UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y COMUNICACIONES

PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y COMUNICACIONES

TEMA:

"INTERNET DE LAS COSAS BASADO EN REDES DEFINIDAS POR SOFTWARE"

Trabajo de Graduación. Modalidad: Proyecto de Investigación, presentado previo la obtención del título de Ingeniero en Electrónica y Comunicaciones.

LÍNEAS DE INVESTIGACIÓN: Tecnologías de la Información y Sistemas de Control

AUTOR: Cristian Santiago Chiliquinga Rodríguez

TUTOR: Ing. Mg. Santiago Manzano

AMBATO – ECUADOR

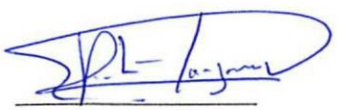
2019

APROBACIÓN DEL TUTOR

En mi calidad de tutor del trabajo de investigación sobre el tema: "INTERNET DE LAS COSAS BASADO EN REDES DEFINIDAS POR SOFTWARE" del señor Cristian Santiago Chiliquinga Rodríguez, estudiante de la Carrera de Ingeniería Electrónica y Comunicaciones , de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, considero que el informe de investigación reúne los requisitos suficientes para que continúe con los trámites y consiguiente aprobación de conformidad con el numeral 7.2 de los lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ambato enero, 2020

EL TUTOR



Ing. Mg. Santiago Manzano

AUTORÍA DEL TRABAJO DE TITULACIÓN

El presente trabajo de investigación titulado "INTERNET DE LAS COSAS BASADO EN REDES DEFINIDAS POR SOFTWARE" es absolutamente original, auténtico y personal en tal virtud, el contenido, efectos legales y académicas que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato enero, 2020

Cristian Santiago Chiliquinga Rodríguez

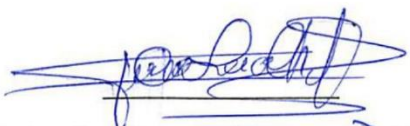
CC: 0502886922

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación como un documento disponible para la lectura, consulta y procesos de investigación.

Concedo los derechos de mi trabajo de Titulación, con fines de difusión pública, además autorizo su reproducción dentro de las regularidades de la Universidad.

Ambato enero, 2020



Cristian Santiago Chiliquinga Rodríguez

CC: 0502886922

APROBACIÓN DEL TRIBUNAL DEL GRADO

La Comisión Calificadora del presente trabajo conformada por los señores docentes aprobó el Informe Final del trabajo de graduación titulado "INTERNET DE LAS COSAS BASADO EN REDES DEFINIDAS POR SOFTWARE" presentado por el señor Cristian Santiago Chiliquinga Rodríguez de acuerdo al numeral 9.1 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato

Ing. Mg. Elsa Pilar Urrutia Urrutia

PRESIDENTA DEL TRIBUNAL

Ing. Mg. Geovanni Brito

DOCENTE CALIFICADOR

Ing. Mg. Andrea Sánchez

DOCENTE CALIFICADOR

AGRADECIMIENTO

A Dios por acompañarme en cada día de mis estudios y darme la fuerza necesaria para seguir adelante.

A mis padres por todo el sacrificio realizado y por su apoyo incondicional.

A mis hermanos por motivarme a no desistir de mis metas.

A mis amigos por compartir sus experiencias y conocimientos dentro y fuera del aula.

ÍNDICE GENERAL

APROBACIÓN DEL TUTOR ii
AUTORÍA DEL TRABAJO DE TITULACIÓN iii
DERECHOS DE AUTOR iv
APROBACIÓN DEL TRIBUNAL DEL GRADO v
AGRADECIMIENTO vi
ÍNDICE GENERAL vii
ÍNDICE DE TABLAS xi
RESUMEN EJECUTIVO xvii
CAPÍTULO I 1
MARCO TEÓRICO 1
1.1 Antecedentes investigativos 1
1.1.1 Fundamentación teórica 5
1.1.1.1 Redes definidas por software 5
1.1.1.2 Switch OpenFlow 8
1.1.1.3 Protocolo OpenFlow 12
1.1.1.4 Open vSwitch 14
1.1.1.5 Controlador SDN 17
1.1.1.6 Aplicaciones SDN 19
1.1.1.7 Controlador RYU 19
1.1.1.8 Internet de las cosas 23
1.1.1.9 Tecnologías soportadas por el IoT 30
1.1.1.10 Influencia de las SDN en aplicaciones del IoT 32
1.1.1.11 Industria 4.0 35
1.2 Objetivos 36
1.2.1 Objetivo General 36

1.2.2 Objetivos Específicos 37
CAPÍTULO II 39
METODOLOGÍA 39
2.1 Materiales 39
2.1.1 Selección de los componentes de hardware y software para el Testbed 39
2.1.1.1 Capa de procesos 39
2.1.1.2 Capa SDN 40
2.1.1.3 Capa de aplicación IoT 43
2.2 Métodos 44
2.2.1 Modalidad de investigación 44
2.2.2 Recolección de información 44
2.2.3 Procesamiento y análisis de datos 44
2.2.4 Desarrollo del proyecto 45
CAPÍTULO III 46
RESULTADOS Y DISCUSIÓN 46
3.1 Análisis y discusión de los resultados 46
3.1.1 Desarrollo de la propuesta 46
3.1.1.1 Descripción del testbed SDN-IoT 46
3.1.1.2 Direccionamiento y topología del prototipo 47
3.1.1.3 Implementación de los procesos de la industria 4.0 aplicados al IoT 49
3.1.1.4 Implementación de la Red Definida por Software 57
3.1.1.4.1 Implementación del controlador RYU compatible con OpenFlow 57
3.1.1.4.2 Implementación del Switch OpenFlow en la Raspberry Pi 3 Model B+ 59
3.1.1.4.3 Implementación de los Access Point OpenFlow en la Raspberry Pi 3 Model B 69
3.1.1.4.4 Implementación de la aplicación SDN 88

3.1.1.4.5 Implementación de la interfaz gráfica para el control de la aplicación Firewall del controlador RYU 100
3.1.1.5 Implementación del servidor IoT 103
3.1.1.5.1 Implementación de la interfaz gráfica de control y monitorización de los procesos de la industria 4.0. 105
3.1.1.6 Pruebas de funcionamiento 109
3.1.1.6.1 Pruebas de la aplicación Firewall en el Testbed 109
3.1.1.6.2 Pruebas de comunicación entre los elementos OpenFlow de la SDN 119
3.1.1.6.3 Pruebas de tráfico de datos en la SDN 129
3.1.1.7 Costos del prototipo (Testbed) 131
CAPÍTULO IV 134
CONCLUSIONES Y RECOMENDACIONES 134
4.1 Conclusiones 134
4.2 Recomendaciones 135
BIBLIOGRAFÍA 136
ANEXOS 141
ANEXO A: Código fuente para la Configuración de las NodeMCUs 141
ANEXO B: Código fuente para la implementación de la cámara IP en la Raspberry Pi 146
ANEXO C: Código fuente para la Configuración del Open vSwitch en el conmutador OpenFlow 148
ANEXO D: Código fuente para la Configuración del Open vSwitch en los Access Point OpenFlow 149
ANEXO E: Código fuente de la aplicación SDN 152
ANEXO F: Código fuente de la interfaz gráfica de usuario para el control de la aplicación SDN 163
ANEXO G: Código fuente para la inserción de acciones en la interfaz gráfica de usuario para el control de la aplicación SDN 173

ANEXO H: Código fuente para la creación de la interfaz gráfica de la aplicación IoT 180

ÍNDICE DE TABLAS

Tabla 1. Longitudes de los campos de cabecera y la forma en que deben aplicarse a las entradas de flujo 11
Tabla 2. Campos de coincidencia de las versiones OpenFlow (OF)..... 13
Tabla 3. Las estadísticas se miden para diferentes partes del interruptor OpenFlow 13
Tabla 4. Versión OpenFlow compatible para diferentes versiones de Open vSwitch16
Tabla 5. Características de los controladores más utilizados..... 18
Tabla 6. Mensajes del protocolo OpenFlow, estructuras y API correspondiente 21
Tabla 7. Campos de aplicación del IoT y sus funciones..... 28
Tabla 8. Comparación de las plataformas de hardware compatibles con IoT existentes 31
Tabla 9. Comparación de las tecnologías de comunicación existentes..... 32
Tabla 10. Direccionamiento de la SDN. 48
Tabla 11. Direccionamiento de la WLAN. 48
Tabla 12. Uso de los canales inalámbricos. 70
Tabla 13. Parámetros inalámbricos de los AP. 71
Tabla 14. Métodos declarados de la clase ControladorFirewall. 94
Tabla 15. Servicios del Testbed. 110
Tabla 16. Descripción de los Mensajes iniciales entre el controlador y los conmutadores 120
Tabla 17. Resumen de los paquetes capturados en el intercambio de mensajes HELLO. 122
Tabla 18. Resumen de los paquetes capturados en el intercambio de mensajes FEATURES..... 124
Tabla 19. Resumen de los paquetes capturados en el intercambio de mensajes MULTIPART..... 126
Tabla 20. Resumen de los paquetes capturados en el intercambio de mensajes ECHO. 127
Tabla 21. Resumen de los paquetes capturados en el envío de mensajes FLOW_MOD. 129
Tabla 22. Costos del prototipo implementado. 132
Tabla 23. Comparación de precios de los dispositivos OpenFlow 133

ÍNDICE DE FIGURA S

Figura 1. Arquitectura SDN 8
Figura 2. Comunicación entre un switch OpenFlow y un controlador SDN a través de un canal seguro mediante el protocolo OpenFlow. 8
Figura 3. Funcionamiento interno OvS 14
Figura 4. Arquitectura Open vSwitch que representa los flujos de procesamiento de datos 15
Figura 5. Clasificación de los Controladores SDN 18
Figura 6. Ubicación del Framework de Ryu 19
Figura 7. Arquitectura RYU 20
Figura 8. Arquitectura funcional de la aplicación Ryu 22
Figura 9. Arquitectura general del IoT. 24
Figura 10. Protocolos IoT. 25
Figura 11. Arquitectura IoT-SDN. 27
Figura 12.Diagrama de bloques del proceso uno. 39
Figura 13. Diagrama de bloques del proceso uno. 40
Figura 14. Diagrama de bloques del Controlador SDN. 41
Figura 15. Diagrama de bloques del switch OpenFlow. 42
Figura 16. Diagrama de bloques del Access Point OpenFlow. 43
Figura 17. Arquitectura IoT-SDN del prototipo a implementar. 46
Figura 18.Topología SDN-IoT del Testbed. 49
Figura 19. Diagrama eléctrico de potencia del proceso uno. 50
Figura 20. Circuito de potencia del proceso uno implementado. 51
Figura 21. Diagrama neumático del proceso dos. 51
Figura 22. Circuito neumático del proceso dos implementado. 52
Figura 23. Diagrama electrónico del proceso uno. 52
Figura 24. Diagrama electrónico del proceso dos. 53
Figura 25. Direccionamiento de la cámara IP del proceso uno 54
Figura 26. Parámetros de conexión al AP1. 54
Figura 27. Ubicación de los archivos para el Streaming de video a través de la Raspberry Pi. 54

Figura 28. Configuración de los parámetros de red del smartphone para la conexión con el AP2. 55
Figura 29. Ejecución de la aplicación DroidCam en el smartphone. 56
Figura 30. Proceso uno implementado. 56
Figura 31. Proceso dos implementado. 57
Figura 32.Versión instalada y arranque del controlador RYU. 58
Figura 33. Dirección IP estática del controlador RYU. 59
Figura 34. Versión actualizada de Raspbian 59
Figura 35. Descompresión del S.O. Raspbian. 60
Figura 36. Escritura de la imagen con Raspbian en la tarjeta microSD. 60
Figura 37. Cabeceras de Linux. 62
Figura 38. Datapath de Linux sin el módulo Open vSwitch. 63
Figura 39. Datapath de Linux con el módulo Open vSwitch. 63
Figura 40. Script de inicio sw_start.sh. 64
Figura 41. Archivo de Configuración "rc.local". 64
Figura 42. Identificador de la primera escritura de datos del Open vSwitch. 65
Figura 43. Versión y componentes del Open vSwitch instalado. 65
Figura 44. Interfaces disponibles en la Raspberry Pi. 66
Figura 45. Configuración de la dirección IP estática para la interfaz eth0. 66
Figura 46. Configuración interna del Open vSwitch. 68
Figura 47. Diagrama de flujo del Switch OpenFlow implementado. 69
Figura 48. Canales inalámbricos ocupados. 70
Figura 49. Imagen iso de OpenWrt para descargar. 71
Figura 50. Descompresión de la imagen iso de OpenWRT. 72
Figura 51. Escritura de la imagen con OpenWrt en la tarjeta microSD. 72
Figura 52. Detección de la dirección IP de la Raspberry Pi analizada con Wireless Network Watcher. 73
Figura 53. Acceso remoto a la Raspberry Pi a través de Putty. 73
Figura 54. Acceso al enrutador OpenWRT. 74
Figura 55. Creación de una contraseña para el acceso a OpenWRT. 74
Figura 56. Detección de la nueva dirección IP de la Raspberry Pi. 75
Figura 57. Interfaz gráfica LUCI del enrutador OpenWrt. 75
Figura 58. Configuración del protocolo DHCP client en el enrutador OpenWrt. 76

Figura 59. Dirección IP asignada por DHCP en la Raspberry Pi..... 76
Figura 60. Paquetes Open vSwitch disponibles. 77
Figura 61. Versión de Open vSwitch instalada en OpenWrt. 77
Figura 62. Lista de paquetes USB instalados en OpenWrt. 78
Figura 63. Nueva lista de paquetes USB instalados en OpenWrt. 78
Figura 64. Comprobación de la conexión del adaptador USB Ethernet. 79
Figura 65. Configuración de la interfaz eth1 en OpenWrt. 79
Figura 66. Adaptador USB Ethernet instalado en OpenWrt. 80
Figura 67. Configuración de IP estática para la interfaz LAN en OpenWrt. 80
Figura 68. Desactivación del servidor DHCP en OpenWrt. 81
Figura 69. Verificación de la IP estática establecida para la interfaz LAN. 81
Figura 70. Creación de una interfaz inalámbrica en OpenWrt. 82
Figura 71. Configuración de IP estática para la interfaz WIFI en OpenWrt. 82
Figura 72. Configuración general del AP1. 83
Figura 73. Configuración general de la interfaz inalámbrica del AP1. 84
Figura 74. Configuración de la seguridad inalámbrica del AP1. 84
Figura 75. Configuración de una zona firewall sobre la interfaz WIFI. 85
Figura 76. Verificación de la zona firewall creada. 85
Figura 77. Archivo de Configuración para el inicio de archivos en el arranque de OpenWrt. 87
Figura 78. Configuración del Open vSwitch en OpenWrt. 87
Figura 79. Diagrama de flujo del script AP1_start.sh. 88
Figura 80. Diagrama de flujo de la aplicación firewall SDN. 89
Figura 81. Ejecución de la aplicación appFirewall.py. 100
Figura 82. Interfaz gráfica de control de la aplicación firewall creada en Glade. ... 101
Figura 83. Sección de la interfaz gráfica para el ingreso de reglas. 102
Figura 84. Sección de la interfaz gráfica para la visualización de reglas. 103
Figura 85. Programación de del Flow 1 en Node-RED. 105
Figura 86. Programación de del Flow 2 en Node-RED. 105
Figura 87. Ejecución del servicio de Apache en XAMPP. 106
Figura 88. Archivos de Configuración para la página web con la interfaz gráfica de control y monitorización. 106
Figura 89. Página web de inicio. 107

Figura 90. Página web con la interfaz gráfica de control y monitorización. 107
Figura 91. Sección de control de la interfaz gráfica del servidor IoT. 108
Figura 92. Sección de monitorización de la interfaz gráfica del servidor IoT. 108
Figura 93. Implementación de las capas SDN y aplicación IoT del prototipo. 109
Figura 94. Bloqueo general del tráfico de datos en la SDN. 111
Figura 95. Bloqueo de paquetes ICMP en la SDN. 111
Figura 96. Desbloqueo general del tráfico de datos en la SDN. 112
Figura 97. Pruebas de conectividad exitosa en la SDN. 112
Figura 98.Interfaz gráfica de la aplicación IoT sin acceso a los dominios de los dispositivos IoT. 113
Figura 99. Ejecución del servicio MQQT en el servidor IoT. 114
Figura 100. Ingreso de reglas con el protocolo MQTT. 114
Figura 101. Recepción de datos en el servidor IoT a través del protocolo MQTT. . 115
Figura 102. Ingreso de reglas con los protocolos SSH y HTTP. 115
Figura 103. Conexión SSH desde el servidor IoT exitosa. 116
Figura 104. Ejecución de la cámara IP en el proceso uno. 116
Figura 105. Conexión HTTP desde el servidor IoT exitosa. 117
Figura 106. Ingreso de regla con el protocolo PGPfone. 117
Figura 107. Conexión PGPfone desde el servidor IoT exitosa. 118
Figura 108. Activación y monitorización de los procesos en la industria 4.0. 118
Figura 109. Desactivación y monitorización de los procesos en la industria 4.0. ... 119
Figura 110. Mensajes iniciales entre el controlador y los conmutadores. 119
Figura 111. Paquetes capturados de los mensajes HELLO entre el controlador y el switch. 121
Figura 112. Paquetes capturados de los mensajes HELLO entre el controlador y el AP1. 122
Figura 113. Paquetes capturados de los mensajes HELLO entre el controlador y el AP2. 122
Figura 114. Paquetes capturados de los mensajes FEATURES entre el controlador y el switch. 123
Figura 115. Paquetes capturados de los mensajes FEATURES entre el controlador y el AP1. 123

Figura 116. Paquetes capturados de los mensajes FEATURES entre el controlador y el AP2. 124
Figura 117. Paquetes capturados de los mensajes MULTIPART entre el controlador y el switch. 125
Figura 118. Paquetes capturados de los mensajes MULTIPART entre el controlador y el AP1. 125
Figura 119. Paquetes capturados de los mensajes MULTIPART entre el controlador y el AP2. 126
Figura 120. Paquetes capturados de los mensajes ECHO entre el controlador y el switch. 127
Figura 121. Paquetes capturados de los mensajes ECHO entre el controlador y el AP1. 127
Figura 122. Paquetes capturados de los mensajes ECHO entre el controlador y el AP2. 127
Figura 123. Paquetes capturados en el envío del mensaje FLOW_MOD desde el controlador al switch. 128
Figura 124. Paquetes capturados en el envío del mensaje FLOW_MOD desde el controlador al AP1. 128
Figura 125. Paquetes capturados en el envío del mensaje FLOW_MOD desde el controlador al AP2. 129
Figura 126. Throughput entre la cámara IP del proceso uno y el servidor IoT. 130
Figura 127. Throughput entre la cámara IP del proceso dos y el servidor IoT. 131

RESUMEN EJECUTIVO

El presente proyecto diseña e implementa un testbed SDN-IoT, con el objetivo de mostrar la influencia que tienen las redes definidas por software sobre las aplicaciones del IoT aplicado a la industria 4.0.

La arquitectura IoT-SDN es considerada como base para la implementación del prototipo, la cual se presenta en tres capas principales: capa de procesos, capa SDN y capa de aplicación IoT. En la capa de procesos se implementan 2 procesos de la industria 4.0, en la capa SDN se implementan dos puntos de acceso, un conmutador, un controlador SDN y una aplicación firewall para el control centralizado del tráfico sobre la red, finalmente en la capa de aplicación IoT se implementa una interfaz gráfica que permite controlar y monitorizar los procesos de la industria 4.0.

En la implementación de la SDN se utilizan elementos de bajo costo y de código libre que son configura dos para trabajar con el protocolo OpenFlow en la versión 1.3, evidenciando el intercambio de mensajes entre los dispositivos de red y el controlador SDN en las pruebas realizadas.

CAPÍTULO I

MARCO TEÓRICO

1.1 Antecedentes investigativos

La implementación innumerable del internet de las cosas (IoT) está extendiendo la conectividad a Internet entre miles de millones de dispositivos y esto conlleva un problema en las tareas de administración y control de la red. Según el informe de Cisco sobre el crecimiento de IoT, a nivel mundial, las conexiones máquina a máquina (M2M) se multiplicarán por un factor de 2.4, de 6.100 millones de dispositivos en 2017 a 14.600 millones en 2022. Si bien el número de conexiones aumenta en un factor de 2.4 veces, el tráfico IP máquina a máquina (M2M) global crecerá más de siete veces durante este mismo período, de 3.7 EB por mes en 2017 (3 por ciento del tráfico IP global) a más de 25 EB para 2022 (6 por ciento del tráfico IP global) [1].

La cantidad de tráfico en IoT está creciendo más rápido que la cantidad de conexiones debido al aumento de la implementación de aplicaciones de video en las conexiones M2M, como la telemedicina y los sistemas de navegación inteligente para automóviles, que requieren un mayor ancho de banda y una menor latencia [2].

El Centro de Información de Privacidad Electrónica (EPIC por sus siglas en ingles) en Washington, D.C., emitió una carta al Comité del Congreso de EE. UU. relacionada a la protección del consumidor y comercio, que trata el tema de la seguridad del consumidor cuando utiliza dispositivos de Internet de las cosas (IoT). En la carta se enuncia que: La recopilación no regulada de datos personales y el crecimiento del Internet de las cosas ("IoT") plantea una gran variedad de problemas de privacidad y seguridad de los datos del consumidor. Las agencias gubernamentales, como la Comisión Federal de Comercio, están preocupadas por temas como la seguridad de los datos, la privacidad móvil y el big data. Los consumidores estadounidenses enfrentan amenazas de seguridad y privacidad sin precedentes desde dispositivos conectados a Internet [3]. Es por ello que, para el desarrollo de ciudades inteligentes rentables, escalables, confiables y resistentes, es esencial que las tecnologías de IoT sean simples,

uniformes y admitan una infraestructura de comunicación segura. Dado que la utilización de IoT en ciudades inteligentes proporciona una integración perfecta entre los mundos físico y cibernético, las debilidades de seguridad en el mundo cibernético no solo amenazan el mundo virtual, sino también el mundo físico real. En este contexto, es crucial que los dispositivos de IoT operen en un entorno seguro [4].

Dos de los desafíos más importantes en la seguridad de IoT son la heterogeneidad y la escalabilidad. A diferencia de los dispositivos tradicionales con recursos adecuados de computación, procesamiento y almacenamiento, los dispositivos de IoT, como sensores y dispositivos móviles, tienen recursos limitados. El diseño de los protocolos de seguridad para IoT, por lo tanto, debe considerar las diferentes capacidades de recursos de los dispositivos heterogéneos que forman parte del entorno de IoT [5].

La implementación de las redes de sensores inalámbricos bajo el contexto de Internet de las cosas probablemente se acerque a las limitaciones de la disponibilidad del ancho de banda de la red y el costo de la infraestructura loT será extremadamente alto si se implementan recursos de red dedicados [6].

Beneficiándose principalmente, las pequeñas, medianas y grandes empresas en los diversos campos de aplicación del IoT, que requieran un nuevo enfoque de diseño y administración para redes, basándose en la integración de una nueva tecnología compatible con redes heterogéneas, de rápida evolución y dinamismo que permita cumplir con las expectativas de control y gestión de la red en numerosas áreas de aplicación como: sistemas productivos, logística, comunicaciones y seguridad.

El desarrollo de este proyecto será de gran utilidad, puesto que el internet de las cosas (IoT) y las redes definidas por software (SDN) están transformando la interacción entre el ciberespacio y el espacio físico, con un tremendo impacto en la vida cotidiana. La efectividad de estas tecnologías presenta nuevos enfoques metodológicos y de ingeniería debido a la impresionante escala del problema y las nuevas solicitudes desafiantes en términos de rendimiento, seguridad y confiabilidad.

Realizar un proyecto investigativo de esta magnitud llamará la atención de las personas involucradas en el paradigma que presenta la integración de aplicaciones IoT basadas en redes definidas por software, ante diferentes desafíos de escalabilidad, heterogeneidad y seguridad que presentan las redes tradicionales en diferentes campos

de aplicación. Además, el desarrollo del presente proyecto brindará una importancia teórica-práctica para futuras propuestas.

Debido a que se trata de un tema extenso y de interés mundial se lograron encontrar varias investigaciones de gran importancia enunciadas a continuación:

En el año 2019 el artículo científico denominado "Secure and Reliable IoT Networks Using Fog Computing with Software-Defined Networking and Blockchain" fue publicado en la revista Sensor and Actuator Networks. Este documento, presenta un marco para IoT que emplea una capa de computación perimetral de nodos de niebla controlada y administrada por una red de SDN para lograr una alta confiabilidad y disponibilidad para aplicaciones de IoT sensibles a la latencia. La red SDN es equipada con controladores distribuidos y conmutadores OpenFlow restringidos de recursos distribuidos. Blockchain se utiliza para asegurar la descentralización de una manera confiada. Además, se desarrolla un algoritmo de descarga de datos para asignar varias tareas de procesamiento y computación a los conmutadores de OpenFlow en función de su carga de trabajo actual. Finalmente, se propone un prototipo para modelar y analizar las partes indiferentes del tráfico de la red. El algoritmo propuesto se evalúa en simulación y en un banco de pruebas. Los resultados experimentales demuestran que el marco propuesto logra una mayor eficiencia en términos de latencia y utilización de recursos [7].

En el año 2018 Jun-Hong Park, Hyeong-Su Kim y Won-Tae Kim publican un artículo científico denominado "DM-MQTT: An Efficient MQTT Based on SDN Multicast for Massive IoT Communications" en la revista Sensors. Los autores de este artículo proponen un MQTT novedoso con un mecanismo de multidifusión para minimizar el retardo de transferencia de datos y el uso de la red para las comunicaciones masivas de IoT. El MQTT propuesto reduce los retrasos en la transferencia de datos estableciendo los árboles de multidifusión de SDN (software Defined Networking) bidireccional entre los editores y los suscriptores por medio de la derivación del intermediario centralizado. Como resultado, reducen el retardo de transmisión en un 65% y el uso de la red en un 58% en comparación con el MQTT estándar [8].

Yifei Lu, Zhen Ling, Shuhong Zhu y Ling Tang en el año 2017 publicaron un artículo científico titulado "SDTCP: Towards Datacenter TCP Congestion Control with SDN

for IoT Applications". Este documento, propone un mecanismo de control de congestión TCP basado en la red definida por software (SDN) para aplicaciones IoT, denominado SDTCP, para aprovechar las características, por ejemplo, los métodos de control centralizados y la vista global de la red, con el fin de resolver los problemas de INCAST de TCP. Al detectar la congestión de la red en un conmutador OpenFlow, el controlador puede seleccionar los flujos de fondo y reducir su ancho de banda ajustando la ventana anunciada de paquetes ACK TCP de los flujos de fondo correspondientes para reservar más ancho de banda para flujos de ráfagas. SDTCP es transparente para los sistemas finales y puede desacelerar con precisión la tasa de flujos de fondo aprovechando la visión global de la red obtenida a través de SDN. Los experimentos realizados demuestran que SDTCP puede proporcionar una alta tolerancia para los flujos de ráfaga y lograr un mejor tiempo de terminación de flujo para flujos cortos. Finalmente se concluye que, SDTCP es una solución eficaz y escalable para el problema de INCAST de TCP [9].

En el año 2016 el artículo científico denominado "Software-Defined Fog Network Architecture for IoT" fue publicado por Slavica Tomovic,Ivo Maljevic, Igor Radusinovic y Kenji Yoshigoe en la revista Springer en New York. En este documento, se propone una arquitectura para IoT, que se basa en dos tecnologías emergentes: SDN y la computación de niebla. La arquitectura propuesta está diseñada en la forma de admitir un alto nivel de escalabilidad, entrega de datos en tiempo real y movilidad. Esta arquitectura de IoT aborda múltiples desafíos combinando SDN y la computación de niebla juntos en un sistema y adaptándolos entre sí. En particular, la funcionalidad de la orquestación de niebla se delega al controlador SDN con el fin de lograr una mayor eficiencia, mientras que el problema de escalabilidad SDN se relaja delegando algunas tareas del controlador a nodos de niebla. Al final se dan a conocer los beneficios que se podrían tener al implementar la arquitectura propuesta a través de la ilustración de casos de uso que van desde el transporte inteligente hasta la video vigilancia [10].

El artículo científico desarrollado por Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, y Slim Abdellatif en el 2015, denominado "Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications", describe los problemas más importantes relacionados con los esfuerzos de estandarización, la

movilidad de objetos, el acceso a la red y la puerta de enlace, y el soporte de QoS. En particular, se describe una arquitectura de red IoT novedosa que integra redes definidas por software (SDN) y el middleware del servicio de distribución de datos (DDS) de Object Management Group [11].

En el año 2019 Liz Panamá Carrasco Espinosa dentro de su trabajo fin de máster titulado "Estudio sobre la aplicación de las redes definidas por software a la internet de las cosas. Diseño de un escenario de Pruebas" en la Universidad Politécnica de Madrid hace una revisión del estado actual de las tecnologías SDN-IoT, identificando los problemas que está atravesando la IoT y de qué manera se pueden solucionar utilizando SDN. Esto se muestra mediante el diseño e implementación de un caso de estudio en escenarios virtuales, simulando una red residencial. El escenario es realizado mediante la herramienta de creación de escenarios virtuales VNX, donde se simula una red IoT-SDN con dispositivos con SO TinyCore, Linux y el controlador RYU [12].

1.1.1 Fundamentación teórica

1.1.1.1 Redes definidas por software

Introducción

Dado que las redes han aumentado mucho en tamaño y en requisitos, moverse por los conmutadores de hardware se ha convertido en una carga. Incluso la Configuración manual de switches de software individuales se ha convertido en una tarea complicada y propensa a errores para las empresas que ejecutan entornos fuertemente virtualizados junto con grandes redes. Aquí es donde las redes definidas por software (SDN), para abreviar, ingresa al juego, a principios de la década de 2010 [13].

"SDN" es parecida un poco a la "computación en la nube": es un tema de moda en computación y, a menudo, se vende como una solución "milagrosa" para todos los problemas de infraestructura de red. Pero además del marketing, también corresponde a un nuevo tipo de arquitectura de red. "Software definido" no significa que solo use conmutadores virtuales en lugar de hardware dedicado (incluso si lo hace en su

mayoría): se refiere al hecho de que los conmutadores pueden programarse. Su comportamiento está definido por una Configuración de software [13].

Definición

La red definida por software (SDN) es una arquitectura emergente que es dinámica, manejable, rentable y adaptable, lo que la hace ideal para la naturaleza dinámica y de alto ancho de banda de las aplicaciones actuales. Esta arquitectura desacopla las funciones de control y reenvío de la red, lo que permite que el control de la red sea programable directamente y la infraestructura subyacente se abstraiga para las aplicaciones y los servicios de red. El protocolo OpenFlow es un elemento fundamental para construir soluciones SDN [14].

Funcionalidades de las SDN

La capa de red en el IoT proporciona conectividad y también debe ocuparse de la administración en términos de seguridad, procesamiento de información, conversión de protocolos y manejo de fallas. Estos desafíos se ven abordados con el enfoque SDN a través de las siguientes funcionalidades:

- *Directamente programable:* El control de la red se puede programar directamente porque está desacoplado de las funciones de reenvío.
- *Ágil:* Al abstraer el control del reenvío, los administradores pueden ajustar dinámicamente el flujo de tráfico en toda la red para satisfacer las necesidades cambiantes.
- *Gestión central:* 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 políticas de las máquinas.
- *Configura ble mediante programación:* SDN permite que los administradores de red configuren, administren, protejan y optimicen los recursos de la red muy rápidamente a través de programas dinámicos y automatizados de SDN, que pueden escribirse por sí mismos porque los programas no dependen de software propietario.

Basado en estándares abiertos y neutrales: 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 [14].

Arquitectura

La arquitectura de una red definida por software ilustrada en la Figura 1 se basa en tres amplias capas: infraestructura, control y aplicación, además las respectivas interfaces entre ellas [15].

- *Capa de infraestructura:* es también llamada plano de datos, comprende los elementos de la red de reenvío que podrían ser un conjunto de conmutadores y enrutadores. Las responsabilidades del plano de reenvío son principalmente el reenvío de datos, así como el monitoreo de la información local y la recopilación de estadísticas.
- *Capa de control o plano de control:* es responsable de la programación y gestión del plano de reenvío. Para ese fin, utiliza la información proporcionada por el plano de reenvío y define el funcionamiento y enrutamiento de la red. Comprende uno o más controladores de software que se comunican con los elementos de red de reenvío a través de interfaces estandarizadas, que se conocen como Southbound Interfaces (interfaces en dirección sur). OpenFlow, que es una Southbound Interface más utilizada, principalmente considera conmutadores, mientras que otros enfoques SDN consideran otros elementos de red, como enrutadores.
- *Capa de aplicación:* contiene aplicaciones de red que pueden introducir nuevas funciones de red, como seguridad y capacidad de administración, esquemas de reenvío o asistencia a la capa de control. La capa de aplicación puede recibir una vista global de la red y utilizar esa información para proporcionar una guía adecuada a la capa de control. La interfaz entre la capa de aplicación y la capa de control se conoce como la Northbound Interface (interfaz hacia el norte). Para este último, no existe una API estandarizada en la actualidad, y en la práctica, el software de control proporciona su propia API a las aplicaciones [15].

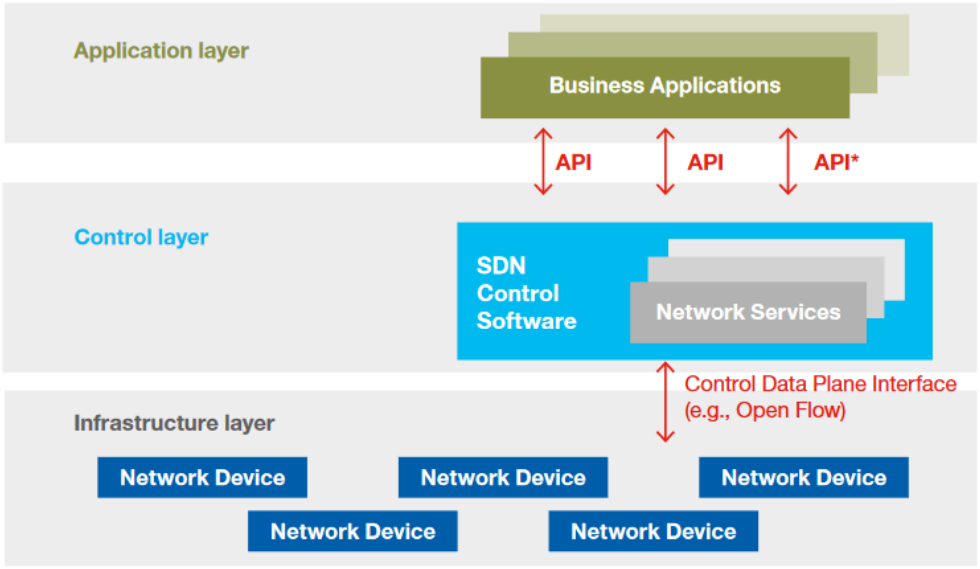


Figura 1. Arquitectura SDN *[16]*.

1.1.1.2 Switch OpenFlow

Un switch OpenFlow consta de una tabla de flujo, que realiza la búsqueda y reenvío de paquetes, y un canal seguro a un controlador externo Figura 2. El controlador gestiona el conmutador sobre el canal seguro mediante el protocolo OpenFlow [17].

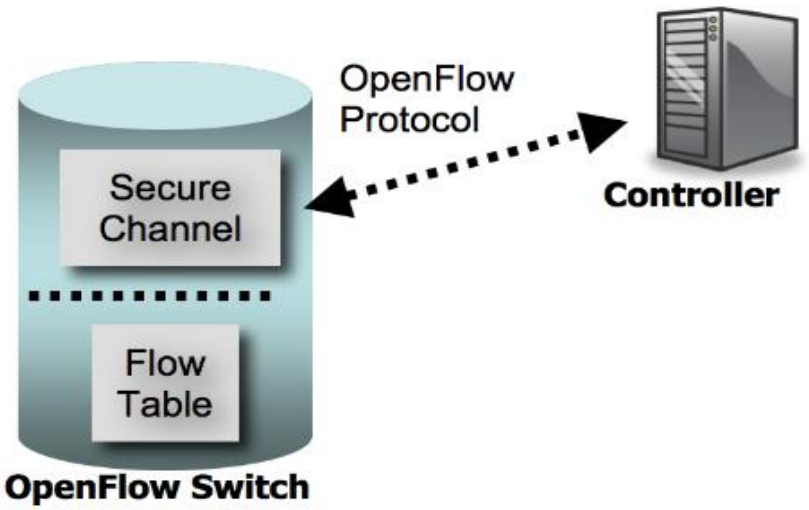


Figura 2. Comunicación entre un switch OpenFlow y un controlador SDN a través de un canal seguro mediante el protocolo OpenFlow *[17]*.

Tabla de flujos

La tabla de flujo contiene un conjunto de entradas de flujo (valores de encabezado para que coincidan con los paquetes), contadores de actividad y un conjunto de cero o más

acciones que se aplicarán a los paquetes coincidentes. Todos los paquetes procesados por el switch se comparan con la tabla de flujo. Si se encuentra una entrada coincidente, todas las acciones para esa entrada se realizan en el paquete (por ejemplo, la acción podría ser reenviar un paquete fuera de un puerto especificado). Si no se encuentra ninguna coincidencia, el paquete se remite al controlador sobre el canal seguro. El controlador es responsable de determinar cómo manejar los paquetes sin las entradas de flujo válidas, y maneja la tabla de flujo del switch agregando y quitando las entradas de flujo [17].

Cada entrada de la tabla de flujo contiene:

- *Campos de encabezado* para que coincida con los paquetes.

La Tabla 1 muestra los campos de encabezado con los que se compara un paquete entrante y los detalles sobre las propiedades de cada uno. Cada entrada contiene un valor específico, o ANY, que coincide con cualquier valor. Si el conmutador admite máscaras de subred en los campos de origen y/o destino de IP, estos pueden especificar coincidencias con mayor precisión.

- *Contadores* para actualizar el paquete correspondiente

Los contadores se mantienen por tabla, por flujo, por puerto y por cola. Los contadores están reservados para recopilar estadísticas sobre flujos. Almacenan el número de paquetes y bytes recibidos, así como la duración del flujo.

- *Acciones* para aplicar a paquetes coincidentes.

Cada entrada de flujo está asociada con cero o más acciones que dictan cómo el conmutador maneja los paquetes coincidentes. Si no hay acciones de reenvío, el paquete se descarta. Un conmutador puede rechazar una entrada de flujo si no puede procesar la lista de acciones en el orden especificado, en cuyo caso debería devolver inmediatamente un error de flujo no admitido.

Las acciones se clasifican en “Acciones requeridas” y “Acciones opcionales” ya que no es necesario que un conmutador admita todos los tipos de acciones. Cuando se conecta al controlador, un conmutador indica cuál de las "Acciones opcionales" es compatible.

- **Acción requerida:** *Adelante*. Los switches OpenFlow deben admitir el reenvío del paquete a los puertos físicos y a los siguientes virtuales:

  ALL: Envíe el paquete hacia todas las interfaces, sin incluir la interfaz entrante.

  CONTROLLER: encapsular y enviar el paquete al controlador.

  LOCAL: Envíe el paquete a la pila de red local del switchs.

  TABLA: realice acciones en la tabla de flujo. Sólo para los mensajes de salida de paquetes.

  IN PORT: Envíe el paquete hacia fuera del puerto de entrada

- **Acción opcional:** *Adelante.* El conmutador puede admitir opcionalmente los siguientes puertos virtuales:

  NORMAL: Procese el paquete utilizando la ruta de reenvío tradicional admitida por el conmutador.

  FLOOD: Inunde el paquete a lo largo del árbol de expansión mínimo, sin incluir la interfaz entrante

- **Acción opcional:** *Encolar*. La acción de puesta en cola reenvía un paquete a través de una cola adjunta a un puerto. El comportamiento de reenvío es dictado por la Configuración de la cola y se usa para proporcionar soporte básico de Calidad de Servicio (QoS).
- **Acción requerida:** *Drop*. Una entrada de flujo sin acción específica indica que todos los paquetes coincidentes deben eliminarse [17].

Tabla 1. Longitudes de los campos de cabecera y la forma en que deben aplicarse a las entradas de flujo *[17]*.

Campo	Bits	Aplicable
Puerto entrante	Dependiente de la implementación	Todos los paquetes
Dirección MAC origen	48	Todos los paquetes en los puertos habilitados
Dirección MAC destino	48	Todos los paquetes en los puertos habilitados
Tipo Ethernet	16	Todos los paquetes en los puertos habilitados
VLAN id	12	Todos los paquetes de tipo Ethernet 0x8100
Prioridad VLAN	3	Todos los paquetes de tipo Ethernet 0x8100
Dirección IP origen	32	Todos los paquetes IP y ARP
Dirección IP destino	32	Todos los paquetes IP y ARP
Protocolo IP	8	Todos los paquetes IP y ARP
Bits ToS (tipo de servicio) IP	6	Todos los paquetes IP
Transport source port / ICMP Type	16	Todos los paquetes TCP, UDP e ICMP
Transport destination port / ICMP Code	16	Todos los paquetes TCP, UDP e ICMP

Canal seguro

El canal seguro es la interfaz que conecta cada conmutador OpenFlow a un controlador. A través de esta interfaz, el controlador configura y administra el conmutador, recibe eventos del conmutador y envía paquetes fuera del conmutador. Entre la ruta de datos y el canal seguro, la interfaz es específica de la implementación, sin embargo, todos los mensajes de canal seguro deben formatearse de acuerdo con el protocolo OpenFlow [17].

Tipos de switches OpenFlow

- *Switches OpenFlow-Only*

Este tipo de switches soportan únicamente el protocolo OpenFlow y no tienen características de control en su interior es decir confían plenamente en un controlador para la toma de decisiones. Un switch que soporta OpenFlow, contiene múltiples tablas de flujos, y cada flujo contiene múltiples entradas de flujo. El procesado define como los paquetes interactuarán con estas tablas de flujos. Requiere que tenga al menos una tabla de flujo y opcionalmente más. Cuantas menos entradas más simplificado será el procesado [18].

- *Switches OpenFlow-Híbridos*

Este tipo de Switches que soportan tanto la operación OpenFlow como el Switching Ethernet convencional. Las operaciones habituales pueden ser: Switching Ethernet de Capa 2, aislamiento de tráfico con VLAN, nivel 3 o de routing (IPV4, IPV6), listas de acceso o QoS. Estos switches deberán proveer un mecanismo de clasificación fuera de OpenFlow que enrute el tráfico o bien ser procesado por OpenFlow. Por ejemplo, un switch deberá usar una etiqueta VLAN o puerto de entrada para decidir si se procesa el paquete de una manera u otra [18].

1.1.1.3 Protocolo OpenFlow

Una de las bases del concepto de SDN es el protocolo OpenFlow, propuesto por miembros de diversas universidades, como Stanford, Berkeley y MIT, como un lenguaje abierto y universal para la "comunicación" entre elementos de red, a partir de la creación de tablas de flujo dinámicas. Actualmente, muchos equipos son compatibles con el protocolo OpenFlow, popularizando y viabilizando la expansión de las redes definidas por software [16].

El protocolo OpenFlow consta de tres conceptos básicos:

- La red está formada por conmutadores compatibles con OpenFlow que componen el plano de datos.
- El plano de control consta de uno o más controladores OpenFlow.
- Un canal de control seguro conecta los conmutadores con el plano de control.

Tipos de mensajes OpenFlow

El protocolo OpenFlow admite tres tipos de mensajes: controlador a conmutador, asíncrono y simétrico, cada uno con varios subtipos [17].

- Los mensajes de controlador a conmutador son iniciados por el controlador y se utilizan para administrar o inspeccionar directamente el estado del conmutador.
- Los mensajes asíncronos son iniciados por el conmutador y se utilizan para actualizar el controlador de los eventos de la red y los cambios en el estado del conmutador.
- Los mensajes simétricos son iniciados por el conmutador o el controlador y se envían sin solicitud.

Versiones del protocolo OpenFLow

La Tabla 2, proporciona los protocolos compatibles y los campos de coincidencia disponibles para cada una de las versiones de OpenFlow.

Tabla 2. Campos de coincidencia de las versiones OpenFlow (OF) *[15]*.

Campos de coincidencia	OF 1.0	OF 1.1	OF 1.2	OF 1.3	OF 1.4
Ingress Port		✓	✓	✓	✓
Metadata		✓	✓	✓	✓
Ethernet: src, dst, type	✓	✓	✓	✓	✓
IPv4: src, dst, proto, ToS	✓	✓	✓	✓	✓
TCP/UDP: src port, dst port	✓	✓	✓	✓	✓
MPLS: label, traffic class		✓	✓	✓	✓
OpenFlow Extensible Match (OXM)			✓	✓	✓
IPv6: src, dst, flow label, ICMPv6			✓	✓	✓
IPv6 Extension Headers				✓	✓

La Tabla 3, compila los tipos de estadísticas recopiladas por un switch, que pueden ser consultadas por un controlador.

Tabla 3. Las estadísticas se miden para diferentes partes del interruptor OpenFlow *[15]*.

Tipo de estadísticas	OF 1.0	OF 1.1	OF 1.2	OF 1.3	OF 1.4
Estadísticas por tabla	✓	✓	✓	✓	✓
Estadísticas por flujo		✓	✓	✓	✓
Estadísticas por puerto	✓	✓	✓	✓	✓
Estadísticas por cola	✓	✓	✓	✓	✓
Estadísticas de grupo	✓	✓	✓	✓	✓
Estadísticas del cubo de acción		✓	✓	✓	✓
Medidor de flujo				✓	✓
Banda del medidor por flujo				✓	✓

1.1.1.4 Open vSwitch

Definición

Open vSwitch es un conmutador basado en software multicapa que trabaja bajo la licencia Apache 2 de código abierto puede usarse como un conmutador virtual puro en entornos virtualizados y como un conmutador de software de propósito general que conecta nodos físicamente separados. Es compatible con OpenFlow y proporciona funciones avanzadas para la virtualización de la red. La instalación es posible en la mayoría de las distribuciones de Linux, incluido OpenWRT [19].

Componentes

Open vSwitch a diferencia de otros switch comunes se encuentra programado en dos segmentos diferentes del sistema operativo como se muestra en la Figura 3 [20].

El primero se desarrolla en el espacio de usuario, donde se hace la gestión del switch a través de:

- *ovs-vswitchd:* demonio que implementa las funcionalidades del conmutador, junto con un módulo de kernel para conmutación basada en flujos.
- *ovsdb-server:* servidor de base de datos liviano que ovs-vswitchd consulta para obtener su Configuración.

El segundo se implementa en espacio de Kernel, donde verdaderamente se realiza la conmutación de los paquetes, en este se encuentra el módulo data path [20].

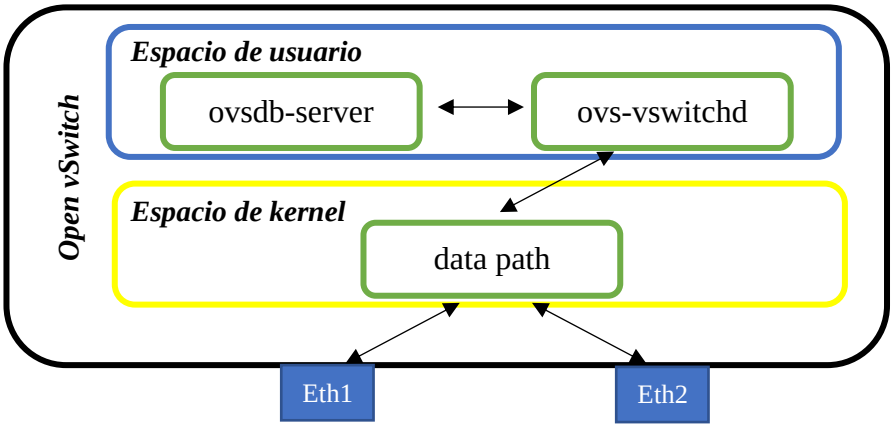


Figura 3. Funcionamiento interno OvS

Elaborado por el autor.

Funcionamiento

La Figura 4 ilustra las diferentes rutas de procesamiento en un OvS. Los dos componentes más importantes son el daemon ovs-vswitchd que controla el módulo del kernel e implementa el protocolo OpenFlow, y el data path, un módulo del kernel que implementa el reenvío de paquetes actual.

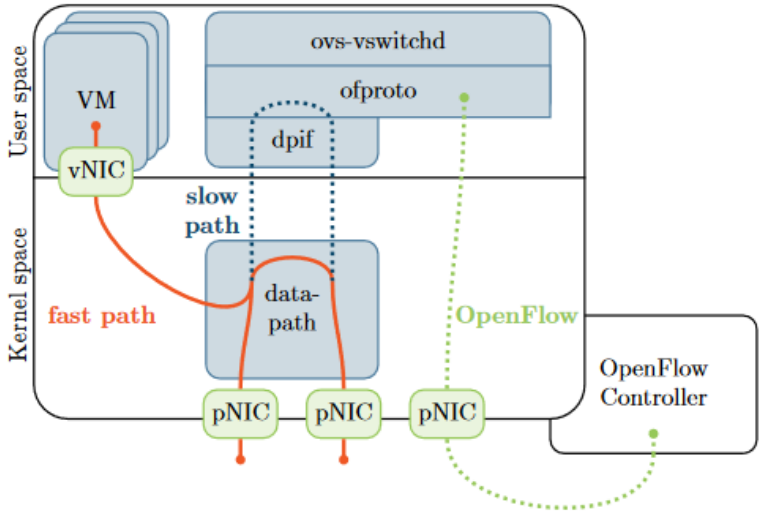


Figura 4. Arquitectura Open vSwitch que representa los flujos de procesamiento de datos *[19]*.

El módulo data path del kernel procesa los paquetes utilizando un sistema basado en reglas: mantiene una tabla de flujo en la memoria, que asocia los flujos con las acciones.

Un ejemplo de dicha regla es reenviar todos los paquetes con una determinada dirección MAC de destino a un puerto físico o virtual específico. Las reglas también pueden filtrar paquetes soltándolos según el destino específico o las direcciones IP de origen. El conjunto de reglas admitido por OvS en su módulo de kernel es más simple que las reglas definidas por OpenFlow. Estas reglas simplificadas se pueden ejecutar más rápido que las reglas posiblemente más complejas de OpenFlow. Por lo tanto, la opción de diseño para no admitir explícitamente todas las características OpenFlow ofrece resultados en un mayor rendimiento para el módulo del núcleo de OvS [21].

Un paquete que puede ser procesado por una regla del data path toma la ruta rápida y se procesa directamente en el módulo del núcleo sin invocar ninguna otra parte de OvS. La Figura 4 resalta este camino rápido con una línea naranja continua. Los paquetes que no coinciden con un flujo en la tabla de flujo se fuerzan en la ruta lenta

(línea azul punteada), que copia el paquete al espacio del usuario y lo reenvía al demonio OvS. Esto es similar a la acción encapsulada en OpenFlow, que reenvía un paquete que no puede procesarse directamente en un conmutador a un controlador OpenFlow. La ruta lenta es implementada por el demonio ovs-vswitchd, que opera con reglas OpenFlow. Los paquetes que toman esta ruta se comparan con las reglas de OpenFlow, que pueden agregarse mediante un controlador externo de OpenFlow o mediante una interfaz de línea de comandos. El daemon deriva las reglas del data path para paquetes basados en las reglas de OpenFlow y las instala en el módulo del núcleo para que los paquetes futuros de este flujo puedan tomar la ruta rápida. Todas las reglas en el data path están asociadas con un tiempo de espera de inactividad. Por lo tanto, la tabla de flujo del data path solo contiene las reglas requeridas para manejar los flujos actualmente activos, por lo que actúa como caché para la tabla de flujo de OpenFlow más grande y complicada en la ruta lenta [21].

Versiones de OpenFlow que admite Open vSwitch

En la tabla 4 se enumeran las versiones de OpenFlow compatibles con cada versión de Open vSwitch:

Tabla 4. Versión OpenFlow compatible para diferentes versiones de Open vSwitch *[17]*.

	OF1.0	OF1.1	OF1.2	OF1.3	OF1.4	OF1.5
1.9 y anterior	✓	×	×	×	×	×
1.10, 1.11	✓	×	☑	☑	×	×
2.0, 2.1	✓	☑	☑	☑	×	×
2.2 2.2	✓	☑	☑	☑	☒	☑
2.3, 2.4	✓	✓	✓	✓	☑	☑
2.5, 2.6, 2.7	✓	✓	✓	✓	☑	☑
2.8, 2.9, 2.10, 2.11	✓	✓	✓	✓	✓	☑
2.12	✓	✓	✓	✓	✓	✓

× No soportado.
✓ Compatible y habilitado de forma predeterminada
☑ Compatible, pero faltan funciones y el usuario debe habilitarlo.
☒ Implementación experimental insegura.

Herramientas de Open vSwitch

- *ovs-dpctl*: una herramienta para Configurar el módulo del kernel del conmutador.
- *ovs-vsctl*: una utilidad para consultar y actualizar la Configuración de ovs-vswitchd.
- *ovs-appctl:* una utilidad que envía comandos para ejecutar demonios Open vSwitch.
- *ovs-ofctl:* una utilidad para consultar y controlar los switchs y controladores OpenFlow.
- *ovs-pki:* una utilidad para crear y administrar la infraestructura de clave pública para los switchs OpenFlow.
- *ovs-testcontroller:* un simple controlador OpenFlow que puede ser útil para pruebas (aunque no para producción) [17].

1.1.1.5 Controlador SDN

Un controlador SDN es el repositorio central de instrucciones de: control, lógica de flujo de datos, políticas de seguridad y políticas comerciales necesarias para implementar, Configurar y administrar los elementos de la red obteniendo un comportamiento deseado. El controlador proporciona una interfaz programática para el aprovisionamiento de servicios de red utilizando un enfoque consistente. En efecto, el controlador SDN sirve como una especie de sistema operativo (SO) para la red.

El controlador SDN utiliza un protocolo abierto o propietario para controlar cada elemento de la red y el flujo de tráfico en la red. La SDN depende en gran medida de los mensajes de control entre un controlador y los dispositivos de reenvío para una operación confiable de la red. Además, un controlador SDN permite extraer información del estado de los elementos en la red y comunica esa información a la aplicación con una visión abstracta, que incluye estadísticas y eventos sobre lo que está sucediendo [22].

Tipos de controladores

Existen múltiples controladores en el mercado actual, en la Figura 5 se pueden apreciar algunos de los controladores más populares y su clasificación.

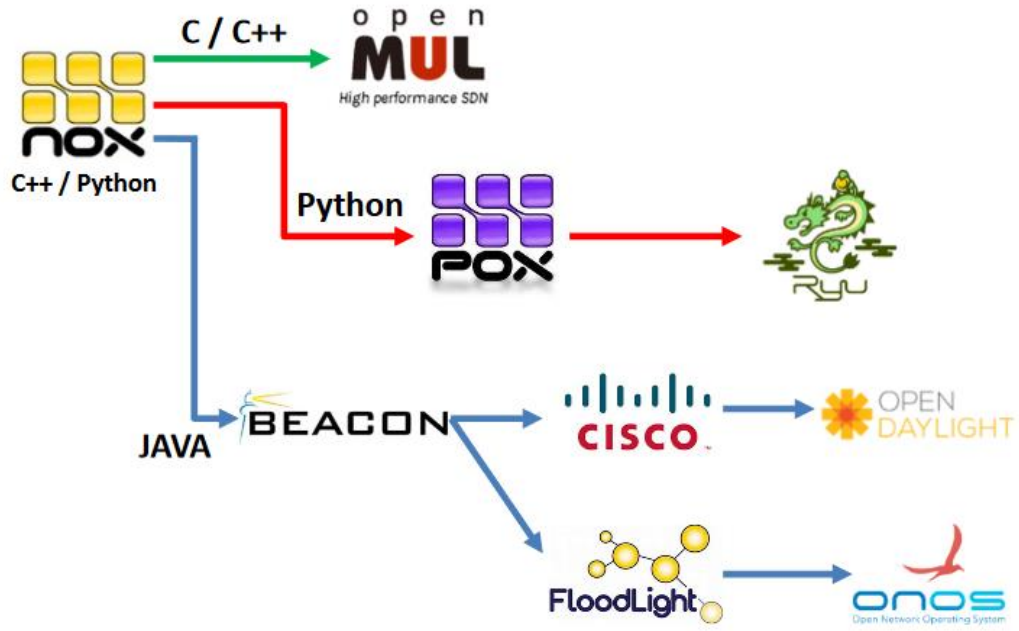


Figura 5. Clasificación de los Controladores SDN *[23]*.

En la tabla 5 se muestran las características principales de los controladores con mayor acogida en el mercado actual.

Tabla 5. Características de los controladores más utilizados *[20]*, *[24]*.

	NOX	*POX*	*ONOS*	*ODL*	*RYU*
SOPORTE OPENFLOW	v1.0	v1.0	v1.0, v1.2, v1.3	v1.0, v1.2, v1.3	v1.0 - v1.5
LENGUAJE DE DESARROLLO	C++	Python	Java	Java	Python
PROVEE REST API	No	No	Si	Si	Si
SOPORTE OPENSTACK	No	No	Si	Si	Si
SIMPLICIDAD	Regular	Regular	Fácil	Fácil	Fácil
SOPORTE DE PLATAFORMAS	Linux	Linux, Mac OS, Windows	Linux, Mac OS, Windows	Linux,	Linux
DOCUMENTACIÓN	Media	Pobre	Buena	Media	Buena

1.1.1.6 Aplicaciones SDN

Las funciones y los servicios de red se pueden agregar dinámicamente en forma de aplicaciones de red, lo que facilita su implementación. La comunidad de investigación SDN ya ha propuesto y evaluado múltiples aplicaciones de red. Se pueden agrupar en las áreas de administración de redes e ingeniería de tráfico, balanceo de carga del servidor de aplicaciones y control de acceso a la red, seguridad SDN, virtualización de red y enrutamiento entre dominios [15].

1.1.1.7 Controlador RYU

Definición

Ryu es un Framework de SDN que proporciona componentes SDN a través de una API que facilita a los desarrolladores la creación de nuevas aplicaciones de control y administración de redes. Ryu admite varios protocolos para administrar dispositivos de red, como OpenFlow, Netconf, OF-config, etc. Todo el código está disponible libremente bajo la licencia Apache 2.0. Ryu significa "flujo" en japonés. La ubicación del Framework se ilustra en la Figura 6 [25].

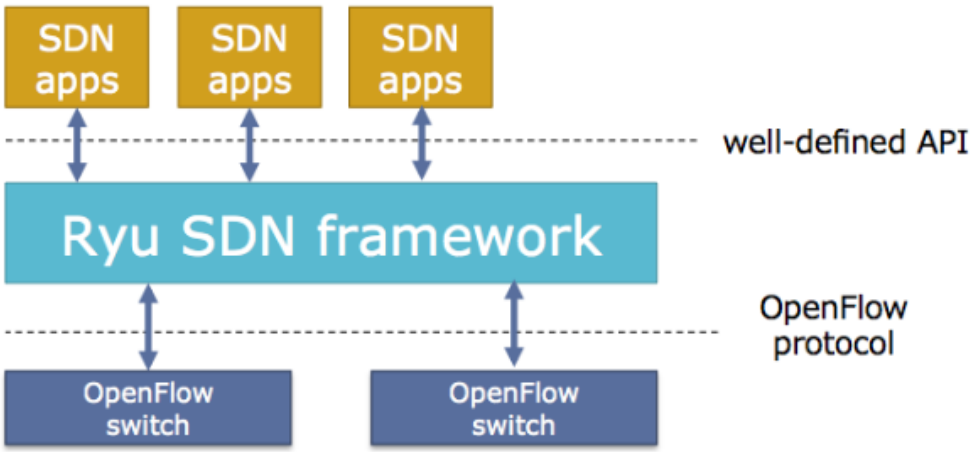


Figura 6. Ubicación del Framework de Ryu *[26]*.

Este controlador está desarrollado en el lenguaje de programación Python, por lo que posee las ventajas de dicho lenguaje. Python es un lenguaje de programación interpretado, por lo que no requiere ser previamente compilado. Su sintaxis y semántica es relativamente sencilla a comparación de lenguajes similares y es totalmente compatible con el paradigma de la programación orientada a objetos [27].

Arquitectura

Al igual que cualquier controlador SDN, Ryu también puede crear y enviar un mensaje OpenFlow, escuchar eventos asincrónicos como flow_removed así como analizar y manejar los paquetes entrantes. La Figura 7 muestra la arquitectura del Framework del controlador Ryu:

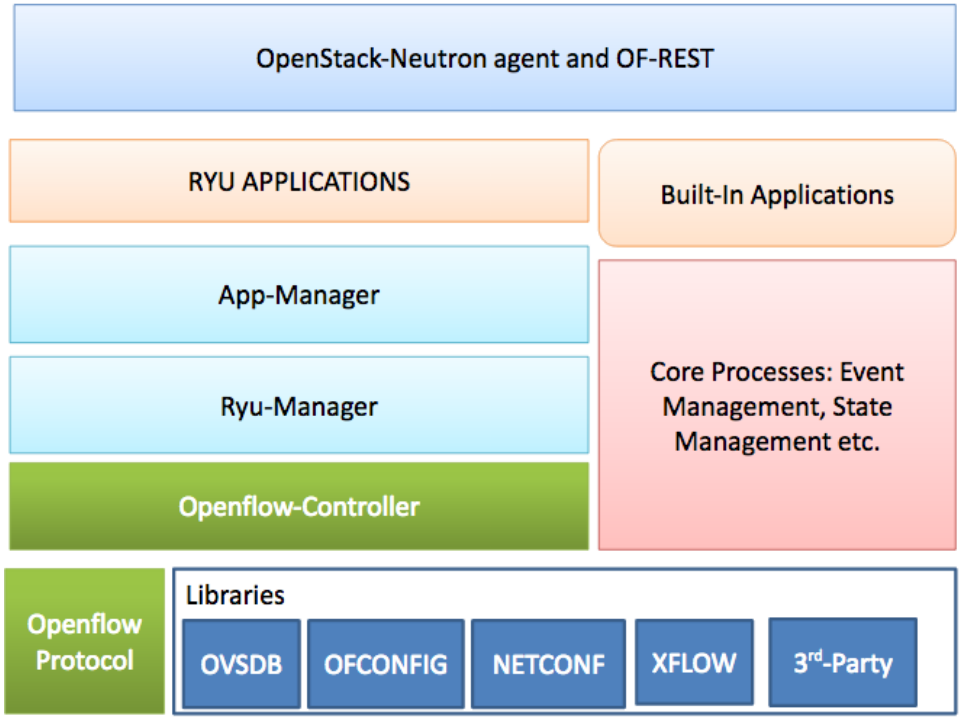


Figura 7. Arquitectura RYU *[28]*.

Librerías Ryu

Ryu tiene una impresionante colección de librerías, que van desde la compatibilidad con múltiples protocolos SouthBound hasta varias operaciones de procesamiento de paquetes de red. Con respecto a los protocolos SouthBound, Ryu admite OF-Config, Open vSwitch Database Management Protocol (OVSDB), NETCONF, XFlow (Netflow y Sflow) y otros protocolos de terceros. Netflow es compatible con Cisco y otros, y es específico para una IP. Los protocolos Netflow y Sflow admiten muestreo y agregación de paquetes, que se utilizan principalmente para la medición del tráfico de red. La librería de paquetes Ryu ayuda a analizar y construir varios paquetes de protocolos, como VLAN, MPLS, GRE, etc [28].

Controlador y protocolo OpenFlow

Ryu admite el protocolo OpenFlow hasta la última versión 1.5. Incluye un codificador de protocolo OpenFlow y una biblioteca de decodificadores. Además, uno de los componentes clave de la arquitectura Ryu es el controlador OpenFlow, que es responsable de administrar los conmutadores OpenFlow utilizados para Configurar flujos, administrar eventos, etc. El controlador OpenFlow es una de las fuentes internas de eventos en la arquitectura Ryu. La tabla 6 resume los mensajes del protocolo OpenFlow, las estructuras y la API correspondiente [28].

Tabla 6. Mensajes del protocolo OpenFlow, estructuras y API correspondiente *[28]*.

Mensajes Controlador-switch	**Mensajes Asíncronos**	**Mensajes Simétricos**	**Estructuras**
▪ Feactures ▪ Configura tion, ▪ Modify/read/state ▪ Send-Packet ▪ Packet-out, ▪ Barrier, ▪ Role-request	▪ Packet-in ▪ Flow-removed ▪ Port-status ▪ Error	▪ Hello ▪ Echo-Request & Reply ▪ Error	▪ Flow-match
▪ send_msg API ▪ packet builder APIs	▪ set_ev_cls API ▪ packet parser APIs	▪ Both Send ▪ Event APIs	

Administradores y procesos del núcleo

El administrador de Ryu es el ejecutable principal. Cuando se ejecuta, escucha la dirección IP especificada (por ejemplo: 0.0.0.0) y el puerto especificado (6633 de forma predeterminada). Después de esto, cualquier conmutador OpenFlow (hardware u OVS) puede conectarse al administrador de Ryu. El administrador de aplicaciones es el componente fundamental para todas las aplicaciones de Ryu. Todas las aplicaciones heredan de la clase RyuApp del administrador de aplicaciones. El componente del proceso central en la arquitectura incluye gestión de eventos, mensajería, gestión de estado en memoria, etc. Curiosamente, el servicio de mensajería Ryu admite componentes desarrollados en otros idiomas [28].

NorthBound Ryu

En la capa API, Ryu incluye un complemento Openstack Neutron que admite configura ciones de VLAN y superposición basadas en GRE. Ryu también admite una interfaz REST para sus operaciones OpenFlow. Además, usando WSGI (un marco para conectar aplicaciones web y servidores web en Python), uno puede introducir fácilmente peticiones API REST en una aplicación [28].

Aplicaciones de Ryu

Ryu se distribuye con múltiples aplicaciones, como un simple_switch, router, firewall, GRE tunnel, topology, VLAN, etc. Las aplicaciones Ryu son entidades de un solo subproceso, que implementan diversas funcionalidades. Las aplicaciones de Ryu se envían eventos asincrónicos entre sí. La arquitectura funcional de una aplicación Ryu se muestra en la Figura 8.

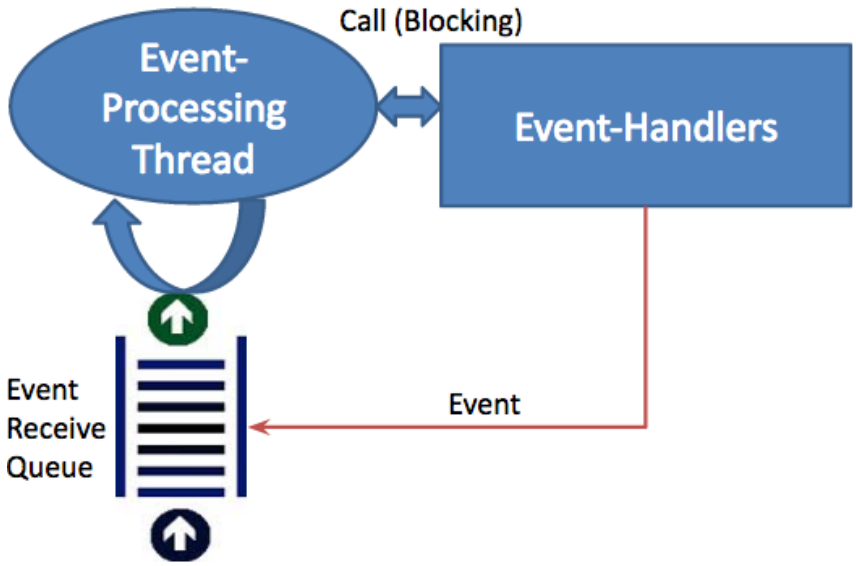


Figura 8. Arquitectura funcional de la aplicación Ryu *[28]*.

Cada aplicación Ryu tiene una cola FIFO de recepción de eventos para preservar el orden de estos. Además, cada aplicación incluye un subproceso para procesar eventos desde la cola. El bucle principal del subproceso extrae eventos de la cola de recepción y llama al controlador de eventos apropiado. Por lo tanto, el controlador de eventos se llama dentro del contexto del subproceso de procesamiento de eventos, que funciona de manera bloqueante, es decir, cuando un controlador de eventos recibe el control, no se procesarán más eventos para la aplicación Ryu hasta que se devuelva el control [28].

1.1.1.8 Internet de las cosas

Introducción

IoT es la evolución de Internet que presenta enormes desafíos en la recopilación, análisis y distribución de datos hacia un uso más productivo de la información para mejorar la calidad de vida [29]. El concepto de IoT implica la gestión de sensores o dispositivos distribuidos en la red, a fin de reconocer y notificar a los usuarios instantáneamente sobre eventos en tiempo real. Estos dispositivos, que tienen habilidades computacionales básicas, son llamados objetos inteligentes. Los objetos inteligentes se caracterizan por un identificador único, es decir, una etiqueta de nombre para la descripción del dispositivo y una dirección para la comunicación. Según [30] hay tres tipos de objetos inteligentes:

- *Actividad:* objetos conscientes que pueden recopilar datos relacionados con las actividades laborales, así como su propio uso.
- *Política:* objetos conscientes que pueden traducir actividades y eventos con respecto a políticas organizacionales específicas
- Proceso: objetos conscientes, donde un proceso es un conjunto de tareas y actividades relevantes que se ordenan según su posición en el espacio y el tiempo.

Los dispositivos IoT se caracterizan principalmente por sus recursos limitados en términos de potencia, procesamiento, memoria y ancho de banda. Debido a este hecho, los protocolos tradicionales relativos a las operaciones de red y la seguridad no se pueden implementar en un entorno específico de IoT. Sin embargo, es el hecho de que, al proporcionar seguridad integrada a los dispositivos por diseño, se ofrecen muchos beneficios, relacionados con la reducción de costos en la arquitectura de seguridad, el aumento de la confiabilidad y la mejora del rendimiento general [31].

Definición del IoT

Según la UIT el internet de las cosas (IoT) se define como "una infraestructura de red global dinámica con capacidades de autoConfiguración basadas en protocolos de comunicación estándar e interoperables donde las cosas físicas y virtuales tienen

identidades, atributos físicos y personalidades virtuales, y usan interfaces inteligentes de manera integrada en la red de información" [32].

Arquitectura IoT y sus componentes

No existe un único consenso sobre la arquitectura para el IoT que se acuerde universalmente. Diferentes arquitecturas han sido propuestas por diferentes investigadores. Sin embargo, para el desarrollo del presente trabajo se toma como base la arquitectura general de IoT [33] mostrada en la Figura 9.

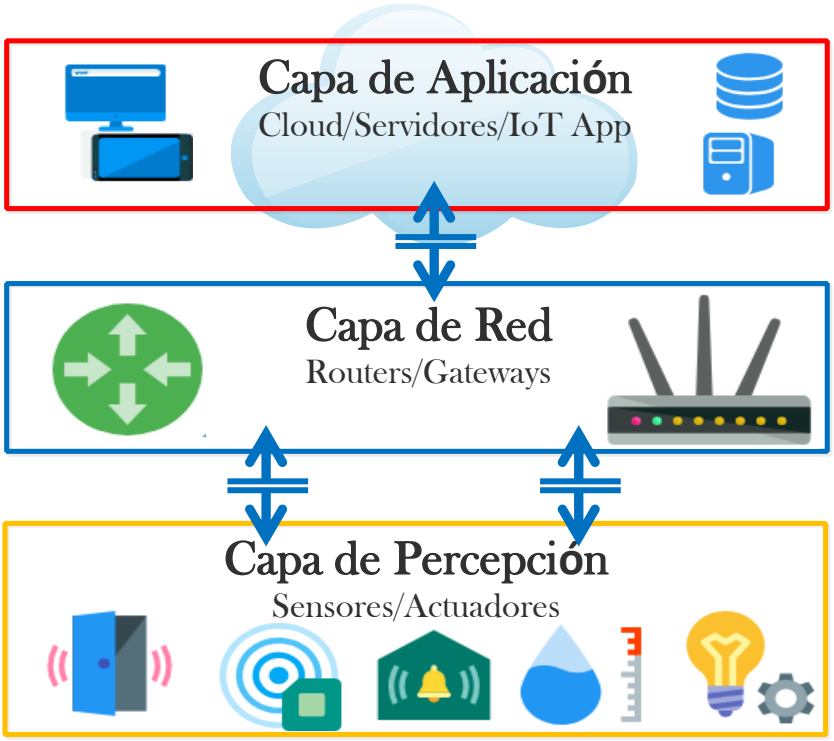


Figura 9. Arquitectura general del IoT.
Elaborado por el autor.

- *Capa de percepción:* es la capa física, que consta de sensores, actuadores, RFID, dispositivos móviles, entre otros, que permiten la comunicación con la puerta de enlace IoT. Detectan y recopilan información sobre el entorno, e identifican diferentes tipos de dispositivos de manera inteligente.
- *Capa de red:* la capa de red recopila los datos de la capa inferior y los enruta a través de la puerta de enlace a Internet para su posterior procesamiento. Diferentes tecnologías de transmisión contribuyen a la heterogeneidad de IoT, como ZigBee,

bluetooth, Wi-Fi, etc. La capa de red proporciona conectividad y también debe ocuparse de la administración en términos de seguridad, procesamiento de información, conversión de protocolo y manejo de fallas.

- *Capa de aplicación:* Define varias aplicaciones en las que se puede implementar el Internet de las cosas, por ejemplo, hogares inteligentes, ciudades inteligentes y salud inteligente. También incluye infraestructuras de servidor y nube que comparten contenido y proporcionan servicios en tiempo real. El procesamiento de datos y la prestación de servicios son una de las funciones más importantes de esta capa. La capa proporciona una gestión global del sistema IoT. Recibe información de la capa de red que permite a los desarrolladores crear varias aplicaciones utilizando una abstracción e interfaces abiertas de alto nivel.

Principales protocolos del IoT

IoT se ejecuta sobre el modelo de red TCP/IP que utiliza un modelo de 4 capas con protocolos definidos en cada nivel como se aprecia en la Figura 10. Debido a la amplia gama de tecnologías que forman parte del IoT, hay una gran cantidad de protocolos. Algunos son más conocidos que otros. A continuación, se hace una descripción general de los principales protocolos del IoT clasificados por capas [34].

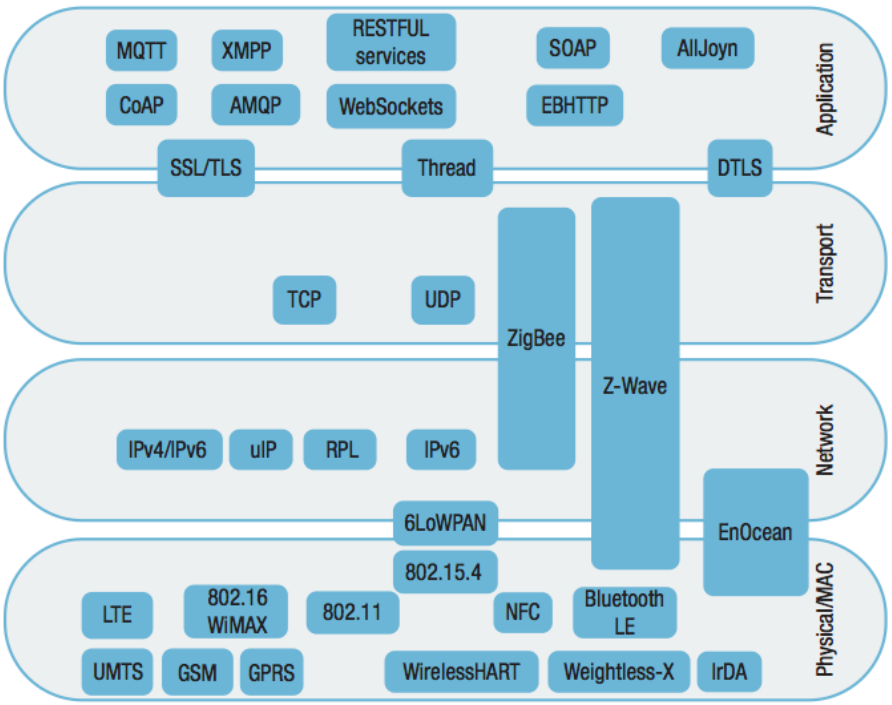


Figura 10. Protocolos IoT *[34]*.

- **Capa de aplicación**

MQTT (Message Queuing Telemetry Transport): está diseñado como un protocolo de mensajería ligero que utiliza operaciones de publicación/suscripción para intercambiar datos entre los clientes y el servidor. Además, su pequeño tamaño, bajo consumo de energía, paquetes de datos minimizados y facilidad de implementación hacen que el protocolo sea ideal para el mundo de "máquina a máquina" o "Internet de las cosas" [35].

- **Capa de transporte**

TCP: es un protocolo orientado a la conexión de la capa de transporte del modelo OSI para aplicaciones de red cliente / servidor basado en el Protocolo de Internet (IP). Por lo tanto, se prefiere UDP porque es un protocolo sin conexión y tiene una sobrecarga baja [35].

- **Capa de red**

IPv4/IPv6: es un protocolo de capa de Internet para la interconexión de redes con conmutación de paquetes y proporciona transmisión de datagramas de extremo a extremo a través de múltiples redes IP [35].

- **Capa física**

Wifi: Para la integración de IoT, WiFi es una opción preferida según muchos diseñadores electrónicos. Es por la infraestructura que soporta. Cuenta con velocidades de transferencia de datos rápidas y con la capacidad de controlar una gran cantidad de datos. El estándar de WiFi 802.11 extendido le ofrece la capacidad de transferir cientos de megabits en solo un segundo. El único inconveniente de este protocolo de IoT es que puede consumir energía excesiva para algunas aplicaciones de IoT [34].

Arquitectura IoT basada en SDN

Esta sección presenta la arquitectura de un sistema IoT con la adopción de una red definida por software.

La capa de red como se muestra en la Figura 11 se divide en dos capas: la capa de datos y la capa de control, mientras que las otras capas permanecen sin cambios.

- La capa de datos que consta de conmutadores y enrutadores SDN no toma decisiones sobre la información que se recibe o transmite hacia la capa de percepción. En su lugar, dejan la toma de decisiones a la capa de control permitiendo el control programable de los switches y routers a través de una Southbound API con OpenFlow u otros protocolos similares que tienen diferentes funciones y capacidades para gestionar diferentes aspectos del funcionamiento de la red.
- La capa de control está lógicamente centralizada por el controlador SDN que opera con una visión de red global aprovechando la información recibida a través de la Southbound API y que puede proporcionar una solución innovadora en términos de desarrollo del sistema e implementación [36].

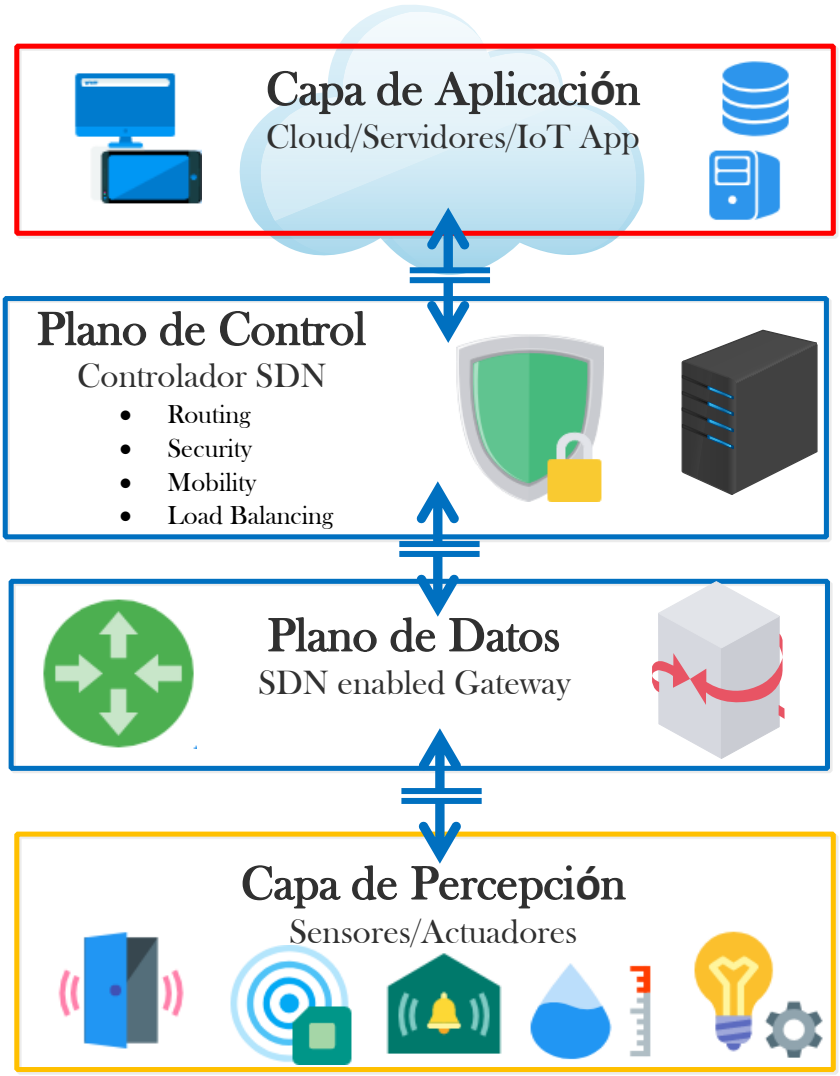


Figura 11. Arquitectura IoT-SDN.

Elaborado por el autor.

Uno de los objetivos clave de la arquitectura propuesta es la sustitución de gateways tradicionales por gateways SDN. Dado que los dispositivos de detección están conectados al dominio de red por una puerta de enlace a través de tecnologías de red local que son inalámbricas como WiFi, Bluetooth, ZigBee, etc. o cableados, la puerta de enlace de IoT debe ser lo suficientemente flexible para administrar una gran cantidad de datos, QoS seguridad y varios recursos disponibles, etc [36]. Debido a la heterogeneidad de los dispositivos de IoT, la puerta de enlace habilitada para SDN puede acomodar una gran cantidad de tráfico procedente de varios objetos inteligentes heterogéneos y también emplear protocolos de enrutamiento inteligentes y técnicas de almacenamiento en caché para enrutar el tráfico a través de rutas poco restringidas.

Aplicaciones del IoT

Debido al uso de objetos inteligentes, se considera que IoT tiene un gran impacto en una amplia variedad de aplicaciones, como WSN y comunicaciones de banda estrecha. La Tabla 7 presenta los campos de aplicación del IoT más importantes y sus funciones relacionadas. IoT puede encontrar su aplicación en casi todos los aspectos de nuestra vida cotidiana. Una de las aplicaciones más convincentes de IoT existe en la conceptualización de ciudades inteligentes, hogares inteligentes y seguridad de objetos inteligentes.

Tabla 7. Campos de aplicación del IoT y sus funciones *[37]*.

Campo de aplicación	Funciones
Movilidad inteligente y turismo inteligente.	▪ Gestión del tráfico, transporte multimodal. ▪ Monitorización del estado de las carreteras, sistema de aparcamiento, recogida de residuos ▪ Sistemas de pago, servicios de guía turístico.
Casa inteligente	▪ Mantenimiento de plantas, gestión energética. ▪ Videovigilancia, gestión de accesos, protección infantil. ▪ Entretenimiento, vida cómoda.
Seguridad pública y vigilancia ambiental.	▪ Seguimiento ambiental y territorial. ▪ Video / radar / vigilancia satelital

	▪ Localización de emergencia / rescate personal de seguimiento, plan de emergencia.
Procesamiento industrial	▪ Monitorio de disponibilidad del producto en tiempo real ▪ Gestión de maletas, operación de embarque, tickets móviles. ▪ Monitoreo de plantas industriales.
Agricultura y cría	▪ Seguimiento animal, certificación y control comercial. ▪ Riego, control de producción agrícola y piensos. ▪ Gestión de registro de fincas
Logística y gestión de la vida útil del producto.	▪ Identificación de materiales / deterioro del producto. ▪ Gestión de agua, venta al por menor, inventario. ▪ Operación de compra, pago rápido
Medicina y salud	▪ Monitorización remota de parámetros médicos, diagnósticos. ▪ Seguimiento de equipos médicos, gestión segura de ambientes interiores. ▪ Servicios hospitalarios inteligentes, servicios de entretenimiento.
Vida independiente	▪ Asistencia a personas mayores, asistencia a discapacitados. ▪ Asistencia personal a domicilio / móvil, inclusión social. ▪ Bienestar individual, impacto del comportamiento personal en la sociedad.

1.1.1.9 Tecnologías soportadas por el IoT

La Tabla 8 muestra algunas de las plataformas de hardware existentes para el IoT clasificadas según parámetros clave tales como: Procesador, Voltaje operativo, Velocidad del reloj, Ancho del bus, Memoria del sistema, Memoria flash, EEPROM, Medios de comunicación y Conectividad de E/S. El estudio comparativo muestra cómo estas plataformas están fomentando el crecimiento de IoT utilizando un comportamiento de restricción.

Tabla 8. Comparación de las plataformas de hardware compatibles con IoT existentes *[38]*.

Parámetros	**Arduino Uno**	**Arduino Yun**	**Intel Galileo Gen 2**	**Intel Edison**	**Beagle Bone Black**	**Imp. Eléctrico 003**	**Raspberry Pi B +**	**ARM mbed NXP LPC1768**
Procesador	ATMega328P	ATmega32u4, y Atheros AR9331	Intel® Quark ™ SoC X1000	Intel® Quark ™ SoC X1000	Sitara AM3358BZCZ100	BRAZO Cortex M4F	Broadcom BCM2835 SoC basado en ARM11 76JZF	ARM Cortex M3
Tensión de funcionamiento	5V	5V, 3V	5V	3.3V	3.3V	3.3V	5V	5V
Velocidad de reloj (MHz)	16	16,400	400	100	1GHz	320	700	96
Ancho de bus (bits)	8	8	32	32	32	32	32	32
Memoria del sistema	2kB	2.5kB, 64MB	256MB	1GB	512MB	120KB	512MB	32KB
Memoria flash	32kB	32kB, 16MB	8MB	4GB	4GB	4Mb	-	512KB
EEPROM	1 kB	1kB	8kB	-	-	-	-	-
Comunicación	IEEE 802.11 b / g / n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serie	IEEE 802.11 b / g / n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serie	IEEE 802.11 b / g / n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serie	IEEE 802.11 b / g / n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serie	IEEE 802.11 b / g / n, 433RF, IEEE 802.15.4, BLE 4.0, Ethernet, serie	IEEE 802.11 b / g / n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serie	IEEE 802.11 b / g / n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serie	IEEE 802.11 b / g / n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serie
Conectividad de E / S	SPI, I2C, UART, GPIO	SPI, I2C, UART, GPIO	SPI, I2C, UART, GPIO	SPI, I2C, UART, I2S, GPIO	SPI, UART, I2C, McASP, GPIO	SPI, I2C, UART, GPIO	SPI, DSI, UART, SDIO, CSI, GPIO	SPI, I2C, CAN, GPIO

Tecnologías de comunicación inalámbrica

Los protocolos de comunicación forman la columna vertebral de los sistemas de IoT y permiten la conectividad de red y el acoplamiento a las aplicaciones. Los protocolos de comunicación permiten que los dispositivos intercambien datos a través de la red. Los protocolos definen los formatos de intercambio de datos, la codificación de datos, los esquemas de direccionamiento de dispositivos y el enrutamiento de paquetes desde el origen hasta el destino. Otras funciones de los protocolos incluyen control de secuencia, control de flujo y retransmisión de paquetes perdidos. La Tabla 9 compara diferentes tecnologías de comunicación inalámbrica con respecto a varios parámetros.

Tabla 9. Comparación de las tecnologías de comunicación existentes [38].

Parámetros	**Wifi**	**WiMAX**	**LR-WPAN**	**Comunicación móvil**	**Bluetooth**	**LoRa**
Estándar	IEEE 802.11 a / c / b / d / g / n	IEEE 802.16	IEEE 802.15.4 (ZigBee)	2G-GSM, CDMA 3G-UMTS, CDMA2000 4G-LTE	IEEE 802.15.1	LoRa WAN R1.0
Banda de frecuencia	5–60 GHz	2–66GHz	868/915MHz, 2.4 GHz	865MHz, 2.4 GHz	2.4GHz	868/900 MHz
Velocidad de datos	1Mb / s – 6,75 Gb / s	1Mb / s – 1 Gb / s (fijo) 50–100Mb / s (móvil)	40–250 Kb / s	2G: 50–100kb / s 3G: 200kb / s 4G: 0.1–1Gb / s	1–24Mb / s	0.3–50 Kb/s
Rango de transmisión	20–100m	<50Km	10-20m	Área celular completa	8-10m	<30 km
Consumo de energía	Alto	Medio	Bajo	Medio	Bluetooth: medio BLE: muy bajo	Muy bajo
Costo	Alto	Alto	Bajo	Medio	Bajo	Alto

1.1.1.10 Influencia de las SDN en aplicaciones del IoT

Realizada la investigación sobre las redes definidas por software y algunas de las aplicaciones del IoT, a continuación, se presenta la influencia de las SDN sobre aplicaciones del IoT, enfocadas a resolver problemas específicos a nivel de red:

1. Interoperabilidad en Internet de las cosas

La interoperabilidad de los dispositivos IoT se puede resolver utilizando SDN superando la heterogeneidad de los dispositivos. El desafío de interoperabilidad en IoT surge cuando se tienen dispositivos heterogéneos que intercambian formatos de datos y diversos protocolos para el intercambio de datos de máquina a máquina (M2M), y también con la interconectividad de una gran cantidad de dispositivos diferentes, hay falta de cooperación y falta de coincidencia de capacidad entre dispositivos que pueden obstaculizar el rendimiento de la red. Sin embargo, el enfoque SDN brinda flexibilidad que puede usarse para permitir que diferentes objetos conectados a redes heterogéneas se comuniquen entre sí. Esto podrá manejar conexiones simultáneas de varias tecnologías de comunicación. Las decisiones de gestión de la red, como el enrutamiento, la programación se pueden hacer en el controlador SDN y, además, la programabilidad permite cualquier actualización para nuevas propuestas o incluso enfoques de estado limpio.

2. Descubrimiento

Si se garantiza la interoperabilidad entre los dispositivos IoT, la capacidad de detección es otro problema. La capacidad de detección en dispositivos IoT es uno de los principales factores para lograr una implementación exitosa de aplicaciones IoT para evitar interrupciones prolongadas y errores de Configuración. La capacidad de autoConfigurarse y adaptarse al entorno sin intervención humana conlleva requisitos tales como el descubrimiento de recursos y servicios. No es factible Configurar manualmente todos y cada uno de los dispositivos para descubrir qué objetos están cerca y qué funciones proporcionan. El enfoque SDN se puede utilizar para abordar este problema que permite que las aplicaciones funcionen con dispositivos con una Configuración mínima o nula.

3. Seguridad

Con una gran cantidad de dispositivos heterogéneos que están involucrados en el Internet de las cosas, existe una alta propensión a ser vulnerable a los ataques, y para garantizar la protección de los datos, la autenticación de datos debe tomarse en serio. SDN se puede utilizar para resolver problemas de madurez de seguridad en aplicaciones IoT. Puede satisfacer las necesidades de asegurar los datos generados

mediante la virtualización de componentes y servicios de red. Con esto, se puede aplicar la autorización y la autenticación que pueden garantizar la entrega segura de datos. Las amenazas de seguridad pueden ser más fáciles de atacar a través de la visibilidad mejorada que SDN proporciona a la red. SDN también puede proporcionar un modelo de seguridad dinámico, inteligente y autoaprendizaje en capas que proporciona reglas de acceso para garantizar la autorización de las personas que pueden cambiar la Configuración de los dispositivos. SDN servirá como una herramienta crítica para garantizar una red segura en un entorno de IoT, gracias a su adaptabilidad, que le permitirá abordar nuevas amenazas a medida que emergen y evolucionan.

4. Gestión

El controlador SDN gestiona y supervisa todo el trabajo de la red. La posición centralizada del controlador SDN lo hace adecuado para tener una visión global de la topología y las condiciones de la red, realizando el control de la red, como el enrutamiento y el control de QoS. El controlador puede determinar las mejores decisiones de enrutamiento e insertar estas decisiones en las tablas de flujo. Los nodos sensores no toman decisiones de enrutamiento, sino que solo envían y descartan paquetes de acuerdo con las reglas establecidas por el controlador. La programación debe construirse sobre rutas definidas y el controlador puede optimizar los ciclos de reposo/activo de los sensores eligiendo el conjunto con mayor eficiencia energética en cada ciclo de programación. La latencia se puede reducir y se pueden lograr importantes ahorros de energía.

5. Escalabilidad

El rápido crecimiento de las tecnologías integradas está llevando a una enorme implementación de dispositivos miniaturizados (sensores, actuadores, etc.). A medida que aumenta el número de dispositivos, los datos producidos por estos dispositivos crecen sin límites. Por lo tanto, manejar el crecimiento del número de dispositivos e información que producen es un desafío enorme en IoT. SDN puede mejorar los problemas de escalabilidad en la red IoT donde el controlador SDN supervisa el dominio de la red y se comunica con otros controladores SDN para intercambiar información agregada en toda la red. El modelo SDN distribuido tiende a compartir la

carga entre varios controladores, lo que ayuda fácilmente a adaptarse a los usuarios y las aplicaciones. Esto trae muchos beneficios como (1) escalabilidad (2) reducción de la latencia desde los nodos del sensor al controlador más cercano, (3) equilibrio de carga (4) tolerancia a fallas, entre otros. Con múltiples controladores, esto se puede utilizar para descargar tareas computacionales que traen beneficios en términos de administración. Cada dominio tiene su controlador SDN que controla todo el tráfico en su dominio. Cuando falla un controlador SDN, otro controlador SDN puede tomar el control para evitar fallas en la red.

1.1.1.11 Industria 4.0

La Cuarta revolución industrial se conoce como Industria 4.0, que es el concepto de lograr una innovación más rápida en la fabricación con una mayor eficiencia en la cadena de suministro. Esta revolución proporciona una mejor manera de utilizar la información, la fabricación y los servicios para proporcionar una mejor calidad de vida. Por lo tanto, ofrece nuevas oportunidades en la fabricación y extiende el límite de la innovación a través de la aplicación de Internet de las cosas e Internet de servicios [39], [40].

Concepto

La industria 4.0 es la combinación del proceso de automatización, las unidades de fabricación y las máquinas inteligentes. Consiste en digitalización, Internet de las cosas, red interna conectada, recursos humanos para supervisión, control de supervisión y adquisición de datos (SCADA), robots para la automatización de muchas funciones críticas, válvulas, sensores, actuadores, sistema PLC, protocolos de comunicación y seguridad cibernética. Utiliza inteligencia artificial para compartir información digitalmente, permite crear una visibilidad total de la operación que proporciona un mayor nivel de información sobre calidad, inventario, materia prima, desperdicio, producción, activos y visibilidad de la demanda de los clientes y ayuda a las industrias a ahorrar tiempo y dinero, mejorar la satisfacción del cliente. Crea una buena relación entre el proveedor y el fabricante [41].

Características

- *Interconexión*

La interconexión entre dispositivos es el contenido central de "Industria 4.0". Por ejemplo, la interconexión entre todo tipo de equipos que tienen diferentes funciones se compone de talleres y fábricas inteligentes. La interconexión entre equipos y productos se refiere a que pueden comunicarse mediante un lenguaje especial. Al leer la información del producto, el equipo puede lograr una producción inteligente. Cyber Physical System (CPS) puede realizar la integración del mundo de la red y el mundo físico conectando el único dispositivo inteligente con Internet. De esta manera, estos dispositivos inteligentes pueden lograr autoadaptación, autodiagnóstico, reparación automática y asistencia remota [42] .

- *Integración*

A través del sistema de información física, "Industrial 4.0" puede combinar sensores de sistemas de producción, sistemas de control y equipos de producción para formar una red inteligente. El objetivo de la integración vertical es lograr una conexión perfecta de la información del producto, como el diseño del producto, la fabricación, la logística, el transporte y el mantenimiento [43].

- *Big Data*

Durante el proceso de producción, el equipo de fabricación inteligente producirá una gran cantidad de información y datos relacionados con la producción. Necesitamos recopilar la información de datos y dar retroalimentación a todos los aspectos de la producción. De esta manera, podemos lograr un proceso de operación eficiente de alta calidad [44].

1.2 Objetivos

1.2.1 Objetivo General

Implementar un testbed de redes definidas por software aplicado al internet de las cosas.

1.2.2 Objetivos Específicos

- Analizar la influencia de las redes definidas por software en aplicaciones del internet de las cosas

Este objetivo es de carácter investigativo y es de fundamental importancia para visualizar las ventajas que pueden llegar a tener las aplicaciones IoT implementadas en un entorno SDN.

Para el cumplimiento de este objetivo se realizan las siguientes actividades:

1. Determinar las características de la arquitectura IoT.
2. Análisis de las funcionalidades de las SDN.
3. Relacionar el uso de SDN en la arquitectura IoT.

- Establecer un escenario de aplicación del internet de las cosas basado en redes definidas por software.

En este objetivo se escoge un campo de aplicación del IoT para ser implementado sobre SDN tomando en cuenta los recursos disponibles.

Este objetivo es cumplido al desarrollar las siguientes actividades:

1. Descripción de un tesbed SDN-IoT.
2. Selección de los componentes de hardware para el Tesbet.
3. Selección de los componentes de software para el Testbed.

- Diseñar el testbed de redes definidas por software aplicado al internet de las cosas.

En este objetivo se lleva a cabo el desarrollo práctico del prototipo, una vez escogido el campo de aplicación IoT se implementan los diferentes componentes de la SDN y se realizan pruebas de funcionamiento.

Para la consecución de este objetivo será necesario realizar las siguientes actividades:

1. Análisis de requerimientos para el desarrollo de una aplicación SDN.
2. Desarrollo de la aplicación SDN.

3. Implementación de Switches OpenFlow.
4. Implementación del controlador OpenFlow.
5. Implementación de la aplicación SDN.
6. Establecimiento de funciones de visualización y control del testbed SDN-IoT.
7. Determinación de pruebas de funcionamiento

CAPÍTULO II

METODOLOGÍA

2.1 Materiales

2.1.1 Selección de los componentes de hardware y software para el Testbed

2.1.1.1 Capa de procesos

Proceso uno

El proceso uno estará estructurado por tres partes: planta, control y monitorización. En la parte de planta se propone hacer uso de los equipos y materiales disponibles en el laboratorio de máquinas eléctricas de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial de la Universidad Técnica de Ambato. Para la parte de control se propone utilizar una NodeMCU basada en el ESP8266, ya que es una placa totalmente abierta a nivel de software y hardware para el desarrollo de aplicaciones IoT, esta placa incorpora un módulo WiFi ideal para la conexión inalámbrica con la SDN del prototipo a implementar. Finalmente, para la parte de monitorización se propone emplear una Raspberry Pi 3 Model B+ configura da como una cámara IP, ya que es una plataforma de hardware libre, de bajo costo y con conexión LAN inalámbrica incluida. En la Figura 12 se puede apreciar el diagrama de bloques de proceso uno.

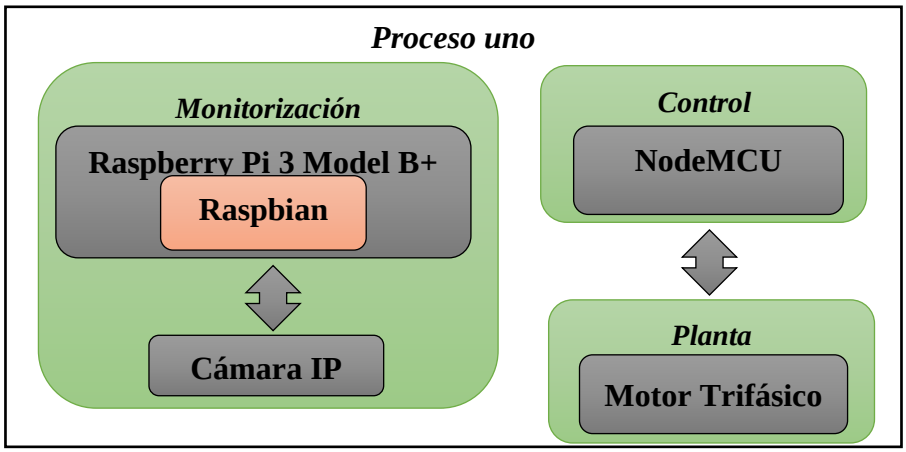


Figura 12.Diagrama de bloques del proceso uno.

Elaborado por el autor.

Proceso dos

El proceso dos estará estructurado por tres partes de igual forma que el proceso uno: planta, control y monitorización. En la parte de planta se propone hacer uso de los equipos y materiales disponibles en el laboratorio de automatización industrial de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial de la Universidad Técnica de Ambato. Para la parte de control se propone utilizar una NodeMCU basada en el ESP8266, ya que es una placa totalmente abierta a nivel de software y hardware para el desarrollo de aplicaciones IoT. Finalmente, para la parte de supervisión se propone implementar una cámara IP haciendo uso de un smartphone a través de una aplicación Android. En la Figura 13 se puede apreciar el diagrama de bloques de proceso dos.

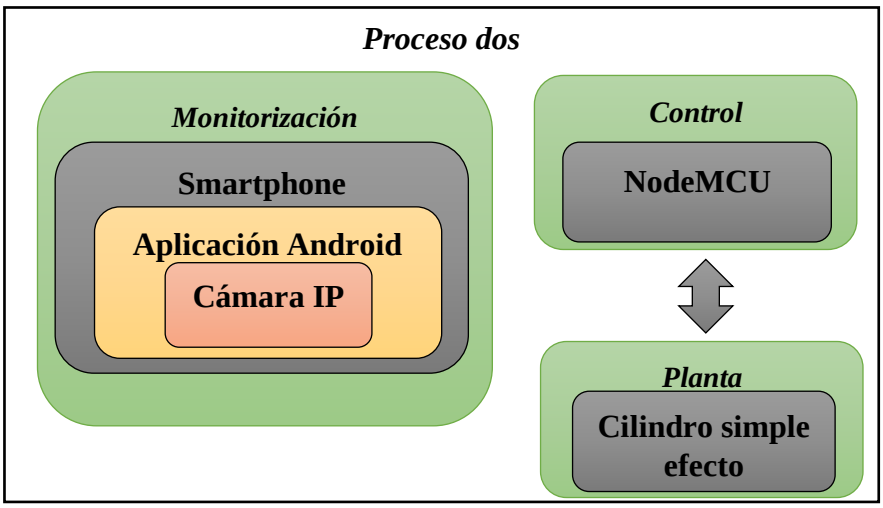


Figura 13. Diagrama de bloques del proceso uno.

Elaborado por el autor.

2.1.1.2 Capa SDN

Controlador SDN

Para la implementación del controlador SDN se propone una PC con requerimientos mínimos como: 2 GB de RAM, 25 GB de espacio libre en el disco duro y procesador de doble núcleo (2 Ghz) y con el sistema operativo Ubuntu 18.04 LTS ya que es un sistema operativo de código abierto y una distribución de Linux.

Al analizar la Tabla 5 se determinó usar el controlador RYU por ser uno de los controladores con mejores características en cuanto al lenguaje de programación,

soporte, documentación y simplicidad. Además, provee una gran cantidad de plataformas en línea que facilitan su interpretación y programación.

Este controlador está desarrollado en el lenguaje de programación Python, por lo que posee las ventajas de dicho lenguaje. Python es un lenguaje de programación interpretado, o también llamado de Scripting, por lo que no requiere ser previamente compilado. Su sintaxis y semántica es relativamente sencilla a comparación de lenguajes similares y es totalmente compatible con el paradigma de la programación orientada a objetos. Además, Ryu posee compatibilidad con el protocolo OpenFLow 1.3, dentro del controlador se desarrollará la aplicación firewall para el control de acceso a la red. En la Figura 14 se aprecia la estructura del controlador SDN a través de un diagrama de bloques.

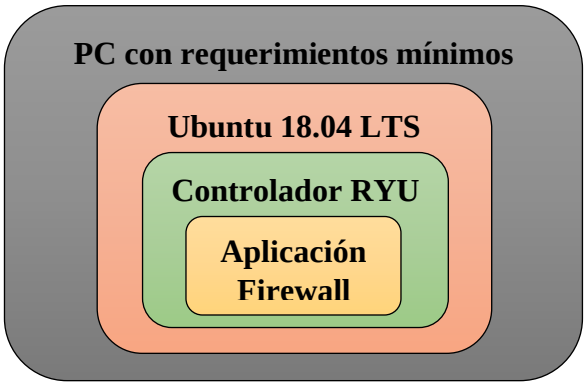


Figura 14. Diagrama de bloques del Controlador SDN.

Elaborado por el autor.

Switch OpenFlow

Para la implementación del switch OpenFlow se propone una Raspberry Pi 3 Model B+ ya que es una plataforma de hardware libre y de bajo costo, esta Raspberry cuenta con un solo puerto de red Fast Ethernet, para extender los puertos de red se propone utilizar 2 adaptadores USB 2.0 a Fast Ethernet para la conexión de los puntos de acceso y 1 adaptador USB 3.0 a Gigabit Ethernet para la conexión con el servidor IoT.

En lo que respecta al software en la Raspberry Pi 3 Model B+ se propone instalar la última versión de Raspbian que es una distribución open source de Linux, además de instalar en Raspbian un switch virtual "*Open vSwitch*" compatible con el protocolo OpenFlow 1.3. En la Figura 15 se puede apreciar la estructura del Switch OpenFlow a través de un diagrama de bloques.

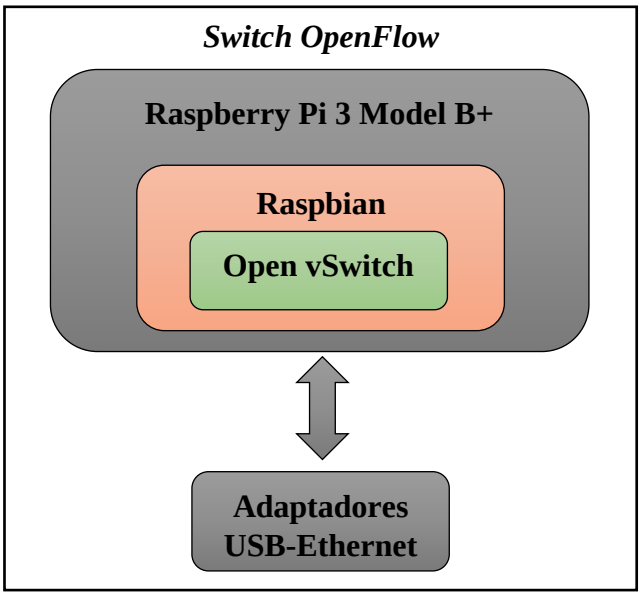


Figura 15. Diagrama de bloques del switch OpenFlow.

Elaborado por el autor.

Access Point OpenFlow

Para la implementación de los Access Point OpenFlow se propone la Raspberry Pi 3 Model B ya que es una plataforma de hardware libre, de bajo costo, además dispone de conexión LAN inalámbrica incluida y con un puerto Fast Ethernet. Se propone añadir un adaptador USB 3.0 a Gigabit Ethernet para la conexión con el switch OpenFlow.

Para la creación del Access Point OpenFlow se propone instalar OpenWrt que es firmware basado en una distribución de Linux utilizado en dispositivos tales como routers personales, además de instalar en OpenWrt un switch virtual "*Open vSwitch*", para que exista compatibilidad con el protocolo OpenFlow 1.3. En la Figura 16 se puede apreciar la estructura del Access Point OpenFlow a través de un diagrama de bloques.

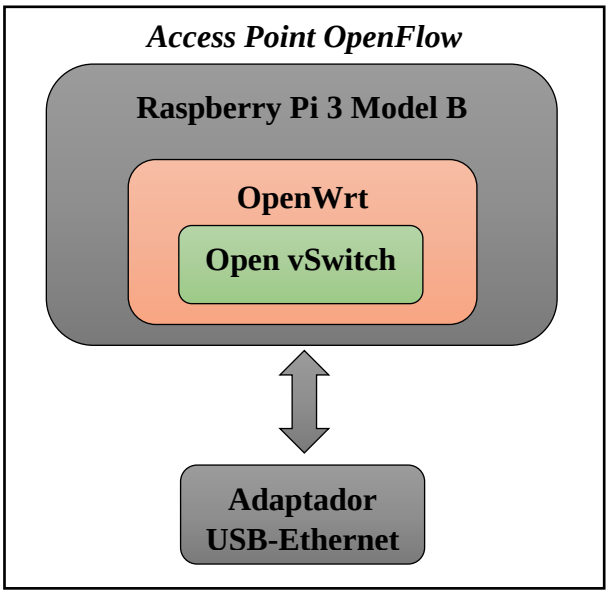


Figura 16. Diagrama de bloques del Access Point OpenFlow.
Elaborado por el autor.

2.1.1.3 Capa de aplicación IoT

Servidor IoT

Para la implementación del servidor IoT se hace uso de una PC con sistema operativo Windows 8.1 disponible por parte del investigador y que cuenta con las siguientes características técnicas:

- Windows 8.1
- Procesador Intel® Core™ i7 2.40GHz
- Pantalla WLED HD de 39,6 cm (15,6 pulg.)
- Memoria 8 GB DDR4-2400 SDRAM (1 x 8 GB)
- Disco duro SSD de 1TB
- Gráficos Intel® NVIDIA

En lo que respecta al software se propone utilizar Mosquitto y Node RED que trabajan bajo el protocolo MQTT, y son softwares open source.

2.2 Métodos

2.2.1 Modalidad de investigación

El presente proyecto fue de carácter investigativo y de desarrollo (I+D), puesto que se aplicaron los conocimientos teóricos indagados, para el desarrollo de este. Haciendo uso de la investigación bibliográfica se buscó en libros, papers, revistas científicas y bases de datos para justificar los argumentos establecidos y brindar soporte al desarrollo de técnicas específicas para la elaboración del proyecto.

La presente investigación también fue de carácter experimental, puesto que se implementó un escenario de pruebas para analizar la influencia de las redes definidas por software en aplicaciones del IoT.

2.2.2 Recolección de información

A través del internet y buscadores conocidos se recolectó la información de papers, revistas científicas, libros, tesis y bases de datos, también se contaró con la información brindada por el tutor.

2.2.3 Procesamiento y análisis de datos

Para el procesamiento y análisis de datos se llevaron a cabo los pasos descritos a continuación:

- Establecer prioridades en la búsqueda de información.
  - Decidir qué información buscar
  - Decidir como buscar la información
- Determinar un sistema de almacenamiento y nombramiento de archivos.
- Analizar la información recopilada a través de una lectura comprensiva.
- Interpretar los resultados

2.2.4 Desarrollo del proyecto

- Determinar las características de la arquitectura IoT.
- Análisis de las funcionalidades de las SDN.
- Relacionar el uso de SDN en la arquitectura IoT.
- Descripción de un tesbed SDN-IoT.
- Selección de los componentes de hardware para el Tesbet.
- Selección de los componentes de software para el Testbed.
- Análisis de requerimientos para el desarrollo de una aplicación SDN.
- Desarrollo de la aplicación SDN.
- Implementación de Switches OpenFlow.
- Implementación del controlador OpenFlow.
- Implementación de la aplicación SDN.
- Establecimiento de funciones de visualización y control del testbed SDN-IoT.
- Determinación de pruebas de funcionamiento.
- Elaboración del trabajo escrito.

CAPÍTULO III

RESULTADOS Y DISCUSIÓN

3.1 Análisis y discusión de los resultados

3.1.1 Desarrollo de la propuesta

3.1.1.1 Descripción del testbed SDN-IoT

Para el presente proyecto se establece un escenario SDN con aplicación IoT basada en la industria 4.0, para lo cual en la Figura 17 se muestra la arquitectura del prototipo a implementar, basada en la arquitectura IoT-SDN propuesta en la sección 1.1.1.8

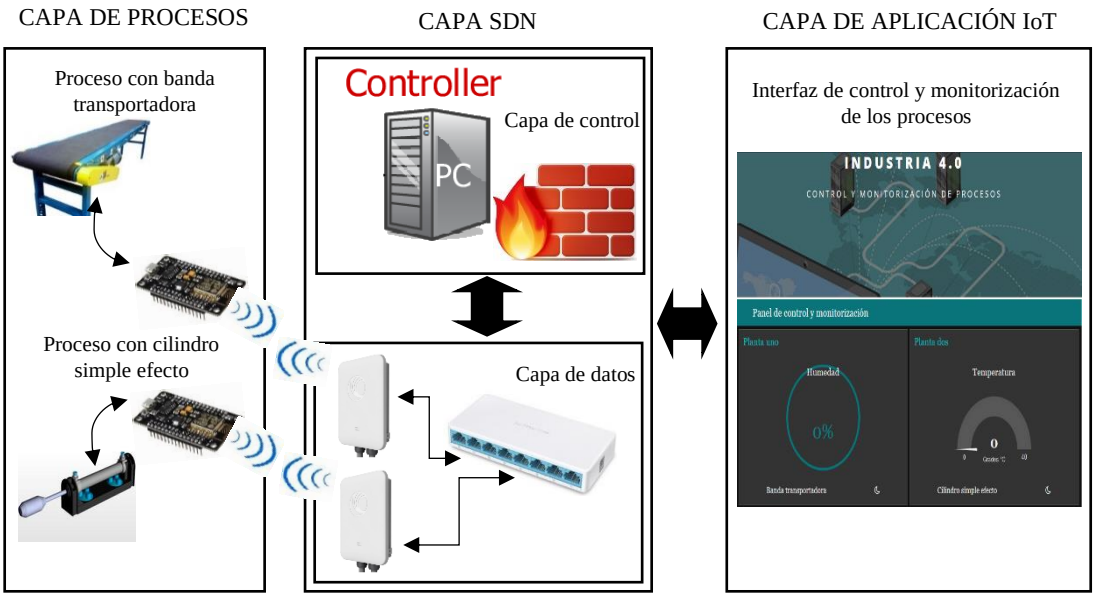


Figura 17. Arquitectura IoT-SDN del prototipo a implementar.
Elaborado por el autor.

- **Capa de procesos**

Esta capa cuenta con 2 procesos referentes a la industria 4.0, estos procesos cuentan con 2 dispositivos IoT que interactuan con la capa SDN para proveer seguridad, escalabilidad y heterogeneidad en la industria 4.0.

El proceso uno es el encargado del control y monitorización en tiempo real de un motor trifásico con arranque estrella triangulo, este tipo de motores comúnmente son utilizados en bandas transportadoras en la industria, mientras que, el proceso dos es el encargado del control y monitorización en tiempo real de un cilindro de simple efecto, este tipo de cilindros comúnmente son usados en aplicaciones de fijación o remache de piezas.

- **Capa SDN**

Esta capa se divide en 2 subcapas, control y datos. La capa de control estará lógicamente centralizada por un controlador SDN que opera de acuerdo con la información obtenida de la capa de datos, en el controlador se implementará una aplicación SDN que permitirá brindar una solución de seguridad, escalabilidad y heterogeneidad a la capa de procesos. La capa de datos estará formada por un conmutador y dos Access Point, estos elementos dejan la toma de decisiones a la capa de control permitiendo un control programable. La capa SDN trabajará con el protocolo OpenFlow v1.3.

Además, en esta capa se desarrollará una interfaz gráfica que brinde una facilidad de manejo de la aplicación SDN al usuario encargado de la administración de la red.

- **Capa de aplicación IoT**

En esta capa se implementará un servidor IoT haciendo uso de del protocolo MQTT para intercambiar datos con la capa de procesos, además, se implementará una interfaz gráfica de usuario que permita hacer el control y monitorización de los procesos de la industria 4.0.

3.1.1.2 Direccionamiento y topología del prototipo

Para el direccionamiento del testbed se consideran dos redes: la red definida por software y la red de área local inalámbrica para la aplicación IoT.

- **Direccionamiento de la red definida por software**

Esta red está compuesta por el controlador, 1 switch OpenFlow y 2 access point OpenFlow, en la Tabla 10 se muestra el direccionamiento para cada dispositivo de la SDN a partir de la dirección IP de clase C ***192.168.2.0/24***.

Tabla 10. Direccionamiento de la SDN.

Dispositivo	**Dirección**	**Mascara**
Controlador	192.168.2.2	/24
Switch OpenFlow	192.168.2.3	/24
Access Point uno	192.168.2.4	/24
Access Point dos	192.168.2.5	/24

Elaborado por el autor.

- **Direccionamiento de la red de área local inalámbrica**

Para el direccionamiento de la WLAN se ha optado por una dirección IP: ***192.168.3.0/24*** de clase C; debido al número de dispositivos IoT para el testbed a implementar. En la Tabla 11 se muestra la distribución de direcciones IP a utilizar.

Tabla 11. Direccionamiento de la WLAN.

Dispositivo	**Dirección**	**Mascara**	**Hosts totales**	**Rango de IP's**
Access Point uno	192.168.3.0	/24	62	192.168.3.1-192.168.3.62
Access Point dos	192.168.3.64	/24	62	192.168.3.65-192.168.3.126
Switch OpenFlow	192.168.3.128	/24	1	192.168.3.129

Elaborado por el autor

- **Topología**

En la Figura 18 se muestra la topología del testbet SDN-IoT a implementar.

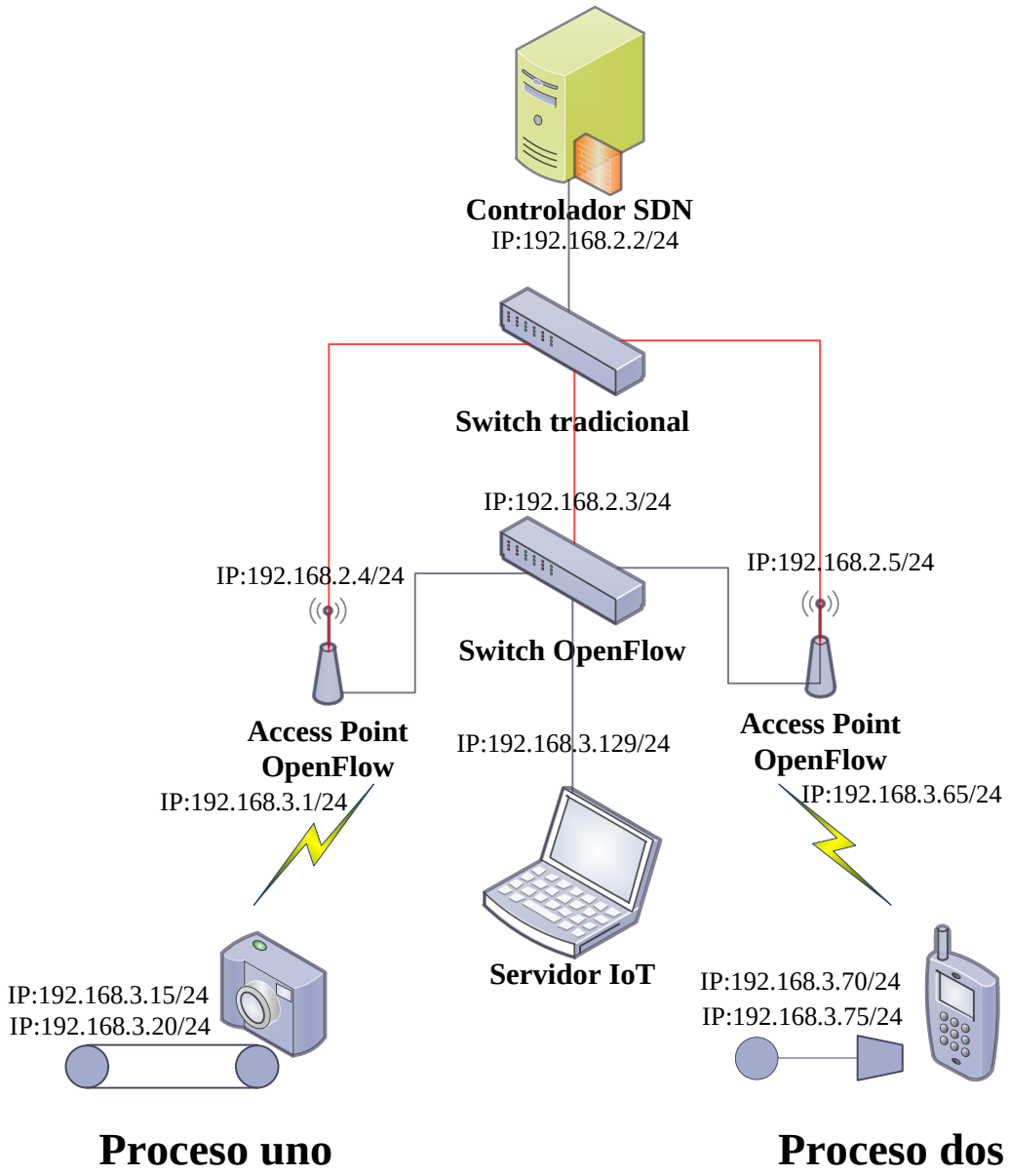


Figura 18.Topología SDN-IoT del Testbed.

Elaborado por el autor.

3.1.1.3 Implementación de los procesos de la industria 4.0 aplicados al IoT

Como se mencionó en la sección 2.1.1.1 cada proceso estará formado por tres partes: planta, control y monitorización, a continuación, se presenta la implementación de las partes de cada proceso.

1. Planta

Proceso uno

Se utilizan dos contactores 220V AC, cables y un motor trifásico. El motor trifásico se conecta de acuerdo con el diagrama mostrado en la Figura 19, para realizar un arranque estrella triangulo.

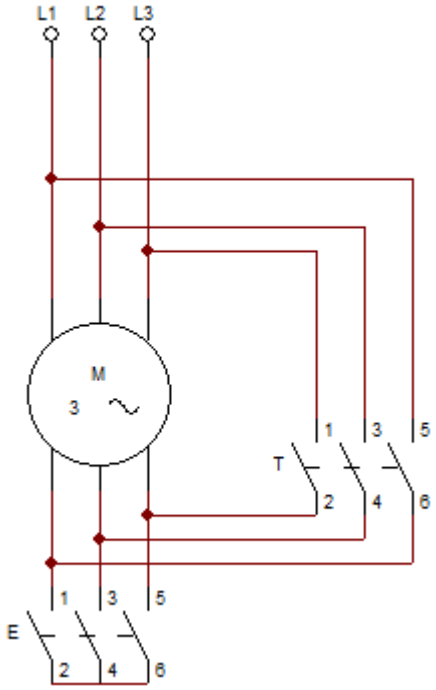


Figura 19. Diagrama eléctrico de potencia del proceso uno.
Elaborado por el autor.

En la Figura 20 se muestra el circuito de potencia del proceso uno implementado para que el motor trifásico tenga un arranque estrella triangulo.

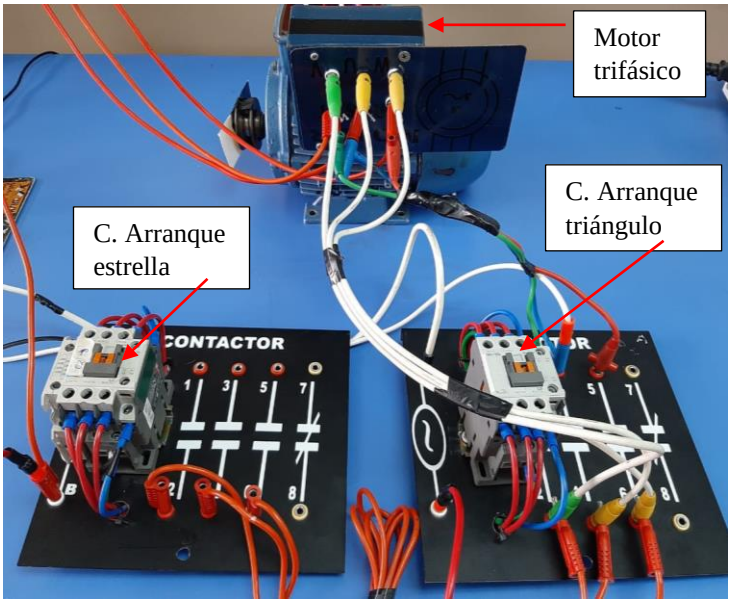


Figura 20. Circuito de potencia del proceso uno implementado.

Elaborado por el autor.

- **Proceso dos**

Se utiliza la unidad de mantenimiento, válvula de distribución neumática, una electroválvula, un cilindro simple efecto y mangueras de 4 y 6 mm. El cilindro simple efecto se conecta de acuerdo con el diagrama mostrado en la Figura 21.

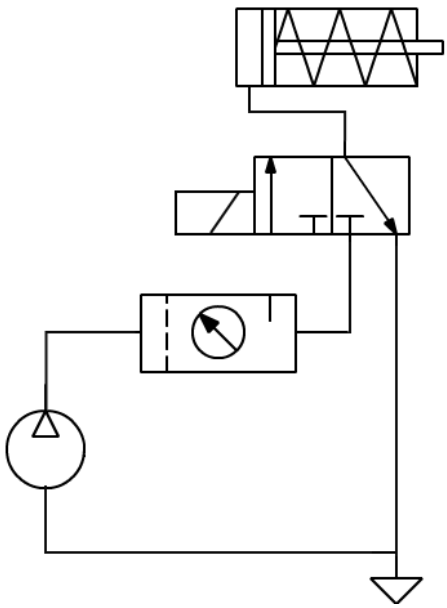


Figura 21. Diagrama neumático del proceso dos.

Elaborado por el autor.

En la Figura 22 se muestra el circuito neumático del proceso dos implementado para la activación y desactivación del cilindro de simple efecto.

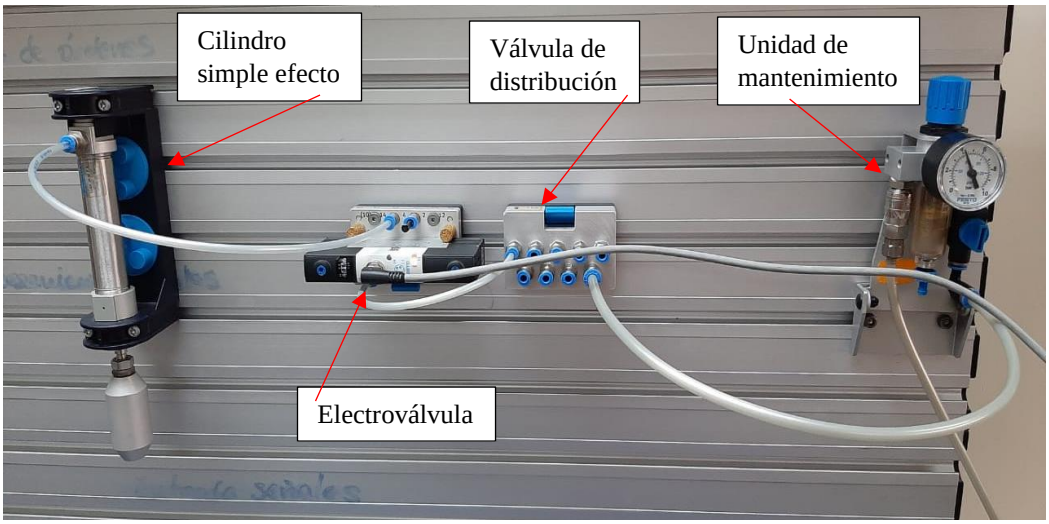


Figura 22. Circuito neumático del proceso dos implementado.

Elaborado por el autor.

2. Control

- **Proceso uno**

Se implementa el diagrama electrónico mostrado en la Figura 23, en el módulo relay de 2 canales se conectan los contactores de 220V para el arranque estrella-triangulo del motor trifásico AC.

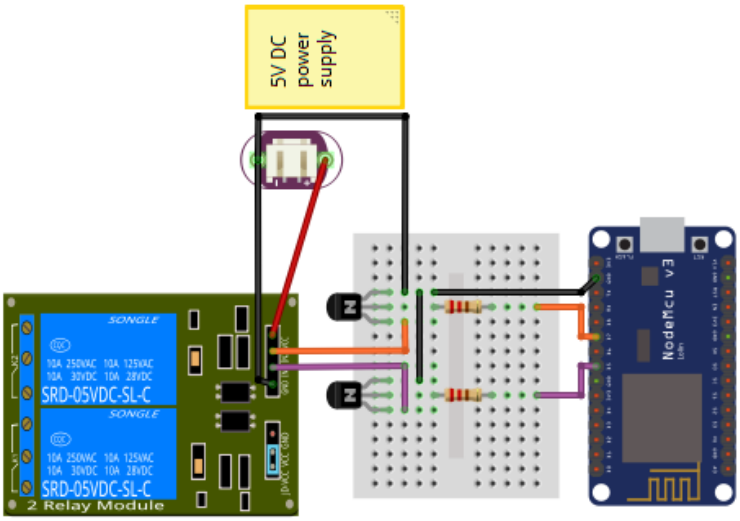


Figura 23. Diagrama electrónico del proceso uno.
Elaborado por el autor.

- **Proceso dos**

Se implementa el diagrama mostrado en la Figura 24, en el relay se conecta la electroválvula de 24V DC para la activación del cilindro de simple efecto.

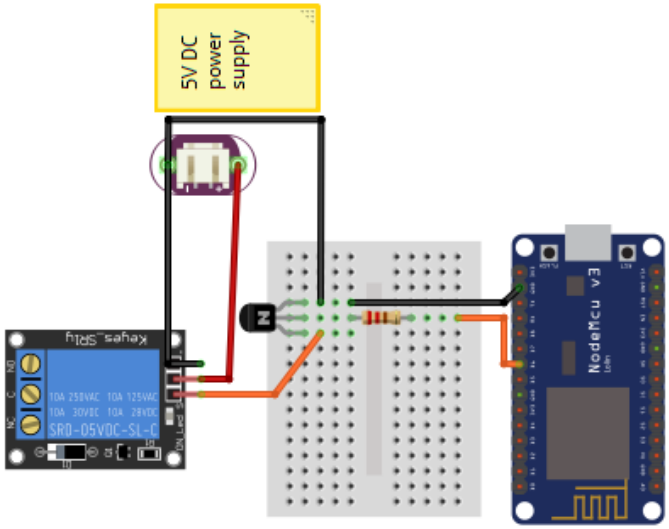


Figura 24. Diagrama electrónico del proceso dos.

Elaborado por el autor.

Una vez implementados los circuitos se procede a programar las NodeMCUs en el IDE de Arduino, el código completo se encuentra en el **ANEXO A**.

3. Monitorización

- **Proceso uno**

Se utiliza una cámara de 5MP incorporada en una Raspberry Pi 3 Modelo B+ con el sistema operativo Raspbian. Para la implementación de la cámara IP se siguen los pasos mostrados a continuación:

1. Configurar la interfaz inalámbrica de la Raspberry Pi para que se conecte directamente con el AP1.

Para Configurar la interfaz wlan0 se ingresa a la Configuración de redes de la Raspberry Pi con el comando: *$sudo nano /etc/network/interfaces* y se establece la Configuración de la interfaz Figura N°25, de acuerdo con el direccionamiento establecido en la topología mostrada en la Figura 18.

```
 GNU nano 2.7.4   File: /etc/network/interfaces

source-directory /etc/network/interfaces.d
auto lo

iface lo inet loopback
iface eth0 inet dhcp

#auto wlan0
allow-hotplug wlan0
iface wlan0 inet static
        address 192.168.3.15
        netmask 255.255.255.0
        gateway 192.168.3.1
        wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

Figura 25. Direccionamiento de la cámara IP del proceso uno.

Elaborado por el autor.

Luego se accede al archivo "*wpa_supplicant.conf"* para la Configuración de conexión con el AP1, para modificar el archivo se utiliza el comando: *sudo nano /etc/wpa_supplicant/wpa_supplicant.conf* y a continuación se establecen los parámetros de conexión mostrados en la Figura 26.

```
      File: /etc/wpa_supplicant/wpa_supplicant.conf

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="Red_AP_uno"
        key_mgmt=NONE
        wep_key0=ab12345678
        wep_tx_keyidx=0
}
```

Figura 26. Parámetros de conexión al AP1.

Elaborado por el autor.

2. Se copian los archivos creados y configura dos que permiten realizar Streaming de video desde la Raspberry Pi, en la ruta mostrada en la Figura 27.

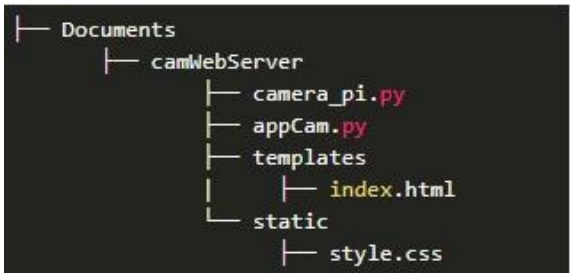


Figura 27. Ubicación de los archivos para el Streaming de video a través de la Raspberry Pi.

Elaborado por el autor.

El código fuente de los archivos se encuentra en el **ANEXO B.**

3. Para la ejecución de la aplicación creada se ejecuta el siguiente comando:

$sudo python3 appCam.py

- **Proceso dos**

Se utiliza un smartphone y en este se instala la aplicación *DroidCam* que permite convertir la cámara de teléfono en una cámara IP, para la ejecución de la aplicación se siguen los pasos mostrados a continuación:

1. Configuración de la red inalámbrica para la conexión del smartphone con el AP2.

En la Configuración de red inalámbrica del smartphone Figura N°28 se establece el direccionamiento propuesto en la topología mostrada en la Figura 18.

Figura 28. Configuración de los parámetros de red del smartphone para la conexión con el AP2.
Elaborado por el autor.

2. Abrir la aplicación

Una vez abierta la aplicación se visualizará la dirección IP del dispositivo y el puerto de red con el que trabaja la aplicación Figura 29.

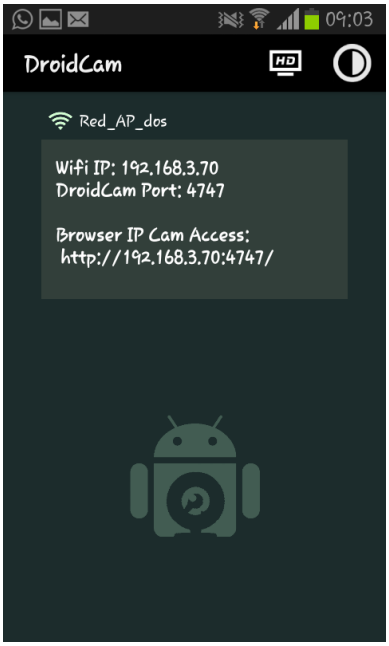


Figura 29. Ejecución de la aplicación DroidCam en el smartphone.

Elaborado por el autor.

Finalmente, se presenta en la Figura 30 y Figura 31, los procesos uno y dos implementados respectivamente.

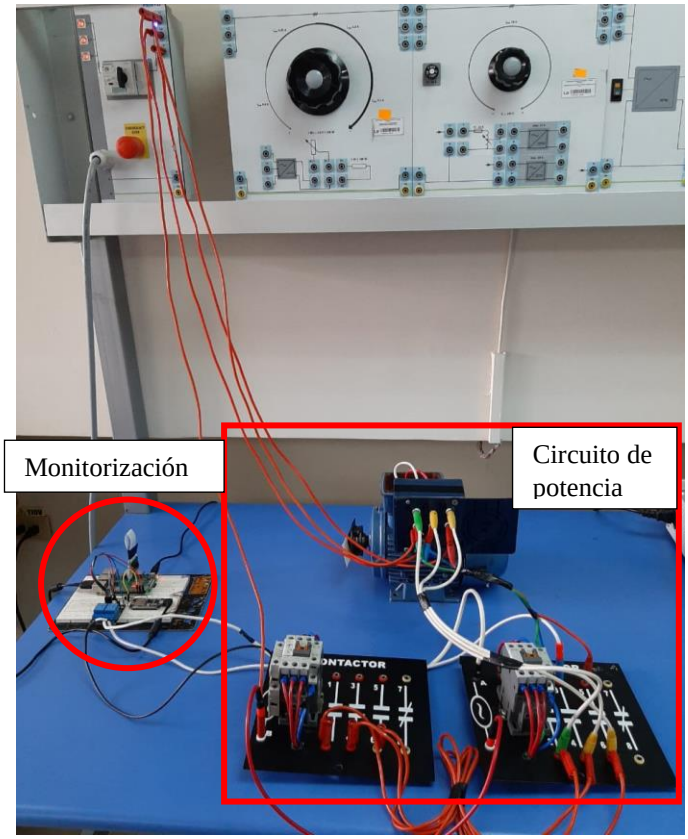


Figura 30. Proceso uno implementado.

Elaborado por el autor.

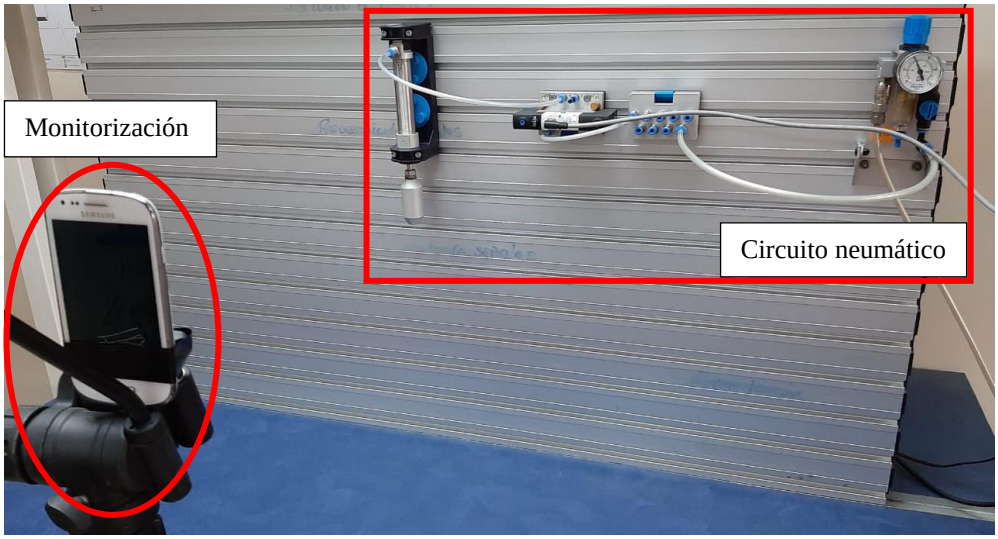


Figura 31. Proceso dos implementado.

Elaborado por el autor.

3.1.1.4 Implementación de la Red Definida por Software

3.1.1.4.1 Implementación del controlador RYU compatible con OpenFlow

- **Instalación del controlador Ryu**

La instalación del controlador Ryu se la lleva a cabo en Ubuntu 18.04 LTS ya que es una distribución de Linux que permite la creación y ejecución de aplicaciones codificadas en Python. A continuación, se describen los pasos para la instalación del controlador Ryu.

1. Actualización de los repositorios de Ubuntu 18.04 con los siguientes comandos:

$sudo apt-get update

$sudo apt-get upgrade

2. Instalación del sistema de gestión de paquetes *pip* utilizado para instalar y administrar paquetes de software escritos en Python, además se instala el paquete Python-dev que contiene los archivos de encabezado que se necesita para construir extensiones de Python.

$sudo apt-get -y install git python-pip python-dev

3. Instalación de paquetes faltantes de Python

$sudo apt-get -y install python-eventlet python-routes python-webob python-paramiko

4. Clonar RYU de los repositorios git

$git clone --depth=1 https://github.com/osrg/ryu.git

5. Instalar RYU

$sudo pip install setuptools –upgrade

$cd ryu

$sudo python ./setup.py install

6. Instalar y actualizar paquetes de Python

$sudo pip install six –upgrade

$sudo pip install oslo.config msgpack-python

$sudo pip install eventlet –upgrade

$sudo pip install -r tools/pip-requires

7. Ejecutar el controlador RYU y verificar su funcionamiento Figura 32.

$ryu-manager –version

$ryu-manager

```
cristian@HP-Notebook-PC:~$ ryu-manager --version
ryu-manager 4.34
cristian@HP-Notebook-PC:~$ ryu-manager
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Figura 32.Versión instalada y arranque del controlador RYU.

Elaborado por el autor.

- **Configuración de la interfaz LAN para el controlador Ryu**

Para Configurar la interfaz LAN del controlador se ejecuta el siguiente comando y se establece la dirección IP estática asignada en la Tabla 10 como se muestra en la Figura 33.

$sudo nano /etc/network/interfaces

```
  GNU nano 2.9.3                  /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
auto enp4s0
iface enp4s0 inet static
address 192.168.2.2
netmask 255.255.255.0
```

Figura 33. Dirección IP estática del controlador RYU.

Elaborado por el autor.

Una vez configura da la dirección IP del controlador se reinicia el equipo.

3.1.1.4.2 Implementación del Switch OpenFlow en la Raspberry Pi 3 Model B+

- **Instalación de Raspbian en la Raspberry Pi 3 Model B+**

Para instalar la distribución Raspbian en la Raspberry Pi 3 Model B+ se siguen los pasos mostrados a continuación.

1. Descargar la distribución Raspbian

Se accede al sitio web de la fundación Raspberry Pi para encontrar la última versión de Raspbian [45], en este caso se procede a la descarga del archivo con extensión ZIP de la versión recomendada por el sitio web como se muestra en la Figura 34.

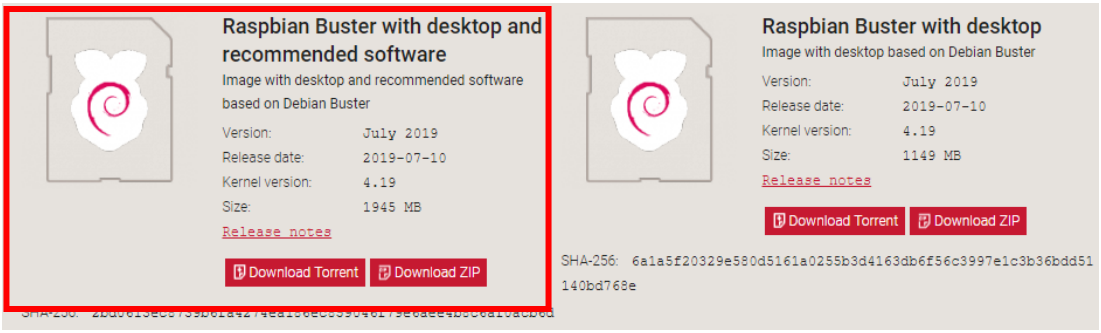


Figura 34. Versión actualizada de Raspbian *[45]*.

2. Descomprimir el archivo

Una vez descargado el archivo (.zip) se descomprime, para obtener el archivo imagen (.img) como se muestran la Figura 35 y proceder a escribirlo en la tarjeta microSD.

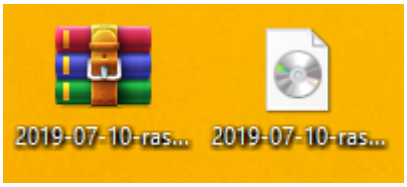


Figura 35. Descompresión del S.O. Raspbian.

Elaborado por el autor

3. Escribir la imagen del disco en una tarjeta microSD

Antes de comenzar es necesario revisar los requisitos de la tarjeta microSD [46], para el presente proyecto se utilizan microSDs clase 10 de 32GB, estos parámetros están dentro de los requisitos establecidos por el sitio web.

Posteriormente se introduce la tarjeta microSD en la computadora y se escribe la imagen del disco en ella. Es necesario utilizar un programa específico para la escritura de la imagen en la tarjeta, en este caso se ha utilizado el programa Rufus portable como se ilustra en la Figura 36, este software permite formatear y crear un soporte de arranque con la distribución del Sistema Operativo en la microSD.

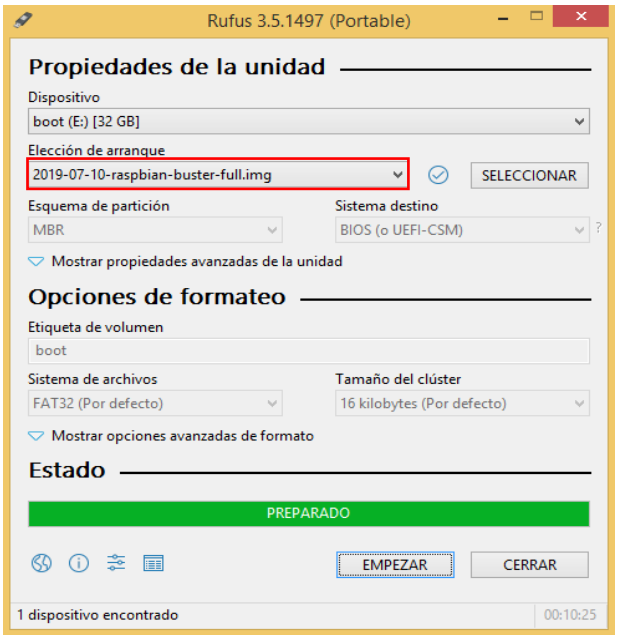


Figura 36. Escritura de la imagen con Raspbian en la tarjeta microSD.

Elaborado por el autor

4. Insertar la tarjeta microSD en la placa y encenderla.

Una vez que la imagen del disco se ha escrito en la tarjeta microSD se procede a insertarla en la placa, conectar los periféricos y la fuente de alimentación. La edición actual de Raspbian se iniciará directamente en el escritorio. Sus credenciales predeterminadas son: username→ ***pi***, password→ ***raspberry***

- **Instalación de Open vSwitch en el sistema operativo Raspbian**

Para instalar Open vSwitch en Raspbian se verifica la conexión a internet de la Raspberry Pi y se realizan los siguientes pasos:

1. Actualización de los repositorios de Raspbian utilizando los siguientes comandos:

$sudo apt-get update

$sudo apt-get upgrade

2. Iniciar sesión en la raíz como super usuario con el comando

$sudo su

3. A diferencia de Ubuntu, no hay un paquete pre compilado disponible a través del comando *apt-get* para lo cual el código fuente para compilar Open vSwitch se lo descarga del sitio web del proyecto OvS [47] a través del siguiente comando.

 # wget https://www.openvswitch.org/releases/openvswitch-2.7.0.tar.gz

4. Descomprimir el archivo

 # tar -xvzf openvswitch-2.7.0.tar.gz

5. Instalar las dependencias de compilación con el siguiente comando.

#apt-get install python-simplejson python-qt4 python-twisted-conch automake autoconf gcc uml-utilities libtool build-essential pkg-config

6. Un problema es elegir los archivos de encabezado del núcleo correctos para la compilación del módulo del núcleo. No hay una coincidencia exacta para la versión del núcleo en uso. Para lo cual se busca en la base de datos interna de APT las cabeceras para Linux y se escoge la última actualización de las cabeceras para proceder a instalarla como se muestra en la Figura 37.

 # apt-cache search linux-headers

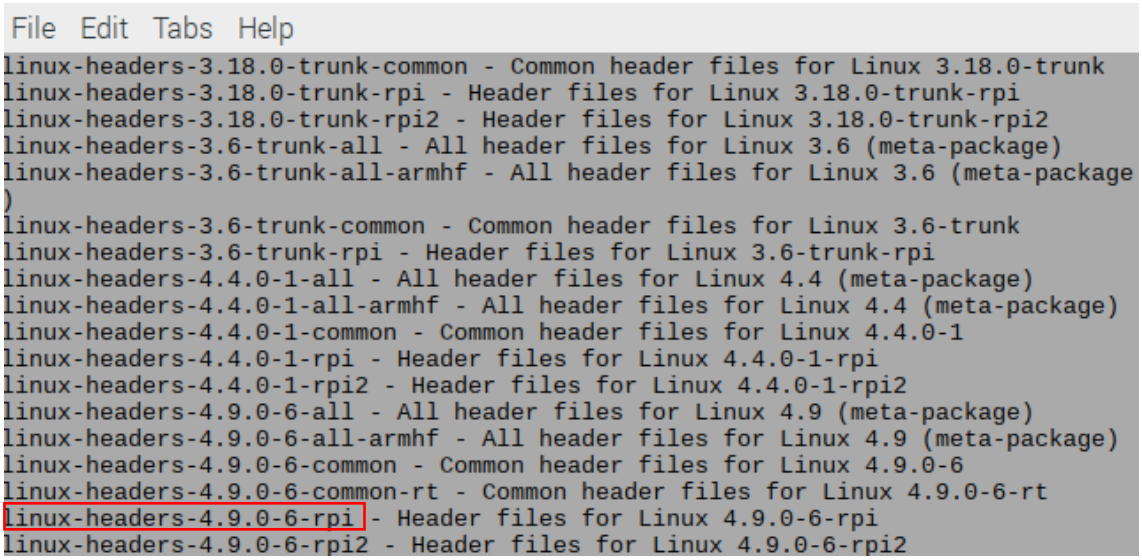


Figura 37. Cabeceras de Linux.

Elaborado por el autor.

7. El paquete correcto de cabeceras a instalar para el kernel 4.14.34-v7+ de raspbian es:

#apt-get install linux-headers-4.9.0-6-rpi

8. A continuación, se configura, crea e instala el módulo del kernel Open vSwitch en el encabezado previamente instalado correspondiente al del sistema operativo:

#cd openvswitch -2.7.0

 # ./boot.sh

 # ./configure --with-linux=/lib/modules/4.9.0-6-rpi/build

#make

#make install

9. Posteriormente se accede a la ruta donde se aloja el *datapath* de Linux Figura 38 para luego cargar el módulo anteriormente creado, al kernel de Linux como se ilustra en las Figura 39, esto se lo realiza ejecutando los siguientes comandos:

#cd /datapath/Linux

#modprobe openvswitch

#cat /etc/modules

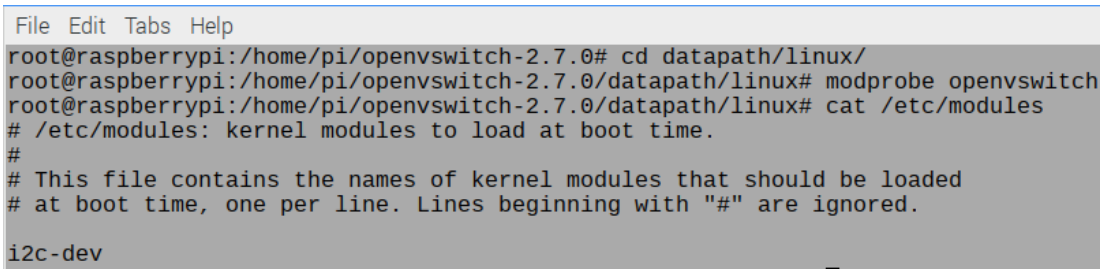


Figura 38. Datapath de Linux sin el módulo Open vSwitch.

Elaborado por el autor.

#echo "openvswitch" >> /etc/modules

#cat /etc/modules

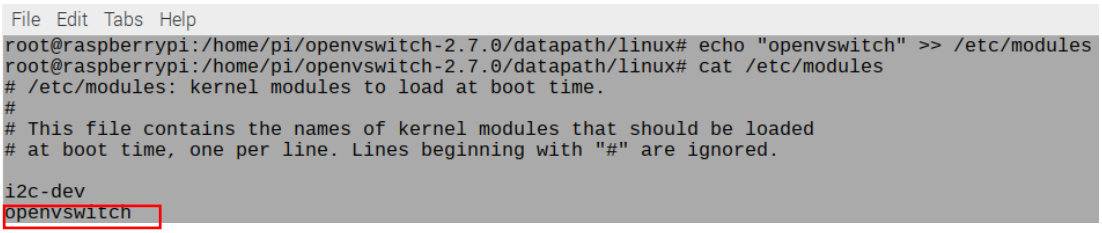


Figura 39. Datapath de Linux con el módulo Open vSwitch.

Elaborado por el autor.

#cd ../..

10. Para completar la instalación, se deben crear dos archivos de Configuración. Por un lado, el archivo de Configuración del vSwitch daemon (ovs-vswitchd), y, por otro lado, la base de datos que contiene la Configuración del conmutador (ovsdb-server), por ejemplo, las interfaces que están conectadas al conmutador virtual.

 # touch /usr/local/etc/ovs-vswitchd.conf

 # mkdir -p /usr/local/etc/openvswitch

#ovsdb-tool create /usr/local/etc/openvswitch/conf.db /root/openvswitch-2.7.0/vswitchd/vswitch.ovsschema

Al principio, estos archivos no contienen ningún contenido, ya que se llenan automáticamente y se pueden modificar a través de las herramientas de línea de comandos de Open vSwitch.

11. Finalmente, se crea un script de inicio Figura N°40 que simplifica el proceso de activación del vSwitch Daemon ejecutando el siguiente comando:

 # nano sw_start.sh

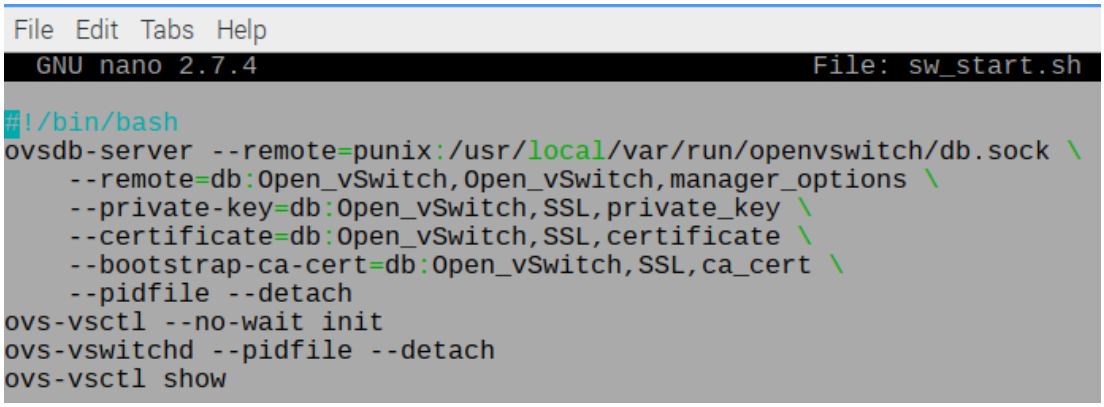


Figura 40. Script de inicio sw_start.sh.

Elaborado por el autor.

 # chmod +x sw_start.sh

#./sw_start.sh

Para que se ejecute automáticamente el script *sw_start.sh* en el arranque de la Raspberry Pi se debe añadir la instrucción mostrada en la Figura 41 dentro del archivo de Configuración */etc/rc.local.* El código fuente completo del script *"sw_start.sh"* es mostrado en el **ANEXO C.**

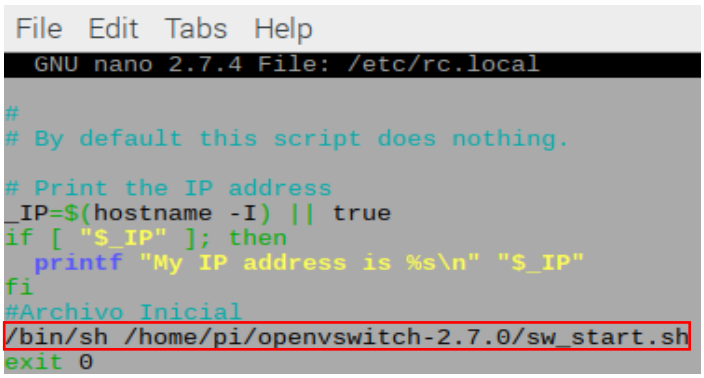


Figura 41. Archivo de Configuración *"rc.local"*.

Elaborado por el autor.

12. Después de ejecutar el script se visualizará el resultado del último comando ejecutado Figura 42. Debe contener la ID de la primera estructura de datos creada automáticamente en el módulo del núcleo que representa un conmutador virtual:

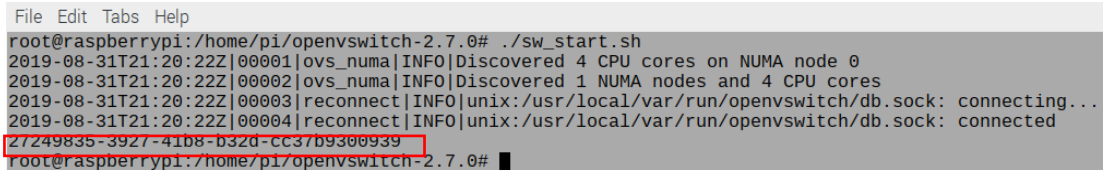


Figura 42. Identificador de la primera escritura de datos del Open vSwitch.

Elaborado por el autor.

13. Finalmente se verifica la versión del Open vSwitch instalado y si sus componentes están funcionando Figura 43.

$ovs-vsctl --version

$ps -e | grep ovs

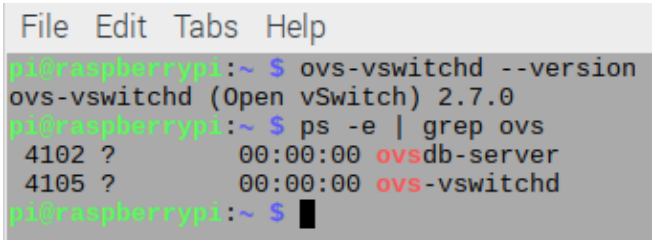


Figura 43. Versión y componentes del Open vSwitch instalado.

Elaborado por el autor.

- **Configuración de los adaptadores ethernet en la Raspberry Pi 3 Model B+**

Se conectan los 3 adaptadores USB ethernet y se verifica que las interfaces estén disponen en la Raspberry Pi Figura 44 con el siguiente comando:

$ifconfing

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.3  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::8bf4:288b:5df6:4be  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:34:4d:50  txqueuelen 1000  (Ethernet)
        RX packets 250  bytes 31085 (30.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 253  bytes 18312 (17.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 00:e0:4d:3a:38:0e  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 00:e0:4c:68:01:f8  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 00:e0:4d:3a:42:2d  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figura 44. Interfaces disponibles en la Raspberry Pi.

Elaborado por el autor.

Configuración interfaz eth0

Se debe Configurar la interfaz eth 0 con la dirección ip estática Figura 45 según el criterio del diseño especificado en la Tabla 10, esta interfaz se conectará con el controlador RYU.

$sudo nano /etc/dhcpcd.conf

```
# Example static IP configuration:
interface eth0
static ip_address=192.168.2.3/24
static routers=192.168.2.1
```

Figura 45. Configuración de la dirección IP estática para la interfaz eth0.

Elaborado por el autor.

Configuración interfaces eth1, eth2

Se preparan las interfaces eth1, eth2 y eth3 para ser agregadas como puertos al conmutador Open vSwitch. Se eliminan las IPs de sus interfaces de datos teniendo cuidado de no eliminar eth0. Para todas las interfaces que no sean eth0, wlan0 y l0, se ejecutan los siguientes comandos:

$sudo ifconfig eth1 0

$sudo ifconfig eth2 0

$sudo ifconfig eth3 0

- **Configuración del Open vSwitch en la Raspberry Pi 3 Model B+**

A continuación, se describen los pasos para la Configuración del Open vSwitch en la Raspberry Pi:

1. Crear un puente Ethernet que actúa como el switch virtual dentro de la raspberry con el siguiente comando:

$sudo ovs-vsctl add-br br0

2. Se configura el puente creado para que trabaje con la versión del protocolo OpenFlow 1.3

$sudo ovs-vsctl set bridge br0 protocols=OpenFlow13

3. Agregar las interfaces eth1, eth2 y eth3 al switch(br0)

$sudo ovs-vsctl add-port br0 eth1

$sudo ovs-vsctl add-port br0 eth2

$sudo ovs-vsctl add-port br0 eth3

4. Establecer un identificador para el Data Path del switch virtual diferente al que tiene por defecto.

$sudo ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000001

5. Apuntar el switch OpenFlow al controlador a través de su dirección IP y del puerto TCP 6633.

$sudo ovs-vsctl set-controller br0 tcp:192.168.2.2: 6633

Un switch OpenFlow no reenviará ningún paquete a menos que lo indique un controlador. Básicamente, la tabla de reenvío está vacía, hasta que un controlador externo inserte reglas de reenvío. El controlador OpenFlow se comunica con el

conmutador a través de la red de control y puede estar en cualquier lugar de Internet siempre que sea accesible por el host OVS.

6. Desactivar el modo a prueba de fallas

$sudo ovs-vsctl set-fail-mode br0 secure

7. Verificar la Configuración de OVS como se muestra en la Figura 46 escribiendo el siguiente comando:

$sudo ovs-vsctl show

```
pi@raspberrypi:~/openvswitch-2.7.0 $ sudo ovs-vsctl show
27249835-3927-41b8-b32d-cc37b9300939
    Bridge "br0"
        Controller "tcp:192.168.2.2:6633"
        fail_mode: secure
        Port "eth1"
            Interface "eth1"
        Port "eth3"
            Interface "eth3"
        Port "eth2"
            Interface "eth2"
        Port "br0"
            Interface "br0"
                type: internal
```

Figura 46. Configuración interna del Open vSwitch.

Elaborado por el autor.

- **Funcionamiento del switch OpenFlow**

En la Figura 47 se muestra el diagrama de flujo que describe el funcionamiento del switch OpenFlow implementado.

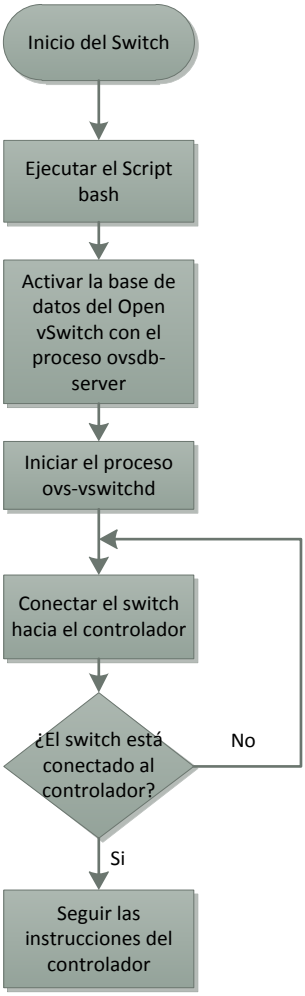


Figura 47. Diagrama de flujo del Switch OpenFlow implementado.

Elaborado por el autor.

3.1.1.4.3 Implementación de los Access Point OpenFlow en la Raspberry Pi 3 Model B

- **Selección de los parámetros inalámbricos de los Access Points**

Para elegir el canal inalámbrico de operación de los Access Points hay que tener en cuenta cuales son los canales menos utilizados en el área donde se implementara el prototipo, con esto se previene la interferencia y se mejora el rendimiento de la comunicación inalámbrica.

Con la ayuda de la herramienta "Analizador WIFI" se puede observar en la Figura 48 el grafico de los canales utilizados por las redes cercanas con su respectiva intensidad de señal.

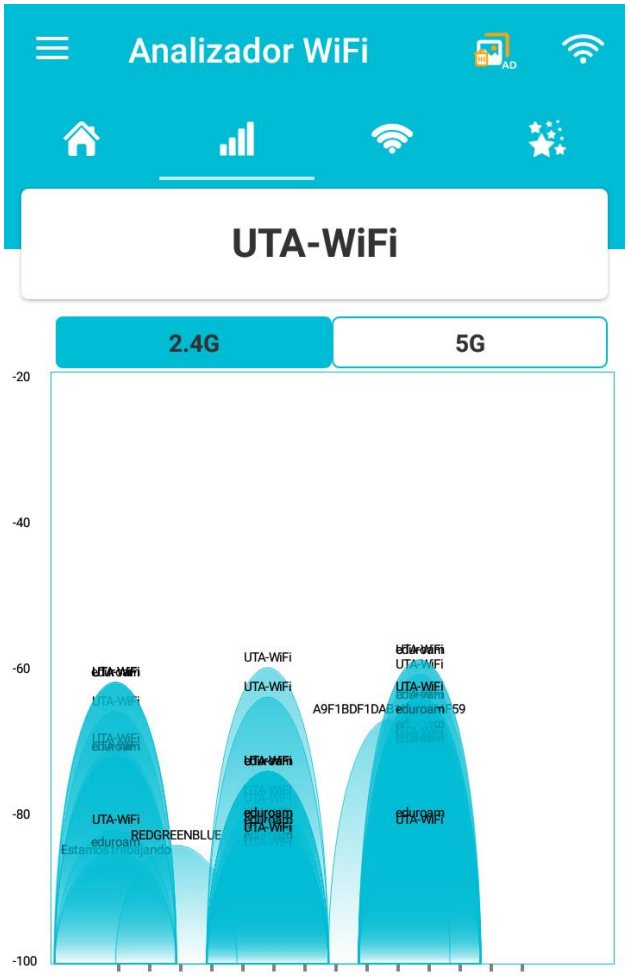


Figura 48. Canales inalámbricos ocupados.

Elaborado por el autor.

En la Tabla 12 se presenta la puntuación de los canales analizados con la ayuda del software "Analizador WIFI", en esta tabla se pueden apreciar los canales inalámbricos menos utilizados por las redes adyacentes al área en el que se implementara el prototipo.

Tabla 12. Uso de los canales inalámbricos.

Canal inalámbrico	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Número de veces de uso del canal	10	9	9	1	3	10	7	2	8	9	10	8	5	6

Elaborado por el autor

Finalmente, en la Tabla 13 se muestran algunos de los parámetros inalámbricos de los puntos de acceso a implementar en el prototipo.

Tabla 13. Parámetros inalámbricos de los AP.

Dispositivo	**Nombre de la red**	**Canal**	**Seguridad**	**Contraseña**	**Modo de operación**
Access Point uno	Red_AP_uno	4	WEP	ab12345678	g
Access Point dos	Red_AP_dos	8			

Elaborado por el autor

- **Instalación de OpenWrt en la Raspberry Pi 3 Model B**

Para instalar la distribución OpenWrt en la Raspberry Pi 3 Model B se siguen los pasos mostrados a continuación.

1. Descargar la imagen iso de OpenWrt para la Raspberry Pi 3 Model B desde la página oficial de OpenWrt [48] como se muestra en la Figura 49.

Figura 49. Imagen iso de OpenWrt para descargar.

Elaborado por el autor.

2. Descompresión del archivo

Una vez descargado el archivo (.zip) se descomprime, para obtener el archivo de imagen (.img) Figura 50 y proceder a escribirlo en la tarjeta microSD.

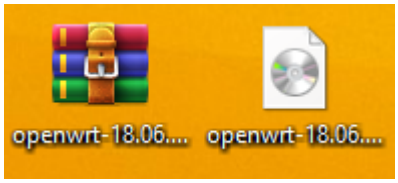


Figura 50. Descompresión de la imagen iso de OpenWRT.

Elaborado por el autor

3. Escribir la imagen del disco en una tarjeta microSD

Posteriormente se introduce la tarjeta microSD en la computadora y se escribe la imagen del disco en ella. Es necesario utilizar un programa específico para la escritura de la imagen en la tarjeta, en este caso se ha utilizado el programa Rufus portable como se ilustra en la Figura 51, este software permite formatear y crear un soporte de arranque con la distribución del Sistema Operativo en la microSD.

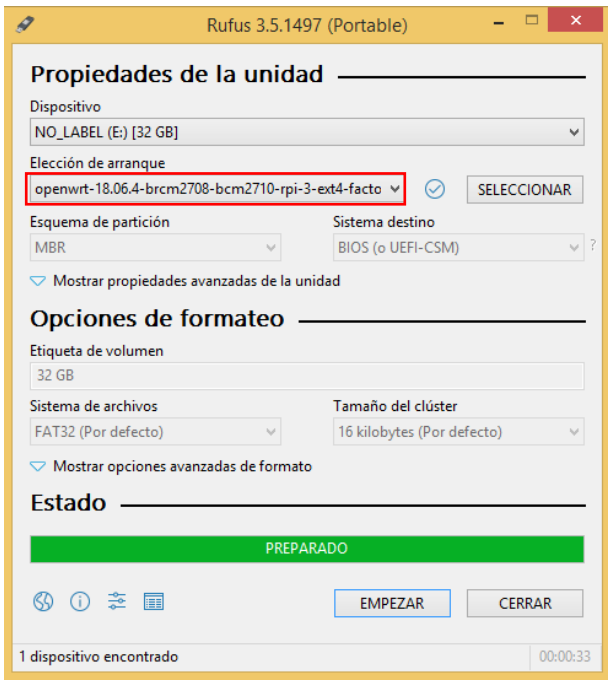


Figura 51. Escritura de la imagen con OpenWrt en la tarjeta microSD.

Elaborado por el autor

4. Insertar la tarjeta microSD en la placa y encenderla.

Ahora se inserta la tarjeta microSD, se conecta el cable Ethernet y la fuente de alimentación para completar la Configuración. Se enciende la Raspberry Pi, y se espera 2 minutos para que inicie el sistema operativo OpenWrt.

5. Se debe ejecutar la herramienta *Wireless Network Watcher* [49], para para verificar la dirección IP de Raspberry Pi como se muestra en la Figura 52.

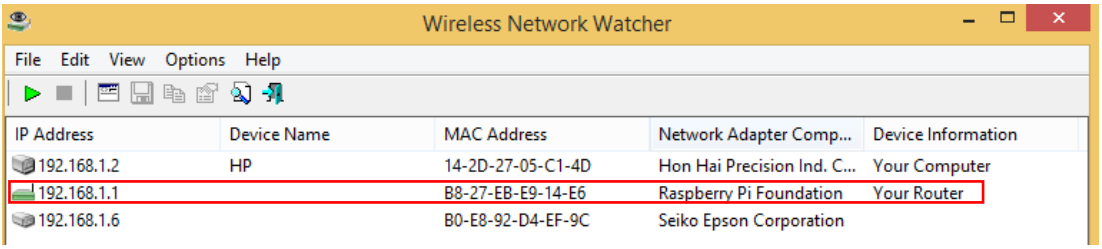


Figura 52. Detección de la dirección IP de la Raspberry Pi analizada con Wireless Network Watcher.

Elaborado por el autor.

6. A continuación, se utiliza Putty para iniciar sesión en el enrutador OpenWRT.

Se escribe la dirección IP de la Raspberry Pi verificada en el paso anterior como se muestra en la siguiente Figura 53.

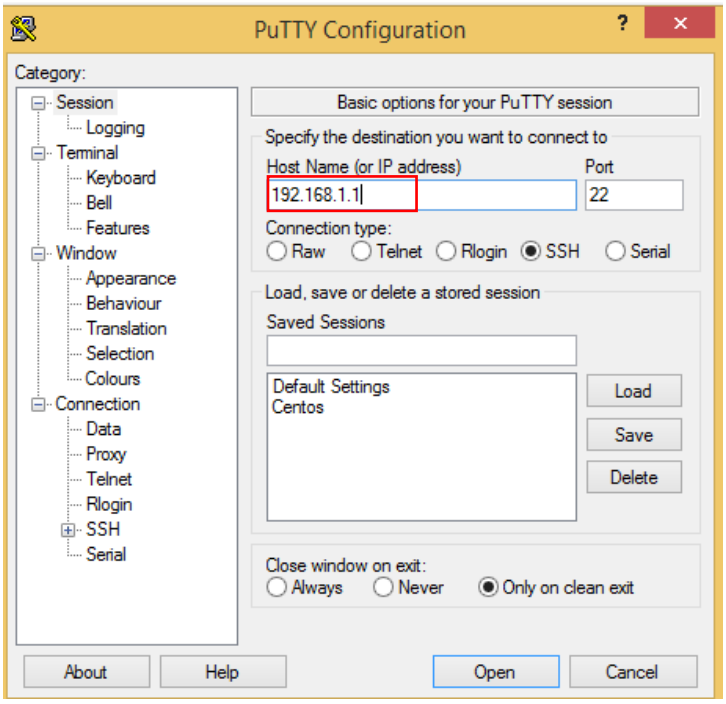


Figura 53. Acceso remoto a la Raspberry Pi a través de Putty.

Elaborado por el autor.

Para acceder al sistema OpenWrt por consola se escribe la credencial por defecto "*root"* como se muestra en la Figura 54.

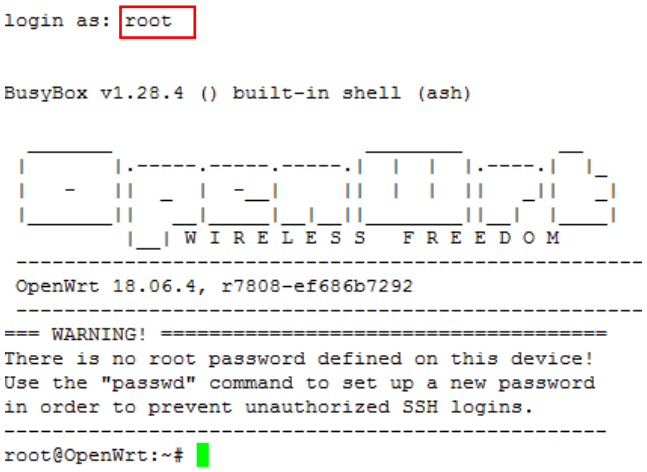


Figura 54. Acceso al enrutador OpenWRT.

Elaborado por el autor.

7. Luego, en el primer inicio de sesión, se debe establecer una contraseña para el enrutador OpenWrt como se ilustra en la Figura 55 porque no hay una contraseña predeterminada para esta imagen.

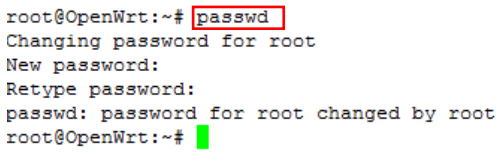


Figura 55. Creación de una contraseña para el acceso a OpenWRT.

Elaborado por el autor.

- **Instalación del Open vSwitch en OpenWRT**

1. Se procede a cambiar la dirección IP por defecto por una IP estática diferente a la del router principal para poder acceder a la interfaz gráfica del enrutador OpenWRT.

La Configuración de una dirección IP estática se la realiza a través de los siguientes comandos:

#uci set network.lan.ipaddr=192.168.1.7

#uci commit

#reboot -f

Para verificar el cambio de la dirección IP se procede a ejecutar nuevamente la herramienta *Wireless Network Watcher* como se muestra en la Figura 56.

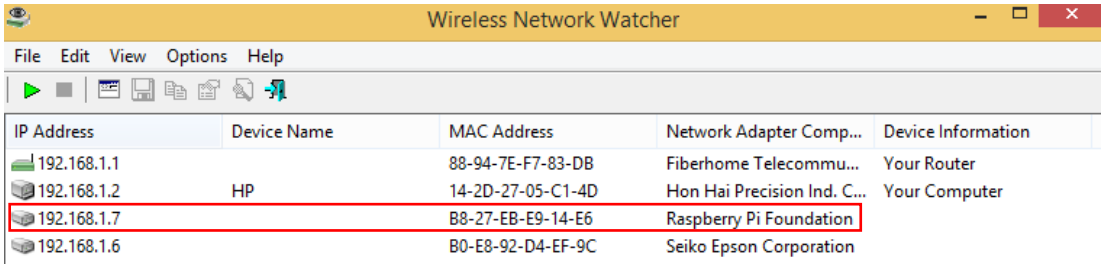


Figura 56. Detección de la nueva dirección IP de la Raspberry Pi.

Elaborado por el autor.

2. Para poder actualizar y descargar los paquetes disponibles del repositorio de paquetes OpenWrt es necesario tener conexión a internet cambiando la dirección IP estática a DHCP.

Para cambiar la dirección IP a DHCP, se inicia sesión en la página del enrutador OpenWRT como se muestra en la Figura 57.

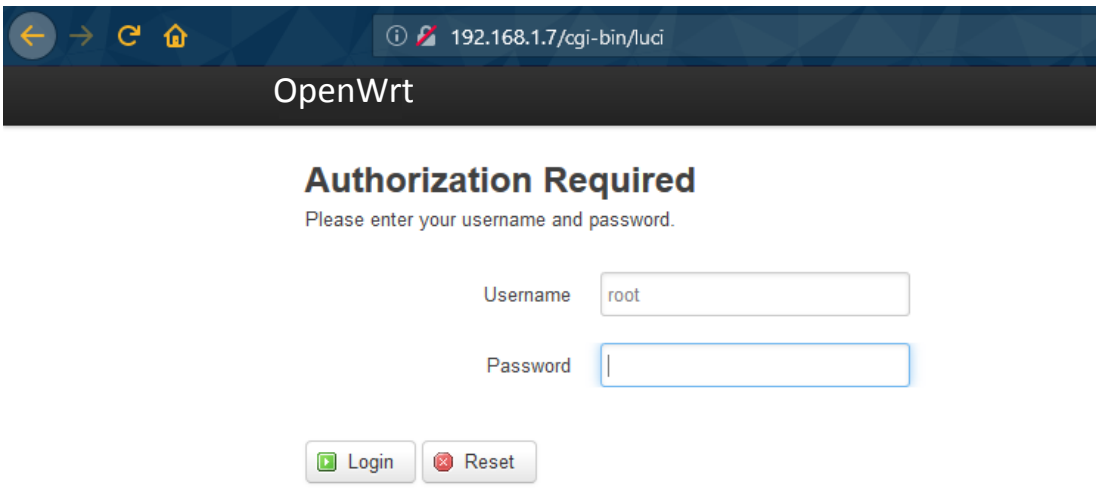


Figura 57. Interfaz gráfica LUCI del enrutador OpenWrt.

Elaborado por el autor.

3. Luego se accede a la siguiente ruta *Network >> Interfaces* y desde ahí se edita la conexión LAN.

Se cambia el protocolo *static* a *DHCP client* y se hace clic en cambiar protocolo como se muestra en la Figura 58. Una vez hecho esto, se guardan los cambios y se obtiene una nueva dirección IP proporcionada por router principal.

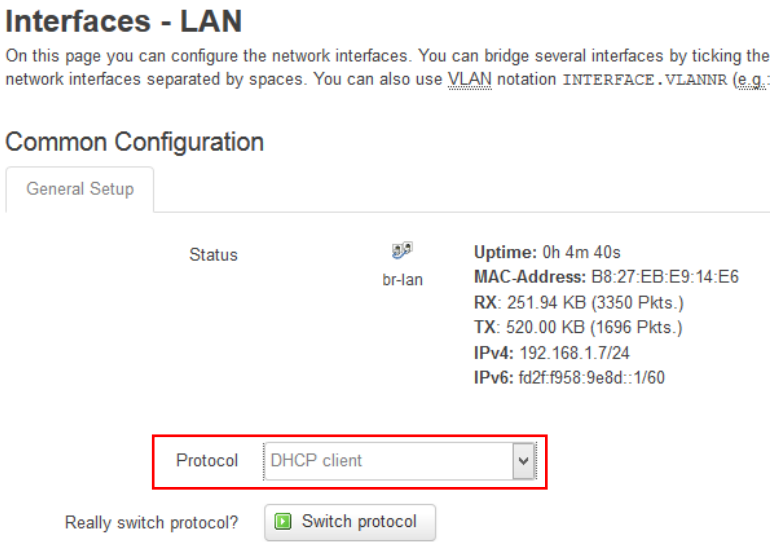


Figura 58. Configuración del protocolo *DHCP client* en el enrutador OpenWrt.

Elaborado por el autor.

4. Se procede a encontrar la nueva dirección IP usando Wireless Network Watcher Figura 59.

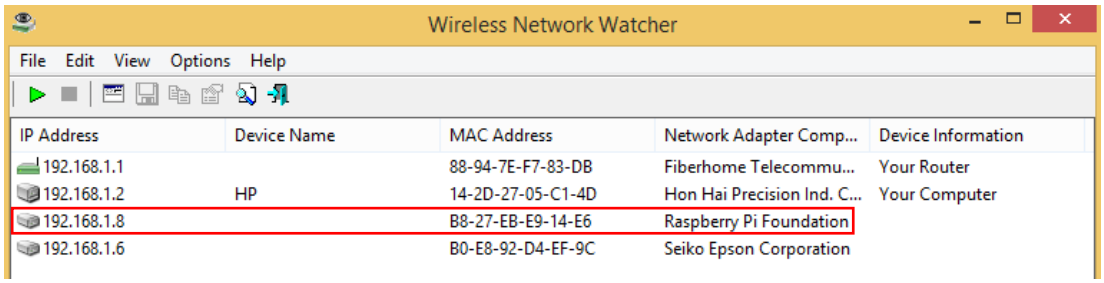


Figura 59. Dirección IP asignada por DHCP en la Raspberry Pi.

Elaborado por el autor.

5. Se accede al enrutador OpenWRT mediante el puerto SSH con la ayuda de Putty.
6. A continuación, se descargar y actualiza la lista de paquetes disponibles del repositorio de paquetes OpenWrt con el siguiente comando:

#opkg update

7. Se busca la versión de Open vSwitch Figura 60 disponible en los paquetes descargados en el paso anterior con el siguiente comando:

#opkg list | grep -e openvswitch

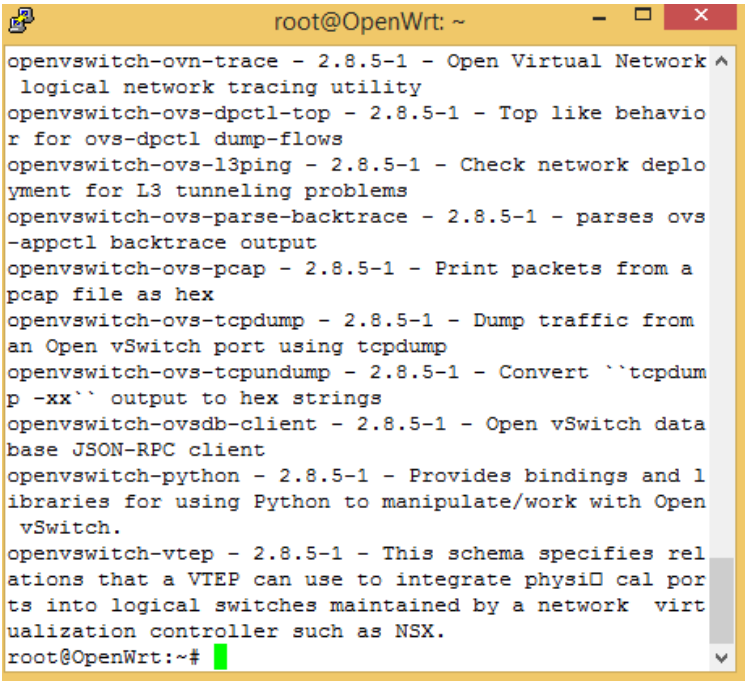


Figura 60. Paquetes Open vSwitch disponibles.

Elaborado por el autor.

En la Figura 60 se puede apreciar la última versión disponible para instalar Open vSwitch en OpenWRT en este caso es la 2.8.5.

8. Se instala Open vSwitch con el siguiente comando:

#opkg install openvswitch kmod-openvswitch

9. Para verificar la instalación se procede a ejecutar el siguiente comando para ver la versión de Open vSwitch instalada como se muestra en la Figura 61:

#ovs-vswitchd --version

```
root@OpenWrt:~# ovs-vswitchd --version
ovs-vswitchd (Open vSwitch) 2.8.5
root@OpenWrt:~#
```

Figura 61. Versión de Open vSwitch instalada en OpenWrt.

Elaborado por el autor.

- **Instalación del adaptador USB 3.0 Gigabit Ethernet en OpenWrt**

1. Se procede a descargar y actualizar la lista de paquetes disponibles del repositorio de paquetes OpenWrt con el siguiente comando:

   #opkg update

2. Se obtiene una lista de paquetes USB ya instalados Figura 62 con el siguiente comando:

   #opkg list-installed | grep usb

```
root@OpenWrt:~# opkg list-installed | grep usb
brcmfmac-firmware-usb - 2017-09-06-a61ac5cf-1
kmod-usb-core - 4.9.184-1
kmod-usb-hid - 4.9.184-1
root@OpenWrt:~#
```

Figura 62. Lista de paquetes USB instalados en OpenWrt.

Elaborado por el autor.

3. Instalar los paquetes faltantes y verificar de nuevo los paquetes instalados Figura 63 con los siguientes comandos:

#opkg install kmod-usb-storage

#opkg install kmod-usb3

#opkg list-installed | grep usb

```
root@OpenWrt:~# opkg list-installed | grep usb
brcmfmac-firmware-usb - 2017-09-06-a61ac5cf-1
kmod-usb-core - 4.9.184-1
kmod-usb-hid - 4.9.184-1
kmod-usb-storage - 4.9.184-1
kmod-usb3 - 4.9.184-1
root@OpenWrt:~#
```

Figura 63. Nueva lista de paquetes USB instalados en OpenWrt.

Elaborado por el autor.

4. Se obtiene más información de diagnóstico sobre las unidades USB conectadas, al instalar el paquete *'usbutils'* con el siguiente comando:

#opkg update && opkg install usbutils

5. Después de colocar el adaptador a la Raspberry Pi se ejecuta el siguiente comando para verificar la conexión del dispositivo como se muestra en la Figura 64:

#lsusb

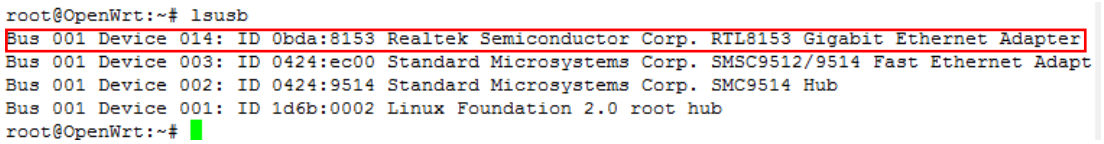


Figura 64. Comprobación de la conexión del adaptador USB Ethernet.

Elaborado por el autor.

6. Una vez identificado el dispositivo se procede a instalarlo con los siguientes comandos:

#opkg update

#opkg install kmod-usb-net-rtl8152

Se instala la versión rtl8152 ya que esta versión del driver también es compatible para rtl8153

7. A continuación, se crea la interfaz "eth1" en la Configuración de redes como se muestra en la Figura 65.

#vim /etc/config/network

```
root@OpenWrt:~# vim /etc/config/network

config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config globals 'globals'
        option ula_prefix 'fd8b:0be8:61d3::/48'

config interface 'lan'
        option type 'bridge'
        option ifname 'eth0'
        option proto 'dhcp'

config interface 'eth1'
        option ifname 'eth1'
        option proto 'static'
```

Figura 65. Configuración de la interfaz eth1 en OpenWrt.

Elaborado por el autor.

8. Para finalizar se reinicia la Raspberry y se busca que el adaptador USB Ethernet estese conectado e instalado Figura 66 a través de los siguientes comandos:

#reboot -f

#dmesg

```
3430-sdio.clm_blob failed with error -2
[    7.801939] brcmfmac mmc1:0001:1: Falling back to user helper
[    7.815024] firmware brcm!brcmfmac43430-sdio.clm_blob: firmware_loading_s
tore: map pages failed
[    7.826943] brcmfmac: brcmf_c_process_clm_blob: no clm_blob available (er
r=-11), device may have limited channels available
[    7.846007] brcmfmac: brcmf_c_preinit_dcmds: Firmware: BCM43430/1 wl0: Au
g 29 2016 20:48:16 version 7.45.41.26 (r640327) FWID 01-4527cfab
[    7.863181] r8152 1-1.4:1.0 eth1: v1.08.9
[    7.891718] usbcore: registered new interface driver brcmfmac
[    7.900431] kmodloader: done loading kernel modules from /etc/modules.d/*
```

Figura 66. Adaptador USB Ethernet instalado en OpenWrt.

Elaborado por el autor.

- **Configuración de las interfaces de red en OpenWrt**

Para Configurar las interfaces de red de la Raspberry Pi 3 modelo B se procede a abrir la interfaz gráfica "LuCI" de OpenWrt y se siguen los pasos a continuación:

Configuración de la interfaz LAN

1. Acceder a la ruta *Network* >> *Interfaces* y establecer la dirección IP estática propuesta en la Tabla 10 para el AP1 en la interfaz LAN cambiando el protocolo *DHCP client* por *Static Address* como se muestra en la Figura N°67.

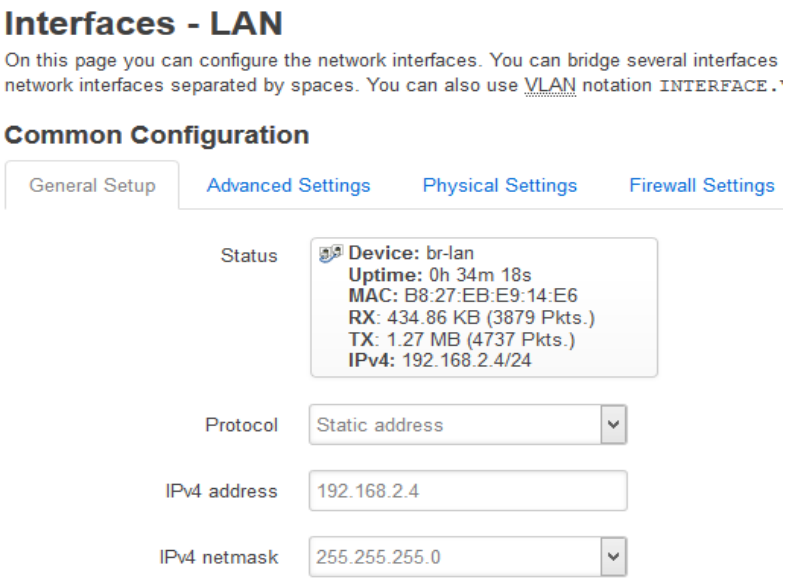


Figura 67. Configuración de IP estática para la interfaz LAN en OpenWrt.

Elaborado por el autor.

Además, es necesario deshabilitar el servidor DHCP Figura 68 puesto que en el presente proyecto se trabaja con un direccionamiento IP estático en una red de área local.

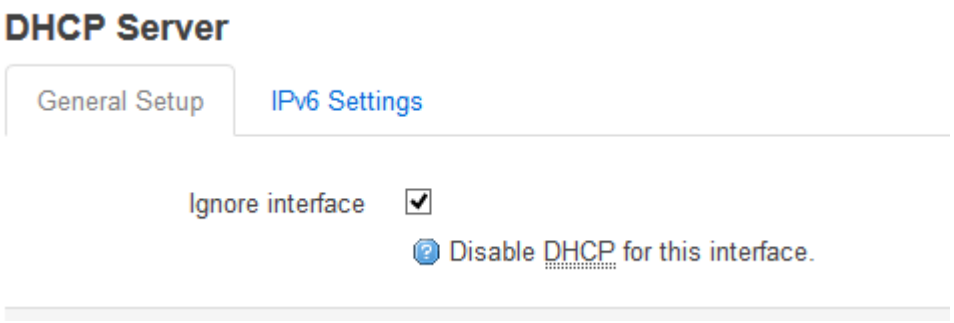


Figura 68. Desactivación del servidor DHCP en OpenWrt.

Elaborado por el autor.

2. Acceder al enrutador OpenWRT mediante el puerto SSH con la ayuda de Putty y verificar la dirección IP estática Figura 69 establecida en el paso anterior.

#ifconfig

```
root@OpenWrt:~# ifconfig
br-lan    Link encap:Ethernet  HWaddr B8:27:EB:E9:14:E6
          inet addr:192.168.2.4  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::ba27:ebff:fee9:14e6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2186 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2604 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:247617 (241.8 KiB)  TX bytes:616872 (602.4 KiB)
```

Figura 69. Verificación de la IP estática establecida para la interfaz LAN.

Elaborado por el autor.

Configuración de la interfaz WLAN0

1. Crear una nueva interfaz Figura 70 con los siguientes parámetros:

- Nombre: wifi
- Protocolo: Static address
- Interfaz por cubrir: wlan0

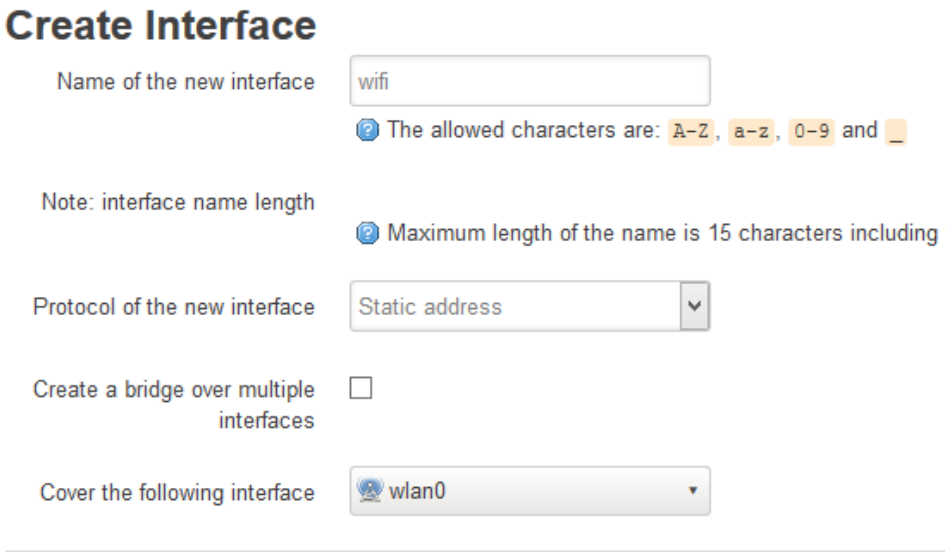


Figura 70. Creación de una interfaz inalámbrica en OpenWrt.

Elaborado por el autor.

2. Establecer el direccionamiento IP estático para la interfaz wifi creada en el paso anterior como se muestra en la Figura 71.

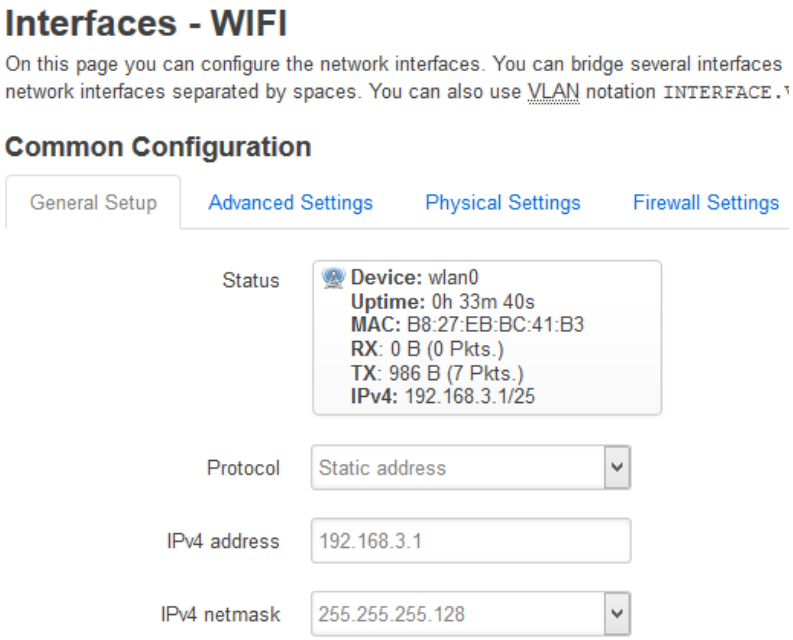


Figura 71. Configuración de IP estática para la interfaz WIFI en OpenWrt.

Elaborado por el autor.

Al igual que la interfaz LAN se deshabilita el servidor DHCP.

Configuración inalámbrica

1. Acceder a la ruta *Network >> Wireless* y añadir un punto de acceso Figura 72-74 con los siguientes parámetros:

- Modo de transmisión: Legacy
- Canal de operación: 11
- Potencia de transmisión: 20dBm (100mW)
- Modo de operación: g
- Nombre de la red: Red_AP_uno
- Interfaz de conexión a la red inalámbrica: wifi
- Protocolo de seguridad inalámbrica: WEP
- Contraseña del protocolo de seguridad: ab12345678

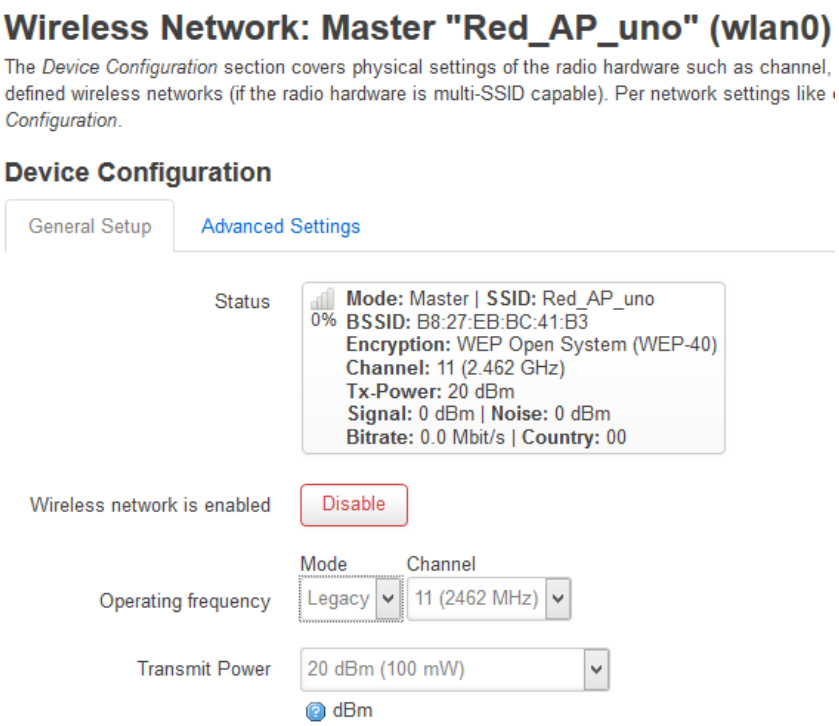


Figura 72. Configuración general del AP1.

Elaborado por el autor.

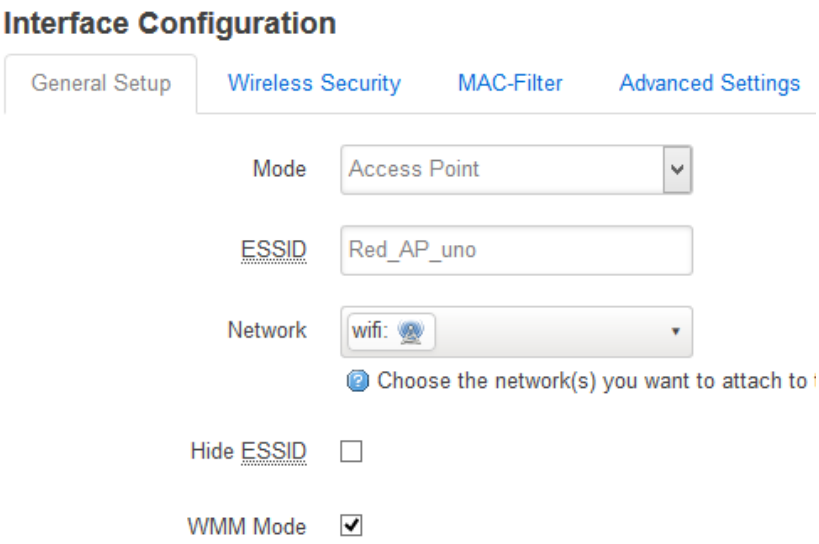


Figura 73. Configuración general de la interfaz inalámbrica del AP1.

Elaborado por el autor.

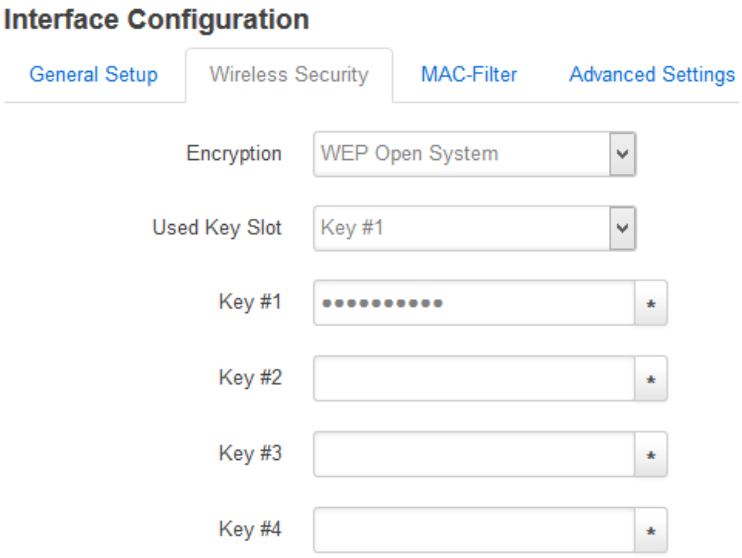


Figura 74. Configuración de la seguridad inalámbrica del AP1.

Elaborado por el autor.

2. Acceder a la ruta *Network >> Firewall* y añadir una zona sobre la interfaz inalámbrica creada para controlar el flujo de tráfico en la red Figura 75.

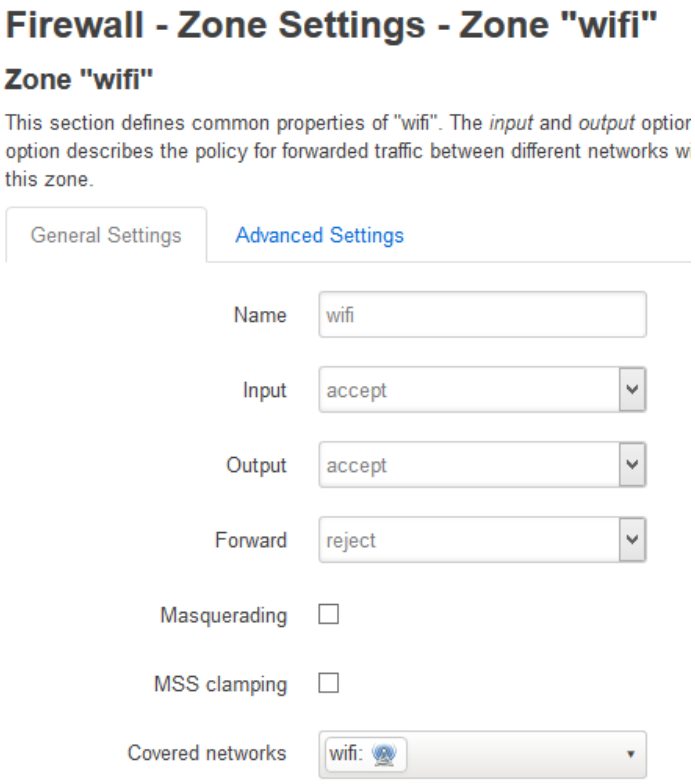


Figura 75. Configuración de una zona firewall sobre la interfaz WIFI.

Elaborado por el autor.

3. Verificar la zona firewall creada y sus parámetros Figura 76.

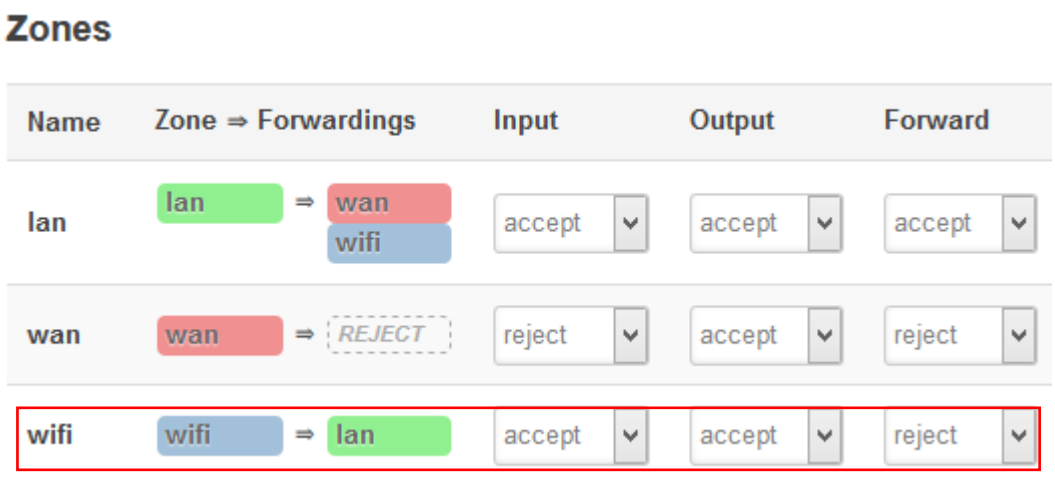


Figura 76. Verificación de la zona firewall creada.

Elaborado por el autor.

- **Configuración del Open vSwitch en OpenWrt**

1. Crear un archivo bash con el siguiente comando

#vi AP1_start.sh

2. A continuación, se describen las partes fundamentales del código desarrollado [50], el código completo para la Configuración de los dos Access Point se lo puede revisar en el **ANEXO D**.

```
#!/bin/sh
```

Declaración de variables como: dirección IP del AP1, dirección IP del controlador, identificador del SW virtual, puente br0, puertos habilitados para el OvS y variables de Configuración del OvS.

```
MYIP=192.168.2.4
CTLIP=192.168.2.2
DPID=0000000000000002
SW=br0
DPPORTS="wlan0 eth1"
VSCTL="ovs-vsctl --db=tcp:$MYIP:9999"
OVSDB=/tmp/ovs-vswitchd.conf.db
```

Eliminación de los servidores, archivos de bloqueo obsoletos y la base de datos OVS.

```
rm /tmp/.ovs-vswitchd.conf.db.~lock~
rm -f $OVSDB
```

Creación de la base de datos OVS.

```
ovsdb-tool create $OVSDB /usr/share/openvswitch/vswitch.ovsschema
ovsdb-server $OVSDB --remote=ptcp:9999:$MYIP &
```

Inicio de la base de datos OVS.

```
sleep 5
ovs-vswitchd tcp:$MYIP:9999 --pidfile=ovs-vswitchd.pid --overwrite-
pidfile -- &
```

Crear el puente br0 y pasar algunas opciones de Configuración.

```
$VSCTL add-br $SW
$VSCTL set bridge $SW protocols=OpenFlow13
```

Bucle para agregar los puertos al conmutador virtual.

```
for i in $DPPORTS ; do
    PORT=$i
        ifconfig $PORT up
    $VSCTL add-port $SW $PORT
done
```

Asignación del ID datapath, controlador OpenFlow a través del puerto 6633 y desactivación del modo a prueba de fallos .

```
$VSCTL set bridge $SW other-config:datapath-id=$DPID
$VSCTL set-controller $SW tcp:$CTLIP:6633
$VSCTL set-fail-mode br0 secure
```

3. Ejecutar automáticamente el script creado en el arranque de la Raspberry Pi con OpenWrt.

Darle permisos al script creado y al archivo: */etc/rc.local.*

#chmod +x AP1_start.sh

#chmod +x /etc/rc.local

Agregar el script creado en el archivo rc.local Figura 77.

#vi /etc/rc.local

```
# Put your custom commands here that should be executed onc
# the system init finished. By default this file does nothi
sh /root/AP1_start.sh &
exit 0
~
```

Figura 77. Archivo de Configuración para el inicio de archivos en el arranque de OpenWrt.

Elaborado por el autor.

4. Reiniciar la Raspberry Pi y con el siguiente comando verificar la Configuración del Open vSwitch Figura 78.

 # ovs-vsctl --db=tcp:192.168.2.4:9999 show

```
307b2e10-95e6-4d27-a2f4-9e6830bda1c2
    Bridge "br0"
        Controller "tcp:192.168.2.2:6633"
        fail_mode: secure
        Port "wlan0"
            Interface "wlan0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth1"
            Interface "eth1"
root@OpenWrt:~#
```

Figura 78. *Configuración* del Open vSwitch en OpenWrt.

Elaborado por el autor.

A continuación, se muestra en la Figura 79 el funcionamiento del script *AP1_start.sh* a través de un diagrama de flujo.

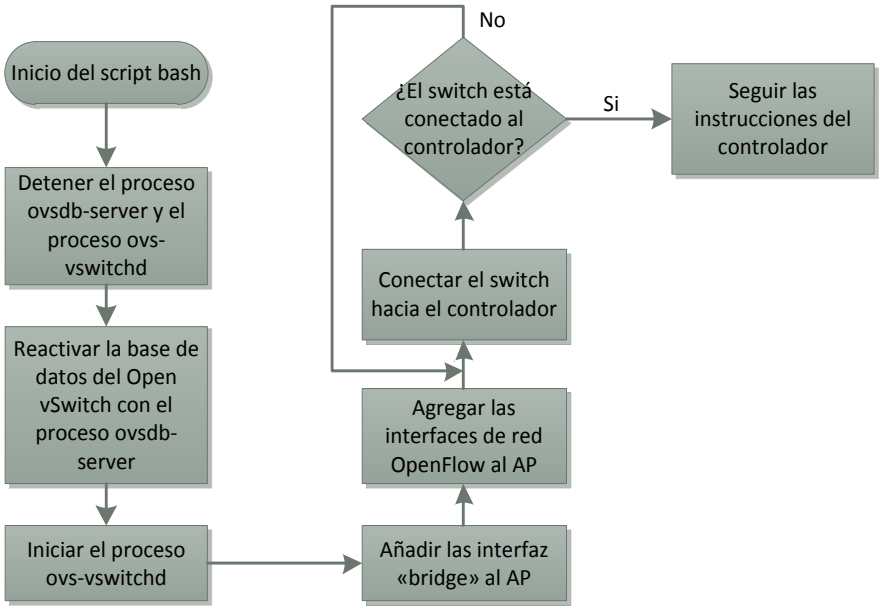


Figura 79. Diagrama de flujo del script AP1_start.sh.

Elaborado por el autor.

Para la implementación del segundo Access Point se siguen los mismos pasos del AP_1, teniendo en cuenta los criterios de diseño establecidos en la sección **3.1.1.2.**

3.1.1.4.4 Implementación de la aplicación SDN

- **Requerimientos para el desarrollo de la aplicación SDN**

La aplicación por desarrollar es una aplicación de tipo firewall que permite controlar el tráfico de los flujos de datos y el acceso a la red de cada uno de los dispositivos conectados a la misma.

En la Figura 80 se muestra un diagrama de flujo que describe el funcionamiento que tendrá la aplicación a implementar.

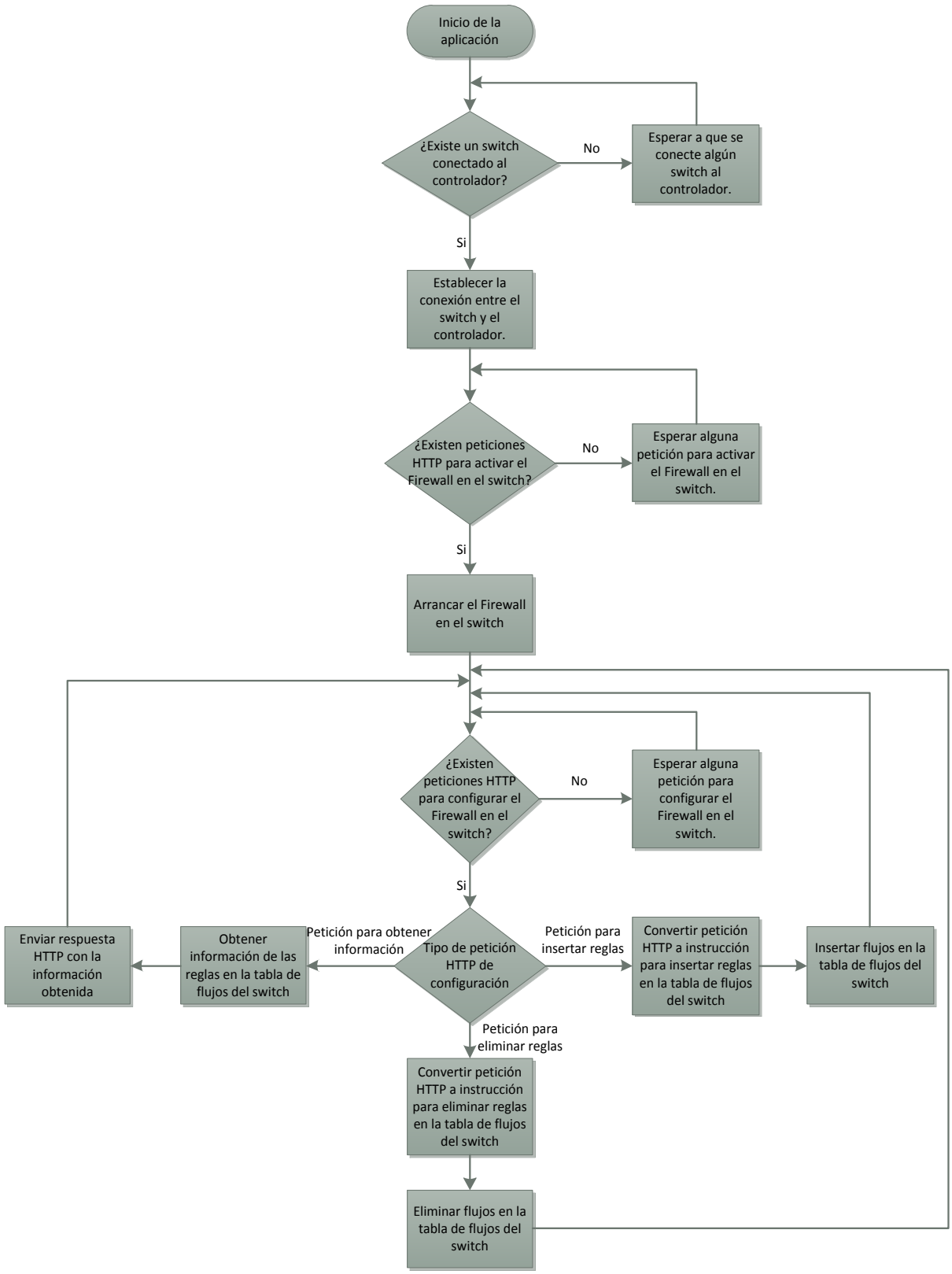


Figura 80. Diagrama de flujo de la aplicación firewall SDN.

Elaborado por el autor.

Los requerimientos para implementar la aplicación firewall son:

a. Compatible con el protocolo OpenFlow en su versión 1.3
b. Hacer uso de la aplicación REST API para dar respuesta a las peticiones HTTP en la ejecución de la aplicación.
c. Las peticiones HTTP deberán permitir la inserción, visualización o eliminación de reglas de tráfico de datos además de la activación o desactivación general de la aplicación.
d. La inserción de reglas para el control de tráfico deberá basarse tanto en las direcciones IP o en las direcciones MAC fuente y destino como en el número de los puertos de red.
e. Las reglas insertadas permiten hacer el control a partir de los protocolos de red ICMP y TCP.
f. La aplicación firewall deberá permitir o denegar el tráfico de datos en la red.

- **Desarrollo de la aplicación Firewall SDN**

Para el desarrollo de la aplicación Firewall se ha tomado en cuenta usar el controlador Ryu que utiliza Python como lenguaje de programación de alto nivel.

En la página oficial del controlador Ryu [25] se puede encontrar la documentación y tutoriales que facilitan la creación de nuevas aplicaciones para la gestión y control de redes definidas por software. Adicionalmente el Framework de Ryu proporciona ejemplos de aplicaciones creadas para que los desarrolladores de SDN puedan comprender de mejor manera el uso de estas. La aplicación ***rest_firewall.py*** disponible en el repositorio de GitHub [51] ha sido de gran ayuda y la base principal para la creación de la aplicación firewall del presente proyecto de titulación.

La aplicación desarrollada cuenta con seis clases transcritas en Python y en un único script con el nombre de ***appFirewall.py***.

1. RestFirewallAPI
2. ListaFirewalls
3. ControladorFirewall
4. Firewall
5. Match

6. Accion

A continuación, se describen las partes más importantes de cada una de las clases mencionadas. El código fuente completo se encuentra en el **ANEXO E**

1. Clase RestFirewallAPI

- **Descripción de la clase**

En esta clase se construye la API REST de la aplicación firewall, se establece la conexión de las peticiones HTTP con las instrucciones programadas en el controlador, es la encargada de enlazar el resto de las clases y es la primera en ser instanciada en todo el programa.

- **Código**

En el siguiente código de programación se establece la declaración de la clase, la versión del protocolo OpenFlow a utilizar y se declaran las variables del método constructor.

```
class RestFirewallAPI(app_manager.RyuApp):

    VERSIONES_OF = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'dpset': dpset.DPSet,'wsgi': WSGIApplication}
```

Método constructor “init” donde se definen los modulos "wsgi" y "dpset" a utilizarse en el resto de la aplicación, además de las propiedades iniciales de las instanciaciones para la clase.

```
    def __init__(self, *args, **kwargs):
        super(RestFirewallAPI, self).__init__(*args, **kwargs)
        self.dpset = kwargs['dpset']
        wsgi = kwargs['wsgi']
        self.waiters = {}
        self.data = {}
        self.data['dpset'] = self.dpset
        self.data['waiters'] = self.waiters
        wsgi.registory['ControladorFirewall'] = self.data
        ruta = '/firewall'
        id_dp = {'switchid': dpid_lib.DPID_PATTERN + r'|TODOS'}
```

El código a continuación crea varios URI (identificador de recursos uniforme) y establece conexión de las peticiones HTTP con la aplicación para el establecimiento de las siguientes funciones:

a. Consultar el estado de los switches conectados a la red.

```
uri = ruta + '/module/status'

wsgi.mapper.connect('firewall',
uri,controller=ControladorFirewall,action='estado_sw',
conditions=dict(method=['GET']))
```

b. Habilitar el firewall sobre los switches conectados a la red a través del identificador datapath establecido para cada uno de ellos.

```
uri = ruta + '/module/enable/{switchid}'

wsgi.mapper.connect('firewall',
uri,controller=ControladorFirewall,action='habilitar_sw',
conditions=dict(method=['PUT']),requirements=id_dp)
```

c. Deshabilitar el firewall sobre los switches conectados a la red a través del identificador datapath establecido para cada uno de ellos.

```
uri = ruta + '/module/disable/{switchid}'

wsgi.mapper.connect('firewall',
uri,controller=ControladorFirewall,action='deshabilitar_sw',
conditions=dict(method=['PUT']),requirements=id_dp)
```

d. Crear, consultar y eliminar las reglas de reenvío de tráfico de un switch a partir de su id.

```
uri = ruta + '/reglas/{switchid}'

wsgi.mapper.connect('firewall',
uri,controller=ControladorFirewall,action='reglas_sw',
conditions=dict(method=['GET']),requirements=id_dp)

wsgi.mapper.connect('firewall',
uri,controller=ControladorFirewall,action='ingresar_regla',
conditions=dict(method=['POST']),requirements=id_dp)

wsgi.mapper.connect('firewall',
uri,controller=ControladorFirewall,action='borrar_regla',
conditions=dict(method=['DELETE']),requirements=id_dp)
```

Finalmente se define un método para controlar los eventos de conexión y desconexión de un switch además de instanciarlos a la clase ControladorFirewall.

```
    @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
    def handler_datapath(self, ev):
        if ev.enter:
            ControladorFirewall.registrar_sw(ev.dp)
        else:
            ControladorFirewall.desregistrar_sw(ev.dp)
```

2. Clase ListaFirewalls

- ***Descripción de la clase***

La clase ListaFirewalls va a contener los switches conectados al controlador y a la aplicación firewall mediante el uso de un diccionario, además la clase contiene un método denominado "*lista_sw*" el cual crea una instancia a partir del datapath y del id de los switches conectados para ser retornada como una cadena de texto.

- ***Código***

```
class ListaFirewalls(dict):

    def __init__(self):
        super(ListaFirewalls, self).__init__()

    def lista_sw(self, dp_id):
        if len(self) == 0:
            raise ValueError('switch no conectado.')
        dps = {}

        if dp_id == 'TODOS':
            dps = self
        else:
            try:
                dpid = dpid_lib.str_to_dpid(dp_id)
            except:
                raise ValueError('ID del switch invalido.')
            if dpid in self:
                dps = {dpid: self[dpid]}
            else:
                msg = 'firewall sw no conectado. :
switchID=%s'%dp_id
                raise ValueError(msg)
        return dps
```

3. Clase ControladorFirewall

- **Descripción de la clase**

Esta clase es la encargada de manejar dos tipos de métodos: los métodos estáticos para la conexión o desconexión de los switches y los métodos llamados por la API REST. Los métodos estáticos para la conexión o desconexión de los switches permiten Configurar las instancias creadas en el diccionario de switches conectados a partir del datapath, mientras que, los métodos invocados por la API REST son los encargados de representar las reglas del firewall a partir de las tablas de flujo de cada switch. En la Tabla 14 se pueden apreciar los métodos de la clase con su respectiva función.

Tabla 14. Métodos declarados de la clase ControladorFirewall.

Métodos estáticos para la conexión o desconexión de los switches	
Nombre del método	**Función que desempeña**
registrar_sw	Registra los switches a través del identificador datapath
desregistrar_sw	Elimina los switches a través del identificador datapath
Métodos llamados por la API REST	
Nombre del método	**Función que desempeña**
estado_sw	Consulta el estado de los switches conectados a la red.
habilitar_sw	Habilita el firewall sobre los switches conectados a la red a través del identificador datapath establecido para cada uno de ellos.
deshabilitar_sw	Deshabilita el firewall sobre los switches conectados a la red a través del identificador datapath establecido para cada uno de ellos.
reglas_sw	Consulta las reglas de reenvío de tráfico de un switch a partir de su id
ingresar_regla	Crea las reglas de reenvío de tráfico de un switch a partir de su id

borrar_regla	Elimina las reglas de reenvío de tráfico de un switch a partir de su id

Elaborado por el autor

- **Código**

El siguiente código de programación establece la declaración de la clase, definición de las variables utilizadas y el constructor.

```
class ControladorFirewall(ControllerBase):
    _LISTA_SW = ListaFirewalls()
    def __init__(self, req, link, data, **config):
        super(ControladorFirewall,self).__init__(req,link,data,
              **config)
        self.dpset = data['dpset']
        self.waiters = data['waiters']
```

4. Clase Firewall

- **Descripción de la clase**

Es la clase encargada de manejar los flujos en los switches, además para mantener una abstracción clara del estado de los flujos se realizan instanciaciones cuando la clase es invocada por la interfaz API REST.

- **Código**

Lo primero que se realiza es la declaración de la clase Firewall, definición de las variables utilizadas, selección del protocolo OpenFlow versión 1.3 y creación de la excepción "*raise*" para cuando se el controlador Ryu detecte un switch que no es compatible con el protocolo seleccionado.

```
class Firewall(object):
    _OFCTL = {ofproto_v1_3.OFP_VERSION: ofctl_v1_3}
    def __init__(self, dp):
        super(Firewall, self).__init__()
        self.dp = dp
        version = dp.ofproto.OFP_VERSION
```

```
        self.cookie = 0

        if version not in self._OFCTL:
            raise OFPUnknownVersion(version=version)
        self.ofctl = self._OFCTL[version]
```

Se programa el método "*activar_flujo*" para activar el envío de flujos en cada switch dependiendo del identificador datapath y de las variables establecidas.

```
    def activar_flujo(self):
        cookie = 0
        prioridad = PRIORIDAD_ESTADO_FLUJO
        match = {}
        actions = []
        flow = self._conv_a_flujo(cookie=cookie, priority=prioridad,
                                  match=match, actions=actions)
        cmd = self.dp.ofproto.OFPFC_DELETE_STRICT
        self.ofctl.mod_flow_entry(self.dp, flow, cmd)
        msg = {'resultado': 'correcto',
               'detalles': 'firewall ejecutandose.'}
        switch_id='%s:%s'%('switch_id',
                           dpid_lib.dpid_to_str(self.dp.id))
        return {switch_id: msg}
```

En el método "*ingresar_regla*" permite verificar que la regla ingresada sea correcta, de no ser el caso se programa una excepción raise de tipo "*ValueError*" que indica que la regla ingresada es invalida.

```
    def ingresar_regla(self, cookie, rest):
        prioridad = int(rest.get('prioridad', 0))
        if prioridad < 0 or PRIORIDAD_FLUJO_ACL_MAX < prioridad:
            raise ValueError('Valor no valido de prioridad Ingrese
                              de[0-%d]'% PRIORIDAD_FLUJO_ACL_MAX)
        match = Match.conv_a_accion_of(rest)
        actions = Accion.conv_a_accion_of(self.dp, rest)
        flow = self._conv_a_flujo(cookie=cookie, priority=prioridad,
                                  match=match, actions=actions)
        cmd = self.dp.ofproto.OFPFC_ADD
        try:
            self.ofctl.mod_flow_entry(self.dp, flow, cmd)
        except:
            raise ValueError('Parametro de regla invalido.')
        msg = {'resultado': 'correcto',
               'detalles': 'Regla ingresada.: regla_id=%d' % cookie}
        switch_id = '%s: %s' % ('switch_id',
                                dpid_lib.dpid_to_str(self.dp.id))
        return {switch_id: msg}
```

5. Clase Match

- **Descripción de la clase**

Esta clase permite convertir las reglas provenientes de la interfaz API REST en reglas de flujo OpenFlow y viceversa dentro de cada switch, por tal motivo esta clase es denominada clase de emparejamiento.

- **Código**

En el siguiente código de programación se declara la clase Match y se establece un diccionario con los protocolos de red para establecer las reglas de reenvío de tráfico.

```
class Match(object):

    _CONVERT = {'dl_type':
                {'ARP': ether.ETH_TYPE_ARP,
                 'IPv4': ether.ETH_TYPE_IP},
                'nw_proto':
                {'TCP': inet.IPPROTO_TCP,
                 'UDP': inet.IPPROTO_UDP,
                 'ICMP': inet.IPPROTO_ICMP}}
```

A continuación, se establece un método estático "*conv_a_accion_of*" que permite convertir una regla de la interfaz REST API en una regla OpenFlow considerando los protocolos del diccionario establecido.

```
    @staticmethod
    def conv_a_accion_of(rest):
        match = {}
        set_dltype_flg = False
        for key, value in rest.items():
            if (key == 'nw_src' or key=='nw_dst'or key=='nw_proto'):
                if ('dl_type' in rest) is False:
                    set_dltype_flg = True
                elif(rest['dl_type']!='IPv4'and
                     rest['dl_type']!='ARP'):
                    continue
            elif key == 'tp_src' or key == 'tp_dst':
                if (('nw_proto' in rest) is False
                    or (rest['nw_proto'] != 'TCP'
                        and rest['nw_proto'] != 'UDP')):
                    continue
            if key in Match._CONVERT:
                if value in Match._CONVERT[key]:
                    match.setdefault(key,
                                     Match._CONVERT[key][value])
                else:
                    raise ValueError('Parametro de regla invalido.:
                                     key=%s'%key)
```

```
            else:
                match.setdefault(key, value)
            if set_dltype_flg:
                match.setdefault('dl_type', ether.ETH_TYPE_IP)
        return match
```

El método estático "*conv_a_comando_rest*" permite convertir una regla OpenFlow en una regla de la interfaz REST API considerando los protocolos del diccionario establecido.

```
    @staticmethod
    def conv_a_comando_rest(openflow):
        of_match = openflow['match']
        match = {}
        for key, value in of_match.items():
            if key == 'dl_src' or key == 'dl_dst':
                if value == mac.haddr_to_str(mac.DONTCARE):
                    continue
            elif key == 'nw_src' or key == 'nw_dst':
                if value == '0.0.0.0':
                    continue
            elif value == 0:
                continue
            if key in Match._CONVERT:
                conv = Match._CONVERT[key]
                conv = dict((value,key) for key,value in
                            conv.items())
                match.setdefault(key, conv[value])
            else:
                match.setdefault(key, value)
        return match
```

6. Clase Accion

- **Descripción de la clase**

La clase acción permite convertir las acciones "*aceptar*, *rechazar*" de la REST API en acciones OpenFlow y viceversa.

- **Código**

Declaración de la clase "Accion" y creación del método estático "*conv_a_accion_of*" que convierte acciones de la REST API en acciones OpenFlow.

```
class Accion(object):

    @staticmethod
    def conv_a_accion_of(dp, rest):
        value = rest.get('actions', 'ACEPTAR')
        if value == 'ACEPTAR':
```

```
            out_port = dp.ofproto.OFPP_NORMAL
            action = [{'type': 'OUTPUT',
                       'port': out_port}]
        elif value == 'RECHAZAR':
            action = []
        else:
            raise ValueError('Tipo de accion invalida.')
        return action
```

Creación del método estático "*conv_a_comando_rest*" que convierte acciones de la REST API en acciones OpenFlow.

```
    @staticmethod
    def conv_a_comando_rest(openflow):
        if 'actions' in openflow:
            if len(openflow['actions']) > 0:
                action = {'actions': 'ACEPTAR'}
            else:
                action = {'actions': 'RECHAZAR'}
        else:
            action = {'actions': 'Unknown action type.'}


        return action
```

- **Ejecución de la aplicación Firewall SDN**

Para la ejecución de la aplicación Firewall creada se siguen los pasos mostrados a continuación.

1. Conectar el Switch OpenFlow, los Access Point OpenFlow y el controlador RYU al router TP-LINK tradicional.
2. Pasar el script appFirewall.py al escritorio de la PC donde se instaló el controlador.
3. Abrir el terminal de Ubuntu y ejecutar el siguiente comando.

$ryu-manager appFirewall.py

El controlador empezará a ejecutarse al igual que la aplicación Firewall creada y a continuación deberán aparecer los componentes de la red conectados como se muestra en la Figura 81.

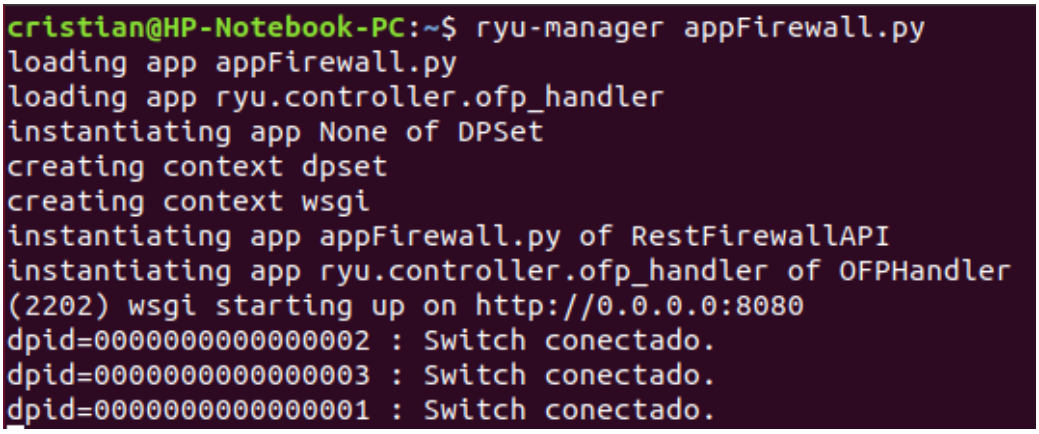


Figura 81. Ejecución de la aplicación *appFirewall.py*.

Elaborado por el autor.

3.1.1.4.5 Implementación de la interfaz gráfica para el control de la aplicación Firewall del controlador RYU

Para la creación de una interfaz gráfica que permita Configurar la aplicación firewall desarrollada en la sección **3.1.1.4.4** se ha optado por utilizar el diseñador de interfaces Glade junto con la librería GTK +3 en Linux a continuación se describen los pasos de instalación.

Instalación de Glade y GTK+3 en Ubuntu 18.04 LTS

1. Para instalar Glade se ingresa la siguiente línea de comando en una ventana de terminal:

$sudo apt-get install glade

2. Los archivos de la librería de desarrollo GTK deben instalarse para poder desarrollar y compilar aplicaciones GTK. Se instalan los archivos de la librería GTK + 3 usando el siguiente comando:

$sudo apt-get install libgtk-3-dev

3. Después de instalar GTK 3 y las herramientas de desarrollo de Glade es necesario la instalación de un editor de texto para la Configuración del archivo XML generado por Glade ya que este desarrollador de interfaces no genera código fuente como tal. Con el siguiente comando se procede a instalar el editor de texto Geany:

$sudo apt-get install geany

Creación de la interfaz gráfica

La interfaz gráfica es desarrollada de forma visual a través de Glade el cual crea un archivo XML con los elementos de la interfaz diseñada, el script del archivo ***gui.xml*** creado se lo encuentra en el **ANEXO F**. Una vez creada la interfaz gráfica se procede a darle las instrucciones a cada uno de sus elementos, esto se lo realiza con la creación de un archivo Python y con la ayuda de la librería GTK+3. En la Figura 82 se puede aprecia la interfaz gráfica creada en Glade.

Figura 82. Interfaz gráfica de control de la aplicación firewall creada en Glade.
Elaborado por el autor.

Para el presente proyecto se ha desarrollado un script con el nombre ***interfaz.py,*** el código fuente se encuentra disponible en el **ANEXO G**, este script es el encargado de controlar las acciones de los elementos de la interfaz gráfica creada en Glade y además configura la aplicación firewall del controlador RYU para la inserción, eliminación y visualización de reglas.

Descripción de la sección para el ingreso de reglas

En esta sección se encuentra la opción de actualizar la red mediante el uso del botón *"Refresh"* con la finalidad de detectar los dispositivos conectados al controlador,

además, también se tiene la opción de activar o desactivar todo el tráfico de red a través de un interruptor. Luego se dispone de los parámetros de red necesarios para el ingreso de reglas Firewall:

- Identificador del switch
- Tipo de protocolo: ICMP, TCP
- Acción: ACEPTAR, RECHAZAR
- Dirección IP fuente
- Dirección IP destino
- Dirección MAC fuente
- Dirección MAC destino
- Puerto destino
- Puerto fuente

Cabe destacar que el ingreso de las reglas para el control de tráfico en el controlador se lo puede hacer a través de las direcciones IP o a través de las direcciones MAC de los dispositivos conectados, pero no se lo realiza simultáneamente ya que existiría redundancia y errores en la aplicación. Finalmente, para el ingreso de la regla se dispone del botón *"Ingresar Regla"* y adicionalmente para eliminar cualquier regla ingresada se dispone del botón *"Borrar Regla"* Figura 83.

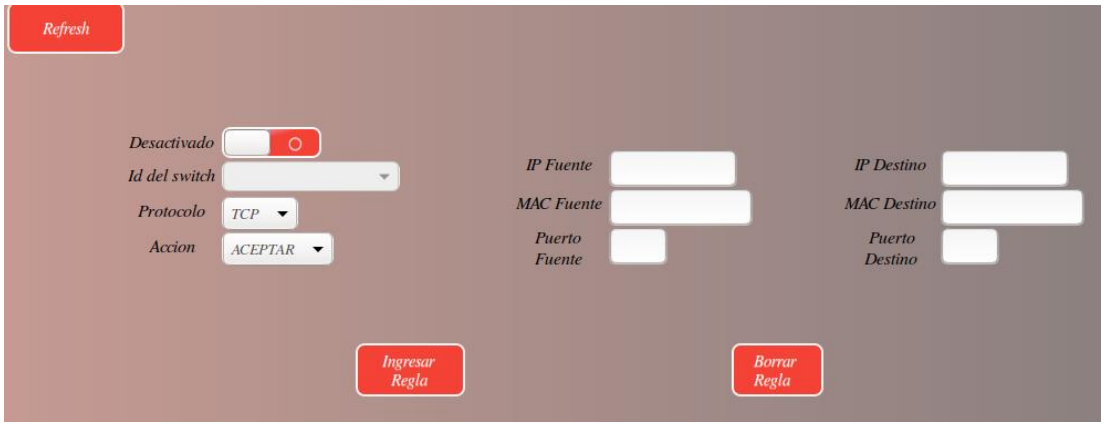


Figura 83. Sección de la interfaz gráfica para el ingreso de reglas.

Elaborado por el autor.

Descripción de la sección para la visualización de reglas

Para la visualizacion de las reglas se dispone de una tabla con scroll Figura 84, en esta su pueden visualizar las reglas ingresadas con sus respectivos parametros y ademas las

reglas disponen de un orden de ingreso que es mostrado en el campo "*Id Regla"*. Los campos que no dispongan de parametros se representan con el simbolo "*".

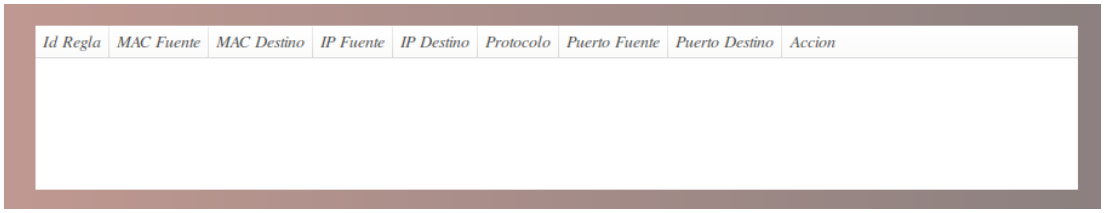


Figura 84. Sección de la interfaz gráfica para la visualización de reglas.
Elaborado por el autor.

3.1.1.5 Implementación del servidor IoT

Para la implementación del servidor IoT se utilizan los softwares Mosquitto y Node RED que trabajan bajo el protocolo MQTT.

Instalación de Mosquitto

Mosquitto es un message Broker open source que trabaja con el protocolo MQTT y es utilizado como mensajería ligera para proyectos con dispositivos IoT como Arduino. A continuación, se describen los pasos de instalación.

1. Instalar mosquitto-1.4.9-install-win32
2. Instalar Win32OpenSSL-1_1_0
3. Ubicar el directorio donde se instaló OpenSSL en C:\, copiar la carpeta "*bin"* y pegarla en el directorio donde se instaló Mosquitto: C:\Program Files (x86).
4. Pegar las librerias "*dll"* en la misma carpeta.
5. Volver a ejecutar el instalador de mosquitto-1.4.9-install-win32
6. Ir a servicios de Windows: ubicar el servicio de mosquitto e iniciarlo
7. Para comprobar la instalación hay que iniciar el cmd y escribir el comando *"netstat -an"* y verificar que esté escuchando el puerto 1883 de mosquitto.
8. Verificar el firewall de Windows, en caso de estar bloqueado el puerto 1883, agregarlo a la lista.

Instalación de Node RED

Node RED es una herramienta de programación para conectar dispositivos de hardware, API y servicios en línea de formas nuevas e interesantes. Proporciona un editor basado en un navegador que facilita la conexión de flujos utilizando la amplia gama de nodos en la paleta, que se pueden implementar en su tiempo de ejecución con un solo clic. A continuación, se describen los pasos para la instalación.

1. Ir al sitio web oficial de Node.js y descargar el instalador: https://nodejs.org/en/download/
2. Ejecutar el instalador como administrador.
3. Ingresar al cmd como administrador y verificar la versión con el comando: *node -v*
4. Se obtiene la respuesta del paso anterior. Ahora se descarga Node RED ejecutando el siguiente comando: *npm install -g node-red*
5. Se ignoran los mensajes de error y advertencia luego se verifica la descarga mediante el comando: *node-red,* una vez ingresado el comando deberá aparecer el mensaje *"welcome to node-red".*
6. Finalmente se instala la dashboard de Node-RED con el comando: *npm install node-red-dashboard*

Una vez realizada la instalación de los 2 programas se procede a programar en el navegador de Node-RED, al tener dos dispositivos IoT se considera programar en flojos diferentes para que no se interfieran los datos enviados y recibidos por las NodeMCUs en la Figura 85 se muestra el flujo 1 con la programación para el proceso uno, mientras que en la Figura 86 se muestra el flujo 1 con la programación del proceso dos.

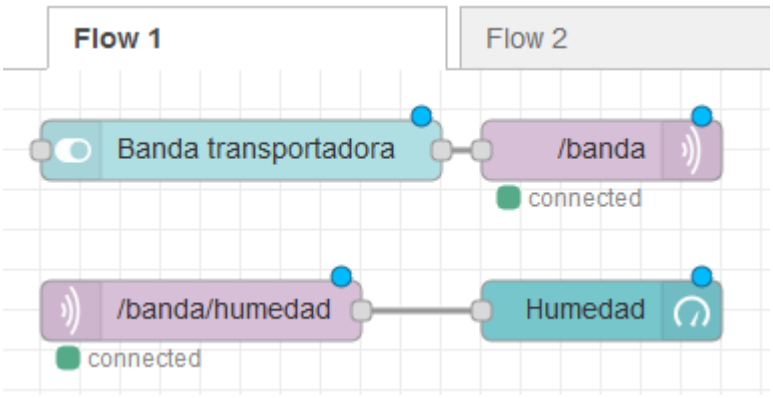


Figura 85. Programación de del Flow 1 en Node-RED.

Elaborado por el autor.

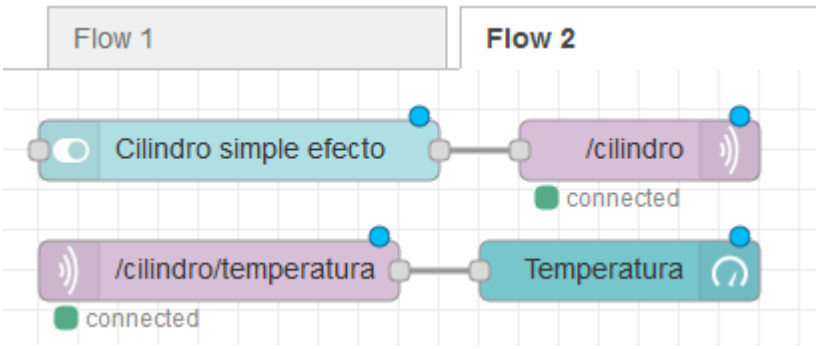


Figura 86. Programación de del Flow 2 en Node-RED.

Elaborado por el autor.

3.1.1.5.1 Implementación de la interfaz gráfica de control y monitorización de los procesos de la industria 4.0.

Para la creación de la interfaz gráfica es necesario implementar un servidor web que permita unir los diferentes dominios de cada dispositivo IoT en una sola página web.

Se procede a instalar XAMPP, que es un paquete de software libre, cuya función consiste en el sistema de gestión de bases de datos MySQL, servidor web Apache y los intérpretes para lenguajes de script PHP y Perl. Con la implementación de XAMPP se procede a ejecutar el servicio de Apache como se muestra en la Figura 87.

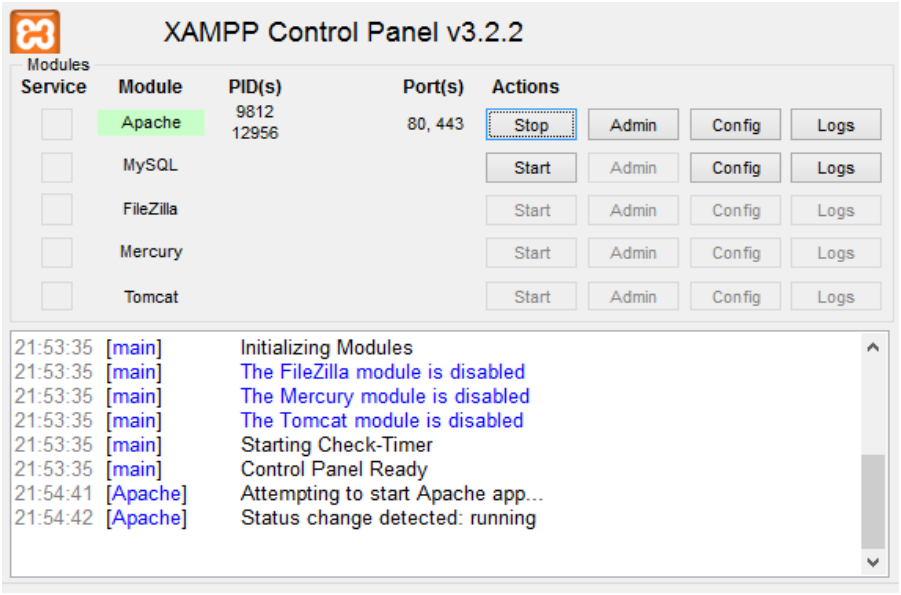


Figura 87. Ejecución del servicio de Apache en XAMPP.

Elaborado por el autor.

Una vez levantado el servicio lo que se hace es dirigirse a la ruta *C:\xampp\htdocs* de la PC y se copia la carpeta Servidor-IoT que contiene los archivos mostrados en la Figura 88.

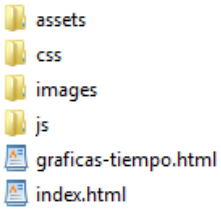


Figura 88. Archivos de Configuración para la página web con la interfaz gráfica de control y monitorización.

Elaborado por el autor.

En el **ANEXO H** se puede encontrar el código fuente de los archivos correspondientes a la página web creada que se muestra en las Figura 89, 90.

Figura 89. Página web de inicio.

Elaborado por el autor.

Figura 90. Página web con la interfaz gráfica de control y monitorización.

Elaborado por el autor.

- **Descripción de la sección de control**

En la sección de control Figura 91 se tiene la adquisición de valores aleatorios enviados desde los dispositivos IoT para que simulen datos de humedad y temperatura en cada planta, además se dispone del control de activación/desactivación de la banda transportadora y del cilindro de simple efecto.

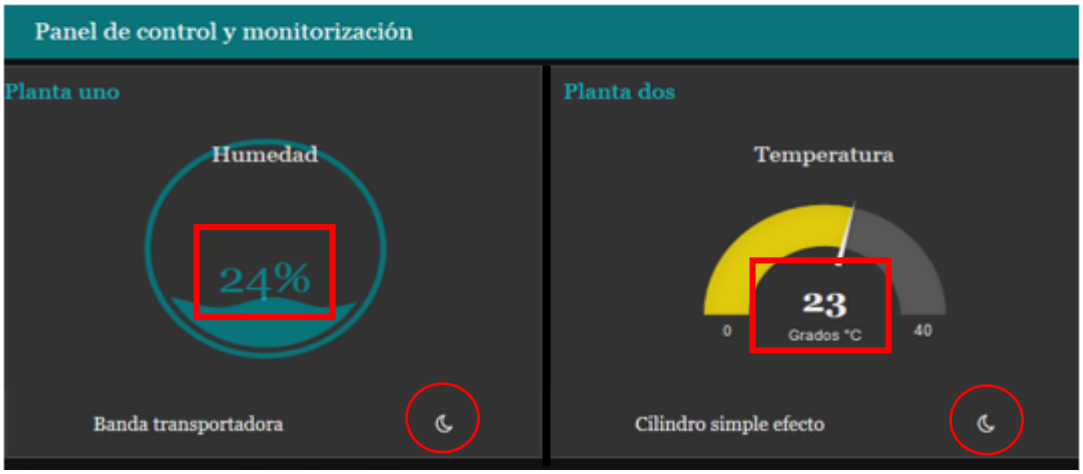


Figura 91. Sección de control de la interfaz gráfica del servidor IoT.

Elaborado por el autor.

- **Descripción de la sección de monitorización**

En la sección de monitorización Figura 92 se encuentran las cámaras IP configura das en los pasos anteriores, en el lado izquierdo se muestra la retransmisión en tiempo real del proceso uno a través de la cámara de la Raspberry Pi mientras que en el lado derecho se muestra la visualización en tiempo real del segundo proceso a través de la cámara del smartphone.

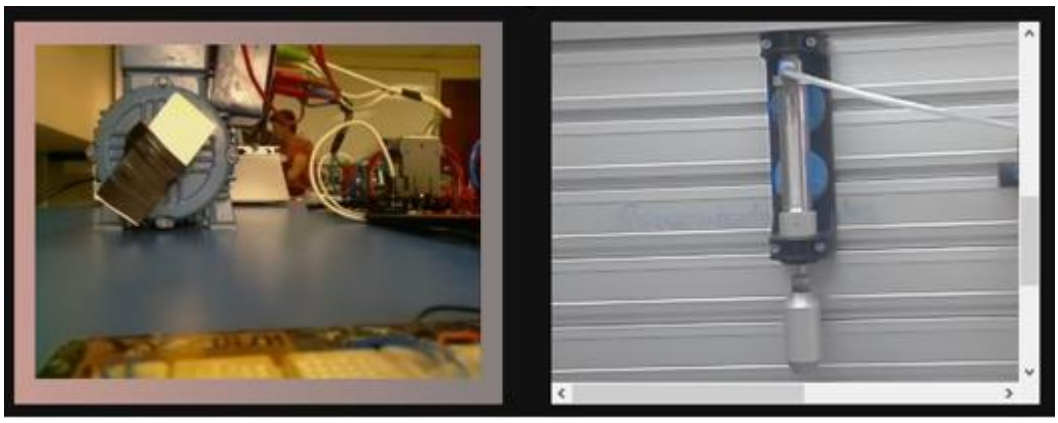


Figura 92. Sección de monitorización de la interfaz gráfica del servidor IoT.

Elaborado por el autor.

Finalmente, en la Figura 93 se puede apreciar la capa SDN y la capa de Aplicación IoT implementadas de acuerdo con la topología presentada en la Figura 18 de la sección 3.1.1.2.

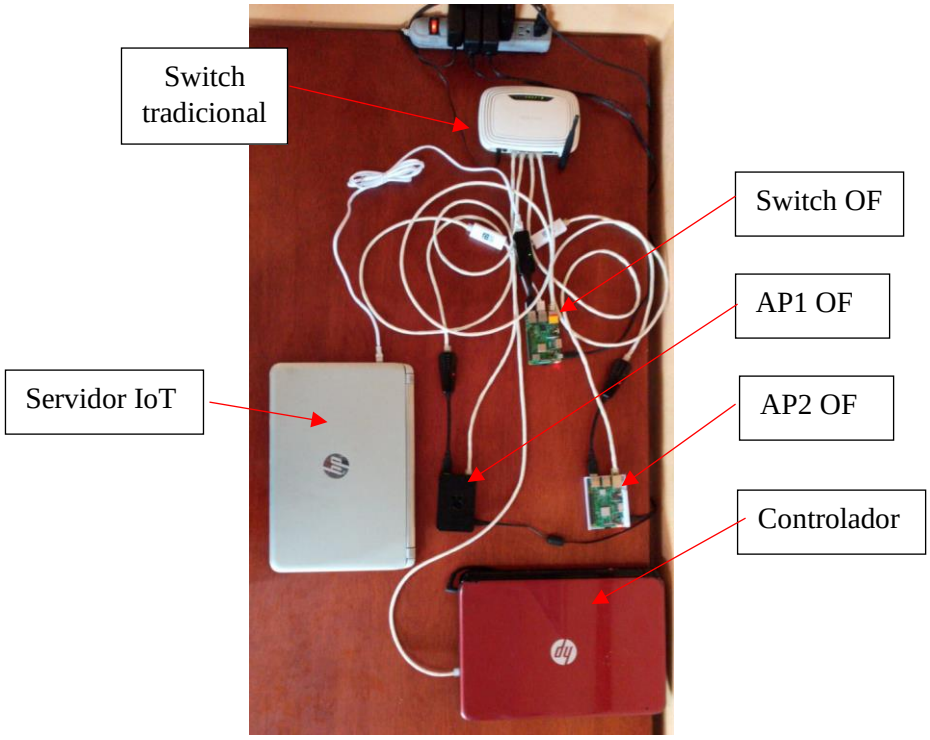


Figura 93. Implementación de las capas SDN y aplicación IoT del prototipo.
Elaborado por el autor.

3.1.1.6 Pruebas de funcionamiento

3.1.1.6.1 Pruebas de la aplicación Firewall en el Testbed

Una vez que se ha implementado el prototipo se procede a probar el funcionamiento de la aplicación firewall desarrollada, para esto se utilizan los servicios mostrados en la Tabla 15, que corresponden a los servidores implementados para cada proceso del Testbed.

Tabla 15. Servicios del Testbed.

Proceso uno				
Dispositivo	**Dirección IP**	**Dirección MAC**	**Servicio**	**Puerto TCP**
Raspberry Pi	*192.168.3.15*	*B8:27:EB:70:BA:85*	*SSH/HTTP*	*22/80*
NodeMCU	*192.168.3.20*	*84:0D:8E:8E:2D:64*	----------	----------
Proceso dos				
Dispositivo	**Dirección IP**	**Dirección MAC**	**Servicio**	**Puerto TCP**
Smartphone	192.168.3.70	F0:72:8C:BA:89:C0	PGPfone	4747
NodeMCU	192.168.3.75	84:0D:8E:8E:36:5D	----------	----------
Servidor IoT				
Dispositivo	**Dirección IP**	**Dirección MAC**	**Servicio**	**Puerto TCP**
PC-HP	192.168.3.129	A0:D3:C1:5A:D8:D8	MQTT	1883

Elaborado por el autor.

Para ejecutar la aplicación firewall y su interfaz gráfica de control se ejecutan los siguientes comandos:

$ ryu-manager appFirewall.py

$python interfaz.py

- **Prueba de bloqueo y activación de tráfico en toda la SDN**

Para probar el bloqueo del tráfico en la SDN se procede a ingresar y aceptar la regla con el protocolo ICMP para todos los switches como se muestra en la Figura 94. una

vez ingresada la regla lo que se hace es bloquear el tráfico con el interruptor incorporado en la interfaz gráfica.

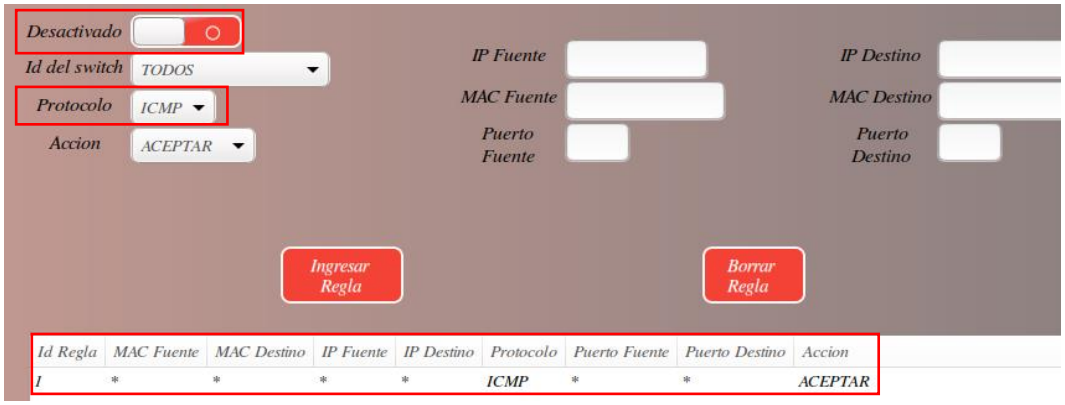


Figura 94. Bloqueo general del tráfico de datos en la SDN.

Elaborado por el autor.

A continuación, se procede a probar conectividad entre el servidor IoT y cada uno de los dispositivos correspondientes al proceso uno y dos con el comando ping. Como se ilustra en la Figura 95 todo tráfico de la red se encuentra bloqueado, incluso el tráfico con los paquetes ICMP ya que no pueden llegar a su destino.

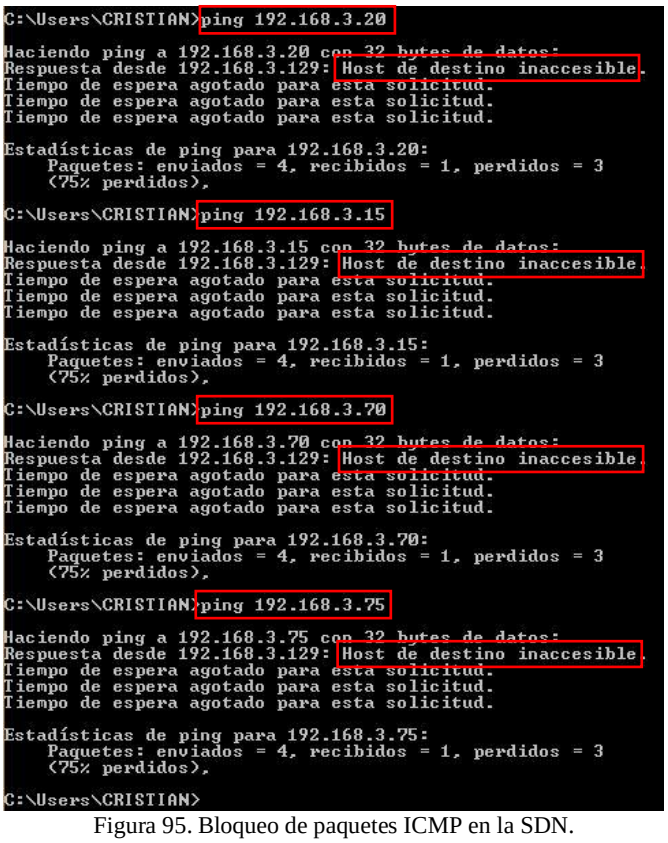


Figura 95. Bloqueo de paquetes ICMP en la SDN.

Elaborado por el autor.

Para habilitar el tráfico de red en la SDN se procede a activar el interruptor, como se muestra en el Figura 96, y en la Figura 97 se puede apreciar como los dispositivos de ambos procesos ya poseen conectividad con el servidor IoT una vez que se ejecuta el comando ping para cada uno de ellos.

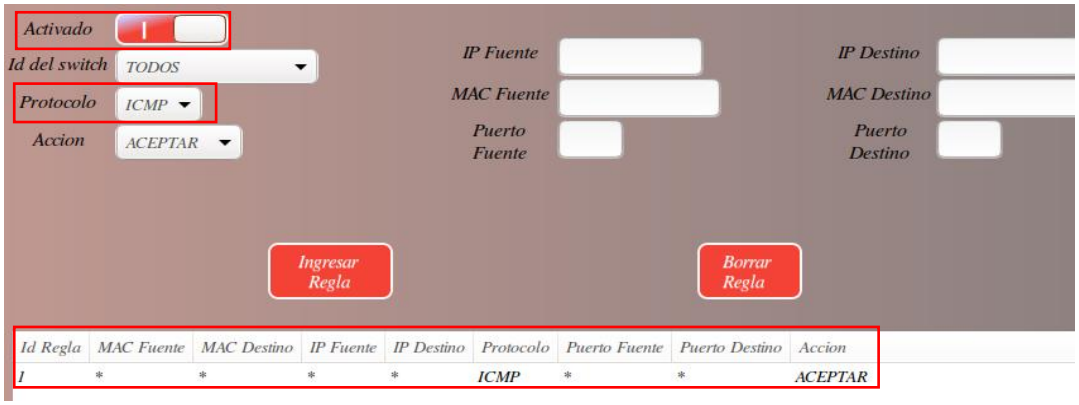


Figura 96. Desbloqueo general del tráfico de datos en la SDN.

Elaborado por el autor.

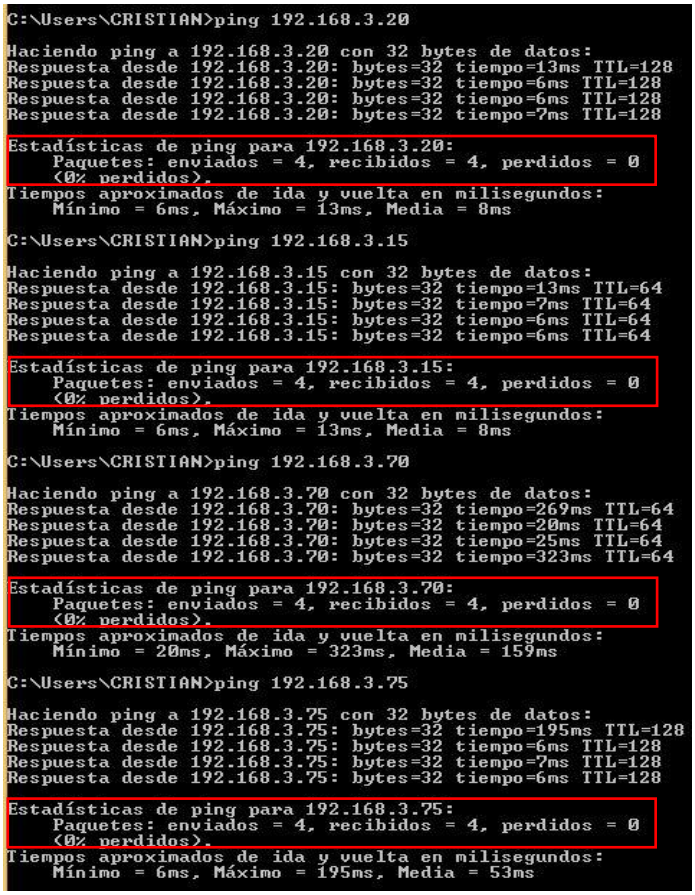


Figura 97. Pruebas de conectividad exitosa en la SDN.

Elaborado por el autor.

- **Prueba de inserción de reglas para permitir el tráfico de datos de los dispositivos de control (NodeMCUs) y el servidor IoT mediante el servicio MQQT**

En la Figura 98 se puede apreciar que la interfaz gráfica de control y monitorización de los procesos no tiene acceso a los dominios de los dispositivos IoT y a la parte de monitorización, debido a que no se han ingresado aun las reglas de control en la aplicación firewall para permitir el tráfico de datos.

Figura 98.Interfaz gráfica de la aplicación IoT sin acceso a los dominios de los dispositivos IoT.
Elaborado por el autor.

Lo primero que se hace es ejecutar *node-red* en el símbolo del sistema (cmd) del servidor IoT como se muestra en la Figura 99 para activar el servicio de MQTT.

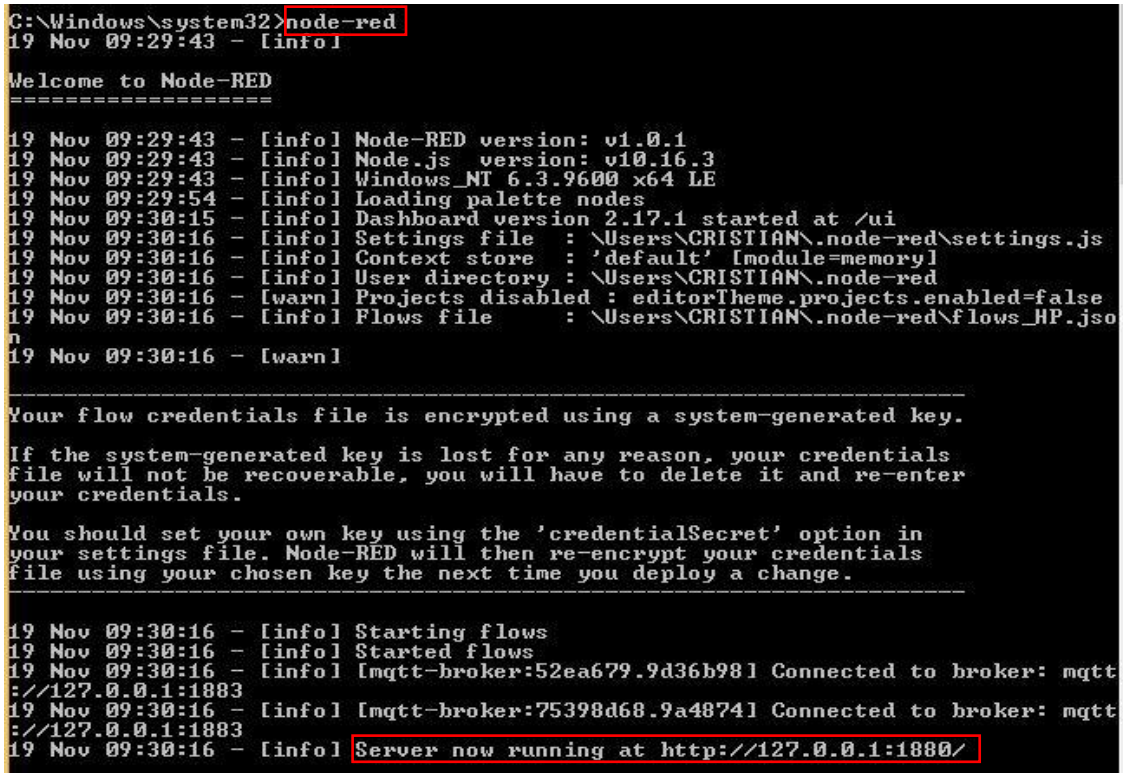


Figura 99. Ejecución del servicio MQQT en el servidor IoT.

Elaborado por el autor.

Luego se ingresan las reglas de control a la aplicación Firewall a través de la interfaz gráfica creada como se muestra en la Figura 100, hay que tener presente que el ingreso de reglas se lo puede hacer a través de las direcciones IP o MAC, pero no ambas a la vez ya que esto podría ocasionar errores en la aplicación firewall.

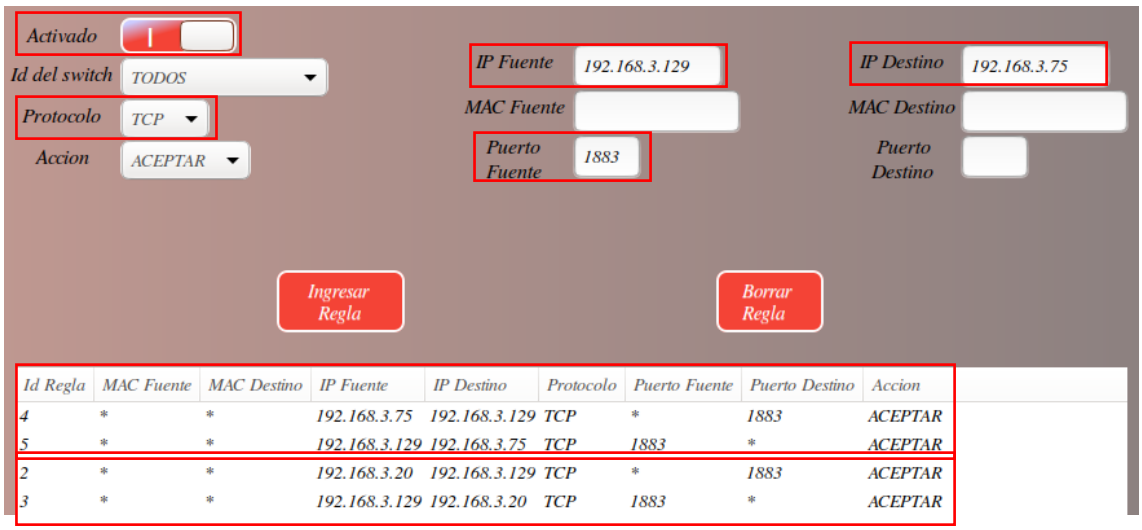


Figura 100. Ingreso de reglas con el protocolo MQTT.

Elaborado por el autor.

Una vez ingresadas las reglas de control se procede a visualizar la interfaz gráfica del servidor IoT para comprobar la conexión y el envío de datos de los dispositivos IoT como se muestra en la Figura 101.

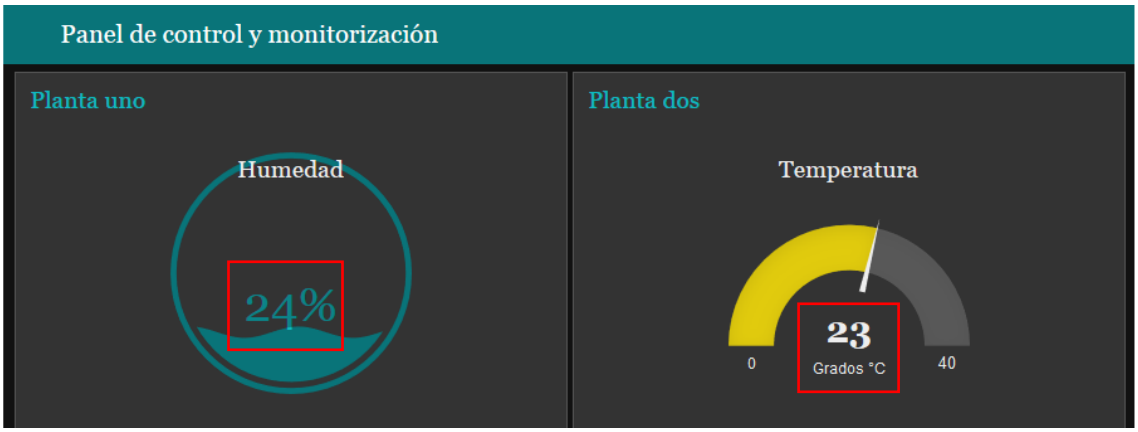


Figura 101. Recepción de datos en el servidor IoT a través del protocolo MQTT.

Elaborado por el autor.

- **Prueba de inserción de reglas para permitir el tráfico de datos del servidor IoT y la cámara IP del proceso uno mediante los servicios FTP y HTTP.**

Se ingresan las reglas de control a la aplicación Firewall a través de la interfaz gráfica creada como se muestra en la Figura 102.

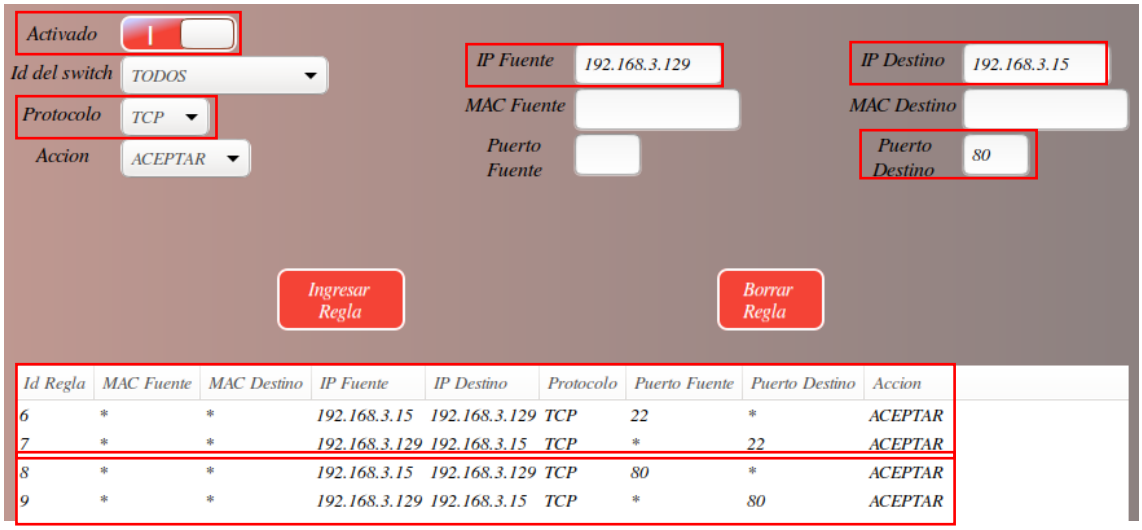


Figura 102. Ingreso de reglas con los protocolos SSH y HTTP.

Elaborado por el autor.

Una vez ingresadas las reglas se procede a ejecutar la cámara IP en la Raspberry Pi, para esto es necesario seguir los pasos mostrados a continuación.

1. Acceder de forma remota al sistema operativo de la Raspberry Pi mediante el software Putty a través del puerto 22 con el servicio SSH como se muestra en la Figura 103.


```
login as: pi
pi@192.168.3.15's password:
Access denied
pi@192.168.3.15's password:
Linux raspberrypi 4.14.34-v7+ #1110 SMP Mon Apr 16 15:18:51 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Oct 16 23:17:11 2019

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.
```

Figura 103. Conexión SSH desde el servidor IoT exitosa.

Elaborado por el autor.

2. Ejecutar la aplicación appCam.py en la ruta especificada como se muestra en la Figura 104.

```
pi@raspberrypi:~ $ cd Documents/
pi@raspberrypi:~/Documents $ cd camWebServer/
pi@raspberrypi:~/Documents/camWebServer $ sudo python3 appCam.py
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger pin code: 794-585-802
```

Figura 104. Ejecución de la cámara IP en el proceso uno.

Elaborado por el autor.

Finalmente, se procede a visualizar la interfaz gráfica del servidor IoT para comprobar la conexión y el envío de datos de la cámara IP configura da en la Raspberry Pi como se muestra en la Figura 105.

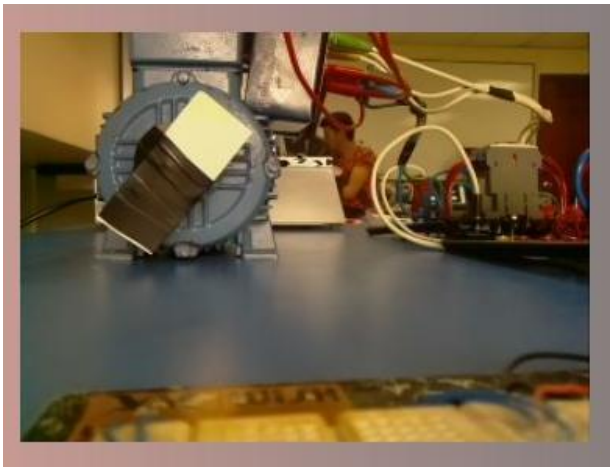


Figura 105. Conexión HTTP desde el servidor IoT exitosa.

Elaborado por el autor.

- **Prueba de inserción de reglas para permitir el tráfico de datos del servidor IoT y la cámara IP del proceso dos mediante el servicio PGPfone.**

Se ingresan las reglas de control a la aplicación Firewall a través de la interfaz gráfica creada como se muestra en la Figura 106.

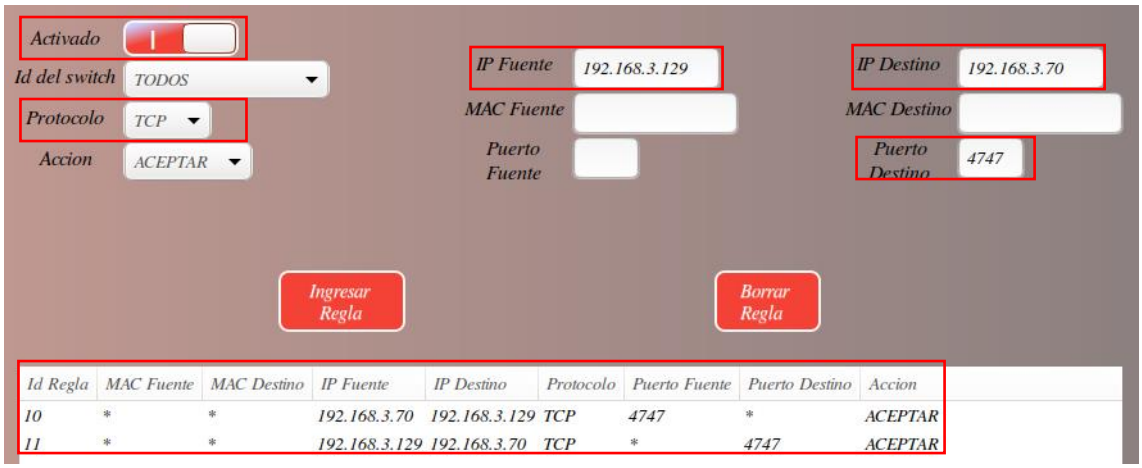


Figura 106. Ingreso de regla con el protocolo PGPfone.

Elaborado por el autor.

Una vez ingresada la regla se procede a visualizar la interfaz gráfica del servidor IoT para comprobar la conexión y el envío de datos de la cámara IP del smartphone como se muestra en la Figura 107.

Figura 107. Conexión PGPfone desde el servidor IoT exitosa.

Elaborado por el autor.

- **Prueba de control de los procesos en la industria 4.0**

Se activan los procesos mediante la ejecución de los switches en la interfaz gráfica del servidor IoT como se aprecia en la Figura 108, a través de las cámaras IP se monitorea el funcionamiento de cada proceso.

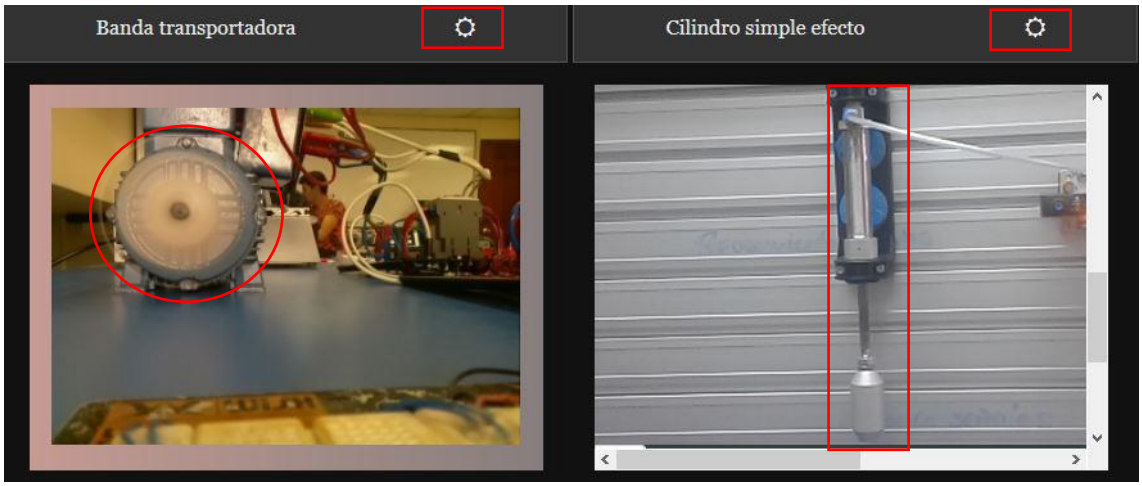


Figura 108. Activación y monitorización de los procesos en la industria 4.0.

Elaborado por el autor.

Luego se desactivan los procesos y se monitorea el resultado como se muestra en la Figura 109.

Figura 109. Desactivación y monitorización de los procesos en la industria 4.0.

Elaborado por el autor.

3.1.1.6.2 Pruebas de comunicación entre los elementos OpenFlow de la SDN

En esta sección se capturan los paquetes de los mensajes básicos Figura 110 que se intercambian entre los diferentes switches implementados y el controlador RYU cuando existe un intercambio de flujos con el protocolo OpenFlow, los paquetes son capturados con el analizador de paquetes gratuito Wireshark.

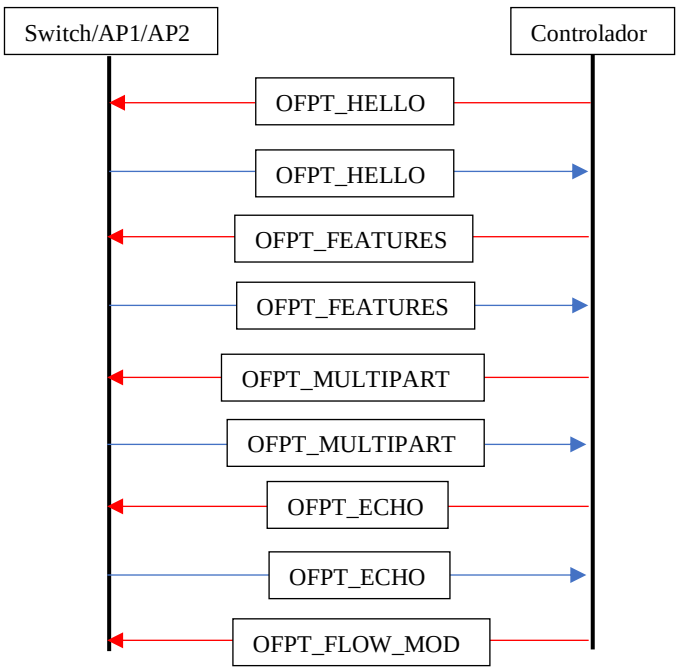


Figura 110. Mensajes iniciales entre el controlador y los conmutadores.

Elaborado por el autor.

En la Tabla 16 se presentan los mensajes OpenFlow capturados con su respectiva descripción.

Tabla 16. Descripción de los Mensajes iniciales entre el controlador y los conmutadores *[52]*.

Mensaje	**Descripción**
OFPT_HELLO	Cuando se establece por primera vez una conexión OpenFlow, cada lado de la conexión debe enviar inmediatamente un mensaje OFPT_HELLO con el campo de versión establecido en la versión de protocolo OpenFlow más alta admitida por el remitente, en este caso es: *ofp_versión= 0x04*. Este mensaje de saludo puede incluir opcionalmente algunos elementos de OpenFlow para ayudar a la Configuración de la conexión
OFPT_FEATURES	Una de las primeras cosas que debe hacer el controlador es enviar un mensaje OFPT_FEATURES_REQUEST para obtener el Datapath ID del conmutador y leer sus capacidades básicas.
OFPT_MULTIPART	Los mensajes OFPT_MULTIPART se utilizan para codificar solicitudes o respuestas que potencialmente transportan una gran cantidad de datos y no siempre caben en un solo mensaje de OpenFlow, que está limitado a 64 KB. La solicitud o respuesta se codifica como una secuencia de mensajes *Multipart* y el receptor la vuelve a ensamblar. Los mensajes *Multipart* se utilizan principalmente para solicitar estadísticas o información de estado del conmutador.
OFPT_ECHO	Los mensajes de *echo* pueden enviarse desde el conmutador o el controlador, y deben devolver una respuesta de "echo". Se usan principalmente para verificar la vida de una

	conexión de controlador-interruptor, y también se pueden usar para medir su latencia o ancho de banda
OFPT_FLOW_MOD	Las solicitudes de modificaciones a una tabla de flujo desde el controlador se realizan con el mensaje OFPT_FLOW_MOD

- **Prueba de intercambio de mensajes OFPT_HELLO entre los conmutadores y el controlador.**

A continuación, en la Figura 111, Figura 112 y Figura 113 se muestra el intercambio de mensajes OFPT_HELLO y en la Tabla 17 se pueden apreciar los campos capturados.

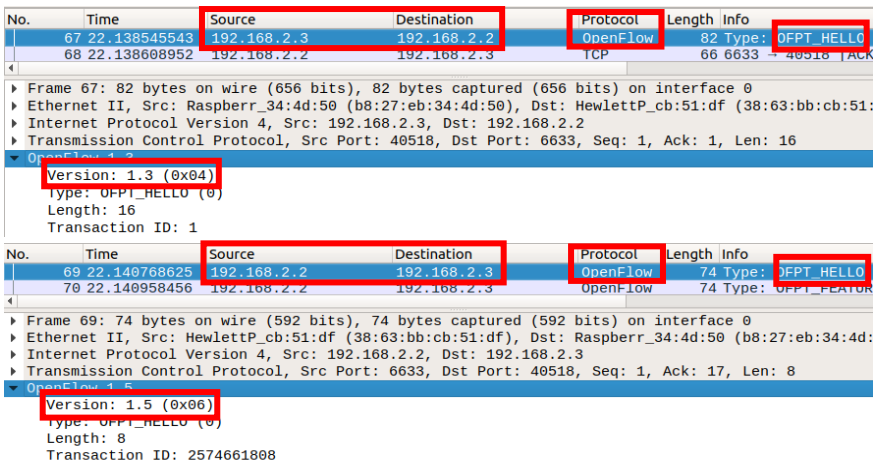


Figura 111. Paquetes capturados de los mensajes HELLO entre el controlador y el switch.

Elaborado por el autor.

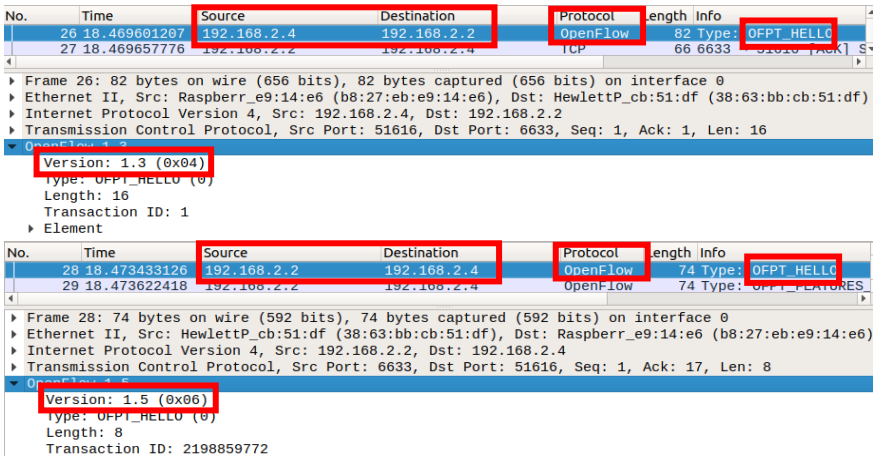


Figura 112. Paquetes capturados de los mensajes HELLO entre el controlador y el AP1.

Elaborado por el autor.

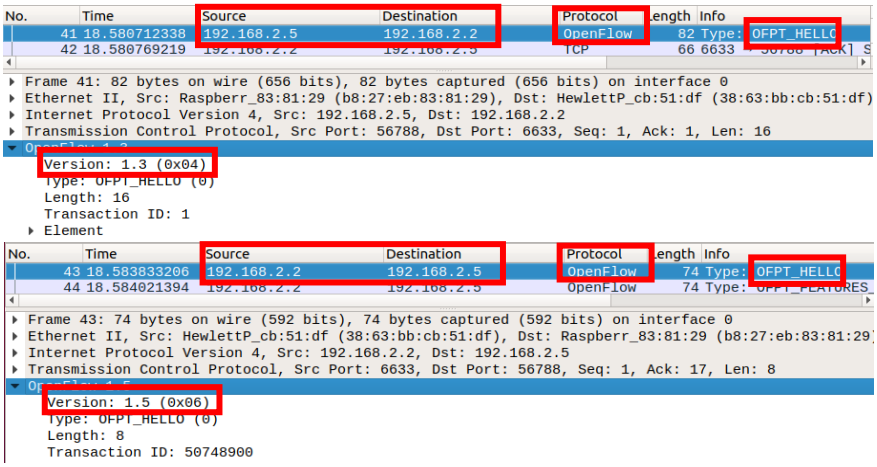


Figura 113. Paquetes capturados de los mensajes HELLO entre el controlador y el AP2.

Elaborado por el autor.

Tabla 17. Resumen de los paquetes capturados en el intercambio de mensajes HELLO.

Elemento	***P/R***	***Protocolo***	***Versión P.***	***Mensaje***
Controlador	Petición	OpenFlow	v1.5 (0x06)	OFPT_HELLO
Switch	Respuesta	OpenFlow	v1.3 (0x04)	OFPT_HELLO
AP1	Respuesta	OpenFlow	v1.3 (0x04)	OFPT_HELLO
AP2	Respuesta	OpenFlow	v1.3 (0x04)	OFPT_HELLO

Elaborado por el autor.

- **Prueba de intercambio de mensajes OFPT_FEATURES entre los conmutadores y el controlador.**

A continuación, en la Figura 114, Figura 115 y Figura 116 se muestra el intercambio de mensajes OFPT_FEATURES y en la Tabla 18 se pueden apreciar los campos capturados.

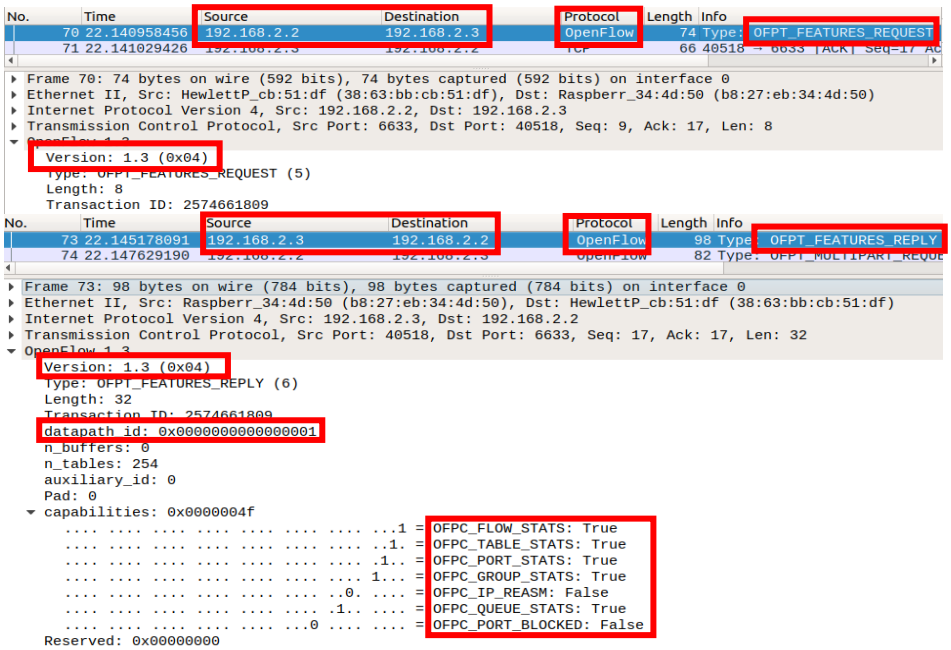


Figura 114. Paquetes capturados de los mensajes FEATURES entre el controlador y el switch.

Elaborado por el autor.

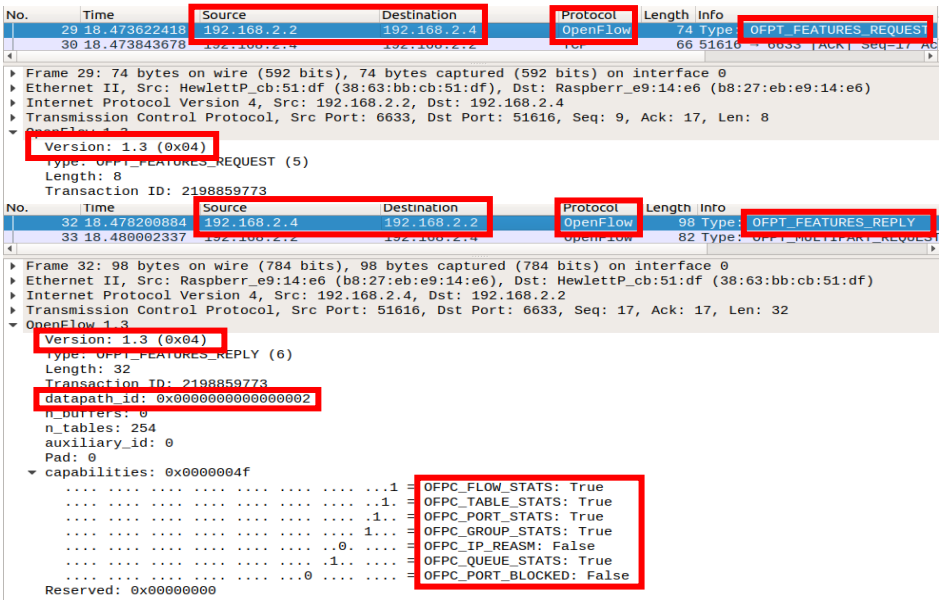


Figura 115. Paquetes capturados de los mensajes FEATURES entre el controlador y el AP1.

Elaborado por el autor.

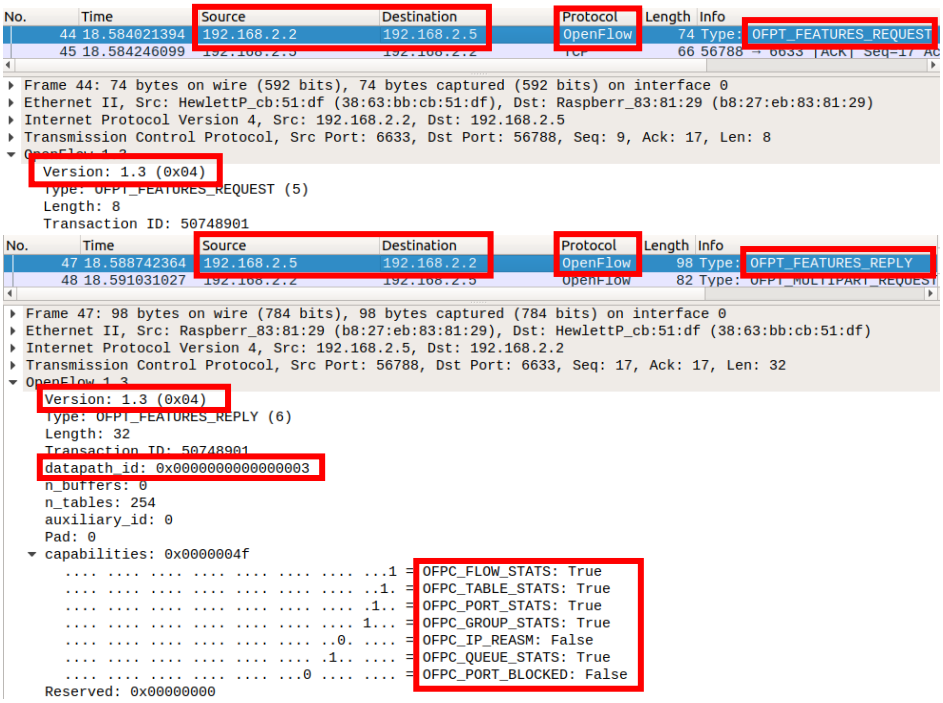


Figura 116. Paquetes capturados de los mensajes FEATURES entre el controlador y el AP2.

Elaborado por el autor.

Tabla 18. Resumen de los paquetes capturados en el intercambio de mensajes FEATURES.

Elemento	***Controlador***	***Switch***	***AP1***	***AP2***
P/R	Petición	Respuesta	Respuesta	Respuesta
Protocolo	OpenFlow	OpenFlow	OpenFlow	OpenFlow
Versión	v1.3 (0x04)	v1.3 (0x04)	v1.3 (0x04)	v1.3 (0x04)
Mensaje	FEATURES_REQUEST	FEATURES_REPLY	FEATURES_REPLY	FEATURES_REPLY
Respuesta	----------	Datapath ID y características del switch	Datapath ID y características del AP1	Datapath ID y características del AP2

Elaborado por el autor.

- **Prueba de intercambio de mensajes OFPT_MULTIPART entre los conmutadores y el controlador.**

A continuación, en la Figura 117, Figura 118 y Figura 119 se muestra el intercambio de mensajes OFPT_ MULTIPART y en la Tabla 19 se pueden apreciar los campos capturados.

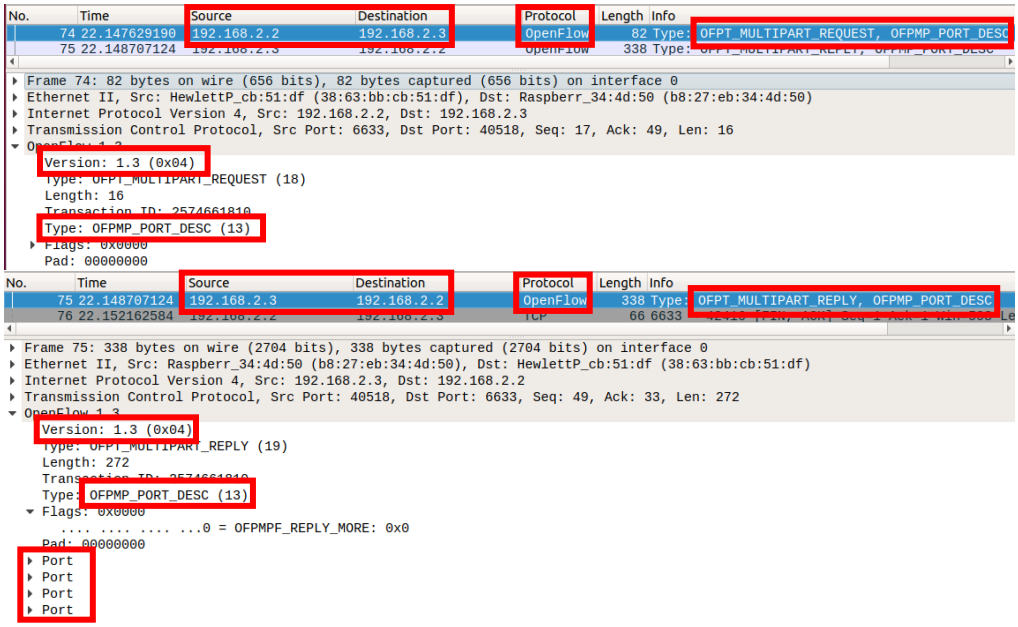


Figura 117. Paquetes capturados de los mensajes MULTIPART entre el controlador y el switch.

Elaborado por el autor.

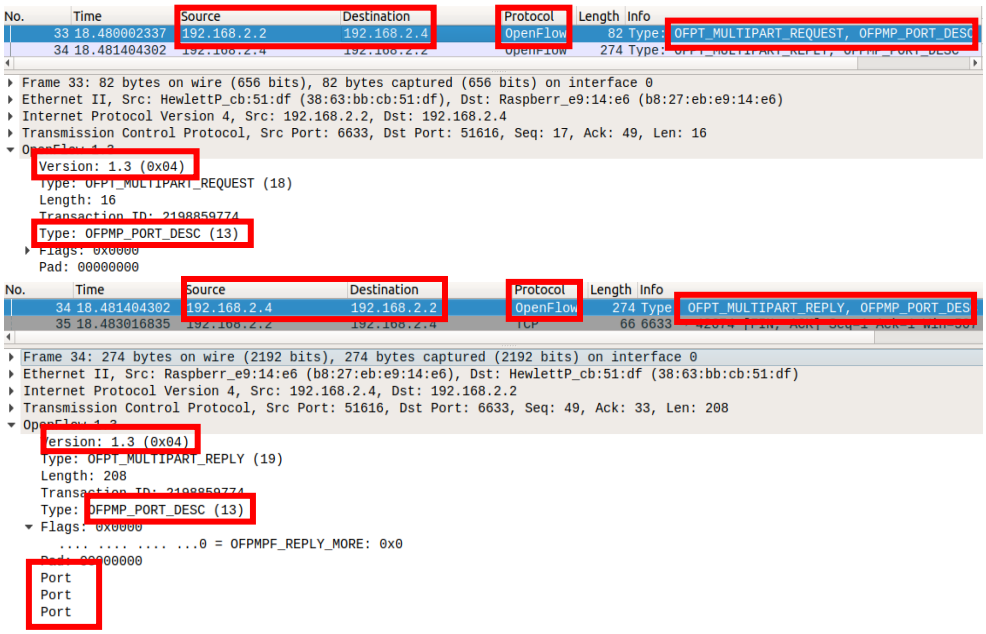


Figura 118. Paquetes capturados de los mensajes MULTIPART entre el controlador y el AP1.

Elaborado por el autor.

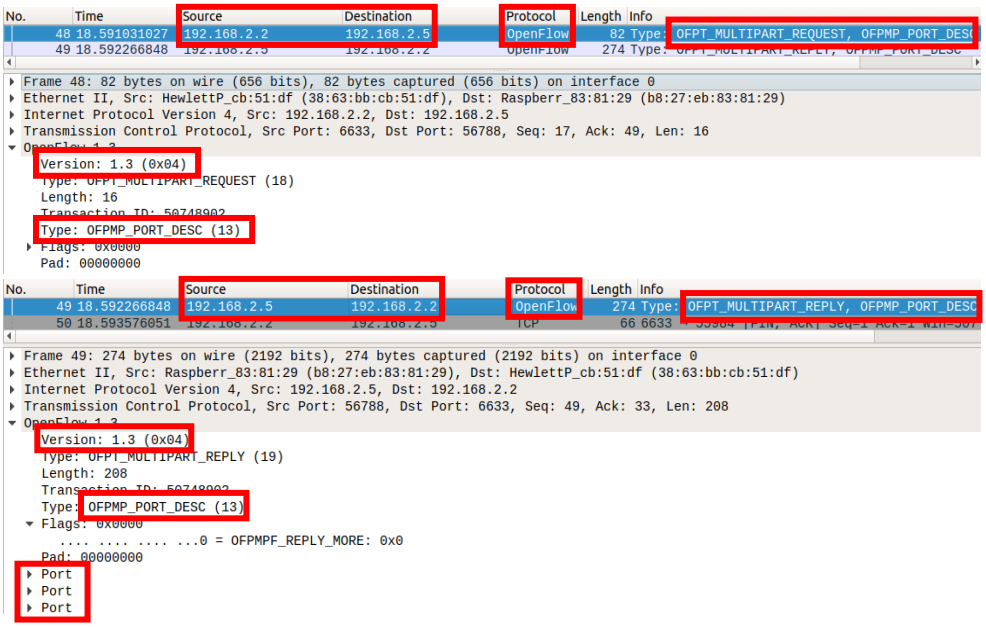


Figura 119. Paquetes capturados de los mensajes MULTIPART entre el controlador y el AP2.

Elaborado por el autor.

Tabla 19. Resumen de los paquetes capturados en el intercambio de mensajes MULTIPART.

Elemento	***Controlador***	***Switch***	***AP1***	***AP2***
P/R	Petición	Respuesta	Respuesta	Respuesta
Protocolo	OpenFlow	OpenFlow	OpenFlow	OpenFlow
Versión	v1.3 (0x04)	v1.3 (0x04)	v1.3 (0x04)	v1.3 (0x04)
Mensaje	MULTIPART_REQUEST	MULTIPART_REPLY	MULTIPART_REPLY	MULTIPART_REPLY
Respuesta	----------	Descripción de los puertos del switch	Descripción de los puertos del AP1	Descripción de los puertos del AP2
Requerimiento	PORT_DESC	PORT_DESC	PORT_DESC	PORT_DESC

Elaborado por el autor.

- **Prueba de intercambio de mensajes OFPT_ ECHO entre los conmutadores y el controlador.**

A continuación, en la Figura 120, Figura 121 y Figura 122 se muestra el intercambio de mensajes OFPT_ ECHO y en la Tabla 20 se pueden apreciar los campos capturados.

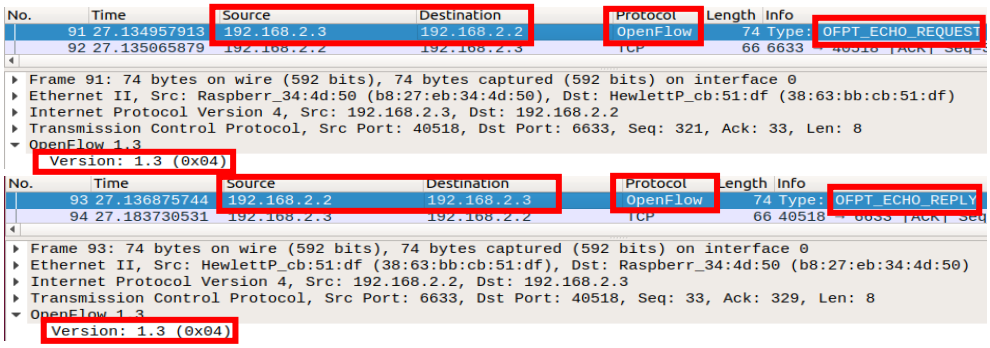


Figura 120. Paquetes capturados de los mensajes ECHO entre el controlador y el switch.

Elaborado por el autor.

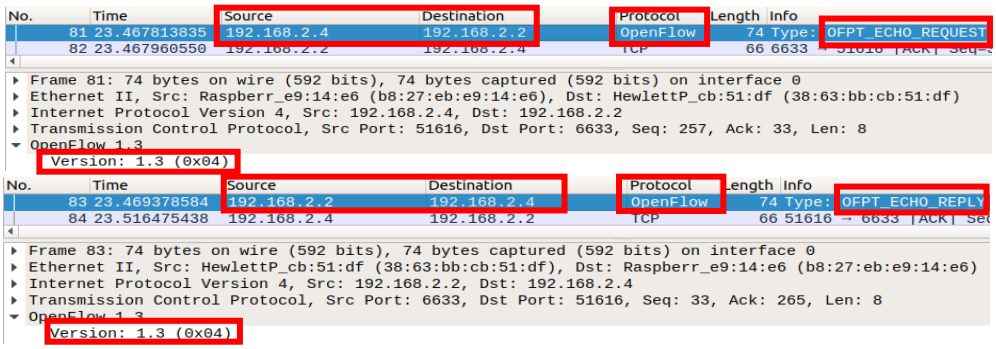


Figura 121. Paquetes capturados de los mensajes ECHO entre el controlador y el AP1.

Elaborado por el autor.

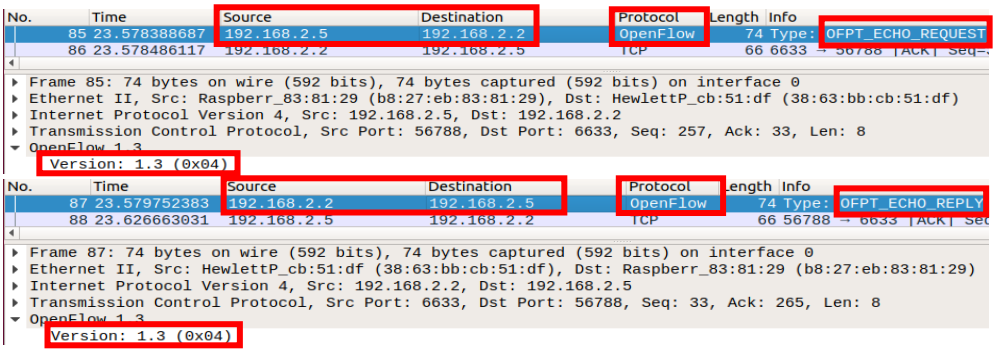


Figura 122. Paquetes capturados de los mensajes ECHO entre el controlador y el AP2.

Elaborado por el autor.

Tabla 20. Resumen de los paquetes capturados en el intercambio de mensajes ECHO.

Elemento	***P/R***	***Protocolo***	***Versión P.***	***Mensaje***
Controlador	Petición	OpenFlow	v1.3 (0x04)	ECHO_REQUEST
Switch	Respuesta	OpenFlow	v1.3 (0x04)	ECHO_REPLY
AP1	Respuesta	OpenFlow	v1.3 (0x04)	ECHO_REPLY
AP2	Respuesta	OpenFlow	v1.3 (0x04)	ECHO_REPLY

Elaborado por el autor.

- **Prueba de intercambio de mensajes OFPT_FLOW_MOD entre los conmutadores y el controlador.**

A continuación, en la Figura 123, Figura 124 y Figura 125 se muestra el intercambio de mensajes OFPT_ FLOW_MOD y en la Tabla 20 se pueden apreciar los campos capturados.

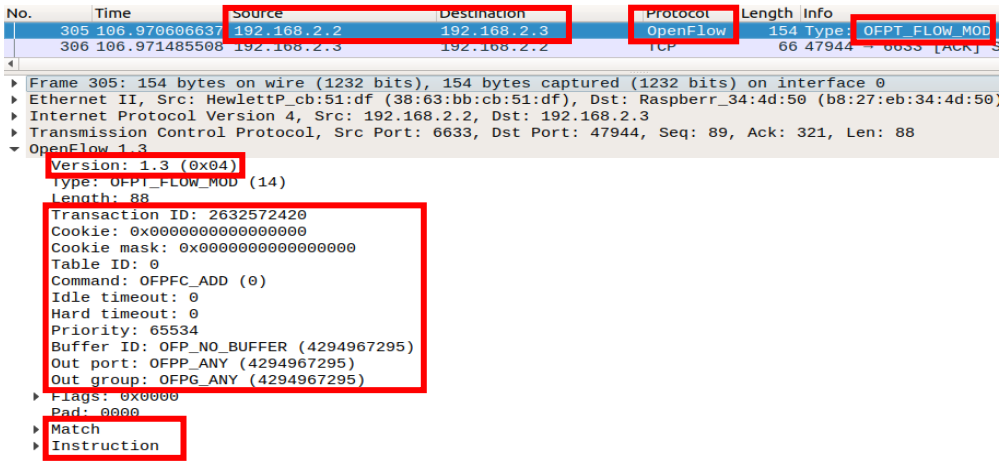


Figura 123. Paquetes capturados en el envío del mensaje FLOW_MOD desde el controlador al switch.

Elaborado por el autor.

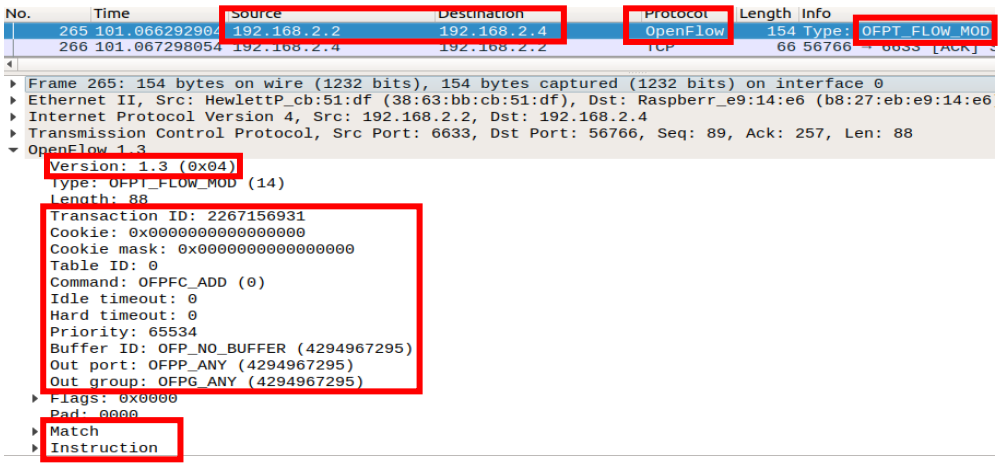


Figura 124. Paquetes capturados en el envío del mensaje FLOW_MOD desde el controlador al AP1.

Elaborado por el autor.

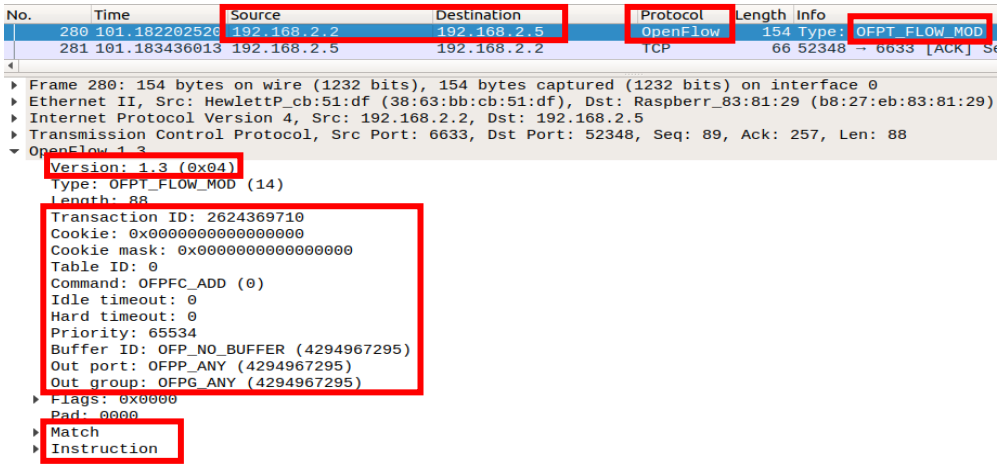


Figura 125. Paquetes capturados en el envío del mensaje FLOW_MOD desde el controlador al AP2.

Elaborado por el autor.

Tabla 21. Resumen de los paquetes capturados en el envío de mensajes FLOW_MOD.

Elemento	***Controlador-Switch***	***Controlador-AP1***	***Controlador.AP2***
Mensaje	FLOW_MOD	FLOW_MOD	FLOW_MOD
Protocolo	OpenFlow	OpenFlow	OpenFlow
Versión	v1.3 (0x04)	v1.3 (0x04)	v1.3 (0x04)
Parámetros de la tabla de flujo	▪ Cookie ▪ Información de emparejamiento ▪ Puerto de salida ▪ Instrucciones para el manejo de flujos	▪ Cookie ▪ Información de emparejamiento ▪ Puerto de salida ▪ Instrucciones para el manejo de flujos	▪ Cookie ▪ Información de emparejamiento ▪ Puerto de salida ▪ Instrucciones para el manejo de flujos

Elaborado por el autor.

3.1.1.6.3 Pruebas de tráfico de datos en la SDN

Para realizar las pruebas de tráfico en la red definida por software se ha utilizado la herramienta *iperf*, que permite crear flujos de datos TCP/UDP y medir el rendimiento de la red. En esta sección se ha podido determinar el Throughput TCP de las cámaras IP implementadas en cada proceso.

- **Throughput TCP entre la cámara IP del proceso uno y el servidor IoT**

Se procede a instalar la herramienta *iperf* en la Raspberry Pi donde se implementó la cámara IP para el proceso uno al igual que en la PC del servidor IoT una vez que se ha instalado el software se procede a realizar los siguientes pasos:

1. Acceder al terminal de la Raspberry Pi y levantar el servidor iperf con el siguiente comando: *$sudo iperf3 -s*
2. En la PC del servidor IoT se accede al *cmd* y se ejecuta el cliente iperf a través de la dirección IP y el puerto de acceso del servidor iperf, con el siguiente comando:

>iperf3.exe -c 192.168.3.15 -p5201 -t180

En el comando también se especifica el tiempo en segundos en que se generara el tráfico entre el cliente y el servidor iperf. Los resultados obtenidos después del máximo tráfico soportado entre las interfaces durante 3 minutos se pueden apreciar en la Figura 126.

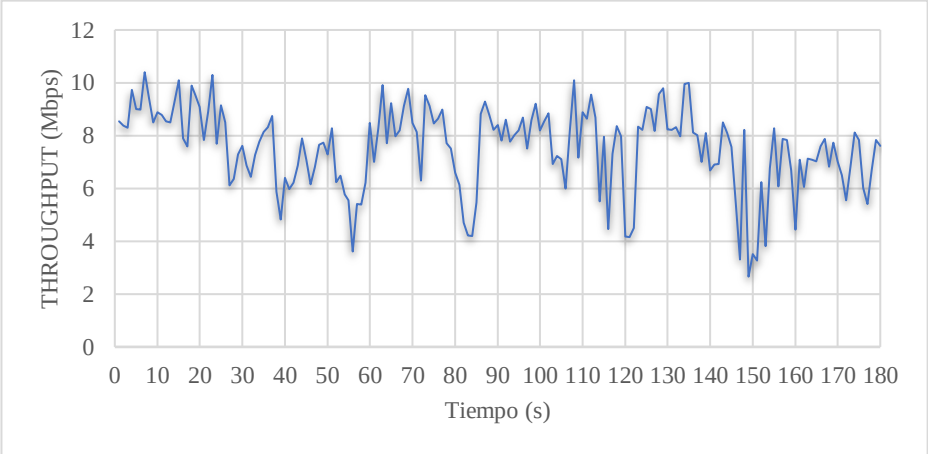


Figura 126. Throughput entre la cámara IP del proceso uno y el servidor IoT.

Elaborado por el autor.

Al analizar la Figura 126 se pueden ver las variaciones propias de la comunicación inalámbrica entre la cámara IP y el servidor IoT, además de los problemas de actualización de entradas en las tablas de flujos al inicio de la comunicación en el switch o AP1.

- **Throughput TCP entre la cámara IP del proceso dos y el servidor IoT**

En este caso se procede a descargar e instalar la aplicación *Magic iPerf* en el smartphone y se siguen los pasos mostrados a continuación:

1. Acceder a la aplicación *Magic iPerf* e ingresar el siguiente comando para ejecutar el servidor iperf en el smartphone:

-s-i 1

2. En la PC del servidor IoT se accede al *cmd* y se ejecuta el cliente iperf a través de la dirección IP y el puerto de acceso del servidor iperf, con el siguiente comando:

>iperf3.exe -c 192.168.3.70 -p5201 -t180

Los resultados obtenidos después del máximo tráfico soportado entre las interfaces durante 3 minutos se pueden apreciar en la Figura 127.

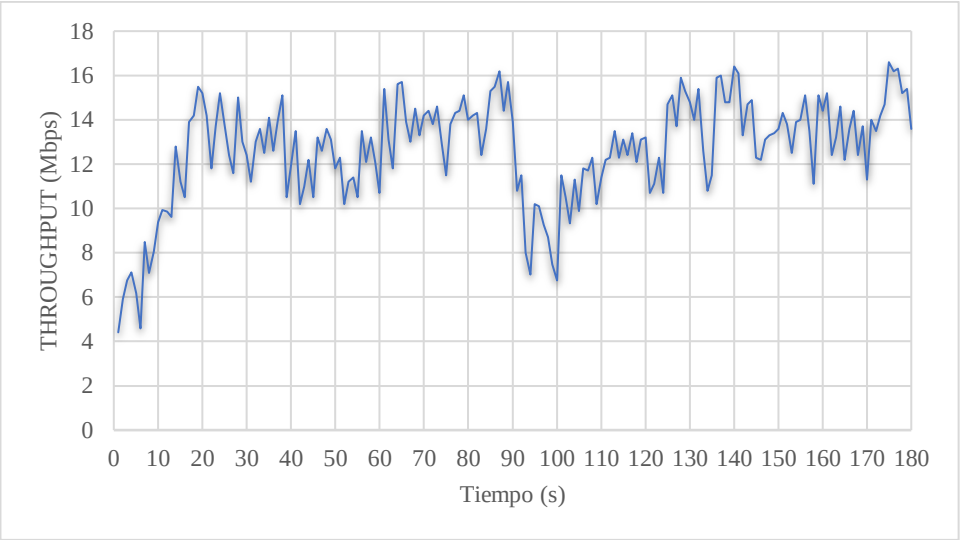


Figura 127. Throughput entre la cámara IP del proceso dos y el servidor IoT.

Elaborado por el autor.

Al igual que en la Figura 126, en la Figura 127 se pueden apreciar las variaciones propias de la comunicación inalámbrica y de los problemas con la actualización de flujos en las tablas del switch o AP2.

3.1.1.7 Costos del prototipo (Testbed)

El Testbed implementado está formado por 2 partes fundamentales, la primera corresponde a la de la red definida por software y la segunda a los procesos de la industria 4.0 con aplicación al IoT.

La implementación de los procesos no ha tenido costo alguno debido a que se han utilizado los recursos disponibles en la Facultad de Sistemas, Electrónica e Industrial, recursos tales como laboratorios y materiales.

En la implementación de la SDN se obtuvieron los costos mostrados en la Tabla 22, las PCs utilizadas para la implementación del servidor IoT y el controlador Ryu pueden

ser reemplazadas por equipos de distintos precios, dependiendo de los requerimientos del prototipo a implementar, en este caso se utilizaron las PCs de la Tabla 22 debido a su disponibilidad. Las Raspberry Pi compradas incluyen los accesorios tales como: microSD clase 10, cargador y disipadores.

Tabla 22. Costos del prototipo implementado.

Equipo	**Cantidad**	**Precio Unitario**	**Total**
Raspberry Pi 3 Model B+	2	$65	$130
Raspberry Pi 3 Model B	2	$53	$106
Adaptador UBS 3.0 /Gigabit Ethernet	3	$12	$36
Adaptador UBS 2.0 /Fast Ethernet	2	$10	$20
Router TP-Link	1	$25	$25
NodeMCU ESP8266	2	$10	$20
Cámara raspberry Pi	1	$15	$15
Smartphone S3 mini	1	$50	$50
PC HP core i7	1	$900	$900
PC HP celeron	1	$300	$300
Cable UTP	20m	$0,50	$10
Modulo relay 2 canales	1	$4	$4
Mini Relay	1	$1	$1
		Total	**$1617**

Elaborado por el autor.

En este Testbed se han implementado dispositivos OpenFlow como Switch y Access point con la ayuda de software y hardware libre, pero este tipo de dispositivos OpenFlow también se los puede encontrar en el mercado con una variación de costos y características específicas para las SDN en la Tabla 23, se presentan los costos de los dispositivos de red implementados y las alternativas disponibles en el mercado que brindan un mejor rendimiento debido a que son fabricados por grandes marcas. El objetivo de mostrar estas opciones es comparar el costo que podría llegar a tener la implementación de estos equipos en un entorno experimental.

Tabla 23. Comparación de precios de los dispositivos OpenFlow *[53]*, *[54]*, *[55]*, *[56]*.

Dispositivo	Marca	Modelo	Puertos ethernet	Interfaz Wifi	Precio
Switch OpenFlow	Implementado	*	4	×	$97
Access Point OpenFlow	Implementado	*	2	✓	$65
Switch OpenFlow	Zodiac	FX	4	×	$119
Access Point OpenFlow	Zodiac	WX	2	✓	$239
Switch OpenFlow	Pica8	P-3297	48	×	$3.960
Switch OpenFlow	EDGE	AS4600	48	×	$3.000

Una vez analizada la Tabla 23 se puede deducir que los dispositivos OpenFlow implementados son una mejor opción para investigadores y estudiantes que deseen experimentar con las redes definidas por software.

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Al finalizar el presente trabajo de titulación se obtuvieron las siguientes conclusiones:

- Se propuso una arquitectura SDN-IoT simple y general con funciones de virtualización de red para abordar los problemas de seguridad, escalabilidad y movilidad en la red IoT. En el intercambio inicial de mensajes HELLO entre los dispositivos de red y el controlador, existe una negociación para el uso de la versión del protocolo OpenFlow a utilizar en la red, en el caso del prototipo implementado es: *ofp_versión= 0x04*, que corresponde a la versión 1.3.

- En la búsqueda de evidenciar las potencialidades de las SDN se establecieron 2 procesos referentes a la industria 4.0 como una aplicación del IoT, los procesos son administrados y controlados a través de una aplicación firewall basada en redes definidas por software con el controlador Ryu.

- Se consiguió implementar un Testbed SDN experimental de bajo costo haciendo uso de dispositivos Open Source que brindaron la posibilidad de crear Switch y Access Point compatibles con el protocolo OpenFlow. El controlador seleccionado como administrador de la SDN fue Ryu, el cual permite hacer un control a nivel de red de los procesos de la industria 4.0 aplicados al IoT.

- El manejo de las reglas de flujo ingresadas en la aplicación Firewall permite realizar el control del tráfico en los procesos a través de la coincidencia de las cabeceras en las tablas de flujos de los conmutadores virtuales implementados en el prototipo.

4.2 Recomendaciones

- Al Configurar los Access Point OpenFlow en la parte de la seguridad inalámbrica, utilizar el estándar WEP debido a que el Open vSwitch disponible para el firmware OpenWrt presenta problemas de incompatibilidad con los estándares de seguridad inalámbrica WPA o WPA2.

- Las microSD de las Raspberry Pi deben ser de clase 10 para que no existan problemas en la velocidad de lectura y escritura con el fin de optimizar los recursos.

- Antes de programar cualquier aplicación con Ryu probar el funcionamiento de los dispositivos OpenFlow implementados, esto se lo puede hacer con las aplicaciones básicas ya disponibles en el Framework de Ryu.

- Verificar el soporte de los adaptadores USB-Ethernet en las distribuciones de Linux utilizadas, ya que para el correcto funcionamiento de las mismas se deben seleccionar e instalar los drivers manualmente, ya que en ocasiones los drivers no se encuentran instalados por defecto.

BIBLIOGRAFÍA

[1] CISCO, «Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper,» 2019.

[2] M. Rotenberg y C. Fitzgerald, «Civil society asks US legislators to adopt UK IoT code of practice,» epig.org, Washington, DC 20009, USA, 2019.

[3] K. Kalkan y S. Zeadally, «Securing Internet of Things (IoT) with Software Defined Networking (SDN),» *IEEE Communications,* vol. 56, nº 9, pp. 186-192, 2017.

[4] M. Mazhar, M. Atif, A. Mazhar, A. Ellahi, M. Saqib y T. Mahmood, «Conceptualization of Software Defined Network layers over internet of things for future smart cities applications,» de *2015 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, Orlando, FL, USA, 2015.

[5] S. Tayyaba, O. Khan, M. Shah y A. Ahmed, «Software Defined Network (SDN) Based Internet of Things (IoT): A Road Ahead,» de *Proceedings of the International Conference on Future Networks and Distributed Systems*, New York, NY, USA, 2017.

[6] M. Ojo, D. Adami y S. Giordano, «A SDN-IoT Architecture with NFV Implementation,» de *2016 IEEE Globecom Workshops (GC Wkshps)*, Washington, DC, EE. UU., 2016.

[7] A. Muthanna, A. Ateya, A. Khakimov, I. Gudkova, A. Abuarqoub, K. Samouylov y A. Koucheryavy, «Secure and Reliable IoT Networks Using Fog Computing with Software-Defined Networking and Blockchain,» *Sensor and Actuator Networks,* vol. 8, nº 1, pp. 1-17, 2019.

[8] J.-H. Park, H.-S. Kim y W.-T. Kim, «DM-MQTT: An Efficient MQTT Based on SDN Multicast for Massive IoT Communications,» *sensors,* vol. 18, nº 9, pp. 1-15, 2018.

[9] Y. Lu, Z. Ling, S. Zhu y L. Tang, «SDTCP: Towards Datacenter TCP Congestion Control with SDN for IoT Applications,» *sensors,* vol. 17, nº 1, pp. 1-20, 2017.

[10] E. Tomovic, K. Yoshigoe, I. Maljevic y I. Radusinovic, «Software-Defined Fog Network Architecture for IoT,» *Springer,* vol. 92, nº 1, pp. 181-196, 2016.

[11] A. Hakiri, P. Berthou, A. Gokhale y S. Abdellatif, «Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications,» *IEEE Communications Magazine,* vol. 53, nº 9, pp. 48-54, 2015.

[12] L. Carrasco, «Universidad Politécnica de Madrid,» 2019. [En línea]. Available: http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2018-19/tfm_CARRASCO_ESPINOSA_LIZ_PANAMA_2018-19.pdf. [Último acceso: 14 Abril 2019].

[13] S. Sezer, S. Scott-Hayward y K. Chouhan, «Are We Ready for SDN? Implementation Challenges for Software-Defined Networks,» *IEEE Communications,* vol. 51, nº 7, pp. 36-43, 2013.

[14] O. N. F. (ONF), «ONF,» Software-Defined Networking (SDN) Definition, 2019. [En línea]. Available: https://www.opennetworking.org/sdn-definition/. [Último acceso: 8 Abril 2019].

[15] W. Braun y M. Menth, «Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices,» *future internet,* vol. 6, nº 2, pp. 302-336, 2014.

[16] Logicalis, «SDN Como el nuevo Universo trazado por las redes definidas por software impactara en los negocios,» *Logicalis Business and Technology Working as One,* 2014.

[17] ONF, «OpenFlow Switch Specification,» Open Networking Foundation, 31 Diciembre 2009. [En línea]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf. [Último acceso: 8 Abril 2019].

[18] A. Núñez, «Red definida por software (SDN) en base a una infraestructura de software de libre distribución,» Universidad Técnica de Ambato, Abril 2015. [En línea]. Available: http://repo.uta.edu.ec/handle/123456789/10587. [Último acceso: 8 Abril 2019].

[19] P. Emmerich, D. Raumer, S. Gallenmuller, F. Wohlfart y G. Carle, «Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis,» *Journal of Network and Systems Management,* vol. 26, nº 2, p. 314–338, 2018.

[20] J. Álvarez, «Universidad de Alcalá: Escuela Politécnica Superior,» 2017. [En línea]. Available: https://ebuah.uah.es/dspace/bitstream/handle/10017/29680/TFM-Alvarez-%20Horcajo-2017.pdf?sequence=1&isAllowed=y. [Último acceso: 24 10 2019].

[21] Z. Zha, A. Wang, Y. Guo, D. Montgomery y S. Chen, «Instrumenting Open vSwitch with Monitoring Capabilities: Designs and Challenges,» *ACM,* nº 16, 2018.

[22] D. f. t. DoD, «First Draft SDN Controller SRG Overview,» United States of America, 2018.

[23] S. Han, «Software Defined Network for Internet of Things,» 14 Julio 2016. [En línea]. Available: file:///C:/Users/CRISTIAN/Documents/10mo%20Semestre/4.%20Tesis/006_WUNCA33%20SDN%20for%20IoT%20Tutorial%2014072016.pdf. [Último acceso: 30 Octubre 2019].

[24] J. Moreno, «Estudio de las redes definidas por software y escenarios virtuales de red orientados al aprendizaje,» Universidad Politécnica de Madrid, 2015. [En línea]. Available: http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2014-2015/TFM_Javier_Cano_Moreno_2015.pdf.

[25] RYU, «github,» 2017. [En línea]. Available: http://osrg.github.io/ryu/. [Último acceso: 11 Noviembre 2019].

[26] M. Velasco, «DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN PROTOTIPO PARA OFRECER EL SERVICIO DE DHCP SOBRE UNA SDN,» ESCUELA POLITÉCNICA NACIONAL, Febrero 2016. [En línea]. Available: https://bibdigital.epn.edu.ec/bitstream/15000/14783/1/CD-6809.pdf. [Último acceso: 11 Noviembre 2019].

[27] E. Tinajero, «Implementación de un prototipo de switch OpenFlow de bajo costo utilizando una Raspberry PI,» Escuela Politécnica Nacional, Octubre 2016. [En línea]. Available: https://bibdigital.epn.edu.ec/handle/15000/16737. [Último acceso: 11 Noviembre 2019].

[28] S. Rao, «THENEWSTACK,» 23 Diciembre 2014. [En línea]. Available: https://thenewstack.io/sdn-series-part-iv-ryu-a-rich-featured-open-source-sdn-controller-supported-by-ntt-labs/. [Último acceso: 11 Noviembre 2019].

[29] B. Javed, M. W. Iqbal y H. Abbas, «Internet of things (IoT) design considerations for developers and manufacturers,» de *Proceedings of the IEEE International Conference on Communications Workshops (ICC Workshops '17)*, Paris-France, 2017.

[30] G. Kortuem, F. Kawsar, D. Fitton y V. Sundramoorthy, «Smart objects as building blocks for the internet of things,» *IEEE Internet Computing,* vol. 14, nº 1, pp. 44-51, 2010.

[31] K. Yang, D. Forte y M. Tehranipoor, «Protecting endpoint devices in IoT supply chain,» de *Proceedings of the 34th IEEE/ACM International Conference on Computer-Aided Design (ICCAD '15)*, United States, 2015.

[32] P. L. González, Seguridad y responsabilidad en la internet de las cosas (IoT), España: Wolters Kluwer, 2018.

[33] P. Sethi y S. Sarangi, «Internet of Things: Architectures, Protocols, and Applications,» *Journal of Electrical and Computer Engineering,* pp. 1-25, 2017.

[34] J. Ferguson y J. Woodard, «glow labs,» 31 Marzo 2017. [En línea]. Available: http://glowlabs.co/wireless-protocols/. [Último acceso: 2019 Noviembre 14].

[35] A. Triantafyllou, P. Sarigiannidis y T. Lagkas, «Network Protocols, Schemes, and Mechanisms for Internet of Things (IoT): Features, Open Challenges, and Trends,» *Wireless Communications and Mobile Computing,* pp. 1-24, 2018.

[36] S. Giordano, D. Adami y M. Ojo, «A SDN-IoT Architecture with NFV Implementation,» de *2016 IEEE Globecom Workshops (GC Wkshps)*, Washington, DC, USA, 2016.

[37] C.-W. Tsai, C.-F. Lai y A. Vasilakos, «Future Internet of Things: open issues and challenges,» *Springer,* vol. 20, nº 8, pp. 2201-2217, 2014.

[38] P. P. Ray, «A survey on Internet of Things architectures,» *ScienceDirect,* vol. 30, nº 1, pp. 291-319, 2018.

[39] C. Thuemmler y C. Bai, «Health 4.0: Application of Industry 4.0 Design Principles in Future Asthma Management,» *Springer,* pp. 23-37, 2017.

[40] A. Pescapé, V. Persico y G. Aceto, «The role of Information and Communication Technologies in healthcare: taxonomies, perspectives, and challenges,» *Journal of Network and Computer Applications,* vol. 107, pp. 125-154, 2018.

[41] S. Kumar y S. Luthra, «Evaluating challenges to Industry 4.0 initiatives for supply chain sustainability in emerging economies,» *Process Safety and Environmental Protection,* vol. 117, pp. 168-179, 2018.

[42] D. Zuehlke, «SmartFactory Towards a factory-of-things,» *Annual Reviews in Control,* vol. 34, p. 129–138, 2010.

[43] L. Wang, J. He y S. Xu, «The Application of Industry 4.0 in Customized Furniture Manufacturing Industry,» de *MATEC Web of Conferences*, Zhengzhou, 2017.

[44] D. Hongmin, «4.0 Industrial and Intelligent Machinery Plant,» *Packaging engineering,* vol. 37, nº 19, pp. 208-209, 2016.

[45] Raspbian, «RaspberryPi,» [En línea]. Available: https://www.raspberrypi.org/downloads/raspbian/. [Último acceso: 23 Octubre 2019].

[46] S. cards, «RaspberryPi,» [En línea]. Available: https://www.raspberrypi.org/documentation/installation/sd-cards.md. [Último acceso: 23 Octubre 2019].

[47] L. F. C. Project, «Open vSwitch,» 2016. [En línea]. Available: http://www.openvswitch.org/download/. [Último acceso: 24 Octubre 2019].

[48] raspberry_pi_foundatio, «OpenWrt,» 2019. [En línea]. Available: https://openwrt.org/toh/raspberry_pi_foundation/raspberry_pi?datasrt=firmware%20openwrt%20install%20url. [Último acceso: 27 Octubre 2019].

[49] wireless_network_watcher, «NirSoft,» 2011. [En línea]. Available: https://www.nirsoft.net/utils/wireless_network_watcher.html. [Último acceso: 27 Octubre 2019].

[50] M. Marcio, «GitHub,» 2015. [En línea]. Available: https://gist.github.com/marciolm/9f0ab13b877372d08e8f. [Último acceso: 27 Octubre 2019].

[51] J. Cool y J. Epling, «GitHub,» 29 Diciembre 2016. [En línea]. Available: https://github.com/osrg/ryu/blob/master/ryu/app/rest_firewall.py. [Último acceso: 17 Noviembre 2019].

[52] O. N. Fundation, «OpenFlow Switch Specification Version 1.3.2 (Wire Protocol 0x04),» 2013.

[53] N. NETWORKS, «Zodiac FX,» 2017. [En línea]. Available: https://northboundnetworks.com/products/zodiac-fx. [Último acceso: 1 Diciembre 2019].

[54] N. NETWORKS, «Zodiac WX,» 2017. [En línea]. Available: https://northboundnetworks.com/products/zodiac-wx. [Último acceso: 1 Diciembre 2019].

[55] MYRYAD360, «Edge-Core AS4600-54T,» [En línea]. Available: https://myriad360.com/product/edge-core-as4600-54t/. [Último acceso: 15 Noviembre 2019].

[56] Netgate, «Pica8 P-3297 TCAM 48 x 1G OpenFlow Switch,» [En línea]. Available: https://store.netgate.com/Pica8/P-3297.aspx. [Último acceso: 15 Noviembre 2019].

ANEXOS

ANEXO A: Código fuente para la Configuración de las NodeMCUs

Configuración de la NodeMCU del proceso uno

```
/**********************************
 ** Librerias **
 **********************************/
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
/**********************************
 ** constantes **
 **********************************/
#define led1 D7
#define led3 D5
#define puertoMqtt 1883
/**********************************
 ** objetos y variables **
 **********************************/
WiFiClient clienteBanda;
PubSubClient client(clienteBanda);
const char * ssid = "Red_AP_uno";
const char * claveWifi = "ab12345678";
// ip del broker
const char * brokerMqtt = "192.168.3.129";
uint32_t ultimoIntentoReconexion;
uint32_t timerEnvioDatos;

void conectarAlWifi() {
 WiFi.begin(ssid, claveWifi);
 Serial.print("Conectando");
 Serial.println(ssid);
//IP estatica
 IPAddress ip(192,168,3,20);
 IPAddress gateway(192,168,3,1);
 IPAddress subnet(255,255,255,0);
 WiFi.config(ip, gateway, subnet);

 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("");
 Serial.println("Wifi Conectado ");
 Serial.println("Direccion IP: ");
 Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* mensaje, unsigned int length) {
 String topico = topic;
 Serial.print("Mensaje recibido del topico: ");
 Serial.println(topico);
 Serial.print("Mensaje: ");
 Serial.println((char)mensaje[0]);
```

```
 if (topico == "/banda") {
 if (mensaje[0]=='0') {
  digitalWrite(led1, LOW);
  digitalWrite(led3, LOW);
 }
 if (mensaje[0]=='1'){
  digitalWrite(led3, LOW);
  delay(100);
  digitalWrite(led1, HIGH);
  delay(1000);
  digitalWrite(led1, LOW);
  digitalWrite(led3, HIGH);
 }
 }
 else {
 Serial.println("error de mensaje");
 }
}

boolean reconexion() {
 Serial.print("Conectandoroker mqtt");
 if (client.connect("ESP_Banda")) {
 Serial.println("Conectado");
 //Publicación del estado "conectado"
 client.publish("/conexion", "Conectado");
 //Subscripción al topico de control
 client.subscribe("/banda");

 } else {
 Serial.print("fall=");
 Serial.print(client.state());
 }
 return client.connected();
}

void setup() {
 Serial.begin(115200);
 pinMode(led1, OUTPUT);
 pinMode(led3, OUTPUT);
 conectarAlWifi();
 //Servidor y puerto al que se debe conectar
 client.setServer(brokerMqtt, puertoMqtt);
 //Llamada la funcion de callback
 client.setCallback(callback);
}

void loop() {

 if (!client.connected()) {
 if (millis() - ultimoIntentoReconexion > 5000) {
 ultimoIntentoReconexion = millis();
 // Attempt to reconnect
 if (reconexion()) {
 ultimoIntentoReconexion = 0;
 }
 }
 } else {
 //cliente conectado
 if (millis() - timerEnvioDatos > 5000) {
 timerEnvioDatos=millis();
```

```
 //Envio de valores aleatorios
 char msg[4];
 snprintf (msg, 4, "%ld", random(100));
 client.publish("/banda/humedad", msg);
 }
 client.loop();
}
}
```

Configuración de la NodeMCU del proceso dos

```
/*************************************
 ** Librerias **
 *************************************/
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
/*************************************
 ** constantes **
 *************************************/
#define led2 D6
#define puertoMqtt 1883
/*************************************
 ** objetos y variables **
 *************************************/
WiFiClient clienteCilindro;
PubSubClient clienteMQTT(clienteCilindro);
const char * ssid = "Red_AP_dos";
const char * claveWifi = "ab12345678";
// ip del broker
const char * brokerMqtt = "192.168.3.129";
uint32_t ultimoIntentoReconexion;
uint32_t timerEnvioDatos;

void conectarAlWifi() {
 WiFi.begin(ssid, claveWifi);
 Serial.print("Conectando");
 Serial.println(ssid);
 //IP estatica
 IPAddress ip(192,168,3,75);
 IPAddress gateway(192,168,3,65);
 IPAddress subnet(255,255,255,0);
 WiFi.config(ip, gateway, subnet);

 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("");
 Serial.println("Wifi Conectado ");
 Serial.println("Direccion IP: ");
 Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* mensaje, unsigned int length) {
 String topico = topic;
 Serial.print("Mensaje recibido del topico: ");
 Serial.println(topico);
 Serial.print("Mensaje: ");
```

```
 Serial.println((char)mensaje[0]);
if (topico == "/cilindro") {
  if (mensaje[0]=='0') digitalWrite(led2, LOW);
  if (mensaje[0]=='1') {
    digitalWrite(led2, HIGH);
    delay(500);
    digitalWrite(led2, LOW);
    delay(500);
    digitalWrite(led2, HIGH);
  }
 }
 else {
 Serial.println("error de mensaje");
 }
}

boolean reconexion() {
 Serial.print("Conectandoroker mqtt");
 //intentando conectar al broker
 if (clienteMQTT.connect("ESP_Cilindro")) {
 Serial.println("Conectado");
 //Publicación del estado "conectado"
 clienteMQTT.publish("/conexion", "Conectado");
 //Subscripción al topico de control
 clienteMQTT.subscribe("/cilindro");
 } else {
 Serial.print("fall=");
 Serial.print(clienteMQTT.state());
 }
 return clienteMQTT.connected();
}

void setup() {
 Serial.begin(115200);
 pinMode(led2, OUTPUT);
 conectarAlWifi();
 //Servidor y puerto al que se debe conectar
 clienteMQTT.setServer(brokerMqtt, puertoMqtt);
 //Llama a la funcion de callback
 clienteMQTT.setCallback(callback);
}

void loop() {

 if (!clienteMQTT.connected()) {
 if (millis() - ultimoIntentoReconexion > 5000) {
 ultimoIntentoReconexion = millis();
 // Attempt to reconnect
 if (reconexion()) {
 ultimoIntentoReconexion = 0;
 }
 }
 } else {
 //cliente conectado
 if (millis() - timerEnvioDatos > 5000) {
 timerEnvioDatos=millis();
 //Envio de valores aleatorios
 char msg[4];
 snprintf (msg, 3, "%ld", random(40));
 clienteMQTT.publish("/cilindro/temperatura", msg);
```

```
 }
 clienteMQTT.loop();
}
}
```

ANEXO B: Código fuente para la implementación de la cámara IP en la Raspberry Pi

Script appCam.py

```
from flask import Flask, render_template, Response

# Raspberry Pi camera module (requires picamera package)
from camera_pi import Camera

app = Flask(__name__)

@app.route('/')
def index():
    """Video streaming home page."""
    return render_template('index.html')


def gen(camera):
    """Video streaming generator function."""
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame +
b'\r\n')


@app.route('/video_feed')
def video_feed():
    """Video streaming route. Put this in the src attribute of an
img tag."""
    return Response(gen(Camera()),
                    mimetype='multipart/x-mixed-replace;
boundary=frame')


if __name__ == '__main__':
    app.run(host='0.0.0.0', port =80, debug=True, threaded=True)
```

Script camera_pi.py

```
import time
import io
import threading
import picamera

class Camera(object):
    # background thread that reads frames from camera
    thread = None
    # current frame is stored here by background thread
    frame = None
    last_access = 0  # time of last client access to the camera

    def initialize(self):
        if Camera.thread is None:
```

```
            # start background frame thread
            Camera.thread = threading.Thread(target=self._thread)
            Camera.thread.start()

            # wait until frames start to be available
            while self.frame is None:
                time.sleep(0)

    def get_frame(self):
        Camera.last_access = time.time()
        self.initialize()
        return self.frame

    @classmethod
    def _thread(cls):
        with picamera.PiCamera() as camera:
            # camera setup
            camera.resolution = (320, 240)
            camera.hflip = True
            camera.vflip = True

            # let camera warm up
            camera.start_preview()
            time.sleep(2)

            stream = io.BytesIO()
            for foo in camera.capture_continuous(stream, 'jpeg',
                                                 use_video_port=True):
                # store frame
                stream.seek(0)
                cls.frame = stream.read()

                # reset stream for next frame
                stream.seek(0)
                stream.truncate()

                # if there hasn't been any clients asking for
                # frames in the last 10 seconds stop the thread
                if time.time() - cls.last_access > 10:
                    break
        cls.thread = None
```

Script index.html

```
<html style="background: linear-gradient(to right,#C69B93,
#867E7E);">
  <head>
    <title>Proyecto de Titulación</title>
  </head>
  <body>
    <h3><img src="{{ url_for('video_feed') }}" width="50%"></h3>
  </body>
</html>
```

ANEXO C: Código fuente para la Configuración del Open vSwitch en el conmutador OpenFlow

```
#!/bin/bash
ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
    --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
    --private-key=db:Open_vSwitch,SSL,private_key \
    --certificate=db:Open_vSwitch,SSL,certificate \
    --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
    --pidfile --detach
ovs-vsctl --no-wait init
ovs-vswitchd --pidfile --detach
ovs-vsctl show
ovs-vsctl --version
ps -e | grep ovs
ovs-vsctl set bridge br0 protocols=OpenFlow13
ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000001
```

ANEXO D: Código fuente para la Configuración del Open vSwitch en los Access Point OpenFlow

Script AP1_start.sh para la Configuración del Access Point 1

```
#!/bin/sh
#Setup variables
#My IP address is required for the ovsdb server.
MYIP=192.168.2.4

# This is the OpenFlow controller ID which we're going to load into
the OVS
CTLIP=192.168.2.2

# This is our DataPath ID
DPID=0000000000000002

# This is the name of the bridge that we're going to be creating
SW=br0

#What ports are we going to put in the OVS?
DPPORTS="wlan0 eth1"

#Alias some variables
VSCTL="ovs-vsctl --db=tcp:$MYIP:9999"
OVSDB=/tmp/ovs-vswitchd.conf.db

# Subroutine to wait until a port is ready
wait_port_listen() {
    port=$1
    while ! `netstat -na | grep $port` ; do
        echo -n .
        sleep 1
    done
}

# Kill off the servers and remove any stale lockfiles
/usr/bin/killall ovsdb-server
/usr/bin/killall ovs-vswitchd
rm /tmp/.ovs-vswitchd.conf.db.~lock~

# Remove the OVS Database and then recreate.
rm -f $OVSDB
ovsdb-tool create $OVSDB /usr/share/openvswitch/vswitch.ovsschema

# Start the OVSDB server and wait until it starts
ovsdb-server $OVSDB --remote=ptcp:9999:$MYIP &
#wait_port_listen 9999
sleep 5

# Start vSwitchd
ovs-vswitchd tcp:$MYIP:9999 --pidfile=ovs-vswitchd.pid --overwrite-
pidfile -- &

# Create the bridge and pass in some configura tion options
$VSCTL add-br $SW
$VSCTL set bridge $SW protocols=OpenFlow13
```

```
#Cycle through the DataPath ports adding them to the switch
for i in $DPPORTS ; do
    PORT=$i
        ifconfig $PORT up
    $VSCTL add-port $SW $PORT
done

#Ensure that the switch has the correct DataPath ID
$VSCTL set bridge $SW other-config:datapath-id=$DPID

#Configure the switch to have an OpenFlow Controller.  This will
contact the controller.
$VSCTL set-controller $SW tcp:$CTLIP:6633
# Turn off the fail-safe mode
$VSCTL set-fail-mode br0 secure
```

Script AP2_start.sh para la Configuración del Access Point 2

```
#!/bin/sh
#Setup variables
#My IP address is required for the ovsdb server.
MYIP=192.168.2.5

# This is the OpenFlow controller ID which we're going to load into
the OVS
CTLIP=192.168.2.2

# This is our DataPath ID
DPID=0000000000000003

# This is the name of the bridge that we're going to be creating
SW=br0

#What ports are we going to put in the OVS?
DPPORTS="wlan0 eth1"

#Alias some variables
VSCTL="ovs-vsctl --db=tcp:$MYIP:9999"
OVSDB=/tmp/ovs-vswitchd.conf.db

# Subroutine to wait until a port is ready
wait_port_listen() {
    port=$1
    while ! `netstat -na | grep $port` ; do
        echo -n .
        sleep 1
    done
}

# Kill off the servers and remove any stale lockfiles
/usr/bin/killall ovsdb-server
/usr/bin/killall ovs-vswitchd
rm /tmp/.ovs-vswitchd.conf.db.~lock~

# Remove the OVS Database and then recreate.
rm -f $OVSDB
ovsdb-tool create $OVSDB /usr/share/openvswitch/vswitch.ovsschema
```

```
# Start the OVSDB server and wait until it starts
ovsdb-server $OVSDB --remote=ptcp:9999:$MYIP &
#wait_port_listen 9999
sleep 5

# Start vSwitchd
ovs-vswitchd tcp:$MYIP:9999 --pidfile=ovs-vswitchd.pid --overwrite-
pidfile -- &

# Create the bridge and pass in some configura tion options
$VSCTL add-br $SW
$VSCTL set bridge $SW protocols=OpenFlow13

#Cycle through the DataPath ports adding them to the switch
for i in $DPPORTS ; do
    PORT=$i
        ifconfig $PORT up
    $VSCTL add-port $SW $PORT
done

#Ensure that the switch has the correct DataPath ID
$VSCTL set bridge $SW other-config:datapath-id=$DPID

#Configure the switch to have an OpenFlow Controller.  This will
contact the controller.
$VSCTL set-controller $SW tcp:$CTLIP:6633
# Turn off the fail-safe mode
$VSCTL set-fail-mode br0 secure
```

ANEXO E: Código fuente de la aplicación SDN

```
import logging
import json
from webob import Response
from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import WSGIApplication
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller import dpset
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.exception import OFPUnknownVersion
from ryu.lib import mac
from ryu.lib import dpid as dpid_lib
from ryu.lib import ofctl_v1_3
from ryu.ofproto import ether
from ryu.ofproto import inet
from ryu.ofproto import ofproto_v1_3
from ryu.ofproto import ofproto_v1_3_parser
LOG = logging.getLogger('ryu.app.firewall')
PRIORIDAD_ESTADO_FLUJO = ofproto_v1_3_parser.UINT16_MAX
PRIORIDAD_FLUJO_ARP = ofproto_v1_3_parser.UINT16_MAX - 1
PRIORIDAD_FLUJO_ACL_MAX = ofproto_v1_3_parser.UINT16_MAX - 2
"""Clase para inicializar las demas clases, es la primera clase
que se instancia en la aplicacion en RYU"""
class RestFirewallAPI(app_manager.RyuApp):
    #Lista que contiene las version del protocolo OpenFlow v.1.3)
    VERSIONES_OF = [ofproto_v1_3.OFP_VERSION]
    """Se define los contextos para poder utilizar objetos de la
clase DPSet y
    de la clase WSGIApplication"""
    _CONTEXTS = {'dpset': dpset.DPSet,
                 'wsgi': WSGIApplication}
    """Se definen en el constructor,  los contextos "wsgi" y "dpset"
a utilizarse
    en el resto de la aplicacion"""
    def __init__(self, *args, **kwargs):
        super(RestFirewallAPI, self).__init__(*args, **kwargs)
        self.dpset = kwargs['dpset']
        wsgi = kwargs['wsgi']
        """Se crea dos diccionarios para y se la da valores a sus
respectivas llaves 'dpset'
        y 'waiters' en funcion del contexto dpset"""
        self.waiters = {}
        self.data = {}
        self.data['dpset'] = self.dpset
        self.data['waiters'] = self.waiters
        """El constructor obtiene la instancia de la aplicacion WSGI
para registrarla en la
        clase ControladorFirewall en funcion del diccionario 'data'
"""
        wsgi.registory['ControladorFirewall'] = self.data
        #Definicion de la ruta donde se empezara a recibir las
peticiones HTTP
        ruta = '/firewall'
        """Creaccion de un diccionario que contendra los sw en
funcion de su id de 16 caracteres
        alfanumericos"""
```

```
        id_dp = {'switchid': dpid_lib.DPID_PATTERN + r'|TODOS'}
        #Creacion de la URI para obtener el(los) estado(s) del(os)
switch(es) conectados al controlador
        uri = ruta + '/module/status'
        """Establecimiento de una conexion entre las peticiones HTTP
y la aplicacion con el fin
        de obtener el(los) estado(s) del(os) switch(es) conectados
al controlador"""
        wsgi.mapper.connect('firewall', uri,
                            controller=ControladorFirewall,
action='estado_sw',
                            conditions=dict(method=['GET']))
        #Creacion de la URI para habilitar el firewall sobre un
switch a partir de su id
        uri = ruta + '/module/enable/{switchid}'
        """Establecimiento de una conexion entre las peticiones HTTP
y la aplicacion con el fin
        de habilitar el firewall sobre un switch a partir de su
id"""
        wsgi.mapper.connect('firewall', uri,
                            controller=ControladorFirewall,
action='habilitar_sw',
                            conditions=dict(method=['PUT']),
                            requirements=id_dp)
        #Creacion de la URI para deshabilitar el firewall sobre un
switch a partir de su id
        uri = ruta + '/module/disable/{switchid}'
        """Establecimiento de una conexion entre las peticiones HTTP
y la aplicacion con el fin
        de deshabilitar el firewall sobre un switch a partir de su
id"""
        wsgi.mapper.connect('firewall', uri,
                            controller=ControladorFirewall,
action='deshabilitar_sw',
                            conditions=dict(method=['PUT']),
                            requirements=id_dp)
        """Creacion de la URI para obtener, establecer o eliminar
las reglas de envio de trafico de un
        switch a partir de su id"""
        uri = ruta + '/reglas/{switchid}'
        """Establecimiento de una conexion entre las peticiones HTTP
y la aplicacion con el fin
        de obtener las reglas de envio de trafico de un switch a
partir de su id"""
        wsgi.mapper.connect('firewall', uri,
                            controller=ControladorFirewall,
action='reglas_fw',
                            conditions=dict(method=['GET']),
                            requirements=id_dp)
        """Establecimiento de una conexion entre las peticiones HTTP
y la aplicacion con el fin
        de establecer las reglas de envio de trafico de un switch a
partir de su id"""
        wsgi.mapper.connect('firewall', uri,
                            controller=ControladorFirewall,
action='ingresar_regla',
                            conditions=dict(method=['POST']),
                            requirements=id_dp)
        """Establecimiento de una conexion entre las peticiones HTTP
y la aplicacion con el fin
```

```
        de eliminar las reglas de envio de trafico de un switch a
partir de su id"""
        wsgi.mapper.connect('firewall', uri,
                            controller=ControladorFirewall,
action='borrar_regla',
                            conditions=dict(method=['DELETE']),
                            requirements=id_dp)
    """Se define una funcion que maneja el evento del mensaje
OpenFlow "ofpstatsreply"
    generado por RYU  y lo delega a la superclase RyuApp"""
    def stats_reply_handler(self, ev):
        #Se instancia el objeto msg que representara los mensajes
del evento mencionado
        msg = ev.msg
        #Se instancia el objeto dp que representara los mensajes
sobre el datapath del evento mencionado
        dp = msg.datapath
        """Se crea una lista a partir de los identificadores del
datapath(switch) y de los
        identificadores de las peticiones"""
        if dp.id not in self.waiters:
            return
        if msg.xid not in self.waiters[dp.id]:
            return
        lock, msgs = self.waiters[dp.id][msg.xid]
        msgs.append(msg)
        """Se crea una variable para delimitar el envio de mensajes
OpenFlow considerando las versiones del
        protoooloo OpenFlow 1.0, 1.2 y 1.3"""
        flags = 0
        flags = dp.ofproto.OFPMPF_REPLY_MORE
        """Compara el delimitador actual con el preestablecido del
envio de mensajes OpenFlow
        y si coinciden procede a borrar ese elemento de la lista a
partir de su id de datapath
        ademas bloquea o detiene el envio de mensajes """
        if msg.flags & flags:
            return
        del self.waiters[dp.id][msg.xid]
        lock.set()
    """Maneja los eventos de conexion o desconexion de un switch y
los delega a una instancia
    de la clase ControladorFirewall"""
    @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
    def handler_datapath(self, ev):
        if ev.enter:
            ControladorFirewall.registrar_sw(ev.dp)
        else:
            ControladorFirewall.desregistrar_sw(ev.dp)
    """Parametriza el manejo del evento con la funcion
stats_reply_handler para el caso
    de respuestas de estado de flujo utilizando un decorador"""
    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_FSR(self, ev):
        self.stats_reply_handler(ev)
    """Parametriza el manejo del evento con la funcion
stats_reply_handler para el caso
    de respuestas de estado utilizando un decorador"""
    @set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_SR(self, ev):
```

```
        self.stats_reply_handler(ev)
#Clase que contendra un diccionario con los switches conectados a la
aplicacion
class ListaFirewalls(dict):
    def __init__(self):
        super(ListaFirewalls, self).__init__()
    """Metodo que retornara la instancia del datapath a partir de su
identificador"""
    def lista_sw(self, dp_id):
        #Si el identificador de switch es nulo salta una execpsion
de "switch no conectado"
        if len(self) == 0:
            raise ValueError('switch no conectado.')
        dps = {}
       #Si el identificador de switch es TODOS, las acciones se
realizaran para todos los switches
        if dp_id == 'TODOS':
            dps = self
        else:
            try:
                dpid = dpid_lib.str_to_dpid(dp_id)
            #Si el identificador de switch posee una sintaxis
incorrecta salta una excepcion de "switch invalido"
            except:
                raise ValueError('ID del switch invalido.')
            #Si el switch no se encuentra conectado a la aplicacion
salta una excpecion de "sw no conectado al firewall"
            if dpid in self:
                dps = {dpid: self[dpid]}
            else:
                msg = 'firewall sw no conectado. : switchID=%s' %
dp_id
                raise ValueError(msg)
        return dps
#Clase heredada de la clase ControllerBase
class ControladorFirewall(ControllerBase):
    #Se crea una lista a partir de instaciar la clase ListaFirewalls
    _LISTA_SW = ListaFirewalls()
    """Se define el constructor de la clase y se declara los valores
para los datos para la
    aplicacion WSGI"""
    def __init__(self, req, link, data, **config):
        super(ControladorFirewall, self).__init__(req, link, data,
**config)
        self.dpset = data['dpset']
        self.waiters = data['waiters']
    #Metodo estatico para registrar los switches a partir de los
datapath
    @staticmethod
    def registrar_sw(dp):
        """Bloque para controlar los errores de utilizacion de un
switch con una version incompatible
        a traves del manejo de excepciones"""
        try:
            f_ofs = Firewall(dp)
        except OFPUnknownVersion, message:
            mes = 'dpid=%s : %s' % (dpid_lib.dpid_to_str(dp.id),
message)
            LOG.info(mes)
            return
```

```
        #Establece un valor por defecto a la clase
        ControladorFirewall._LISTA_SW.setdefault(dp.id, f_ofs)
        f_ofs.desactivar_flujo()
        f_ofs.activar_flujo_arp()
        LOG.info('dpid=%s : Switch conectado.' %
                 dpid_lib.dpid_to_str(dp.id))
    #Metodo estatico para eliminar  los switches registrados a
partir de los datapath
    @staticmethod
    def desregistrar_sw(dp):
        if dp.id in ControladorFirewall._LISTA_SW:
            del ControladorFirewall._LISTA_SW[dp.id]
            LOG.info('dpid=%s : Switch desconectado.' %
                     dpid_lib.dpid_to_str(dp.id))
    """Metodo que conecta las acciones que se deben realizar con la
API REST para
    obtener el estado de el(los) estado(s) del(os) switch(es)
conectados al controlador"""
    def estado_sw(self, req, **_kwargs):
        try:
            dps = self._LISTA_SW.lista_sw('TODOS')
        except ValueError, message:
            return Response(status=400, body=str(message))
        msgs = {}
        for f_ofs in dps.values():
            status = f_ofs.estado_sw(self.waiters)
            msgs.update(status)
        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)
    """Metodo que conecta las acciones que se deben realizar con la
API REST para
    habilitar el firewall sobre un switch a partir de su id"""
    def habilitar_sw(self, req, switchid, **_kwargs):
        try:
            dps = self._LISTA_SW.lista_sw(switchid)
        except ValueError, message:
            return Response(status=400, body=str(message))
        msgs = {}
        for f_ofs in dps.values():
            msg = f_ofs.activar_flujo()
            msgs.update(msg)
        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)
    """Metodo que conecta las acciones que se deben realizar con la
API REST para
    deshabilitar el firewall sobre un switch a partir de su id"""
    def deshabilitar_sw(self, req, switchid, **_kwargs):
        try:
            dps = self._LISTA_SW.lista_sw(switchid)
        except ValueError, message:
            return Response(status=400, body=str(message))
        msgs = {}
        for f_ofs in dps.values():
            msg = f_ofs.desactivar_flujo()
            msgs.update(msg)
        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)
    """Metodo que conecta las acciones que se deben realizar con la
API REST para
```

```
    obtener las reglas de envio de trafico de un switch a partir de
su id"""
    def reglas_fw(self, req, switchid, **_kwargs):
        try:
            dps = self._LISTA_SW.lista_sw(switchid)
        except ValueError, message:
            return Response(status=400, body=str(message))
        msgs = {}
        for f_ofs in dps.values():
            reglas = f_ofs.reglas_fw(self.waiters)
            msgs.update(reglas)
        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)
    """Metodo que conecta las acciones que se deben realizar con la
API REST para
    establecer las reglas de envio de trafico de un switch a partir
de su id"""
    def ingresar_regla(self, req, switchid, **_kwargs):
        try:
            regla = eval(req.body)
        except SyntaxError:
            LOG.debug('Sintaxis no valida %s', req.body)
            return Response(status=400)
        try:
            dps = self._LISTA_SW.lista_sw(switchid)
        except ValueError, message:
            return Response(status=400, body=str(message))
        msgs = {}
        for f_ofs in dps.values():
            try:
                msg = f_ofs.ingresar_regla(f_ofs.get_cookie(),
regla)
                msgs.update(msg)
            except ValueError, message:
                return Response(status=400, body=str(message))
        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)
    """Funcion que conecta las acciones que se deben realizar con la
API REST para
    eliminar las reglas de envio de trafico de un switch a partir de
su id"""
    def borrar_regla(self, req, switchid, **_kwargs):
        try:
            ruleid = eval(req.body)
        except SyntaxError:
            LOG.debug('Sintaxis no valida %s', req.body)
            return Response(status=400)
        try:
            dps = self._LISTA_SW.lista_sw(switchid)
        except ValueError, message:
            return Response(status=400, body=str(message))
        msgs = {}
        for f_ofs in dps.values():
            try:
                msg = f_ofs.borrar_regla(ruleid, self.waiters)
                msgs.update(msg)
            except ValueError, message:
                return Response(status=400, body=str(message))
        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)
```

```
#Clase que contiene la funcionalidad para la escritura de flujos en
un switch
class Firewall(object):
    _OFCTL = {ofproto_v1_3.OFP_VERSION: ofctl_v1_3}
    def __init__(self, dp):
        super(Firewall, self).__init__()
        self.dp = dp
        version = dp.ofproto.OFP_VERSION
        self.cookie = 0
        #Maneja el error de version OpenFlow no compatible en un
switch a traves de una excepcion
        if version not in self._OFCTL:
            raise OFPUnknownVersion(version=version)
        self.ofctl = self._OFCTL[version]
    def get_cookie(self):
        self.cookie += 1
        self.cookie &= ofproto_v1_3_parser.UINT64_MAX
        return self.cookie
    #Funcion que obtiene el estado del switch
    def estado_sw(self, waiters):
        msgs = self.ofctl.get_flow_stats(self.dp, waiters)
        status = 'enable'
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                if flow_stat['priority'] == PRIORIDAD_ESTADO_FLUJO:
                    status = 'disable'
        msg = {'status': status}
        switch_id = '%s: %s' % ('switch_id',
                                dpid_lib.dpid_to_str(self.dp.id))
        return {switch_id: msg}
    #Funcion que desactiva el envio de flujos en un switch
    def desactivar_flujo(self):
        cookie = 0
        prioridad = PRIORIDAD_ESTADO_FLUJO
        match = {}
        actions = []
        flow = self._conv_a_flujo(cookie=cookie, priority=prioridad,
                                  match=match, actions=actions)
        cmd = self.dp.ofproto.OFPFC_ADD
        self.ofctl.mod_flow_entry(self.dp, flow, cmd)
        msg = {'resultado': 'correcto',
               'detalles': 'firewall detenido.'}
        switch_id = '%s: %s' % ('switch_id',
                                dpid_lib.dpid_to_str(self.dp.id))
        return {switch_id: msg}
    #Funcion que activa  el envio de flujos en un switch
    def activar_flujo(self):
        cookie = 0
        prioridad = PRIORIDAD_ESTADO_FLUJO
        match = {}
        actions = []
        flow = self._conv_a_flujo(cookie=cookie, priority=prioridad,
                                  match=match, actions=actions)
        cmd = self.dp.ofproto.OFPFC_DELETE_STRICT
        self.ofctl.mod_flow_entry(self.dp, flow, cmd)
        msg = {'resultado': 'correcto',
               'detalles': 'firewall ejecutandose.'}
        switch_id = '%s: %s' % ('switch_id',
                                dpid_lib.dpid_to_str(self.dp.id))
```

```
        return {switch_id: msg}
    #Funcion para establecer los flujos ARP en un switch
    def activar_flujo_arp(self):
        cookie = 0
        prioridad = PRIORIDAD_FLUJO_ARP
        match = {'dl_type': ether.ETH_TYPE_ARP}
        action = {'actions': 'ACEPTAR'}
        actions = Accion.conv_a_accion_of(self.dp, action)
        flow = self._conv_a_flujo(cookie=cookie, priority=prioridad,
                                  match=match, actions=actions)
        cmd = self.dp.ofproto.OFPFC_ADD
        self.ofctl.mod_flow_entry(self.dp, flow, cmd)
    #Funcion que establece una regla de envio de trafico del
firewall en un switch
    def ingresar_regla(self, cookie, rest):
        prioridad = int(rest.get('prioridad', 0))
        if prioridad < 0 or PRIORIDAD_FLUJO_ACL_MAX < prioridad:
            raise ValueError('Valor no valido de prioridad. Ingrese
de [0-%d]'
                             % PRIORIDAD_FLUJO_ACL_MAX)
        match = Match.conv_a_accion_of(rest)
        actions = Accion.conv_a_accion_of(self.dp, rest)
        flow = self._conv_a_flujo(cookie=cookie, priority=prioridad,
                                  match=match, actions=actions)
        cmd = self.dp.ofproto.OFPFC_ADD
        try:
            self.ofctl.mod_flow_entry(self.dp, flow, cmd)
        except:
            raise ValueError('Parametro de regla invalido.')
        msg = {'resultado': 'correcto',
               'detalles': 'Regla ingresada. : regla_id=%d' %
cookie}
        switch_id = '%s: %s' % ('switch_id',
                                dpid_lib.dpid_to_str(self.dp.id))
        return {switch_id: msg}
    #Funcion que obtiene las reglas de envio de trafico del firewall
en un switch
    def reglas_fw(self, waiters):
        reglas = {}
        msgs = self.ofctl.get_flow_stats(self.dp, waiters)
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                if (flow_stat['priority'] != PRIORIDAD_ESTADO_FLUJO
                        and flow_stat['priority'] !=
PRIORIDAD_FLUJO_ARP):
                    regla = self._conv_a_regla_rest(flow_stat)
                    reglas.update(regla)
        get_data = []
        for regla in reglas.items():
            get_data.append(regla)
        switch_id = '%s: %s' % ('switch_id',

dpid_lib.dpid_to_str(self.dp.id)+"@")
        return {switch_id: get_data}
     #Funcion que elimina una regla de envio de trafico del firewall
en un switch
    def borrar_regla(self, rest, waiters):
        try:
            if rest['regla_id'] == 'TODOS':
```

```
                regla_id = 'TODOS'
            else:
                regla_id = int(rest['regla_id'])
        except:
            raise ValueError('ID de regla invalido.')
        lista_eliminada = []
        msgs = self.ofctl.get_flow_stats(self.dp, waiters)
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                cookie = flow_stat['cookie']
                prioridad = flow_stat['priority']
                if (prioridad != PRIORIDAD_ESTADO_FLUJO
                        and prioridad != PRIORIDAD_FLUJO_ARP):
                    if regla_id == 'TODOS' or regla_id == cookie:
                        match =
Match.conv_a_borrar_of(flow_stat['match'])
                        lista_eliminada.append([cookie, prioridad,
match])
                    if regla_id == cookie:
                        break
        if len(lista_eliminada) == 0:
            msj_detalles = 'Regla no existe.'
            if regla_id != 'TODOS':
                msj_detalles += ' : ruleID=%d' % regla_id
            msg = {'resultado': 'falla',
                   'detalles': msj_detalles}
        else:
            cmd = self.dp.ofproto.OFPFC_DELETE_STRICT
            actions = []
            msj_detalles = 'Regla eliminada. : ruleID='
            for cookie, priority, match in lista_eliminada:
                flow = self._conv_a_flujo(cookie=cookie,
priority=prioridad,
                                          match=match,
actions=actions)
                self.ofctl.mod_flow_entry(self.dp, flow, cmd)
                msj_detalles += '%d,' % cookie
            msg = {'resultado': 'correcto',
                   'detalles': msj_detalles}
        switch_id = '%s: %s' % ('switch_id',
                                dpid_lib.dpid_to_str(self.dp.id))
        return {switch_id: msg}
    #Funcion que retorna un flujo a partir de las acciones y el
proceso de emparejamiento
    def _conv_a_flujo(self, cookie, priority, match, actions):
        flow = {'cookie': cookie,
                'priority': priority,
                'flags': 0,
                'idle_timeout': 0,
                'hard_timeout': 0,
                'match': match,
                'actions': actions}
        return flow
    #Funcion que retorna una regla para enviar a la REST API a
partir de los flujos
    def _conv_a_regla_rest(self, flow):
        regla_id = '%s: %d' % ('regla_id', flow['cookie'])
        regla = {'prioridad': flow['priority']}
        regla.update(Match.conv_a_comando_rest(flow))
```

```
        regla.update(Accion.conv_a_comando_rest(flow))
        return {regla_id : regla}
"""Clase que realiza las conversiones entre las distintas reglas de
la REST API
 y las reglas de reenvio de trafico OpenFlow"""
class Match(object):
    """Se crea un diccionario con los protocolos que soportara el
firewall
    para el establecimiento de reglas de reenvio de trafico"""
    _CONVERT = {'dl_type':
                {'ARP': ether.ETH_TYPE_ARP,
                 'IPv4': ether.ETH_TYPE_IP},
                'nw_proto':
                {'TCP': inet.IPPROTO_TCP,
                 'UDP': inet.IPPROTO_UDP,
                 'ICMP': inet.IPPROTO_ICMP}}
    """Metodo estatico para convertir de una regla de la REST API a
una regla de reenvio
    de trafico OpenFlow a partir de los protocolos que intervedran
en el trafico de la red"""
    @staticmethod
    def conv_a_accion_of(rest):
        match = {}
        set_dltype_flg = False
        for key, value in rest.items():
            if (key == 'nw_src' or key == 'nw_dst'
                    or key == 'nw_proto'):
                if ('dl_type' in rest) is False:
                    set_dltype_flg = True
                elif (rest['dl_type'] != 'IPv4'
                        and rest['dl_type'] != 'ARP'):
                    continue
            elif key == 'tp_src' or key == 'tp_dst':
                if (('nw_proto' in rest) is False
                    or (rest['nw_proto'] != 'TCP'
                        and rest['nw_proto'] != 'UDP')):
                    continue
            if key in Match._CONVERT:
                if value in Match._CONVERT[key]:
                    match.setdefault(key,
Match._CONVERT[key][value])
                else:
                    raise ValueError('Parametro de regla invalido. :
key=%s' % key)
            else:
                match.setdefault(key, value)
            if set_dltype_flg:
                match.setdefault('dl_type', ether.ETH_TYPE_IP)
        return match
    """Metodo estatico para convertir de una regla de reenvio de
trafico OpenFlow
    a una regla de la REST API a partir de los protocolos que
intervendran en el trafico de la red"""
    @staticmethod
    def conv_a_comando_rest(openflow):
        of_match = openflow['match']
        match = {}
        for key, value in of_match.items():
            if key == 'dl_src' or key == 'dl_dst':
                if value == mac.haddr_to_str(mac.DONTCARE):
```

```
                    continue
                elif key == 'nw_src' or key == 'nw_dst':
                    if value == '0.0.0.0':
                        continue
                elif value == 0:
                    continue
                if key in Match._CONVERT:
                    conv = Match._CONVERT[key]
                    conv = dict((value, key) for key, value in
conv.items())
                    match.setdefault(key, conv[value])
                else:
                    match.setdefault(key, value)
        return match
    #Metodo estatico para eliminar reglas de reenvio de trafico """
    @staticmethod
    def conv_a_borrar_of(of_match):
        match = {}
        for key, value in of_match.items():
            if key == 'dl_src' or key == 'dl_dst':
                if value == mac.haddr_to_str(mac.DONTCARE):
                    continue
            elif key == 'nw_src' or key == 'nw_dst':
                if value == '0.0.0.0':
                    continue
            elif value == 0:
                continue
            match.setdefault(key, value)
        return match
"""Clase que realiza las conversiones entre las distintas acciones
de la REST API
 y las acciones  OpenFlow"""
class Accion(object):
    #Metodo estatico para convertir las acciones de la REST API a
una accion OpenFlow
    @staticmethod
    def conv_a_accion_of(dp, rest):
        value = rest.get('actions', 'ACEPTAR')
        if value == 'ACEPTAR':
            out_port = dp.ofproto.OFPP_NORMAL
            action = [{'type': 'OUTPUT',
                       'port': out_port}]
        elif value == 'RECHAZAR':
            action = []
        else:
            raise ValueError('Tipo de accion invalida.')
        return action
    #Metodo estatico para convertir las acciones OpenFlow a una
accion  de la REST API
    @staticmethod
    def conv_a_comando_rest(openflow):
        if 'actions' in openflow:
            if len(openflow['actions']) > 0:
                action = {'actions': 'ACEPTAR'}
            else:
                action = {'actions': 'RECHAZAR'}
        else:
            action = {'actions': 'Unknown action type.'}

        return action
```

ANEXO F: Código fuente de la interfaz gráfica de usuario para el control de la aplicación SDN

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated with glade 3.22.1 -->
<interface>
  <requires lib="gtk+" version="3.6"/>
  <!-- interface-css-provider-path styles.css -->
  <object class="GtkWindow" id="ventana">
    <property name="can_focus">False</property>
    <property name="default_width">441</property>
    <property name="default_height">501</property>
    <signal name="destroy" handler="on_ventana_destroy"
swapped="no"/>
    <child>
      <placeholder/>
    </child>
    <child>
      <object class="GtkFixed" id="fijo">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="margin_right">100</property>
        <property name="border_width">16</property>
        <child>
          <object class="GtkButton" id="botonVerReglas">
            <property name="label" translatable="yes"> Ver
reglas</property>
            <property name="width_request">79</property>
            <property name="height_request">47</property>
            <property name="can_focus">True</property>
            <property name="receives_default">True</property>
            <signal name="clicked"
handler="on_botonVerReglas_clicked" swapped="no"/>
            <style>
              <class name="boton"/>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">661</property>
            <property name="y">550</property>
          </packing>
        </child>
        <child>
          <object class="GtkEntry" id="entradaIPDestino">
            <property name="name">1</property>
            <property name="width_request">116</property>
            <property name="height_request">34</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="invisible_char">●</property>
            <property name="width_chars">14</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">850</property>
            <property name="y">200</property>
```

```
          </packing>
        </child>
        <child>
          <object class="GtkEntry" id="entradaIPFuente">
            <property name="name">1</property>
            <property name="width_request">116</property>
            <property name="height_request">34</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="tooltip_text" translatable="yes">bh h
hhhhhhhhhhh</property>
            <property name="invisible_char">●</property>
            <property name="width_chars">14</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">550</property>
            <property name="y">200</property>
          </packing>
        </child>
        <child>
          <object class="GtkEntry" id="entradaMACDestino">
            <property name="width_request">130</property>
            <property name="height_request">34</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="invisible_char">●</property>
            <property name="width_chars">14</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">850</property>
            <property name="y">235</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaMACFuente">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">MAC
Fuente</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">450</property>
            <property name="y">235</property>
          </packing>
```

```
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaMACDestino">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">MAC
Destino</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">750</property>
            <property name="y">235</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaIPFuente">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">IP
Fuente</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">450</property>
            <property name="y">200</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaIPDestino">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">IP
Destino</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
```

```
            </style>
          </object>
          <packing>
            <property name="x">750</property>
            <property name="y">200</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaProtocolo">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label"
translatable="yes">Protocolo</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">100</property>
            <property name="y">242</property>
          </packing>
        </child>
        <child>
          <object class="GtkEntry" id="entradaPuertoDestino">
            <property name="width_request">53</property>
            <property name="height_request">34</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="invisible_char">●</property>
            <property name="width_chars">5</property>
            <property name="max_width_chars">5</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">850</property>
            <property name="y">270</property>
          </packing>
        </child>
        <child>
          <object class="GtkEntry" id="entradaPuertoFuente">
            <property name="width_request">53</property>
            <property name="height_request">34</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="invisible_char">●</property>
            <property name="width_chars">5</property>
            <property name="max_width_chars">5</property>
            <property name="input_purpose">digits</property>
            <style>
              <class name="etiqueta"/>
            </style>
```

```
          </object>
          <packing>
            <property name="x">550</property>
            <property name="y">270</property>
          </packing>
        </child>
        <child>
          <object class="GtkButton" id="botonIngresarReglas">
            <property name="label" translatable="yes">Ingresar
  Regla</property>
            <property name="width_request">99</property>
            <property name="height_request">47</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="receives_default">True</property>
            <signal name="clicked"
handler="on_botonIngresarReglas_clicked" swapped="no"/>
            <style>
              <class name="boton"/>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">321</property>
            <property name="y">375</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaPuertoFuente">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">Puerto
Fuente</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">450</property>
            <property name="y">270</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaPuertoDestino">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes"> Puerto
Destino</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
```

```
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">750</property>
            <property name="y">270</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaAccion">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label"
translatable="yes">Accion</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">100</property>
            <property name="y">274</property>
          </packing>
        </child>
        <child>
          <object class="GtkComboBox" id="comboboxProtocolo">
            <property name="width_request">69</property>
            <property name="height_request">29</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">200</property>
            <property name="y">242</property>
          </packing>
        </child>
        <child>
          <object class="GtkComboBox" id="comboboxAccion">
            <property name="width_request">97</property>
            <property name="height_request">29</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">200</property>
```

```
            <property name="y">274</property>
          </packing>
        </child>
        <child>
          <object class="GtkButton" id="botonBorrarReglas">
            <property name="label" translatable="yes">Borrar
Regla</property>
            <property name="width_request">83</property>
            <property name="height_request">47</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="receives_default">True</property>
            <signal name="clicked"
handler="on_botonBorrarReglas_clicked" swapped="no"/>
            <style>
              <class name="boton"/>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">661</property>
            <property name="y">375</property>
          </packing>
        </child>
        <child>
          <object class="GtkComboBox" id="comboboxInsertarIdSwitch">
            <property name="width_request">161</property>
            <property name="height_request">29</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">200</property>
            <property name="y">210</property>
          </packing>
        </child>
        <child>
          <object class="GtkButton" id="botonActualizarRed">
            <property name="label"
translatable="yes">Refresh</property>
            <property name="width_request">106</property>
            <property name="height_request">47</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="receives_default">True</property>
            <property name="image_position">right</property>
            <signal name="clicked"
handler="on_botonActualizarRed_clicked" swapped="no"/>
            <style>
              <class name="etiqueta"/>
              <class name="boton"/>
            </style>
          </object>
          <packing>
            <property name="x">6</property>
            <property name="y">67</property>
          </packing>
```

```
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaInsertarIdSwitch">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="app_paintable">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">Id del
switch</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">100</property>
            <property name="y">210</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaVerIdSwitch">
            <property name="width_request">100</property>
            <property name="height_request">22</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">Id del
switch</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">11</property>
            <property name="y">367</property>
          </packing>
        </child>
        <child>
          <object class="GtkComboBox" id="comboboxVerIdSwitch">
            <property name="width_request">161</property>
            <property name="height_request">29</property>
            <property name="app_paintable">True</property>
            <property name="can_focus">False</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">120</property>
            <property name="y">361</property>
          </packing>
        </child>
```

```
        <child>
          <object class="GtkSwitch" id="interruptorTrafico">
            <property name="width_request">90</property>
            <property name="height_request">27</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <style>
              <class name="boton_sw"/>
            </style>
          </object>
          <packing>
            <property name="x">200</property>
            <property name="y">180</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaTrafico">
            <property name="width_request">110</property>
            <property name="height_request">28</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label"
translatable="yes">Desactivado</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Italic
12"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">100</property>
            <property name="y">180</property>
          </packing>
        </child>
        <child>
          <object class="GtkEntry" id="entradaMACFuente">
            <property name="width_request">130</property>
            <property name="height_request">34</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="width_chars">14</property>
            <property name="max_width_chars">17</property>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">550</property>
            <property name="y">235</property>
          </packing>
        </child>
        <child>
          <object class="GtkScrolledWindow" id="scrollReglas">
            <property name="width_request">860</property>
            <property name="height_request">136</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
```

```
            <property name="kinetic_scrolling">False</property>
            <child>
              <object class="GtkTreeView" id="vistaReglas">
                <property name="width_request">860</property>
                <property name="height_request">200</property>
                <property name="visible">True</property>
                <property name="can_focus">True</property>
                <child internal-child="selection">
                  <object class="GtkTreeSelection" id="treeview-
selection"/>
                </child>
                <style>
                  <class name="tr"/>
                  <class name="etiqueta"/>
                </style>
              </object>
            </child>
            <style>
              <class name="etiqueta"/>
            </style>
          </object>
          <packing>
            <property name="x">119</property>
            <property name="y">450</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="etiquetaTitulo">
            <property name="width_request">728</property>
            <property name="height_request">31</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">
PROYECTO DE TITULACIÓN
IoT BASADO EN REDES DEFINIDAS POR SOFTWARE
                                "APLICACIÓN FIREWALL"</property>
            <attributes>
              <attribute name="font-desc" value="FreeSerif Bold
Italic 16"/>
              <attribute name="foreground" value="#000000000000"/>
            </attributes>
            <style>
              <class name="titulo"/>
            </style>
          </object>
          <packing>
            <property name="x">179</property>
          </packing>
        </child>
        <style>
          <class name="fondo"/>
        </style>
      </object>
    </child>
    <style>
      <class name="fondo"/>
    </style>
  </object>
</interface>
```

ANEXO G: Código fuente para la inserción de acciones en la interfaz gráfica de usuario para el control de la aplicación SDN

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk,Gdk
import requests
import json
from ast import literal_eval
from collections import defaultdict
#Variables globales para manejar la seleccion de opciones, como
protocolo, reglas, etc.
global seleccprotocolo
global seleccaccion
global seleccregla
global seleccinsertaridswitch
global seleccveridswitch
#Creacion de las listas de reglas, para guardar y actualizar la
lista de reglas
listareglas=[]
listareglasactualizar=[]
#Asignar elementos por defecto a la listareglas
for item in listareglas:
    item = defaultdict(lambda: "", item)
    listareglas.append(item)
#Lista de almacenamiento de reglas para mostrar en la GUI
almacen = Gtk.ListStore(str, str, str, str, str, str, str, str, str)
#Lista de almacenamiento de ID de los switch para insertar en las
reglas en la GUI
listainsertaridswitch=Gtk.ListStore(str)
#Lista de almacenamiento de ID de los switch para ver las reglas en
la GUI
listaveridswitch=Gtk.ListStore(str)
#Metodo para reemplazo de caracteres en diccionarios
def remplazo_letras(text, dic):
 for i, j in dic.iteritems():
        text = text.replace(i, j)
 return text
#Metodo de fltrado de numeros en entradas de texto de la GUI
def numeros(widget):
     def filtrar_numeros(entry, *args):
            text = entry.get_text().strip()
            entry.set_text(''.join([i for i in text if i in
'0123456789']))
     widget.connect('changed', filtrar_numeros)
#Metodo de fltrado de direcciones IP en entradas de texto de la GUI
def dirip(widget):
     def filtrar_ip(entry, *args):
            text = entry.get_text().strip()
            entry.set_text(''.join([i for i in text if i in
'0123456789.']))
     widget.connect('changed', filtrar_ip)
#Metodo de fltrado de direcciones MAC en entradas de texto de la GUI
def dirmac(widget):
     def filtrar_mac(entry, *args):
            text = entry.get_text().strip()
            entry.set_text(''.join([i for i in text if i in
'0123456789abcdefABCDEF:']))
     widget.connect('changed', filtrar_mac)
```

```
#Definicion de la clase principal de la GUI
class ClienteFirewall:
 def __init__(self):
  #Definicion del objeto gui a partir de un archivo creado con glade
  gui= Gtk.Builder()
  gui.add_from_file("gui.xml" )
  #Obtencion del obejto ventana del archivo glade
  self.ventana = gui.get_object("ventana")
  #Establecer el titulo de la ventana
  self.ventana.set_title("Tesis")
  #Establecer la utilizacion del archivo css para dar estilos CSS a
los elementos de la GUI
  cssProvider = Gtk.CssProvider()
  cssProvider.load_from_path('styles.css')
  screen = Gdk.Screen.get_default()
  styleContext = Gtk.StyleContext()
  styleContext.add_provider_for_screen(screen, cssProvider,
Gtk.STYLE_PROVIDER_PRIORITY_USER)
  """Obtencion de los elementos de la GUI a partir del archivo
creado con glade"""
  self.fijo =gui.get_object("fijo" )
  self.interruptorTrafico = gui.get_object("interruptorTrafico")
  self.vistaReglas = gui.get_object("vistaReglas" )
  self.entradaMACFuente = gui.get_object("entradaMACFuente")
  self.entradaIPDestino = gui.get_object("entradaIPDestino")
  self.entradaIPFuente = gui.get_object("entradaIPFuente")
  self.entradaMACDestino = gui.get_object("entradaMACDestino")
  self.entradaPuertoDestino = gui.get_object("entradaPuertoDestino")
  self.entradaPuertoFuente = gui.get_object("entradaPuertoFuente")
  self.etiquetaTrafico= gui.get_object("etiquetaTrafico")
  self.comboboxProtocolo = gui.get_object("comboboxProtocolo")
  self.comboboxAccion = gui.get_object("comboboxAccion")
  self.comboboxInsertarIdSwitch =
gui.get_object("comboboxInsertarIdSwitch")
  self.comboboxVerIdSwitch = gui.get_object("comboboxVerIdSwitch")
  #Conectar los eventos de los elementos de la GUI con metodos que
se definan en esta aplicacion de Python
  gui.connect_signals(self)
  """Filtrar la entrada de texto de los campos de puertos,
direcciones IP y direcciones MAC"""
  numeros(self.entradaPuertoFuente)
  numeros(self.entradaPuertoDestino)
  dirip(self.entradaIPFuente)
  dirip(self.entradaIPDestino)
  dirmac(self.entradaMACFuente)
  dirmac(self.entradaMACDestino)
  #Poner las cabeceras de la lista de reglas de la GUI
  for i in listareglas:
        almacen.append([i['regla_id'], i['dl_src'], i['dl_dst'],
i['nw_src'], i['nw_dst'], i['nw_proto'],  i['tp_src'], i['tp_dst'],
i['actions']])
  #Establecer las columnas a la lista de reglas de la GUI
  column0 = Gtk.TreeViewColumn("Id Regla", Gtk.CellRendererText(),
text=0)
  self.vistaReglas.append_column(column0)
  column1 = Gtk.TreeViewColumn("MAC Fuente", Gtk.CellRendererText(),
text=1)
  self.vistaReglas.append_column(column1)
  column2 = Gtk.TreeViewColumn("MAC Destino",
Gtk.CellRendererText(), text=2)
```

```
  self.vistaReglas.append_column(column2)
  column3 = Gtk.TreeViewColumn("IP Fuente", Gtk.CellRendererText(),
text=3)
  self.vistaReglas.append_column(column3)
  column4 = Gtk.TreeViewColumn("IP Destino", Gtk.CellRendererText(),
text=4)
  self.vistaReglas.append_column(column4)
  column5 = Gtk.TreeViewColumn("Protocolo", Gtk.CellRendererText(),
text=5)
  self.vistaReglas.append_column(column5)
  column6 = Gtk.TreeViewColumn("Puerto Fuente",
Gtk.CellRendererText(), text=6)
  self.vistaReglas.append_column(column6)
  column7 = Gtk.TreeViewColumn("Puerto Destino",
Gtk.CellRendererText(), text=7)
  self.vistaReglas.append_column(column7)
  column8 = Gtk.TreeViewColumn("Accion", Gtk.CellRendererText(),
text=8)
  self.vistaReglas.append_column(column8)
  #Lista de almacenamiento de protocolos para insertar en las reglas
en la GUI
  listaproto=Gtk.ListStore(str)
  #Lista de almacenamiento de acciones para insertar en las reglas
en la GUI
  listaaccion=Gtk.ListStore(str)
  #Lista de almacenamiento de ID de los switch para insertar en las
reglas en la GUI
  listainsertaridswitch=Gtk.ListStore(str)
  #Lista de almacenamiento de ID de los switch para ver las reglas
en la GUI
  listaveridswitch=Gtk.ListStore(str)
  """Poner celdas en los comboboxes que contienen protocolos,
acciones e ID de switches de la GUI"""
  celdaprotocolo = Gtk.CellRendererText()
  self.comboboxProtocolo.pack_start(celdaprotocolo,  0)
  self.comboboxProtocolo.add_attribute(celdaprotocolo, 'text', 0)
  celdaaccion = Gtk.CellRendererText()
  self.comboboxAccion.pack_start(celdaaccion,  0)
  self.comboboxAccion.add_attribute(celdaaccion, 'text', 0)
  celdainsertaridswitch = Gtk.CellRendererText()
  self.comboboxInsertarIdSwitch.pack_start(celdainsertaridswitch,
0)
  self.comboboxInsertarIdSwitch.add_attribute(celdainsertaridswitch,
'text', 0)
  celdaveridswitch = Gtk.CellRendererText()
  self.comboboxVerIdSwitch.pack_start(celdaveridswitch, 0)
  self.comboboxVerIdSwitch.add_attribute(celdaveridswitch, 'text',
0)
  #Lista de protocolos para insertar en las reglas en la GUI
  listaproto.append(['TCP'])
  #listaproto.append(['UDP'])
  listaproto.append(['ICMP'])
  self.comboboxProtocolo.set_model(listaproto)
  #Lista de acciones para insertar en las reglas en la GUI
  listaaccion.append(['ACEPTAR'])
  listaaccion.append(['RECHAZAR'])
  self.comboboxAccion.set_model(listaaccion)
  """Actualizar lista de reglas en funcion de su ID de switch"""
  global listareglasactualizar
  for y in listareglasactualizar:
```

```
    listainsertaridswitch.append([y])
  for y in listareglasactualizar:
    listaveridswitch.append([y])
  self.comboboxInsertarIdSwitch.set_model(listainsertaridswitch)
  self.comboboxVerIdSwitch.set_model(listaveridswitch)
  self.vistaReglas.set_model(almacen)
  self.ventana.show()
  #Establecer los metodos para activar y desactivar todo el trafico
mediante el interruptor
  self.interruptorTrafico.connect("notify::active", self.on_toggled)
  self.interruptorTrafico.connect("notify::active",
self.on_toggled2)
  self.vistaReglas.show()
  self.vistaReglas.get_selection().connect("changed",
self.on_changed)
  """Establecer los metodos para seleccionar los protocolos,
acciones e ID  de switch mediante los comboboxes"""
  self.comboboxProtocolo.connect("changed",
self.on_comboboxProtocolo_changed)
  self.comboboxProtocolo.set_active(0)
  self.comboboxAccion.connect("changed",
self.on_comboboxAccion_changed)
  self.comboboxAccion.set_active(0)
  self.comboboxInsertarIdSwitch.connect("changed",
self.on_comboboxInsertarIdSwitch_changed)
  self.comboboxInsertarIdSwitch.set_active(0)
  self.comboboxVerIdSwitch.connect("changed",
self.on_comboboxVerIdSwitch_changed)
  self.comboboxVerIdSwitch.set_active(0)
 #Metodo para cambiar el texto a Activado/Desactivado cuando se
accione el interruptor
 def on_toggled2(self, state,interruptorTrafico):
     if self.interruptorTrafico.get_active():
                 self.etiquetaTrafico.set_text("Activado")
     else:
                 self.etiquetaTrafico.set_text("Desactivado")
 #Metodo para cambiar activar o desactivar el trafico cuando se
accione el interruptor
 def on_toggled(self, state,interruptorTrafico):
     if self.interruptorTrafico.get_active():

requests.put('http://localhost:8080/firewall/module/enable/TODOS')
     else:

requests.put('http://localhost:8080/firewall/module/disable/TODOS')

 #Metodo para definir la seleccion de protocolo en el combobox

 def on_comboboxProtocolo_changed(self, comboboxProtocolo):
     model = comboboxProtocolo.get_model()
     index = comboboxProtocolo.get_active()
     global seleccprotocolo
     if index > -1:
         seleccprotocolo = model[index][0]
     return True
 #Metodo para definir la seleccion de id de switch en el combobox

 def on_comboboxInsertarIdSwitch_changed(self,
comboboxInsertarIdSwitch):
     model = comboboxInsertarIdSwitch.get_model()
```

```
        index = comboboxInsertarIdSwitch.get_active()
        global seleccinsertaridswitch
        if index > -1:
            seleccinsertaridswitch = model[index][0]
        return True
    #Metodo para definir la seleccion de id de switch en el combobox

    def on_comboboxVerIdSwitch_changed(self, comboboxVerIdSwitch):
        model = comboboxVerIdSwitch.get_model()
        index = comboboxVerIdSwitch.get_active()
        global seleccveridswitch
        if index > -1:
            seleccveridswitch = model[index][0]
            self.obtener_reglas()
        return True
    #Metodo para definir la seleccion de accion en el combobox
    def on_comboboxAccion_changed(self, comboboxAccion):
        model = comboboxAccion.get_model()
        index = comboboxAccion.get_active()
        global seleccaccion
        if index > -1:
            seleccaccion = model[index][0]
        return True
    #Metodo para obtener la regla seleccionada en la lista de reglas

    def on_changed(self, selection):
            # get the model and the iterator that points at the data in
the model
            (model, iter) = selection.get_selected()
            lista=list(model[iter])
            global seleccregla
            seleccregla=str(lista[0])
            return True
    #Metodo para actualizar la topologia de red mediante el boton

    def on_botonActualizarRed_clicked(self, widget, data=None):
     global listareglasactualizar
     sw=requests.get('http://localhost:8080/firewall/module/status')
     if sw.status_code==400:
        d={}
        d=defaultdict(lambda: "", d)
     else:
        d=sw.json()
     listadiccionario = []
     listainsertaridswitch.clear()
     listaveridswitch.clear()
     for x in d.keys():
       #Dar la llave al diccionario
       listadiccionario.append(x)
     remplazo_idswitch = {'u':'', 'switch_id: ':''}
     text = remplazo_letras(str(listadiccionario), remplazo_idswitch)
     l = literal_eval(text)
     listareglasactualizar= l
     for y in listareglasactualizar:
       listaveridswitch.append([y])
     for y in listareglasactualizar:
       listainsertaridswitch.append([y])
     listainsertaridswitch.append(['TODOS'])
     """Actualizar valores en los combobox despues de accionar el boton
de actualizar red"""
```

```
  self.comboboxInsertarIdSwitch.set_model(listainsertaridswitch)
  self.comboboxVerIdSwitch.set_model(listaveridswitch)
  self.comboboxInsertarIdSwitch.set_active(0)
  self.comboboxVerIdSwitch.set_active(0)
 #Metodo para borrar regla seleccionada despues de accionar el boton
 def on_botonBorrarReglas_clicked(self, widget, data=None):
  payloadborrar={}
  payloadborrar.update({'regla_id':seleccregla})

requests.delete('http://localhost:8080/firewall/reglas/'+str(seleccv
eridswitch), data=json.dumps(payloadborrar), headers={"content-
type": "text/javascript"})
  self.obtener_reglas()
 #Metodo para ingresar regla despues de accionar el boton
 def on_botonIngresarReglas_clicked(self, widget, data=None):
  payload={}
  if self.entradaMACFuente.get_text()!='':
     payload.update({'dl_src':self.entradaMACFuente.get_text()})
  else:
     pass
  if self.entradaIPDestino.get_text()!='':

payload.update({'nw_dst':self.entradaIPDestino.get_text()+"/32"})
  else:
     pass
  if self.entradaIPFuente.get_text()!='':

payload.update({'nw_src':self.entradaIPFuente.get_text()+"/32"})
  else:
     pass
  if self.entradaMACDestino.get_text()!='':
     payload.update({'dl_dst':self.entradaMACDestino.get_text()})
  else:
     pass
  if self.entradaPuertoDestino.get_text()!='':

payload.update({'tp_dst':int(self.entradaPuertoDestino.get_text())})
  else:
     pass
  if self.entradaPuertoFuente.get_text()!='':

payload.update({'tp_src':int(self.entradaPuertoFuente.get_text())})
  else:
     pass
  payload.update({'nw_proto':seleccprotocolo})
  payload.update({'actions':seleccaccion})

requests.post('http://localhost:8080/firewall/reglas/'+str(seleccins
ertaridswitch)  , data=json.dumps(payload), headers={"content-type":
"text/javascript"})
  payload.clear()
  self.obtener_reglas()
  #Metodo para obtener reglas y mostrarlas en la GUI
 def obtener_reglas(self):
  r =
requests.get('http://localhost:8080/firewall/reglas/'+str(seleccveri
dswitch))
  respuesta=str(r.json())
  texto_base=respuesta.split("@")
  #Texto a reemplazar en el diccionario obtenido
```

```
  texto = texto_base[1]
   #Caracteres a remplazar en el diccionario obtenido
  remplazo_reglas = {'u':'', '\': [[':'[{', 'd: ':'d\': \'', ',
{u\'':', \'', '}], [u\'r':'},{\'r', '}]]}':'}]'}
   #Llamada al metodo para remplazar reglas
  texto_final = remplazo_letras(texto, remplazo_reglas)
  almacen.clear()
  l = literal_eval(texto_final)
  lista= l
  for item in lista:
    for key, value in item.iteritems():
            item[key] = str(value)
  listareglas=[]
  for item in lista:
      item = defaultdict(lambda: "*", item)
      listareglas.append(item)
  for i in listareglas:
      almacen.append([i['regla_id'], i['dl_src'], i['dl_dst'],
i['nw_src'], i['nw_dst'], i['nw_proto'],  i['tp_src'], i['tp_dst'],
i['actions']])
  self.vistaReglas.set_model(almacen)
  self.vistaReglas.show()
  #Metodo para mostrar reglas luego de accionar el boton
 def on_botonVerReglas_clicked(self, widget, data=None):
  self.obtener_reglas()
  #Metodo para salir de la GUI
 def on_ventana_destroy(self, widget, data=None):
  Gtk.main_quit()
ClienteFirewall()
Gtk.main()
```

ANEXO H: Código fuente para la creación de la interfaz gráfica de la aplicación IoT

```
<!DOCTYPE HTML>
<html>
	<head>
		<title>Proyecto de Titulación</title>
		<meta charset="utf-8" />
		<meta name="viewport" content="width=device-width,
		 initial-scale=1" />
		<link rel="stylesheet" href="assets/css/main.css" />
	</head>
	<body>
		<!-- Page Wrapper -->
			<div id="page-wrapper">

				<!-- Header -->
					<header id="header">
						<h1><a href="index.html">Proyecto
							de Titulación</a></h1>
						<nav id="nav">
							<ul>
								<li class="special">
									<a href="#menu"
class="menuToggle"><span>Menu</span></a>
									<div id="menu">
										<ul>

	<li><a href="index.html">Inicio</a></li>

	<li><a href="graficas-tiempo.html">Servidor IoT</a></li>
										</ul>
									</div>
								</li>
							</ul>
						</nav>
					</header>

				<!-- Main -->
					<article id="main">
						<header>
							<h2>Industria 4.0</h2>
							<p>Control y monitorización
							    de procesos</p>
						</header>
						<section class="wrapper style5">
							<div class="inner">

		<section>
		    <iframe style="float:left;
margin-left:80px;float:top; margin-top:-100px;"
src="http://127.0.0.1:1880/ui/" width="900" height="700"></iframe>
		    <iframe style="float:left;
margin-left:110px; float:top; margin-top:-310px;"
src="http://192.168.3.15/" width="400" height="300"></iframe>
		    <iframe style="float:left;
margin-left:550px; float:top; margin-top:-310px;"
src="http://192.168.3.70:4747/" width="400" height="300"></iframe>
```

```
                                    </section>


      <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><b
r><br><br><br><br><br><br><br>
                                    <hr>
                                                </div>
                                            </section>
                                        </article>

                                <!-- Footer -->
                                    <footer id="footer">
                                        <ul class="copyright">
                                            <li><span
class="label">Cristian Chiliquinga- 2019</span></a></li>
                                        </ul>
                                    </footer>

                            </div>

<!-- Scripts -->
                            <script src="assets/js/jquery.min.js"></script>
                            <script
                            src="assets/js/jquery.scrollex.min.js"></script>
                            <script
                            src="assets/js/jquery.scrolly.min.js"></script>
                            <script src="assets/js/skel.min.js"></script>
                            <script src="assets/js/util.js"></script>
                            <script src="assets/js/main.js"></script>
                            <script type="text/javascript"
                            src="js/canvasjs.min.js"></script>
      </body>
</html>
```