreciar varios de los perfiles más utilizados en términos de QoS. Los eventos que activan la QoS dinámica están vinculados a un dispositivo que no es IP y se activan cuando ese dispositivo coincide con los criterios especificados por el usuario [30].

Tabla 3.1. Perfiles de QoS predeterminados [30]

	Nivel QoS	Ancho de banda (Mbps)	Latencia (ms)	Paquetes perdidos (%)
Video	Bajo	2,8	15	2,5
	Medio	4,8	10	2
	Alto	7,2	5	1
Sensor	Bajo	n.a	500	2
	Medio	n.a	300	1
	Alto	n.a	150	0
Actuador	Bajo	n.a	1000	2
	Medio	n.a	1000	1
	Alto	n.a	100	0
Audio	Bajo	0,064	30	25
	Medio	0,248	10	13
	Alto	1,4	3	1

3.4 Desafíos de SDN y soluciones existentes

En general, SDN tiene un controlador lógicamente centralizado que toma decisiones de gestión y control de red (por ejemplo, encontrar rutas basadas en la topología actual y los requisitos de tráfico dados). Luego, el controlador instala las reglas de reenvío necesarias en los interruptores de reenvío simples, en el plano de datos a través de un protocolo abierto estándar (por ejemplo, OpenFlow) [31].

Uno de los desafíos clave en SDN, es cómo actualizar el plano de datos para una nueva ruta rápidamente sin violar la consistencia de la red subyacente además de la actualización precisa y consistente de múltiples flujos, especialmente en la presencia de entradas múltiples [31].

A medida que se ha avanzado, los esfuerzos se han centrado en cómo proporcionar estabilidad en el rendimiento, mediante el diseño de nuevos algoritmos de enrutamiento que puedan capturar de manera eficiente la información del estado de la red y utilizar esta información para determinar rutas óptimas para los flujos que necesitan ser cambiados por nuevos caminos [31].

3.5 Volcado del tráfico de datos en SDN

3.5.1 Balance de carga

A medida que IoT se vuelve común en la vida diaria, es posible recopilar información en cualquier momento y en cualquier entorno. Sin embargo, es probable que el uso de una gran cantidad de equipos terminales genere enormes demandas de ancho de banda,

seguido del aumento en el tiempo de transmisión de datos. Para abordar este problema, en [32] se propuso un mecanismo de equilibrio de carga SDN-SFC orientado a servicios, el cual consideró y clasificó el tipo y la prioridad del servicio requerido por cada dispositivo terminal. Luego, adoptó el algoritmo heurístico para planificar las rutas de transmisión entre las SFC (Service Function Chain) para reducir la carga de cada SF y mejorar el rendimiento general de la red.

Para conseguir un mecanismo de equilibro de carga, finalmente se presentó el diseño de reglas con el método del algoritmo de reconocimiento codicioso. En general, la estrategia más común para establecer la ruta de transmisión de datos es el algoritmo codicioso que procesa los objetivos más cercanos primero. Este consideró la distancia e introdujo el concepto de orientación de servicio, para proponer un algoritmo codicioso de orientación basado en el servicio (GSOA, por sus siglas en inglés). En la Figura 3.3 es posible apreciar el Algoritmo [32].

Greedy-based Service-orientation Algorithm (GSOA)

Input: $UserBW_Request_q$, $UserSF_Request_q$
Output: sfc_i
1. **For each** $UserBW_Request_q$
2. **If** ($SF_BWoverload_j > 0$ && $StatusPacket_z = 1$)
3. **For each** sf_j
4. **If** SF_Type_j meet the service requests
5. **If** SF_j has maximum $SF_BWSurplus_j$
6. Replace the current choice;
7. Update the $sf_s_{(i,j)}$ and $sf_o_{(i,j)}$;
8. **End If**
9. **End If**
10. **End For**
11. **Else**
12. **For each** sf_j
13. **If** SF_Type_j meet the request
14. **If** SF_j is closer $User_i$ than SF_{best}
15. Replace the current choice;
16. Update the $sf_s_{(i,j)}$ and $sf_o_{(i,j)}$;
17. **End If**
18. **End If**
19. **End For**
20. **End If**
21. **End For**

Figura 3.3. Algoritmo (GSOA) [32].

3.5.2 Descarga de datos con balanceo de carga

El crecimiento explosivo del tráfico de datos móviles y la escasez de los recursos espectrales disponibles han creado nuevos desafíos para las futuras redes celulares. En particular, la gestión de recursos en un entorno de red heterogéneo se ha convertido en un problema crítico. En el entorno de red celular 3G/3.5G/4G, es muy fácil congestionarse en el momento pico [33].

La descarga de datos se considera una solución habilitadora para abordar las preocupaciones de escasez de espectro. La descarga se refiere al uso de redes complementarias para entregar los datos originalmente destinados a las redes celulares. El 55% del uso de datos ocurre en el hogar y el 26% ocurre en la oficina o puntos de acceso. Por lo tanto, los puntos de acceso (AP) Wi-Fi ya implementados se convierten en una solución natural para que los operadores ejecuten la descarga de datos [20].

Por otro lado, para lidiar con la congestión de la red, el balance de carga se utiliza para distribuir uniformemente el tráfico a través de múltiples celdas. Toda la toma de decisiones se realiza mediante el controlador SDN centralizado, la introducción de SDN facilita la coordinación eficiente entre las redes celulares y Wi-Fi. En [20] se presenta un algoritmo de descarga parcial de datos basado en SDN. En este caso, la descarga parcial se refiere al hecho de que solo una parte de los datos del usuario se descarga en la red Wi-Fi, mientras que el tráfico restante se transfiere a través de la red celular, En la Figura 3.4 es posible apreciar el Algoritmo de descarga parcial.

Algorithm 1 Partial data offloading

1: **procedure** PDO(T_s)
2: T_s: delay threshold
3: d: Total file size
4: $V_{s2} = b_2 \min(t_r, T_s)$: size in bytes to be transferred in Wi-Fi within T_s
5: **if** $d < V_{s2}$ **then**
6: offload all data to Wi-Fi and update d
7: **else**
8: **if** $d_1 \leq V_{s2}$ and $d = d_1 + d_2$ **then**
9: send d_1 on Wi-Fi, d_2 on cellular concurrently and update d
10: **end if**
11: **end if**
12: **end procedure**

Figura 3.4. Algoritmo de descarga parcial de datos [20]

4. Mininet y ONOS

4.1 Emulador de redes Mininet

Antes de realizar la instalación del emulador "Mininet", se recomienda realizar una actualización a la versión más reciente de todos los paquetes haciendo uso de los comandos "sudo apt-get update" y posteriormente "sudo apt-get upgrade".

Para realizar la instalación del emulador es necesario tener instalado previamente el software git, para instalar este software se hace uso del comando "sudo apt-get install git" como se muestra en la Figura 4.1. Luego de esto se debe obtener una copia del código fuente de Mininet con el comando "git clone git://github.com/mininet/mininet", como se muestra en la Figura 4.2.



Figura 4.1. Comando de instalación de git

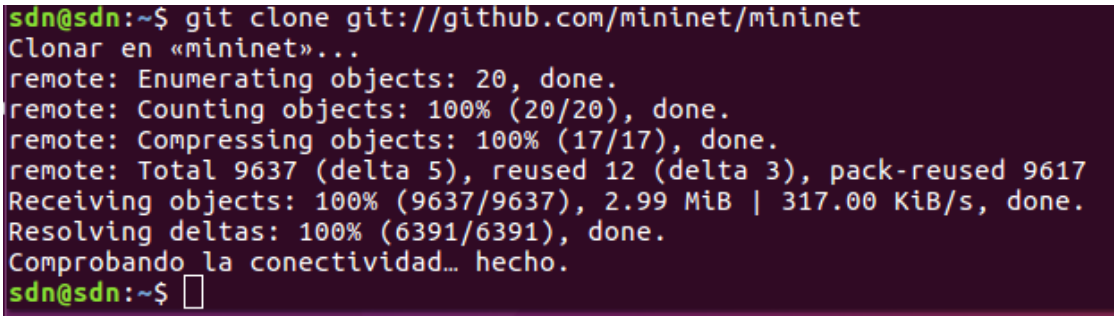


Figura 4.2. Comando para obtener copia del código fuente de mininet

Al realizar el paso anterior se habrá clonado el código fuente de Mininet en una carpeta con el nombre "mininet", en esta se encuentra otra carpeta de nombre "util" donde estará un archivo llamado "install.sh" que permite instalar Mininet. Para instalar todos los componentes que ofrece Mininet, se hace uso del siguiente comando "mininet/util/install.sh -a" como se muestra en la Figura 4.3.

```
sdn@sdn:~$ mininet/util/install.sh -a
Detected Linux distribution: Ubuntu 16.04 xenial amd64
python is version 2
Installing all packages except for -eix (doxypy, ivs, nox-classic)...
Install Mininet-compatible kernel if necessary
Des:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109
 kB]
Obj:2 http://co.archive.ubuntu.com/ubuntu xenial InRelease
Obj:3 http://co.archive.ubuntu.com/ubuntu xenial-updates InRelease
Obj:4 http://co.archive.ubuntu.com/ubuntu xenial-backports InRelease
Descargados 109 kB en 4s (22,0 kB/s)
Leyendo lista de paquetes... Hecho
Leyendo lista de paquetes...
Creando árbol de dependencias...
Leyendo la información de estado...
```

Figura 4.3. Comando de instalación del emulador mininet

Para comprobar la correcta instalación del emulador es posible realizar una prueba creando una topología por defecto y verificando conexión entre los diferentes hosts por medio de ping's, para ello se utiliza el comando "sudo mn --test pingall". El resultado luego de ejecutar el comando se muestra en la Figura 4.4.

```
sdn@sdn:~$ sudo mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 6.080 seconds
```

Figura 4.4. Comando para comprobar la correcta instalación de Mininet

4.2 Controlador ONOS

El controlador ONOS tiene unos requerimientos de software que se deben instalar previamente:

- Java8

  Para la instalación de este paquete se deben ejecutar los siguientes comandos:

  > sudo apt-get install openjdk-8-jre-headless -y && \
  > sudo apt-get update -y && \
  > sudo apt-get install openjdk-8-jdk-headless -y

  Esto también lo podemos observar en la Figura 4.5.

```
sdn@sdn:~$ sudo apt-get install openjdk-8-jre-headless -y && \
> sudo apt-get update -y && \
> sudo apt-get install openjdk-8-jdk-headless -y
Leyendo lista de paquetes... Hecho
```

Figura 4.5. Comandos de instalación de Java8

- Curl

  Para la instalación de este paquete solo basta con ejecutar el comando "sudo apt-get install curl", como se muestra en la Figura 4.6.

```
sdn@sdn:~$ sudo apt-get install curl
[sudo] password for sdn:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  curl
```

Figura 4.6. Comando de instalación de curl

Una vez instalados estos paquetes, se procede a descargar el archivo "onos-2.1.0.tar.gz", para esto se hace uso del comando "wget -c https://repo1.maven.org/maven2/org/onosproject/onos-releases/2.1.0/onos-2.1.0.tar.gz" como se puede observar en la Figura 4.7.

```
sdn@sdn:~$ wget -c https://repo1.maven.org/maven2/org/onosproject/onos
-releases/2.1.0/onos-2.1.0.tar.gz
--2019-09-12 19:23:55--  https://repo1.maven.org/maven2/org/onosprojec
t/onos-releases/2.1.0/onos-2.1.0.tar.gz
Resolviendo repo1.maven.org (repo1.maven.org)... 151.101.204.209
Conectando con repo1.maven.org (repo1.maven.org)[151.101.204.209]:443.
.. conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 373395382 (356M) [application/x-gzip]
Grabando a: "onos-2.1.0.tar.gz"
```

Figura 4.7. Comando para descargar el archivo "onos-2.1.0.tar.gz"

Luego se ejecutan los comandos “tar xzf onos-2.1.0.tar.gz” y “mv onos-2.1.0 onos” como se muestra en la Figura 4.8, los cuales crearán una carpeta con el nombre “onos”, con lo necesario para hacer uso del controlador.

```
sdn@sdn:~$ tar xzf onos-2.1.0.tar.gz
sdn@sdn:~$ mv onos-2.1.0 onos
```

Figura 4.8. Comandos para extraer el archivo “onos-2.1.0.tar.gz” y cambiar el nombre de la carpeta por “onos”.

Para poder acceder e iniciar la interfaz de línea comandos del controlador (CLI), es necesario modificar el archivo “onos-service” ubicado en el directorio “onos/bin”, quitando “service” en la sentencia “KARAF_ARGS=service”, como se muestra en la Figura 4.9.

ANTES

```
# If ONOS_HOME is set, respect its value.
# If ONOS_HOME is not set (e.g. in the init or service environment),
# set it based on this script's path.
ONOS_HOME=${ONOS_HOME:-$(cd $(dirname $0)/.. >/dev/null 2>&1 && pwd)}
KARAF_ARGS=service
SYS_APPS=drivers
ONOS_APPS=${ONOS_APPS:-}  # Empty means don't activate any new apps

cd $ONOS_HOME
```

DESPUÉS

```
# If ONOS_HOME is set, respect its value.
# If ONOS_HOME is not set (e.g. in the init or service environment),
# set it based on this script's path.
ONOS_HOME=${ONOS_HOME:-$(cd $(dirname $0)/.. >/dev/null 2>&1 && pwd)}
KARAF_ARGS=
SYS_APPS=drivers
ONOS_APPS=${ONOS_APPS:-}  # Empty means don't activate any new apps

cd $ONOS_HOME
```

Figura 4.9. Modificación del archivo “onos-service”

Para iniciar onos se hace uso del comando “onos/bin/onos-service start” y como resultado, en la CLI se imprimirá “onos” tal como se muestra en la Figura 4.10.

```
Welcome to Open Network Operating System (ONOS)!
    ____  _  ______  ____
   / __ \/ |/ / __ \/ __/
  / /_/ /    / /_/ /\ \
  \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.
```

Figura 4.10. Inicio de ONOS en la CLI

Para acceder a la interfaz gráfica (GUI) de “ONOS”, se debe abrir el navegador y acceder a la dirección URL “localhost:8181/onos/ui/login.html”, de esta manera se obtiene la GUI de ONOS como se muestra en la Figura 4.11.



Figura 4.11. Inicio de la GUI de ONOS

En un futuro “ONOS” pretende personalizar las sesiones de usuario, es por esta razón que exige un usuario y contraseña, actualmente no se dispone de este servicio y por defecto el usuario es “onos” y la contraseña es “rocks”.

4.3 Conectar mininet con ONOS

Una vez ejecutado “onos/bin/onos-service start”, se debe activar el protocolo openflow (en la GUI de ONOS es posible ingresar al menú de aplicaciones, donde se puede notar que openflow se encuentra desactivado, ver Figura 4.12), para esto se ejecutan los siguientes comandos:

> apps -s
> feature:list | grep onos-app
> feature:install onos-apps-fwd
> list | grep onos-*
> app activate org.onosproject.openflow
> apps -a -s

Luego de ejecutar estos comandos se puede verificar en la GUI de ONOS que en la sección de aplicaciones ya se encuentra activo el protocolo openflow, como se puede ver en la Figura 4.13.

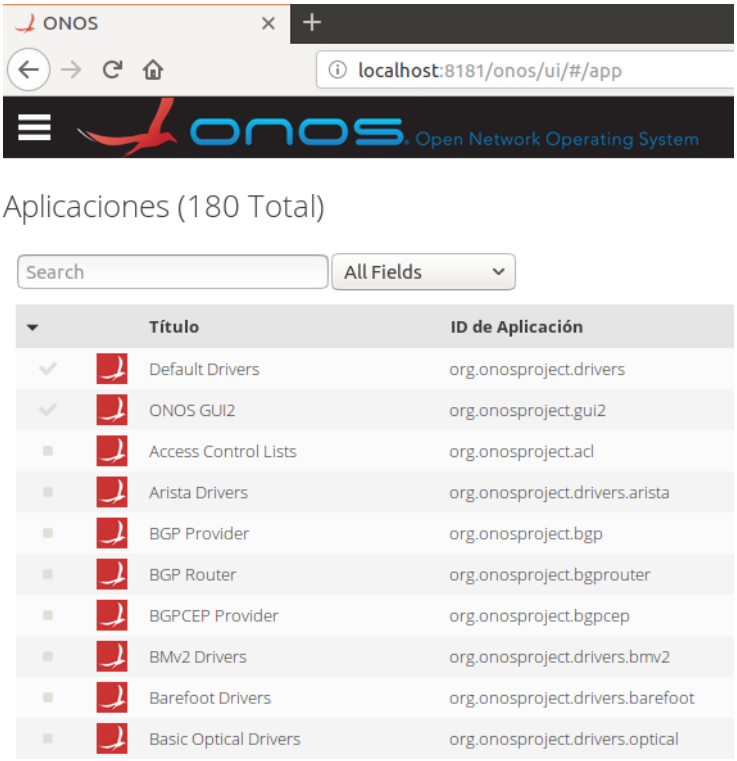


Figura 4.12. Sección de aplicaciones en la GUI de ONOS – OpenFlow desactivado

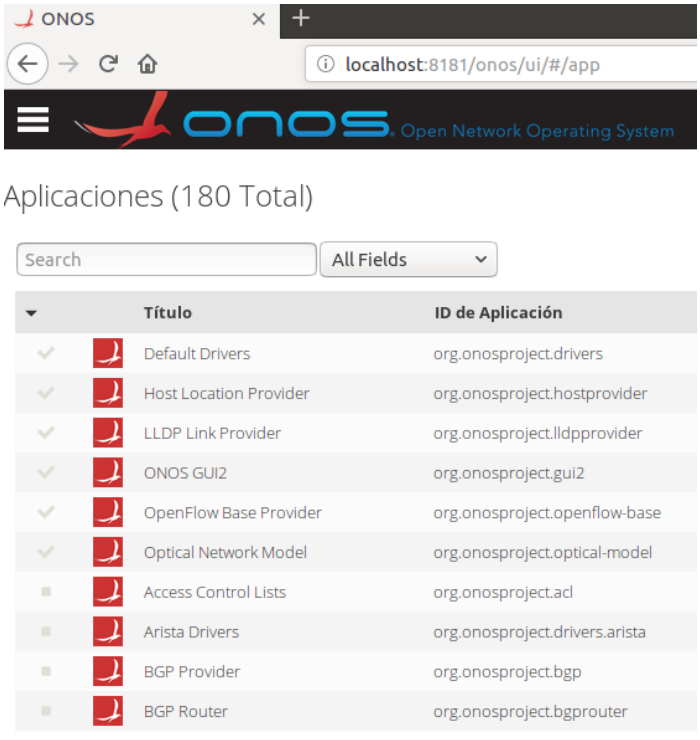


Figura 4.13. Sección de aplicaciones en la GUI de ONOS – OpenFlow activado

La ejecución de estos comandos solo es necesaria una vez, ya que esta configuración queda guardada.

Ahora para comprobar el funcionamiento, se abre una terminal y se ejecuta el comando “sudo mn --controller=remote,ip=192.168.0.11 --topo=tree,2,2” en ip se debe poner la dirección del equipo o en su defecto se puede hacer uso de la dirección local “127.0.0.1”.

Una vez ejecutado este comando, la GUI de ONOS se verá como en la Figura 4.14.



Figura 4.14. Prueba de conexión entre mininet y onos

Y luego es posible ejecutar “pingall” para que de esta manera los hosts sean visibles en la GUI de ONOS como se puede observar en la Figura 4.15. Si después de ejecutar “pingall” no se hacen visibles los hosts, basta con ir a la GUI de ONOS y presionar la tecla “h”.

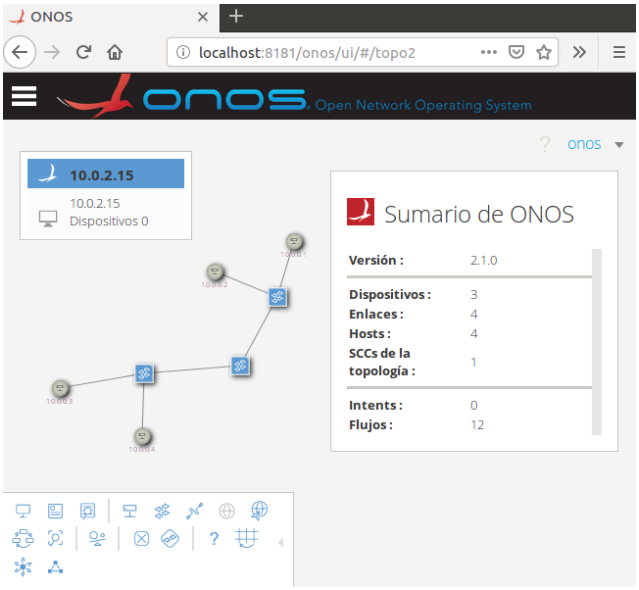


Figura 4.15. Prueba de conexión entre mininet y onos – Con hosts visibles

4.4 Generador de tráfico D-ITG

Con el fin de realizar pruebas más realistas dentro de la red emulada mediante mininet, se realizará la instalación del generador de tráfico D-ITG, haciendo uso del comando "sudo apt-get install d-itg", como se muestra en la Figura 4.16.

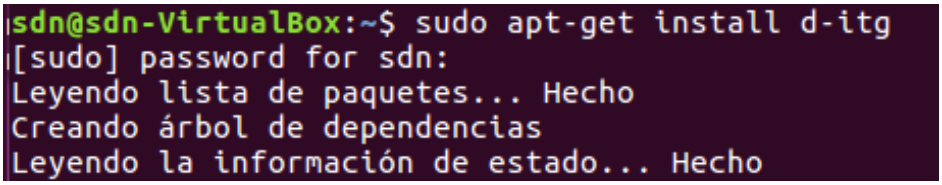


Figura 4.16. Comando de instalación del generador de tráfico D-ITG

El generador de tráfico D-ITG, permite configurar un host como emisor o receptor, de manera que al tener configurado un host como emisor y otro host como receptor es posible generar tráfico dentro de la red, para realizar dicha configuración se hace uso del comando "ITGRecv" para el receptor y el comando "ITGSend" para el emisor. Algunos de los parámetros básicos que permite modificar D-ITG son:

- Protocolo de comunicación: Con el comando "-T" seguido del protocolo (UDP, TCP).
- Seleccionar el host al cual se va transmitir: Con el comando "-a" seguido de la dirección ip de dicho host.
- Tamaño en bytes de los datos que se envían: Con el comando "-c" seguido del tamaño de los datos en bytes.
- Cantidad de paquetes: Con el comando "-C" seguido de la cantidad de paquetes por segundo.
- Tiempo total: Con el comando "-t" seguido del tiempo durante el que se generara el tráfico.
- Almacenamiento de un registro de los resultados: Con el comando "-l" y "-x" seguido del nombre del archivo en el que se almacenara dicha información (por ejemplo: sender.log o receiver.log) se almacena un registro tanto del transmisor como del receptor.

Una vez finaliza el tiempo que se configuro para generar el tráfico es posible consultar el registro mediante el comando "ITGDec receiver.log" para el caso del receptor o "ITGDec sender.log" para el caso del emisor.

A continuación, se realizará una descripción de cómo hacer uso del generador de tráfico en conjunto con mininet y la topología de red mostrada en la Figura 4.15Figura 4.17:

- Desde la interfaz de mininet se ejecuta el comando "xterm" seguido de los hosts que se desean configurar como emisor y receptor, en la Figura 4.17 se aprecia un ejemplo con los hosts h1 y h4, en la Figura 4.18 es posible observar el resultado del comando anterior, el cual abre una ventana para cada host.

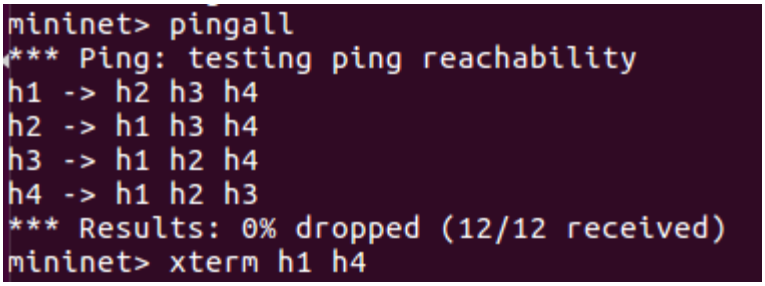


Figura 4.17. Comando para abrir emulador de terminal

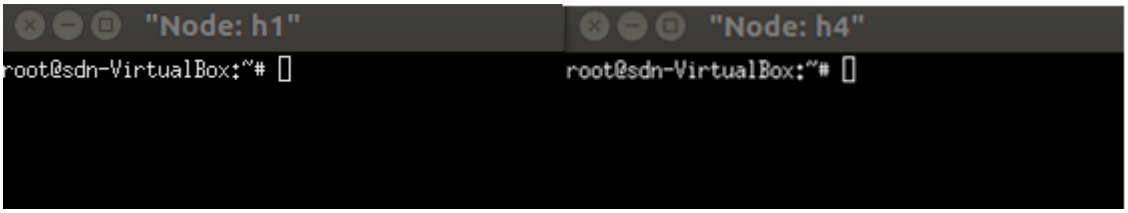


Figura 4.18. Emulador de terminales para h1 y h4

- Para este caso se configura h1 como receptor como se muestra en la Figura 4.19, de modo que está a la espera de tráfico entrante.
- Se configura h4 como emisor como se muestra en la Figura 4.20 y además se guarda un registro de resultados en los archivos “receiver.log y “sender.log”.

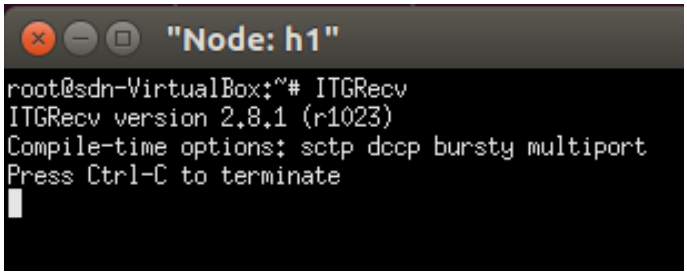


Figura 4.19. Host h1 configurado como receptor

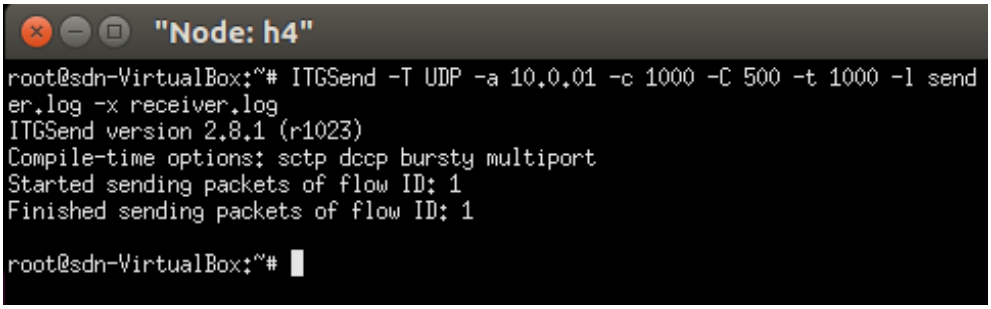


Figura 4.20. Host h4 configurado como emisor

En la GUI de ONOS, al presionar la letra “A”, permite visualizar el flujo de bytes dentro de la red en tiempo real, en la figura 21, es posible apreciar dicho flujo de bytes.

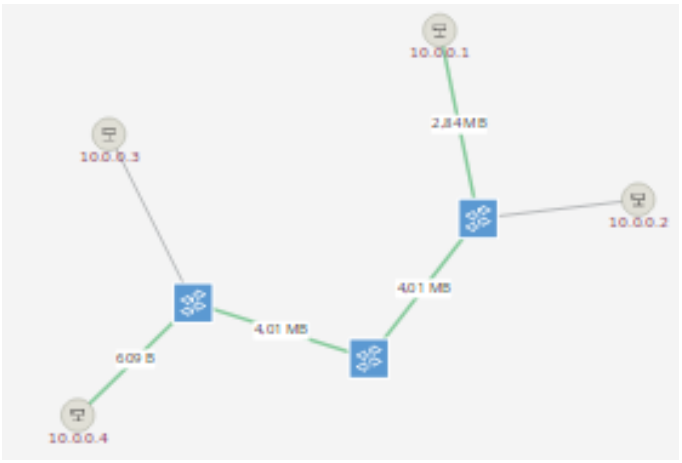


Figura 4.21. Flujo de bytes en la GUI de ONOS

En la Figura 4.22, se evidencia el registro almacenado en el archivo "receiver.log", dicho registro posee información de algunos parámetros de la red durante el tráfico generado, algunos de los parámetros, son el delay máximo, mínimo y promedio, el jitter, el número de paquetes perdidos, entre otros.

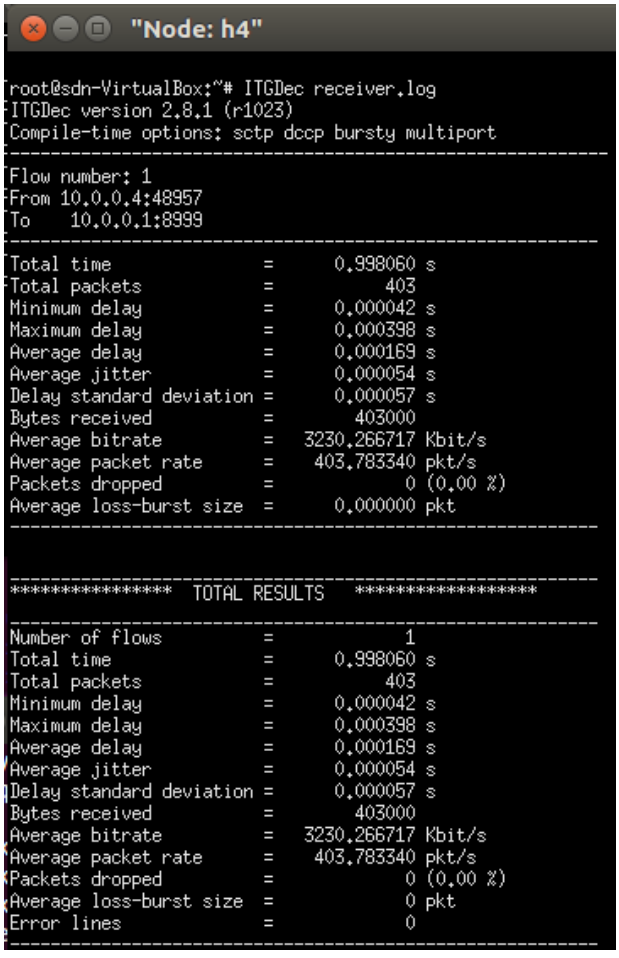


Figura 4.22. Registro almacenado en el archivo "receiver.log"

4.5 Generador de tráfico iperf

Para realizar la instalación del generador de tráfico iperf, se hace uso del comando "sudo apt-get install iperf", como se muestra en la Figura 4.23.

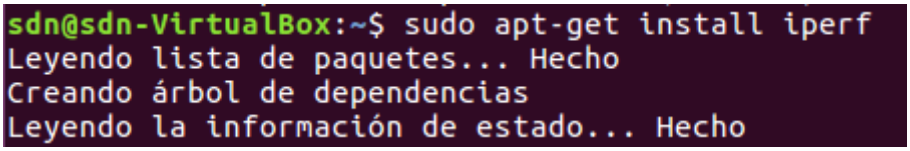


Figura 4.23. Comando de instalación del generador de tráfico iperf

El generador de tráfico iperf, al igual que el generador D-ITG, requiere configurar un host como emisor y otro como receptor, al principio se debe especificar el tipo de tráfico de la siguiente manera:

Trafico TCP Emisor / Receptor:

- Receptor: Con el comando "iperf -s" se configura host como receptor de mensajes tcp.
- Emisor: Con el comando "iperf" se configura host como emisor.

Trafico UDP Emisor / Receptor:

- Receptor: Con el comando "iperf -s -u" se configura host como receptor de mensajes udp.
- Emisor: Con el comando "iperf -u" se configura host como emisor.

Comandos para el emisor:

- Seleccionar el host al cual se va transmitir: Con el comando "-c" seguido de la dirección ip de dicho host.
- Tiempo total: Con el comando "-t" seguido del tiempo durante el que se generara el tráfico.
- Tasa de transmisión: Con el comando "-b" seguido de la tasa de transmisión en bytes por segundo.
- Visualización de un registro de los resultados: Con el comando "-i" seguido del tiempo en segundos deseado para los intervalos de visualización de resultados.

A continuación, se realizará una descripción de cómo hacer uso del generador de tráfico en conjunto con mininet y la topología de red mostrada en la Figura 4.15:

- Desde la interfaz de mininet se ejecuta el comando "xterm" seguido de los hosts que se desean configurar como emisor y receptor, en la Figura 4.17 se aprecia un ejemplo con los hosts h1 y h4, en la Figura 4.18 es posible observar el resultado del comando anterior, el cual abre una ventana para cada host.
- Para este caso se configura h1 como receptor, esto se puede observar en la Figura 4.24, de modo que está a la espera de tráfico udp entrante.

- Se configura h4 como emisor como se muestra en la Figura 4.25 y además se evidencia un registro que posee información de algunos parámetros de la red durante el tráfico generado.

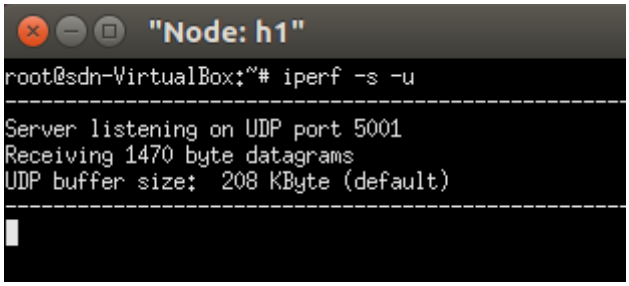


Figura 4.24. Host h1 configurado como receptor

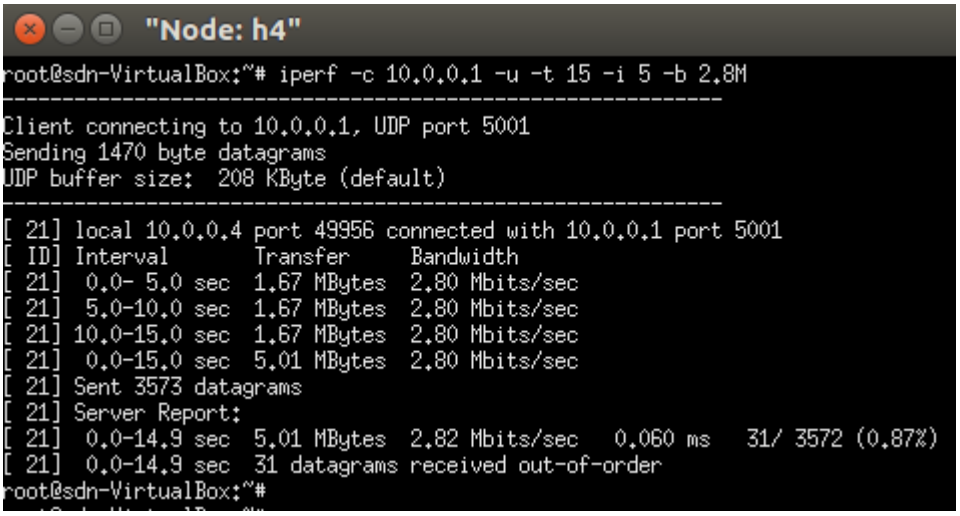


Figura 4.25. Host h4 configurado como emisor

Por otro lado, en el receptor se observa un reporte similar al del emisor, pero con nueva información, en la Figura 4.26, es posible apreciar que el reporte en el receptor nos da información sobre el jitter durante el tráfico generado.

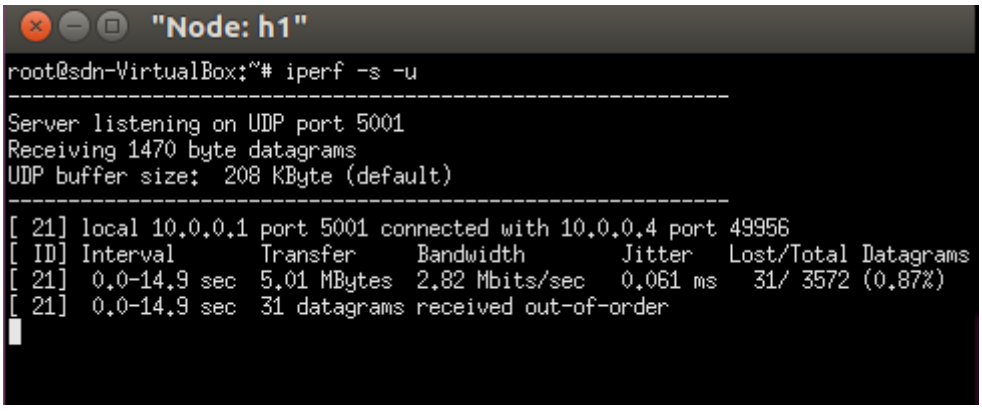


Figura 4.26. Reporte en el receptor

5. Desarrollo del algoritmo

5.1 Crear una nueva aplicación en ONOS

Para crear una nueva aplicación en ONOS es necesario realizar la instalación de Apache Ant haciendo uso del comando "sudo apt-get install ant" como se muestra en la Figura 5.1.

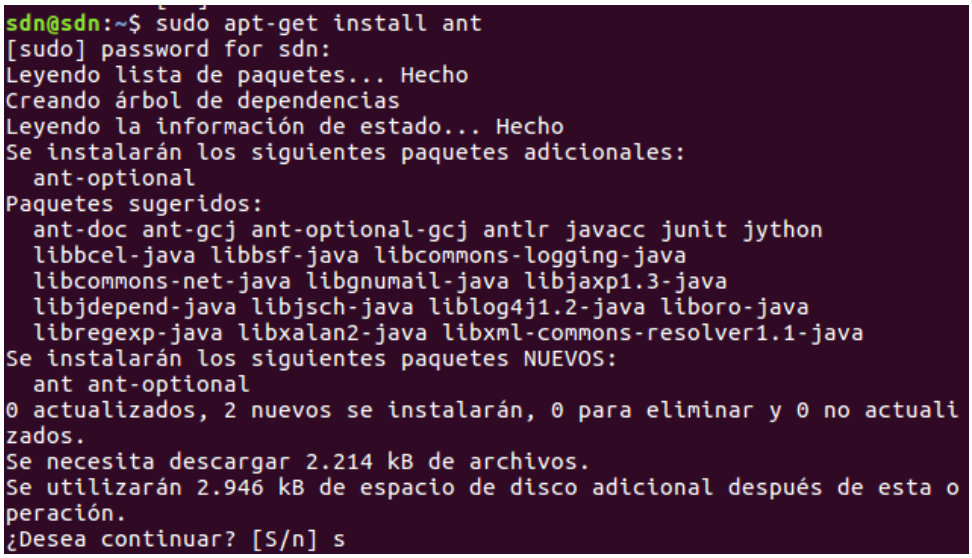


Figura 5.1. Comando de instalación de Apache Ant

Posteriormente se realiza la instalación de maven con el comando "sudo apt-get -y install maven" como se muestra en la Figura 5.2.

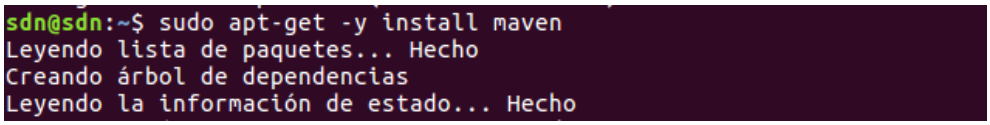


Figura 5.2. Comando de instalación de maven

Una vez realizada la instalación de Apache Ant y de maven, es necesario realizar las siguientes modificaciones:

- Se crea una carpeta con el nombre "onos-aux" (podría ser cualquiera) en la carpeta personal.
- Se abre una terminal y se ejecuta el comando "cd onos-aux".
- Luego se descarga la carpeta con el código fuente de onos haciendo uso del comando "git clone https://gerrit.onosproject.org/onos".
- Por último, vamos a la ubicación "onos-aux/onos/" y le damos copiar a la carpeta "tools", y la pegamos en la dirección donde se tiene instalado onos ("onos/").

Para empezar con la creación de una nueva aplicación para onos, se hace uso del comando "mvn archetype:generate -DarchetypeGroupId=org.onosproject -

DarchetypeArtifactId=onos-bundle-archetype” como se muestra en la Figura 5.3, y en el apartado para la definición de la aplicación se completa de la siguiente manera:

Define value for property 'groupId': : org.foo
Define value for property 'artifactId': : foo-app
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': org.foo: : org.foo.app
Confirm properties configuration:
groupId: org.foo
artifactId: foo-app
version: 1.0-SNAPSHOT
package: org.foo.app
Y: :

```
sdn@sdn:~$ mvn archetype:generate -DarchetypeGroupId=org.onosproject -
DarchetypeArtifactId=onos-bundle-archetype
[INFO] Scanning for projects...
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plu
gins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plug
ins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom (4 KB at 1.2 KB/
sec)
```

Figura 5.3. Comando para crear una nueva aplicación en ONOS

En la Figura 5.4, es posible apreciar el apartado para la definición de la aplicación además de la correcta construcción de la misma.

```
Define value for property 'groupId': org.foo
Define value for property 'artifactId': foo-app
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' org.foo: : org.foo.app
Confirm properties configuration:
groupId: org.foo
artifactId: foo-app
version: 1.0-SNAPSHOT
package: org.foo.app
 Y: :
[INFO] ----------------------------------------------------------------
------------
[INFO] Using following parameters for creating project from Archetype:
 onos-bundle-archetype:2.0.0
[INFO] ----------------------------------------------------------------
------------
[INFO] Parameter: groupId, Value: org.foo
[INFO] Parameter: artifactId, Value: foo-app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: org.foo.app
[INFO] Parameter: packageInPathFormat, Value: org/foo/app
[INFO] Parameter: package, Value: org.foo.app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: groupId, Value: org.foo
[INFO] Parameter: artifactId, Value: foo-app
[INFO] Project created from Archetype in dir: /home/sdn/foo-app
[INFO] ----------------------------------------------------------------
---------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------------------
```

Figura 5.4. Resultado luego de la correcta construcción de la aplicación

Una vez realizada la correcta construcción de la aplicación, dentro de la carpeta con el nombre "foo-app" es necesario localizar el archivo pom.xml y descomentar la sección de propiedades, tal como se muestra en la Figura 5.5.

ANTES

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <onos.version>2.0.0</onos.version>
    <!-- Uncomment to generate ONOS app from this module.
    <onos.app.name>org.foo.app</onos.app.name>
    <onos.app.title>Foo App</onos.app.title>
    <onos.app.origin>Foo, Inc.</onos.app.origin>
    <onos.app.category>default</onos.app.category>
    <onos.app.url>http://onosproject.org</onos.app.url>
    <onos.app.readme>ONOS OSGi bundle archetype.</onos.app.readme>
    -->
</properties>
```

DESPUÉS

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <onos.version>2.0.0</onos.version>
    <!-- Uncomment to generate ONOS app from this module.-->
    <onos.app.name>org.foo.app</onos.app.name>
    <onos.app.title>Foo App</onos.app.title>
    <onos.app.origin>Foo, Inc.</onos.app.origin>
    <onos.app.category>default</onos.app.category>
    <onos.app.url>http://onosproject.org</onos.app.url>
    <onos.app.readme>ONOS OSGi bundle archetype.</onos.app.readme>

</properties>
```

Figura 5.5. Modificación del archivo "pom.xml"

A continuación, se hace uso del comando "mvn clean install" dentro de la carpeta foo-app, como se muestra en la Figura 5.6, esto permite realizar la compilación de la aplicación para posteriormente ser instalada en onos.

```
sdn@sdn:~/foo-app$ mvn clean install
[INFO] Scanning for projects...
Downloading: https://repo.maven.apache.org/maven2/org/apache/felix/mav
en-bundle-plugin/3.5.0/maven-bundle-plugin-3.5.0.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/felix/mave
n-bundle-plugin/3.5.0/maven-bundle-plugin-3.5.0.pom (10 KB at 3.3 KB/s
ec)
```

Figura 5.6. Comando para compilar el código fuente de la aplicación

Finalmente se realiza la instalación de la nueva aplicación creada con el nombre de foo-app, para esto se hace uso del comando "onos-app localhost install target/foo-app-1.0-SNAPSHOT.oar", como se muestra en la Figura 5.7. Cabe resaltar que antes de ejecutar este comando, el controlador onos se debe estar ejecutando.

```
sdn@sdn:~/foo-app$ onos-app localhost install target/foo-app-1.0-SNAPS
HOT.oar
{"name":"org.foo.app","id":180,"version":"1.0.SNAPSHOT","category":"de
fault","description":"ONOS OSGi bundle archetype.","readme":"ONOS OSGi
 bundle archetype.","origin":"Foo, Inc.","url":"http://onosproject.org
","featuresRepo":"mvn:org.foo/foo-app/1.0-SNAPSHOT/xml/features","stat
e":"INSTALLED","features":["foo-app"],"permissions":[],"requiredApps":
[]}
```

Figura 5.7. Comando para instalar la aplicación en ONOS

Por otra parte, si no se reconoce el comando onos-app se debe hacer lo siguiente:

- Se ejecuta el comando “cd”
- Se ejecuta el comando “source onos/tools/dev/bash_profile”
- Se ejecuta el comando “cd foo-app”
- Por último, se ejecuta de nuevo el comando “onos-app localhost install target/foo-app-1.0-SNAPSHOT.oar”

Lo anterior se apreciar en la Figura 5.8.

```
sdn@sdn:~/foo-app$ onos-app localhost install target/foo-app-1.0-SNAPSHOT.oar
onos-app: no se encontró la orden
sdn@sdn:~/foo-app$ cd
sdn@sdn:~$ source onos/tools/dev/bash_profile
sdn@sdn:~$ cd foo-app
sdn@sdn:~/foo-app$ onos-app localhost install target/foo-app-1.0-SNAPSHOT.oar
```

Figura 5.8. Comandos que se deben ejecutar si no reconoce el comando onos-app

Al hacer uso del comando “app -s” en la CLI de onos, se pueden ver todas las aplicaciones instaladas, y la nueva aplicación aparece correctamente instalada en el controlador onos como se pude apreciar en la Figura 5.9.

```
169 org.onosproject.pim                  2.0.0    Protocol Independent Multicast Emulation
170 org.onosproject.events               2.0.0    Event History
171 org.onosproject.loadtest             2.0.0    Distributed Load Test
172 org.onosproject.drivers.juniper      2.0.0    Juniper Drivers
173 org.onosproject.newoptical           2.0.0    Packet/Optical Use-Case
174 org.onosproject.drivers.corsa        2.0.0    Corsa Drivers
175 org.onosproject.openstacknetworkingui 2.0.0    Openstack Networking UI
176 org.onosproject.cluster-ha           2.0.0    Cluster HA Test
177 org.onosproject.reactive-routing     2.0.0    SDN-IP Reactive Routing
180 org.foo.app                          1.0.SNAPSHOT Foo App
```

Figura 5.9. Resultado del comando “app -s” en la CLI de ONOS

Para permitir que la aplicación agregue comandos a la CLI de ONOS, se superpone la interfaz de la CLI haciendo uso del comando “onos-create-app cli org.foo foo-app 1.0-SNAPSHOT org.foo.app”, como se muestra en la Figura 5.10.

```
sdn@sdn:~/foo-app$ onos-create-app cli org.foo foo-app 1.0-SNAPSHOT or
g.foo.app
[INFO] Scanning for projects...
Downloading: https://repo.maven.apache.org/maven2/org/apache/felix/mav
en-scr-plugin/1.26.0/maven-scr-plugin-1.26.0.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/felix/mave
n-scr-plugin/1.26.0/maven-scr-plugin-1.26.0.pom (4 KB at 0.5 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/felix/mav
en-scr-plugin/1.26.0/maven-scr-plugin-1.26.0.jar
Downloaded: https://repo.maven.apache.org/maven2/org/apache/felix/mave
n-scr-plugin/1.26.0/maven-scr-plugin-1.26.0.jar (32 KB at 31.6 KB/sec)
[INFO]

[INFO] ------------------------------------------------------------------
---------
[INFO] Building foo-app 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------
```

Figura 5.10. Creación de un nuevo comando con la aplicación foo-app

Luego de esto se compila (con el comando “mvn clean install”) y finalmente se reinstala la aplicación foo-app mediante el comando “onos-app localhost reinstall! org.foo.app target/foo-app-1.0-SNAPSHOT.oar” como se muestra en la Figura 5.11.

```
sdn@sdn:~/foo-app$ onos-app localhost reinstall! target/foo-app-1.0-SN
APSHOT.oar
{"name":"org.foo.app","id":180,"version":"1.0.SNAPSHOT","category":"de
fault","description":"ONOS OSGi bundle archetype.","readme":"ONOS OSGi
 bundle archetype.","origin":"Foo, Inc.","url":"http://onosproject.org
","featuresRepo":"mvn:org.foo/foo-app/1.0-SNAPSHOT/xml/features","stat
e":"ACTIVE","features":["foo-app"],"permissions":[],"requiredApps":[]}
```

Figura 5.11. Comando para reinstalar la aplicación en ONOS

Finalmente, solo hace falta realizar la activación de la aplicación desde la interfaz de comandos CLI de onos, mediante el comando “app activate org.foo.app”, y para comprobar el nuevo comando agregado por defecto se ejecuta “sample”, que imprime “Hello world”, tal como se muestra en la Figura 5.12.

```
karaf@root > app activate org.foo.app                    21:31:46
Activated org.foo.app
karaf@root > sample                                      21:32:15
Hello World
```

Figura 5.12. Comando para activar la aplicación en ONOS

Para realizar la modificación de la aplicación se debe acceder al directorio “foo-app/src/main/java/org/foo/app/” en esta carpeta se encuentran los archivos “AppComponent.java” y “AppCommand.java”, los cuales permiten la implementación del algoritmo para el volcado del tráfico de datos. Una vez modificados estos archivos y se quieran implementar en onos, basta con volver a compilar, reinstalar y activar la aplicación como se mostró anteriormente.

5.2 Algoritmo del volcado del tráfico de datos

Antes de explicar el funcionamiento del algoritmo del volcado del tráfico de datos, veamos algunas definiciones previas:

$$t_p = K_p * t_s$$

$$t_c = K_c * t_p = K_c * K_p * t_s$$

$$K_p = 1,2,3, \ldots$$

$$K_c = 1,2,3, \ldots$$

Donde:

- t_s (tiempo de sondeo): Cada cuanto tiempo se van a medir los parámetros sobre la red.

- t_p (tiempo para promediar): Cada cuanto tiempo se toman decisiones sobre la red.
- t_c (tiempo de cambio): Tiempo de espera para volver a tomar una decisión sobre un host especifico.
- K_p (número de datos a promediar).
- K_c (constante de cambio).

Para dejar más claro lo anterior y explicar el funcionamiento del algoritmo, veamos un ejemplo en el cual solo se tendrá en cuenta un parámetro, el delay o retardo.

Para el ejemplo vamos a suponer:

$$t_s = 2\ s$$

$$K_p = 4$$

Entonces:

$$t_p = K_p * t_s = 8\ s$$

Esto significa que cada 2 segundos (t_s) se realizaran mediciones de los parámetros sobre la red, pero cada 8 segundos (t_p) se tomaran decisiones con base en el promedio de los últimos 4 (K_p) datos medidos. Esto se puede observar en la Figura 5.13.

Además, si suponemos:

$$K_c = 2$$

Entonces:

$$t_c = K_c * t_p = 16\ s$$

Esto quiere decir que cuando se efectúa un cambio (volcado del tráfico de datos) en determinado host, se debe esperar 16 segundos (t_c) para volver a tomar una decisión sobre dicho host. Pero para el resto de hosts en la red se seguirán evaluando los parámetros y decidiendo si realizar o no el volcado del tráfico de datos. Esto se puede observar en la Figura 5.14.

Luego de ver el funcionamiento del algoritmo del volcado del tráfico de datos, pasamos a realizar una definición del algoritmo en forma de pseudocódigo, esto se puede observar en la Figura 5.15.

El algoritmo del volcado de tráfico de datos planteado en la Figura 5.15 supone una serie de características que deben verse reflejadas dentro de la red en donde se desee implementar, dichas características son:

- Para cada host dentro de la red, existen dos canales de comunicación cuyo acceso está determinado por el puerto 1 y el puerto 2 respectivamente.
- El canal de comunicación conectado mediante el puerto 1 posee un ancho de banda inferior al del canal de comunicación conectado al puerto 2. Por ejemplo,

el puerto 1 permite una comunicación mediante WiFi, mientras que el puerto 2 permite una comunicación alámbrica como Ethernet.

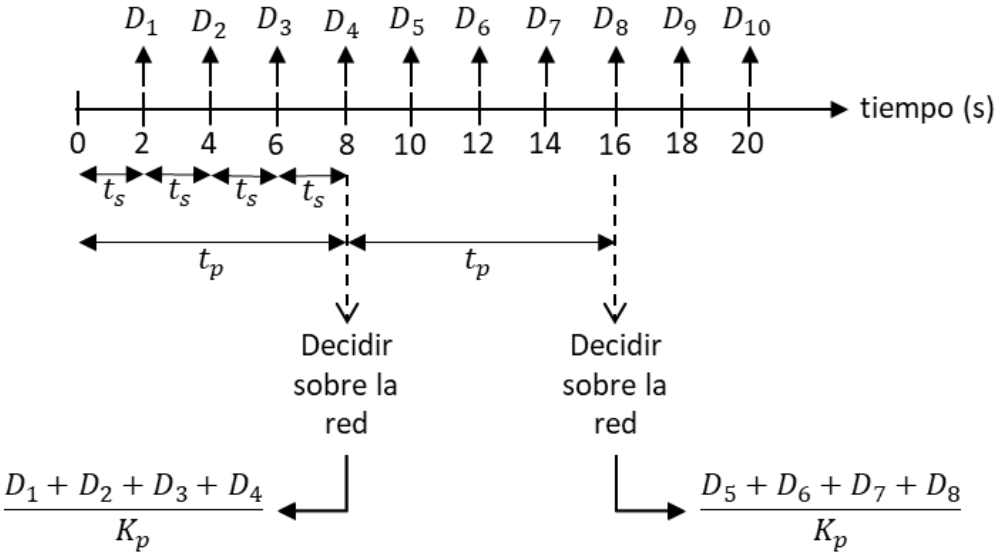


Figura 5.13. Funcionamiento del algoritmo en el tiempo – Parte 1

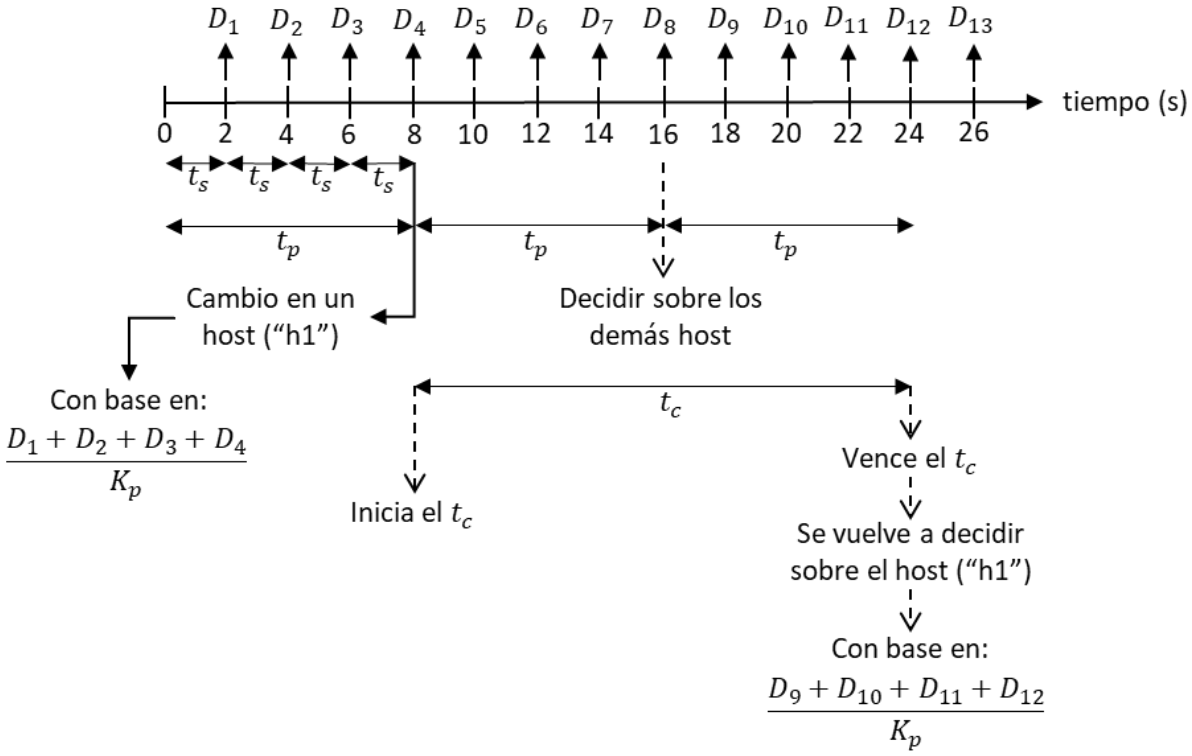


Figura 5.14. Funcionamiento del algoritmo en el tiempo – Parte 2

Algoritmo 1 Volcado del tráfico de datos

D_{TX} : Retardo transmisión
D_{RX} : Retardo recepción
T_{TX} : Tasa transmisión
T_{RX} : Tasa recepción
x : Número de sondeos
T_{max} : Umbral throughput
D_{max} : Umbral retardo transmisión/recepción
BW : Ancho de banda canal asociado al Puerto1
t_s : Tiempo de sondeo
t_c : Tiempo de cambio
k_p : Número de datos a promediar
k_c : Constante de cambio

```
for i=1 to x do
  for each host do
    Se capturan estadísticas del host;
    if (i > t_c) && (i mod k_p=0) then
      Se promedian los últimos k_p datos;
      if (D_TX > D_max) && ((T_TX + T_RX) > BW/2) || (D_RX > D_max) && ((T_TX + T_RX) > BW/2) || ((T_TX
      + T_RX) > T_max) then
        if Puerto1 = Activo then
          Se realiza volcado del tráfico de datos:
          Puerto2 = Activo;
          Puerto1 = Inactivo;
          t_c = i + k_p * k_c;
        end if
      else
        if Puerto2 = Activo then
          Puerto1 = Activo;
          Puerto2 = Inactivo;
          t_c = i + k_p * k_c;
        end if
      end if
    end if
  end for
  Espera t_s;
end for
```

Figura 5.15. Algoritmo del volcado de tráfico de datos

Como se puede observar en la Figura 5.15, para el caso de los retardos el algoritmo realiza el volcado del tráfico de datos cuando se sobrepasa el umbral, únicamente si la tasa de transmisión está por encima del 50% del ancho de banda del canal de comunicación asociado al puerto 1. La razón para definir dicha restricción, es que entre el controlador y la red existe un tráfico permanente y aunque dicho tráfico no tiene una tasa de trasmisión muy elevada, los retardos que se presentan en la mayoría de casos son bastante elevados.

En la Figura 5.16, se puede evidenciar el tráfico existente entre el controlador y la red; se trata de mensajes Broadcast 0x8942 y multicas LLDP (Link Layer Discovery Protocol). Es gracias a estos mensajes que el controlador descubre los dispositivos conectados, obtiene información del estado actual de la red y además le permite realizar el control sobre la misma. En la Figura 5.17 es posible apreciar una gráfica de la tasa de transmisión del tráfico generado entre el controlador y la red, esta tasa es de 4.4 kbps a intervalos regulares de 3 segundos.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	TTL = 120
2	0.001160464	02:eb:9f:67:c9:42	Broadcast	0x8942	139	Ethernet II
3	0.099235362	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	TTL = 120
4	0.099354856	02:eb:9f:67:c9:42	Broadcast	0x8942	139	Ethernet II
5	3.096382647	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	TTL = 120
6	3.096908937	02:eb:9f:67:c9:42	Broadcast	0x8942	139	Ethernet II
7	3.200001070	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	TTL = 120
8	3.200697625	02:eb:9f:67:c9:42	Broadcast	0x8942	139	Ethernet II
9	6.197822303	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	TTL = 120
10	6.198817966	02:eb:9f:67:c9:42	Broadcast	0x8942	139	Ethernet II
11	6.306126383	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	TTL = 120
12	6.306261786	02:eb:9f:67:c9:42	Broadcast	0x8942	139	Ethernet II

```
▸ Frame 1: 139 bytes on wire (1112 bits), 139 bytes captured (1112 bits) on interface 0
▾ Ethernet II, Src: 02:eb:9f:67:c9:42 (02:eb:9f:67:c9:42), Dst: LLDP_Multicast (01:80:c2:00:00:0e)
  ▸ Destination: LLDP_Multicast (01:80:c2:00:00:0e)
  ▸ Source: 02:eb:9f:67:c9:42 (02:eb:9f:67:c9:42)
    Type: 802.1 Link Layer Discovery Protocol (LLDP) (0x88cc)
▸ Link Layer Discovery Protocol
```

Figura 5.16. Mensajes entre controlador y red

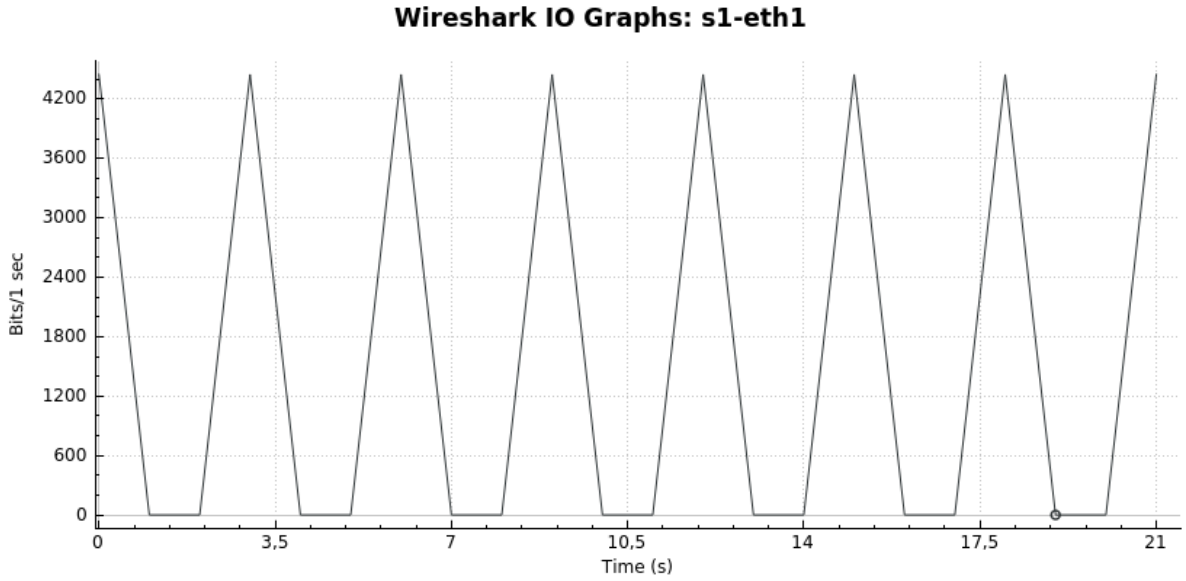


Figura 5.17. Tasa de transmisión mensajes entre el controlador y la red

5.3 Comandos preestablecidos en ONOS

Para la implementación del algoritmo se han tomado como base algunos comandos preestablecidos en ONOS.

Antes de ver los comandos, cabe resaltar que los ejemplos de estos comandos se realizaron emulando una red con mininet tipo árbol como la de la Figura 4.15.

- *SUMMARY:* Con este comando es posible determinar varias características de la red conectada al controlador, tales como nodos, número de dispositivos, número enlaces, número de hosts, entre otras, en la Figura 5.18 se aprecia un ejemplo de este comando.

```
karaf@root > summary                                          21:46:48
node=10.0.2.15, version=2.0.0 clusterId=default
nodes=1, devices=3, links=4, hosts=4, SCC(s)=1, flows=12, intents=0
```

Figura 5.18. Comando “summary” en la CLI de ONOS

- *PORTSTATS:* Con este comando se puede adquirir estadísticas en cada uno de los puertos de los dispositivos en la red, tales como paquetes transmitidos y recibidos, bytes transmitidos y recibidos, paquetes perdidos y el tiempo que el puerto ha estado activo, en la Figura 5.19 se evidencia un ejemplo de este comando.

```
karaf@root > portstats                                        21:42:19
deviceId=of:0000000000000001
   port=1, pktRx=208, pktTx=207, bytesRx=27618, bytesTx=27658, pktRxDr
p=0, pktTxDrp=0, Dur=254
   port=2, pktRx=206, pktTx=209, bytesRx=27470, bytesTx=27826, pktRxDr
p=0, pktTxDrp=0, Dur=254
deviceId=of:0000000000000002
   port=1, pktRx=25, pktTx=204, bytesRx=1866, bytesTx=27392, pktRxDrp=
0, pktTxDrp=0, Dur=254
   port=2, pktRx=25, pktTx=208, bytesRx=1866, bytesTx=27700, pktRxDrp=
0, pktTxDrp=0, Dur=254
   port=3, pktRx=207, pktTx=208, bytesRx=27658, bytesTx=27618, pktRxDr
p=0, pktTxDrp=0, Dur=254
```

Figura 5.19. Comando “portstats” en la CLI de ONOS

- *LINKS:* Con este comando se pueden adquirir los distintos enlaces establecidos para un dispositivo en concreto, presentando la id del dispositivo y el puerto mediante el cual está conectado, además del estado del enlace (activo/inactivo), en la Figura 5.20 se evidencia un ejemplo de este comando.

```
karaf@root > links of:0000000000000001                        14:46:51
src=of:0000000000000001/1, dst=of:0000000000000002/3, type=DIRECT, sta
te=ACTIVE, expected=false
src=of:0000000000000001/2, dst=of:0000000000000003/3, type=DIRECT, sta
te=ACTIVE, expected=false
src=of:0000000000000002/3, dst=of:0000000000000001/1, type=DIRECT, sta
te=ACTIVE, expected=false
src=of:0000000000000003/3, dst=of:0000000000000001/2, type=DIRECT, sta
te=ACTIVE, expected=false
```

Figura 5.20. Comando “links” en la CLI de ONOS

- *EDGE-PORTS:* Con este comando se puede adquirir el puerto mediante el cual un dispositivo en concreto está conectado a uno o varios hosts, en la Figura 5.21 se evidencia un ejemplo de este comando.

```
karaf@root > edge-ports of:0000000000000001                   14:49:46
karaf@root > edge-ports of:0000000000000002                   14:50:07
of:0000000000000002/1
of:0000000000000002/2
```

Figura 5.21. Comando “edge-ports” en la CLI de ONOS

- *PORTSATATE:* Haciendo uso de este comando y de las opciones “enable” o “disable” es posible habilitar o deshabilitar un puerto, especificando el dispositivo y el respectivo puerto, en la Figura 5.22 se evidencia un ejemplo de este comando y en la Figura 5.23 se observa como al deshabilitar un puerto, desaparece el enlace de comunicación entre dos dispositivos.

```
karaf@root > portstate of:0000000000000002 3 disable          18:40:30
18:40:37.362 INFO  [pipe-portstate of:0000000000000002 3 disable] Por
t 3 on device of:0000000000000002 being administratively brought DOWN
18:40:37.396 INFO  [onos-of-worker-0] Received port status message fr
om 00:00:00:00:00:00:00:02/3: OFPortStatusVer13(xid=0, reason=MODIFY,
 desc=OFPortDescVer13(portNo=3, hwAddr=46:59:b5:6f:4f:c0, name=s2-eth
3, config=[PORT_DOWN], state=[LINK_DOWN], curr=[PF_10GB_FD, PF_COPPER
], advertised=[], supported=[], peer=[], currSpeed=10000000, maxSpeed
=0))
18:40:37.409 INFO  [onos-of-worker-0] Device of:0000000000000002 port
 3 status changed
karaf@root > 18:40:37.486 INFO  [onos-of-worker-0] Received port stat
us message from 00:00:00:00:00:00:00:01/1: OFPortStatusVer13(xid=0, r
eason=MODIFY, desc=OFPortDescVer13(portNo=1, hwAddr=d6:36:43:8c:d0:1b
, name=s1-eth1, config=[], state=[LINK_DOWN], curr=[PF_10GB_FD, PF_CO
PPER], advertised=[], supported=[], peer=[], currSpeed=10000000, maxS
peed=0))
18:40:37.489 INFO  [onos-of-worker-0] Device of:0000000000000001 port
 1 status changed
18:40:37.529 INFO  [onos-topo-build-6] Topology DefaultTopology{time=
3818048659491, creationTime=1562110837437, computeCost=409905, cluste
rs=2, devices=3, links=2} changed
```

Figura 5.22. Comando “portstate” en la CLI de ONOS

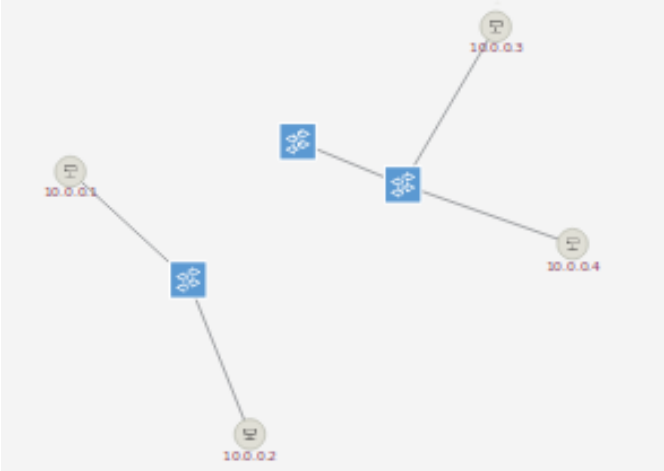


Figura 5.23. GUI de ONOS después de ejecutar el comando “portstate”

5.4 Preparación del entorno de pruebas en ONOS

Antes de realizar la implementación del algoritmo, se debe hacer una modificación en la aplicación “OpenFlow Base Provider” de ONOS, esto debido a que la medición de las

estadísticas de la red se hace por medio de esta aplicación, la cual tiene un intervalo de sondeo por defecto de 5 segundos. Este intervalo se puede modificar de manera sencilla con el comando "cfg set org.onosproject.provider.of.device.impl.OpenFlowDeviceProvider portStatsPollFrequency 1", donde el número 1 se puede cambiar por el intervalo de sondeo que se quiera en segundos, pero de esta manera no es posible obtener un intervalo de sondeo menor a 1 segundo, ya que la variable es de tipo entero (integer), es por esta razón que se debe realizar un ajuste en el código fuente de la aplicación "OpenFlow Base Provider" de ONOS.

Para realizar la modificación que nos permite tener un intervalo de sondeo menor a 1 segundo, debemos realizar los siguientes pasos:

- Teniendo en cuenta que anteriormente ya se había descargado el código fuente de onos en la carpeta onos-aux, nos dirigimos a la ubicación "onos-aux/onos/providers/openflow/device/src/main/java/org/onosproject/provider/of/device/impl", seleccionamos el archivo "PortStatsCollector.java" y le damos copiar. Luego nos dirigimos a la ubicación "foo-app/src/main/java/org/foo/app" (acá se encuentran los archivos de la aplicación creada anteriormente) y pegamos el archivo en esta ubicación.
- Estando en la ubicación "foo-app/src/main/java/org/foo/app", abrimos el archivo "PortStatsCollector.java" en el cual modificamos la línea número 39, asignando a la variable "SECONDS" el valor "10L", lo cual nos permitirá tener intervalos de sondeo múltiplos de 10 ms y no de 1000 ms (1 s) cómo está por defecto. Esto se puede observar en la Figura 5.24.

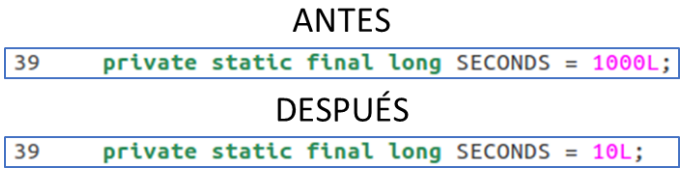


Figura 5.24. Modificación en el archivo "PortStatsCollector.java"

- Luego nos dirigimos a la ubicación "foo-app" y abrimos el archivo "pom.xml", el cual modificamos como se puede observar en la Figura 5.25.
- Luego abrimos una terminal y ejecutamos el comando "cd foo-app" y posteriormente el comando "mvn clean install", para de esta manera compilar el archivo modificado "PortStatsCollector.java".
- Nos dirigimos a la ubicación "foo-app/target" y con el gestor de archivadores abrimos el archivo "foo-app-1.0-SNAPSHOT.jar", luego desde el gestor de archivadores nos dirigimos a la ubicación "/org/onosproject/provider/of/device/impl/" y seleccionamos los 3 archivos que se encuentran en esta ubicación (como se puede observar en la Figura 5.26), y extraemos estos archivos en la carpeta de descargas.

ANTES

```
    <dependency>
  <groupId>org.onosproject</groupId>
  <artifactId>onos-cli</artifactId>
  <version>${onos.version}</version>
  <scope>provided</scope>
</dependency>

    <dependency>
  <groupId>org.osgi</groupId>
  <artifactId>org.osgi.core</artifactId>
  <version>5.0.0</version>
  <scope>provided</scope>
</dependency>
```

DESPUÉS

```
    <dependency>
  <groupId>org.onosproject</groupId>
  <artifactId>onos-cli</artifactId>
  <version>${onos.version}</version>
  <scope>provided</scope>
</dependency>

    <dependency>
  <groupId>org.onosproject</groupId>
  <artifactId>onos-protocols-openflow-api</artifactId>
  <version>${onos.version}</version>
</dependency>

    <dependency>
  <groupId>org.osgi</groupId>
  <artifactId>org.osgi.core</artifactId>
  <version>5.0.0</version>
  <scope>provided</scope>
</dependency>
```

Figura 5.25. Modificación en el archivo “pom.xml”

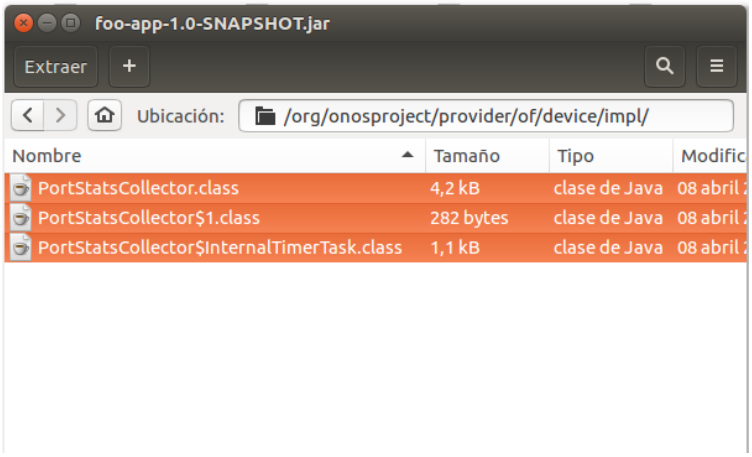


Figura 5.26. Contenido del archivo “foo-app-1.0-SNAPSHOT.jar”

- Ahora ejecutamos onos y abrimos la GUI de onos, desde donde vamos a abrir el menú desplegable en la esquina superior izquierda y en este menú damos clic en “Aplicaciones”.

- Luego seleccionamos la aplicación “OpenFlow Base Provider” y le damos “Descargar la aplicación seleccionada (fichero .oar file)”, como se puede observar en la Figura 5.27.

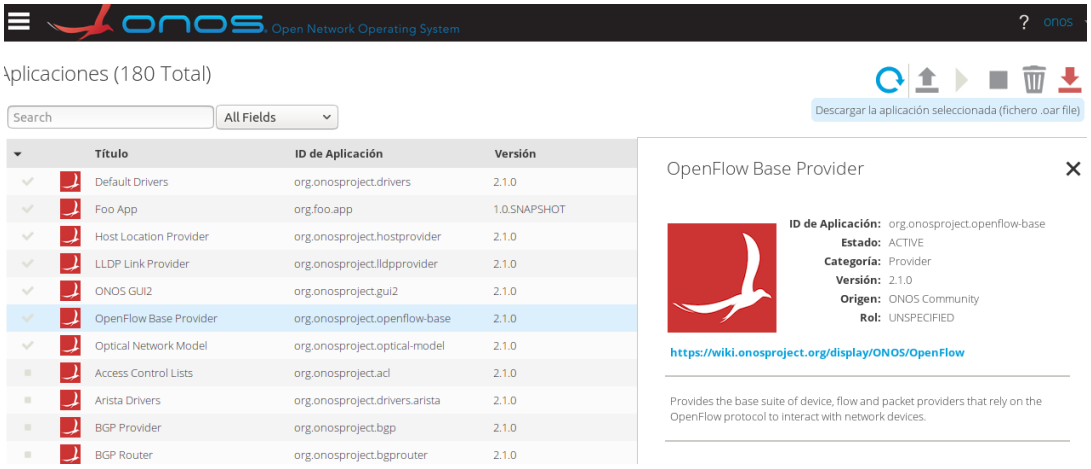


Figura 5.27. Sección de aplicaciones en ONOS – Descarga de la aplicación “OpenFlow Base Provider”

- Y ahora le damos clic en “Desinstalar aplicación seleccionada”, como se puede observar en la Figura 5.28.

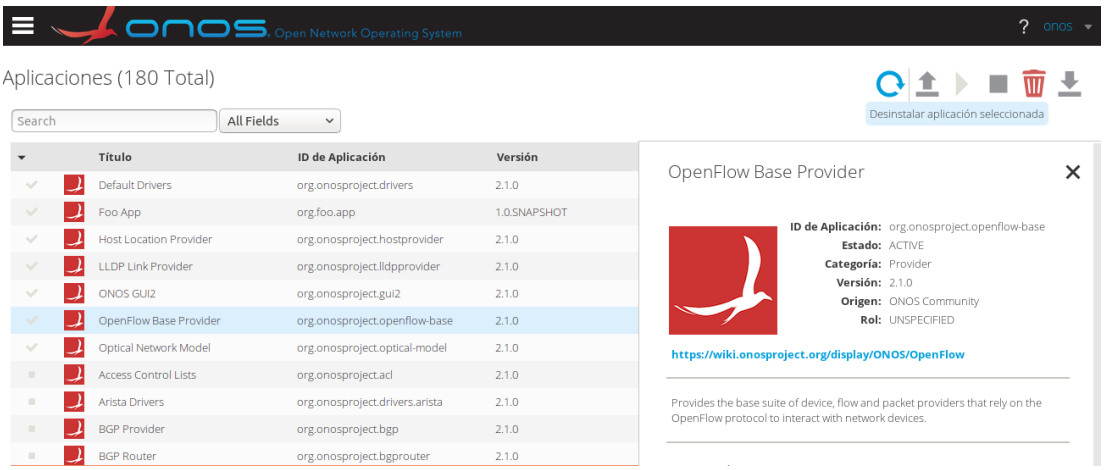


Figura 5.28. Sección de aplicaciones en ONOS – Desinstalación de la aplicación “OpenFlow Base Provider”

- Posteriormente nos dirigimos a la carpeta de descargas y con el gestor de archivadores abrimos el archivo (“org.onosproject.openflow-base.oar”) descargado anteriormente desde la GUI de onos, y desde el gestor de archivadores nos dirigimos a la ubicación “/m2/org/onosproject/onos-providers-openflow-device/2.0.0/” y extraemos el archivo “onos-providers-openflow-device-2.0.0.jar”.

- Ahora en la carpeta de descargas seleccionamos el archivo “onos-providers-openflow-device-2.0.0.jar” y abrimos las propiedades del archivo, luego vamos a la pestaña de permisos y en acceso seleccionamos “lectura y escritura”.
- Con el paso anterior ya podemos abrir el archivo “onos-providers-openflow-device-2.0.0.jar” con el gestor de archivadores. Y desde el gestor de archivadores nos dirigimos a la ubicación “/org/onosproject/provider/of/device/impl/” y acá le damos añadir, lo cual abre una ventana donde seleccionamos los 3 archivos que habíamos extraído antes en la carpeta de descargas, y antes de dar en aceptar nos aseguramos de que la casilla “Añadir solo si es más nuevo” este activa. En la Figura 5.29 podemos observar esto, y los 3 archivos seleccionados en esta figura deben tener una fecha de modificación más reciente que el resto.

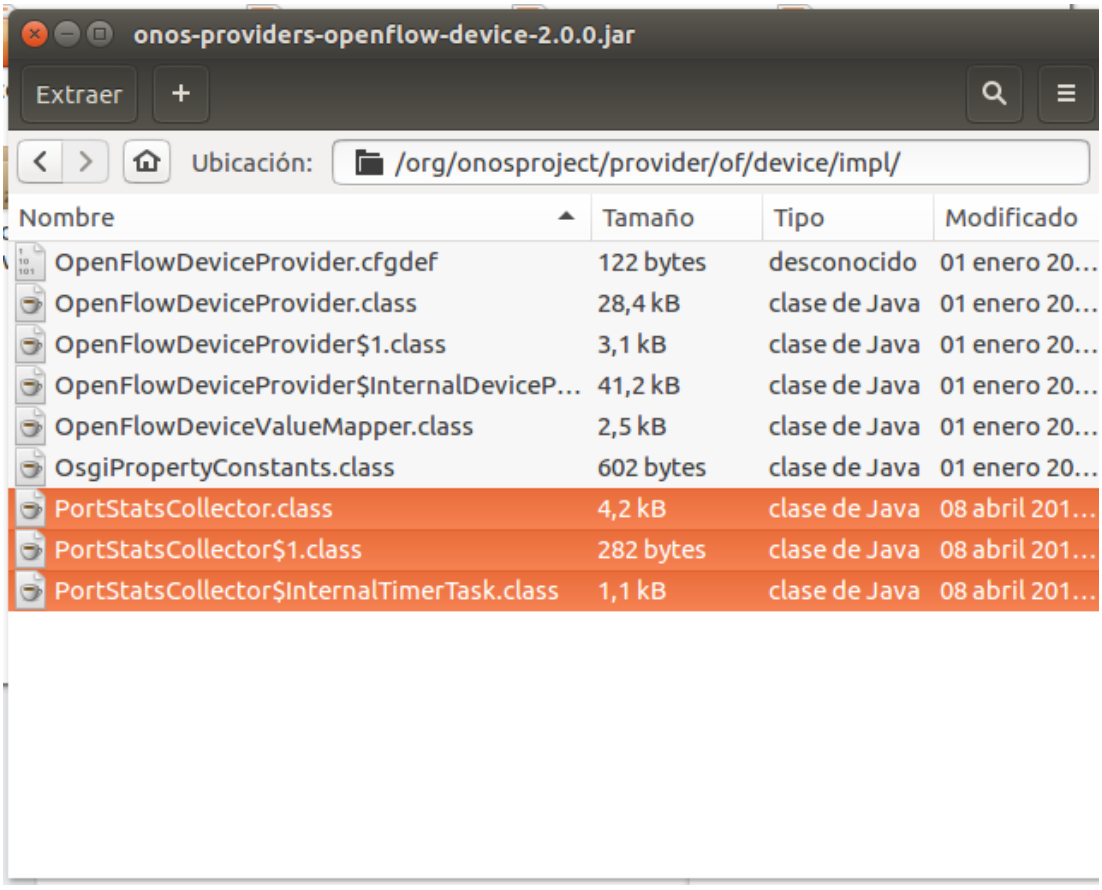


Figura 5.29. Contenido del archivo “onos-providers-openflow-device-2.0.0.jar”

- Luego nos dirigimos a la carpeta de descargas y con el gestor de archivadores abrimos el archivo “org.onosproject.openflow-base.oar”, y nos dirigimos a la ubicación “/m2/org/onosproject/onos-providers-openflow-device/2.0.0/” y acá le damos añadir, lo cual abre una ventana donde seleccionamos el archivo “onos-providers-openflow-device-2.0.0.jar” que se modificó en el paso anterior, y antes de dar en aceptar nos aseguramos de que la casilla “Añadir solo si es más nuevo” este activa.

- De esta manera ya tendríamos lista la modificación en la aplicación “OpenFlow Base Provider”, así que solo queda instalar y activar la aplicación. Para esto vamos a la GUI de onos y desde la sección de “Aplicaciones” le damos en “Cargar una aplicación (fichero .oar)” como se muestra en la Figura 5.30, y en la ventana que se abre seleccionamos el archivo “org.onosproject.openflow-base.oar” y damos en aceptar, de esta manera la aplicación se instalara de nuevo. Finalmente, para activar la aplicación simplemente damos clic en “Activar aplicación seleccionada”. Luego de esto se debe ver como en la Figura 5.31.

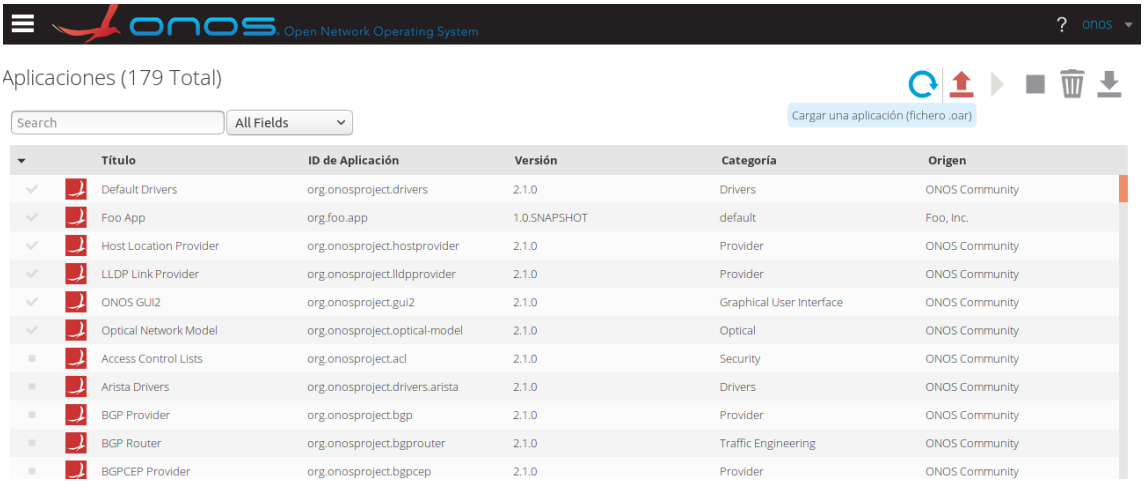


Figura 5.30. Sección de aplicaciones en ONOS – Instalación de la aplicación “OpenFlow Base Provider”

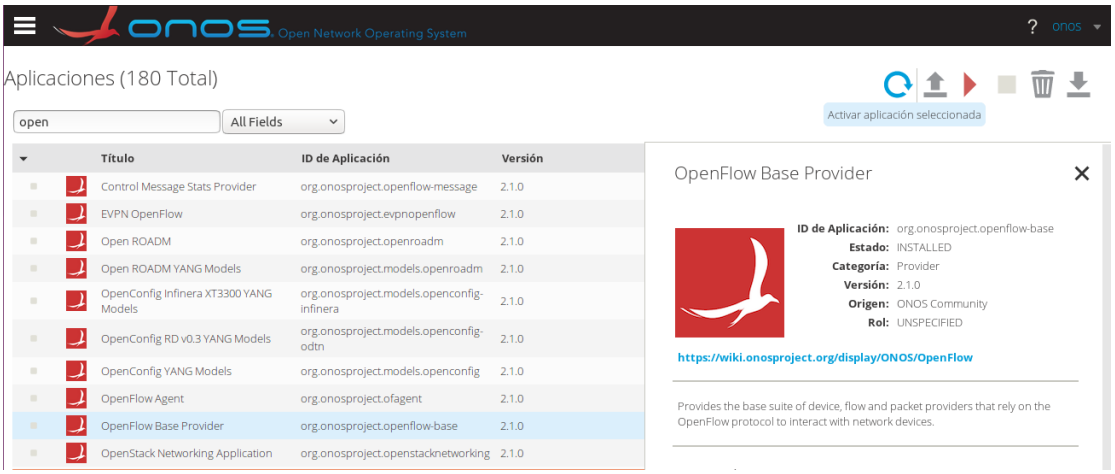


Figura 5.31. Sección de aplicaciones en ONOS – Activación de la aplicación “OpenFlow Base Provider”

Para verificar que todo haya quedado bien, se puede realizar una prueba sencilla:

- Con el comando “sudo mn --controller=remote,ip=10.0.2.15 --topo=tree,2,2” se emula una red (poner en “ip” la dirección del controlador onos).

- Luego desde la CLI de onos ejecutamos el comando "cfg set org.onosproject.provider.of.device.impl.OpenFlowDeviceProvider portStatsPollFrequency 10" con el cual estamos ajustando un tiempo de sondeo de 100 ms (0,1 s), y para comprobar que efectivamente haya quedado este tiempo ejecutamos el comando "portstats -d" el cual muestra estadísticas de la red en el último intervalo de sondeo y adicionalmente muestra el tiempo de este intervalo. Esto se puede observar en la Figura 5.32.

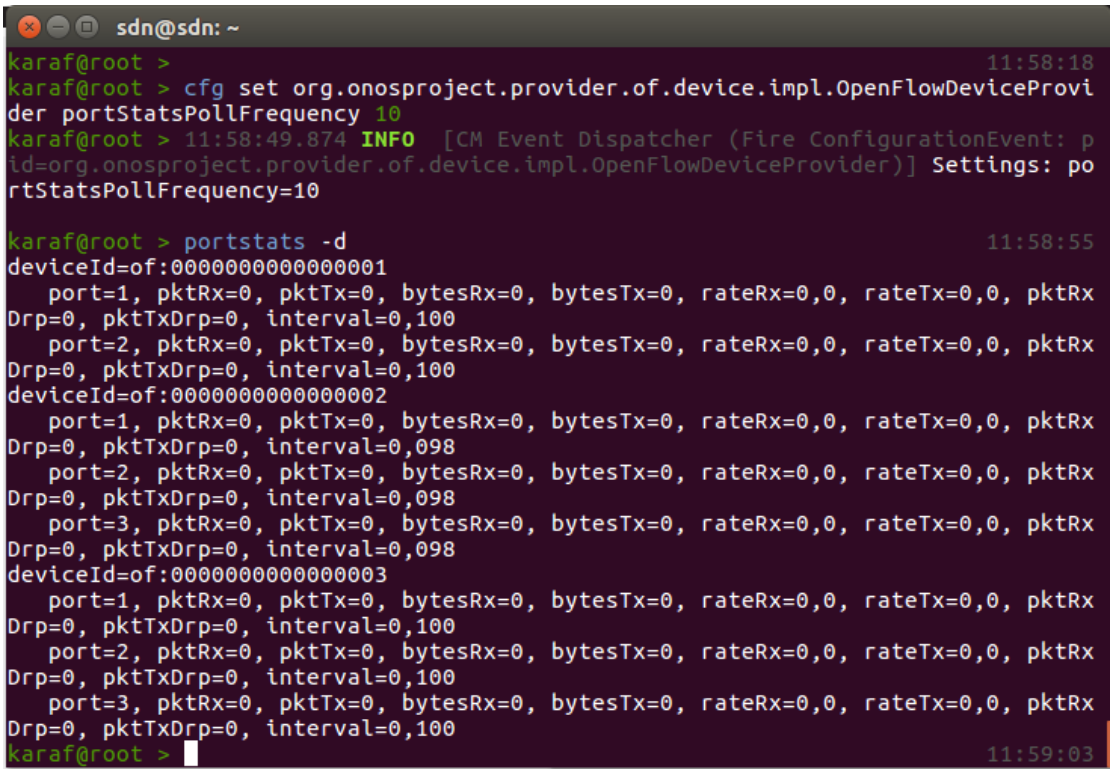


Figura 5.32. Comando "portstats -d" en la CLI de ONOS

5.5 Escalado de datos

Según pruebas realizadas en ONOS al ajustar un intervalo de sondeo muy pequeño este tiende a ser muy inestable, es a partir de 100 ms que dicho intervalo se hace un poco más estable ya que muestra una precisión de (100 ± 2) ms, esto se puede observar en la Figura 5.32.

Esta es una limitación del controlador ONOS ya que no nos permite tener intervalos de sondeo muy pequeños, debido a esto para la realización de las pruebas se hará un escalado de los datos.

Para esto decimos que el tiempo emulado (t_E) es directamente proporcional al tiempo real (t_R):

$$t_E = x \cdot t_R \tag{5.1}$$

En ese orden de ideas si tenemos:

$$T_R = \left[\frac{bits}{s}\right]$$

Si $t_R = 1\ s$ entonces $t_E = x\ s$, lo cual hace que:

$$T_E = \left[\frac{bits}{x\ s}\right] = \frac{1}{x} \cdot \left[\frac{bits}{s}\right] = \frac{1}{x} \cdot T_R \tag{5.2}$$

Donde T_R es la tasa de bits real y T_E es la tasa de bits emulada.

Para evaluar las capacidades del algoritmo se pensó en un tiempo de sondeo de 10 ms (T_R), pero debido a la limitación que posee el controlador ONOS mencionada anteriormente, no es posible tener este tiempo de sondeo. En ese orden de ideas se realiza un escalado de los datos definiendo $T_E = 100\ ms$, conociendo estos valores despejamos x de la ecuación 5.1, de esta manera tenemos que $x = 10$, ya conociendo este valor es posible determinar la tasa de bits que será emulada (T_E), por medio de la ecuación 5.2.

5.6 Topología

La topología usada para la implementación del algoritmo del volcado del tráfico de datos se puede observar en la Figura 5.33. Esta es una topología en estrella, la cual cuenta con dos hosts, un servidor, y un switch que es controlado por el controlador ONOS.

Como se mencionó antes, cada host cuenta con dos enlaces de comunicación cuyo acceso está determinado por dos puertos diferentes, además uno de los enlaces cuenta con un mayor ancho de banda que el otro.

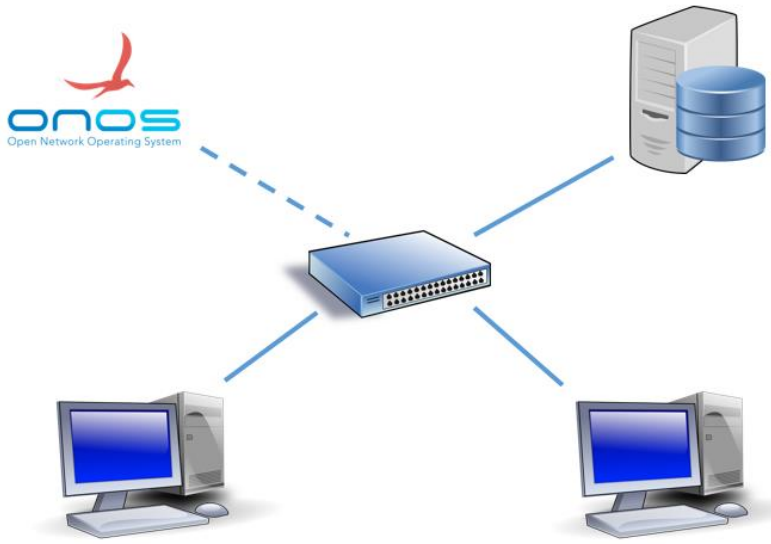


Figura 5.33. Topología

5.6.1 Implementación en mininet

Para realizar las pruebas pertinentes de emulación se requiere crear una topología de red en mininet como la de la Figura 5.33, para esto se hace uso del lenguaje de programación Python. A continuación, se evidencian las secciones del código más importantes (en la sección de anexos podrá encontrar el código completo):

- *Agregar controlador remoto:* Es necesario agregar un controlador remoto a la red, indicando su dirección IP, el protocolo de comunicación y el puerto que este utiliza. En la Figura 5.34 se puede apreciar cómo se agrega un controlador ONOS a la red.

```
19    info( '*** Adding controller\n' )
20    c0=net.addController(name='c0',
21                         controller=RemoteController,
22                         ip='10.0.2.15',
23                         protocol='tcp',
24                         port=6633)
```

Figura 5.34. Agregar controlador ONOS en la red emulada

- *Agregar enlaces*: En la red es necesario definir dos canales de comunicación con diferentes anchos de banda, en primer lugar, se asigna el ancho de banda deseado para los enlaces (Link1 y Link2), y finalmente se indica los puntos extremos (dispositivos) en los que se encuentra ubicado cada enlace. En la Figura 5.35 se muestra el código que permite agregar dichos enlaces.

```
36    info( '*** Add links\n')
37    Link1 = {'bw':3}
38    Link2 = {'bw':6}
39    # HOSTS - SWITCH
40    net.addLink(s3, s1, cls=TCLink, **Link1)
41    net.addLink(s3, s1, cls=TCLink, **Link2)
```

Figura 5.35. Definición de los enlaces - anchos de banda

Como se puede apreciar en la Figura 5.35, se definen dos canales de comunicación con anchos de banda de 3 Mbps y 6 Mbps. Las razones para escoger dichos anchos de banda se deben en gran medida a la necesidad de saturar el canal de comunicación, dadas las tasas de transmisión seleccionadas; En la sección 7.1 se profundizará en los detalles de las tasas de transmisión, además de otras consideraciones para las posteriores pruebas.

6. Implementación física

6.1 OpenvSwitch

Para la implementación física, se hace necesario configurar un switch que admita el protocolo OpenFlow, ya que este permite la comunicación con el controlador y de esta manera es posible configurar una red SDN funcional. En ese orden de ideas, se realizará la implementación de un switch virtual en una Raspberry Pi 3 de la siguiente manera:

- Se instala OpenvSwitch, mediante el comando "sudo apt install openvswitch-switch openvswitch-common bridge-utils".
- Para la creación del switch, es necesario conocer primero las diferentes interfaces de red además de los nombres que poseen en la Raspberry Pi 3. En la Figura 6.1, es posible apreciar las diferentes interfaces, donde:
  - "enxb827eb8c7c25": Interfaz de ethernet.
  - "lo": Interfaz de red virtual.
  - "wlan0": Tarjeta de red inalámbrica.

```
sdn@sdn:~$ ifconfig
enxb827eb8c7c25 Link encap:Ethernet  HWaddr b8:27:eb:8c:7c:25
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:280 errors:0 dropped:0 overruns:0 frame:0
          TX packets:280 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:20954 (20.9 KB)  TX bytes:20954 (20.9 KB)

wlan0     Link encap:Ethernet  HWaddr b8:27:eb:d9:29:70
          inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::96d2:e739:6034:4ca2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:145 errors:0 dropped:40 overruns:0 frame:0
          TX packets:118 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:38769 (38.7 KB)  TX bytes:15974 (15.9 KB)
```

Figura 6.1. Interfaces de red

- Posteriormente se crea un enlace o puente en el switch y se habilita esta nueva "interfaz", en este caso se le da el nombre de "sw", y finalmente verificamos su correcta creación. En la Figura 6.2, es posible observar los comandos necesarios para realizar la configuración descrita anteriormente.
- Se añade el puerto ethernet identificado en la Figura 6.1, luego se asigna la IP con mascara de 24 bits al switch, se deshabilita el puerto ethernet para que no entre en conflicto con el puente, y se le atribuye su correspondiente dirección física. Finalmente se verifica la correcta agregación del puerto ethernet a "sw", en la Figura 6.3 se evidencian los comandos que permiten agregar dicha interfaz.

```
sdn@sdn:~$ sudo ovs-vsctl add-br sw
sdn@sdn:~$ sudo ifconfig sw up
sdn@sdn:~$ sudo ovs-vsctl show
372a02a8-7c85-4e25-a3a5-92b4ea524d72
    Bridge sw
        Port sw
            Interface sw
                type: internal
    ovs_version: "2.5.5"
```

Figura 6.2. Creación de un nuevo enlace o puente

```
sdn@sdn:~$ sudo ovs-vsctl add-port sw enxb827eb8c7c25
sdn@sdn:~$ sudo ifconfig sw 192.168.0.4/24
sdn@sdn:~$ sudo ifconfig enxb827eb8c7c25 0
sdn@sdn:~$ sudo ifconfig sw hw ether b8:27:eb:8c:7c:25
sdn@sdn:~$ sudo ovs-vsctl show
372a02a8-7c85-4e25-a3a5-92b4ea524d72
    Bridge sw
        Port "enxb827eb8c7c25"
            Interface "enxb827eb8c7c25"
        Port sw
            Interface sw
                type: internal
    ovs_version: "2.5.5"
```

Figura 6.3. Interfaz Ethernet agregada

- Para añadir la interfaz inalámbrica es necesario descargar un editor de redes inalámbricas, en este caso se seleccionó el editor "plasma", para realizar la instalación de dicho editor se hace uso del comando "sudo apt-get install plasma-nm" y se ejecuta mediante el comando "kde5-nm-connection-editor".
- En la Figura 6.4 se puede apreciar la interfaz gráfica que posee el editor, en donde es necesario añadir una nueva conexión wifi (compartida).

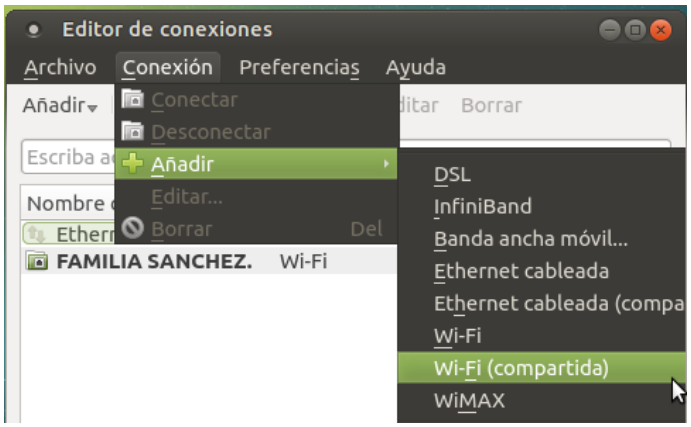


Figura 6.4. Editor de redes inalámbricas

- Una vez creada la red wifi en la Raspberry, es necesario conectarse a dicha red, esto con el fin de asociarla a la tarjeta de red inalámbrica de la Raspberry, además de hacerla visible para otros dispositivos.

- Se añade el puerto correspondiente a la red inalámbrica identificado en la Figura 6.1, además de ello se deshabilita el puerto para que no entre en conflicto con el puente, y se le atribuye su correspondiente dirección física, dicha configuración se realiza mediante los comandos evidenciados en la Figura 6.5. Finalmente se verifica la correcta agregación del puerto de la red inalámbrica a "sw", en la Figura 6.6 se observa la correcta agregación de la interfaz de red inalámbrica.

```
sdn@sdn:~$ sudo ovs-vsctl add-port sw wlan0
sdn@sdn:~$ sudo ifconfig wlan0 0
sdn@sdn:~$ sudo ifconfig sw hw wlan0 b8:27:eb:d9:29:70
```

Figura 6.5. Interfaz de red inalámbrica agregada

```
sdn@sdn:~$ sudo ovs-vsctl show
372a02a8-7c85-4e25-a3a5-92b4ea524d72
    Bridge sw
        Port "enxb827eb8c7c25"
            Interface "enxb827eb8c7c25"
        Port "wlan0"
            Interface "wlan0"
        Port sw
            Interface sw
                type: internal
    ovs_version: "2.5.5"
```

Figura 6.6. Interfaz de red inalámbrica agregada

```
sdn@sdn:~$ ifconfig
enxb827eb8c7c25 Link encap:Ethernet  HWaddr b8:27:eb:8c:7c:25
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:344 errors:0 dropped:0 overruns:0 frame:0
          TX packets:344 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:24794 (24.7 KB)  TX bytes:24794 (24.7 KB)

sw        Link encap:Ethernet  HWaddr b8:27:eb:8c:7c:25
          inet addr:192.168.0.5  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::1019:f7ff:feba:8c4f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:38 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:6558 (6.5 KB)

wlan0     Link encap:Ethernet  HWaddr b8:27:eb:d9:29:70
          inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::96d2:e739:6034:4ca2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:306 errors:0 dropped:40 overruns:0 frame:0
          TX packets:195 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:70369 (70.3 KB)  TX bytes:23800 (23.8 KB)
```

Figura 6.7. Resultado final al crear el puente o enlace

- Finalmente, mediante el comando “ifconfig” es posible validar la correcta creación del puente con el nombre “sw”, en la Figura 6.7 se aprecia un ejemplo de dicho resultado.

Para conectar el switch configurado anteriormente con el controlador se hace uso del comando “sudo ovs-vsctl set-controller sw tcp: 192.168.0.7:6633” especificando la respectiva dirección ip asociada al controlador, además del puerto de escucha generalmente puede utilizar el puerto “6633” o el puerto “6653”.

6.2 Topología red SDN

Una vez se ha configurado exitosamente el switch virtual en la Raspberry Pi 3, además de tener instalado y configurado el controlador en un computador, se dispone entonces de los elementos básicos para configurar una red SDN, pero se hace necesario disponer de un servidor DHCP para el correcto funcionamiento del controlador ONOS, en ese orden de ideas la topología física que se implementará se puede apreciar en la Figura 6.8.

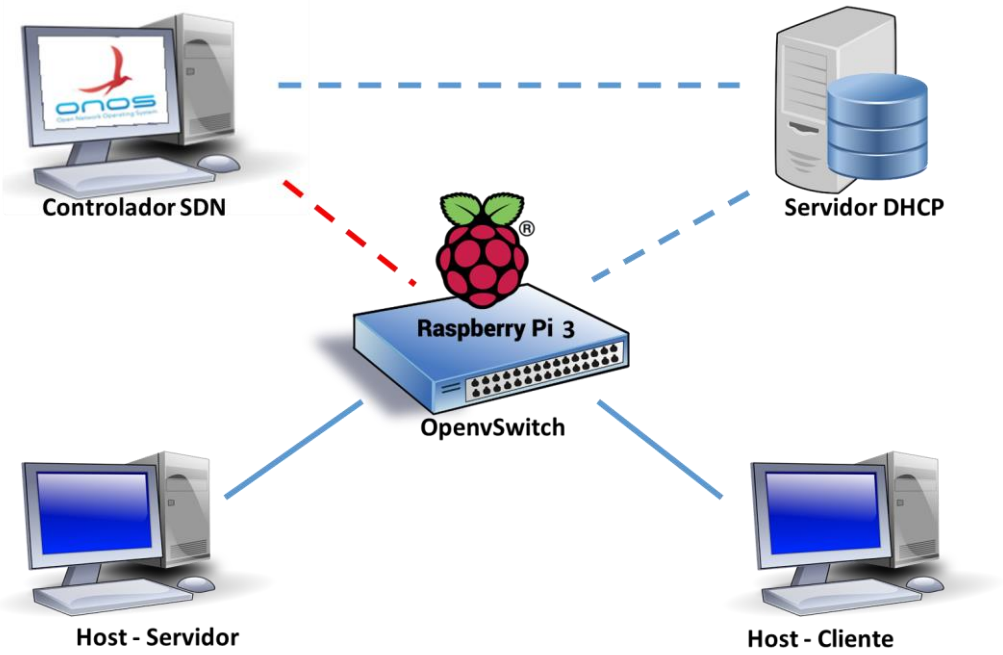


Figura 6.8. Topología de la red SDN implementada

En la Figura *6.8* se puede observar un enlace en rojo, este es un enlace virtual ya que desde OpenvSwitch se hace un puente entre el controlador y el mismo. Además, el host – servidor tiene un enlace wifi con OpenvSwicth y el host – cliente tiene dos enlaces ethernet.

Como bien se sabe el switch virtual u OpenvSwitch será configurado en una Raspberry Pi 3, pues esta supone la implementación de bajo costo más factible dadas las

necesidades que se tienen con la red planteada anteriormente. En la Figura 6.9 se puede apreciar el montaje final del dispositivo, provisto de dos adaptadores USB a Ethernet.

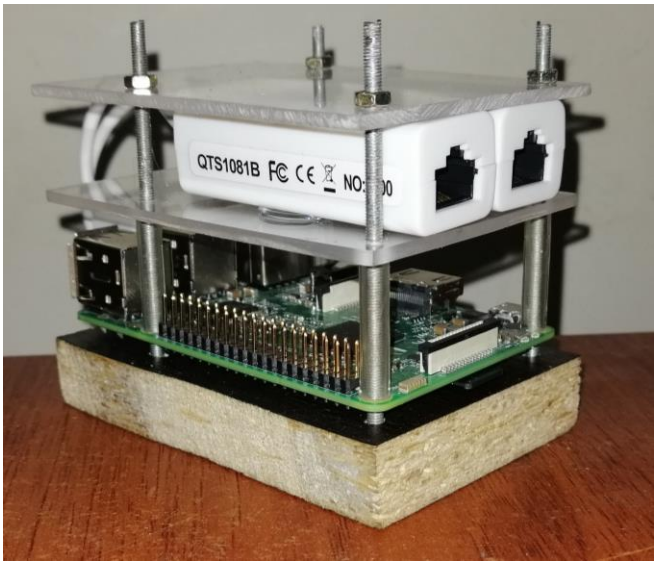


Figura 6.9. Raspberry Pi 3 – OpenvSwitch

6.3 Tráfico generado

Se plantea realizar la transmisión de un video mediante el software VLC desde el host - servidor hacia todos los nodos de la red (transmisión Broadcast). Por otro lado, es necesario transmitir dicho video mediante el protocolo UDP, debido a que el host – cliente posee dos interfaces ethernet (cada una con una IP asociada diferente). Esto implica que una transmisión mediante el protocolo TCP impediría realizar el volcado del tráfico de datos y a su vez mantener el flujo de información hacia el host – cliente.

Para realizar la transmisión del video mediante VLC, se realizan los siguientes pasos:

- Configuración del host – servidor:
  - Se selecciona “Emitir”, posteriormente se elige el archivo de video que se desea transmitir.
  - Se selecciona en método de trasmisión UDP (Legacy).
  - Se pone en dirección de destino la IP broadcast *192.168.1.255* y el puerto *1234.*
  - Se deshabilita la “transcodificacion” y se habilita “emitir de todas las emisiones elementales”
  - Finalmente se selecciona “Emitir”.

- Configuración del host – cliente:
  - Se selecciona “Abrir ubicación de red” y en la URL se pone *udp://@:1234*
  - Finalmente se da clic en “Reproducir”

7. Resultados

7.1 Consideraciones previas

Para evaluar las capacidades del algoritmo del volcado del tráfico de datos, se tendrá en cuenta el caso de aplicación de IPTV (Televisión sobre IP), tomando como base dos diferentes estándares de calidad:

- *Definición estándar (SD):* Corresponde a una resolución de hasta 576i y una tasa de transmisión entre 1.75 Mbps y 5 Mbps.
- *Alta definición (HD):* Corresponde a una resolución de hasta 1080i y una tasa de transmisión entre 4 Mbps y 12 Mbps.

Cabe resaltar que los datos anteriormente expuestos están relacionados con los parámetros de rendimiento provisionales seleccionados por la Unión Internacional de Telecomunicaciones [34].

Por otro lado para definir ciertos criterios en cuanto a la calidad del servicio (QoS) en IPTV, la Unión Internacional de Telecomunicaciones plantea una serie de restricciones o requisitos de calidad para servicios de IPTV, en su recomendación G.1080 (12/08) actualmente vigente [34], se resalta:

- El retardo máximo aceptable para IPTV es de 100 ms y se conoce como retardo de transferencia de paquetes IP (IPTD, IP Packet Transfer Delay).
- Un jitter menor o igual a 50 ms.

Con base en la información anterior se tomó la decisión de emular diferentes tráficos, además de diseñar una serie de pruebas, cuyas características se pueden apreciar en la tabla 7.1, cabe destacar que dichas pruebas serán evaluadas con tráficos TCP y UDP.

Tabla 7.1. Pruebas diseñadas

No. Prueba	Tasa de transmisión (Mbps)	Retardo máximo tolerado (ms)	Umbral throughput - Puerto 1 (Mbps)
1	5	100	3.0
2	2	100	3.0
3	5	100	2.7
4	2	100	2.7
5	5	100	2.4
6	2	100	2.4

Antes de ver los resultados obtenidos de las pruebas diseñadas, es interesante ver qué sucede con la red cuando el algoritmo no se encuentra funcionando y se crean las condiciones necesarias para realizar un cambio de puerto (tasa de transmisión superior al ancho de banda del canal de comunicaciones). En la Figura 7.1 y en la Figura 7.2, es posible evidenciar como el tráfico es limitado al ancho de banda disponible en el canal de comunicaciones asociado al puerto 1 (3 Mbps).

Es importante resaltar que para el caso de la Figura 7.1 (reporte generado desde wireshark), los datos se encuentran escalados tal como se mencionó anteriormente, es por ello que el tráfico se encuentra alrededor de 300 kbps y no de 3 Mbps, mientras que en el caso de la Figura 7.2, los datos han sido des-escalados y se presentan de forma más acorde con las pruebas diseñadas.

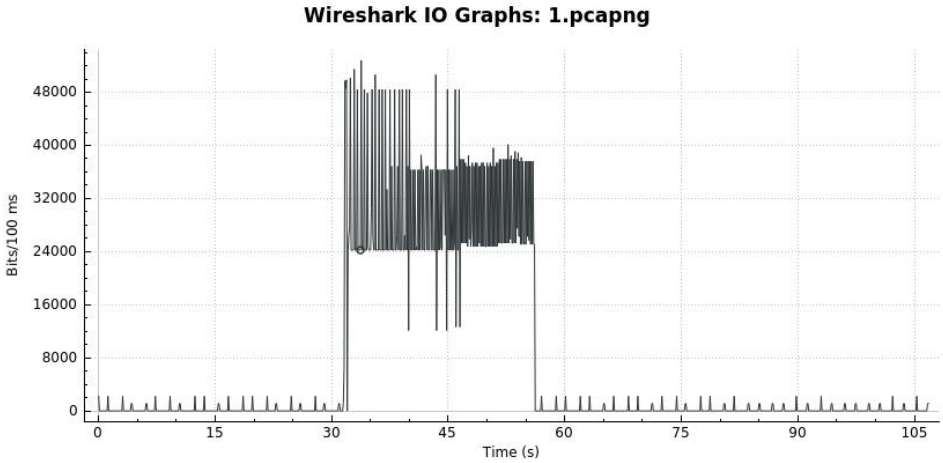


Figura 7.1. Host 1 – Throughput – Reporte de Wireshark

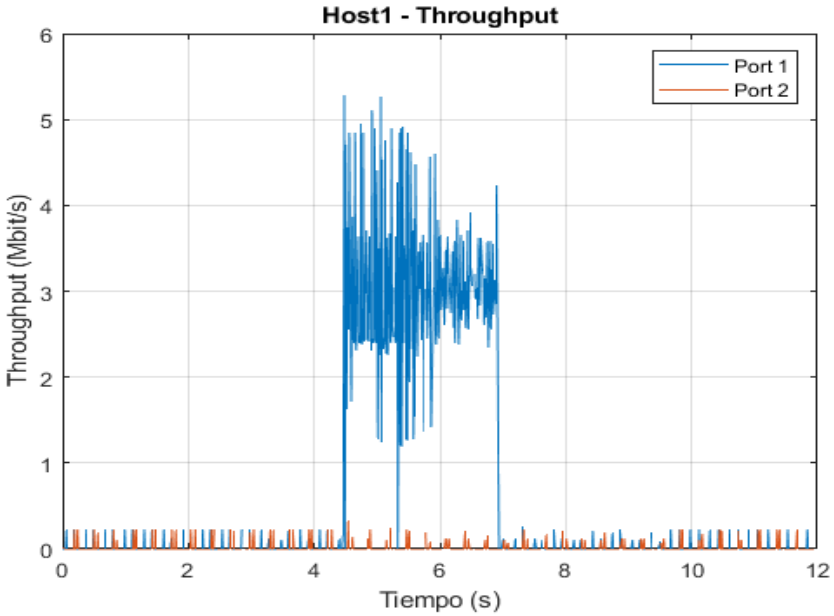


Figura 7.2. Host 1 – Throughput – Reporte de ONOS

7.2 Tráfico UDP

Con el fin de evaluar las capacidades del algoritmo del volcado del tráfico de datos, se realizaron 6 pruebas con tráfico UDP (ver tabla 7.1), además estas pruebas se replicaron modificando los parámetros K_p y K_c del algoritmo. Estas pruebas se pueden observar de la Figura 7.3 a la 7.20.

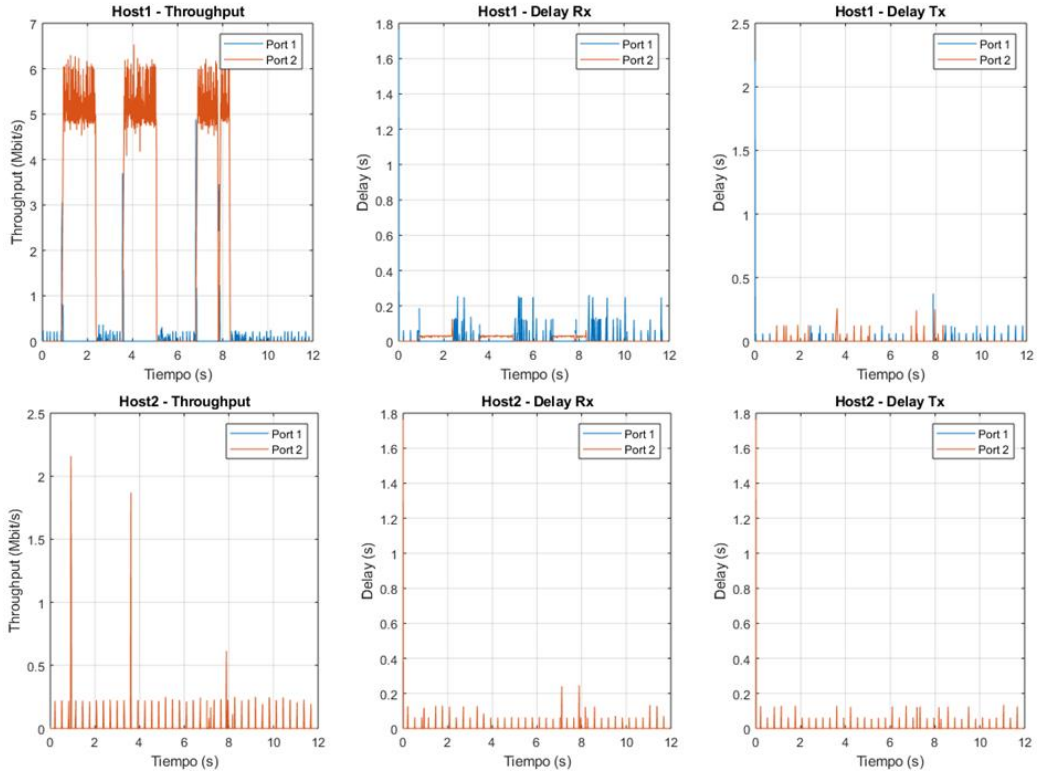


Figura 7.3. Prueba 1 con $K_p = 1$ y $K_c = 4$

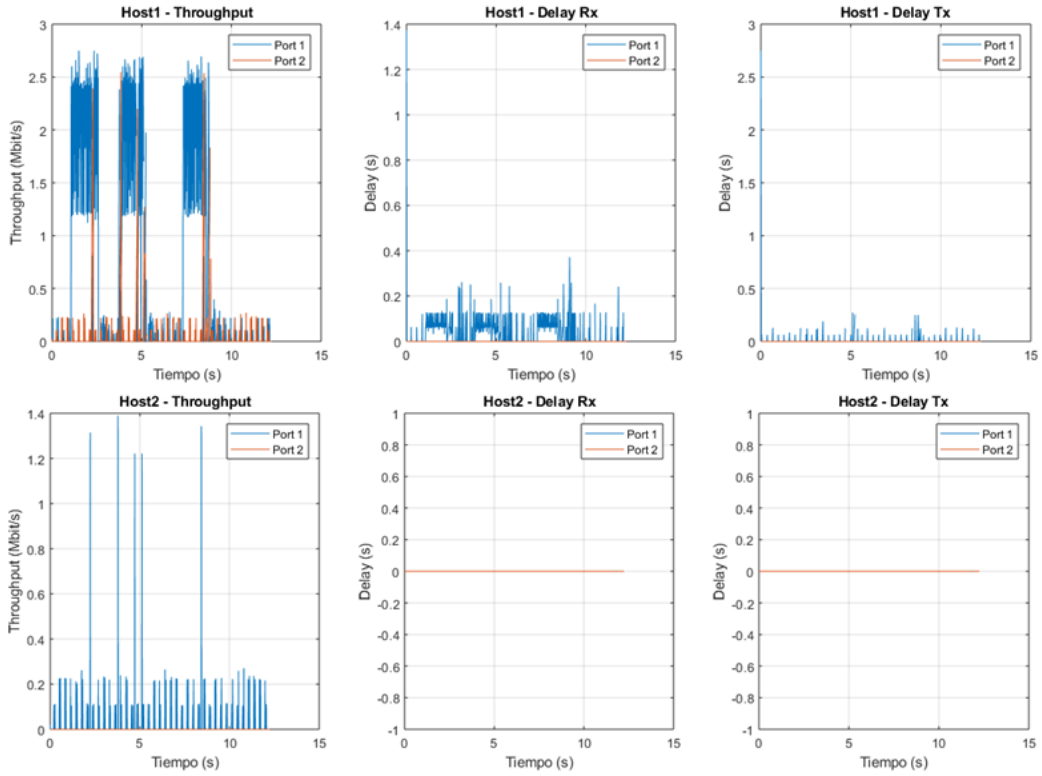


Figura 7.4. Prueba 2 con $K_p = 1$ y $K_c = 4$

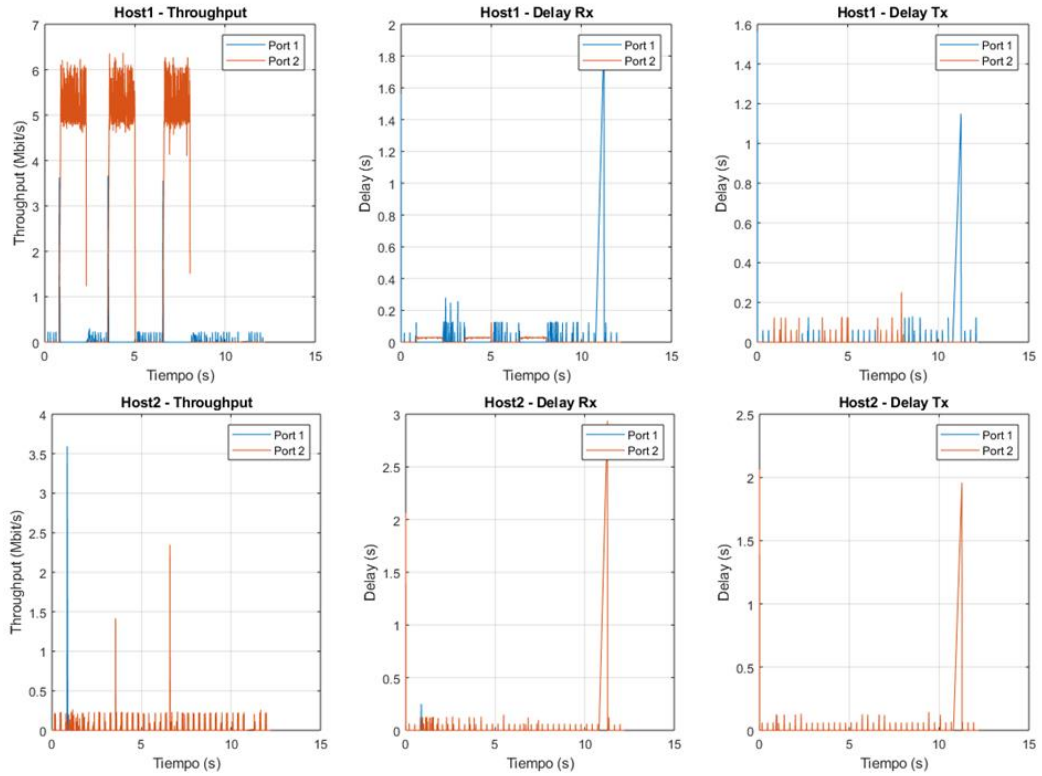


Figura 7.5. Prueba 3 con $K_p = 1$ y $K_c = 4$

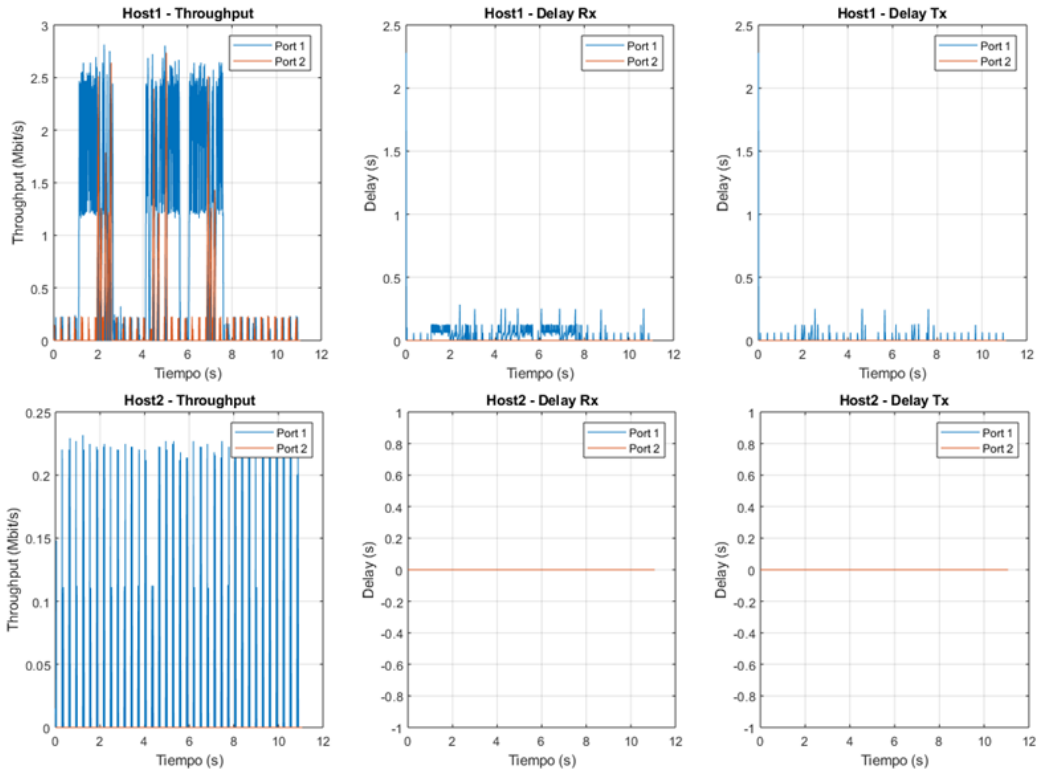


Figura 7.6. Prueba 4 con $K_p = 1$ y $K_c = 4$

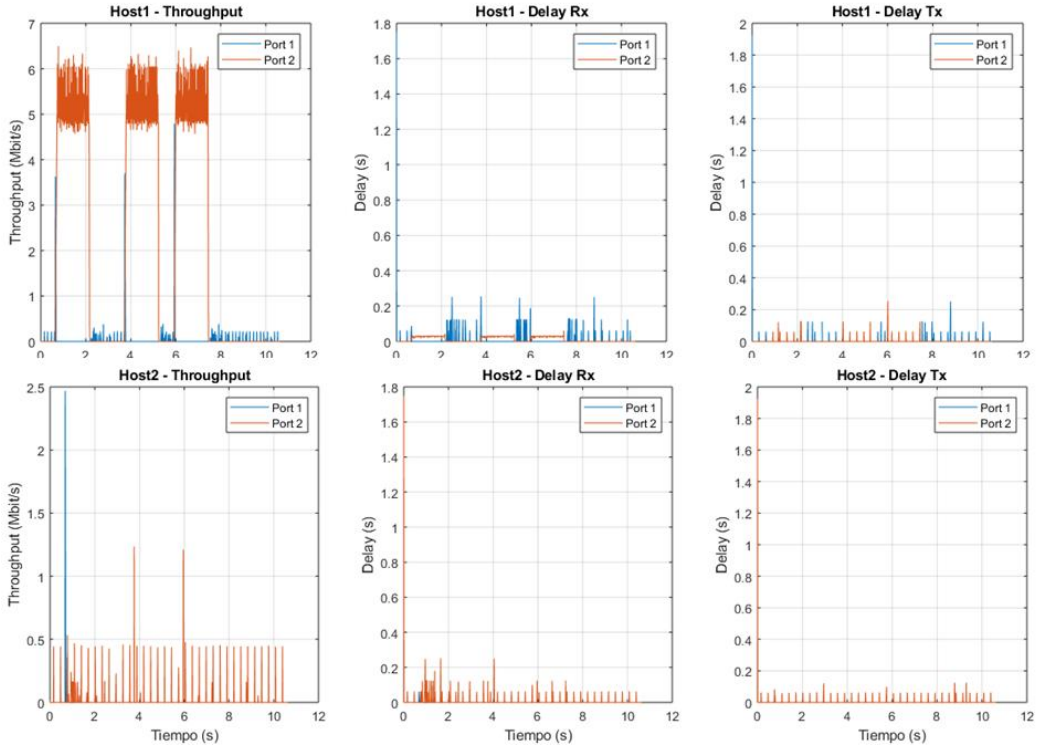


Figura 7.7. Prueba 5 con $K_p = 1$ y $K_c = 4$

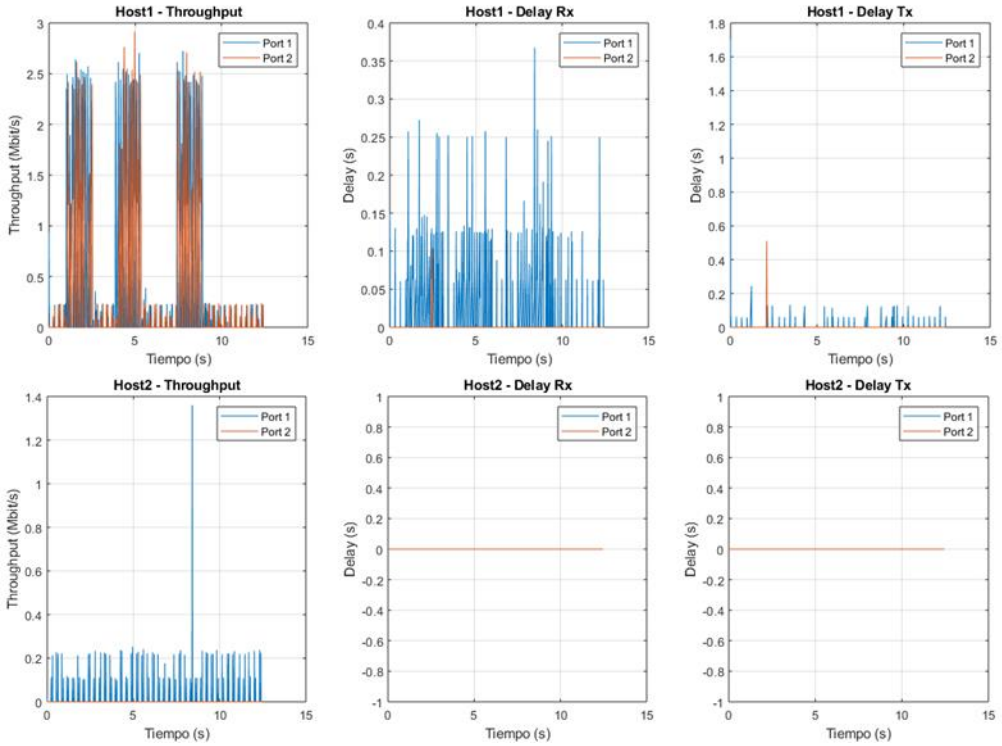


Figura 7.8. Prueba 6 con $K_p = 1$ y $K_c = 4$

Al observar de la Figura 7.3 a la 7.8, se puede apreciar que en algunos casos los delays superan el límite establecido de 100 ms, pero como el throughput no supera la mitad del ancho de banda del canal de comunicaciones, el algoritmo no realiza el cambio de puerto ya que, si se realiza dicho cambio sería innecesario.

Además, en la Figura 7.8, es posible apreciar una conmutación constate entre el puerto 1 y el puerto 2, esto debido a que el tráfico generado es bastante variable y no mantiene una tasa de transmisión constante.

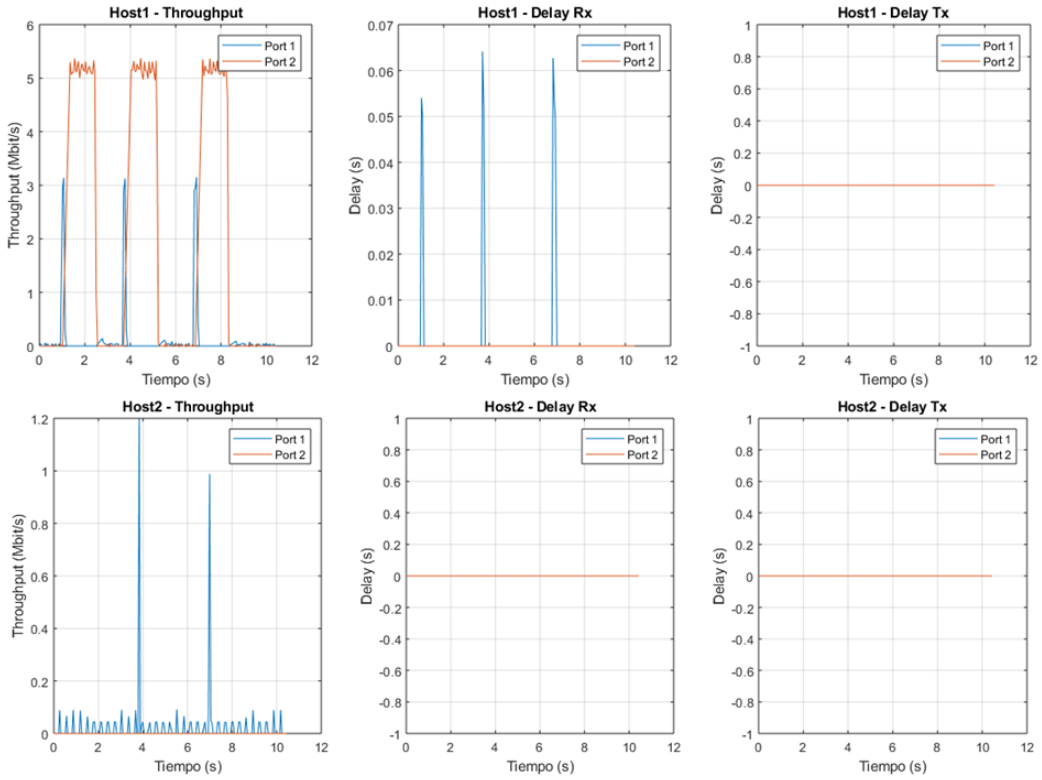


Figura 7.9. Prueba 1 con $K_p = 5$ y $K_c = 4$

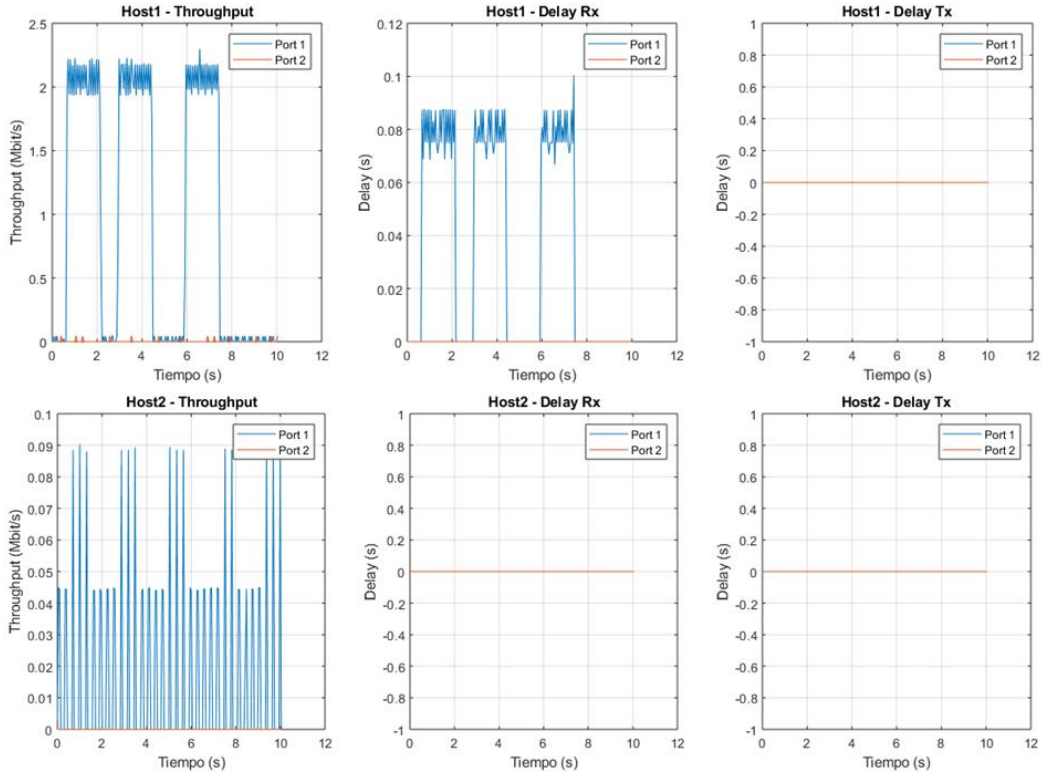


Figura 7.10. Prueba 2 con $K_p = 5$ y $K_c = 4$

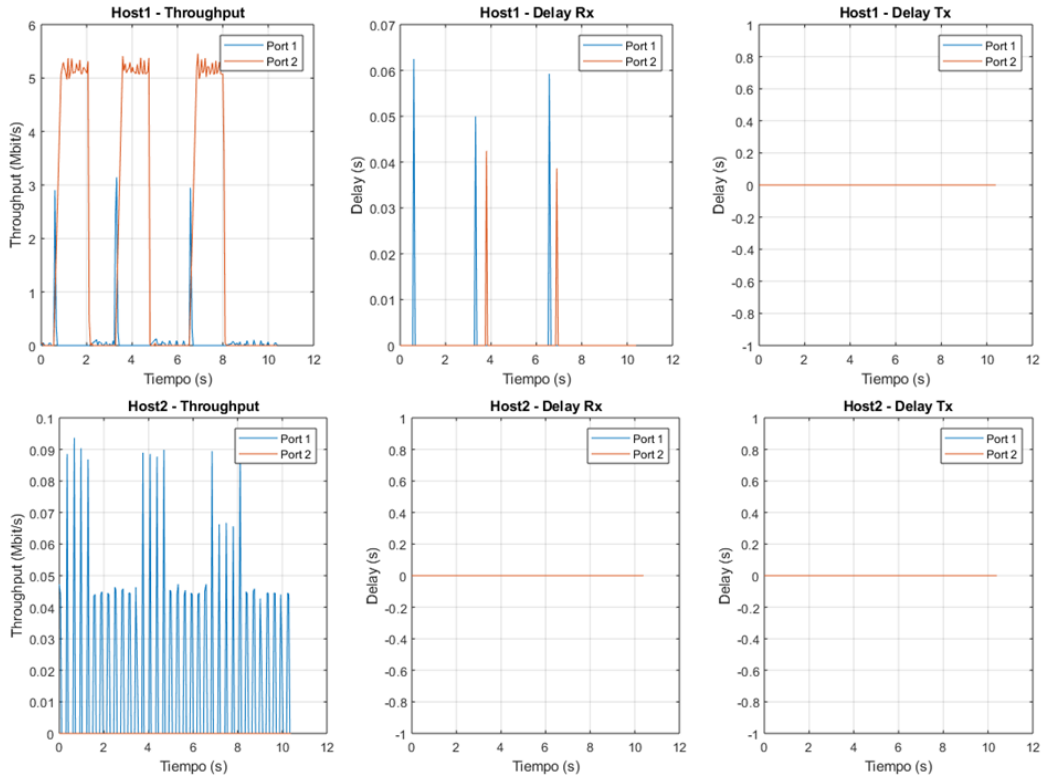


Figura 7.11. Prueba 3 con $K_p = 5$ y $K_c = 4$

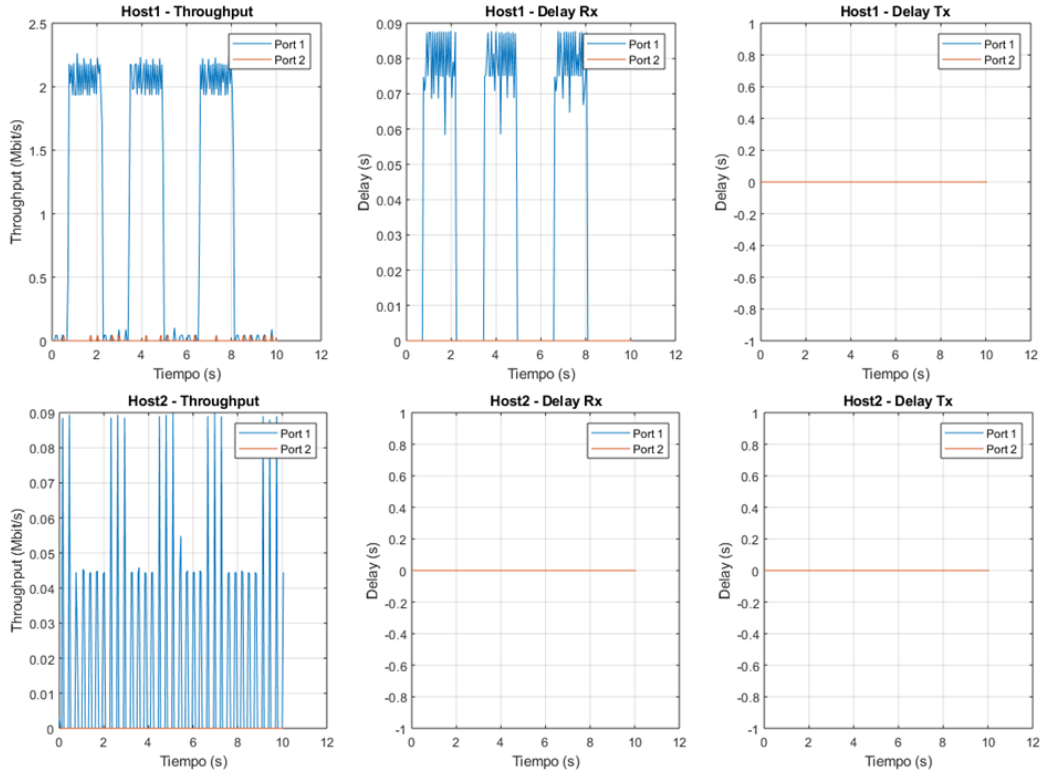


Figura 7.12. Prueba 4 con $K_p = 5$ y $K_c = 4$

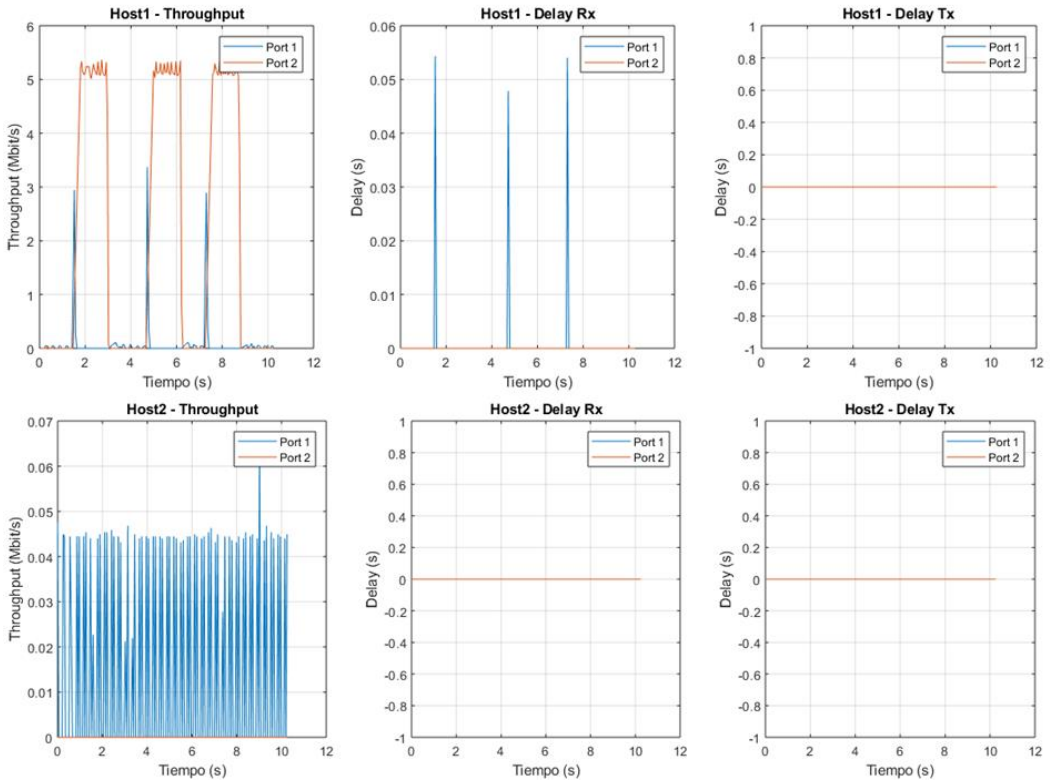


Figura 7.13. Prueba 5 con $K_p = 5$ y $K_c = 4$

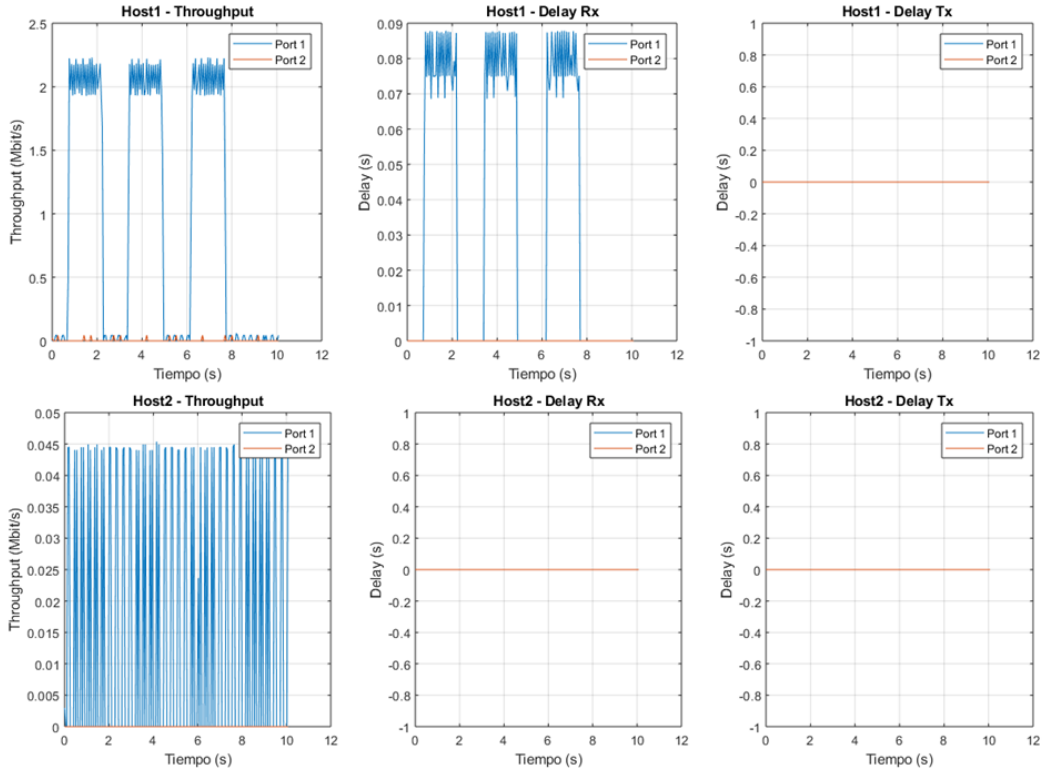


Figura 7.14. Prueba 6 con $K_p = 5$ y $K_c = 4$

Al observar de la Figura 7.9 a la 7.14, se puede apreciar como K_p actúa como un filtro para las señales, haciendo que el rizado de estas disminuya. Esto hace que el algoritmo se comporte de una mejor manera ya que esto evita que en un evento transitorio se produzca un cambio de puerto innecesario, además si comparamos la Figura 7.8 y la Figura 7.14, el aumentar K_p hizo que en esta prueba ya no quedara conmutando el puerto 1 y el puerto 2, debido a que la señal de throughput ahora es un poco más estable.

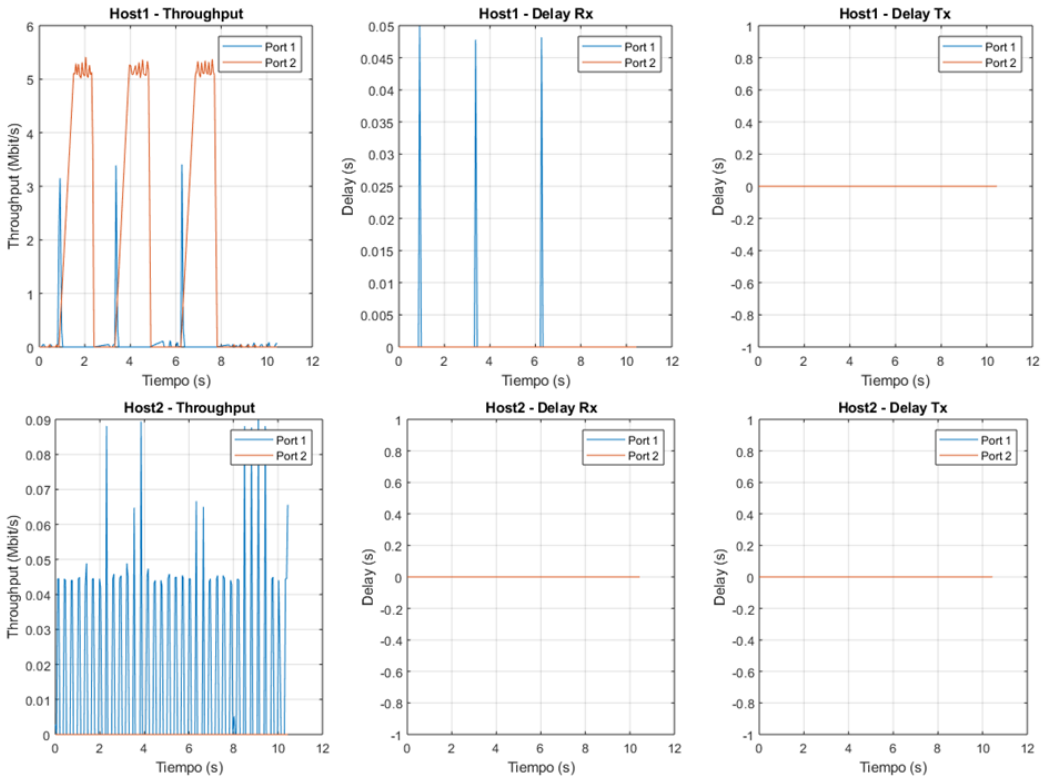


Figura 7.15. Prueba 1 con $K_p = 5$ y $K_c = 10$

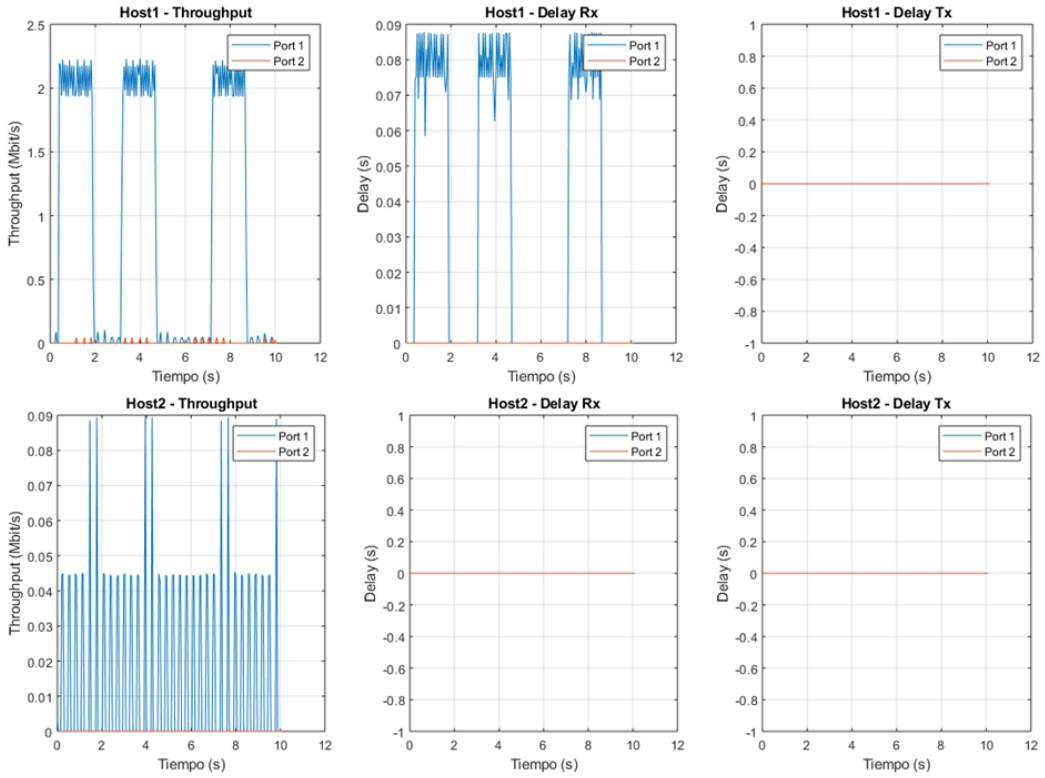


Figura 7.16. Prueba 2 con $K_p = 5$ y $K_c = 10$

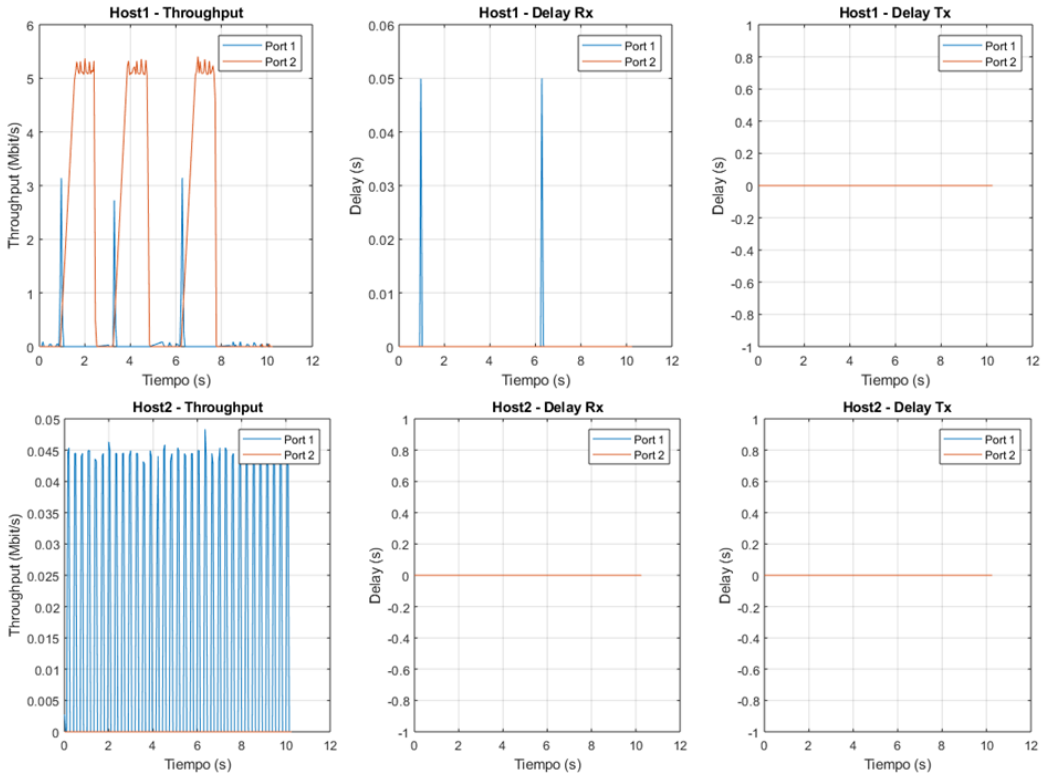


Figura 7.17. Prueba 3 con $K_p = 5$ y $K_c = 10$

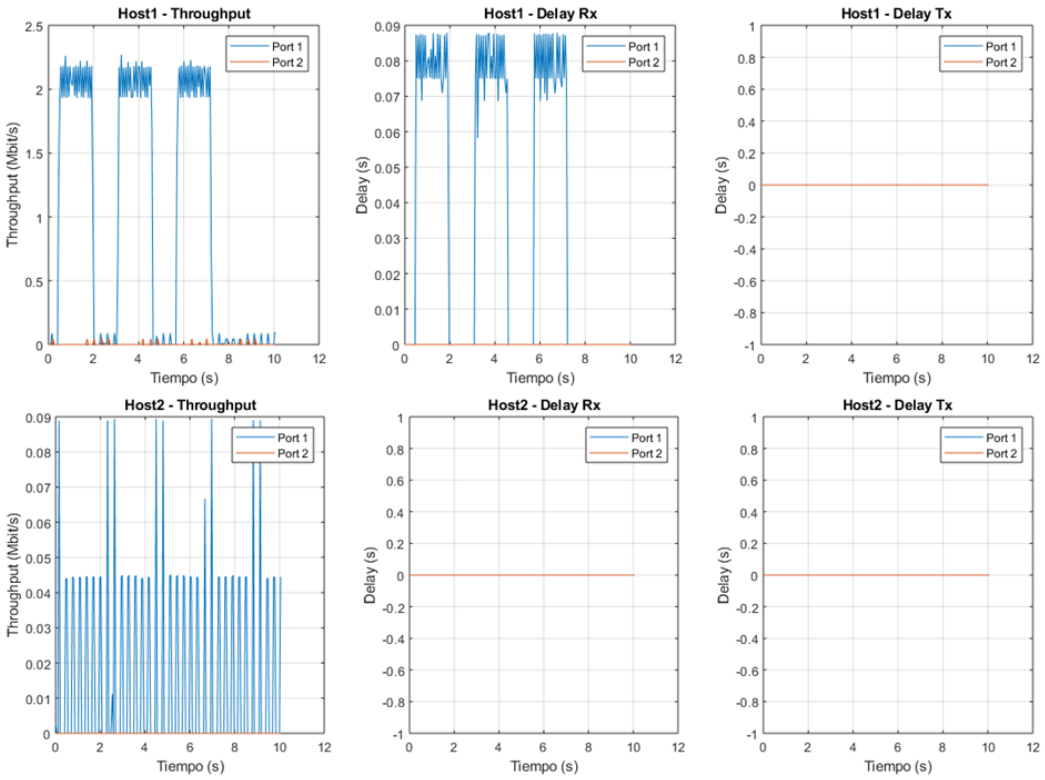


Figura 7.18. Prueba 4 con $K_p = 5$ y $K_c = 10$

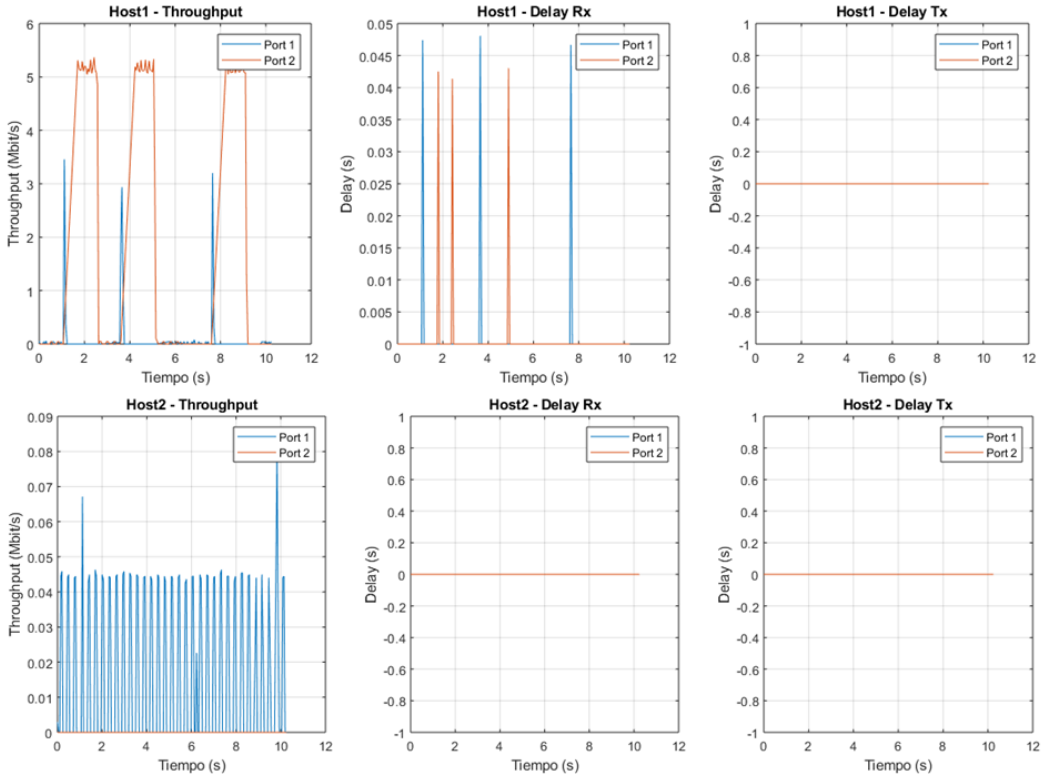


Figura 7.19. Prueba 5 con $K_p = 5$ y $K_c = 10$

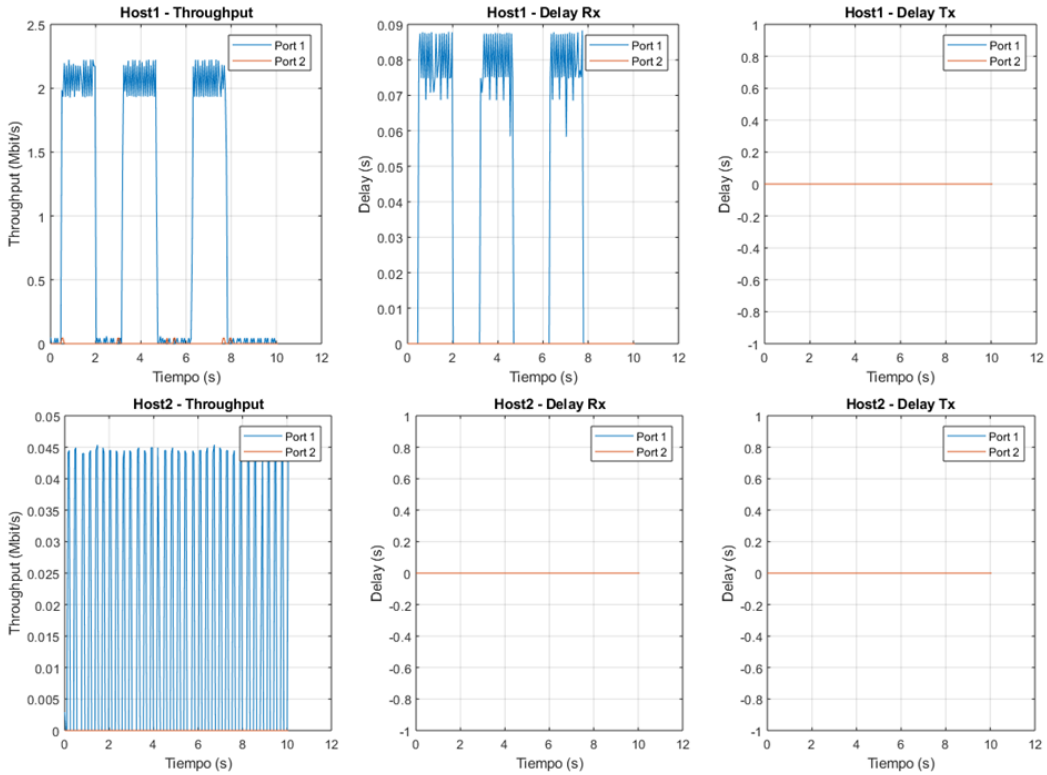


Figura 7.20. Prueba 6 con $K_p = 5$ y $K_c = 10$

Tabla 7.2. Paquetes perdidos – Pruebas con tráfico UDP

No. Prueba	K_p	K_c	Paquetes perdidos		
			Flujo 1	Flujo 2	Flujo 3
1	1	4	14/639	10/639	36/639
2			5/257	16/257	17/257
3			8/639	9/639	7/639
4			14/257	16/257	7/257
5			7/639	7/639	6/639
6			38/257	39/257	33/257
1	5	4	25/639	22/639	31/639
2			0/257	0/257	0/257
3			11/639	21/639	11/639
4			0/257	0/257	0/257
5			15/639	14/639	18/639
6			0/257	0/257	0/257
1	5	10	20/639	16/639	14/639
2			0/257	0/257	0/257
3			14/639	12/639	17/639
4			0/257	0/257	0/257
5			15/639	21/639	14/639
6			0/257	0/257	0/257

Si comparamos las figuras de la 7.15 a la 7.20 con las figuras de la 7.9 a la 7.14, se puede apreciar que el aumentar el K_c, hace que cuando se realiza un cambio de puerto el algoritmo ignora los datos sondeados sobre el host en el que se realizó el cambio durante el t_c (tiempo de cambio), lo cual previene que cuando se realice un cambio de puerto, este quede conmutando con el otro puerto debido a algún transitorio durante el cambio de puerto.

Con respecto a la cantidad de paquetes perdidos, en la Tabla 7.2 se evidencia como el comportamiento del algoritmo mejora considerablemente, cuando el límite del throughput en el canal de comunicaciones del Puerto 1 está por debajo del ancho de banda del canal, es decir cuando este límite está en 2.7 Mbps y 2.4 Mbps.

7.3 Tráfico TCP

También para evaluar las capacidades del algoritmo del volcado del tráfico de datos, se realizaron 6 pruebas con tráfico TCP (ver tabla 7.1), además estas se replicaron modificando los parámetros K_p y K_c del algoritmo. Estas pruebas se pueden observar de la Figura 7.21 a la 7.38.

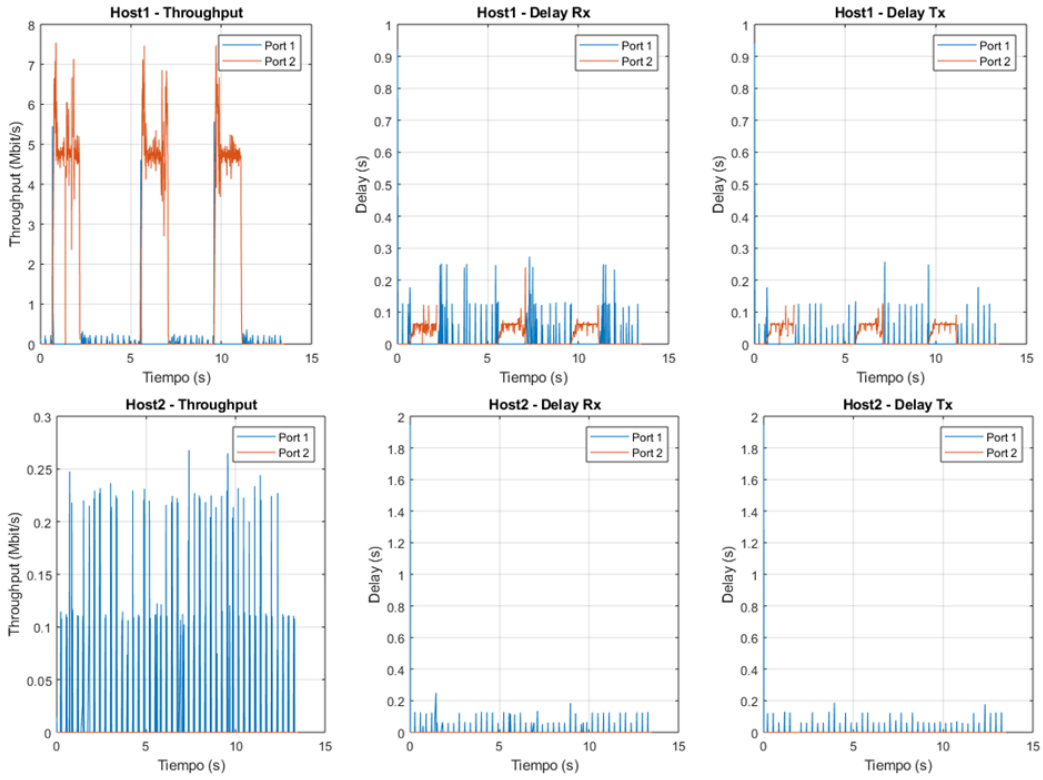


Figura 7.21. Prueba 1 con $K_p = 1$ y $K_c = 4$

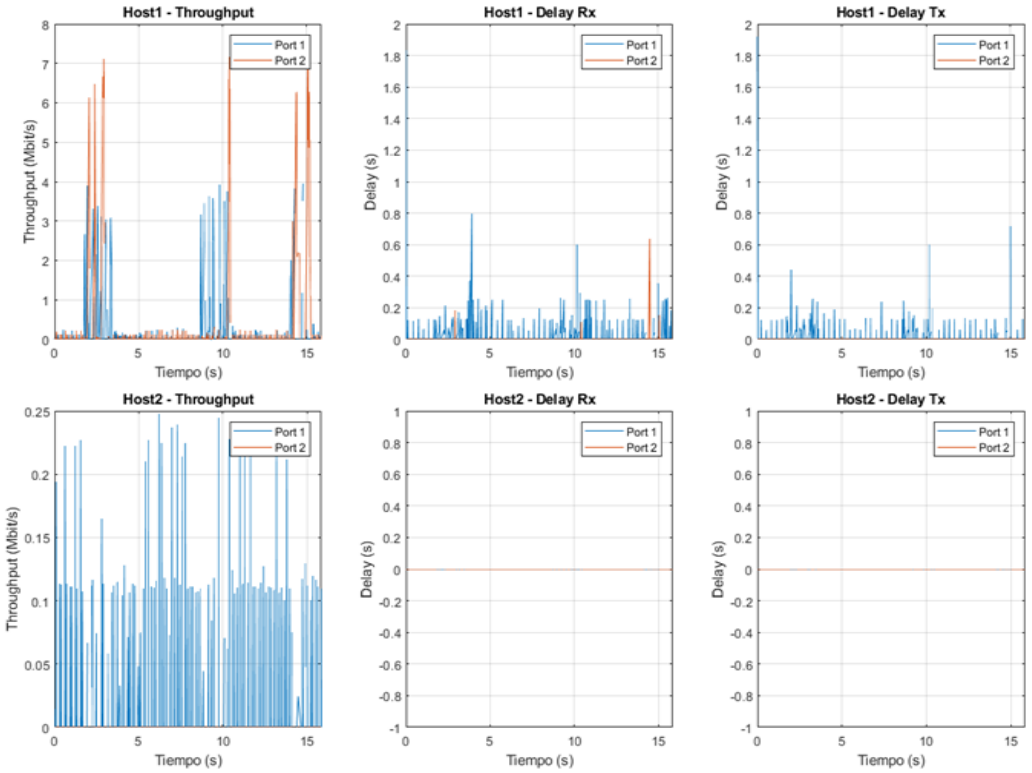


Figura 7.22. Prueba 2 con $K_p = 1$ y $K_c = 4$

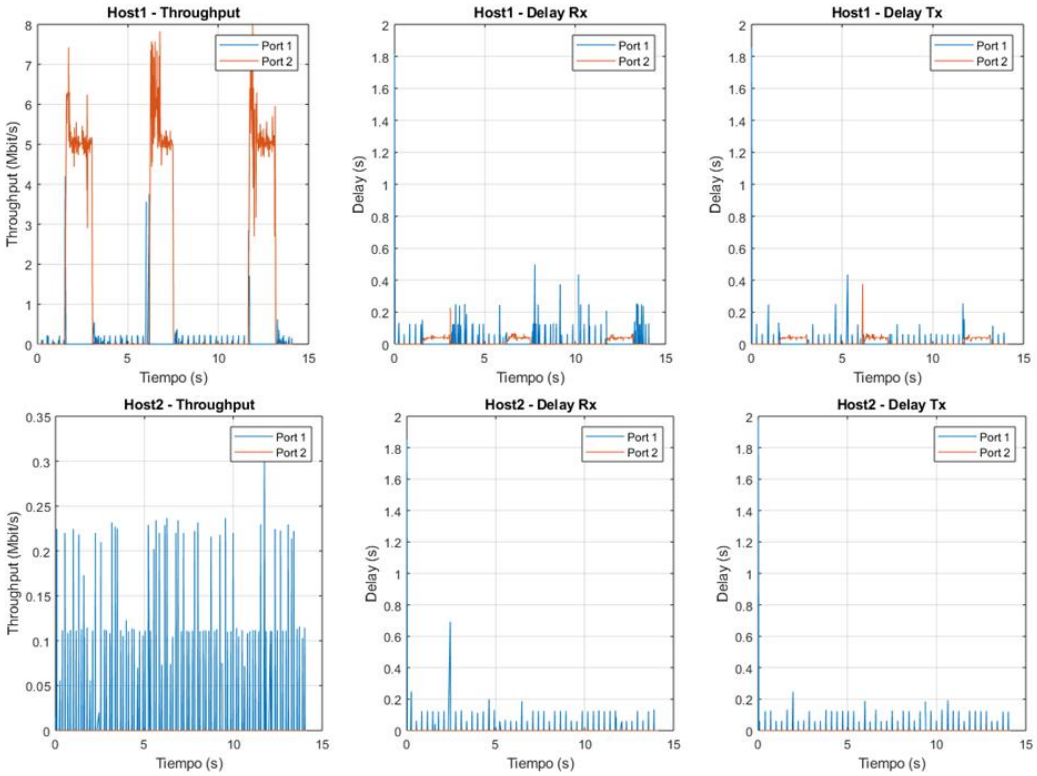


Figura 7.23. Prueba 3 con $K_p = 1$ y $K_c = 4$

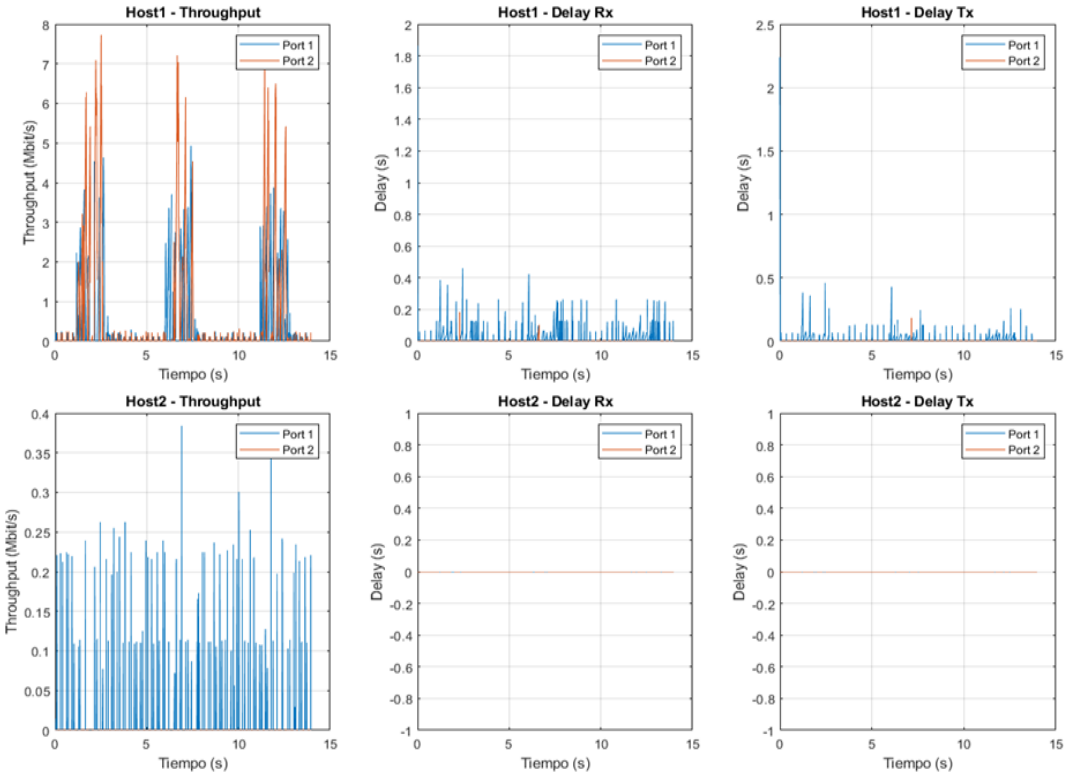


Figura 7.24. Prueba 4 con $K_p = 1$ y $K_c = 4$

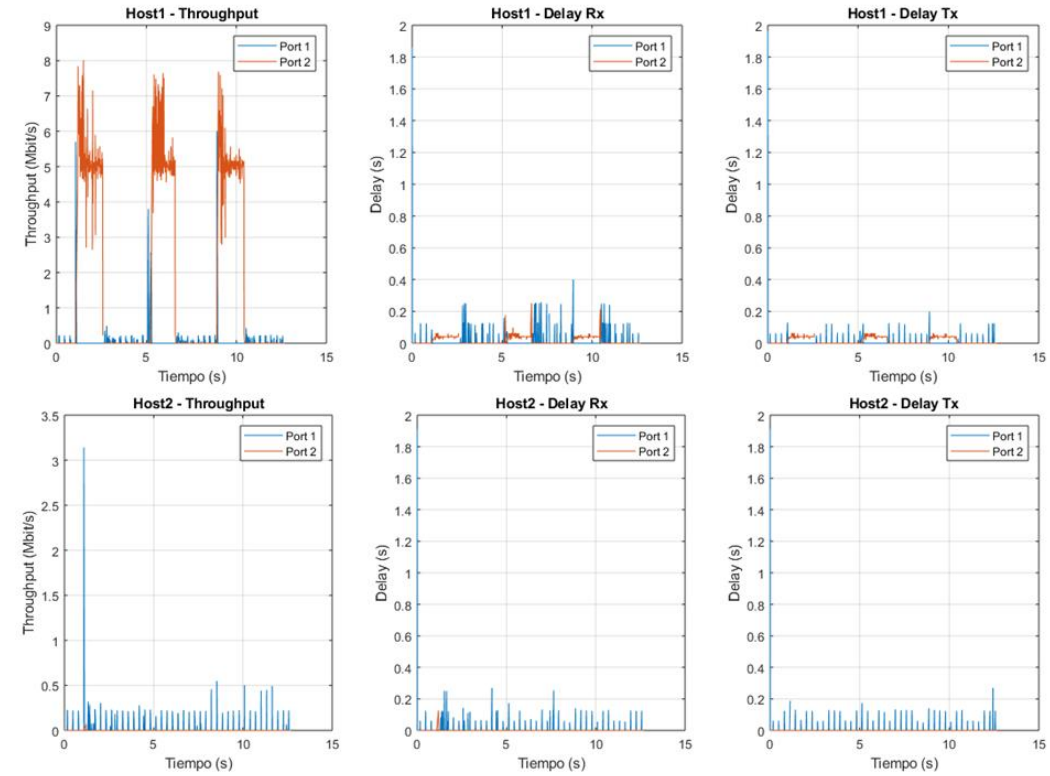


Figura 7.25. Prueba 5 con $K_p = 1$ y $K_c = 4$

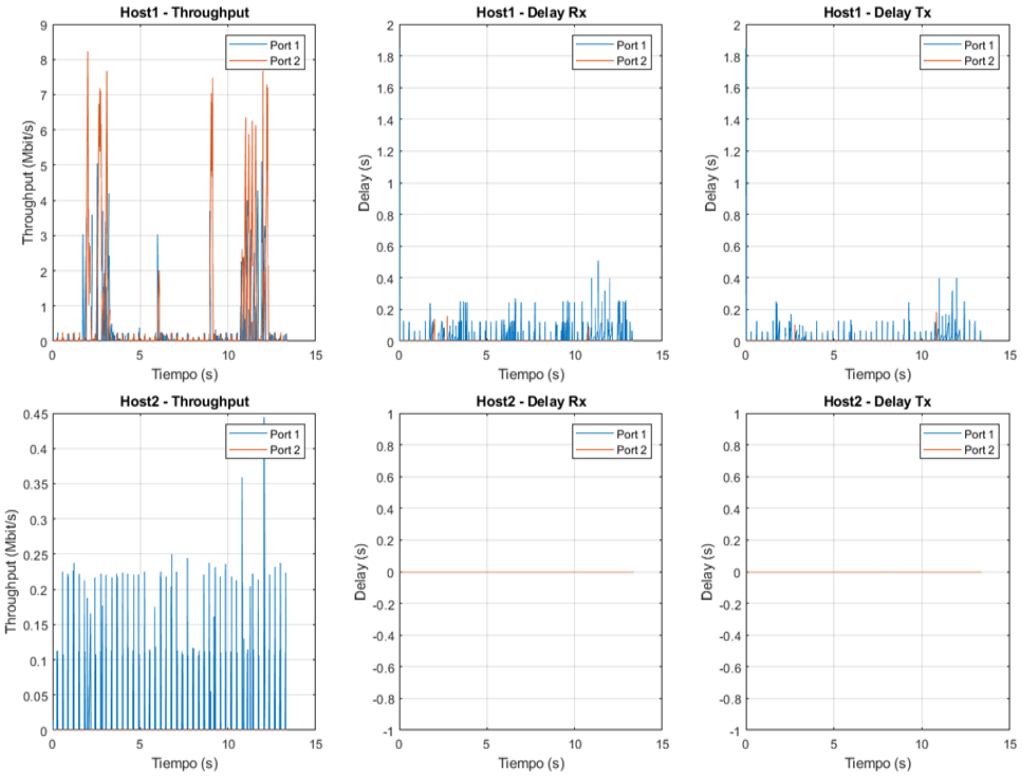


Figura 7.26. Prueba 6 con $K_p = 1$ y $K_c = 4$

Al observar de la Figura 7.21 a la 7.26, se puede apreciar que, para las pruebas con una tasa de transmisión de 2 Mbps, los retardos que se presentan oscilan entre 0 ms y valores por encima de 100 ms, razón por la cual el algoritmo realiza una conmutación constate entre el puerto 1 y el puerto 2. Cabe resaltar que, durante dicha conmutación, el protocolo TCP impidió la perdida de paquetes, esto debido a sus características de confiabilidad y al hecho de ser orientado a la conexión.

Por otro lado, con una tasa de transmisión de 5 Mbps se puede apreciar que, en algunos casos los delays de igual forma superan el límite establecido de 100 ms, pero como el throughput no supera la mitad del ancho de banda del canal de comunicaciones, el algoritmo no realiza el cambio de puerto.

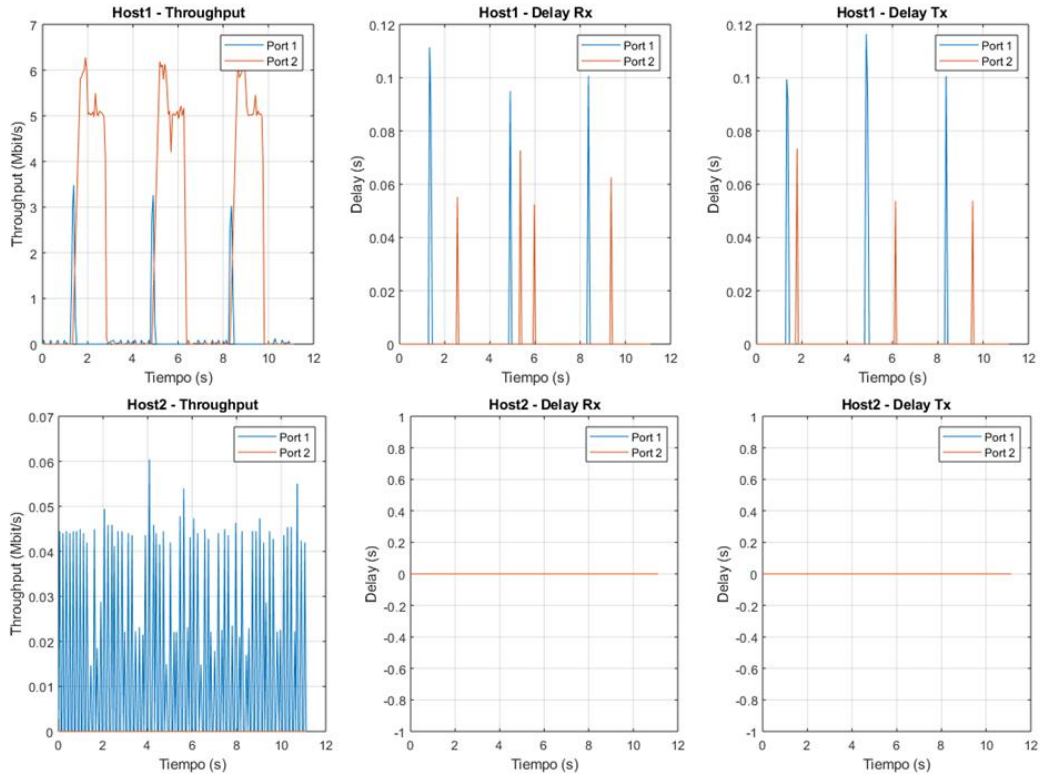


Figura 7.27. Prueba 1 con $K_p = 5$ y $K_c = 4$

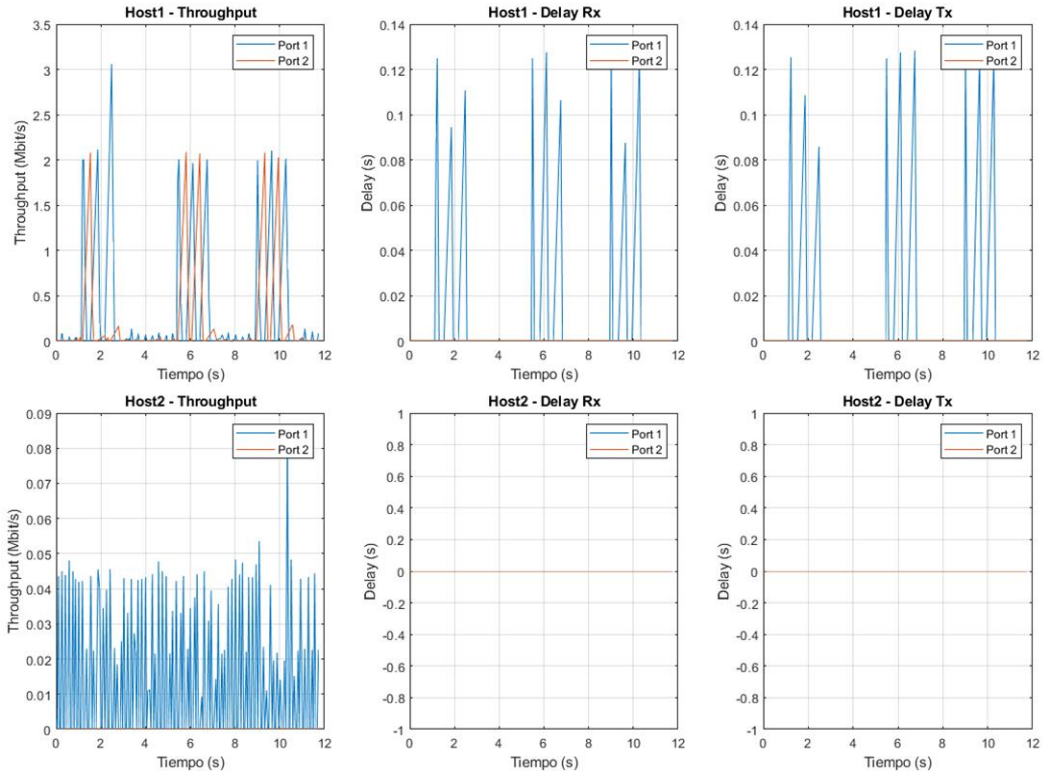


Figura 7.28. Prueba 2 con $K_p = 5$ y $K_c = 4$

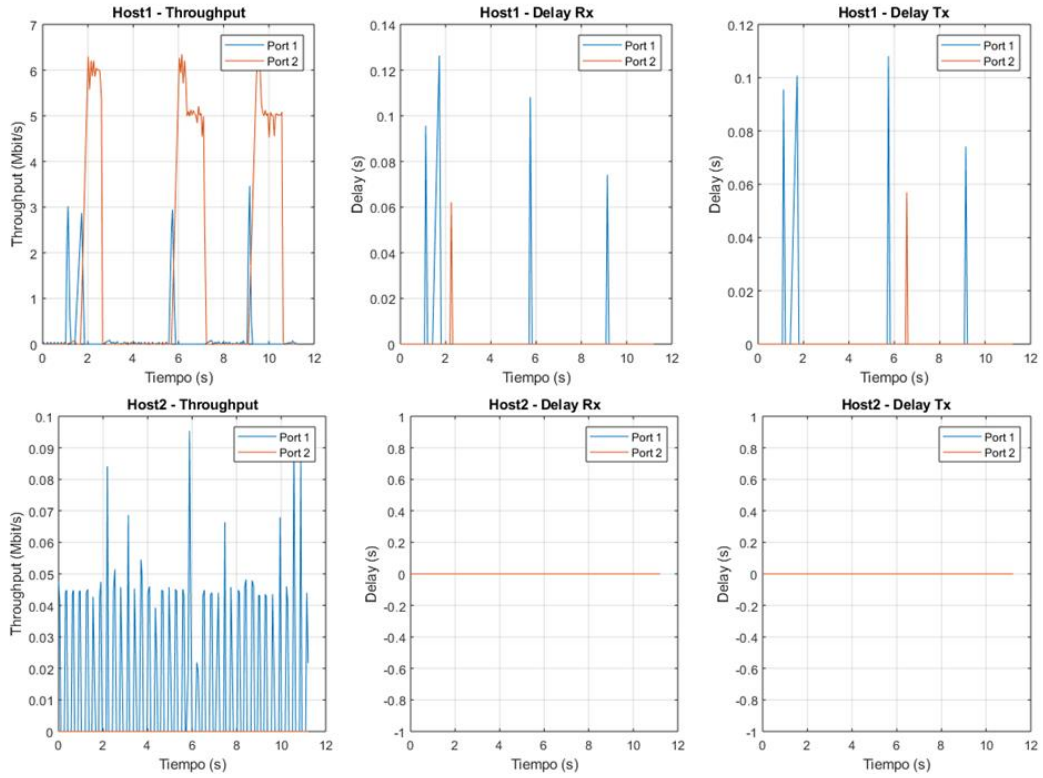


Figura 7.29. Prueba 3 con $K_p = 5$ y $K_c = 4$

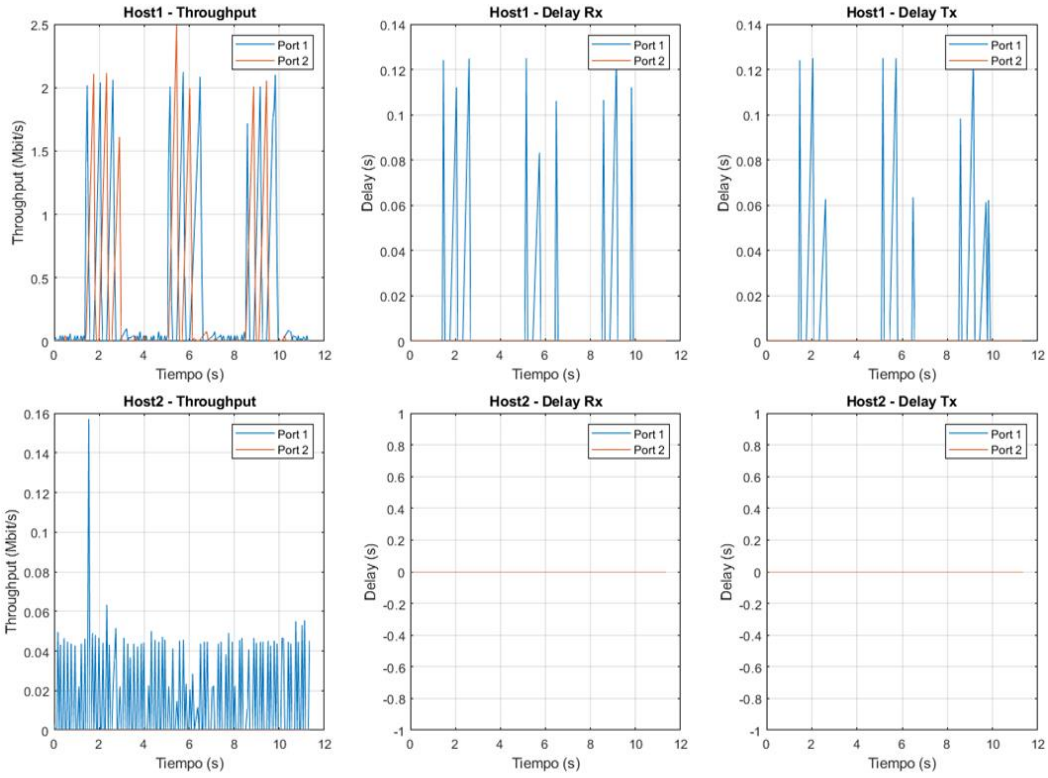


Figura 7.30. Prueba 4 con $K_p = 5$ y $K_c = 4$

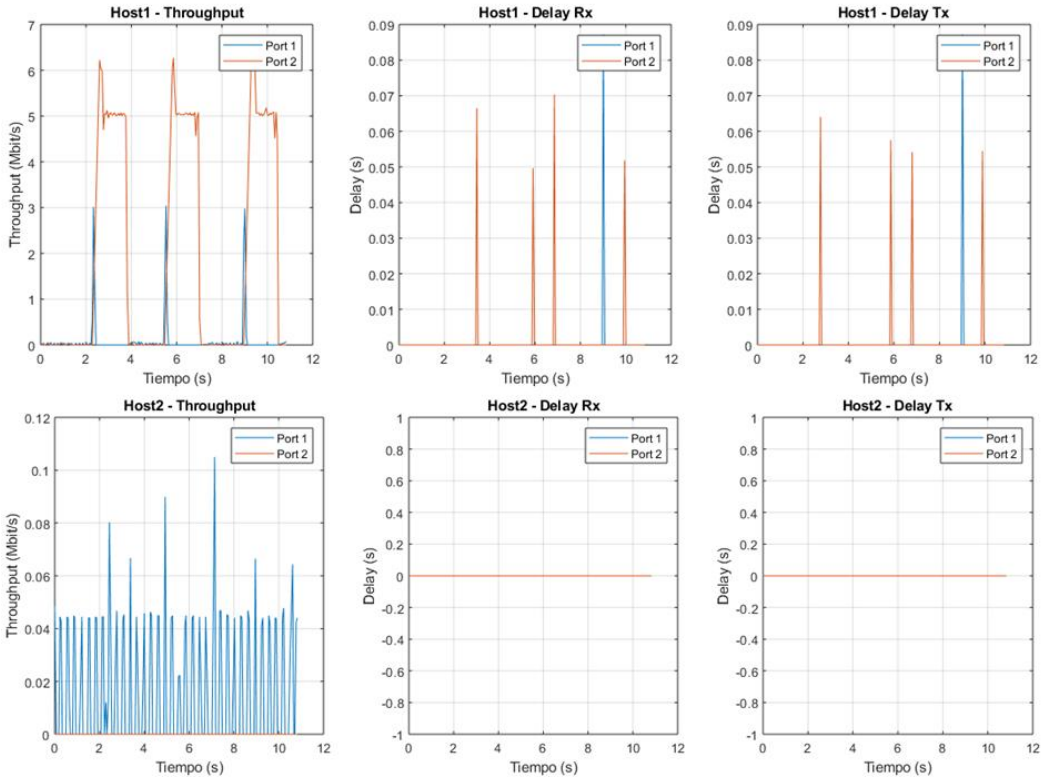


Figura 7.31. Prueba 5 con $K_p = 5$ y $K_c = 4$

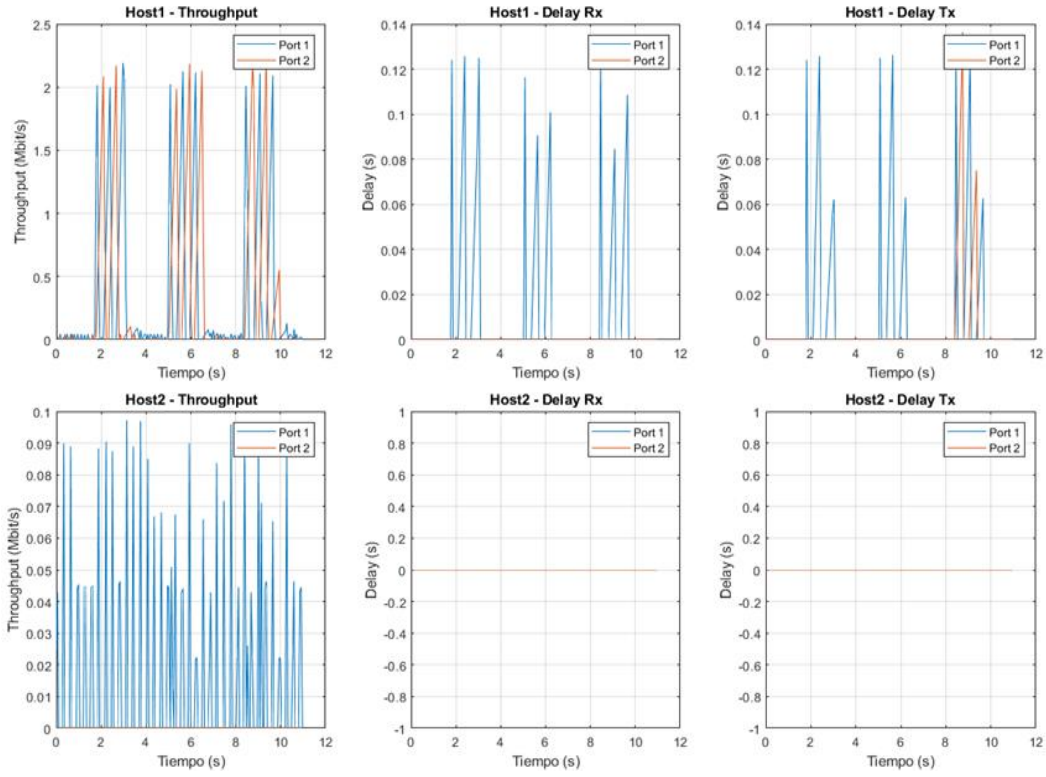


Figura 7.32. Prueba 6 con $K_p = 5$ y $K_c = 4$

Al observar de la Figura 7.27 a la 7.32, de igual manera que para mensajes UDP, se hace evidente que K_p actúa como un filtro para las señales, disminuyendo considerablemente el rizado y para la prueba con una tasa de transmisión de 2 Mbps se observa un menor número de conmutaciones entre el puerto 1 y el puerto 2.

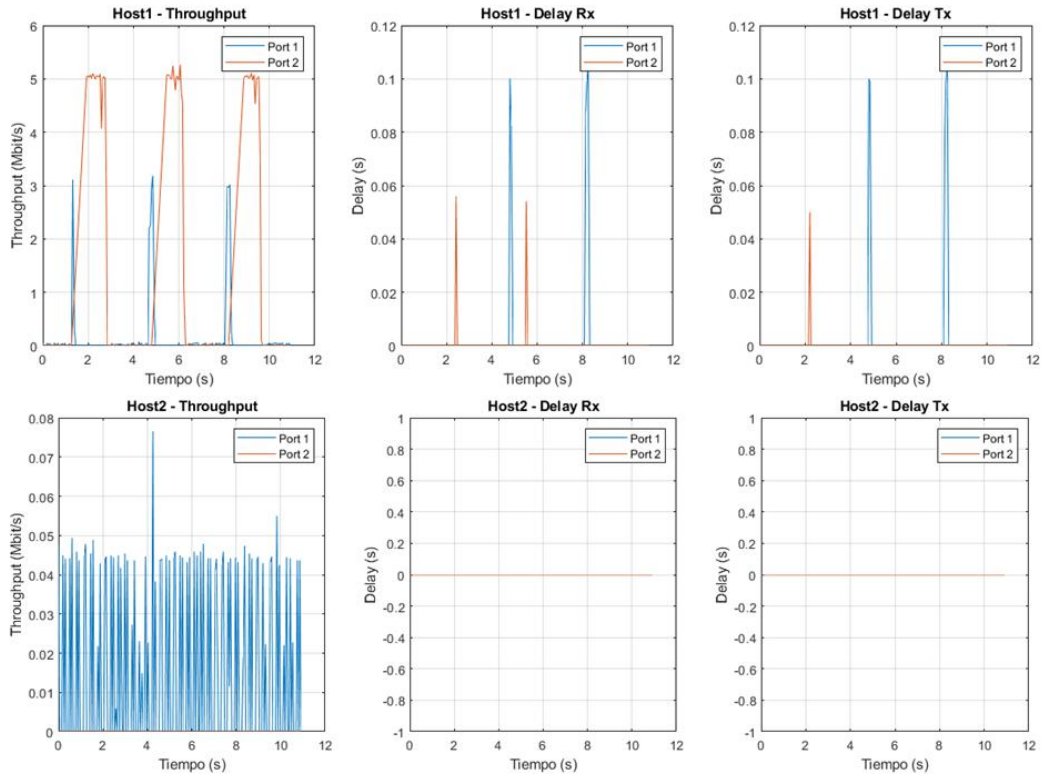


Figura 7.33. Prueba 1 con $K_p = 5$ y $K_c = 10$

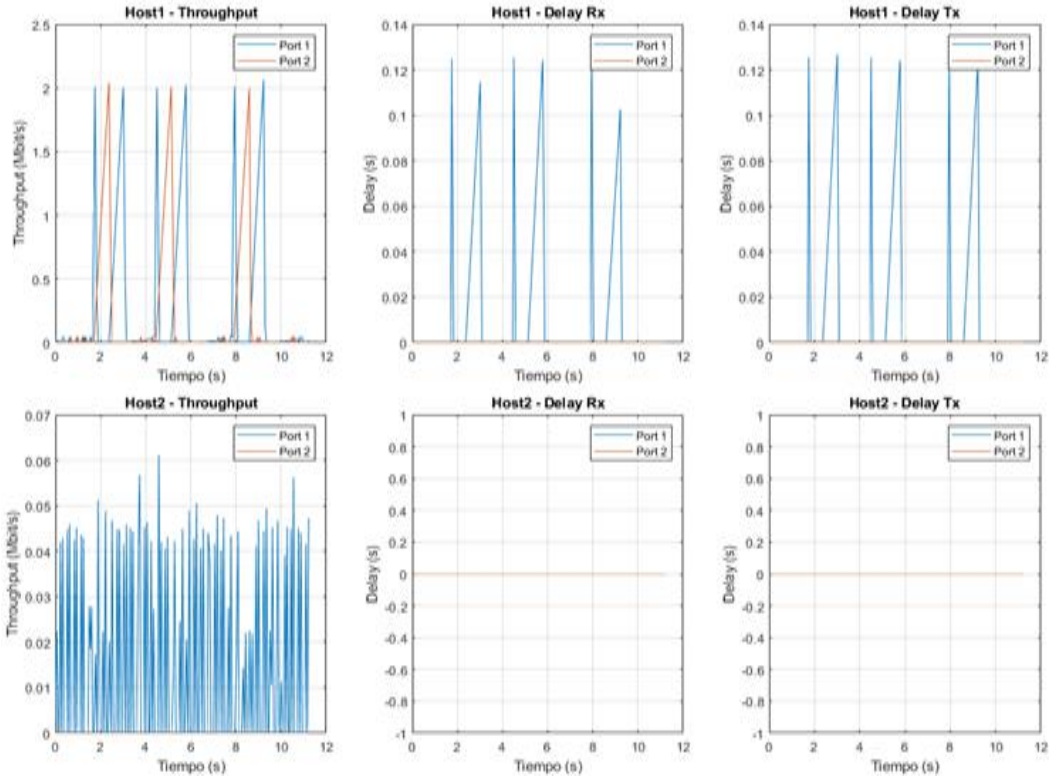


Figura 7.34. Prueba 2 con $K_p = 5$ y $K_c = 10$

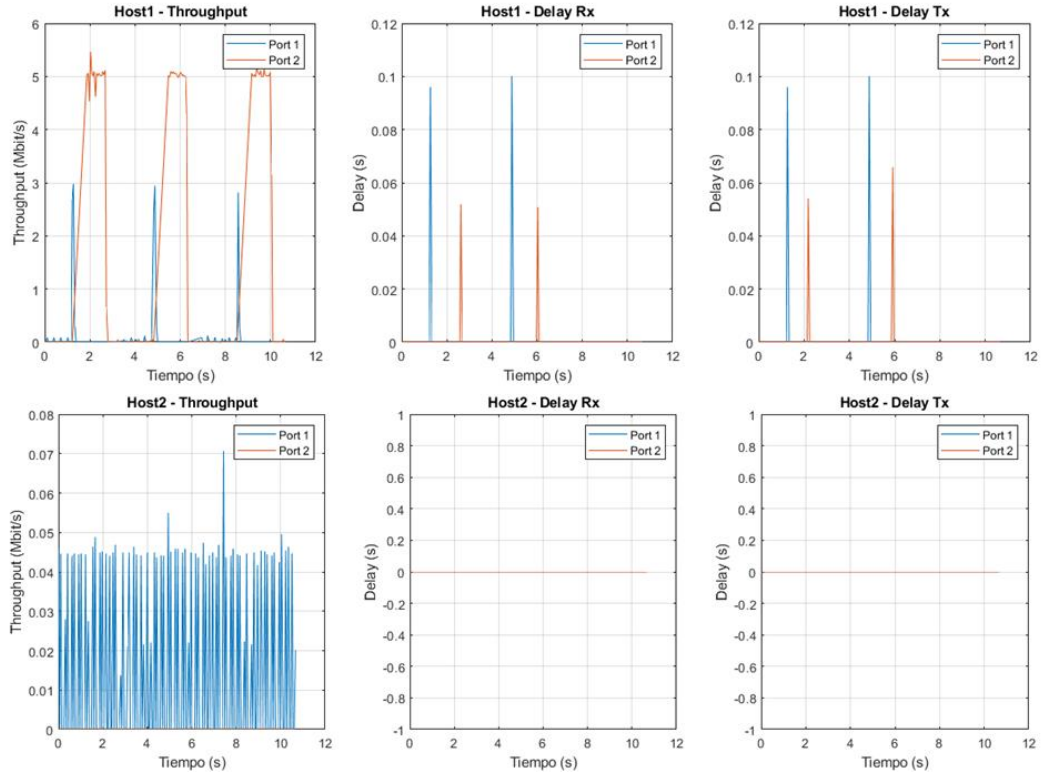


Figura 7.35. Prueba 3 con $K_p = 5$ y $K_c = 10$

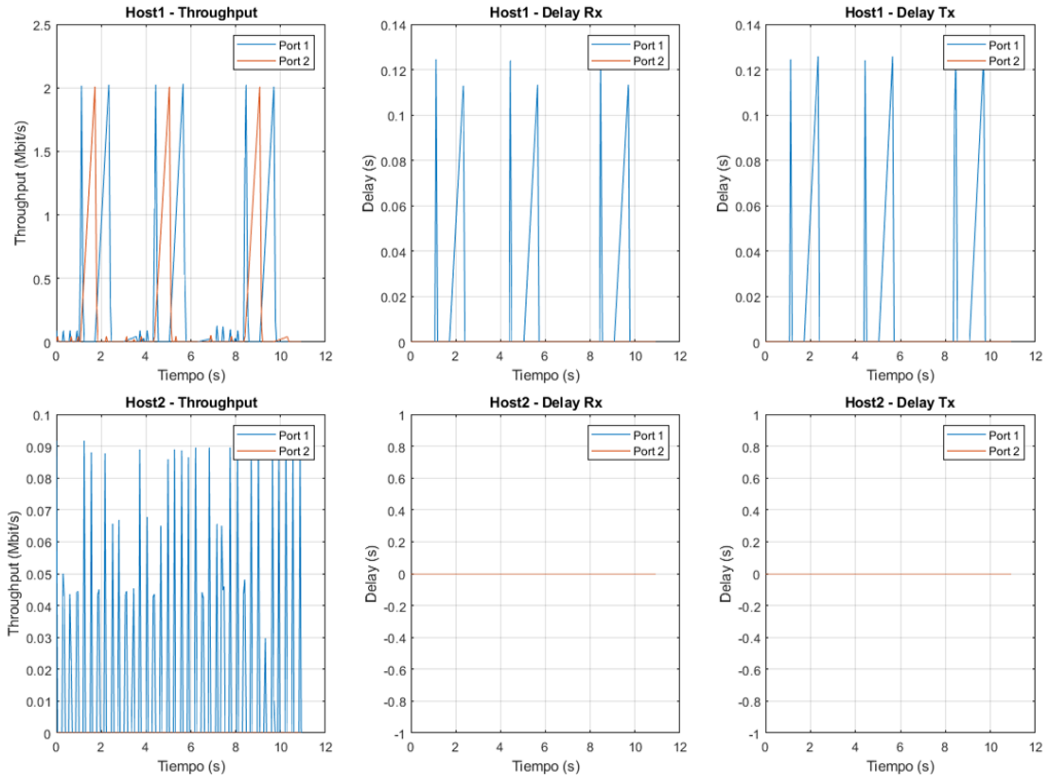


Figura 7.36. Prueba 4 con $K_p = 5$ y $K_c = 10$

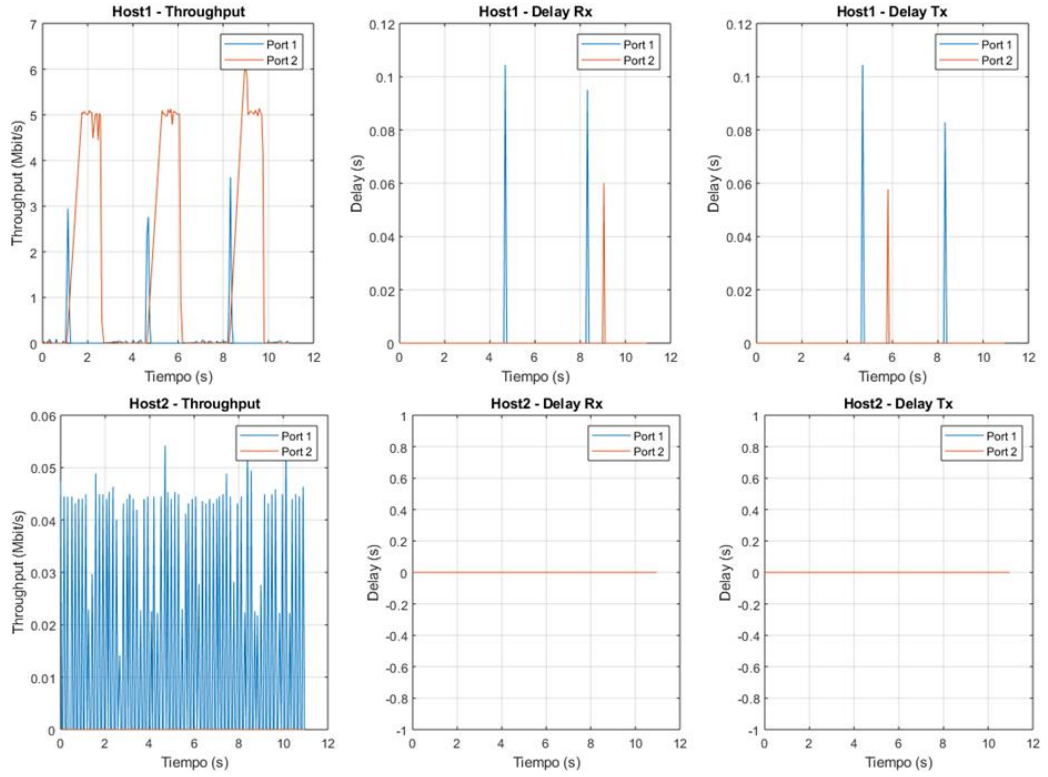


Figura 7.37. Prueba 5 con $K_p = 5$ y $K_c = 10$

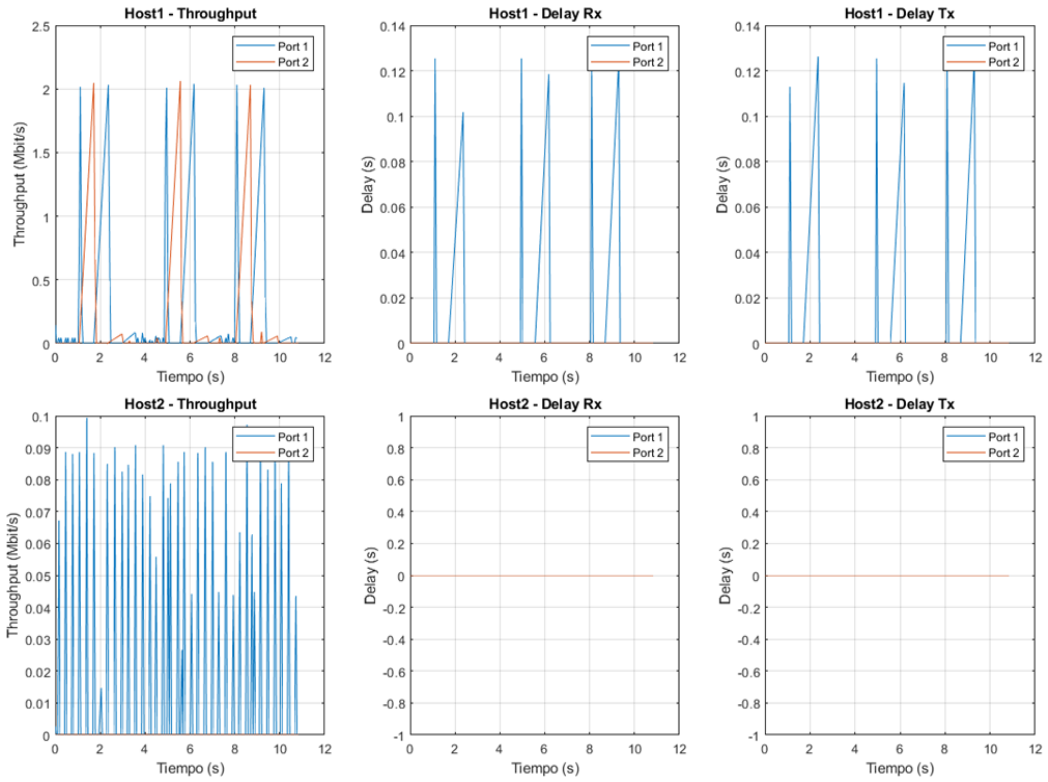


Figura 7.38. Prueba 6 con $K_p = 5$ y $K_c = 10$

Al observar de la Figura 7.33 a la 7.38, es posible identificar un menor número de conmutaciones entre el puerto 1 y el puerto 2, esto para el caso de las pruebas con una tasa de transmisión de 2 Mbps. La reducción en el número de conmutaciones es debido a que al incrementar el parámetro K_c, se incrementa el tiempo de espera para tomar una nueva decisión sobre la red.

En cada una de las pruebas realizadas con mensajes TCP, se registraron cero paquetes perdidos sin importar las condiciones de la red, además es posible apreciar como el protocolo TCP determina dos alternativas para evitar la pérdida de parques, por un lado es posible que incremente la tasa de transmisión al realizar una serie de reenvíos, este caso es posible identificarlo por ejemplo en la Figura 7.29, donde el Throughput se incrementó hasta alcanzar en ocasiones todo el ancho de banda del canal asociado al puerto 2 (6 Mbps). Por otro lado, es posible que el tiempo fijado para la trasmisión en el generador (1.5 segundos) sea forzado a incrementarse, este caso se puede evidenciar en la Figura 7.26, y es aún más claro si vemos el reporte del generador de tráfico de la misma prueba en la Figura 7.39, específicamente en el apartado de tiempo total.

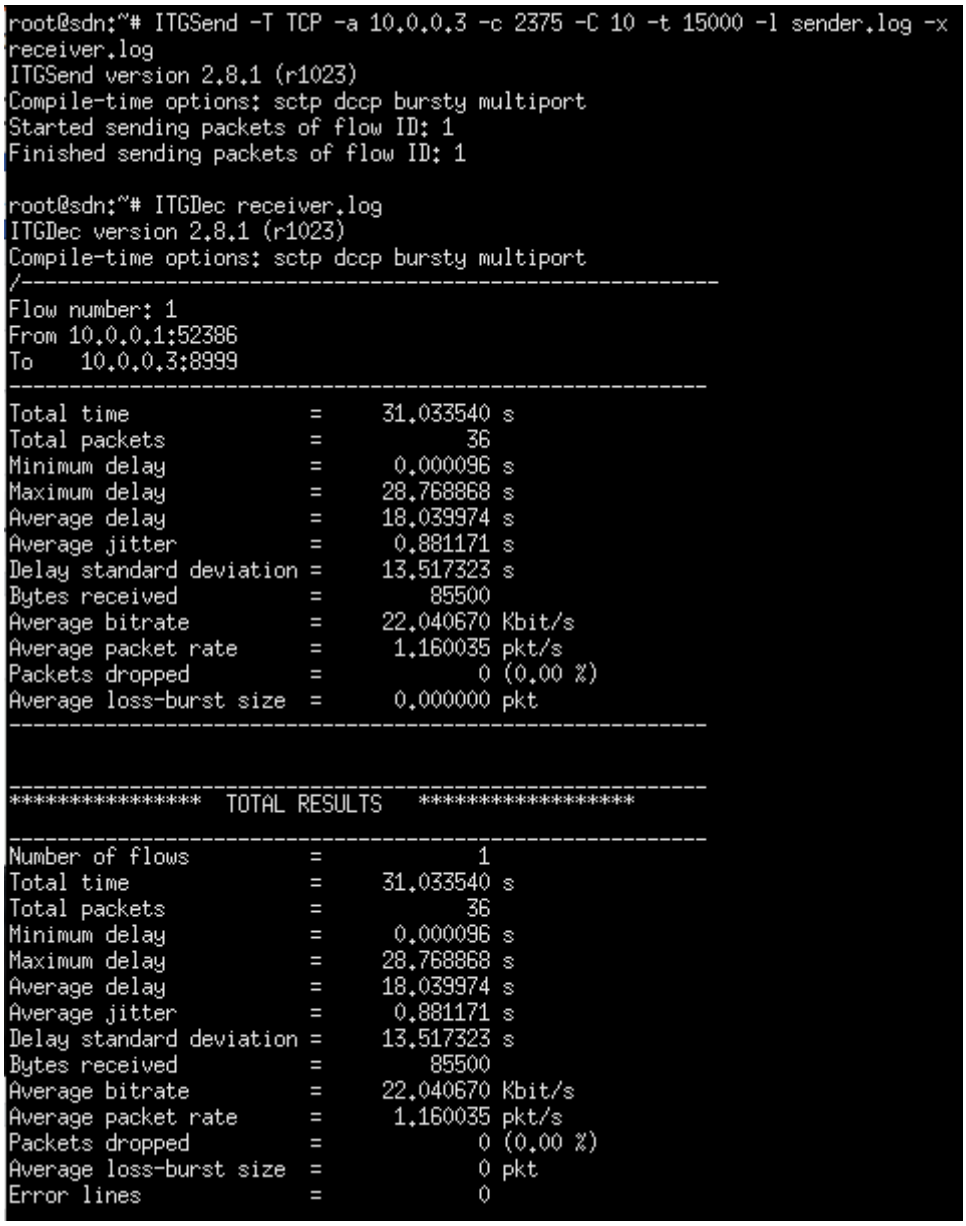


Figura 7.39. Reporte generador de tráfico

7.4 Implementación física

Para la implementación física, se seleccionó un video con una tasa de transmisión aproximada de 1.5 Mbps, en base en esto se decidió fijar en el algoritmo un umbral de throughput de 0.8 Mbps para que de esta manera se generara la conmutación del puerto 1 al puerto 2. Luego de fijar este umbral, para evaluar las capacidades del algoritmo se realizaron 3 pruebas variando los parámetros K_p y K_c, estas pruebas se pueden observar en las figuras 7.40, 7.41 y 7.42.

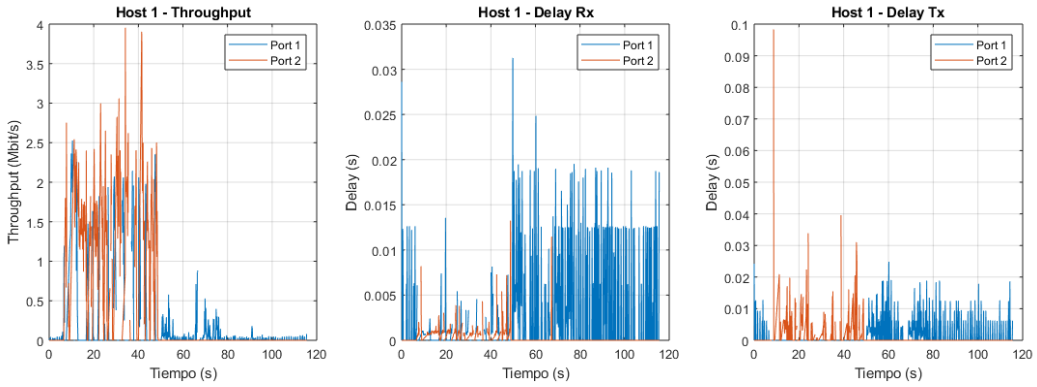


Figura 7.40. Prueba con $K_p = 1$ y $K_c = 4$

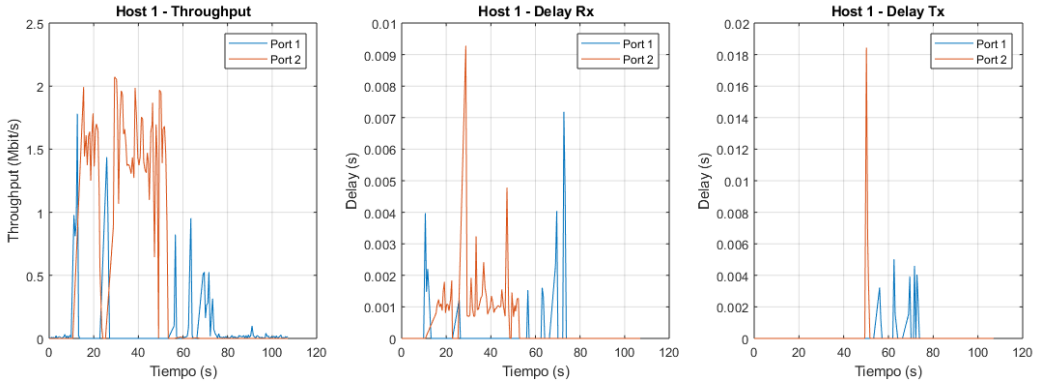


Figura 7.41. Prueba con $K_p = 5$ y $K_c = 4$

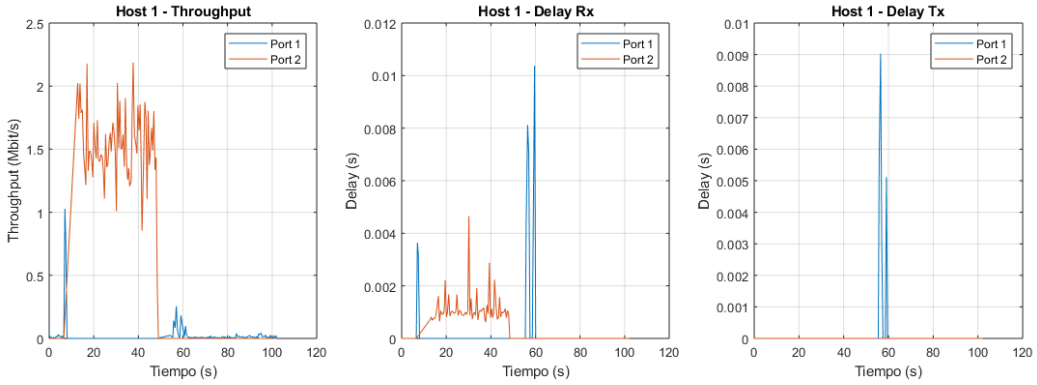


Figura 7.42. Prueba con $K_p = 5$ y $K_c = 10$

De acuerdo con los resultados de la implementación física se puede observar que no difieren demasiado de los resultados obtenidos en emulación, y también se hace evidente como el aumentar el valor de los parámetros K_p y K_c hacen que el comportamiento de la red mejore, ya que esto evita que el puerto 1 y el puerto 2 queden conmutando como en la Figura 7.40.

8. Trabajos futuros

Para mejorar las capacidades del algoritmo del volcado del tráfico de datos, se podría hacer uso de la inteligencia artificial para realizar predicciones sobre el comportamiento de la red y en base a estas predicciones variar los parámetros del algoritmo, de esta manera se podrían evitar conmutaciones innecesarias, y mejoraría el rendimiento de la red.

En el algoritmo del volcado del tráfico de datos propuesto sólo se tuvieron en cuenta como parámetros para decidir sobre la red el throughput y el delay, como trabajo futuro se podrían evaluar otros parámetros de decisión como el jitter o los paquetes perdidos, para de esta manera evaluar si las capacidades del algoritmo mejoran teniendo en cuenta una mayor cantidad de parámetros.

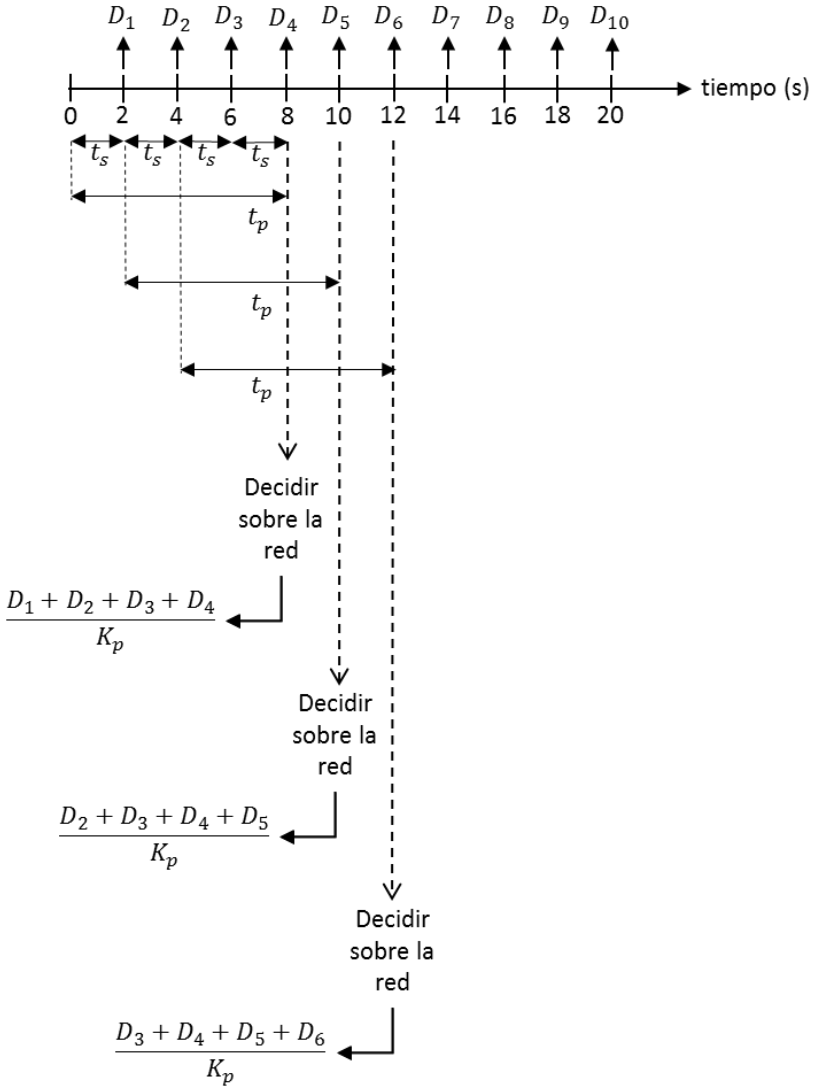


Figura 8.1. Propuesta para mejorar el funcionamiento del algoritmo

Al observar los efectos de variar el parámetro K_p es posible evidenciar como al incrementar significativamente dicho parámetro, el algoritmo en general se hace más lento a la hora de tomar una decisión sobre la red. Es por esta razón que se plantea una pequeña variación con el fin de mejorar los tiempos de reacción del algoritmo, en la Figura 8.1 es posible evidenciar otra forma de promediar los datos en la red, de manera muy similar al proceso de ventaneo ampliamente utilizado en el análisis y procesamiento de señales.

9. Conclusiones

De acuerdo a los resultados obtenidos, se evidencia como el algoritmo del volcado del tráfico de datos mejora las capacidades de la red en términos de QoS, por ejemplo, aumentado el ancho de banda disponible en la misma, ya que para el caso de la emulación cuando se realizaba un cambio de puerto, el ancho de banda disponible en el host 1 aumentaba al doble (pasaba de 3 Mbits a 6 Mbits). Esto a la vez permite tener mayores velocidades o tener la capacidad de soportar una mayor cantidad de hosts sin que la red se sature.

Aunque el desempeño del algoritmo en varias de las pruebas tuvo un resultado bastante bueno y cumplió las expectativas planteadas, se hace evidente la necesidad de un análisis más profundo de los parámetros k variados a lo largo de las pruebas diseñadas. Es decir, con este análisis se pretendería obtener los valores de los parámetros k con los cuales la red tendría mejor comportamiento.

El principal problema que se destaca del algoritmo es que en ocasiones se corre el riesgo de quedar conmutando entre dos canales de comunicaciones, lo cual hace que para el caso del tráfico UDP la pérdida de paquetes se incremente y para el caso del tráfico TCP la red aumente la tasa de transmisión (lo que puede provocar que la red se sature) o que se alargue el tiempo de la transmisión de datos (lo cual significa que la tasa de transmisión disminuye). Sin embargo, esto es fácilmente corregible con la variación de ciertos parámetros, como se pudo evidenciar en los resultados.

Se pudo evidenciar que la conmutación de puerto generada por el algoritmo del volcado del tráfico de datos, no tiene un impacto importante en cuanto a la cantidad de paquetes perdidos. Ya que en el caso del protocolo UDP la perdida de paquetes generada por la conmutación es baja, y en el caso del protocolo TCP la perdida de paquetes generada por la conmutación es nula. Este hecho es de suma importancia debido a que, si se hubiese producido una pérdida de paquetes muy elevada durante la conmutación, la viabilidad del algoritmo del volcado del tráfico de datos sobre una red SDN, estaría totalmente comprometida.

Para el caso de la implementación física, indiferentemente de las interfaces de red que se usen en el host - cliente, se hace evidente la necesidad de mantener una sola dirección IP, esto con el fin de evitar problemas durante el volcado del tráfico de datos en términos de enrutamiento, permitiendo de esta manera hacer uso de cualquier protocolo de red.

Referencias

[1] S. Kemp, “Digital in 2018: World’s internet users pass the 4 billion mark.” [Online]. Available: https://wearesocial.com/blog/2018/01/global-digital-report-2018. [Accessed: 08-Aug-2018].

[2] Huawei, “GIV 2025 Unfolding the Industry Blueprint of an Intelligent World,” 2018.

[3] Cisco, “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021,” 2017.

[4] C. Crespo Garay, “La basura electrónica alcanzará los 120 millones de toneladas en 2050.” [Online]. Available: https://www.nationalgeographic.es/medio-ambiente/2019/03/la-basura-electronica-alcanzara-los-120-millones-de-toneladas-en-2050. [Accessed: 15-Sep-2019].

[5] “Conceptos Básicos de Redes Fijas – ¿Qué necesito saber? | Temas Tecnologicos de Interes.” [Online]. Available: https://www.temastecnologicos.com/redes-fijas/. [Accessed: 15-Oct-2018].

[6] “Conceptos Básicos de Redes Móviles – ¿Qué necesito saber? | Temas Tecnologicos de Interes.” [Online]. Available: https://www.temastecnologicos.com/redes-moviles/. [Accessed: 15-Oct-2018].

[7] VIU Universidad Internacional de Valencia, “Que es LTE y por qué difiere del 4G.” [Online]. Available: https://www.universidadviu.es/que-es-lte-y-por-que-difiere-del-4g/. [Accessed: 18-Oct-2018].

[8] H. Soffar, “4G Technology uses , features , advantages and disadvantages.” [Online]. Available: https://www.online-sciences.com/technology/4g-technology-uses-features-advantages-and-disadvantages/. [Accessed: 18-Oct-2018].

[9] M. Rouse, “What is 5G?” [Online]. Available: https://searchnetworking.techtarget.com/definition/5G. [Accessed: 18-Oct-2018].

[10] P. Sharma, “Evolution of Mobile Wireless Communication Networks-1G to 5G as well as Future Prospective of Next Generation Communication Network,” 2013.

[11] M. Paliwal, D. Shrimankar, and O. Tembhurne, “Controllers in SDN: A Review Report,” 2018.

[12] SDxCentral, “SDN.” [Online]. Available: https://www.sdxcentral.com/sdn/.

[Accessed: 17-Oct-2018].

[13] SDxCentral, “What is OpenFlow? Definition and How it Relates to SDN.” [Online]. Available: https://www.sdxcentral.com/sdn/definitions/what-is-openflow/. [Accessed: 17-Oct-2018].

[14] SDxCentral, “What’s Software-Defined Networking (SDN)?” [Online]. Available: https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/. [Accessed: 17-Oct-2018].

[15] Open Networking Foundation, “ONOS.” [Online]. Available: https://www.opennetworking.org/onos/. [Accessed: 17-Oct-2018].

[16] ONOS, “Features - ONOS.” [Online]. Available: https://onosproject.org/features/. [Accessed: 17-Oct-2018].

[17] Mininet, “Mininet Overview.” [Online]. Available: http://mininet.org/overview/. [Accessed: 17-Oct-2018].

[18] M. A. Baquero Rojas and F. R. Garcia Perdomo, “Aplicación de las Redes Definidas por Software en el Contexto del Internet de las Cosas,” Universidad Distrital Francisco José de Caldas, 2018.

[19] S. Scherzer, “Femtocell vs. Wi-Fi for Data Offloading & Indoor Coverage.” [Online]. Available: https://www.ecnmag.com/article/2010/02/femtocell-vs-wi-fi-data-offloading-indoor-coverage. [Accessed: 18-Oct-2018].

[20] X. Duan, A. M. Akhtar, and X. Wang, “Software-defined networking-based resource management: data offloading with load balancing in 5G HetNet,” 2015.

[21] R.-S. Cheng, C.-M. Huang, and S.-Y. Pan, “WiFi offloading using the device-to-device (D2D) communication paradigm based on the Software Defined Network (SDN) architecture,” 2018.

[22] M. Jammal, T. Singh, A. Shami, and Y. Li, “Software-Defined Networking: State of the Art and Research Challenges,” 2014.

[23] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN: An Intellectual History of Programmable Networks,” 2014.

[24] Ixia, *EBOOK. Software Defined Networking (SDN)*. 2014.

[25] Hewlett-Packard Development Company, “Virtual networking technologies at the server-network edge,” 2011.

[26] H.-M. Tseng, H.-L. Lee, J.-W. Hu, T.-L. Liu, J.-G. Chang, and W.-C. Huang, “Network Virtualization with Cloud Virtual Switch,” 2011.

[27] S. Din, M. M. Rathore, A. Ahmad, A. Paul, and M. Khan, “SDIoT: Software Defined Internet of Thing to Analyze Big Data in Smart Cities,” 2017.

[28] S. S. Bhunia and M. Gurusamy, “Dynamic attack detection and mitigation in IoT using SDN,” 2017.

[29] S. Sharma *et al.*, “Implementing Quality of Service for the Software Defined Networking Enabled Future Internet,” 2014.

[30] S. Peros, H. Janjua, S. Akkermans, W. Joosen, and D. Hughes, “Dynamic QoS support for IoT backhaul networks through SDN,” 2018.

[31] S. Komajwar and T. Korkmaz, “Challenges and solutions to consistent data plane update in software defined networks,” 2018.

[32] W.-C. Chien, C.-F. Lai, H.-H. Cho, and H.-C. Chao, “A SDN-SFC-based service-oriented load balancing for the IoT applications,” 2018.

[33] C.-M. Huang, M.-S. Chiang, D.-T. Dao, H.-M. Pai, S. Xu, and H. Zhou, “Vehicle-to-Infrastructure (V2I) offloading from cellular network to 802.11p Wi-Fi network based on the Software-Defined Network (SDN) architecture,” 2017.

[34] ITU-T, “Multimedia quality of service and performance – Generic and user-related aspects: Quality of experience requirements for IPTV services,” pp. 1–89, 2008.

Anexos

Topología personalizada – Mininet

```python
#!/usr/bin/python

# LIBRERIAS IMPORTADAS
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    # SE ESTABLECE LA CLASE Y MASCARA IP
    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    # SE AGREGA UN CONTROLADOR REMOTO
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='10.0.2.15',
                         protocol='tcp',
                         port=6633)

    # SE AGREGAN SWITCHES
    info( '*** Add switches\n')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    # SE AGREGAN HOSTS - SERVER
    info( '*** Add hosts\n')
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)

    # SE DEFINEN ANCHOS DE BANDA PARA LOS ENLACES
    info( '*** Add links\n')
    Link1 = {'bw':3}
    Link2 = {'bw':6}

    # SE ESTABLECEN ENLACES ENTRE DISPOSITIVOS
    # (SWITCH - SWITCH) AGREGANDO EL ANCHO DE BANDA DESEADO
    net.addLink(s3, s1, cls=TCLink, **Link1)
    net.addLink(s3, s1, cls=TCLink, **Link2)
    # (HOST - SWITCH)
    net.addLink(h1, s1)
    # (SWITCH - SWITCH) AGREGANDO EL ANCHO DE BANDA DESEADO
    net.addLink(s3, s2, cls=TCLink, **Link1)
    net.addLink(s3, s2, cls=TCLink, **Link2)
    # (HOST - SWITCH)
    net.addLink(h2, s2)
    # (SERVER - SWITCH)
    net.addLink(h3, s3)

    # SE CONSTRUYE LA RED E INICIALIZAN DISPOSITIVOS
    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    # SE ESTABLE UNA CONEXION ENTRE EL CONTROLADOR Y LOS SWITCHES
    info( '*** Starting switches\n')
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s1').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()

```

Algoritmo del volcado del tráfico de datos – ONOS

```
/*
 * Copyright 2019-present Open Networking Foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.foo.app;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;

import org.apache.karaf.shell.api.action.Argument;
import org.apache.karaf.shell.api.action.Command;
import org.apache.karaf.shell.api.action.Completion;
import org.apache.karaf.shell.api.action.lifecycle.Service;
import org.apache.karaf.shell.api.action.Option;

import org.onlab.packet.IpAddress;
import org.onlab.util.Tools;

import org.onosproject.cli.AbstractShellCommand;
import org.onosproject.cli.app.ApplicationIdWithIntentNameCompleter;

import org.onosproject.cluster.ClusterService;
import org.onosproject.cluster.ControllerNode;

import org.onosproject.core.ApplicationId;
import org.onosproject.core.CoreService;
import org.onosproject.core.Version;

import org.onosproject.net.ConnectPoint;
import org.onosproject.net.config.NetworkConfigService;
import org.onosproject.net.config.basics.BasicLinkConfig;
import org.onosproject.net.device.DeviceAdminService;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.device.PortStatistics;
import org.onosproject.net.Device;
import org.onosproject.net.DeviceId;
import org.onosproject.net.edge.EdgePortService;
import org.onosproject.net.FilteredConnectPoint;
import org.onosproject.net.flow.TrafficSelector;
import org.onosproject.net.flow.TrafficTreatment;
import org.onosproject.net.host.HostService;
import org.onosproject.net.intent.Constraint;
import org.onosproject.net.intent.constraint.ProtectedConstraint;
import org.onosproject.net.intent.Intent;
import org.onosproject.net.intent.IntentEvent;
import org.onosproject.net.intent.IntentListener;
import org.onosproject.net.intent.IntentService;
import org.onosproject.net.intent.IntentState;
import org.onosproject.net.intent.Key;
import org.onosproject.net.intent.PointToPointIntent;
import org.onosproject.net.Link;
import org.onosproject.net.LinkKey;
import org.onosproject.net.link.LinkService;
import org.onosproject.net.Port;
import org.onosproject.net.PortNumber;
import org.onosproject.net.statistic.Load;
import org.onosproject.net.statistic.StatisticService;

import org.onosproject.utils.Comparators;

import static org.onosproject.cli.net.DevicesListCommand.getSortedDevices;
import static org.onosproject.net.DeviceId.deviceId;
import static org.onosproject.net.PortNumber.portNumber;
import static org.onosproject.net.intent.constraint.ProtectionConstraint.protection;
import static org.onosproject.net.intent.IntentState.FAILED;
import static org.onosproject.net.intent.IntentState.WITHDRAWN;
import static org.onosproject.net.LinkKey.linkKey;

import static com.google.common.base.Strings.isNullOrEmpty;
import static com.google.common.collect.Lists.newArrayList;

import java.io.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.util.*;
import java.util.Comparator;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;
import java.util.List;
import java.util.EnumSet;
import java.util.Objects;
import java.util.ArrayList;
import java.util.Collections;
```

```
 99
100 import com.google.common.collect.Lists;
101
102
103 /**
104  * Sample Apache Karaf CLI command
105  */
106 @Service
107 @Command(scope = "onos", name = "sample",
108          description = "Sample Apache Karaf CLI command")
109 public class AppCommand extends AbstractShellCommand {
110
111     private static final String FORMAT =
112     " port=%s \n  rateRx=%s, rateTx=%s, \ndelayRx=%s, delayTx=%s, Dur=%s\n";
113     private static final String FMT = "dst=%s/%s";
114
115     @Argument(index = 0, name = "Nserv", description = "Numero de Servidores",
116               required = true, multiValued = false)
117     int Nserv = 0;
118
119     private long activeNodes(Set<ControllerNode> nodes) {
120         ClusterService clusterService = get(ClusterService.class);
121
122     return nodes.stream()
123                 .map(node -> clusterService.getState(node.id()))
124                 .filter(nodeState -> nodeState.isActive())
125                 .count();
126
127     }
128
129     private static long t_s = 100; // tiempo de sondeo [ms]
130     private static int kp = 10; // tp = ts*kp;  kp: Número de datos a promediar
131     private static int kc = 4; // tc = tp*kc = ts*kp*kc
132     private static float Lim_rate = 30000000; // Limite de tasa [bit/s]
133     private static float Lim_delayTx = 0.01f; // Limite del delay de Tx [s]
134     private static float Lim_delayRx = 0.01f; // Limite del delay de Rx [s]
135     private static float BW = 30000000; // Ancho de banda de canal con menor ancho de banda [bit]
136
137
138     public static boolean flag = true;
139     public static ArrayList<Boolean> flag_time = new ArrayList<Boolean>();
140     public static ArrayList<Integer> t_c = new ArrayList<Integer>();
141     public static ArrayList<Float> time_past = new ArrayList<Float>();
142     public static ArrayList<Float> pktRx_past = new ArrayList<Float>();
143     public static ArrayList<Float> pktTx_past = new ArrayList<Float>();
144     public static ArrayList<Float> bytesRx_past = new ArrayList<Float>();
145     public static ArrayList<Float> bytesTx_past = new ArrayList<Float>();
146
147     // Arreglos para los datos a promediar
148     public static ArrayList<Float> rateRx_av = new ArrayList<Float>();
149     public static ArrayList<Float> rateTx_av = new ArrayList<Float>();
150     public static ArrayList<Float> delayRx_av = new ArrayList<Float>();
151     public static ArrayList<Float> delayTx_av = new ArrayList<Float>();
152
153     public static int cambio = 0;
154
155     @Override
156     protected void doExecute() {
157
158         /* Busca e imprime el numero de hosts en la red */
159         DeviceService deviceService = get(DeviceService.class);
160         IpAddress nodeIp = get(ClusterService.class).getLocalNode().ip();
161         Version version = get(CoreService.class).version();
162         int numHosts = get(HostService.class).getHostCount();
163         print("hosts=%d",numHosts);
164
165         // Numero de servidores en la red
166         print("servers=%s",Nserv);
167
168         int Nhost_Real = numHosts-Nserv;
169
170         // Solo se ejecuta la primera vez
171         if(flag) {
172            float aux = 0;
173            for(int i=1;i<=(Nhost_Real*2+1);i++) {
174               flag_time.add(false);
175               t_c.add(0);
176               time_past.add(aux);
177               pktRx_past.add(aux);
178               pktTx_past.add(aux);
179               bytesRx_past.add(aux);
180               bytesTx_past.add(aux);
181            }
182
183            for(int i=1;i<=Nhost_Real;i++) {
184               String deviceAso = portAssociate(deviceId("of:000000000000000"+i),
185                                    portNumber("2")); // device/port asociado
186               portState(getDeviceId(deviceAso),getPortNumber(deviceAso),false);
187            }
188
189            long aux2 = 0;
190            for(int i=1;i<=(Nhost_Real*2*kp+1);i++) {
191               rateRx_av.add(aux);
192               rateTx_av.add(aux);
193               delayRx_av.add(aux);
194               delayTx_av.add(aux);
195            }
```

```
196
197             WriteFile("Sondeo,Device,Port,rateRx,rateTx,delayRx,delayTx,duración,cambio",false);
198         }
199         flag = false;
200
201         /* Ciclo donde se analiza la red */
202         for(int i=1;i<=1000;i++){
203             int cont = 1; // Número de dispositivo
204             print("\nSondeo=%s", i);
205             for (Device d : getSortedDevices(deviceService)) {
206                 if (cont<=Nhost_Real) {
207                     printPortStats(d.id(),
208                             deviceService.getPortStatistics(d.id()),cont,i);
209                 }
210                 cont++;
211             }
212             try {
213                 Thread.sleep(t_s);
214             } catch (InterruptedException ignored) {
215
216             }
217         }
218
219     }
220
221     /**
222     * Imprime estadisticas de la red.
223     *
224     * @param deviceId
225     * @param portStats
226     */
227     private void printPortStats(DeviceId deviceId, Iterable<PortStatistics> portStats,
228                                 int Ndevice, int sequence) {
229         StatisticService service = get(StatisticService.class);
230         int cont = 1;
231         int cont2 = 1+(Ndevice-1)*2;
232         int cont3 = (Ndevice-1)*2*kp;
233         boolean flag_change = false;
234         boolean flag_aux = true;
235         for (PortStatistics stat : sortByPort(portStats)) {
236             if (cont<=2){ // No. de interfaces (WiFi - Ethernet)
237                 ConnectPoint cp = new ConnectPoint(deviceId, stat.portNumber());
238
239                 float time = ((float) stat.durationSec()) +
240                         (((float) stat.durationNano()) / TimeUnit.SECONDS.toNanos(1));
241                 float pktRx = stat.packetsReceived();
242                 float pktTx = stat.packetsSent();
243                 float bytesRx = stat.bytesReceived();
244                 float bytesTx = stat.bytesSent();
245
246                 // Calculate rate Rx/Tx [bit/s]
247                 float rateRx = ((bytesRx-bytesRx_past.get(cont2))*8)/(time-time_past.get(cont2));
248                 float rateTx = ((bytesTx-bytesTx_past.get(cont2))*8)/(time-time_past.get(cont2));
249
250                 // Calculate Delay [s]
251                 float delayRx = (time-time_past.get(cont2))/((pktRx-pktRx_past.get(cont2))*8);
252                 float delayTx = (time-time_past.get(cont2))/((pktTx-pktTx_past.get(cont2))*8);
253
254                 time_past.set(cont2,time);
255                 pktRx_past.set(cont2,pktRx);
256                 pktTx_past.set(cont2,pktTx);
257                 bytesRx_past.set(cont2,bytesRx);
258                 bytesTx_past.set(cont2,bytesTx);
259
260
261                 int pos = cont3+((cont-1)*kp)+sequence%kp;
262                 if(pos==0){ pos = kp; }
263
264                 rateRx_av.set(pos,rateRx);
265                 rateTx_av.set(pos,rateTx);
266                 delayRx_av.set(pos,delayRx);
267                 delayTx_av.set(pos,delayTx);
268
269
270                 // Si se realizo cambio de puerto, espera x segundos para volver a tomar una decisión
271                 // sobre dicho host
272                 if( ( (sequence>(t_c.get(cont2))) || !flag_time.get(cont2) ) && (sequence%kp)==0 ) {
273                     if(flag_time.get(cont2)){
274                         flag_time.set(cont2,false);
275                     }
276
277                     rateRx = 0;
278                     rateTx = 0;
279                     delayRx = 0;
280                     delayTx = 0;
281                     int aux_1 = 0;
282                     int aux_2 = 0;
283                     int aux_3 = 0;
284                     int aux_4 = 0;
285
286                     for(int i=(cont3+((cont-1)*kp));i<=(cont3+((cont-1)*kp)+kp);i++){
287                         if(!Float.toString(rateRx_av.get(i)).equals("NaN")) { rateRx = rateRx+rateRx_av.get(i);}
288                         else { aux_1++;}
289                         if(!Float.toString(rateTx_av.get(i)).equals("NaN")) { rateTx = rateTx+rateTx_av.get(i);}
290                         else { aux_2++;}
291                         if(!Float.toString(delayRx_av.get(i)).equals("NaN")) { delayRx = delayRx+delayRx_av.get(i);}
292                         else { aux_3++;}
```

```
293                     if(!Float.toString(delayTx_av.get(i)).equals("NaN")) { delayTx = delayTx+delayTx_av.get(i);}
294                     else { aux_4++;}
295                 }
296
297                 // Datos promediados
298                 rateRx = rateRx/(kp-aux_1);
299                 rateTx = rateTx/(kp-aux_2);
300                 delayRx = delayRx/(kp-aux_3);
301                 delayTx = delayTx/(kp-aux_4);
302
303                 if(flag_aux) {
304                     print("\ndeviceId=%s", deviceId);
305                     flag_aux = false;
306                 }
307                 if(Float.toString(delayTx).equals("Infinity")) { delayTx = 0;}
308                 if(Float.toString(delayRx).equals("Infinity")) { delayRx = 0;}
309                 print(FORMAT, stat.portNumber(), rateRx, rateTx, delayRx, delayTx, time);
310
311                 cambio = 0;
312
313                 if( (rateTx+rateRx)>Lim_rate || ( (delayTx>Lim_delayTx)&&((rateTx+rateRx)>(BW/2)) ) ||
314                     ( (delayRx>Lim_delayRx)&&((rateTx+rateRx)>(BW/2)) ) ) {
315                     String deviceAsso = portAssociate(deviceId("of:000000000000000"+Ndevice),
316                                                       portNumber("1")); // device/port associate
317                     if (!deviceAsso.equals("") && !flag_change && sequence!=1) {
318                         String uriAsso = getDeviceId(deviceAsso);
319                         String NportAsso = getPortNumber(deviceAsso);
320                         IntentService intentService = get(IntentService.class);
321                         boolean NportS = NportState(uriAsso,Integer.parseInt(NportAsso));
322                         if (NportS) {
323                             try {
324                                 portState(uriAsso,""+(Integer.parseInt(NportAsso)+1),true);
325                                 portState(getDeviceId(deviceAsso),getPortNumber(deviceAsso),false);
326                                 print("Cambio de puerto realizado en host "+Ndevice+"\n");
327                                 cambio = 1;
328
329                                 // guarda tiempo en que se debe volver a realizar cambio de puerto
330                                 t_c.set(cont2+1,sequence+kp*kc);
331                                 // Bandera para informar que se realizó el cambio en determinado puerto
332                                 flag_time.set(cont2+1,true);
333                                 flag_change = true;
334                             } catch (Exception e) { }
335                         }
336                     }
337                 } else if (cont!=1) {
338                     String deviceAsso2 = portAssociate(deviceId("of:000000000000000"+Ndevice),
339                                                        portNumber("2")); // device/port associate
340                     if (!deviceAsso2.equals("")  && !flag_change) {
341                         String uriAsso2 = getDeviceId(deviceAsso2);
342                         String NportAsso2 = getPortNumber(deviceAsso2);
343                         boolean NportS2 = NportState(uriAsso2,Integer.parseInt(NportAsso2));
344                         if (NportS2) {
345                             try {
346                                 portState(uriAsso2,""+(Integer.parseInt(NportAsso2)-1),true);
347                                 portState(getDeviceId(deviceAsso2),getPortNumber(deviceAsso2),false);
348                                 print("Cambio de puerto realizado en host "+Ndevice+"\n");
349                                 cambio = 2;
350
351                                 // guarda tiempo en que se debe volver a realizar cambio de puerto
352                                 t_c.set(cont2-1,sequence+kp*kc);
353                                 // Bandera para informar que se realizó el cambio en determinado puerto
354                                 flag_time.set(cont2-1,true);
355                                 flag_change = true;
356                             } catch (Exception e) { }
357                         }
358                     }
359                 }
360
361                 WriteFile(sequence+","+deviceId+","+stat.portNumber()+","+rateRx+","+rateTx+","+delayRx+","+delayTx+","+time
    +","+cambio,true);
362             }
363             cont2++;
364         }
365         cont++;
366     }
367 }
368
369 private static List<PortStatistics> sortByPort(Iterable<PortStatistics> portStats) {
370     List<PortStatistics> portStatsList = Lists.newArrayList(portStats);
371     portStatsList.sort(Comparator.comparing(ps -> ps.portNumber().toLong()));
372     return portStatsList;
373 }
374
375 /* Devuelve el puerto asociado */
376 private String portAssociate(DeviceId devId, PortNumber portN){
377     LinkService service = get(LinkService.class);
378     String dest = "";
379     for (Link link : service.getDeviceLinks(devId)) {
380         String aux1 = link.src().deviceId()+"/"+link.src().port();
381         String aux2 = devId+"/"+portN;
382         if (aux1.equals(aux2)) {
383             dest = link.dst().deviceId()+"/"+link.dst().port();
384         }
385     }
386     return dest;
387 }
388
```

```
    // Extrae porción uri (device id)
    private String getDeviceId(String deviceString) {
        int slash = deviceString.indexOf('/');
        if (slash <= 0) {
            return "";
        }
        return deviceString.substring(0, slash);
    }

    // Extrae porción de numero de puerto
    private String getPortNumber(String deviceString) {
        int slash = deviceString.indexOf('/');
        if (slash <= 0) {
            return "";
        }
        return deviceString.substring(slash + 1, deviceString.length());
    }

    // Return port state
    private boolean NportState(String uri, int Nport) {
        DeviceService service = get(DeviceService.class);
        Device device = service.getDevice(deviceId(uri));
        if (device == null) { }
        else {
           boolean aux = false;
           for (Port port : service.getPorts(device.id())) {
              if (aux) {
                 if (!port.isEnabled() && Integer.parseInt(port.number().toString())==Nport) {
                    return false; // Nport disable
                 }
              }
              aux = true;
           }
           return true;  // Nport enable
        }
        return true; // No find device
    }

    // Configure link
    private void configLink(String srcDevice, String dstDevice, boolean remove){
        DeviceService deviceService = get(DeviceService.class);
        NetworkConfigService netCfgService = get(NetworkConfigService.class);

        ConnectPoint srcCp = ConnectPoint.deviceConnectPoint(srcDevice);
        if (deviceService.getPort(srcCp) == null) {
            return;
        }

        ConnectPoint dstCp = ConnectPoint.deviceConnectPoint(dstDevice);
        if (deviceService.getPort(dstCp) == null) {
            return;
        }

        LinkKey link = linkKey(srcCp, dstCp);
        if (remove) {
            netCfgService.removeConfig(link, BasicLinkConfig.class);
            return;
        }

        BasicLinkConfig cfg = netCfgService.addConfig(link, BasicLinkConfig.class);
        cfg.isAllowed(false);
        cfg.apply();
    }

    private void WriteFile(String data, boolean flag)
    {
        FileWriter fichero = null;
        PrintWriter pw = null;
        try
        {
            fichero = new FileWriter("/home/sdn/Data.txt", flag);
            pw = new PrintWriter(fichero);
            pw.println(data);
        } catch (Exception e) {}
        finally {
           try {
           if (null != fichero)
              fichero.close();
           } catch (Exception e2) {}
        }
    }

    // Estado del puerto - set
    private void portState(String deviceID, String Nport, boolean state){
        DeviceService deviceService = get(DeviceService.class);
        DeviceAdminService deviceAdminService = get(DeviceAdminService.class);
        Device dev = deviceService.getDevice(DeviceId.deviceId(deviceID));
        PortNumber pnum = PortNumber.fromString(Nport);
        Port p = deviceService.getPort(dev.id(), pnum);
        deviceAdminService.changePortState(dev.id(), p.number(), state);
    }

}

/* BASADO EN CODIGO FUENTE DE LOS SGTES COMANDOS ONOS:
 *  get-stats
 *  portstats
 *  summary
```