UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería del Software

Desarrollo y validación de servicios para redes dirigidas por software con P4

Service development and validation for software driven networking using P4

Realizado por
Carlos Castaño Moreno

Tutorizado por
Pedro Merino Gómez
Francisco Luque Schempp

Departamento
Lenguajes y Ciencias de la Computación
Universidad de Málaga

MÁLAGA, septiembre de 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA/O EN INGENIERÍA DEL SOFTWARE

DESARROLLO Y VALIDACIÓN DE SERVICIOS PARA REDES DIRIGIDAS POR SOFTWARE CON P4

SERVICE DEVELOPMENT AND VALIDATION FOR SOFTWARE DRIVEN NETWORKING USING P4

Realizado por
Carlos Castaño Moreno

Tutorizado por
Pedro Merino Gómez
Francisco Luque Schempp

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2025

Agradecimientos

Con la realización y finalización de este proyecto cierro una importante etapa académica. Han sido unos años de dedicación, esfuerzo constante y gran resiliencia, que me han permitido obtener una base profesional sólida, madurar el sentido crítico, explorar nuevas perspectivas del conocimiento y crecer personalmente.

El presente trabajo refleja el resultado del aprendizaje y puesta en práctica de competencias trasversales en el ejercicio de la disciplina de *Ingeniería del Software*, constituyendo una evidencia de la evolución experimentada a lo largo de los estudios.

En este camino me han acompañado multitud de personas; especialmente familia y amigos, que han sido un apoyo incuestionable e importante para tener siempre motivación, y nunca rendirme en esta carrera de fondo que emprendía.

Me gustaría agradecer en especial al profesor *Juan José Ortega Daza* la orientación y ayuda recibida para elegir y afrontar la temática abordada en el trabajo; a *Rafael García* por su disposición para facilitarme el trabajo con los equipos de laboratorio del equipo de investigación *morse*; y finalmente, a *mis tutores*, por el asesoramiento, apoyo y confianza que siempre me han transmitido, y por haberme dado la oportunidad de trabajar con recursos software y hardware del grupo de investigación mencionado, que ha sido una oportunidad única de aplicar las capacidades adquiridas a lo largo del grado en un área novedosa y con un futuro prometedor.

Resumen

La constante y acelerada evolución que están experimentando las tecnologías de la información y la comunicación, está transformando las soluciones que se están diseñando y desarrollando. En el ámbito de las redes de comunicación, los protocolos y las tecnologías base son de naturaleza cambiante, algo que contrasta con el empleo de conmutadores no modificables, derivando en la obsolescencia acelerada de los dispositivos de red.

En búsqueda de una mayor flexibilidad, surgen las redes definidas o dirigidas por software (SDN), donde los dispositivos integrantes pueden ser programados con software, favoreciendo una adaptación rápida y eficiente, de tal forma que ofrecen un determinado servicio en su entorno. Este paradigma de red induce una división del dispositivo en dos unidades lógicas según su función: el plano de datos, que define el procesamiento del tráfico de red entrante, y el plano de control, que dirige el comportamiento del anterior.

Este trabajo pretende mostrar una metodología estructurada para el diseño y desarrollo de diferentes servicios de red, con el empleo del lenguaje P4, creado específicamente para la programación del plano de datos de conmutadores compatibles, de tal forma que puedan ser validados y desplegados de forma fiable.

Palabras clave:

red SDN, conmutador programable, servicio de red, plano de datos, plano de control.

Abstract

The constant and accelerated evolution of information and communications technologies is transforming the solutions being designed and developed. In the field of communications networks, protocols and underlying technologies are constantly changing, contrasting with the use of non-modifiable switches, leading to the accelerated obsolescence of network devices.

In search of greater flexibility, software-defined or software-driven networks (SDN) are emerging, where component devices can be programmed by software, enabling fast and efficient adaptation, making them capable of offering a specific service in their environment. This network paradigm divides the device into two logical units based on their offered function: the data plane, which defines the processing of incoming network traffic, and the control plane, which governs the behavior of the previous one.

This work aims to demonstrate a structured methodology for the design and development of different network services, using the P4 language, created specifically for programming the data plane of compatible switches, so that they can be validated and safely deployed.

Keywords:

SDN network, programmable switch, network service, data plane, control plane.

Definiciones

La siguiente relación de definiciones y términos se corresponde con conocimientos estrechamente ligados a la temática abordada en este trabajo; la programación de servicios en redes SDN, y más concretamente en el ámbito del lenguaje P4. Pretenden ser de ayuda para facilitar la comprensión del presente documento.

- **Arquitectura programable**: Abstracción de alto nivel relativa a un dispositivo con plano de datos programable, que establece bloques que pueden ser programados, además de ofrecer interfaces para llevar a cabo operaciones de bajo nivel.
- **Bloque programable**: Espacio lógico definido por la arquitectura de un dispositivo SDN cuyo comportamiento es programable.
- **Cabecera de protocolo**: Sección o cadena de datos binaria relativa a un protocolo, que contiene un conjunto de campos predefinidos.
- **Conmutador de red (switch)**: Dispositivo integrante de una red que opera a nivel de enlace, permitiendo la interconexión entre varios sectores que conforman una sola red física.
- **Conmutador SDN (SDN switch)**: Dispositivo integrante de una red cuyo comportamiento es definible mediante software. Es una evolución del conmutador tradicional, preservando en el nivel físico una interfaz similar, pero con un comportamiento flexible a alto nivel.
- **Deparser (ensamblador de cabeceras)**: Unidad lógica que ensambla un paquete de red para su transmisión.
- **Dispositivo programable con P4**: Es un dispositivo de red compatible con la programación de su plano de datos en el lenguaje P4. Presenta un compilador adaptado en base a una arquitectura lógica definida en el mismo lenguaje, así como un protocolo que permite la comunicación entre los planos de datos y de control.
- **Extern**: Objeto externo al ámbito del lenguaje P4, ofrecido por una determinada arquitectura, que presenta una interfaz con operaciones invocables desde un

programa P4. Su principal objetivo es la realización de operaciones que salen del ámbito de la programación con P4 y puede presentar tanto naturaleza software como hardware.

- **Paquete**: Unidad mínima de información transmitida en una red de comunicaciones, con origen y destino definidos.
- **Parser (analizador de cabeceras)**: Analizador definible con P4 con máquinas estados estática, que lee y extrae cabeceras de un determinado paquete.
- **Pipeline (tubería de procesamiento)**: Conjunto de etapas o bloques que conforman la línea de procesamiento de paquetes en el plano de datos.
- **Plano de control**: División lógica de un dispositivo de red SDN encargada de orquestar y establecer las condiciones de tránsito de los paquetes entrantes. Además, gestiona las diferentes estructuras de datos definidas por el plano de datos y presenta mecanismos para comunicarse con él.
- **Plano de datos**: División lógica de un dispositivo de red SDN encargada de definir el procesamiento de paquetes que transitan por el mismo dispositivo.
- **Programa P4**: Artefacto software que define el comportamiento del plano de datos de un dispositivo, en base a una arquitectura lógica definida.
- **Protocolo**: Conjunto de normas o reglas que permiten la comunicación y el intercambio de datos de forma efectiva.
- **Servicio de red**: Conjunto de funcionalidades que ofrece un dispositivo SDN programable a una red.
- **Target**: Término empleado para referirse a un dispositivo SDN que presenta una determinada arquitectura programable. Es tratado frecuentemente como sinónimo del tipo de conmutador con el que se trabaja al programar con P4.
- **Topología de red**: Estructura lógica que especifica los equipos que conforman una red junto a los enlaces existentes entre ellos. Puede ser representada con un grafo, donde los nodos son modelados como vértices y los enlaces como aristas.
- **Torre o pila de protocolos**: Estructura conceptual que modela niveles de protocolos, según su área de operación (red local, redes extensas, extremo a extremo) y funcionalidad que ofrece, que coincide con el orden en el que se encuentran las cabeceras de protocolo presentes en un paquete de red. Se destaca la torre TCP/IP, que sigue el modelo OSI.

Índice

1 INTRODUCCIÓN 23

1.1. Motivación 23
1.2. Objetivos 25
1.3. Estructura de la memoria 29

2 ESTADO DEL ARTE 33

2.1. Las redes TCP/IP 33
2.1.1. Direccionamiento en Ethernet 34
2.1.2. Direccionamiento en IPv4 35
2.1.3. Direccionamiento en TCP y UDP 35
2.2. Las redes definidas por software – SDN 36
2.3. La nueva generación de redes SDN – NG-SDN 40
2.4. El lenguaje de programación P4 41
2.4.1. Modelo de compilación de P4 44
2.4.2. Tipos de datos y estructuras básicas 46
2.5. Arquitecturas programables 49
2.6. El protocolo de control P4Runtime 54

3 TECNOLOGÍAS EMPLEADAS 57

4 METODOLOGÍA DE TRABAJO 67

4.1. Descripción de la metodología 67
4.2. Fases de trabajo 68
4.3. Organización de tareas 71
4.4. Control de versiones 72
4.5. Trabajo con diagramas y modelos 73
4.6. Resultados de su empleo 73

5 DEFINICIÓN Y MODELADO DE SERVICIOS DE RED 75

5.1. Conmutador con aprendizaje 77
5.2. Router con firewall de filtrado 83
5.3. Router con traducción NAT 89
5.4. Router con monitorización de tráfico 95
5.5. Router con políticas de calidad de servicio – QoS (Quality of Service) 99

6 DESARROLLO DE UN CONTROLADOR P4 107

6.1. Definición de requisitos de control 108
6.2. Modelado UML de un conector de control 108
6.3. Implementación del conector modelado 111
6.4. Desarrollo de controladores universales 113

7 DESARROLLO Y VALIDACIÓN EN BMV2 115

7.1. La arquitectura V1model en BMv2 115
7.2. Integración con el emulador Mininet 117
7.3. Desarrollo y validación de servicios de red 119
7.3.1. Conmutador con aprendizaje 119
7.3.2. Router con firewall de filtrado 125
7.3.3. Router con traducción NAT 135
7.3.4. Router con monitorizacion de tráfico 139
7.3.5. Router con políticas de calidad de servicio 144

8 ADAPTACIÓN A LA PLATAFORMA TOFINO 151

8.1. La arquitectura PSA 151
8.2. Entorno de desarrollo de Tofino 152
8.3. Adaptación de servicios y verificación 154
8.3.1. Router con firewall de filtrado 155
8.3.2. Router con traducción NAT 159
8.3.3. Router con políticas de calidad de servicio 162

9 CONCLUSIONES Y LÍNEAS FUTURAS 167

Referencias 173

Apéndice A: Estructura del proyecto 177

A.1: DESCRIPCIÓN DE LA ESTRUCTURA 177
A.2: INSTALACIÓN DEL PROYECTO 180
A.3: GUÍA DE EJECUCIÓN 182

Apéndice B: La suma de comprobación de Internet 187

B.1: CÁLCULO Y VERIFICACIÓN 188
B.2: ACTUALIZACIÓN INCREMENTAL 188
B.3: VERIFICACIÓN Y ACTUALIZACIÓN EN P4 189

Apéndice C: Declaración de cabeceras en P4 191

RELACIÓN DE CABECERAS DE PROTOCOLOS 192
DEFINICIÓN DE DATOS 195
IDENTIFICADORES DE PROTOCOLOS 196

Apéndice D: Las interfaces de red 197

Índice de figuras

FIGURAS

Figura 1. Etapas de desarrollo de servicios de red 24
Figura 2. Torre de protocolos TCP/IP 33
Figura 3. Orden de aparición de cabeceras en un paquete de red 34
Figura 4. Bloques de una dirección MAC 34
Figura 5. Ejemplo de dirección IPv4 35
Figura 6. Ejemplo de máscara de bits en Ipv4 35
Figura 7. Puerto de transporte acompañado de una dirección IPv4 35
Figura 8. Estructura general de un conmutador SDN 36
Figura 9. Modelo de comunicación en el control 37
Figura 10. Escenario de control en OpenFlow 38
Figura 11. Escenario básico de control 38
Figura 12. Escenario de control centralizado 39
Figura 13. Escenario de control segregado 39
Figura 14. Escenario de control especializado 39
Figura 15. Escenario de control descentralizado 39
Figura 16. Paradigma tradicional del plano de datos 40
Figura 17. Paradigma renovado del plano de datos 41
Figura 18. Abstracción del entorno de un conmutador P4 42
Figura 19. Función main de un programa P4 43
Figura 20. Importación de librerías en P4 43
Figura 21. Diagrama del modelo de compilación en P4 44
Figura 22. Modelo de ejecución de un programa P4 44
Figura 23. Estructura de procesamiento del compilador de referencia de P4 45
Figura 24. Diagrama de actividad de compiladores P4 específicos 45
Figura 25. Estructura de una tabla en P4 47
Figura 26. Secuencia match-action de múltiples tablas en P4 47
Figura 27. Ejemplo de la estructura de una tabla declarada en P4 48

Figura 28. Ejemplo de declaración de tabla con acciones en código P4 48
Figura 29. Arquitectura de conmutador independiente de protocolo 49
Figura 30. Estructura física de un conmutador P4 50
Figura 31. Arquitectura con múltiples líneas de procesamiento 50
Figura 32. Representación del Parser 51
Figura 33. Representación de Ingress 51
Figura 34. Representación del Traffic Manager 52
Figura 35. Estructura lógica del componente "Traffic Manager" 52
Figura 36. Representación del Egress 53
Figura 37. Representación del Deparser 53
Figura 38. Esquema de conexión P4Runtime 54
Figura 40. Operaciones ofrecidas por P4Runtime 55
Figura 39. Esquema de ejecución con P4Runtime 55
Figura 41. Roles de control 56
Figura 42. Creación de diagramas con Draw.io 57
Figura 43. Desglose de tareas en Trello 71
Figura 44. Sección del árbol de versiones del repositorio utilizado 72
Figura 45. Etapas de modelado de servicios 76
Figura 46. Estructura de la tabla P4 llamada " Ethernet Timeout " 79
Figura 47. Diagrama de actividad de la tabla P4 llamada “Ethernet Timeout” 79
Figura 48. Estructura de la tabla P4 llamada "Forward Ethernet" 80
Figura 49. Diagrama de actividad de la tabla P4 llamada “Forward Ethernet” 80
Figura 50. Máquina de estados del parser del conmutador con aprendizaje 80
Figura 51. Diagrama de actividad del conmutador con aprendizaje 81
Figura 52. Controlador para el conmutador con aprendizaje 81
Figura 53. Pipeline lógico del conmutador con aprendizaje 82
Figura 54. Máquina de estados del parser del router con firewall 83
Figura 55. Estructura de la tabla P4 llamada "Firewall" 85
Figura 56. Diagrama de actividad de la tabla P4 llamada “Firewall” 85
Figura 57. Estructura de la tabla P4 llamada "Forward IPv4" 86
Figura 58. Diagrama de actividad de la tabla P4 llamada “Forward IPv4” 86
Figura 59. Estructura de la tabla P4 llamada "Port-Mac mapping" 87
Figura 60. Diagrama de actividad del router con firewall 87
Figura 61. Controlador para el router con firewall 87

Figura 62. Pipeline lógico del router con firewall 88
Figura 63. Máquina de estados del parser del router NAT 89
Figura 64. Ejemplo de tabla de traducción NAT 90
Figura 65. Estructura de la tabla P4 llamada "Destination NAT" 91
Figura 66. Diagrama de actividad de la tabla P4 llamada "Destination NAT" 91
Figura 67. Estructura de la tabla P4 llamada "Source NAT" 92
Figura 68. Diagrama de actividad de la tabla P4 llamada "Source NAT" 92
Figura 69. Diagrama de actividad del router NAT 93
Figura 70. Controlador para el router NAT 93
Figura 71. Pipeline lógico del router NAT 94
Figura 72. Máquina de estados del parser del router monitor 95
Figura 73. Cabecera personalizada "Monitor" 95
Figura 74. Torre de protocolos con la cabecera "Monitor" 96
Figura 75. Diagrama de actividad del router monitor 97
Figura 76. Controlador para el router monitor 97
Figura 77. Pipeline lógico del router monitor 98
Figura 78. Máquina de estados del parser del router con calidad de servicio 99
Figura 79. Cabecera personalizada "Flow", junto la torre de protocolos que lo incluye 100
Figura 80. Diagrama de actividad de un medidor de tres colores 101
Figura 81. Estructura de la tabla P4 llamada "flow_meter" 103
Figura 82. Diagrama de actividad de la tabla P4 llamada "flow_meter" 103
Figura 83. Medidor aplicado por defecto 103
Figura 84. Diagrama de actividad del router con calidad de servicio 104
Figura 85. Controlador para el router con calidad de servicio 104
Figura 86. Pipeline lógico del router con calidad de servicio 105
Figura 87. Interoperabilidad del controlador 107
Figura 88. Jerarquía de implementación de conectores 109
Figura 89. Diagrama UML completo del conector de control 110
Figura 90. Implementación en Python del método "getTableEntry" 111
Figura 91. Modelo de clasificación de mensajes entrantes al conector 112
Figura 92. Tareas concurrentes del conector 113
Figura 93. Jerarquía de controladores comunes 114
Figura 94. Pipeline de la arquitectura V1model 116

Figura 95. Estructura lógica de conmutador BMv2 116
Figura 96. Modelo del compilador de BMv2 117
Figura 97. Diagrama de actividad del generador de Mininet 118
Figura 98. Ejemplo de topología especificada en JSON para Mininet 118
Figura 99. Pipeline del conmutador con aprendizaje desarrollado en BMv2 120
Figura 100. Primera topología de prueba del conmutador con aprendizaje 121
Figura 101. Grupo multicast del primer escenario de prueba 121
Figura 102. Captura del tráfico de prueba del temporizador 122
Figura 103. Captura de prueba de interconexión en la primera topología de prueba del conmutador con aprendizaje 123
Figura 104. Segunda topología de prueba del conmutador con aprendizaje 123
Figura 105. Grupos de difusión del segundo escenario de prueba 123
Figura 106. Captura de prueba de interconexión en la segunda topología de prueba del conmutador con aprendizaje 124
Figura 107.Pipeline del router con firewall desarrollado en BMv2 125
Figura 108. Primera topología de prueba del router con firewall en BMv2 126
Figura 109. Estado de la tabla de encaminamiento en el primer escenario de prueba del router con firewall en BMv2 126
Figura 110. Captura de la prueba del campo "options" del protocolo IPv4 en BMv2 127
Figura 111. Captura de la prueba de descarte de paquetes mal formados en BMv2 128
Figura 112. Captura de la prueba de encaminamiento de la primera topología del router con firewall de filtrado en BMv2 128
Figura 113. Evidencia de decremento del valor de TTL 129
Figura 114. Evidencia del establecimiento de una nueva dirección MAC origen 129
Figura 115. Captura de la prueba de filtrado de la primera topología del router con firewall de filtrado en BMv2 130
Figura 116. Segunda topología de prueba del router con firewall en BMv2 131
Figura 117. Estado de las tablas de encaminamiento en el segundo escenario de prueba del router con firewall en BMv2 132
Figura 118. Captura de la prueba de encaminamiento de la primera topología del router con firewall de filtrado en BMv2 133

Figura 119. Estado de las tablas Firewall en la segunda topología del router con firewall en BMv2 134
Figura 120. Captura de la prueba de filtrado de la segunda topología del router con firewall de filtrado en BMv2..... 134
Figura 121. Pipeline del router NAT desarrollado en BMv2..... 135
Figura 122. Topología de prueba del router NAT en BMv2 136
Figura 123. Estructura lógica con valores de la tabla NAT en BMv2..... 136
Figura 124. Tabla de encaminamiento del router NAT en BMv2 137
Figura 125. Captura de la prueba de traducción en BMv2 137
Figura 126. Evidencia de traducción NAT bidireccional 138
Figura 127. Evidencia de comunicación entre redes privadas..... 138
Figura 128. Evidencia de bloqueo de tráfico no permitido..... 138
Figura 129. Tablas P4 que determinan la privacidad de direcciones IPv4 138
Figura 130. Pipeline del router monitor desarrollado en BMv2..... 139
Figura 131. Topología de prueba del router monitor en BMv2 140
Figura 132. Sesión de clonación del monitor..... 140
Figura 133. Tabla de encaminamiento del router monitor en BMv2 140
Figura 134. Captura del tráfico de prueba monitorizado..... 141
Figura 135. Salida por consola del controlador del router monitor 142
Figura 136. Gráficas temporales de monitorización de tiempos de procesamiento 143
Figura 137. Detalle aumentado de la gráfica de monitorización..... 143
Figura 138. Diagrama de rueda generado a partir de estadísticas temporales 143
Figura 139. Pipeline del router con QoS desarrollado en BMv2..... 144
Figura 140. Topología de prueba para el router con QoS en BMv2 145
Figura 141. Tabla de encaminamiento del router con QoS en BMv2 145
Figura 142. Estado de los meters en BMv2..... 145
Figura 143. Captura de paquetes sin flujo en BMv2..... 146
Figura 144. Captura del flujo 1 de QoS en BMv2 147
Figura 145. Paquete del flujo 1 marcado como verde 148
Figura 146. Paquete del flujo 1 marcado como amarillo..... 148
Figura 147.Captura del flujo 2 de QoS en BMv2 149
Figura 148. Pipeline de la arquitectura PSA..... 151
Figura 149. Estructura conceptual del simulador de Tofino..... 153
Figura 150. Estructura conceptual a alto nivel del hardware de Tofino 153

Figura 151. Pipeline del router con firewall adaptado a Tofino 155
Figura 152. Topología del router con firewall en Tofino 156
Figura 153. Estado de las tablas del router con firewall en Tofino 156
Figura 154. Captura de la prueba del campo "options" del protocolo IPv4 en Tofino 157
Figura 155. Captura de la prueba de encaminamiento del router con firewall de filtrado en Tofino 157
Figura 156. Evidencia de decremento del valor de TTL en Tofino 157
Figura 157. Evidencia del establecimiento de una nueva dirección MAC origen en Tofino 158
Figura 158. Variación del valor de la suma de comprobación en Tofino 158
Figura 159. Captura de la prueba de filtrado del router con firewall de filtrado en Tofino 158
Figura 160. Pipeline del router NAT adaptado a Tofino 159
Figura 161. Topología del router NAT en Tofino 160
Figura 162. Estado de las tablas del router NAT en Tofino 160
Figura 163. Captura de la prueba de traducción en Tofino 161
Figura 164. Detalle ampliado del paquete descartado en el router NAT en Tofino 161
Figura 165. Detalle ampliado del paquete que se transmite de h1 a h2 en NAT de Tofino 161
Figura 166. Detalle ampliado del paquete que se transmite de h2 a h1 en NAT de Tofino 161
Figura 167. Pipeline del router con QoS adaptado a Tofino 162
Figura 168. Topología del router con QoS en Tofino 163
Figura 169. Estado de las tablas del router con QoS en Tofino 163
Figura 170. Captura de paquetes sin flujo en Tofino 164
Figura 171. Captura de paquetes con flujo 1 de QoS en Tofino 165
Figura 172. Paquete del flujo 1 marcado como amarillo en Tofino 165
Figura 173. Captura de paquetes con flujo 2 de QoS en Tofino 166
Figura 174. Jerarquía general del proyecto 178
Figura 175. Sección del proyecto titulada "P4-project" 179
Figura 176. Sección del proyecto titulada "P4Control_connector" 179
Figura 177. Sección del proyecto titulada "Flow plugin" 179
Figura 178. Contenido del primer nivel del repositorio 181

Figura 179. Formato de un archivo de requisitos de Python 181
Figura 180. Variables de entorno para el controlador..... 182
Figura 181. Contenido del directorio "Run scripts" 183
Figura 182. Contenido del directorio "Learning-bridge" 183
Figura 183. Contenido del archivo makefile del conmutador con aprendizaje..... 184
Figura 184. Prompt de Mininet 184
Figura 185. Ejecución simultánea de Mininet y un controlador en dos terminales . 185
Figura 186. Propiedad de actualización incremental de una suma de comprobación 189
Figura 187.Torre de protocolos TCP/IP con capa personalizada 191
Figura 188. Conexión entre las interfaces física y lógica..... 198
Figura 189. Ejecución del comando ifconfig en Linux..... 198
Figura 190. Tubería de interfaces virtuales 199
Figura 191. Gestión de interfaces en un switch SDN 199

TABLAS

Tabla 1. Unidades funcionales de SDN..... 37
Tabla 2. Escenarios de control 39
Tabla 3. Relación de bloques programables notables..... 53
Tabla 4. Tecnologías empleadas en el proyecto 65
Tabla 5. Protocolos reconocidos por el conmutador con aprendizaje 77
Tabla 6. Requisitos funcionales del conmutador con aprendizaje..... 78
Tabla 7. Protocolos reconocidos por el router con firewall 83
Tabla 8. Requisitos funcionales del router con firewall..... 84
Tabla 9. Protocolos reconocidos por el router con traducción NAT..... 89
Tabla 10. Requisitos funcionales del router NAT..... 90
Tabla 11. Protocolos reconocidos por el router monitor 95
Tabla 12. Requisitos funcionales del router monitor..... 96
Tabla 13. Protocolos reconocidos por el router con calidad de servicio 99
Tabla 14. Descripción de los medidores de tres colores 101
Tabla 15. Requisitos funcionales del router con calidad de servicio..... 102
Tabla 16. Requisitos de control 108
Tabla 17. Elementos destacados del desarrollo del conmutador con aprendizaje . 120

Tabla 18. Variación del valor de la suma de comprobación 129
Tabla 19. Estado de la tabla Firewall en la primera topología del router con firewall en BMv2 130
Tabla 20. Áreas tecnológicas del grado cubiertas 172
Tabla 21. Requisitos de instalación 180
Tabla 22. Ejemplos de manipulación de la suma de comprobación en P4 190
Tabla 23. Relación de mapeo de cabeceras de protocolos 194
Tabla 24. Datos definidos en P4 195
Tabla 25. Identificadores de protocolos 196

1

INTRODUCCIÓN

Para comenzar el desarrollo de este trabajo, es necesaria una correcta comprensión del contexto en el que se enmarca y las necesidades que se desean cubrir para cumplir ciertos objetivos relacionados con la temática que se aborda. En este capítulo se plantea de forma inicial el problema que se desea abordar, los resultados que se pretenden obtener y cómo la memoria estructura el desarrollo seguido de manera lineal, para plasmar la evolución existente desde el inicio hasta la finalización del proyecto.

1.1. Motivación

En el mundo de la programación de computadores, existe una gran variedad de lenguajes multipropósito, que permiten al desarrollador implementar soluciones para multitud de escenarios de trabajo; desde la aplicación corporativa de una institución, hasta el sistema de frenado de un vehículo. En este contexto, encontramos también lenguajes destinados a una finalidad particular, como es el caso de P4 [1], un lenguaje de alto nivel que permite al programador definir el comportamiento del plano de datos de dispositivos de redes SDN (Redes Definidas por Software), una división lógica del dispositivo que se encarga de definir el procesamiento de los paquetes que circulan por él. A pesar de su función tan específica, presenta una repercusión que va más allá del plano de datos, teniendo así un gran impacto en su entorno; y es por ello por lo que a menudo mencionaremos el “entorno P4”, que es el conjunto de tecnologías que se ven implicadas.

Este trabajo se enmarca en el mundo de la Ingeniería Telemática [2], área de las TIC que busca entablar puentes entre las Telecomunicaciones y la Informática, mediante

la combinación de conocimientos de ambas disciplinas, en una búsqueda por automatizar y adaptar las tecnologías de comunicaciones a los nuevos avances tecnológicos. Dentro de ella, nos centraremos en la programación de conmutadores SDN (SDN switches), una evolución del conocido dispositivo tradicional que presenta ciertos espacios programables con funciones bien definidas. Uno de los puntos fuertes que presenta la creación de programas para estos dispositivos es la reutilización de la circuitería existente, por lo que se garantiza un mejor aprovechamiento de la tecnología hardware y también un incremento considerable de la velocidad de adaptación a cambios que pueden ocurrir en su área de influencia.

Por su parte, la Ingeniería del software surgió para desarrollar, operar y mantener software de forma ordenada y eficiente, y aplicando las prácticas y metodologías rigurosas que ofrecen el campo de las ingenierías. Una de las señas de identidad de esta disciplina es la creación de modelos fiables, que permitan llevar a cabo las labores de desarrollo de manera ordenada y eficiente. Aplicar estos conocimientos es una oportunidad para descubrir y aplicar un método de desarrollo de software adaptado a la creación de programas que definan el comportamiento del ya mencionado plano de datos, verificar su correcto funcionamiento y poder ponerlos en producción en una red determinada, siempre con seguridad y control sobre su comportamiento.

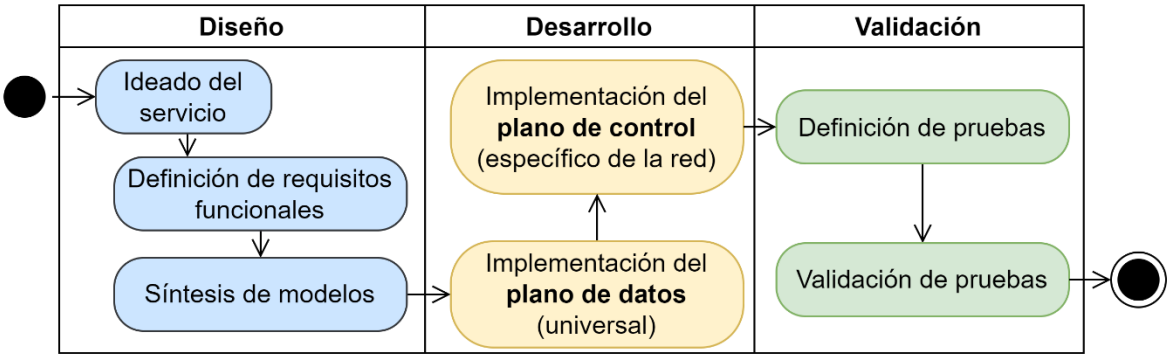


Figura 1. Etapas de desarrollo de servicios de red

En este trabajo se aplicará la metodología de desarrollo adaptada a la síntesis de servicios de red (como se aprecia en la imagen anterior), constituidos por la definición de un plano de datos y la existencia de un controlador. Para ello, se verán todas las etapas de concepción de servicios de red con adaptación al entorno de P4, que van

desde la definición de servicios para el dispositivo con el que trabajamos, la creación de modelos y diagramas que reflejen a alto nivel funcionalidades especificadas por requisitos establecidos, el desarrollo de las realidades modeladas, y finalmente la validación y puesta en ejecución, que podrá ser mantenido en el futuro para garantizar el correcto funcionamiento en red.

A nivel de dispositivos, trabajaremos principalmente con el conmutador (switch) virtual *simple_switch* de BMv2 [3], un framework (marco de trabajo) de desarrollo que permite implementar switches de naturaleza software, para validar servicios ofrecidos por los programas que ejecuta. Este anterior se considera el switch de referencia de P4, pues es idóneo para realizar validaciones sin tener que lidiar con configuraciones complejas inherentes al hardware. Finalmente, se introducirá la metodología necesaria para llevar a cabo la adaptación y el desarrollo para un dispositivo físico, que servirá como cierre del proyecto, y que demostrará la repercusión real que presentan los desarrollos realizados.

1.2. Objetivos

Como sucede en la mayoría de los proyectos de desarrollo de software, una gran parte de los objetivos están enfocados en el producto que se desea obtener, las funciones que cumple y aquello que se persigue con su creación. En este caso, el valor más importante aportado por el proyecto es el empleo estructurado de la tecnología objeto de estudio, esto es P4, y proporcionar conocimientos derivados que permitan una mayor adaptación al ingeniero que desea especializarse en ella. Todo lo anterior podría ser recogido en las siguientes líneas:

- **Diseño de servicios de red para conmutadores**. Mediante una recopilación de requisitos variados, que permitan cubrir todas las prestaciones básicas que puede ofrecer el plano de datos, y que han sido agrupados y definidos en diferentes servicios, se han creado modelos de aquellas funcionalidades que han de ofrecer a alto nivel de tal manera que establecerán a su vez otros requisitos de cara al plano de control. Los modelos mencionados son independientes de la tecnología, y constituyen diseños estables.

- **Desarrollo de módulos telemáticos**. La implementación de diferentes programas de plano de datos con P4 pretende mostrar cómo ofrecer un servicio al entorno del conmutador con el que se trabaja. Para ello, se ha trabajado con una arquitectura de referencia, que es propia de un conmutador de red virtual, el programa *simple_switch* de *BMv2*. En ella se ejecutan los programas junto a un controlador que gestiona los recursos lógicos del dispositivo, de tal forma que los servicios que ofrecen puedan ser validados desde una perspectiva lógica, y esto permita su futura adaptación a diferentes conmutadores de producción.
- **Compilación de programas P4**. Dada la naturaleza universal y agnóstica del lenguaje P4, que implica su uso en cualquier arquitectura que lo implemente, existen diferentes compiladores con modelos de compilación propios. Con los desarrollos llevados a cabo, se pretende incidir en esta independencia existente entre el lenguaje y la arquitectura del conmutador que lo adopta, permitiendo una profundización el proceso de verificación de código y posterior compilado especializado que deriva en un archivo ejecutable específico de la plataforma.
- **Gestión y control conmutadores**. Trabajar con la programación de conmutadores SDN requiere de un dominio importante de las funciones desempeñadas por el plano de control, permitiendo el entendimiento de la interacción de gestión existente con el aparato. Se implementarán controladores, que ofrezcan operaciones de gestión del plano de datos de un conmutador, que permitirá una mayor comprensión de las responsabilidades de cada uno de los planos, y del comportamiento general del tipo de dispositivo con el que se trabaja. En definitiva, es esencial para un correcto dominio de la tecnología abordada.
- **Validación de servicios en red**. Una vez desarrollados los programas de plano de datos, junto a una lógica de control adaptada a una situación de red experimental, es importante verificar propiedades lógicas que validan diferentes requisitos funcionales, y que garantizan una respuesta correcta del dispositivo programado ante diferentes tipos de paquetes de red entrantes. Para garantizar la validación correcta del comportamiento de un determinado programa, se han empleado pruebas unitarias en diferentes escenarios de redes, con un enfoque de inyección de tráfico en “caja negra”, mediante la

agrupación de paquetes en clases de equivalencia según la respuesta/salida generada por el conmutador, algo necesario para validar el correcto funcionamiento global en todas las posibles situaciones.

- **Adaptación de servicios a arquitecturas de producción**. Algo de gran interés, que permitirá observar el proceso de traslación que se ha de seguir para integrar un programa de plano de datos con validación lógica en una arquitectura de producción. Para cumplir con este objetivo, se han seleccionado los servicios más completos, centrando el foco en el proceso de adaptación de un plano de datos validado a una plataforma física. Se pretende incidir en la reducción temporal que supone el trabajar en primera instancia con la arquitectura de referencia (v1model), algo que permite prestar la atención casi exclusivamente en detalles técnicos de adaptación, tales como configuraciones y operaciones de optimización de hardware que han de ser incluidas en la programación del plano de datos.
- **Profundización en el dominio de redes de comunicación**. Una temática no dominada al máximo en el campo de la informática, y que es importante estudiar y comprender con una visión técnica, que repercute de forma notable en la calidad y eficiencia existente en el uso de la infraestructura de redes por parte de diferentes aplicaciones usuarias que se pueden desarrollar. Por otro lado, permite presentar perfiles especializados en esta área, que como muestra el presente trabajo, se encuentra en un momento de gran expansión. En definitiva, que este proyecto sirva en cierta medida para comprender que las redes son una parte imprescindible de los computadores.
- **Ofrecer recursos para facilitar el manejo de P4**. Uno de los mayores valores de este proyecto es sin lugar a duda, el material teórico y práctico que se ofrece para iniciarse en el universo de la programación de dispositivos SDN. Ello es posible al trabajo existente en las áreas y componentes más destacados de un dispositivo de red SDN, que abarca la programación del plano de datos, el estudio de las arquitecturas programables, el desarrollo de operaciones ejercidas por el plano de control, que presenta un formato/protocolo para entablar la comunicación y el seguimiento de buenas prácticas de desarrollo que han sido descubiertas. Todo esto permite evitar situaciones complicadas, ofreciendo una adaptación rápida al empleo del lenguaje P4 y todo su entorno tecnológico.

- **Monitorización de tiempos en un dispositivo**. La correcta visualización de los recursos físicos y temporales relativos a un dispositivo, y en particular un conmutador SDN, es crucial para la detección de anomalías y fallos existentes en el mismo; o incluso en la red de su entorno. La visualización del estado presente de todo lo anterior es compleja en estos dispositivos dedicados, que no presentan una interfaz directa para acometer esta tarea. Es por ello, que se han empleado mecanismos de emisión de mensajes procedentes del plano de datos y con destino el plano de control; algo que permite a este último realizar procesamientos tales como el almacenamiento de estadísticas de tráfico en una base de datos, que ofrezca realizar una consulta instantánea, y a su vez, tener un registro adecuado del tráfico que ha circulado por el conmutador. Para poder mostrar esto de forma clara, uno de los servicios incorpora este mecanismo, y mediante una base de datos de series temporales, es posible monitorizar estadísticas de circulación, ofreciéndose así un mecanismo adicional de visualización.
- **Aplicar técnicas de Ingeniería de Software**. Este último podría ser considerado como uno de los objetivos con mayor potencial, puesto que, mediante la realización de este proyecto, se ha podido experimentar la adaptación de las metodologías de software existentes a una temática bien concreta, que además se encuentra en plena expansión y que presenta además un importante vínculo con el campo de la informática. Llevar a cabo de forma ordenada una metodología que permite proceder con el análisis y recogida de requisitos, la confección de los ya citados modelos, y el desarrollo de los diferentes módulos, garantiza obtener calidad y rigor en los servicios que han sido desarrollados, además de sacar el máximo partido a un lenguaje de programación como lo es P4. En definitiva, es una situación ideal para mostrar la gran importancia de utilizar una buena metodología de desarrollo de software y la positiva repercusión que genera en el resultado final.

1.3. Estructura de la memoria

Para abordar de forma ordenada y escalonada el proceso que se ha seguido en el desarrollo del proyecto, se ha estimado una organización del contenido que permita transmitir la metodología de trabajo, introducir el estado actual de los conocimientos que se abordan, y documentar los diseños y desarrollos implementados con sus respectivas evidencias de validación. De esta manera, el documento muestra de forma clara la evolución y progresión existente desde el planteamiento inicial del proyecto y los escenarios con los que se trabaja, hasta obtener un producto sólido que ofrezca resultados de interés.

En el desarrollo del presente documento se irán incluyendo diferentes diagramas y modelos que faciliten el correcto entendimiento de los conceptos que serán expuestos. Dada la naturaleza de la tecnología empleada, que no presenta apenas documentación en habla castellana, se emplearán muchos términos en lengua inglesa, pero siempre serán debidamente explicados y contextualizados. En ese sentido, los diagramas vendrán expresados a su vez con terminología inglesa, para evitar posibles confusiones derivadas de su traducción, y persiguiendo también el objetivo de que el código desarrollado y sus modelos pueda ser más accesible con el empleo de los términos originales. Cabe destacar que todos los diagramas son de elaboración propia, para presentar un contenido homogéneo a lo largo de todo el documento, además de haber sido de gran utilidad para la interiorización de los conceptos.

Se ha estipulado una **división temática** que cumpla con lo anterior, y que permita al lector partir de los principios elementales hasta las conclusiones y resultados que han sido extraídos del producto final. Las unidades temáticas abordadas son las siguientes:

- **Estado del arte** (CAPÍTULO 2): Para comenzar de forma exitosa con la redacción del desarrollo del trabajo, es necesario partir de los conocimientos mínimos que requiere el uso del lenguaje P4, para garantizar una comprensión veraz con unos principios sólidos. Por ello, se abordan los conocimientos relativos al ámbito de las redes con las que se trabaja y los protocolos presentes, para así introducir el ámbito de las redes definidas por software, la

evolución que han experimentado, y finalmente poder profundizar adecuadamente sobre los fundamentos de P4 y las prestaciones que ofrece. Todo esto expone el estado actual de los conocimientos abordados en este proyecto.

- **Tecnologías empleadas** (CAPÍTULO 3): Un capítulo que recoge en detalle todas las tecnologías y herramientas que han servido de apoyo en el desarrollo del proyecto, pero que no son objeto de estudio.
- **Metodología de trabajo** (CAPÍTULO 4): Se explica el método ágil que ha sido aplicado en el desarrollo del trabajo, sus características más importantes, las fases en las que se divide y los resultados que se extraen de su empleo.
- **Desarrollo del proyecto** (CAPÍTULOS 5-8): Sección que contiene los capítulos que relatan de forma detallada el desarrollo técnico que ha sido realizado, partiendo de la definición y documentación técnica de diferentes servicios de red, junto a los requisitos con los que han de cumplir, Que pretenden mostrar las potenciales funcionalidades que ofrece el empleo del lenguaje P4. Seguidamente se documenta el desarrollo de un controlador SDN, componente necesario para la correcta ejecución de servicios. Para finalizar con esta sección, se detallará el proceso de implementación seguido en el conmutador virtual, junto a sus pruebas que avalan el correcto funcionamiento, y poder mostrar en última instancia la adaptación seguida a una plataforma hardware.
- **Conclusiones y líneas futuras** (CAPÍTULO 9): Sección que abarca conclusiones sintetizadas a partir de los resultados obtenidos tras la finalización del proyecto, así como diferentes extensiones y líneas de profundización hacia las que podría conducir los artefactos tecnológicos creados.
- **Apéndices**: Secciones que no tienen cabida en el hilo principal de redacción, pero que contienen conocimientos adicionales relacionados con el tema abordado y que pueden ser útiles para un mejor entendimiento de la temática abordada. Concretamente se incluyen cuatro:

    - **Estructura del proyecto**: Aborda la organización jerárquica de todos los componentes del proyecto, e indica instrucciones para su instalación y un correcto empleo.

- **Suma de verificación**: Explica y contextualiza un mecanismo fundamental de comprobación de integridad de datos, así como su empleo desde un programa P4 para actualizar su valor en protocolos de comunicación empleados.
- **Cabeceras de protocolos**: Concentra las cabeceras empleadas a lo largo del proyecto, junto a su mapeo en código P4. Además, se incluye una relación de identificadores de protocolos, necesarios para el correcto análisis de las cabeceras presentes en un determinado paquete.
- **Interfaces de red**: Profundiza sobre las interfaces de red de un dispositivo, su propósito, parámetros fundamentales, y cómo gestiona y permite la conexión con su entorno.

La división de capítulos del presente documento condensa de manera adecuada todos los conocimientos y características propios del proyecto, que garantiza la presencia de una correcta documentación.

2

ESTADO DEL ARTE

Una vez definido y delimitado el ámbito del proyecto, debemos introducir conceptos importantes para garantizar un dominio básico de las redes SDN, su estructura, cómo pueden ser programados y gestionados los conmutadores con P4, para poder así presentar el posterior desarrollo con fluidez. Empezaremos por conceptos más elementales, como son los conceptos básicos en los que se fundamentan las redes que han sido empleadas, hasta aproximarnos al propio desarrollo del trabajo. De esta forma se irán abordando los temas objeto de estudio de forma incremental, mostrando y plasmando fielmente el proceso de estudio que ha sido seguido para poder abordar el tema principal con total solidez.

2.1. Las redes TCP/IP

Las redes TCP/IP son aquellas que adoptan el modelo de protocolos definido en la torre o pila de protocolos TCP/IP [4], que es una abstracción que introduce capas que modularizan diferentes funciones, la transmisión fiable de bits, el establecimiento de enlace en una misma red, la comunicación entre redes conectadas, y el establecimiento de una conexión extremo a extremo, responsabilidades propias de las capas física, de enlace, de red, y de transporte, respectivamente. El ordenamiento viene justificado por la dependencia existente entre capas, de tal forma que la capa de enlace es usuaria de la física, y a su vez la de red de la de enlace. Esto proporciona una modularizarían de las funciones dependiendo de la complejidad que presentan, para presentar una estructura clara y eficiente.

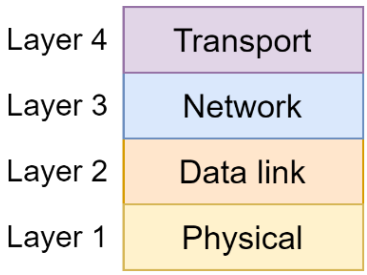


Figura 2. Torre de protocolos TCP/IP

En cada una de las capas existen diferentes protocolos disponibles, que cubren la funcionalidad de aquella capa en la que se alojan, pero con un contexto particular. Por ejemplo, en la capa física se empleará un protocolo u otro dependiendo de si el medio presenta naturaleza física o inalámbrica.

Teniendo todo esto en cuenta, debemos comprender que un paquete de red, que es la unidad mínima de transmisión, aplica un encapsulamiento de cabeceras de izquierda a derecha de menor a mayor nivel. La primera cabecera que aparecerá será la de enlace, pues la capa física se encarga únicamente de la trasmisión, y a partir de esa, será la de red, y la de transporte. Nótese que nos limitamos a la capa de transporte, pues es la máxima con la que se ha trabajado. En la siguiente figura se puede observar el orden de cabeceras de protocolos presente en un paquete.

Figura 3. Orden de aparición de cabeceras en un paquete de red

Veamos a continuación el direccionamiento que se aplica en los protocolos aplicados por capa.

2.1.1. Direccionamiento en Ethernet

Los elementos físicos que conectan un determinado dispositivo con una red son las denominadas *tarjetas de red* (se recomienda consultar el Apéndice D:), y determinan el protocolo de enlace empleado, que ha de ser común en todo el medio físico. El más común y extendido es *Ethernet* [5]. Este protocolo emplea lo que se denominan direcciones MAC (Medium Access Control), que identifican a una interfaz física de forma unívoca. Es una cadena binaria de 48 bits, que comúnmente se representa en 6 bloques con base hexadecimal.

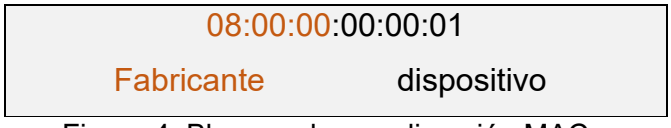


Figura 4. Bloques de una dirección MAC

En la figura anterior, se puede visualizar las estructuras de bloques que existe en este tipo de direcciones, donde los tres primeros octetos resaltados se corresponden con el fabricante. El id 08:00:00 se emplea en redes experimentales.

2.1.2. Direccionamiento en IPv4

El modelo de direccionamiento introducido por el protocolo de red IPv4 (versión 4 del protocolo de internet) [6] establece un formato de dirección que permita identificar a un determinado host (equipo) en el contexto de la localización en la que se encuentra, esto es un identificador de equipo en red. La dirección se compone de 32 bits, que visualmente son agrupados en cuatro grupos de 8 bits. Las siguientes figuras muestran el formato de una dirección, y la división de secciones de una dirección para representar a la red y al host que se encuentra en ella.

192.168.1.2

Figura 5. Ejemplo de dirección IPv4

192.168.1.2/24

Máscara de 24 bits

- Id de red: 192.168.1.0
- Id de host: 2

Figura 6. Ejemplo de máscara de bits en Ipv4

La máscara de red es la que indica la longitud de la sección identificadora de la red y la del host. De esta forma, todos los equipos presentes en la red con ese id tendrán un formato similar de dirección, que sólo varía en la zona destinada a equipos.

2.1.3. Direccionamiento en TCP y UDP

Para entender el direccionamiento aplicado en los protocolos TCP [7] y UDP [8], debemos contextualizarlos en su capa, la de transporte, y entender el propósito que presenta. Su principal función es gestionar la comunicación “extremo a extremo”, que se produce entre diferentes servicios o aplicaciones presentes en un determinado host.

En ese sentido, se introduce un identificador de servicio dentro de sistemas operativos que siguen este protocolo, que es el puerto de transporte, un identificador de 16 bits común a ambos protocolos. Esto se combina con una dirección IP, y permite identificar al servicio de manera universal.

192.168.1.2:**8080**

Figura 7. Puerto de transporte acompañado de una dirección IPv4

2.2. Las redes definidas por software – SDN

Las redes definidas por software – SDN [4] son aquellas que introducen la **programación con software** en los dispositivos que la conforman, permitiendo una definición y dirección flexible de los mismos. Uno de los principales objetivos que propone es la separación de un determinado programa que define el comportamiento de un dispositivo con respecto a la plataforma que se manipula. En este caso explicaremos los fundamentos básicos aplicados a los conmutadores programables, que son dispositivos similares a los conmutadores tradicionales que operaban hasta el nivel de enlace, pero con comportamiento definible, y con un espectro de trabajo que va más allá de la citada capa, lo cual permite que ofrezca la funcionalidad deseada con la presencia de un mismo hardware.

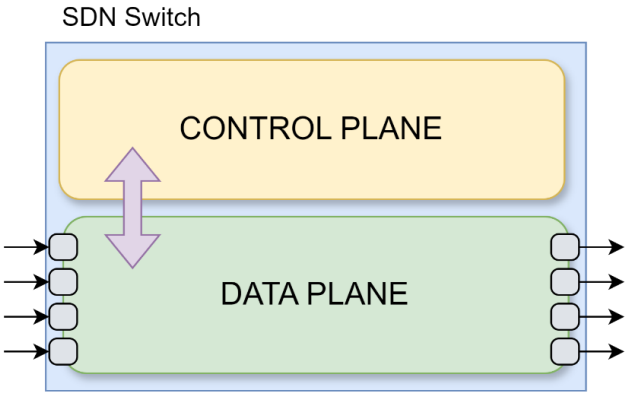
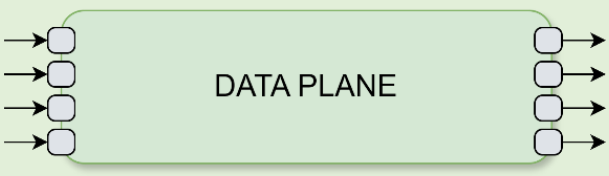


Figura 8. Estructura general de un conmutador SDN

Tradicionalmente, configurar los conmutadores y otros dispositivos era complejo y costoso, y es por ello por lo que SDN establece una división en dos unidades lógicas con funciones bien diferenciadas: el plano de datos y el plano de control. Veamos cuál es el propósito de cada una de ellas, aplicándolo al contexto de conmutadores SDN:

PLANO DE DATOS
Se encarga de definir de manera general la línea de procesamiento que se aplica a un paquete que circula por el dispositivo, mediante una sucesión de acciones que son programadas. Permite además emplear operaciones que establezca una comunicación con el plano de control. Presenta un ligero aislamiento con respecto del hardware, pero a nivel funcional está muy ligado a la estructura del conmutador.



PLANO DE CONTROL
Frecuentemente conocido como controlador, se encarga de dirigir y establecer condiciones sobre los paquetes que circulan por un dispositivo. En el plano de datos existen diferentes estructuras de selección y otros objetos, y es el control el que establece el valor de sus parámetros, e impone condiciones sobre el procesamiento llevado a cabo por el otro plano. Está totalmente desligado de la plataforma que controla, pudiendo estar en remoto y gestionar a más de un dispositivo que ejecute un mismo programa.

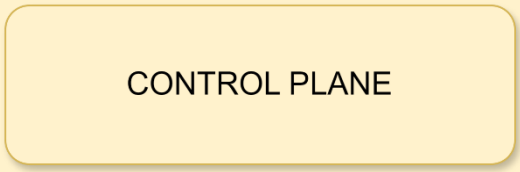


Tabla 1. Unidades funcionales de SDN

Tras entender las funciones de ambos planos, se deduce que es necesaria una correcta comunicación entre las dos divisiones, que permitirá ofrecer un servicio de calidad. Existen multitud de formatos y protocolos que definen un conjunto de reglas de comunicación, que presentan el objetivo de desempeñar diferentes operaciones sobre el plano de datos. Por ejemplo, encontramos el protocolo P4Runtime en este contexto, y por otro lado GNMI, que se encarga de forma exclusiva de la gestión de las interfaces del conmutador. De esta forma podemos distinguir dos tipos de control en función del aspecto que se manipula: los de gestión de plano de datos y los de gestión de interfaces externas.

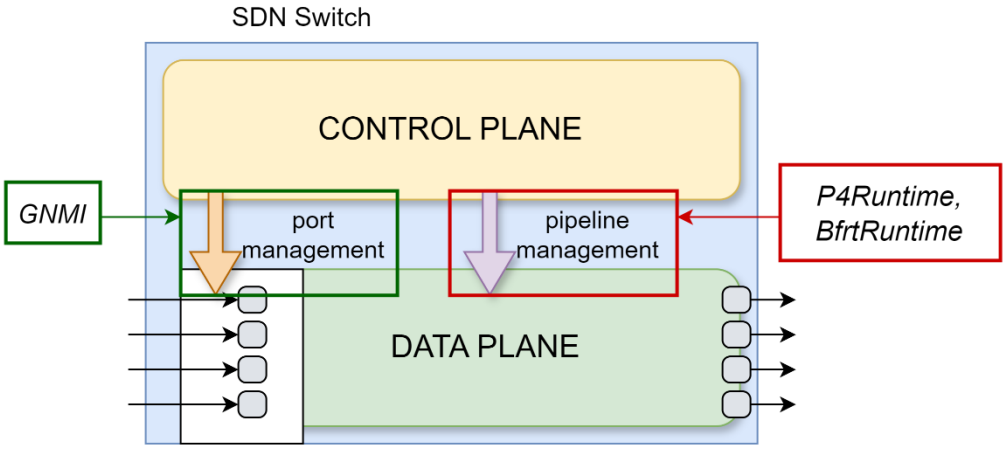


Figura 9. Modelo de comunicación en el control

En este contexto, encontramos un protocolo de gran importancia en el inicio de las redes, OpenFlow [10], que define un modelo de comunicación para el control de dispositivos SDN que presentan un plano de datos con entidades configurables. Estas entidades eran tablas de selección de acciones, que tomando un campo del paquete que se procesa, en función del valor que posea se decide una acción u otra que se ha realizar. Los controladores que operaban con este tipo de conmutadores tenían libertad de procesamiento condicionada al estándar de comunicación que establecía el protocolo, de tal forma que este componente era flexible, pero el plano de datos realizaba un procesamiento con entidades en memoria que presentaban la misma estructura.

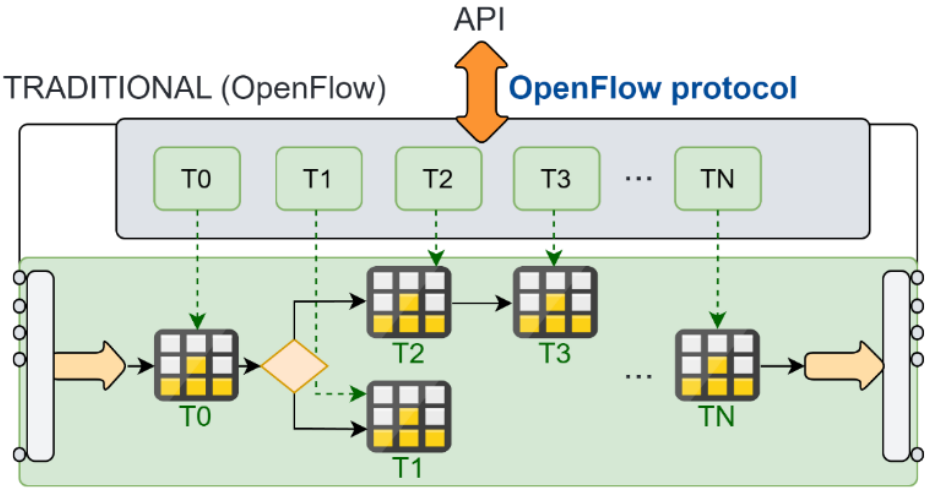


Figura 10. Escenario de control en OpenFlow

A nivel de infraestructura el modelo básico seguido presenta una denominada *conexión de control*, que es el vínculo existente entre un controlador de naturaleza software y la interfaz remota de control ofrecida por el dispositivo. En este sentido se puede ver con una mayor claridad la segregación de control existente, pues el controlador se sitúa al exterior del propio hardware del dispositivo. En la Figura 11 se puede observar una topología simple de red, donde se aprecia el controlador SDN, y la mencionada conexión.

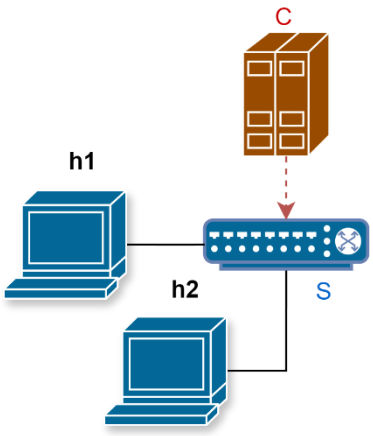


Figura 11. Escenario básico de control

En diversas ocasiones, pueden presentarse diferentes escenarios de control, dependiendo de los requisitos que pudieran existir en la red donde se sitúa el conmutador controlado. Entre las situaciones más destacables podemos encontrar escenarios de control centralizado, donde diferentes dispositivos que tienen su configuración de plano de datos establecida, pero son gobernados por una misma entidad de control. En la siguiente tabla se recogen las situaciones más comunes, que van desde un control especializado, donde varios controladores gestionan diferentes áreas de un plano de datos, hasta situaciones que requieran de un control independiente.

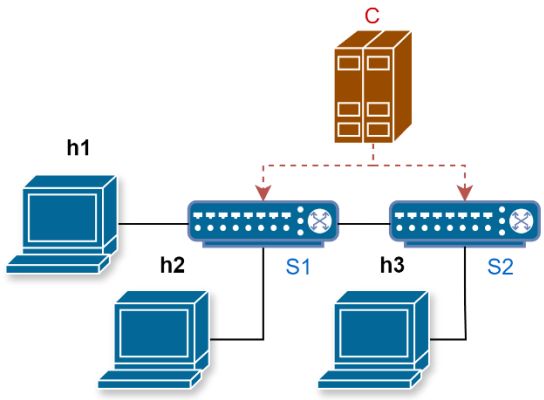
<table>
<tr><th colspan="2">ESCENARIOS DE CONTROL</th></tr>
<tr><td> 

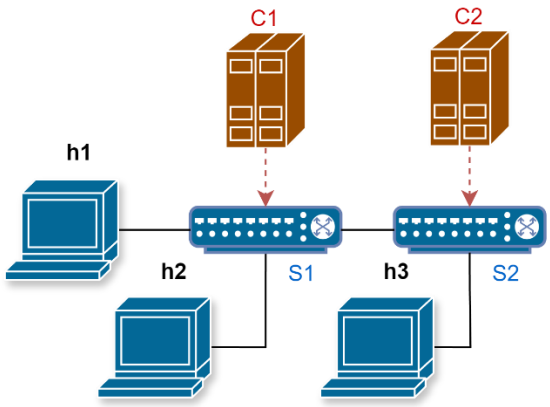
Figura 12. Escenario de control centralizado</td><td> 

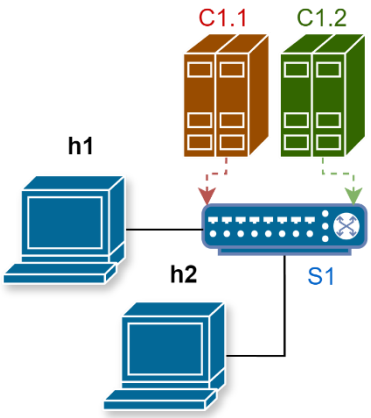
Figura 13. Escenario de control segregado</td></tr>
<tr><td> 

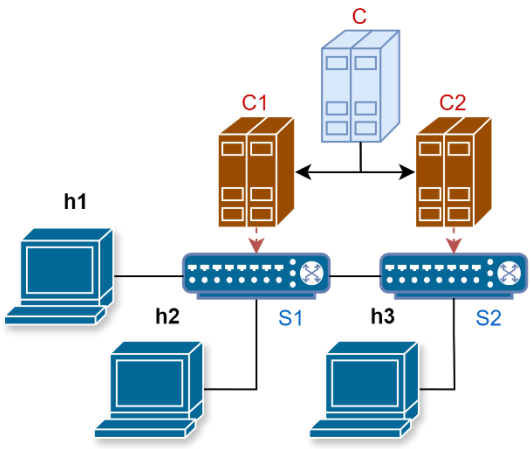
Figura 14. Escenario de control especializado</td><td> 

Figura 15. Escenario de control descentralizado</td></tr>
</table>

Tabla 2. Escenarios de control

2.3. La nueva generación de redes SDN – NG-SDN

En casos como en el de OpenFlow, donde existe un plano de datos rígido con entidades configurables, pero sin definiciones de procesamiento de paquetes complejas, el protagonismo de programación recae en el plano de control, pues es el que verdaderamente adopta en su totalidad el empleo de software que dirige el comportamiento dinámico del conmutador controlado. Este tipo de tecnologías fueron en su momento muy vanguardistas, por el paradigma de programación de redes que se introducía, pero tenía un gran inconveniente, la imposibilidad de manipular el repertorio de protocolos reconocidos.

Durante un tiempo, las redes no eran tan cambiantes, y por lo tanto los protocolos empleados eran más o menos estables. Hoy en día esto ya no es así, debido a que están apareciendo nuevas versiones de protocolos tradicionales, pero que introducen cabeceras completamente renovadas. Incluso en entornos empresariales, pueden crearse protocolos nuevos para el ámbito de una organización, por lo que el empleo de conmutadores con estas características no es viable. Este enfoque se denomina *Bottom Up* (De abajo hacia arriba), en el que el conmutador ha de conocer el contexto tecnológico de la red en la que se encuentra, los protocolos que ha de reconocer, y en definitiva como se ha de comportar de cara al medio externo.

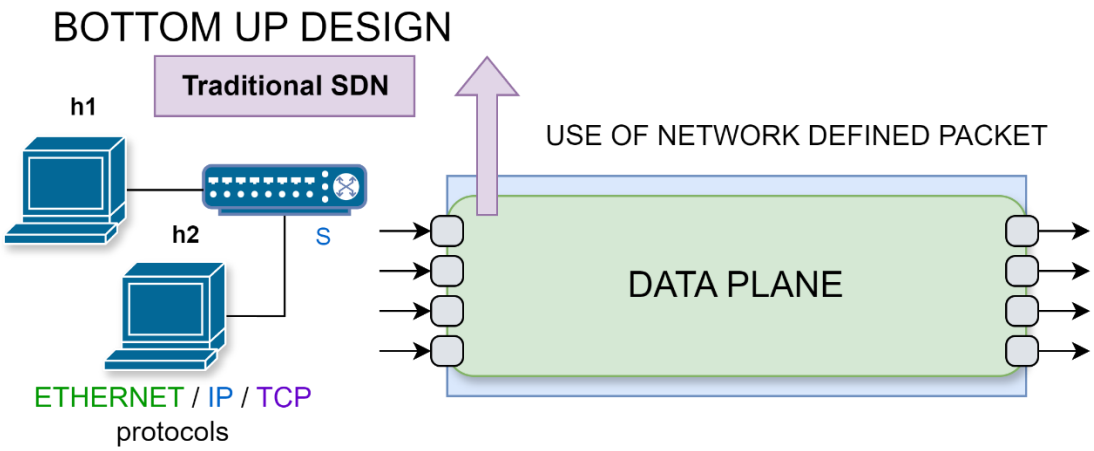


Figura 16. Paradigma tradicional del plano de datos

Para poner solución a esta problemática, y eliminar la rigidez existente en el plano de datos, surge el concepto de redes SDN de nueva generación (Next Generation SDN

– NG-SDN) [11], que permiten especificar el conjunto de protocolos que reconoce el dispositivo que se controla. Este enfoque es inverso al anterior, y es por ello denominado *Top-Down*, pues el administrador del dispositivo es el encargado de establecer una configuración que incluya el formato de cabeceras que se han de reconocer.

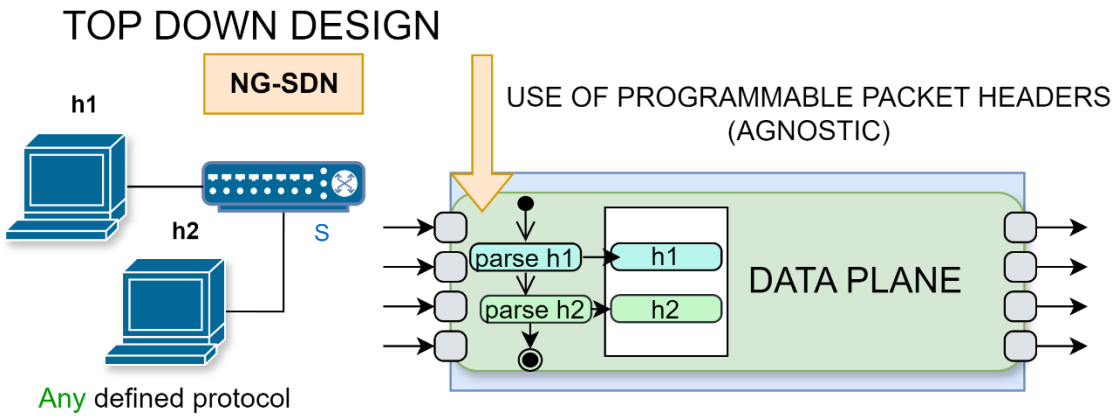


Figura 17. Paradigma renovado del plano de datos

Otros cambios significativos introducidos por esta nueva generación es la programación completa del plano de datos, lo cual permite adoptar el uso de software en los dos planos, y por lo tanto garantiza una mayor flexibilidad, y la posibilidad de implementar algoritmos de mayor complejidad. Esto repercute positivamente en la adopción de nuevos protocolos que se empleen en el ámbito del dispositivo programado.

2.4. El lenguaje de programación P4

En esta sección se abordarán los fundamentos básicos del lenguaje P4, cuyo principal propósito es la programación del procesamiento realizado por el plano de datos de un determinado dispositivo de red NG-SDN, con una arquitectura concreta y especificada. Su nombre tiene origen en el titular “Programando Procesadores de Paquetes Independientes de Protocolo” [4] (traducido al castellano), algo que ya expone de forma clara el objetivo principal para el cual fue creado. Existe una organización que lo mantiene, y actualiza los cambios que se introducen [13].

Particularmente, se ha empleado la última versión, esto es P4_16, que introduce una sintaxis renovada con respecto a la anterior. El lenguaje combina operaciones de bajo nivel, para tener una aproximación cercana al modelo de procesamiento realizado por

el dispositivo que se programa, con operaciones de más alto nivel que facilitan la simplicidad de diferentes instrucciones desarrollables; presentando así una sintaxis similar a la del conocido lenguaje C.

Una de las características más fuertes es el agnosticismo que presenta de cara a su entorno, que implica la independencia del lenguaje con respecto al medio externo del dispositivo que se programa. Ello se extiende a la definición de reglas sintácticas independientes de los protocolos de red empleados y de la propia naturaleza de las interfaces físicas de un determinado dispositivo programable con P4; en este caso los conmutadores SDN. Esto garantiza ventajas de universalidad y aislamiento del lenguaje con respecto a la plataforma, minimizando así el acoplamiento existente.

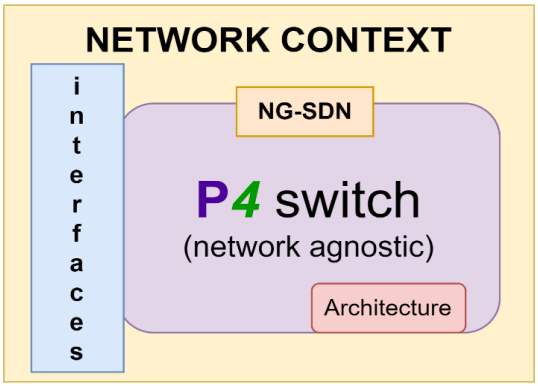


Figura 18. Abstracción del entorno de un conmutador P4

En ese sentido, la Figura 18 muestra adecuadamente el contexto en el que se enmarca un conmutador (switch) programable con P4, donde se puede apreciar un soporte crucial para poder escribir un determinado programa; esto es, la arquitectura, que es un componente lógico que viene implementado en el mismo lenguaje de programación y especifica de forma detallada la estructura de plano de datos que puede ser programada.

Partiendo de lo anterior, cuando se desarrolla un determinado programa, debe estar siempre orientado a una arquitectura, que viene especificada en un archivo fuente P4, y cuya principal función es la declaración de diversos bloques programables ofrecidos por el dispositivo. Siempre será aportada por el fabricante del dispositivo, y contiene definiciones de los citados bloques, que no son más que unidades lógicas en las que se divide la línea de procesamiento del plano de datos o *pipeline*, que determina el orden de ejecución de estos.

En ese sentido, entendemos el procesamiento realizado como una sucesión de etapas, que garantizan control y orden sobre el tráfico de red que circula. De forma

análoga a la función principal presente en la mayoría de los lenguajes de programación, denominada habitualmente *main*, P4 la presenta también, pero con un formato especial que especifica los métodos implementados por un programa, que especifican la interfaz de bloques definida en el archivo de arquitectura.

```
// Package implementation declaration.
/** Declares implemented method defined by
architecture.
*/
V1Switch(
    parseHeaders(),
    verifyChecksum(),
    ingressFlow(),
    egressFlow(),
    calculateChecksum(),
    deparser()
) main;
```

Figura 19. Función main de un programa P4

En la Figura 19 se puede apreciar la definición de los bloques implementados por un programa P4, y vienen definidos por su respectiva arquitectura. El término **main** indica que se trata de la función principal comentada. De esta forma, un determinado programa viene sustentado en una arquitectura que se importa.

En este contexto es preciso indicar las instrucciones de importación de librerías (del sistema o locales), similares a las del lenguaje C:

```
#include <core.p4> // Librería de sistema (incluida por P4C).
#include "../../../common_p4_files/common.p4" // Librería local.
```

Figura 20. Importación de librerías en P4

Es importante entender el ámbito del lenguaje, que es la definición del algoritmo de procesamiento implementado por el plano de datos de un dispositivo programado, por lo que se encuentra optimizado para realizar sobre un paquete una secuencia de acciones, pero no será posible por ejemplo un cómputo complejo o la introducción de bucles de repetición. Todo esto es debido a su propósito tan marcado.

2.4.1. Modelo de compilación de P4

Una vez introducido el lenguaje y su ámbito, es necesario entender su modelo de compilación, y cómo repercute en ello la arquitectura con la que se trabaja. Típicamente encontramos el escenario en el que se desarrolla un programa P4 para una determinada arquitectura adoptada por un dispositivo. Esta será aportada por el fabricante, junto a un compilador de P4, que resultará en la generación de ejecutables y otros ficheros de diversa naturaleza.

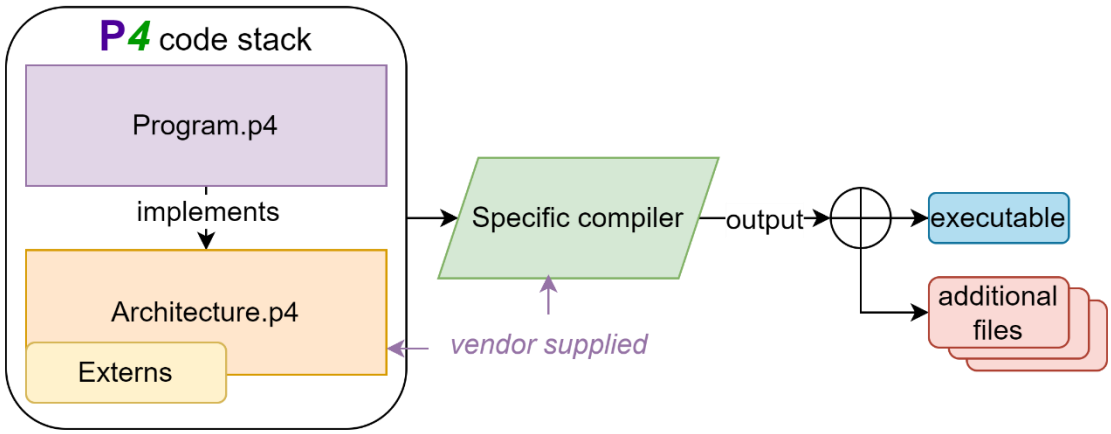


Figura 21. Diagrama del modelo de compilación en P4

Tras la obtención de los citados archivos, estos deben ser empleados para poner en ejecución el programa en el dispositivo para el cual han sido desarrollados. Por un lado, se tiene el ejecutable compilado, que presenta el formato adecuado para ser aceptado por el procesador presente en el conmutador que lo ejecuta, y por otro, en el proceso de compilación se generan archivos adicionales que permiten interpretar de manera adecuada los servicios de gestión que se ofrecen de cara al controlador, lo cual es esencial para permitir un control fiable sobre el plano de datos.

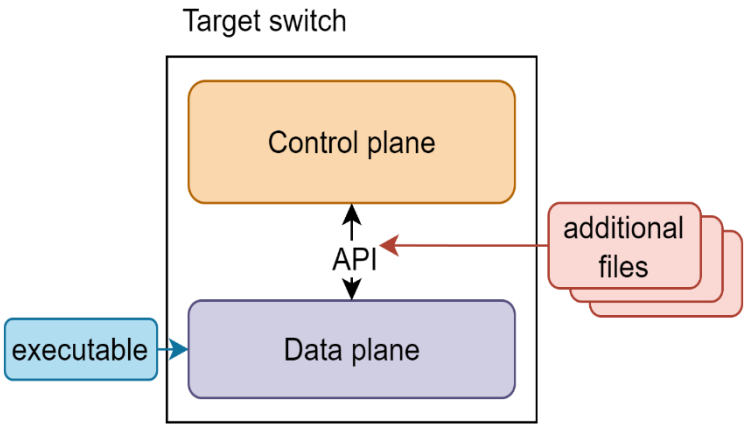


Figura 22. Modelo de ejecución de un programa P4

Para garantizar la universalidad del lenguaje P4 con respecto de la arquitectura que se programe es necesario que todos los fabricantes de dispositivos programables con P4 presenten en sus respectivos compiladores una etapa de procesamiento común. En este contexto se puede encontrar al compilador de referencia de P4, llamado P4C [14]. Su principal objetivo es garantizar la corrección sintáctica de un programa p4, independientemente de la arquitectura integrada en él; esto es, comprobar que verdaderamente se cumplen las normas de sintaxis. Esto es un componente esencial para tener una interpretación unívoca de la escritura correcta de código P4. Este compilador puede ser empleado de forma aislada simplemente con el objetivo de verificar un programa.

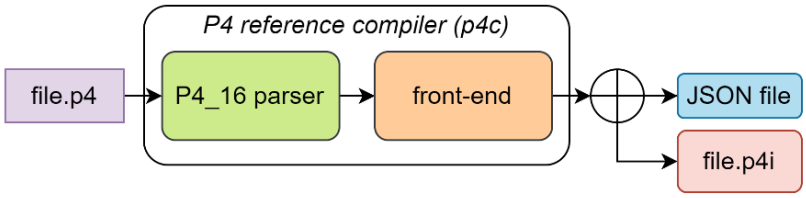


Figura 23. Estructura de procesamiento del compilador de referencia de P4

Siguiendo la funcionalidad aportada por P4C, los fabricantes de dispositivos han de garantizar su integración en los compiladores que estos desarrollen, de tal forma que se cumple con la universalidad del lenguaje ya mencionada con anterioridad.

La estructura de un compilador se divide en dos componentes bien diferenciados, el compilador P4C, y unos componentes *mid-end* y *back-end* que componen el módulo de extensión, que permite obtener el resultado final de compilación para una arquitectura.

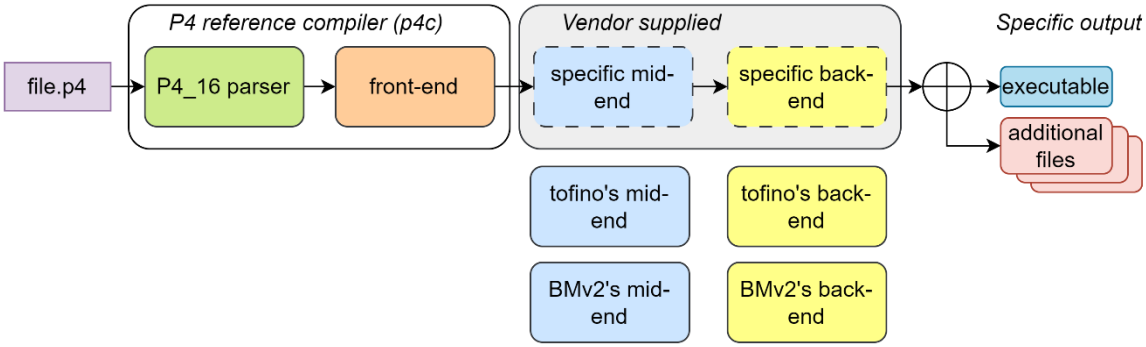


Figura 24. Diagrama de actividad de compiladores P4 específicos

2.4.2. Tipos de datos y estructuras básicas

Como ya sabemos, un paquete de red es una cadena binaria de longitud definida. Dado que el lenguaje P4 está ideado para definir el procesamiento de paquetes en el plano de datos, no existe una gran variedad de tipos de datos.

El tipo de datos principal es el *bit*, que almacena como su nombre de datos un dígito binario. De esa definición básica se pueden crear vectores de una longitud n, introduciendo así el tipo de datos *bit*<*n*>. Ambos son la base de definición de campos y tipos estructurados de organización diversa. También encontramos el tipo de datos *bool*, que permite tratar un bit como valor booleano en expresiones de control. Todos estos tipos de datos vienen definidos en la especificación oficial del lenguaje P4 [1].

A partir de las definiciones básicas, es posible definir tipos de datos especializados, para mapear por ejemplo diversos tipos de direcciones. Cualquier tipo de datos, ya sea original o creado puede ser agrupado en estructurados (struct), que son estructuras de datos que presentan varios campos. Encontramos también de manera análoga un tipo de datos denominado *header*, que permite mapear la cabecera de un determinado protocolo analizado desde un programa. Se recomienda consultar el Apéndice C para profundizar más sobre ello.

Finalmente, llegamos a la estructura más importante y central de todo el lenguaje P4, la tabla (*table*). Consisten en una estructura de selección en función del valor de un determinado campo del ámbito del programa donde se declara, esto es, campos de protocolos, variables de metadatos o incluso variables locales. Cada tabla presenta una clave (simple o compuesta), que especifica el campo (pueden ser varios) con el que se buscará la coincidencia, y que determinará la acción que se desencadenará. Para ello existen tres tipos de coincidencia (match-kinds):

- **Exact**: Se selecciona la entrada que coincida con el valor actual del campo de clave.
- **Ternary**: Se selecciona la entrada que coincida con una máscara del campo (una sección) previamente declarada. Como puede haber varias coincidencias, adicionalmente se introduce una prioridad que servirá para desempatar en casos de conflicto.

- **LPM (Longest Prefix Mask)**: Se selecciona la entrada que coincida con la mayor máscara disponible. Esto lo emplean protocolos como IPv4.

Para modelar el comportamiento de la tabla se especifican acciones, que son funciones optimizadas para ser llamadas desde estas estructuras, y como se muestra en la imagen, permiten ser especificadas para un determinado valor de la clave declarada. Hay de tenerse en cuenta que para un determinado paquete puede darse la situación de no tener coincidencia con ningún valor, y se ejecutará una acción por defecto que previamente viene especificada (Por defecto no se ejecuta ninguna acción). El responsable de poblar la estructura con pares de clave y acción, y especificar la acción por defecto es el controlador, mientras que la estructura y las acciones asociadas se declaran en P4. Adicionalmente, las acciones pueden tener parámetros que son especificados en cada entrada insertada.

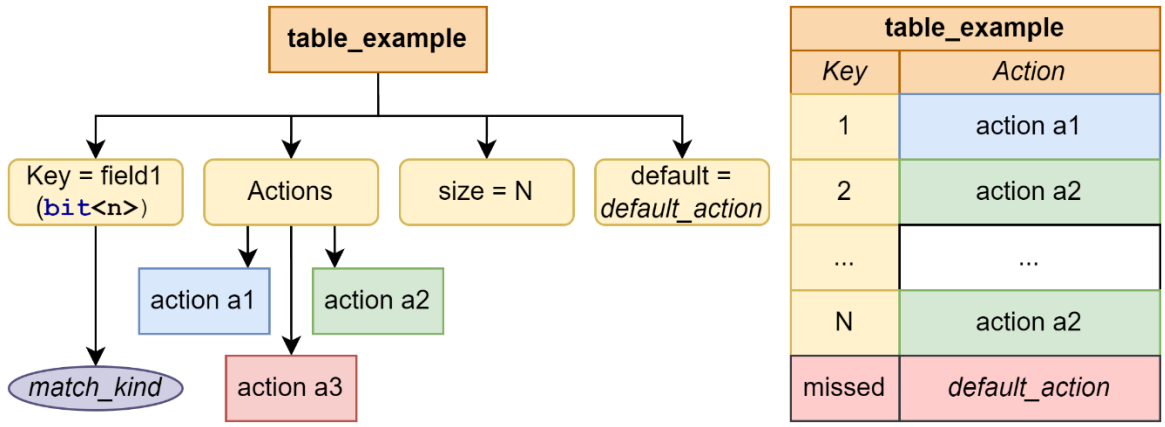


Figura 25. Estructura de una tabla en P4

Cuando tenemos diversas tablas ocurre lo que se conoce como secuencia match-action, que es lo que ocurre en la mayor parte del flujo de un paquete. Cabe destacar que cada tabla sólo podrá ser aplicada una vez, como restricción general impuesta por el lenguaje.

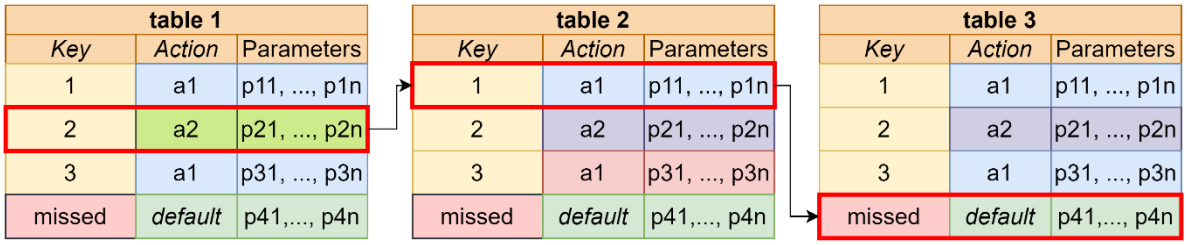


Figura 26. Secuencia match-action de múltiples tablas en P4

Se muestra un ejemplo de declaración de una tabla que enruta tráfico P4:

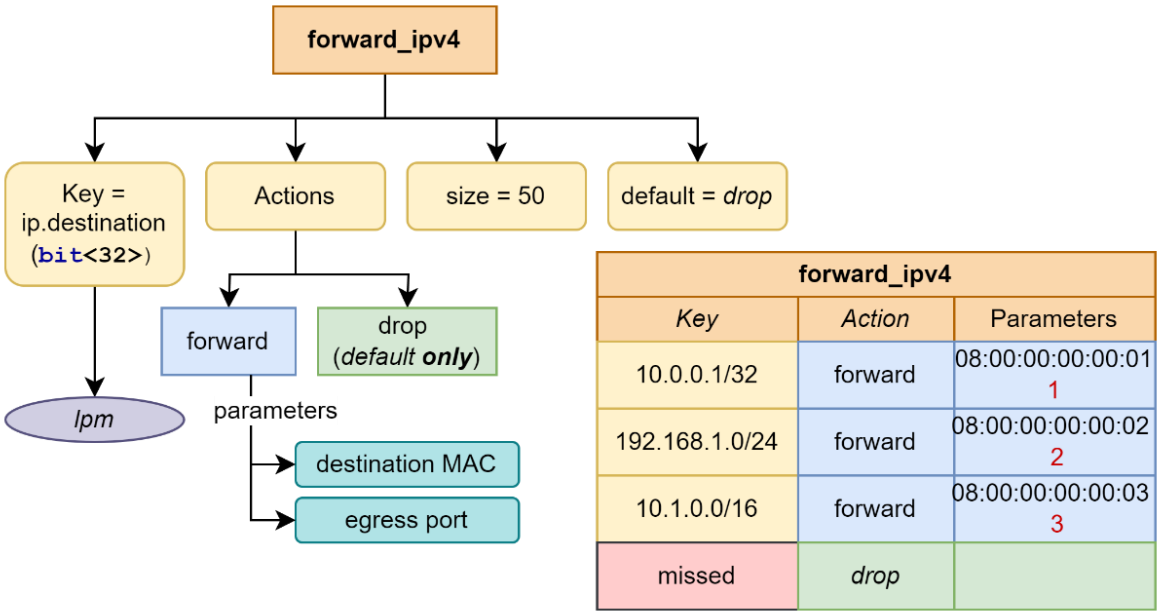


Figura 27. Ejemplo de la estructura de una tabla declarada en P4

```
    action drop(){ // Drop packet & end execution.
        mark_to_drop(std_meta); // set egress_port to drop.
        exit; // end pipeline.
    }
    action forward(MacAddress_t mac_dest, Port_t egress_port){
        ipv4.ttl = ipv4.ttl - 1;
        ethernet.src = ethernet.dest;
        ethernet.dest = mac_dest;
        std_meta.egress_spec = egress_port;
    }
    // Tables
    table forward_ipv4 {
        key = { ipv4.dest : lpm; }
        actions = {
            drop;
            forward;
        }
        default_action = drop();
        size = forward_table_size;
    }
```

Figura 28. Ejemplo de declaración de tabla con acciones en código P4

2.5. Arquitecturas programables

En el contexto de P4, una arquitectura es una abstracción lógica que modela la línea de procesamiento o tubería (pipeline) programable ofrecida por el plano de datos de un determinado conmutador. En ella se puede definir un determinado procesamiento basado en una sucesión de acciones que ocurren sobre un paquete que atraviesa el dispositivo. Una arquitectura viene definida en un archivo de extensión p4, proporcionado siempre por el fabricante del dispositivo. Este archivo debe estar debidamente integrado en el compilador del dispositivo que lo adopta, que permite establecer una plantilla de programación necesaria para estructurar el archivo ejecutable que se genera.

Estas arquitecturas se caracterizan por ser "independientes de protocolo", pues cumplen con el agnosticismo mencionado en la sección del lenguaje P4. Los elementos más importantes que están presentes en su definición son:

- **Identificadores de puertos bidireccionales**: Se encargan de identificar de manera universal un puerto bidireccional del dispositivo (entrada y salida), que están a su vez asociados a interfaces físicas que conectan con el exterior. Es un mecanismo clave para mantener el agnosticismo de cara al exterior, debido a que el programa P4 está aislado de las tecnologías físicas que operan en la red, algo característico de NG-SDN.
- **Bloques programables**: Son espacios programables que permiten desempeñar una serie de acciones definidas al desarrollar un determinado programa.

De esta manera se emplea el lenguaje P4 para seguir la plantilla arquitectónica definida, pero manteniendo su universalidad y aislamiento de cualquier contexto.

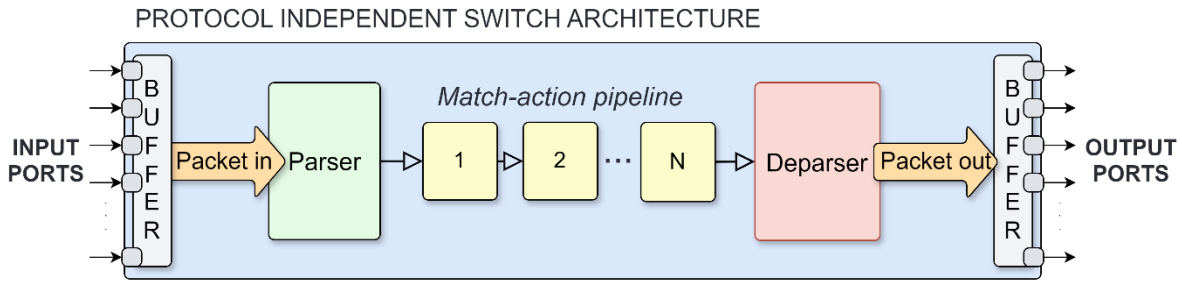


Figura 29. Arquitectura de conmutador independiente de protocolo

Debemos entender que realmente la arquitectura es una abstracción lógica del procesamiento que se aplica a un determinado paquete que atraviesa el conmutador. Esto significa que en muchos casos el hardware presente es un procesador tradicional que ejecuta un programa escrito en lenguaje máquina, que fue generado tras la compilación de un archivo P4. No obstante, en otros casos, se mantiene un hardware dividido en las etapas presentes, y en cada unidad física se incluye un espacio de ejecución relativo a cada bloque programable, para realizar así una ejecución eficiente.

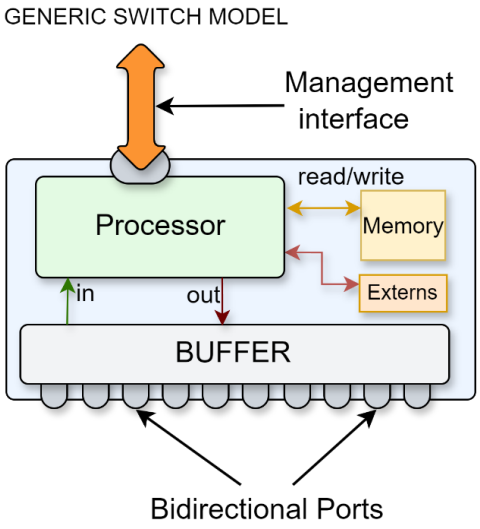


Figura 30. Estructura física de un conmutador P4

En los supuestos de tener bloques hardware especializados por cada etapa definida por un bloque programable, el fabricante puede incluir varias líneas de procesamiento para poner en ejecución varios programas P4, y así aprovechar de forma razonable la infraestructura física disponible.

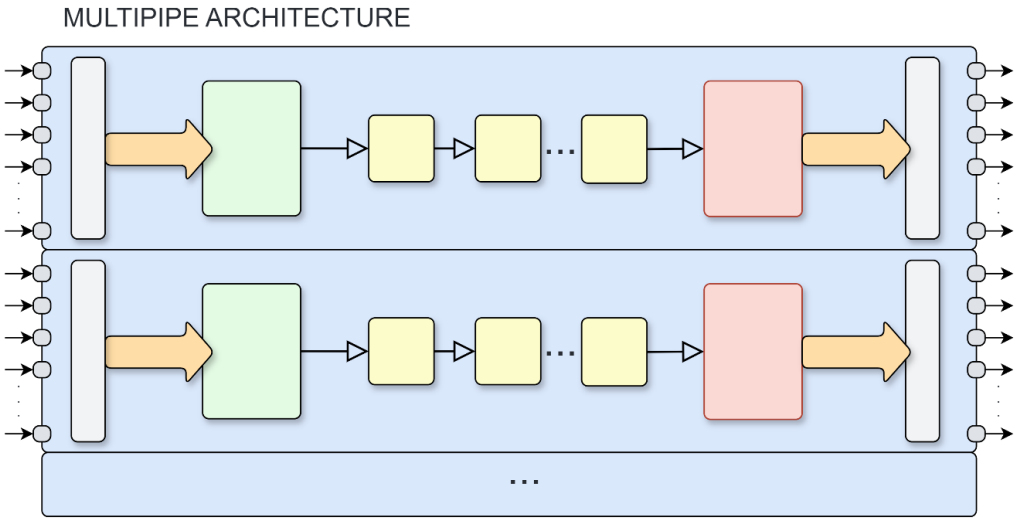


Figura 31. Arquitectura con múltiples líneas de procesamiento

En la siguiente tabla se muestran los bloques que aparecen de forma frecuente en la amplia mayoría de arquitecturas, junto a una descripción de su comportamiento y el propósito que pretenden cubrir.

BLOQUES PROGRAMABLES NOTABLES

1) PARSER

El analizador de cabeceras del paquete es el **bloque programable** encargado de extraer las diferentes cabeceras relativas a los protocolos encapsulados en el paquete que se analiza.

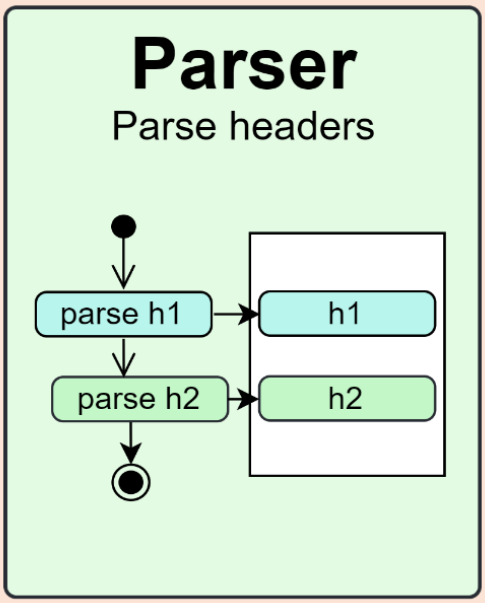


Figura 32. Representación del Parser

En P4, emplea el extern *Packet_in* para leer bloques de bytes del paquete, y mediante una máquina de estados programada, se extraen diferentes cabeceras en función de diversas condiciones preestablecidas. En esta primera etapa se almacenan las cabeceras en estructuras de datos temporales para garantizar un rápido acceso.

2) INGRESS

El flujo de ingreso es un **bloque programable** que se aplica sobre cabeceras analizadas, y constituye una secuencia de acciones programable con P4 aplicable tanto sobre datos relativos a las mismas cabeceras, como a variables de la arquitectura que desencadenan acciones sobre el destino del paquete.

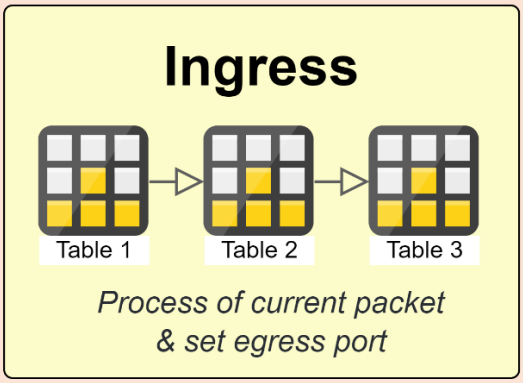


Figura 33. Representación de Ingress

En esta fase se pueden aplicar tanto tablas como diferentes externs ofrecidos por la arquitectura del conmutador, y define un procesamiento del tráfico entrante en el dispositivo que presenta un comportamiento definido por las condiciones impuestas desde el plano de control. Se destaca la clonación hacia la etapa de egreso, para enviar una copia del paquete original a otro destino, la emisión multicast de paquetes y el establecimiento del puerto de salida.

3) TRAFFIC MANAGER

El gestor de tráfico es un componente del dispositivo **no programable**. Su función principal es ordenar y priorizar los distintos paquetes que transitan desde *ingress* hacia *egress*. Mantiene una cola de espera que permite gestionar y extraer de forma ordenada paquetes en escenarios de clonación y envío multicast, mediante parámetros configurables desde el controlador del dispositivo.

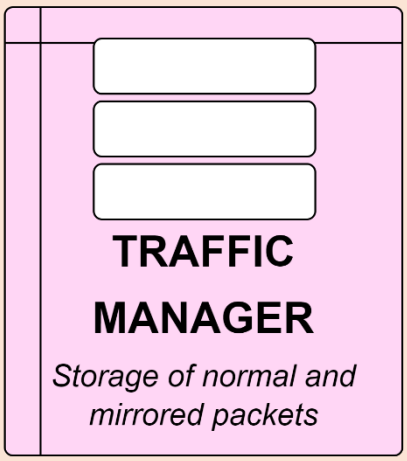


Figura 34. Representación del Traffic Manager

Es de gran importancia para garantizar un correcto flujo del tráfico que circula a través de un conmutador, además de intervenir en el descarte temprano de paquetes que han sido rechazados en la etapa anterior, lo cual descarga de forma considerable las siguientes etapas. En la siguiente figura se puede apreciar su estructura lógica, destacando el soporte existente para las operaciones de clonación de paquetes disponibles: desde *ingress* a *egress*, y desde *egress* a *egress*. Ello posibilita por ejemplo la emisión de una copia al plano de control, para el análisis o procesamiento que implemente.

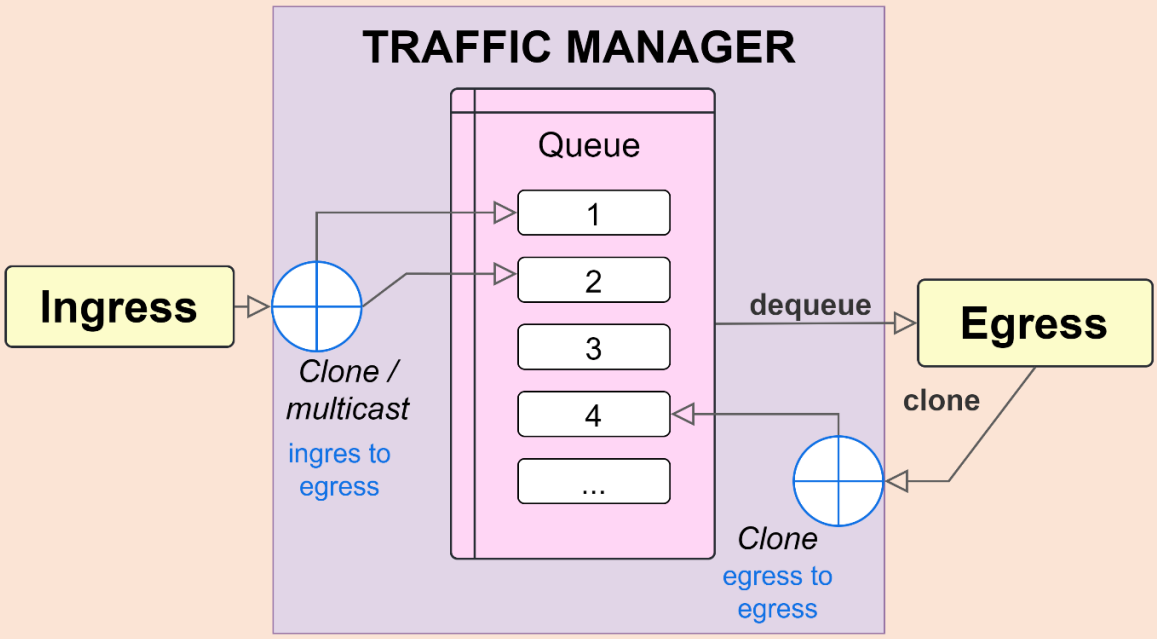


Figura 35. Estructura lógica del componente "Traffic Manager"

4) EGRESS

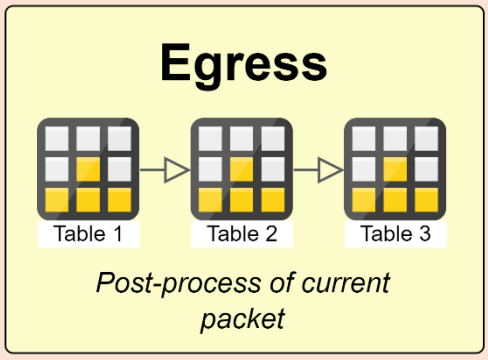


Figura 36. Representación del Egress

El flujo de egreso es el **bloque programable** que define el posprocesamiento que se ha de realizar sobre los paquetes analizados. Permite llevar a cabo modificaciones de campos de las cabeceras que han de ser realizadas estrictamente tras el paso por el *Traffic Manager*, pero en diversas ocasiones es un bloque que puede no tener un comportamiento definido. Muchas de las operaciones disponibles en *ingress* están disponibles, pero en ningún caso permite modificar el puerto del conmutador por el que saldrá un determinado paquete, ya que es competencia exclusiva de la etapa mencionada. En definitiva, es un bloque presente para ser utilizado en situaciones que así lo requieran, como por ejemplo realizar cambios exclusivamente en la copia realizada tras una clonación, o la emisión de estadísticas temporales relativas al gestor de tráfico.

5) DEPARSER

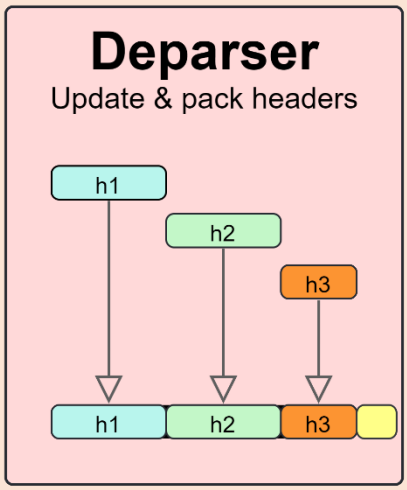


Figura 37. Representación del Deparser

El ensamblador de cabeceras analizadas es el último **bloque programable** de un pipeline de procesamiento típico. Su comportamiento es definible con el lenguaje P4, y presenta una función clara y sencilla, que no es más que la inclusión de las cabeceras almacenadas temporalmente en el paquete original, para incluir los cambios que se hayan podido realizar en el procesamiento del paquete. Emplea un extern denominado *packet_out*, que se encarga de incluir las cabeceras mencionadas en el paquete al que pertenecen, para dejarlo preparado para su emisión desde el dispositivo. Con esta última labor finaliza el camino recorrido por un paquete a través del dispositivo programado, continuando así su navegación por la red.

Tabla 3. Relación de bloques programables notables

2.6. El protocolo de control P4Runtime

P4Runtime es el protocolo de control de referencia de P4 [5], cuya principal utilidad es la manipulación de estructuras y elementos presentes en el Pipeline de un conmutador programable con P4. Permite ser adoptado por diferentes conmutadores con diversos tipos de arquitecturas.

Establece una interfaz (API) declarada en el lenguaje Protobuf [16] dentro del marco de trabajo GRPC [17], una herramienta usuaria del protocolo HTTP. Esta declara el modelo de negocio que ha de seguirse en una comunicación, detallando así las operaciones de comunicación y manipulación que pueden sucederse entre el plano de control y el plano de datos. Este esquema de comunicación viene definido en un archivo de extensión *.proto*, que permite ser compilado mediante el compilador Protoc (disponible en GRPC), para generar así estructuras de código empleables desde cualquier lenguaje de programación. Con respecto al esquema de conexión seguido, encontramos dos **actores** bien diferenciados:

- **Servidor**: Está situado en el conmutador controlable con P4Runtime, y ofrece el conjunto de operaciones definidas por el modelo de negocio de este protocolo de control. Está íntimamente ligado a la naturaleza arquitectónica del dispositivo que lo implementa.
- **Cliente**: Es cualquier controlador software que establezca conexión con el servidor, y presenta un aislamieno total con respecto a la arquitectura del dispositivo controlado.

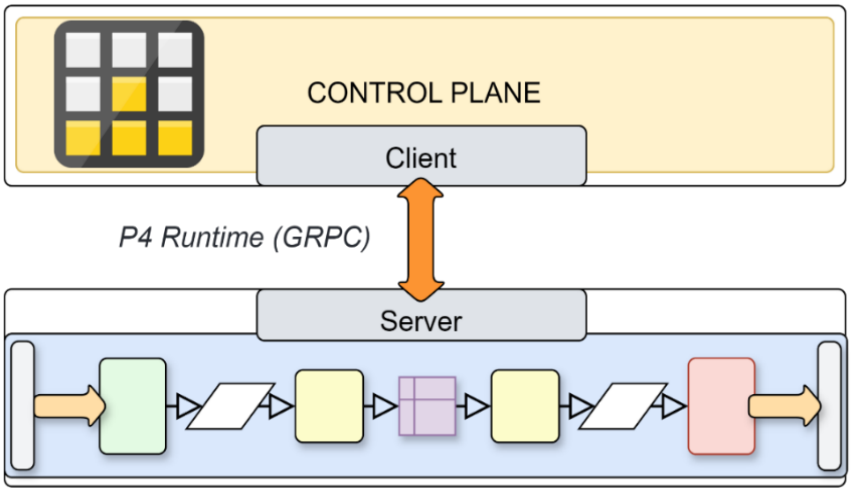


Figura 38. Esquema de conexión P4Runtime

Podemos distinguir dos tipos de operaciones que pueden realizarse en un escenario de control:

- **Operaciones de manipulación**: Permiten manipular las entradas de tablas, y otros objetos que forman parte del pipeline. También permiten cambiar el programa que se ejecuta en un determinado instante.
- **Operaciones de comunicación**: Permiten recibir distintos mensajes procedentes del controlador. Destacamos dos:
    - **Paquetes de CPU**: Son paquetes que se emiten por un puerto especial del conmutador, el puerto de CPU, que mapea una conexión interna existente en el dispositivo que comunica directamente con el controlador mediante el servidor.
    - **Mensajes Digest**: Son mensajes ligeros que permiten encapsular determinados campos que se desean enviar desde el plano de datos.

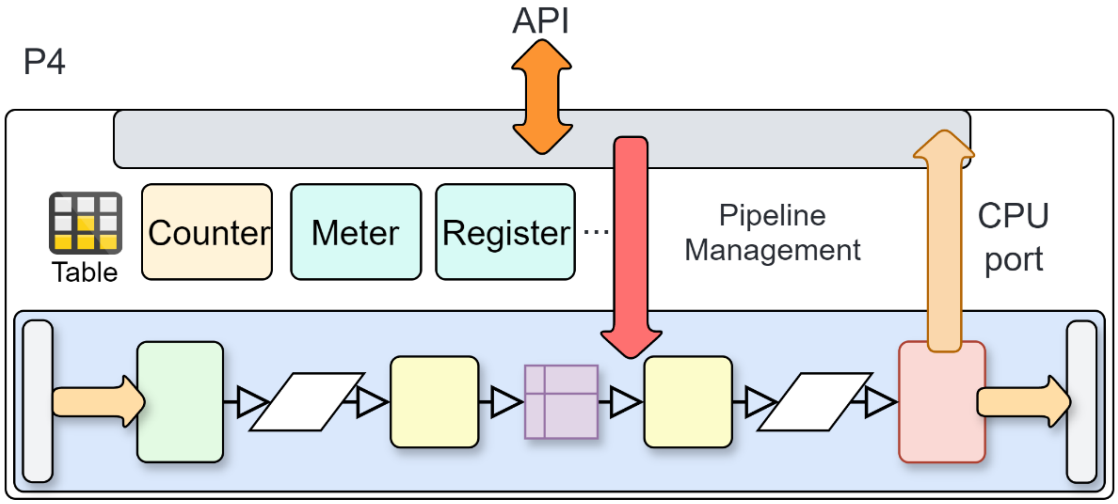


Figura 40. Operaciones ofrecidas por P4Runtime

Cuando se lanza la ejecución de un determinado programa de plano de datos, habrá que especificar un archivo (P4 Info), que especifica el conjunto de entidades manipulables, e identificadores asociados. Esto debe ser ofrecido por los compiladores de aquellas arquitecturas que pretendan adoptar el uso de este protocolo.

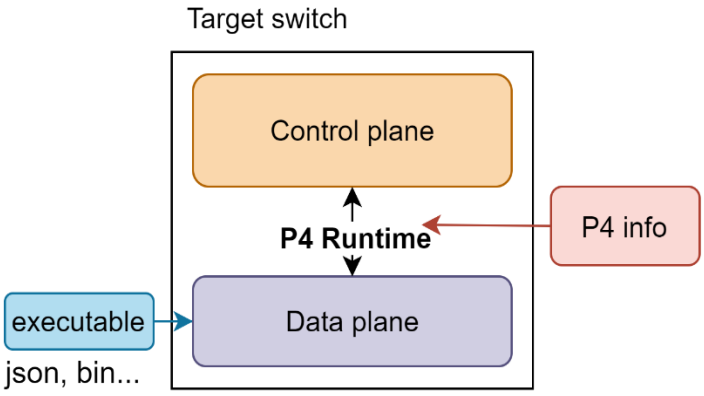


Figura 39. Esquema de ejecución con P4Runtime

Gracias al esquema de conexión existente en este protocolo, las manipulaciones que se lleven a cabo sobre el conmutador podrán ser ejecutadas de manera remota, lo cual es una ventaja para realizar gestiones de una rápida y controlada.

En diversos contextos, puede ser eficiente mantener diferentes controladores con funciones bien diferenciadas. Para ello se introduce lo que se conoce como *roles* que permiten ser asignados a una determinada conexión, y que presentan diferentes permisos de manipulación. Sólo habrá un controlador que puede manipular todo el pipeline y determinar el programa que se pone en ejecución (rol por defecto), y los demás podrán únicamente leer entidades o recibir mensajes del dispositivo. Se denominan controladores primarios (master) o secundarios respectivamente.

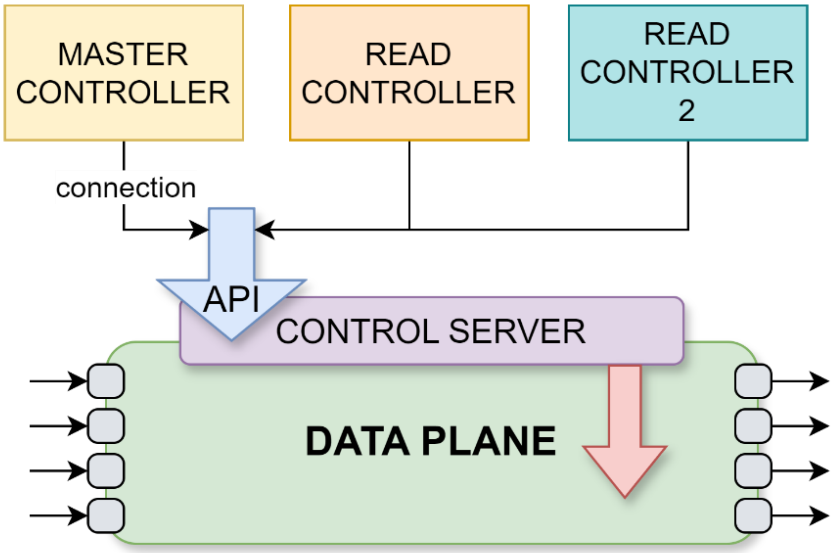


Figura 41. Roles de control

Una excepción que se impone es lo que se denomina *permisos de manipulación ortogonales*, pues permiten segmentar, mantener y repartir de manera aislada el listado de entidades manipulables por distintos controladores primarios, lo cual cumple con la restricción de que sólo puede existir un controlador primario por cada entidad manipulable.

En definitiva, P4Runtime es un mecanismo ordenado que aporta innumerables ventajas en el control de conmutadores P4, y favorece una equivalencia de manipulación lógica independientemente de la arquitectura subyacente.

3

TECNOLOGÍAS EMPLEADAS

Una vez definida la tecnología principal y conocimientos asociados con los que se ha trabajado en este proyecto, es preciso definir cada una de las tecnologías y herramientas que han servido de soporte en el desarrollo del trabajo, o que han presentado un impacto significativo sobre el mismo. Según el propósito que persiguen, encontramos algunas estrechamente ligadas con la propia labor técnica que se ha realizado en el proyecto, otras como componente de apoyo técnico y de documentación, y algunas como base de la correcta gestión y seguimiento de diferentes tareas en las que se ha divido el desarrollo del propio trabajo.

A continuación, se enumeran todas ellas, debidamente referenciadas y acompañadas de una breve descripción sobre sus características más importantes y el uso específico que ha tenido en el desarrollo del trabajo, para que puedan comprenderse adecuadamente las labores específicas en las que han estado involucradas.

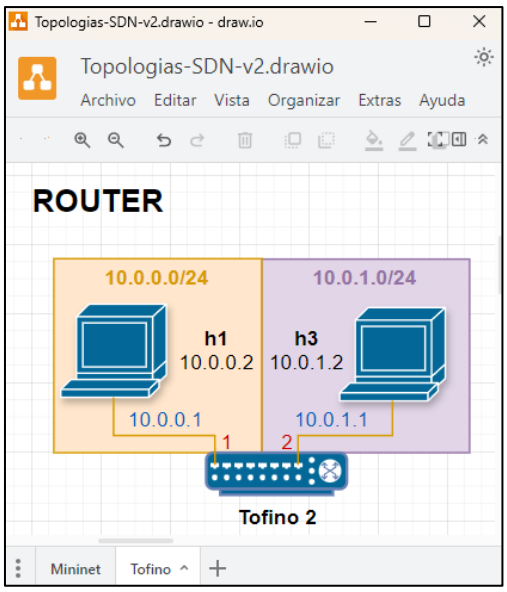


Figura 42. Creación de diagramas con Draw.io

TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
BMv2 **[3]**  BMv2 Logotipo 1. Icono de creación propia para representar a BMv2	**Framework** de libre acceso y distribución que contiene diferentes **arquitecturas de conmutador** simuladas con software. Ofrece además herramientas básicas para probar programas, controladores básicos con línea de comandos (CLI) y compiladores P4 especializados para las arquitecturas contenidas. Podría considerarse la plataforma de desarrollo de referencia en la programación de conmutadores SDN.	Su utilización se ha enfocado principalmente en el trabajo con el switch *simple_switch_grpc* ofrecido por esta plataforma. En etapas tempranas de desarrollo, el controlador básico ofrecido ha sido de gran utilidad para un rápido depurado de código y pruebas rápidas. Es la parte más importante de este trabajo, pues su flexibilidad y gran compatibilidad ha permitido una cómoda validación de programas.
DRAW.IO **[4]**  Logotipo 2. Icono oficial de Draw.io	Software de código abierto que permite crear diversos tipos de **diagramas y modelos**, de manera rápida y flexible. Ofrece multitud de plantillas y recursos gráficos de libre uso, por lo que es una alternativa muy potente.	Empleado para el modelado de: • Topologías de red (formato CISCO). • Modelos de clases (UML). • Diagramas de actividad (UML). • Contenido gráfico para la documentación.

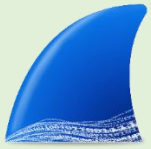
TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
GITHUB **[5]**  Logotipo 3. Icono oficial de Github	**Repositorio de código** que emplea el controlador de versiones *Git*. Permite llevar un registro de los diferentes cambios que se suceden en el desarrollo de un proyecto, así como disponer de copias de seguridad de este.	Empleado para el almacenamiento del proyecto de desarrollo del trabajo, con un control ordenado de los cambios producidos.
GRPC **[22]**  Logotipo 4. Icono oficial de GRPC	**Framework** de comunicación entre clientes y servidores que permite la llamada a procedimientos remotos ofrecidos por un determinado servidor. Permite declarar servicios en formato *Protobuf* [11], que declara y establece el formato de las comunicaciones que se pueden producir, sus campos involucrados, y el flujo que ocurrirá entre peticiones y respuestas. Permite compilar el servicio en objetos de la mayoría de los lenguajes de programación, facilitando la interoperabilidad de diferentes tecnologías que se comunican en un mismo formato.	El principal uso que ha tenido ha sido en el desarrollo del conector P4Runtime, ya que este protocolo está definido en *Protobuf*, y por lo tanto ha sido necesaria una gran profundización en el formato de las comunicaciones desde el lado de cliente. Esto ha permitido ser de base para implementar el conector de control a alto nivel.

TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
INFLUXDB **[6]**  Logotipo 5. Icono oficial de InfluxDB	**Base de datos** de series temporales, que permite la inserción de una o varias medidas (formato numérico) en un determinado instante. Las mediciones se agrupan en lo que se conoce como *punto*, acompañadas frecuentemente por etiquetas explicativas que aportan información adicional. Ofrece herramientas para visualizar gráficos y presenta una integración rápida con los lenguajes de programación más populares.	Utilizada en el almacenamiento y visualización de estadísticas almacenadas por el plano de control de un conmutador monitor, un servicio de red desarrollado en el trabajo. Las mediciones tomadas permiten monitorizar el rendimiento interno del dispositivo que se programa y tener etiquetados el origen y destino y la unidad temporal empleada.
LUA **[7]**  Logotipo 6. Icono oficial de Lua	**Lenguaje de programación** de script interpretado que permite desarrollar extensiones para diferentes aplicaciones informáticas, con un enfoque rápido y ligero.	Ha servido para la creación de una extensión del analizador de paquetes de Wireshark, para poder visualizar correctamente la cabecera personalizada *Flow*.

TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
MAKEFILE **[8]**  Logotipo 7. Icono oficial de GNU	**Herramienta básica** ofrecida por GNU/Linux, que proporciona soporte para la compilación automática de programas y la carga selectiva de dependencias, algo que reduce considerablemente el tiempo de procesamiento en grandes proyectos. Define un lenguaje de script que permite especificar reglas que definen a su vez una sucesión de comandos invocables desde un terminal. Es muy útil para la rápida ejecución de programas con un gran número de dependencias.	Su utilización ha facilitado la compilación de programas P4, que requiere de diversos argumentos y flags especificados en línea de comandos. Además, ha facilitado el flujo de ejecución, levantamiento del entorno y lanzamiento de pruebas de tráfico preparadas, tanto en la plataforma virtual como en el hardware.
MININET **[9]**  Logotipo 8. Icono de creación propia para representar a Mininet	**Emulador de red** para sistema operativo con Linux. Permite crear topologías de equipos y conmutadores con interfaces virtuales del sistema (por ello emulador y no simulador).	Ha proporcionado una estructura fundamental para constituir diferentes topologías que han permitido establecer escenarios de prueba de diferentes servicios ejecutados en el conmutador virtual.

TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
MOBAXTERM [10]  Logotipo 9. Icono oficial de MobaXTerm	Programa **gestor de diversos tipos y formatos de conexiones** a equipos remotos, que proporciona soporte para servidores y clientes. Ofrece un entorno gráfico en el lado del cliente que facilita la navegación por los directorios del sistema.	Empleado para establecer una conexión remota a los equipos de trabajo de laboratorio, que están enfocados al desarrollo en la plataforma hardware, lo cual ha permitido una rápida conexión para permitir un uso flexible en el acceso.
P4C [11]  Logotipo 10. Icono oficial de P4	**Compilador** de referencia del lenguaje P4 (particularmente la versión P4_16), que proporciona el analizador sintáctico básico del lenguaje. Su principal función es garantizar la corrección de la sintaxis de un programa P4, además de ser el compilador básico que ha de ser extendido por las diferentes arquitecturas. Incluye además el compilador especializado para el conmutador *simple_switch*, por ser el dispositivo de referencia.	Ha sido utilizado para compilar todos los programas de los conmutadores con los que se ha trabajado, debido a que es la base de cualquier compilador P4 especializado en una determinada arquitectura. Se destaca el uso de *p4c-bm2-ss*, el compilador oficial para *simple_switch*.

TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
P4Studio - SDE de Intel Tofino *CONFIDENCIAL* *Sólo es accesible por grupos de investigación y personal autorizado*.	**Entorno de programación** comercializado por Intel, que permite desplegar programas p4 en dispositivos hardware de la familia Tofino. Ofrece un simulador del procesador del conmutador físico y diferentes herramientas de conexión, gestión y ejecución en un dispositivo de este tipo.	Adaptación de una selección de servicios de red a la arquitectura Tofino, y posterior validación en el conmutador hardware. Su uso ha garantizado el correcto despliegue de los programas desarrollados.
PYTHON [12]  Logotipo 11. Icono oficial de Python	**Lenguaje de programación** interpretado y multipropósito, que ofrece multitud de librerías que facilitan el análisis de datos y la simplicidad de conexión a servidores, entre otras funcionalidades.	Empleado para: • La programación del plano de control. • Programación de generadores de paquetes para pruebas, con la librería *Scapy*.
TRELLO [13]  Logotipo 12. Icono oficial de Trello	**Gestor de trabajo** que presenta tableros con desglose de tareas agrupadas en listas, que a su vez aceptan una descripción y fechas de compleción, entre otras funciones.	Ha sido de soporte para la gestión de las tareas del trabajo, etiquetadas según a la fase en la que se enmarcan, y agrupadas por listas en función de su estado de realización.

TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
VIRTUALBOX [14]  Logotipo 13. Icono oficial de VirtualBox	**Software** ampliamente conocido de **virtualización local** que permite la creación y el mantenimiento de máquinas virtuales con un determinado sistema operativo, su interacción con el equipo hospedador, y sus recursos asociados en tiempo de ejecución.	Este programa ha sido empleado para montar una máquina con la distribución Linux *Lubuntu*, con los recursos de proceso y memoria necesarios para poder tener el entorno de desarrollo BMv2, y otros componentes.
VISUAL STUDIO CODE [15]  Logotipo 14. Icono oficial de VSCode	**Editor de texto** optimizado para el desarrollo de software, que permite tener los recursos de programación integrados y que acepta extensiones para facilitar las labores de desarrollo.	Empleado para el desarrollo del proyecto P4, la depuración de los programas implementados y la gestión centralizada de terminales de Linux.
WIRESHARK [12]  Logotipo 15. Icono oficial de Wireshark	**Analizador de tráfico de red** (sniffer), que permite capturar paquetes que circulan por una determinada interfaz de red, para su visualización y análisis. Las capturas analizadas pueden ser almacenadas con posterioridad.	Utilizado para el análisis y captura de los paquetes generados por las pruebas, y su posterior procesamiento tras atravesar el conmutador SDN. Además, se han almacenado todas las capturas realizadas, para dejar constancia de los resultados obtenidos tras realizar todas las pruebas definidas.

TECNOLOGÍA	DESCRIPCIÓN	EMPLEO
XTERM **[18]**  Logotipo 16. Icono oficial de Xterm	**Software** empleado en distribuciones GNU/Linux que permite emular un terminal de sistema operativo, ofreciendo las mismas funciones que el nativo. Presenta multitud de utilidades, tales como ser soporte de interacción en escenarios de virtualización, apertura de terminal desde programas, o cubrir una limitación del bash de Linux: abrir un nuevo terminal en otra ventana diferenciada.	En el trabajo ha tenido un uso muy específico en el contexto de la emulación con *Mininet* [10], pues es de gran utilidad para acceder mediante terminal a los distintos hosts de la red virtualizada que se crea. Esto proporciona una interfaz limpia del equipo al que se desea acceder, como puede ser la emisión de tráfico desde él, o consultar parámetros de enlace y/o enrutamiento.

Tabla 4. Tecnologías empleadas en el proyecto

4

METODOLOGÍA DE TRABAJO

Para la gestión de un proyecto con una complejidad considerable, es necesario llevar una monitorización constante de su progreso, así como el establecimiento de planes dinámicos, que permita cumplir con los objetivos planteados en su comienzo, y gestionar ágilmente los recursos temporales disponibles. En este capítulo abordaremos las principales características de la metodología de trabajo seguida, así como detalles importantes de su empleo y aquellos resultados que pueden ser extraídos tras su adecuado empleo.

4.1. Descripción de la metodología

Para elegir correctamente una metodología de trabajo en un proyecto del ámbito tecnológico, es necesario tener presente el contexto en el que se va a desarrollar, los actores principales que van a intervenir, y las características del producto que se desea obtener.

En el caso de este proyecto centrado en el empleo de P4 para la implementación de servicios SDN, se ha ideado una metodología ágil, que permita realizar una gestión eficiente del tiempo y responder de forma dinámica a cualquier imprevisto o dificultad que pudiera aparecer. Una de las características básicas de la realización del trabajo es la naturaleza individual del mismo, por lo que han sido seleccionadas buenas prácticas de métodos de gestión de proyectos, pero con adaptación a este caso tan particular.

Las principales señas de identidad del método de trabajo seguido han sido la monitorización continua del estado del proyecto, para tener un balance constante del progreso existente en cada etapa, y así garantizar siempre una correcta comunicación para facilitar las tareas de tutorización.

Unas de las principales problemáticas que se ha contemplado en etapas tempranas del proyecto ha sido la pronunciada curva de aprendizaje existente en el área de estudio, donde ha sido necesario mantener un registro de las numerosas fuentes de aprendizaje, como son los fundamentos básicos de los protocolos de red, las redes SDN o conocimientos básicos de P4, de los que se ha tenido que realizar una ordenación temática para garantizar la superación de la mencionada curva en el menor tiempo posible.

4.2. Fases de trabajo

Para obtener el resultado final que se pretendía al inicio del proyecto, ha sido necesaria una planificación temporal con naturaleza dinámica, que permitiera organizar los distintos trabajos que tenían que ser completados, y optimizar las dependencias existentes entre diferentes actividades, para emplear un uso razonable del tiempo. Para garantizar un progreso lineal y organizado, fueron estimadas las siguientes fases:

1. **Fase de formación**: Previamente a iniciar con el trabajo en el proyecto, ha sido necesario llevar a cabo un estudio intenso de diferentes fundamentos en los que se basa el trabajo, que podrían resumirse en estudiar los fundamentos del área de las redes de comunicación, entender correctamente los protocolos que se emplean junto a sus respectivos estándares y dónde son consultados, interiorizar adecuadamente los fundamentos teóricos de las redes SDN, y finalmente llegar al ámbito de P4, para visualizar su propósito y el dominio que abarca. Tener todo lo anterior debidamente entendido proporciona una base sólida para idear el trabajo que se deseaba realizar, las funcionalidades que debía cubrir, y también organizar el procedimiento de gestión de la monitorización del estado de este. Esta fase como se puede notar es de una importancia trasversal, que ha determinado un correcto inicio del proyecto, con decisiones claras y nuevos conocimientos aprendidos.

2. **Fase de diseño**: En esta fase, los trabajos fueron centrados en diseñar las soluciones que se pretendía implementar. Ya se conocía adecuadamente el lenguaje P4, por lo que se idearon los servicios con modelos iniciales del plano de datos, y verificando que pudieran ser adoptados adecuadamente por las arquitecturas de conmutadores con las que se ha trabajado. Análogamente, fue diseñado el modelo de controlador SDN empleado en los servicios, mediante el ideado de una librería de Python que permitiera suplir las necesidades de control demandadas por los programas de plano de datos De forma paralela, se fue recopilando una relación de dependencias necesarias para poder montar entornos de desarrollo que permitieran acoger en un futuro los servicios de red que habían sido ideados, así como que se realizaron pruebas básicas en los conmutadores de trabajo, para ir tomando rapidez en el trabajo con ellos.
3. **Fase de desarrollo**: Una vez ideados y plasmados los diseños de servicios de red, es necesario trasladarlos a un entorno tecnológico, que garantice su implementación y posterior ejecución. Para ello, en primer lugar, se ha trabajado con la plataforma de referencia del lenguaje P4, implementada por el conmutador virtual BMv2, y que ha permitido realizar un desarrollo de los algoritmos de plano de datos ideados con los modelos de la fase anterior, manipulando así los recursos ofrecidos por su arquitectura. En este proceso se ha tenido que adaptar y mejorar diferentes aspectos contemplados en los mismos algoritmos, lo que ha supuesto un depurado de código constante a lo largo del desarrollo, que ha derivado en la primera versión de los diferentes programas que definen el comportamiento dinámico del plano de datos que ha de ofrecer un determinado servicio. Seguidamente, y derivado de la existencia de los mismos programas, se desarrolló el controlador SDN que fue ideado en la fase anterior, y que ha permitido la presencia de programas usuarios para controlar los recursos y condiciones de los servicios.
4. **Fase de validación**: Cuando se ha llevado a cabo el desarrollo de los programas definidos, estos tienen sus estructuras y dependencias definidas, es necesario que sean validados desde una perspectiva del comportamiento lógico que presentan, que permitan verificar un funcionamiento adecuado y fiel a las especificaciones definidas. Para ello se han establecido escenarios de prueba, que permitieran visualizar el comportamiento de un determinado servicio en diferentes topologías de red y situaciones, lo que garantiza una

validación del comportamiento dinámico en todos los posibles casos. Esto ha servido para detectar errores en el procesamiento, lo que ha implicado gestionar nuevas tareas de desarrollo nuevamente para que fueran debidamente subsanados, siguiendo con este ciclo hasta validar de forma definitiva el comportamiento e interacción de la funcionalidad del servicio.

5. **Fase de adaptación**: Tras validar los servicios desde un punto de vista algorítmico y funcional, se ha realizado una selección de los servicios más completos, para profundizar en el proceso necesario para trasladar los programas de plano de datos a una plataforma hardware, en nuestro caso el conmutador físico Tofino [24]. Los trabajos han estado centrados en gestionar los recursos y configuraciones inherentes a la plataforma, pero con una descarga considerable al haber validado en el conmutador virtual el servicio desde una perspectiva lógica, pues ha permitido mantener un aislamiento entre ambas agrupaciones de tareas, y agilizar el desarrollo en una plataforma de producción, como es la elegida. Para comprobar finalmente mediante pruebas su correcto funcionamiento, y que ello permitiera realizar reajustes necesarios, y validar de forma definitiva la adaptación llevada a cabo, En esencia, el principal valor que aporta esta fase es el procedimiento de traslación realizado.
6. **Fase de documentación**: Su principal propósito ha sido la redacción paralela de la presente memoria, que recoge toda la documentación técnica que se ha establecido a lo largo del desarrollo del proyecto, por lo que podríamos considerarla como una fase continua y paralela. No obstante, la naturaleza dinámica y cambiante que ha presentado el proyecto ha exigido documentar diversos aspectos cambiantes que repercuten en el resultado final tras la validación definitiva del producto desarrollado. Eso es precisamente algo muy importante a tener en cuenta, esto es, documentar y estructurar aquello que no tiene perspectiva de cambiar, como puede ser un algoritmo validado de forma temprana, o la definición de topologías de red que iban a ser empleadas, ya que permite mantener un equilibrio entre una correcta progresión en el desarrollo, pero sin introducir detalles que pueden ser mutables, y que únicamente crearán confusión al finalizar. Por lo tanto, hay que priorizar la calidad y estabilidad de lo que se documenta, frente a la cantidad en muchos casos.

4.3. Organización de tareas

Para garantizar un flujo ordenado de los trabajos que hay que realizar en las distintas fases, se ha mantenido un tablero de tareas en la herramienta *Trello*. Cada tarea presenta una fase en la que se enmarca, a su vez que presenta dependencias que han de ser completadas con anterioridad a su comienzo.

Así mismo, este registro de tareas ha permitido precisamente facilitar la monitorización que caracteriza a la metodología seguida, siendo un lugar donde se han anotado posibles dificultades, tras su análisis posibles causas, y cuando han sido resueltas, el procedimiento que se ha seguido para que fueran subsanadas, dejando constancia de cara a una posible repetición de situaciones similares.

La estructura del tablero se divide en etapas de trabajo que permiten situar las tareas en función de su estado: iniciades, en proceso o finalizadas. Cada una de ellas se corresponde con un objetivo particular, esto es, que tienen un propósito claramente definido, por lo que el desglose de tareas contempla su simplicidad. Una división de tales características proporciona control sobre la evolución del proyecto en función del estado de sus tareas. Adicionalmente, cada tarea tiene asignado un color dependiendo de la fase en la que se enmarcan, algo importante para visualizar que se completan en el orden correcto.

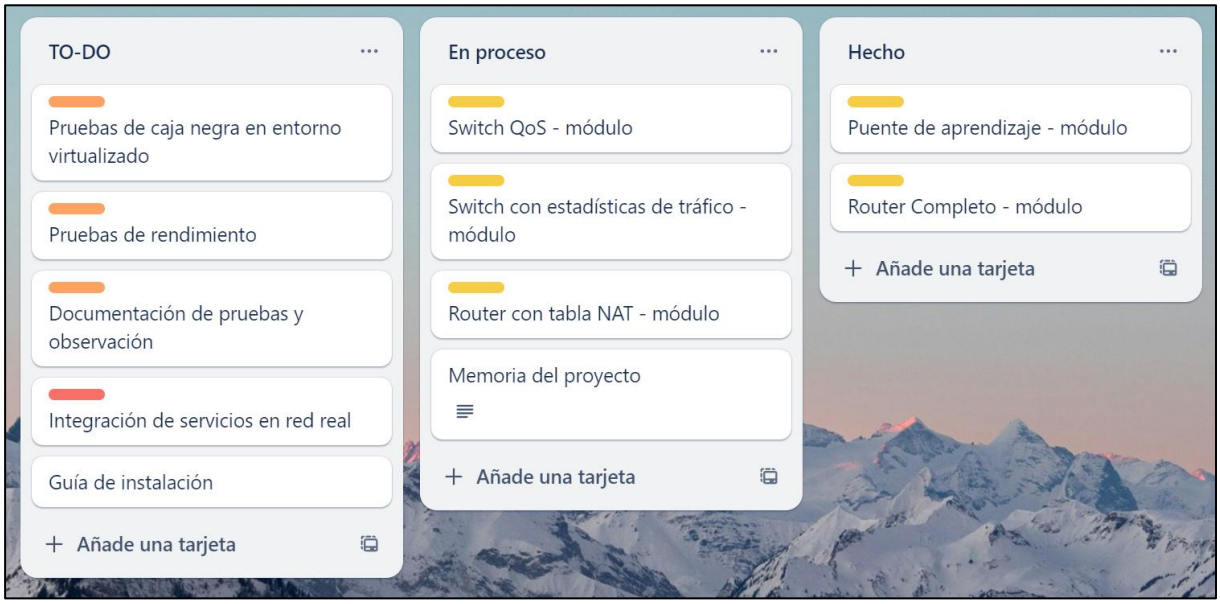


Figura 43. Desglose de tareas en Trello

4.4. Control de versiones

Otro mecanismo de control esencial es el empleo de un repositorio con control de versiones. Permite mantener un registro de los cambios que se van sucediendo, las versiones en las que va evolucionando el proyecto y mantener una estructura de seguridad frente a cualquier eventual pérdida.

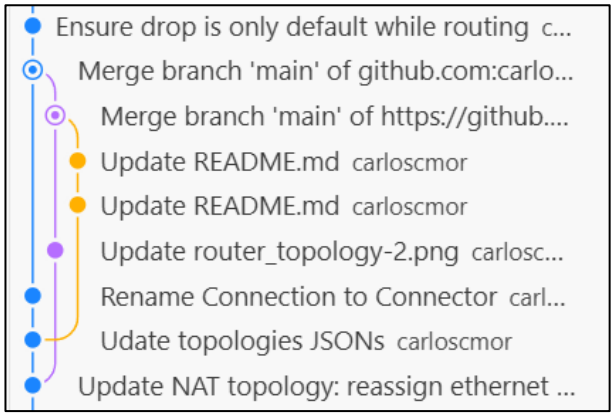


Figura 44. Sección del árbol de versiones del repositorio utilizado

Desde la fase de diseño, el repositorio ha sido debidamente estructurado y configurado para albergar un proyecto de desarrollo con el lenguaje P4; pero no ha sido hasta la de desarrollo cuando ha incrementado de manera notable su actividad, y ha empezado a incluir cambios progresivos desde el estado inicial del proyecto. De alguna manera, se ha intentado que las confirmaciones de cambios incluyeran un nuevo requisito implementado en un determinado servicio, para presentar una limpieza de confirmaciones que posibilitara el retorno a una determinada versión en caso de que fuera necesario.

Esto también ha permitido dejar constancia del trabajo de validación que se ha realizado para verificar los diferentes servicios, pues las versiones han tenido de motivación siempre reflejar cambios que aporten robustez adicional al proyecto existente en un determinado instante.

En definitiva, tras haber expuesto el flujo de trabajo incremental derivado del trabajo con repositorio, se deja constancia que facilita la dinámica establecida por la metodología ágil que hemos expuesto, y constituye una práctica obligatoria en proyectos de esta naturaleza, donde los eventuales cambios pueden presentar consecuencias negativas, y se necesita una seguridad de control y monitorización que refleje los cambios sucedidos.

4.5. Trabajo con diagramas y modelos

Como comentábamos en el apartado relativo a la *fase de documentación*, en proyectos con naturaleza tan cambiante y evolución impredecible, se hace difícil y costoso trabajar con documentación rígida que puede ser fácilmente reemplazada.

Para solucionar esta problemática, se han empleado modelos y diagramas explicativos del software, donde su evolución ha sido constantemente reflejada en ellos. Su principal ventaja es la naturaleza visual que poseen, que ha permitido llevar a cabo las pertinentes modificaciones de manera rápida y flexible.

El mantenimiento de ellos y su paulatina mejora han permitido documentar de forma redactada diferentes aspectos y características del comportamiento de los distintos procesamientos realizados por los programas desarrollados, lo cual es útil para explicar de forma clara y fluida el estado final de un artefacto software.

Por lo que una de las características de la metodología es la utilización de soporte gráfico, lo cual se adapta perfectamente a realidades cambiantes y dinámicas del proyecto, y ha permitido llevar un registro completo del estado de implementación de las distintas funcionalidades.

4.6. Resultados de su empleo

Tras explicar adecuadamente las fases abordadas en el trabajo, y haber expandido aspectos clave de su empleo, se puede afirmar que ha cumplido con los propósitos preestablecidos, y que la característica ágil del método de trabajo se amolda perfectamente con un proyecto de desarrollo de servicios en P4. En los inicios de la fase de formación se buscaba una organización del trabajo que permitiera lidiar sin grandes dificultades las posibles complicaciones derivadas de una temática compleja e incierta por los escasos recursos que pueden existir.

En definitiva, se puede extraer que es necesario tener un registro correcto de la compleción de diferentes objetivos y sus dependencias, y que ello permita mantener una organización tolerante a cambios inciertos.

5

DEFINICIÓN Y MODELADO DE SERVICIOS DE RED

La definición de los conceptos teóricos en los que se basan las redes SDN, entender sus unidades lógicas funcionales y el propósito que pretenden cubrir es de gran importancia para entender el propósito que persigue un servicio de red y la estructura que han de seguir. En este capítulo se definen los servicios ideados, con sus especificaciones lógicas, para que ello permita una adaptación a una determinada tecnología de forma ordenada y con dominio de las operaciones que se suceden, aunque con orientación al lenguaje P4. Su definición viene motivada en mostrar el conjunto de funcionalidades básicas ofrecidas por el plano de datos. Para documentar cada uno de ellos de manera adecuada, se emplearán:

- **Requisitos funcionales**: Plasman las funciones a alto nivel que el servicio ha de ofrecer. Dado que se pretende mostrar el comportamiento de los planos SDN a un nivel funcional, no se incluirán requisitos no funcionales, cuyos objetivos serían imponer restricciones sobre una funcionalidad (tiempos de latencia, recursos empleados…).
- **Estructura de tablas**: Declara la estructura de tablas que permiten seleccionar la acción a ejecutar en función de un determinado campo de un paquete. Se extrae de la definición de los requisitos, y formarán parte del algoritmo definido para el plano de control. Esta estructura es frecuente en planos de datos SDN.

- **Diagramas de actividad**: Diagrama con nodos que definen una determinada acción elemental, y flechas que modelan las transiciones posibles. Empleados para especificar el comportamiento dinámico de los algoritmos definidos tanto en el *plano de datos*, como en el *plano de control*. Se emplean también para documentar el comportamiento de una tabla de P4 definida en el plano de datos.
- **Diagramas de Pipeline**: Adaptación de los diagramas anteriores, que permite mostrar la integración del plano de datos junto a su controlador en el contexto de una arquitectura P4 genérica, esto es, una abstracción que contiene los elementos y bloques que aparecen frecuentemente.

Para confeccionar la documentación técnica de los servicios se han definido en primer lugar los requisitos, que han permitido confeccionar los diferentes algoritmos, para que así posteriormente sean especificados en diagramas de actividad que constituyan modelos técnicos de los servicios definidos. La principal ventaja introducida por el adecuado uso de modelos es la independencia de la documentación con respecto a la tecnología de desarrollo que se desee emplear, aunque esta ha de tener aplicación en el contexto SDN, y debe de incorporar la estructura de datos "tabla" presente en el plano de datos.

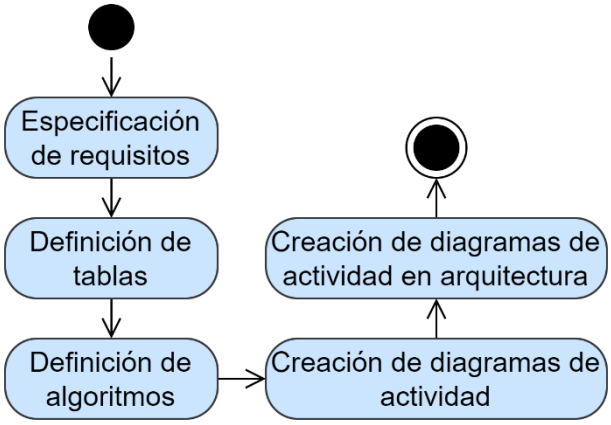


Figura 45. Etapas de modelado de servicios

En el caso del presente trabajo, esta documentación permite emplear la técnica de desarrollo por modelos, que permite posteriormente realizar un desarrollo tecnológico ordenado y cumpliendo principios de calidad algorítmica. Empezaremos definiendo los cinco servicios que han sido ideados, teniendo en cuenta que muchos de los requisitos serán comunes en algunos de ellos, por lo que se indicará en tales casos.

Para permitir una identificación clara y universal de los requisitos definidos, se ha establecido una nomenclatura para introducir un identificador en cada uno de ellos:

- **RF***Ps.r* es el formato de identificador de un **requisito funcional**, donde P es el plano SDN involucrado, teniendo la letra D para el plano de datos y la letra C para el de control, s es el primer servicio donde fue introducido, y r es el orden que presenta dentro del ámbito del mismo servicio. Por ejemplo, RFD1.1 es el primer requisito funcional del plano de datos del primer servicio.

Con posterioridad a la definición de los requisitos, se establecerá los distintos algoritmos, que modelen los dos planos SDN. Serán modelados en diagramas de actividad UML (Lenguaje Unificado de Modelado) [20], junto a adaptaciones a una arquitectura genérica, que permita visualizar la implementación del servicio en un conmutador a alto nivel. Cabe destacar que en algunos servicios se utilizarán requisitos ya definidos en otros anteriores, por eso se ha elegido un sistema de numeración que lo contemple.

5.1. Conmutador con aprendizaje

DESCRIPCIÓN

Un **conmutador con aprendizaje** es un dispositivo tradicional de red que opera hasta el **nivel 2** (enlace) de la torre de protocolos TCP/IP. Permite la interconexión de varias redes físicas de naturaleza Ethernet en una sola, y mediante la memorización de la localización de determinadas direcciones MAC a partir del tráfico que circula por él, puede reenviar paquetes de forma exclusiva por el puerto adecuado para evitar la sobrecarga innecesaria en el resto de la red. Cuando no conoce la localización de un determinado destino, debe realizar un envío de difusión por todos los puertos del dispositivo con conexión activa, a excepción lógicamente del puerto por el que entró el paquete que se procese. Este servicio pretende cubrir la funcionalidad expuesta en un conmutador SDN.

En la siguiente tabla se incluyen los protocolos que reconoce el servicio:

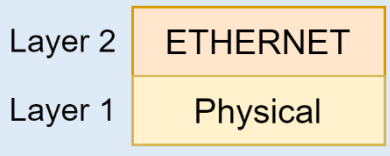
PROTOCOLOS RECONOCIDOS
• Ethernet (nivel de enlace) [23]. 

Tabla 5. Protocolos reconocidos por el conmutador con aprendizaje

REQUISITOS

- **Actores**: Conmutador y controlador.

REQUISITOS FUNCIONALES
PLANO DE DATOS • **RFD1.1** El conmutador ha de admitir y analizar tramas Ethernet. • **RFD1.2** El conmutador ha de reenviar por todos los puertos activos paquetes con dirección MAC de destino no conocida, excepto por el puerto de ingreso. • **RFD1.3** Si el conmutador conoce el puerto de salida asociado a la dirección MAC de un paquete, deberá reenviarlo por ese puerto. • **RFD1.4** El conmutador deberá implementar un mecanismo de aprendizaje de direcciones MAC, asociando cada dirección recibida al puerto de entrada. Sólo será realizado en casos de desconocimiento de la dirección, o cuando el puerto asociado haya sido alterado. • **RFD1.5** El conmutador sólo aprenderá en supuestos de desconocimiento de una dirección MAC o cuando haya cambiado de puerto. • **RFD1.6** El conmutador deberá notificar al controlador cada vez que se aprende una dirección MAC y su respectivo puerto. **PLANO DE CONTROL** • **RFC1.1** El controlador deberá crear un grupo de difusión (multicast) que incluya todos los puertos del conmutador activos. • **RFC1.2** El controlador deberá recibir mensajes de aprendizaje, y almacenar con posterioridad las direcciones aprendidas, junto a su puerto de localización. • **RFC1.3** El controlador deberá establecer una marca temporal en cada una de las direcciones aprendidas • **RFC1.4** El controlador deberá borrar las direcciones aprendidas una vez estas hayan caducado.

Tabla 6. Requisitos funcionales del conmutador con aprendizaje

ESTRUCTURA DE TABLAS

Para implementar la lógica de aprendizaje, se han ideado dos tablas de selección de acciones:

- **Ethernet Timeout**: Se encarga de determinar si hay que aprender una dirección MAC origen. Presenta una única acción que implementa la lógica de aprendizaje. Además, presenta temporizadores de caducidad en sus entradas, que expiran cuando agotan un tiempo configurado tras no ser referenciadas (ausencia de match).

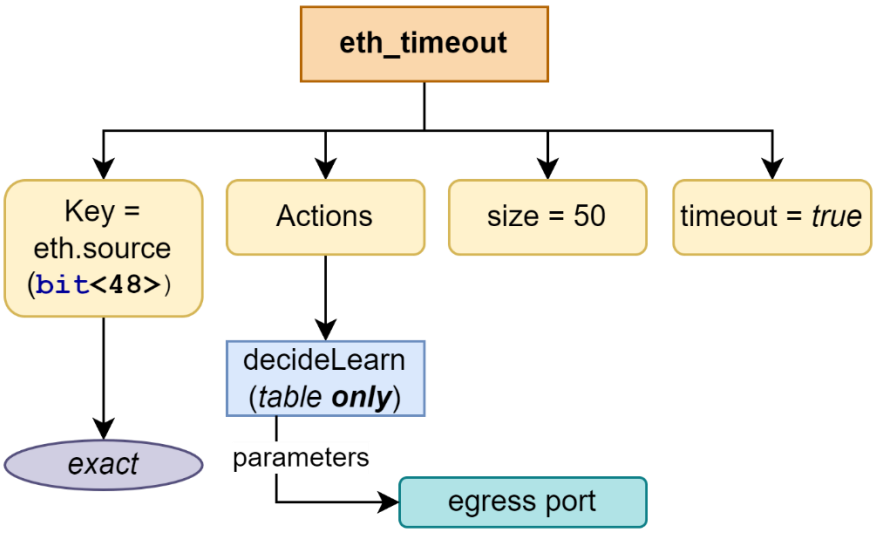


Figura 46. Estructura de la tabla P4 llamada " Ethernet Timeout "

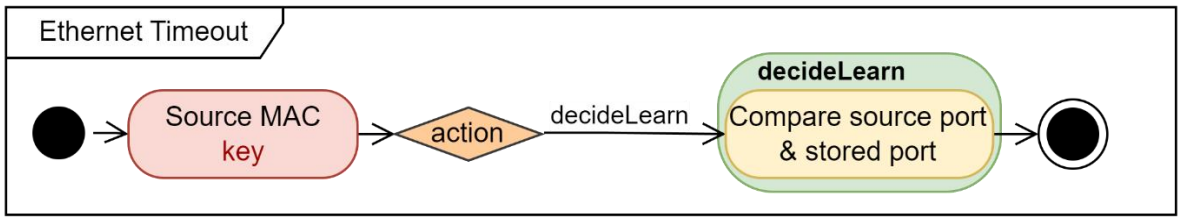


Figura 47. Diagrama de actividad de la tabla P4 llamada “Ethernet Timeout”

- **Forward Ethernet**: Tomando como clave la dirección MAC de destino, decide el modo de emisión que hay que aplicar (por un puerto o por difusión).

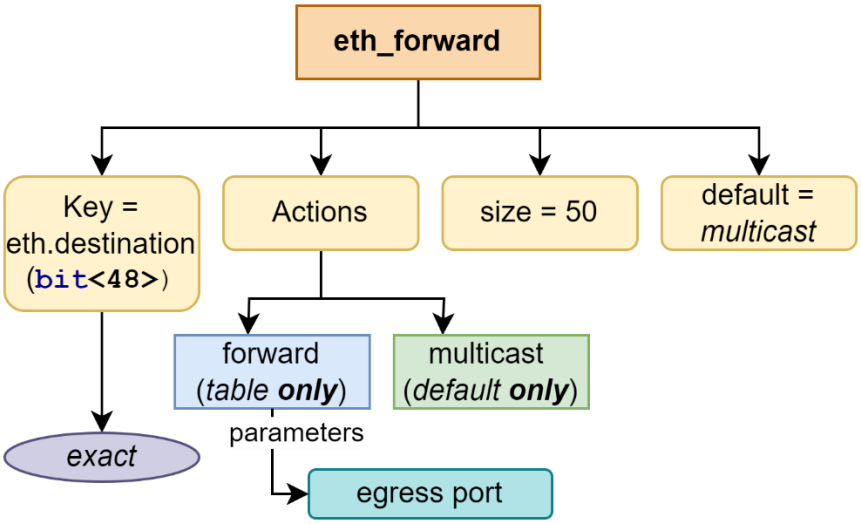


Figura 48. Estructura de la tabla P4 llamada "Forward Ethernet"

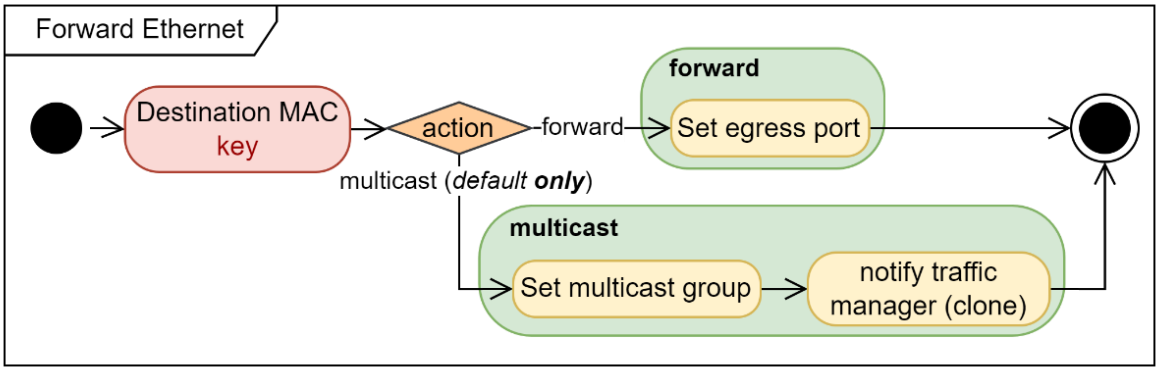


Figura 49. Diagrama de actividad de la tabla P4 llamada “Forward Ethernet”

DIAGRAMAS DE ACTIVIDAD

En primer lugar, se expone el diagrama de actividad del parser, que analiza las cabeceras *Ethernet*:

Figura 50. Máquina de estados del parser del conmutador con aprendizaje

PLANO DE DATOS

El diagrama modela el algoritmo de plano de datos incluye el flujo especificado en los requisitos funcionales.

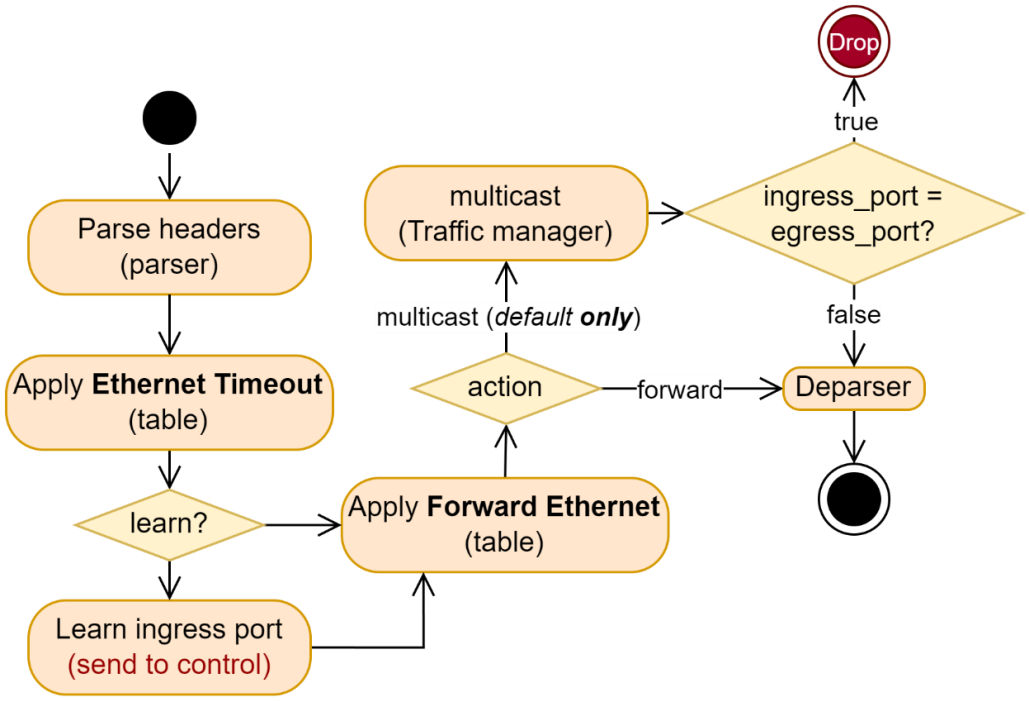


Figura 51. Diagrama de actividad del conmutador con aprendizaje

PLANO DE CONTROL

El controlador establece el grupo de difusión, y posteriormente crea dos hebras de ejecución: Una encargada de memorizar el aprendizaje, y la otra de borrar las direcciones caducadas.

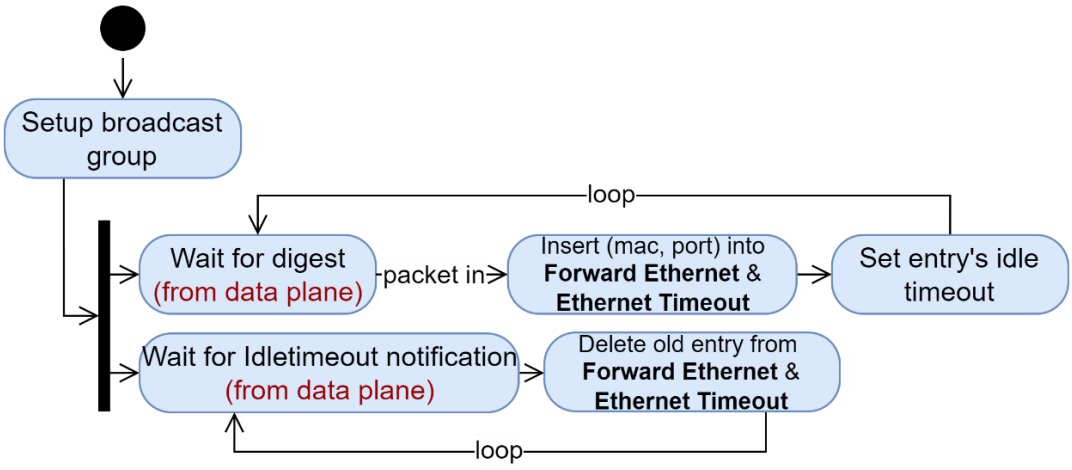


Figura 52. Controlador para el conmutador con aprendizaje

DIAGRAMA DE PIPELINE

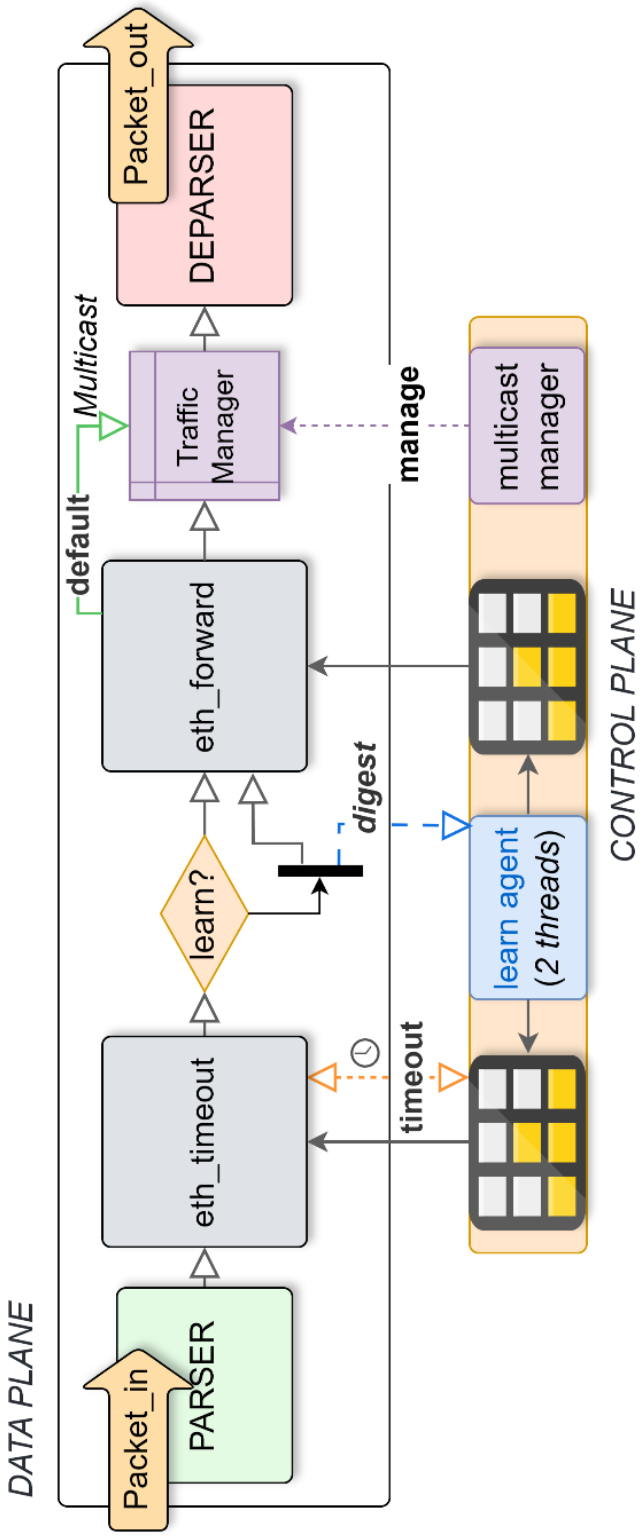


Figura 53. Pipeline lógico del conmutador con aprendizaje

5.2. Router con firewall de filtrado

DESCRIPCIÓN

El servicio pretende cubrir la funcionalidad básica de un **router**, que es un dispositivo de red que opera en el **nivel 3 (red)** de la torre de protocolos TCP/IP, y permite la interconexión lógica de diferentes redes que pueden presentar una tecnología física subyacente de distinta naturaleza, pero que en esencia presentan a un nivel lógico de red el protocolo IPv4. Cuando se interconectan redes, también se están diferenciando espacios de direccionamiento IP, por lo que cada conexión deberá corresponderse con una subred diferente. Las funcionalidades ofrecidas comprenderán el enrutamiento de paquetes IPv4, con el empleo de la conocida *tabla de enrutamiento*, además de la gestión de un firewall de filtrado, para bloquear paquetes por diversos campos que contiene este último.

En la siguiente tabla se incluyen los protocolos que reconoce el servicio:

PROTOCOLOS RECONOCIDOS	
• Ethernet (nivel de enlace) [23]. • IPv4 (nivel de red) [24]. • TCP (nivel de transporte) [25]. • UDP (nivel de transporte) [26].	Layer 4 TCP/UDP Layer 3 IP Layer 2 ETHERNET Layer 1 Physical

Tabla 7. Protocolos reconocidos por el router con firewall

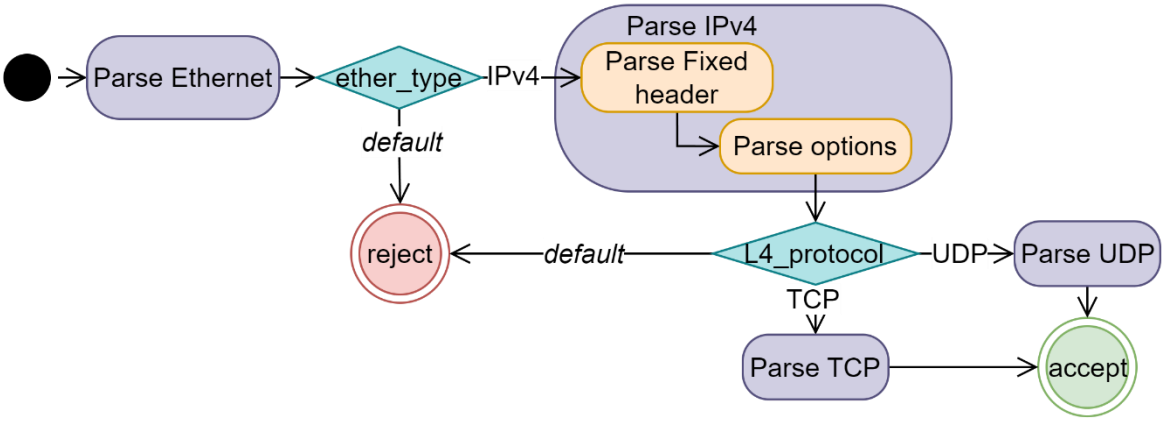


Figura 54. Máquina de estados del parser del router con firewall

En este servicio, el diagrama de procesamiento del parser se ha incluido con anterioridad, para que así pueda ser visualizado junto a la tabla de protocolos reconocidos.

REQUISITOS

- **Actores**: Conmutador y controlador.

REQUISITOS FUNCIONALES
PLANO DE DATOS
• **RFD2.1** El conmutador ha de admitir y analizar cabeceras Ethernet, IPv4, TCP y UDP.
• **RFD2.2** El conmutador ha de rechazar paquetes que no tengan la cabecera IPv4.
• **RFD2.3** El conmutador ha de mantener un firewall de filtrado, que permitirá descartar paquetes por direcciones IPv4 origen y destino y por puertos de transporte origen y destino.
• **RFD2.4** El conmutador ha de encaminar el paquete en función de su dirección IPv4 destino. En casos donde no se conozca la localización de destino, el paquete ha de ser descartado.
• **RFD2.5** El conmutador ha de decrementar el TTL de la cabecera IPv4 (exigido por el protocolo al atravesar un router L3).
• **RFD2.6** El conmutador ha de establecer nuevas direcciones MAC origen y destino. Esto es consecuencia directa del cambio de red local.
• **RFD2.7** El conmutador ha de actualizar la suma de comprobación (checksum) de IPv4.
PLANO DE CONTROL
• **RFC2.1** El controlador ha de insertar valores de entradas en las tablas del plano de datos.

Tabla 8. Requisitos funcionales del router con firewall

ESTRUCTURA DE TABLAS

Para poder cumplir con los requisitos, se establecen las siguientes tablas:

- **Firewall**: Tabla que permite implementar el comportamiento de un firewall de filtrado. Su clave es múltiple, y permite descartar o permitir tráfico en función de cualquier combinación de los campos: direcciones IPv4 y puertos (TCP o UDP). Tiene como acción por defecto el descarte, pero es modificable desde el controlador.

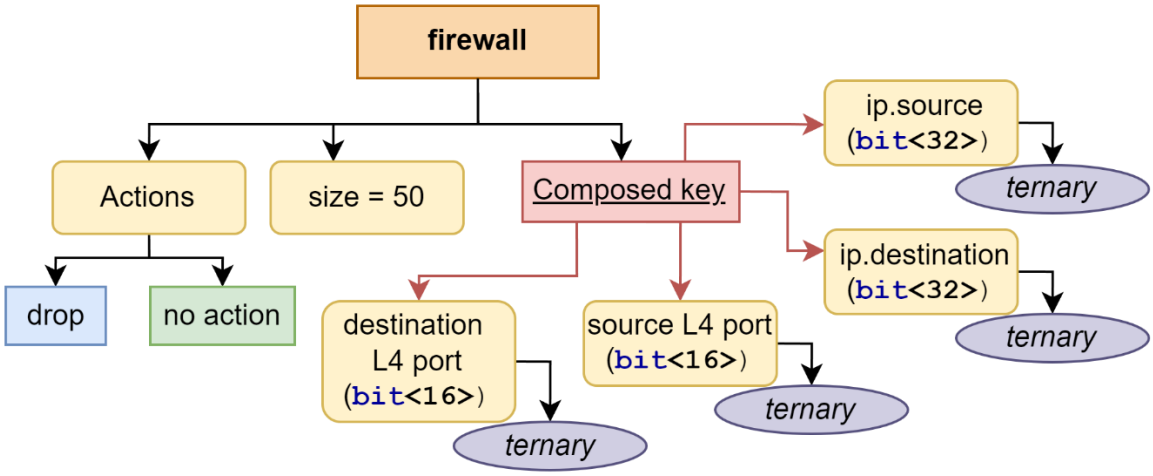


Figura 55. Estructura de la tabla P4 llamada "Firewall"

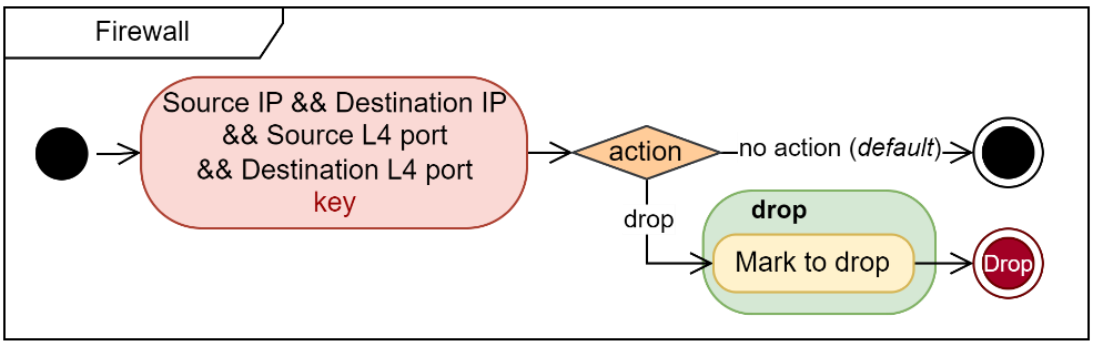


Figura 56. Diagrama de actividad de la tabla P4 llamada “Firewall”

- **Forward IPv4**: Tabla que implementa el flujo de encaminamiento IPv4, decrementa el TTL y establece la dirección MAC del siguiente destino (salto) y el puerto de egreso. Su clave es la dirección IPv4 de destino, y se busca la entrada con mayor máscara de red entre las que presentan coincidencia (LPM).

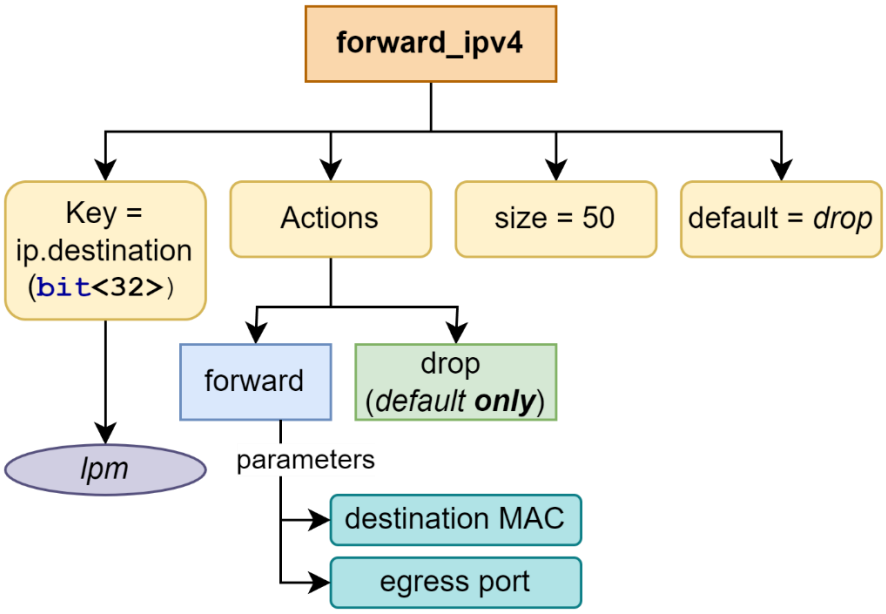


Figura 57. Estructura de la tabla P4 llamada "Forward IPv4"

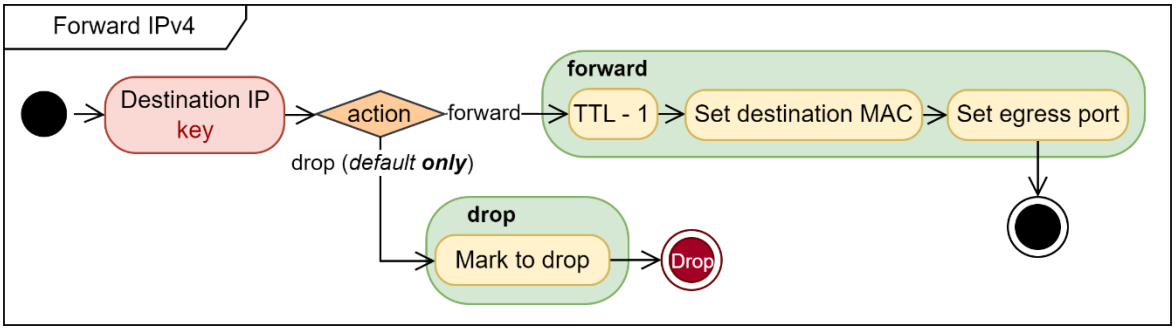


Figura 58. Diagrama de actividad de la tabla P4 llamada “Forward IPv4”

- **Port-Mac mapping**: Tabla que permite establecer direcciones MAC de origen en función del puerto de egreso por el que sale un paquete del conmutador.

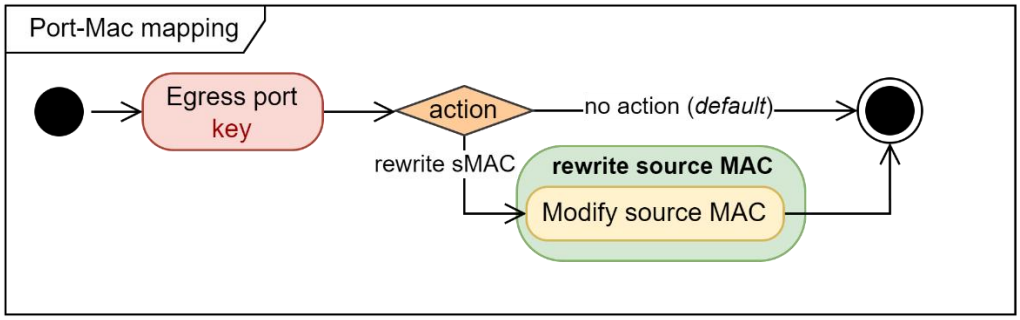


Figura 59. Estructura de la tabla P4 llamada "Port-Mac mapping"

DIAGRAMAS DE ACTIVIDAD

En este apartado se exponen los diagramas de actividad resultantes de los requisitos definidos, que además incluyen las tablas de selección previamente definidas.

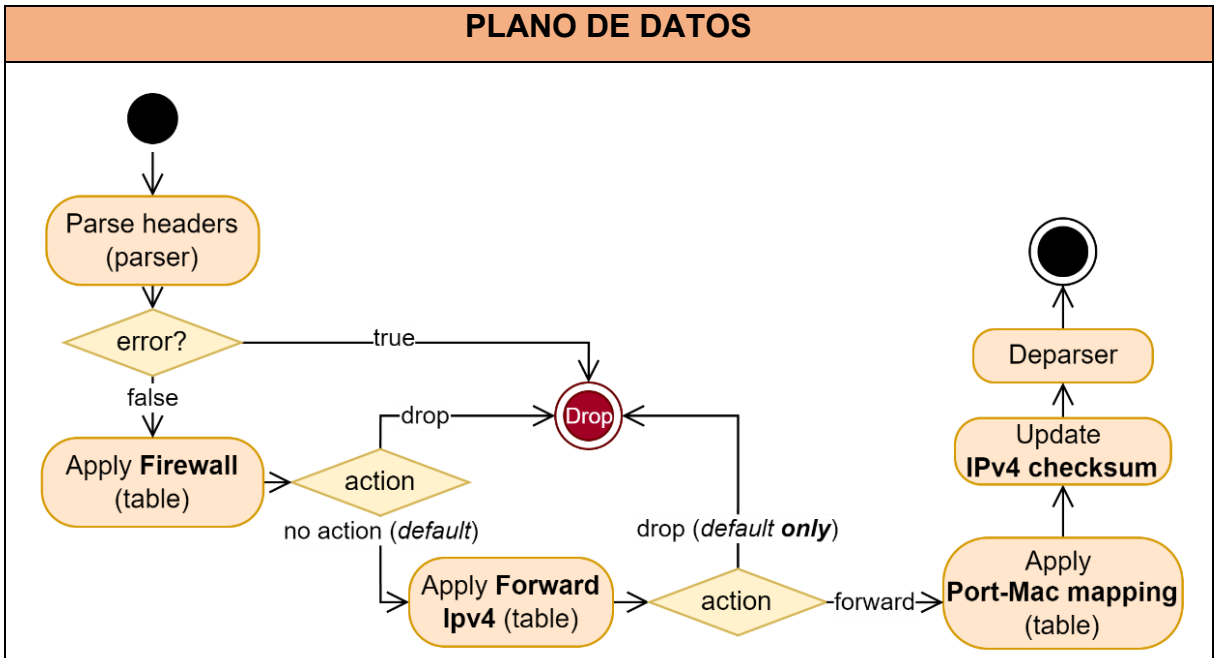


Figura 60. Diagrama de actividad del router con firewall

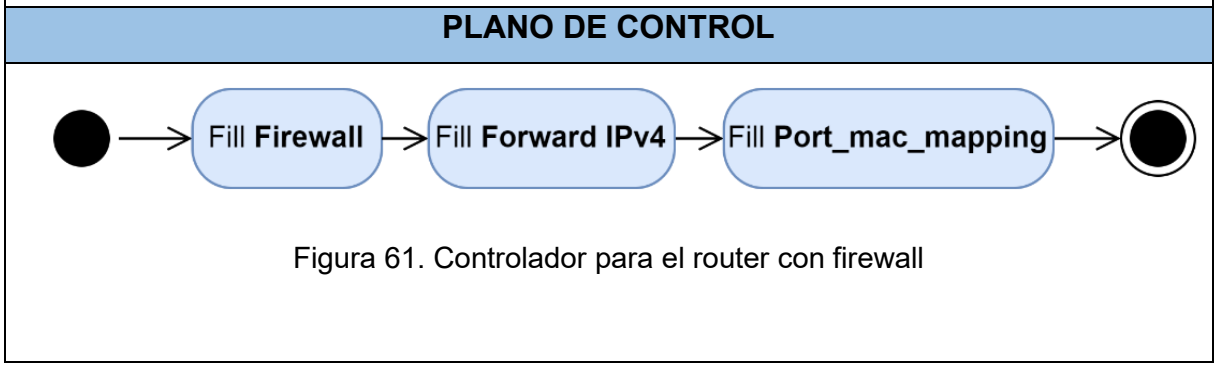


Figura 61. Controlador para el router con firewall

DIAGRAMA DE PIPELINE

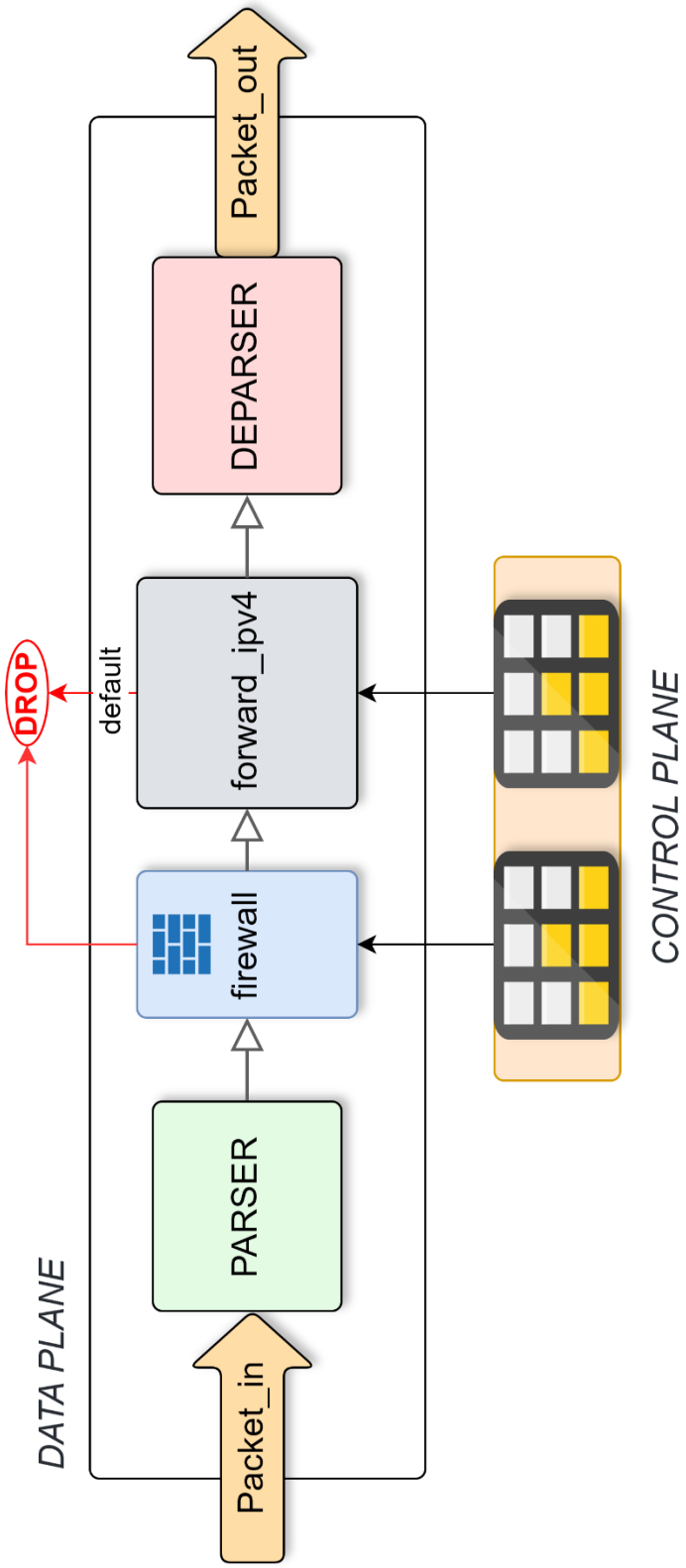


Figura 62. Pipeline lógico del router con firewall

5.3. Router con traducción NAT

DESCRIPCIÓN

En el protocolo IPv4 se declaran bloques de direcciones que se corresponden con redes privadas [33], que no pueden ser visibles en redes públicas. Partiendo del router básico (servicio anterior sin el firewall), se pretende implementar un servicio que permita la interconexión entre redes públicas y privadas en los protocolos TCP y UDP, de tal manera que las traducciones se especifiquen en una tabla denominada NAT. Esto permitirá exponer una dirección IPv4 de forma pública, y los equipos internos de la zona privada serán identificados con un puerto de transporte, dirección presente tanto en TCP como en UDP.

En la siguiente tabla se incluyen los protocolos que reconoce el servicio:

PROTOCOLOS RECONOCIDOS	
• Ethernet (nivel de enlace) [23]. • IPv4 (nivel de red) [24]. • TCP (nivel de transporte) [25]. • UDP (nivel de transporte) [26].	Layer 4 TCP/UDP Layer 3 IP Layer 2 ETHERNET Layer 1 Physical

Tabla 9. Protocolos reconocidos por el router con traducción NAT

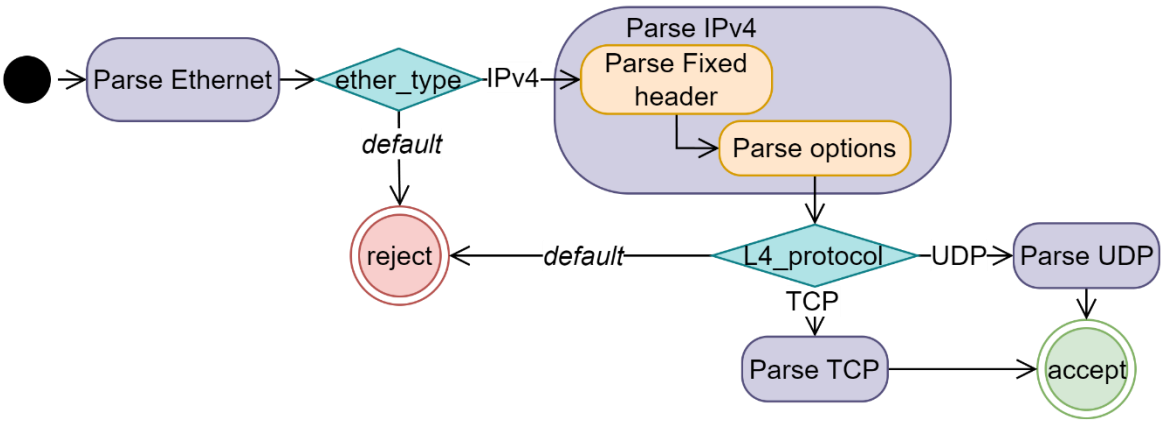


Figura 63. Máquina de estados del parser del router NAT

La traducción de direcciones entre espacios públicos y privados requiere de una estructura que en la interfaz pública de un router emplee una dirección pública que sea accesible, y mapee los equipos de la red privada que conecta con puertos de transporte virtuales. En la imagen se muestra lo que se comenta.

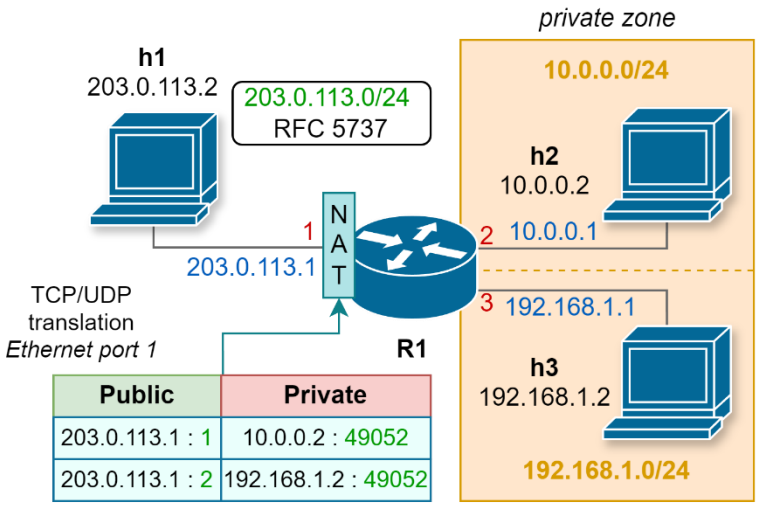


Figura 64. Ejemplo de tabla de traducción NAT

REQUISITOS

- **Actores**: plano de datos y plano de control.
- **Requisitos ya definidos que son incluidos**:
  - **RFD2.1 RFD2.2 RFD2.4 RFD2.5 RFD2.6**

REQUISITOS FUNCIONALES
PLANO DE DATOS
• **RFD3.1** El conmutador debe garantizar que no se reenvían paquetes con origen privado a un espacio público.
• **RFD3.2** El conmutador deberá traducir de forma exclusiva paquetes TCP o UDP.
• **RFD3.3** El conmutador deberá traducir direcciones IPv4 públicas a privadas y viceversa (origen y destino) cuando sea configurado para ello.
• **RFD3.4** El conmutador deberá actualizar la suma de comprobación (checksum) de los protocolos IPv4, TCP y UDP.
PLANO DE CONTROL
• **RFC3.1** El controlador ha de insertar valores de entradas en las tablas del plano de datos, entre ellas las entradas de traducción precisas.

Tabla 10. Requisitos funcionales del router NAT

ESTRUCTURA DE TABLAS

Tablas ya definidas que son incluidas: Forward IPv4 y Firewall (del servicio 2).

Para poder cumplir con los requisitos, se establecen las siguientes tablas:

- **Destination NAT**: Tabla que presenta como clave el puerto de ingreso en el dispositivo, la dirección IPv4 de destino y el puerto de transporte de destino. Se encarga de modificar la dirección y el puerto de la clave, tomando los nuevos valores especificados en la acción *destination translation*.

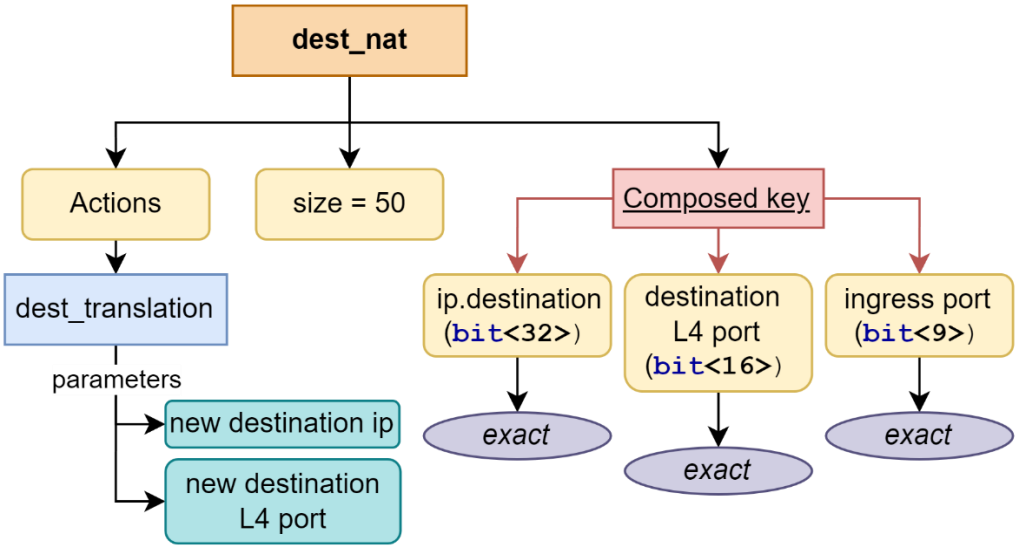


Figura 65. Estructura de la tabla P4 llamada "Destination NAT"

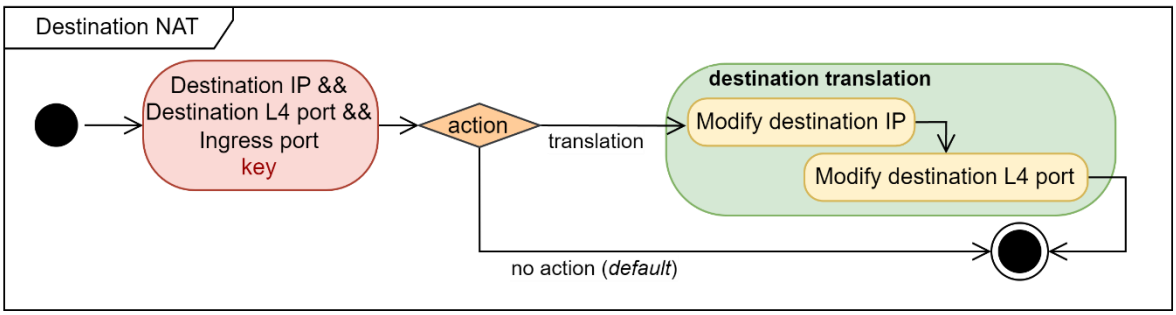


Figura 66. Diagrama de actividad de la tabla P4 llamada "Destination NAT"

- **Source NAT**: Tabla que presenta como clave el puerto de egreso en el dispositivo, la dirección IPv4 de origen y el puerto de transporte de origen. Se encarga de modificar la dirección y el puerto de la clave, tomando los nuevos valores especificados en la acción *source translation*.

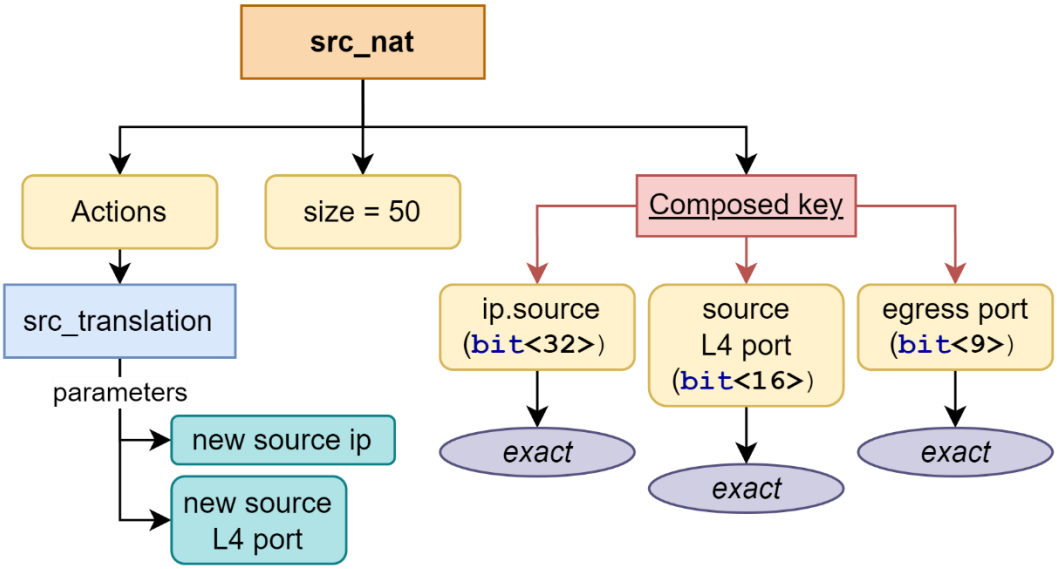


Figura 67. Estructura de la tabla P4 llamada "Source NAT"

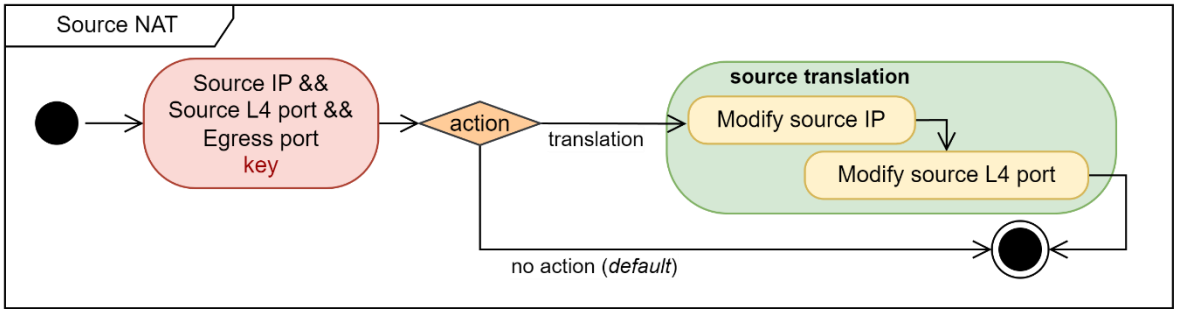


Figura 68. Diagrama de actividad de la tabla P4 llamada "Source NAT"

JUSTIFICACIÓN DEL ORDEN DE TRADUCCIÓN

- La tabla **Destination NAT** debe aplicarse antes que la tabla de encaminamiento, para garantizar una emisión por el puerto correcto.
- La tabla **Source NAT** debe aplicarse después de la tabla de encaminamiento, pues requiere conocer el puerto de egreso.

DIAGRAMAS DE ACTIVIDAD

En este apartado se exponen los diagramas de actividad resultantes de los requisitos definidos, que además incluyen las tablas de selección previamente definidas.

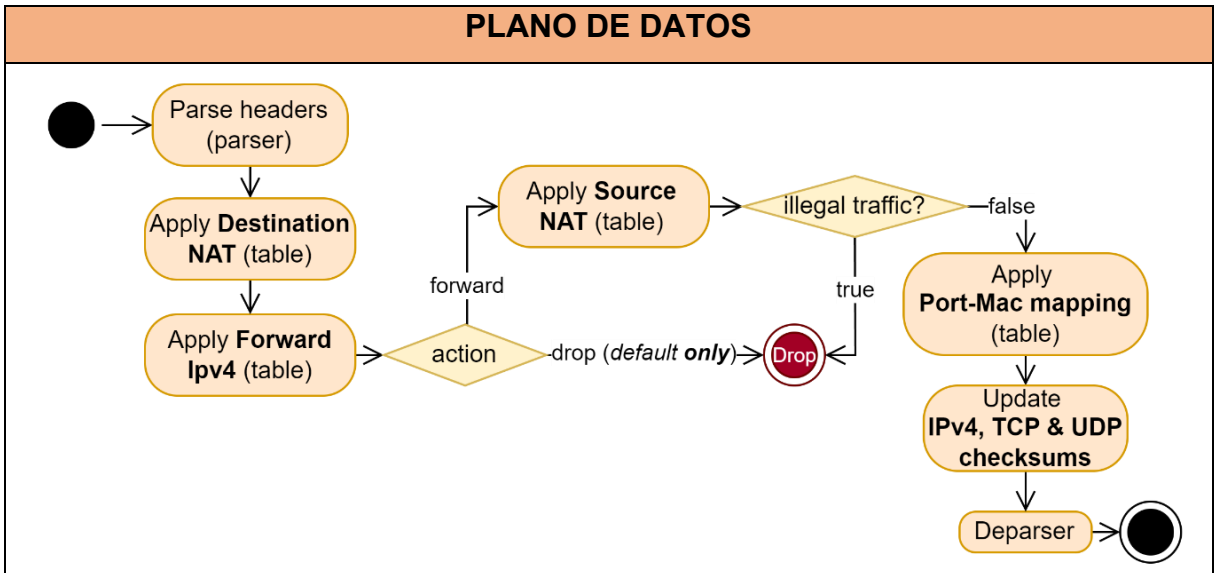


Figura 69. Diagrama de actividad del router NAT

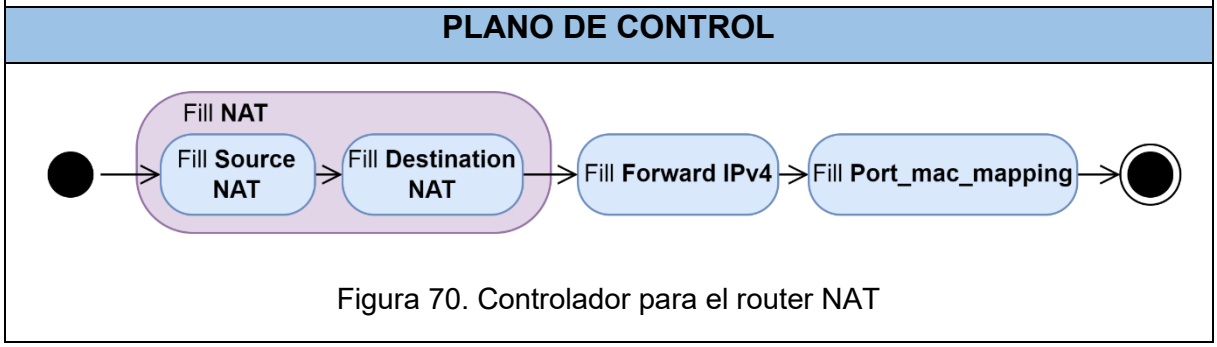


Figura 70. Controlador para el router NAT

DIAGRAMA DE PIPELINE

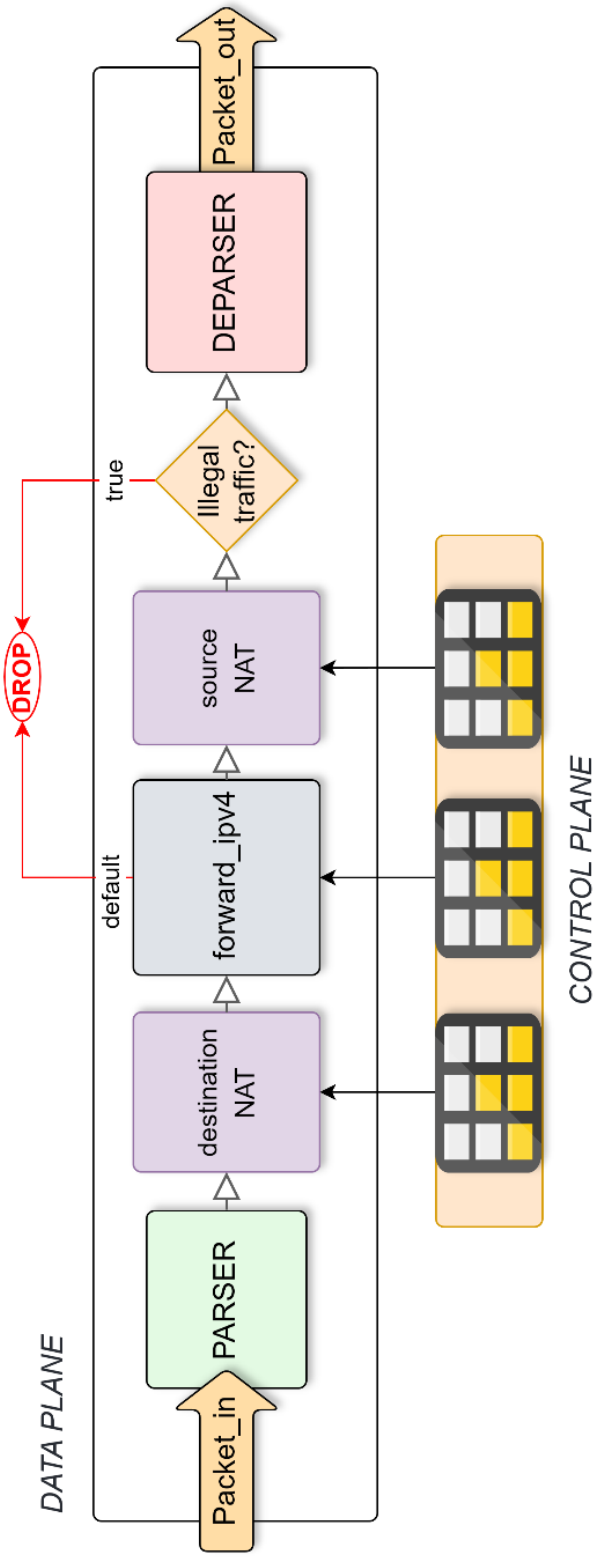


Figura 71. Pipeline lógico del router NAT

5.4. Router con monitorización de tráfico

DESCRIPCIÓN

En este servicio extiende el router básico, y añade una sesión de clonación para enviar copias de todos los paquetes de todos los paquetes que circulan por el hacia el controlador por el puerto CPU, que los recibirá para su análisis, y posteriormente insertar estadísticas en una base de datos. Esto es útil para observar en tiempo real las estadísticas del conmutador, y tener así una visión del tráfico que transita por él.

En la siguiente tabla se incluyen los protocolos que reconoce el servicio:

PROTOCOLOS RECONOCIDOS	
• Ethernet (nivel de enlace) [23]. • IPv4 (nivel de red) [24].	Layer 3 IP Layer 2 ETHERNET Layer 1 Physical

Tabla 11. Protocolos reconocidos por el router monitor

El parser deberá analizar únicamente cabeceras IPv4:

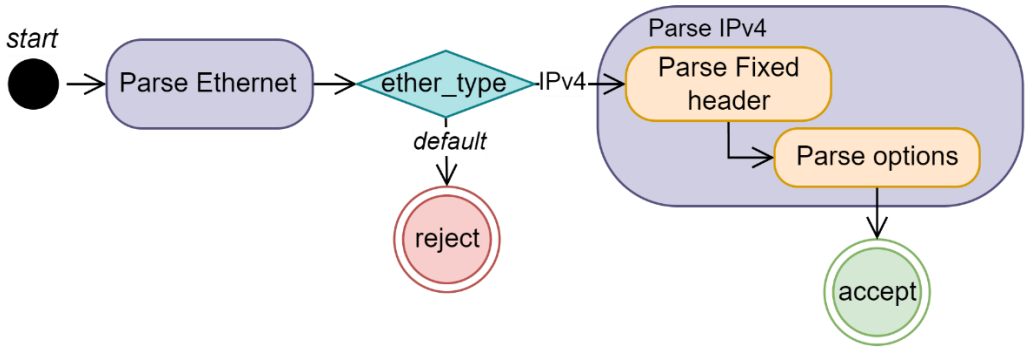


Figura 72. Máquina de estados del parser del router monitor

Para poder emitir correctamente estadísticas de tiempos de procesamiento del conmutador, se introduce una cabecera personalizada, denominada *Monitor*:

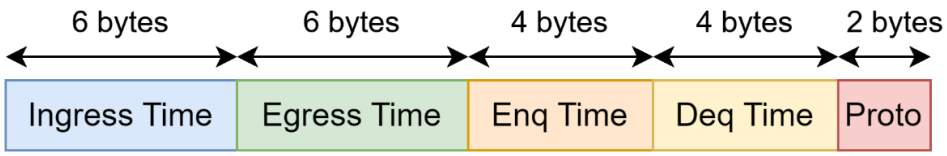


Figura 73. Cabecera personalizada "Monitor"

Los campos que incluye son extraídos de metadatos que ofrece la arquitectura programada en el ámbito del programa P4:

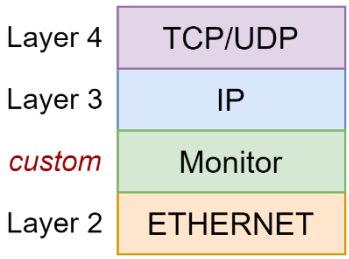


Figura 74. Torre de protocolos con la cabecera "Monitor"

- **Ingress Time**: Marca temporal registrada al iniciar la etapa de ingress.
- **Egress Time**: Marca temporal registrada al iniciar la etapa de egress.
- **Enq Time**: Marca temporal registrada al ingresar en la cola del Traffic Manager.
- **Deq Time**: Tiempo que pasa el paquete en la cola del Traffic Manager.

El campo *Proto* indica el identificador del protocolo empleado en la capa de red, que se toma del protocolo Ethernet, que a su vez tiene un identificador experimental de la cabecera Monitor. Se recomienda consultar el Apéndice C.

REQUISITOS

- **Actores**: Conmutador y controlador.
- **Requisitos ya definidos que son incluidos**:
  - **RFD2.2 RFD2.4 RFD2.5 RFD2.6 RFD2.7**

REQUISITOS FUNCIONALES
PLANO DE DATOS
• **RFD4.1** El conmutador ha de admitir y analizar cabeceras Ethernet e IPv4.
• **RFD4.2** El conmutador ha de clonar todos los paquetes que circulen por él.
• **RFD4.3** El conmutador ha de añadir estadísticas en una nueva cabecera.
• **RFD4.4** El conmutador ha de emitir las copias de los paquetes clonados por el puerto de CPU.
PLANO DE CONTROL
• **RFC4.1** El controlador deberá crear una sesión de clonación al puerto de CPU.
• **RFC4.2** El controlador deberá recibir mensajes con paquetes clonados.
• **RFC4.3** El controlador deberá procesar y almacenar estadísticas del paquete en una base de datos.

Tabla 12. Requisitos funcionales del router monitor

ESTRUCTURA DE TABLAS

Tablas ya definidas que son incluidas: Forward IPv4 y Firewall (del servicio 2).

En este servicio no se especifican nuevas tablas.

DIAGRAMAS DE ACTIVIDAD

PLANO DE DATOS

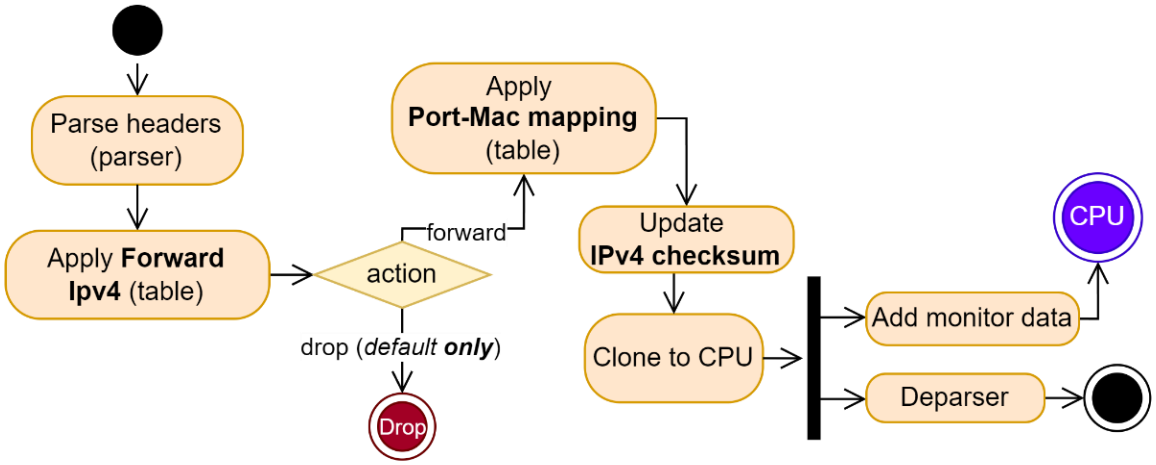


Figura 75. Diagrama de actividad del router monitor

PLANO DE CONTROL

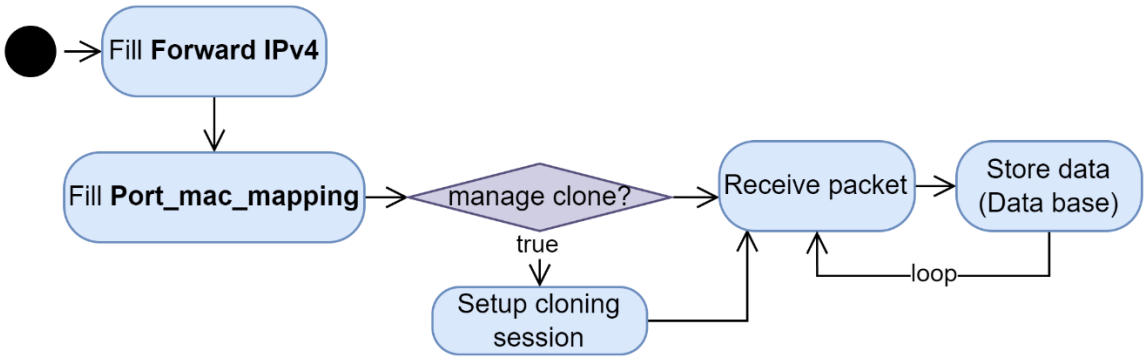


Figura 76. Controlador para el router monitor

DIAGRAMA DE PIPELINE

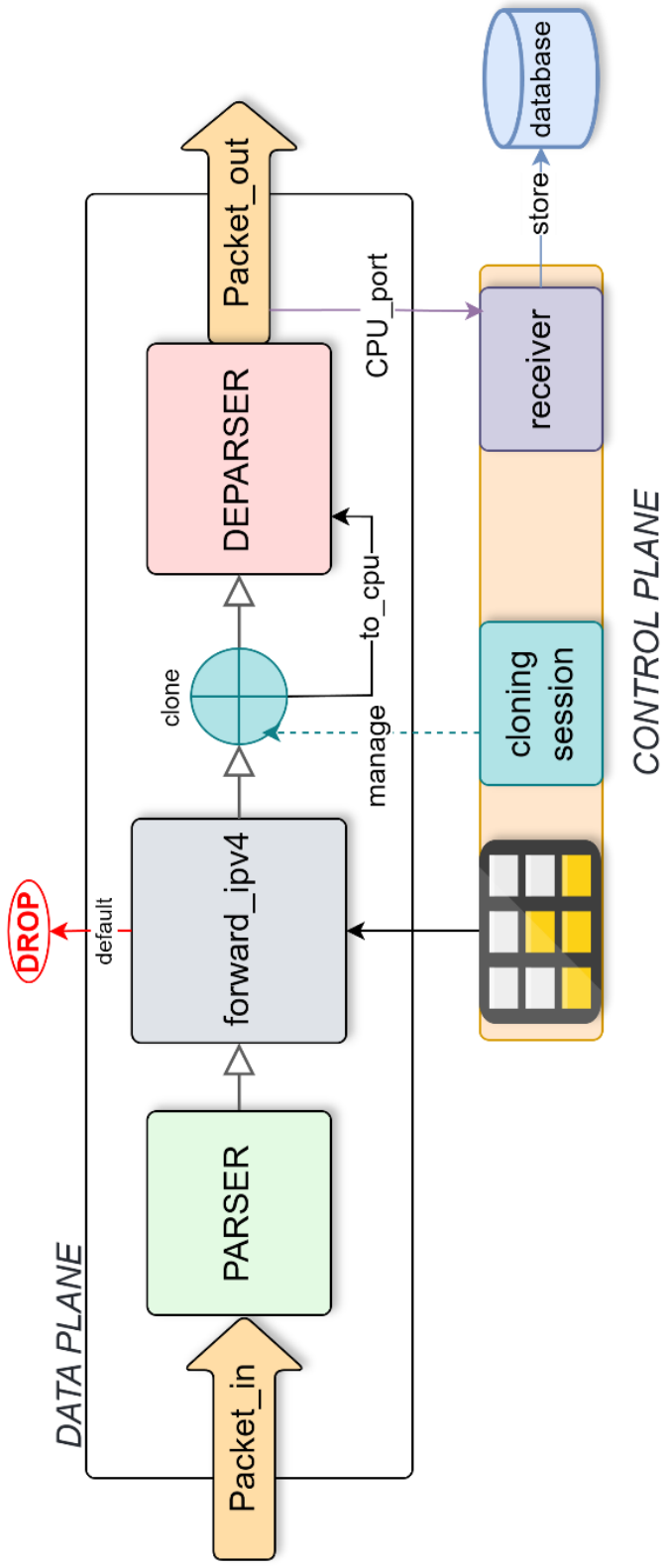


Figura 77. Pipeline lógico del router monitor

5.5. Router con políticas de calidad de servicio – QoS (Quality of Service)

DESCRIPCIÓN

En este servicio extiende el router básico, y añade un sistema de flujos para trabajar con un escenario que proporcione medidas para establecer una política de calidad de servicios. Para ello, se introduce una nueva cabecera, que permitirá identificar al flujo de comunicación al que pertenece un determinado paquete, y en una estructura del conmutador se establecerá el ancho de banda máximo que presenta cada uno de los flujos, de manera que si es excedido se tomarán diversas medidas que podrán derivar en un bloqueo temporal de todos los paquetes de ese flujo, hasta que pasado un tiempo vuelvan a poder emitir tráfico por el conmutador que los bloqueó. Esto permite implantar un sistema de priorización de tráfico, para garantizar disponibilidad de ancho de banda suficiente a comunicaciones críticas. Adicionalmente, admitirá también paquetes sin flujo, a los que también se les aplicará un ancho de banda máximo.

En la siguiente tabla se incluyen los protocolos que reconoce el servicio:

PROTOCOLOS RECONOCIDOS	
• Ethernet (nivel de enlace) [23]. • IPv4 (nivel de red) [24]. • Cabecera personalizada *Flow*.	Layer 3 — IP *custom* — Flow Layer 2 — ETHERNET

Tabla 13. Protocolos reconocidos por el router con calidad de servicio

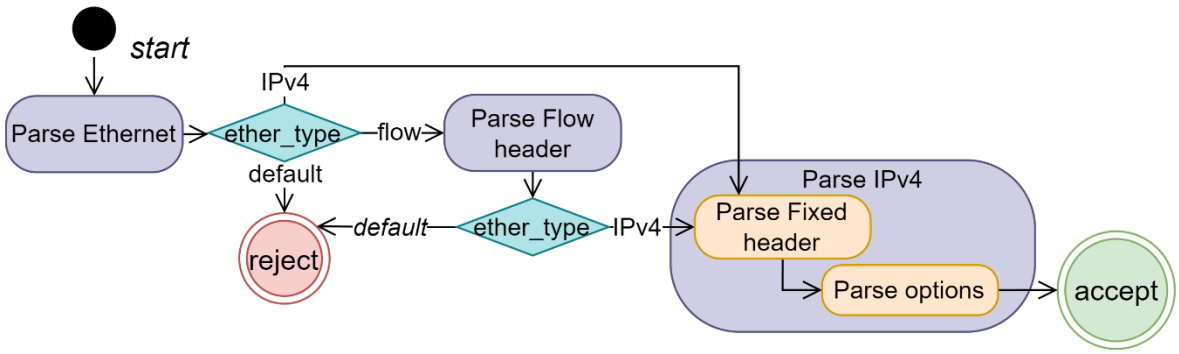


Figura 78. Máquina de estados del parser del router con calidad de servicio

Como se ha mencionado, se introduce una cabecera que identifica el flujo del paquete que la posea:

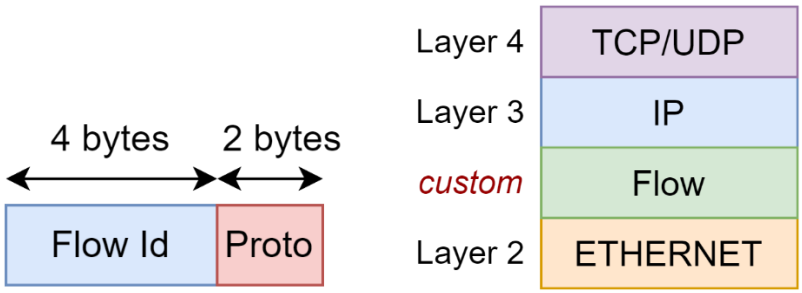


Figura 79. Cabecera personalizada "Flow", junto la torre de protocolos que lo incluye

Para poder aplicar la política de asignación de ancho de banda definida, se aplicará el extern “meter”, un objeto accesible desde un programa P4. Lo define cada arquitectura en base a un comportamiento común.

MEDIDORES DE TRES COLORES
Un **medidor de tres colores** es una entidad abstracta ofrecida por la arquitectura de un conmutador, cuya función es medir la tasa de tráfico que circula en un determinado instante, y a partir de ello evaluar la calidad de la medida. Emplea tres colores para marcar al paquete que invoca la medición: verde, amarillo y rojo, que indican un uso bueno, regular o malo respectivamente. En este caso se empleará el modelo de medidor que se basa en dos parámetros [33], el **CIR (commitment)**, que indica la cota superior de ancho de banda para obtener el color verde, y el **PIR (Peak)**, la cota superior para el rojo. Existen otros dos parámetros (CBURST y PBURST) que se corresponden con el tamaño máximo de ráfagas de paquetes, pero que no son tenido en cuenta en este caso. Estos parámetros son configurables desde el controlador, y los colores son obtenidos en el plano de datos (accesibles desde el programa P4), por lo que garantiza un esquema de trabajo en el que el programa del plano de datos puede tomar acciones en función del color resultante, pero con independencia de los valores asignados.

Se empleará un medidor que tome mediciones para cada flujo, y así poder controlar el ancho de banda máximo que tiene asignado cada uno de ellos, que lógicamente viene declarado por el parámetro **PIR,** pues es el umbral de tasa para obtener un resultado de color rojo. En este caso las mediciones de ancho de banda estarán expresadas en *Bytes por segundo*.

En el diagrama se puede observar con detalle el comportamiento dinámico de un medidor con las características descritas, y cómo se realiza una medición sobre un paquete, obteniendo el resultado de su flujo según el valor medido, y finalmente siendo marcado con uno de los colores disponibles.

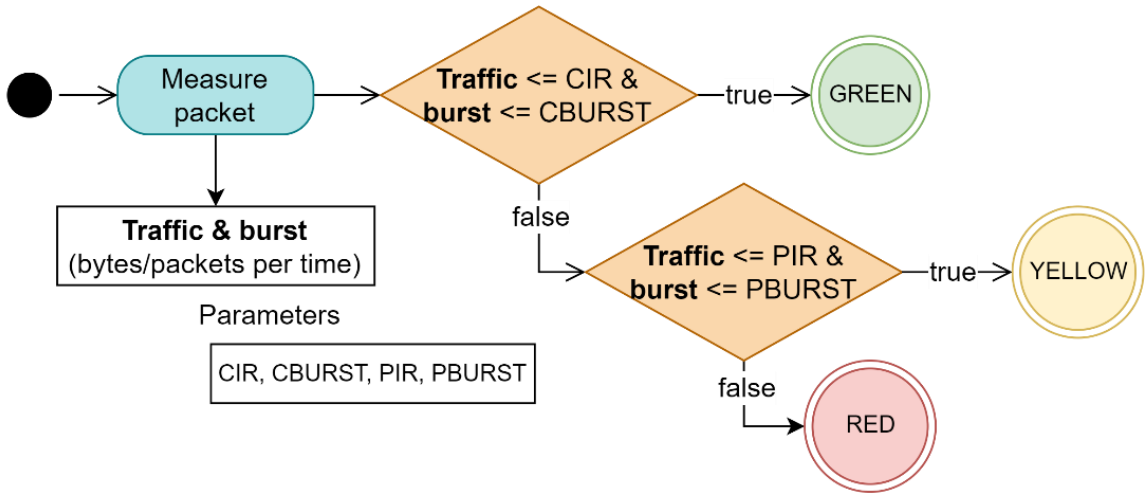


Figura 80. Diagrama de actividad de un medidor de tres colores

Tabla 14. Descripción de los medidores de tres colores

REQUISITOS

- **Actores**: Conmutador y controlador.
- **Requisitos ya definidos que son incluidos**:
  - **RFD2.2 RFD2.4 RFD2.5 RFD2.6 RFD2.7**

REQUISITOS FUNCIONALES
PLANO DE DATOS • **RFD5.1** El conmutador ha de admitir y analizar cabeceras Ethernet, Flow e IPv4. • **RFD5.2** El conmutador deberá medir el ancho de banda cuando procese un paquete con flujo especificado. • **RFD5.3** El conmutador ofrecerá un medidor específico para paquetes que no poseen un flujo asignado o aquellos cuyo flujo no se encuentra registrado. • **RFD5.4** El conmutador permitirá reenviar paquetes que han sido marcados con el color VERDE. • **RFD5.5** El conmutador permitirá reenviar paquetes que han sido marcados con el color AMARILLO, en el supuesto de que su flujo permita resultados de ese color. En ese caso será marcado con ECN, un flag que indica congestión de tráfico en la cabecera IPv4. • **RFD5.6** El conmutador bloqueará todo paquete que sea marcado con el color ROJO. **PLANO DE CONTROL** • **RFC5.1** El controlador ha de insertar valores de entradas en las tablas del plano de datos, entre ellas las tasas de los medidores presentes.

Tabla 15. Requisitos funcionales del router con calidad de servicio

ESTRUCTURA DE TABLAS

Tablas ya definidas que son incluidas: Forward IPv4 y Firewall (del servicio 2).

Para poder cumplir con los requisitos, se establece la siguiente tabla:

- **Flow table**: Tabla que presenta como clave el identificador de flujo. En caso de haber coincidencia, se aplica el medidor asociado a ese flujo, marcando así el paquete que se procesa.

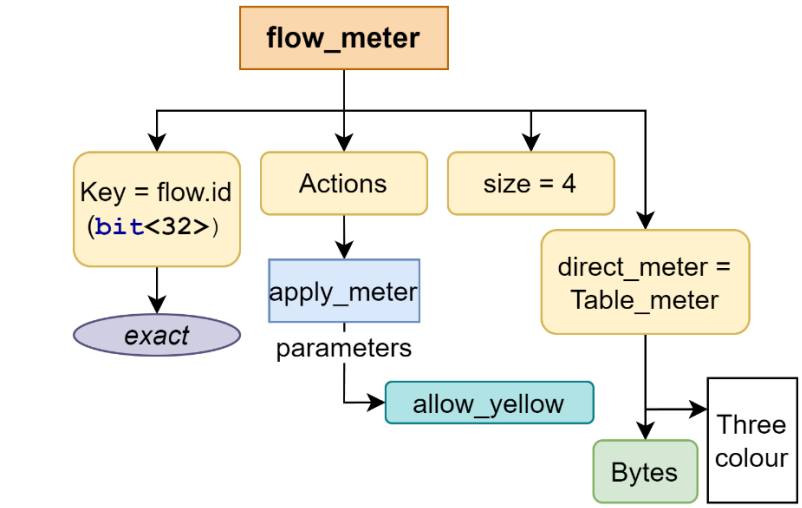


Figura 81. Estructura de la tabla P4 llamada "flow_meter"

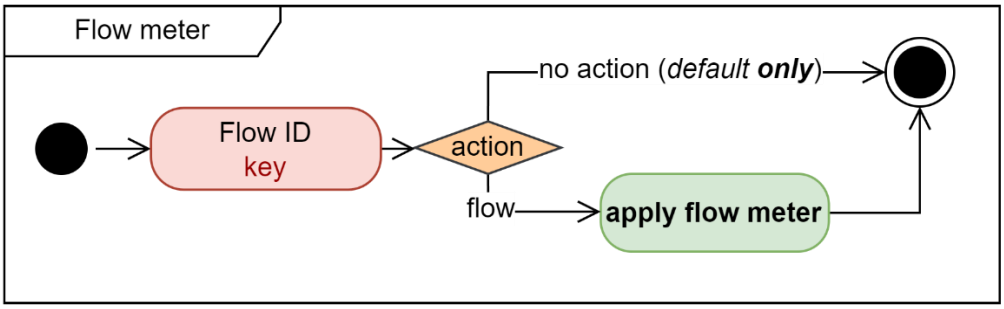


Figura 82. Diagrama de actividad de la tabla P4 llamada "flow_meter"

MEDIDOR AISLADO

En el plano de datos se declara un medidor que se utiliza para paquetes que no poseen la cabecera Flow, o que su identificador de flujo no se encuentra registrado en el dispositivo.

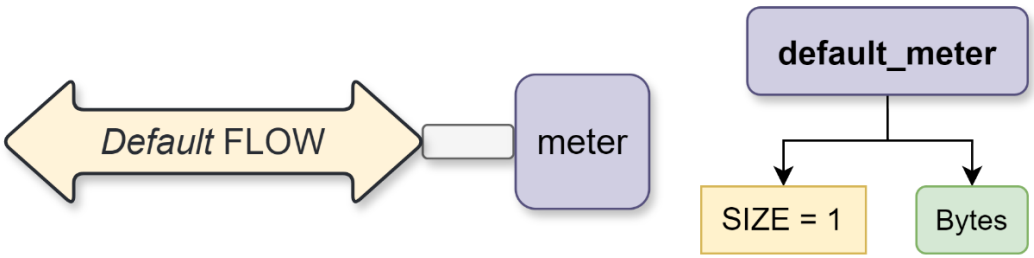


Figura 83. Medidor aplicado por defecto

DIAGRAMAS DE ACTIVIDAD

A continuación, se puede apreciar en los diagramas de actividad cómo se aplica de forma ordenada el comportamiento expuesto en los requisitos, con el empleo de medidores de flujo y uno por defecto.

PLANO DE DATOS

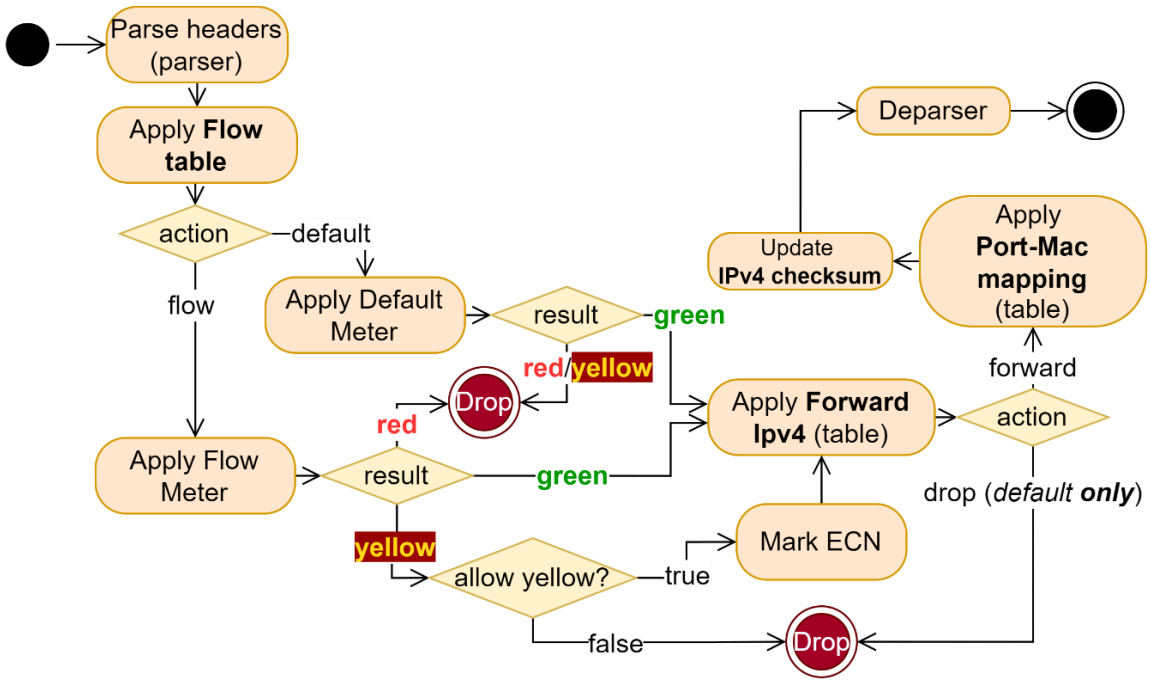


Figura 84. Diagrama de actividad del router con calidad de servicio

PLANO DE CONTROL

Figura 85. Controlador para el router con calidad de servicio

DIAGRAMA DE PIPELINE

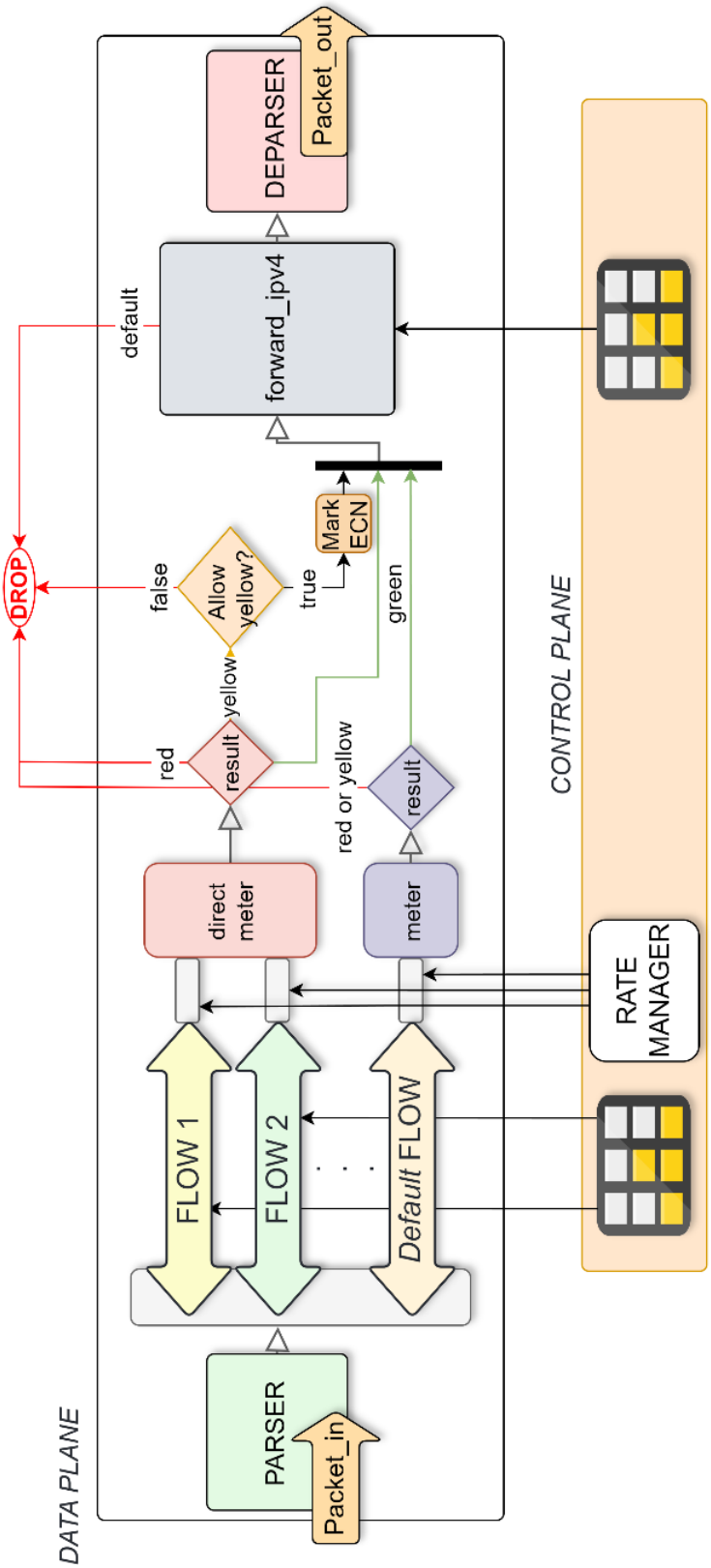


Figura 86. Pipeline lógico del router con calidad de servicio

6

DESARROLLO DE UN CONTROLADOR P4

Tras haber expuesto los diferentes modelos de los servicios de red que se desean implementar, han surgido operaciones que han de ser ejecutadas por un controlador remoto al conmutador, y con ello la necesidad de implementar un conector para controladores que sigan el protocolo **P4Runtime**. Este conector será utilizado por programas que implementan la lógica de control de un determinado servicio, y requieren de operaciones fiables que operen sobre una conexión GRPC.

Uno de los principales objetivos que se desean cubrir es la interoperabilidad entre dispositivos con diferentes arquitecturas. Para ello con el empleo de P4Runtime se puede controlar dispositivos que adopten este protocolo de control, pero también se llevará a cabo un diseño que permita integrar otros formatos de API.

Resumidamente, implementaremos el componente más importante del controlador, que podrá ser reutilizado en diversos escenarios particulares.

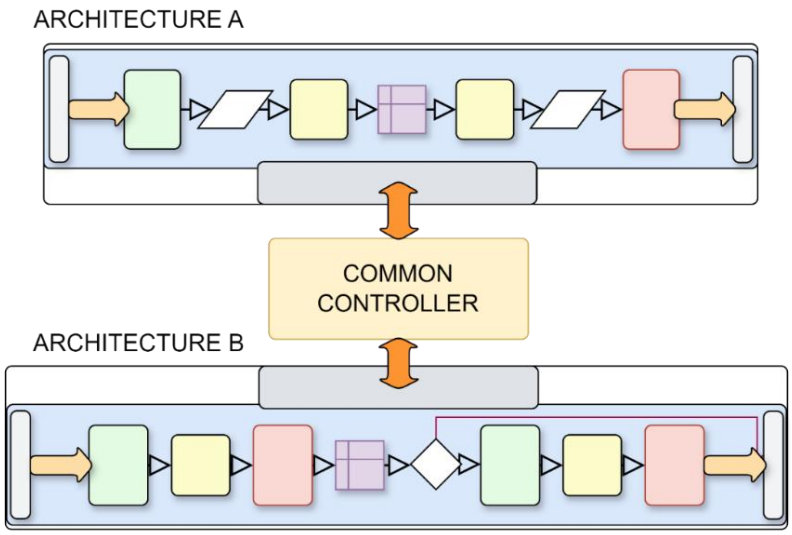


Figura 87. Interoperabilidad del controlador

6.1. Definición de requisitos de control

En primer lugar, estableceremos los requisitos mínimos de control, que vienen derivados de los requisitos funcionales que han de cumplir los controladores en la definición de servicios realizada en el capítulo anterior.

REQUISITOS DE CONTROL
• **RC1** El conector deberá poder establecer, mantener y cerrar una conexión con un conmutador P4 de forma confiable. • **RC2** El conector deberá ofrecer un mecanismo para instalar o modificar el programa P4 que ejecuta el conmutador controlado. • **RC3** El conector permitirá la inserción de una determinada entrada en una tabla P4 del plano de datos. • **RC4** El conector permitirá modificar la acción por defecto de una tabla. • **RC5** El conector permitirá consultar y obtener las entradas de una tabla P4. • **RC6** El conector permitirá establecer las tasas (rates) de un medidor P4 (meter). • **RC7** El conector permitirá establecer un grupo de difusión (multicast), con un identificador y un listado de puertos del conmutador. • **RC8** El conector permitirá gestionar una sesión de clonación en el plano de datos, con un id y un nuevo puerto de egreso del conmutador. • **RC9** El conector ofrecerá mecanismos para recibir mensajes del plano de datos. Concretamente de tipo Digest, paquetes de CPU (packet-in) y notificaciones de caducidad de entradas obsoletas (IdleTimeout notification).

Tabla 16. Requisitos de control

6.2. Modelado UML de un conector de control

Una vez definidos los requisitos que ha de cumplir la implementación del conector, se ha ideado un diseño orientado a objetos que ha sido modelado con el Lenguaje Unificado de Modelado (UML) [32]. Para garantizar la reutilización de controladores usuarios del conector, se ha ideado una solución que emplea una interfaz común con las operaciones ofrecidas, y que puede ser extendida por cualquier conector heredero.

Este diseño se basa en el patrón de diseño de software *Estrategia* [33], una solución probada que establece una jerarquía de clases para diferentes estrategias que poseen una interfaz pública común. En el contexto del conector las estrategias son los formatos de conexión empleados. En el siguiente diagrama se puede observar la jerarquía de conectores ideada, teniendo en cuenta que el único que ha sido implementado es *P4rConnector*, debido a que el otro ha sido introducido como soporte visual.

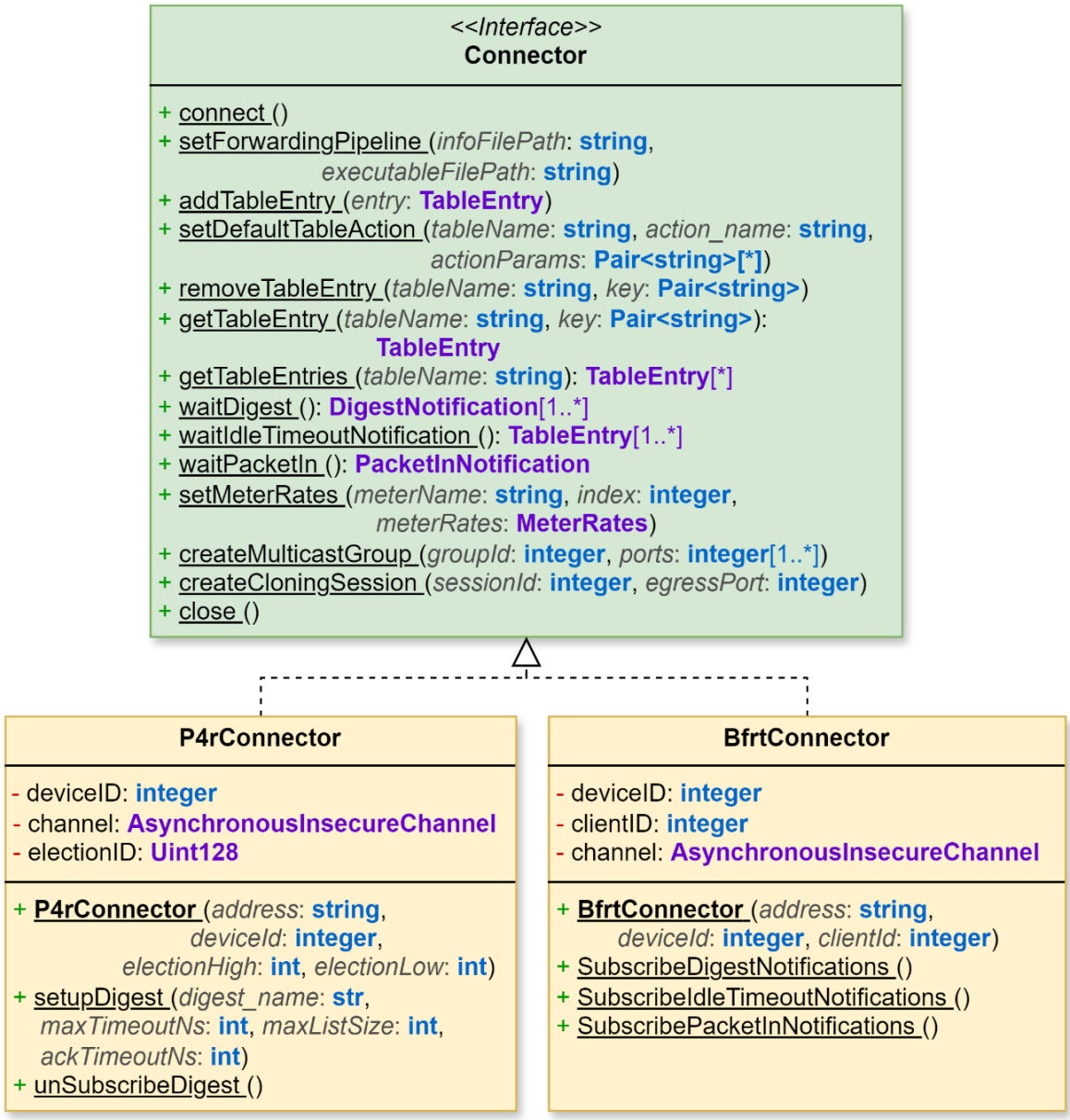


Figura 88. Jerarquía de implementación de conectores

Para garantizar el total aislamiento del conector con respecto a las interfaces de la conexión, se han definido tipos de objetos que modelan diferentes entidades P4.

Teniendo en cuenta todo lo anterior, podemos visualizar que una clase *Controller*, puede presentar una instancia de un conector, y que, mediante el Patrón Estrategia, presenta un total agnosticismo con respecto al formato de conexión empleado. Se muestra el paquete completo ideado para ser implementado:

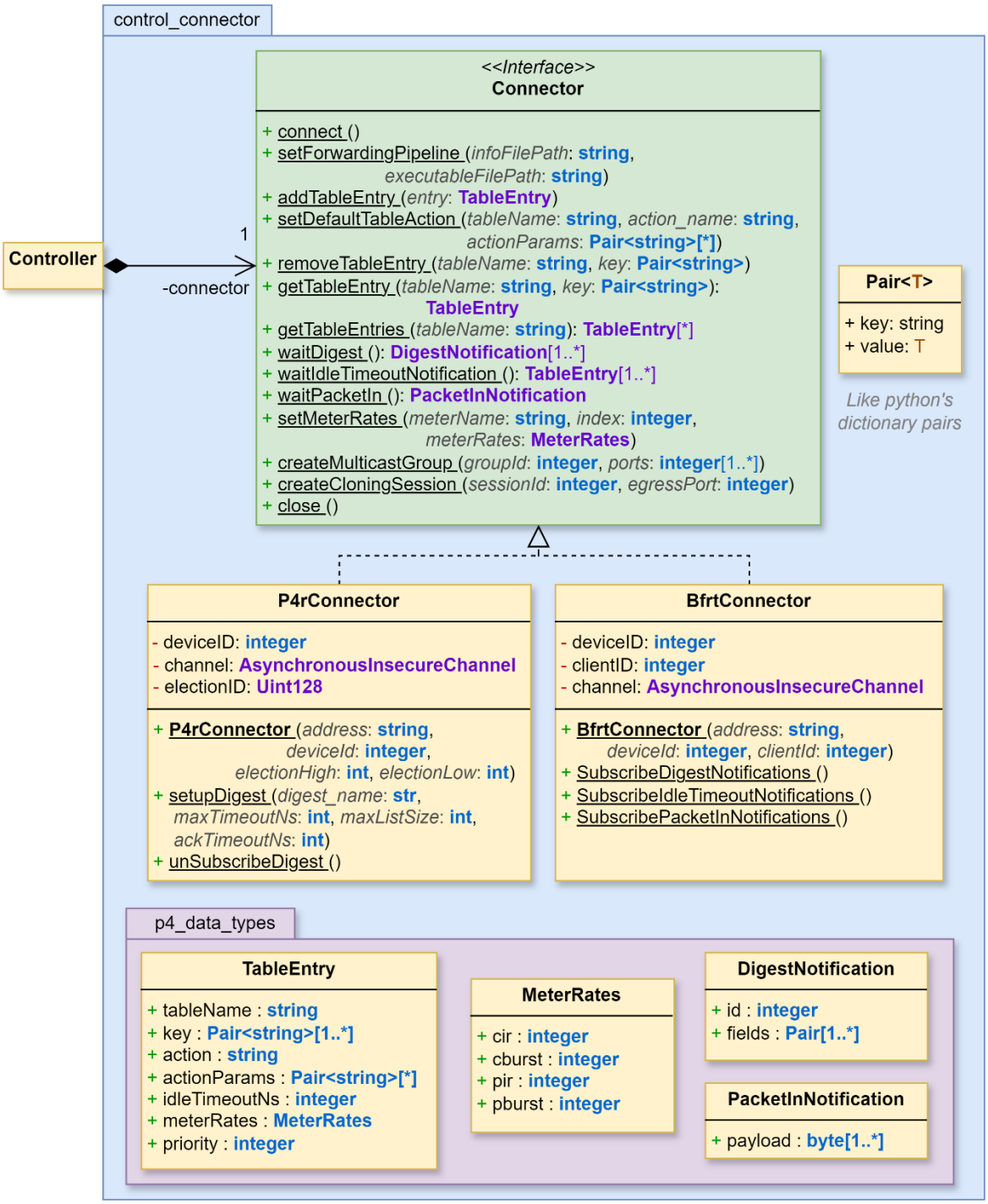


Figura 89. Diagrama UML completo del conector de control

6.3. Implementación del conector modelado

Una vez tenemos un diseño fiable y organizado, estamos en disposición de implementar el conector, y por lo tanto definir un comportamiento dinámico. Para implementar el conector se empleará el lenguaje de programación *Python* [25]. Cuando trabajamos con un servidor que posee implementado el protocolo de control P4Runtime, hay que entender correctamente su modo de operación para así poder entablar una comunicación correcta con él.

Para poder utilizar de manera correcta GRPC con la definición de P4Runtime, se ha compilado y organizado el código en un directorio accesible por la clase que se implementa, esto es *P4rConnector*. De esa forma se puede invocar las diferentes operaciones de modificación desde el código Python.

```
async def get_table_entry(self, table_name: str, key: Dict[str,
Any]) -> Optional[TableEntry]:
        """ Get an existing table entry, given the key. """
        result: Optional[TableEntry] = None

        table_id = self.__p4_info_helper.get_table_id(table_name)
        table_entry = p4runtime_pb2.TableEntry(table_id= table_id)
        for (field_name, field_value_str) in key.items():
            table_entry.match.append(
                self.__serialize_match_field(
                    table_name= table_name,
                    field_name= field_name,
                    value_str= field_value_str
                )
            )
        entity = p4runtime_pb2.Entity()
        entity.table_entry.CopyFrom(table_entry)

        request = p4runtime_pb2.ReadRequest(device_id=
self.__device_id)
        request.entities.append(entity)

        response: Optional[p4runtime_pb2.ReadResponse] = await
self.__get_first_read_response(self.__stub, request)
```

Figura 90. Implementación en Python del método "getTableEntry"

Como ya sabemos el requisito **RC9** establece la necesidad de recibir tres tipos de mensajes y es por ello por lo que en el diseño se han definido tres métodos para obtener los mensajes *Digest*, *Packet-in* e *Idle-timeout.* En la definición de P4Runtime existe un solo método para recibir todos los mensajes que se emiten desde el plano de datos. La conexión GRPC mantiene una cola de espera de la que puede ser extraído el que llegó primero (estrategia FIFO. First In First Out).

Esto anterior supone un problema, sobre todo en controladores que reciben varios tipos de ellos. Para ello, el conector de P4Runtime mantiene de manera interna una hebra concurrente que permite recibir del flujo de conexión los distintos mensajes, y los clasifica en diferentes colas gestionadas por el mismo conector, de manera que cuando se invoque a un método de recepción especializado, se obtendrá el primer mensaje de la cola correspondiente, y en caso de no existir ninguno, la hebra que ejecute el método de recepción, se quedará en estado de bloqueo hasta que aparezca el primer mensaje.

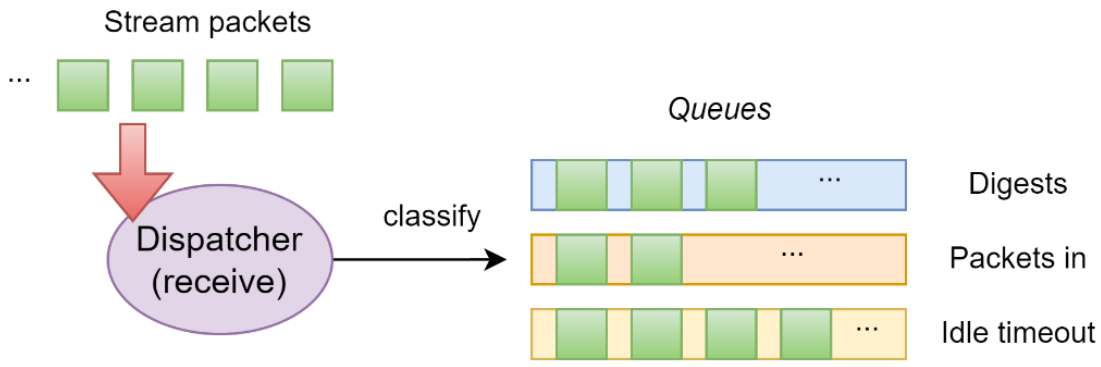


Figura 91. Modelo de clasificación de mensajes entrantes al conector

Todo esto nos introduce en un escenario típico de concurrencia, donde se necesitan tres hebras en ejecución concurrente, y que presentan funciones diferenciadas:

- **Hebra Despachadora (Dispatcher)**: Recibe los mensajes procedentes del servidor y los clasifica en su cola de espera correspondiente.
- **Hebra de Emisión (Send)**: Mantiene una cola para la emisión de confirmaciones de recepción hacia el servidor. Es necesario para evitar una repetición de mensajes por pérdida.

- **Hebra de Control**: Es la que ejecuta la lógica del controlador que es usuario del conector. Pueden existir varias en un determinado programa.

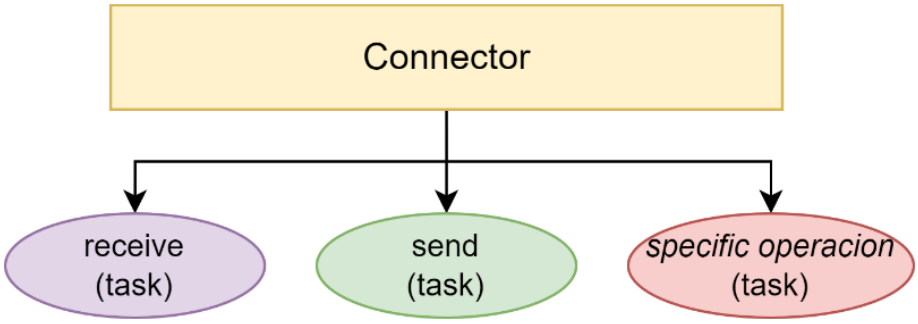


Figura 92. Tareas concurrentes del conector

Esta implementación no desempeña cómputos demasiado complejos, sino que posee muchas operaciones de entrada y salida. Por ello se ha llevado a cabo la implementación con un enfoque asíncrono, que permite ejecutar de forma eficiente las hebras cuando una de ellas queda bloqueada a la espera de un evento, como puede ser la recepción de un mensaje. Para permitir este comportamiento, se ha empleado la librería *asyncio* [34] de Python, que permite desarrollar este paradigma eficiente de concurrencia, e incluye un mecanismo que garantiza la exclusión mutua.

6.4. Desarrollo de controladores universales

Una vez hemos desarrollado el conector, componente base del controlador, se han definido dos controladores comunes y referenciables para desempeñar comportamientos avanzados, *LearningAgent* y *MonitorAgent*. En la mayoría de los servicios, las operaciones se reducen a la inserción de entradas en tablas, por lo que se emplean controladores específicos que emplean el conector para tal fin.

Siguiendo nuevamente el enfoque del patrón estrategia, Se han ideado dos controladores comunes que implementan de forma eficaz el gran número de comunicaciones que manejan:

- **Agente de aprendizaje (LearningAgent)**: Controlador para el servicio “conmutador con aprendizaje”, que mantiene dos hebras concurrentes que gestionan la inserción y el borrado de entradas en las tablas del plano de datos.
- **Agente de monitorización (MonitorAgent)**: Controlador para el servicio “router con monitorización de tráfico”, que gestiona la recepción de mensajes

Packet-In, y el posterior almacenamiento de estadísticas en una base de datos InfluxDB [20].

Cada uno de los controladores declarados interactúan con el conector mediante la interfaz *Connector*, para garantizar un desacoplamiento de la lógica de control con respecto a la interfaz de conexión empleada.

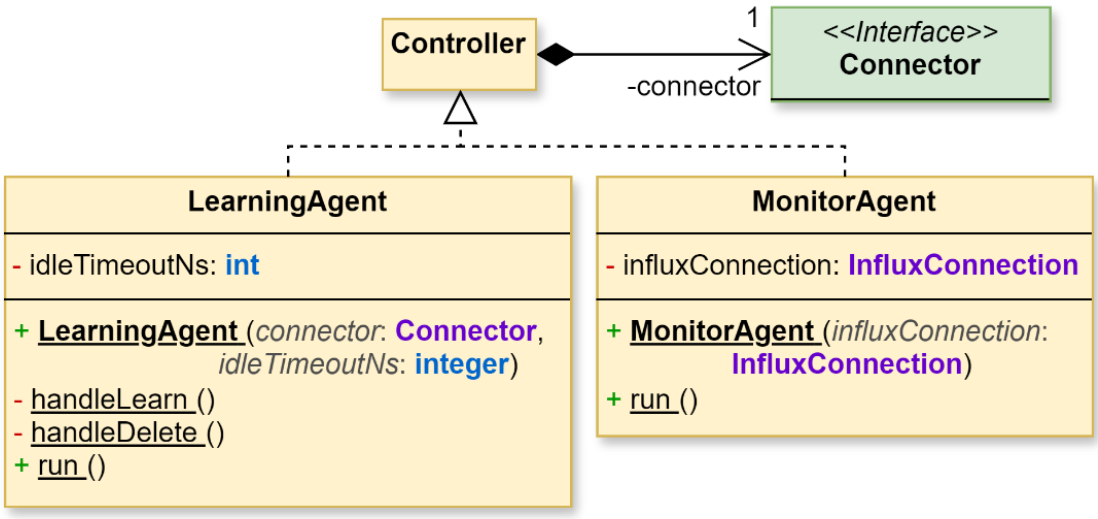


Figura 93. Jerarquía de controladores comunes

7

DESARROLLO Y VALIDACIÓN EN BMV2

Una vez definidos los servicios de red con los que se ha trabajado, y modelado y desarrollado un controlador que gestionar correctamente el plano de datos, se está en disposición de desarrollar los servicios en la arquitectura de referencia del lenguaje P4, que viene definida por el modelo de arquitectura *V1model* [21].

En este capítulo se abordará el proceso seguido para trasladar los servicios de red especificados al conmutador BMv2, definir escenarios de prueba de tráfico, y finalmente, validar los servicios con una perspectiva de verificación del comportamiento lógico mediante la emisión de tráfico de prueba.

7.1. La arquitectura V1model en BMv2

La arquitectura de referencia del lenguaje P4, ofrece una interfaz programable simple y con bloques que presentan un propósito bien especificado. Recuerde el lector que en la sección de arquitecturas programables se vieron los bloques destacables comunes a la mayoría de las arquitecturas lógicas.

Gracias a este modelo se pueden implementar soluciones de forma clara, algo importante para probar las funcionalidades que se desarrollan sin la presencia de alguna complejidad accidental inducida por el dispositivo de trabajo.

En el siguiente diagrama, se puede observar el pipeline de la arquitectura, donde se aprecian claramente dos componentes: *Verify checksum* y *Update checksum*, que permiten verificar y actualizar sumas de comprobación, respectivamente.

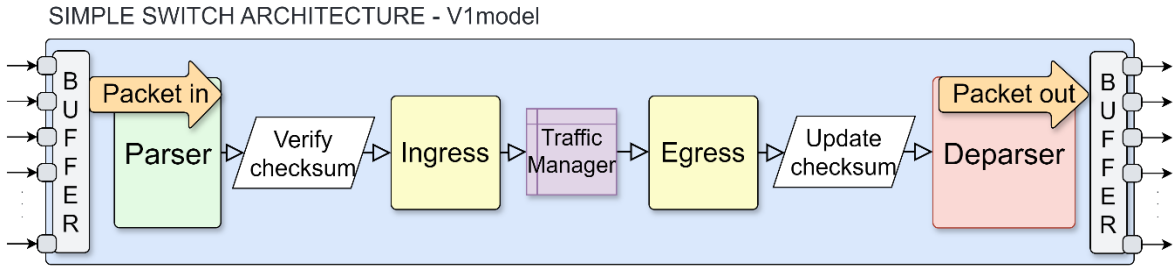


Figura 94. Pipeline de la arquitectura V1model

Por su parte, el conmutador BMv2 presenta naturaleza software, y por ello nos podemos referir a él con el término “conmutador virtual”. Cuando se considera el conmutador de referencia, se intenta expresar su importancia en la primera etapa de desarrollo con P4, pues permite desarrollar programas validables en cualquier equipo, sin grandes infraestructuras, lo cual hace muy accesible la programación de plano de datos, y es sin duda algo positivo para poder idear y compartir soluciones con la comunidad existente. Para su puesta en ejecución emplea recursos del sistema operativo, tales como interfaces virtuales, elementos de memoria, y lógicamente el procesador del equipo donde se hospeda.

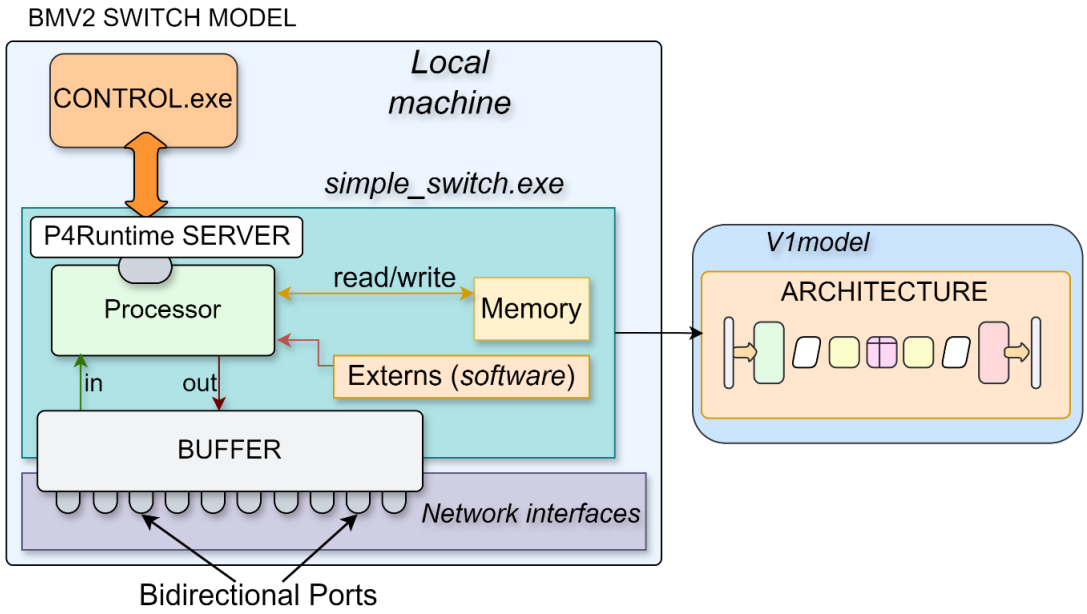


Figura 95. Estructura lógica de conmutador BMv2

Para poder compilar de forma adecuada los programas escritos en V1model, el dispositivo establece una extensión del compilador P4C, siguiendo el esquema de compiladores P4, de tal manera que el archivo que se genera tras la compilación no es un ejecutable en sí, sino que presenta extensión *.json*, y establece configuraciones que modelan el comportamiento del conmutador que se ejecuta, para que se comporte funcionalmente como se especifica en el programa implementado.

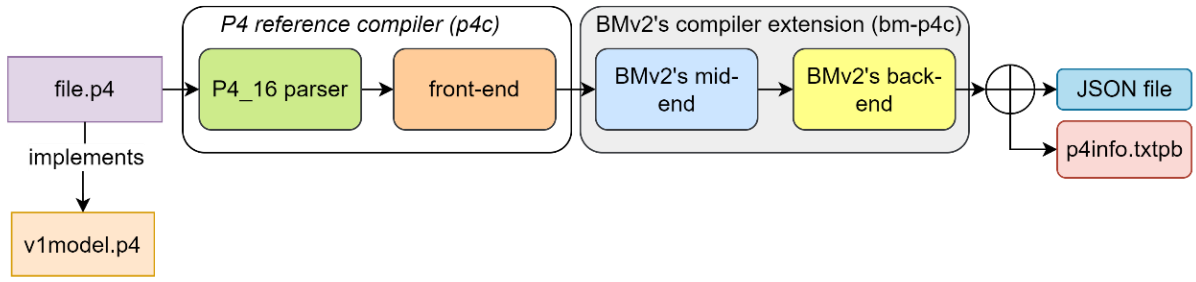


Figura 96. Modelo del compilador de BMv2

Todos los programas que se diseñan para esta arquitectura incluyen la librería *v1model.p4*, que define la interfaz de cada bloque programable, así como datos estructurados y parámetros específico.

7.2. Integración con el emulador Mininet

Hemos definido el comportamiento interno del conmutador virtual, pero para poder probar topologías de redes con diversas direcciones o incluso varias instancias del conmutador, se hace necesario el uso del emulador Mininet [23].

Para ello, se ha ideado un script implementado con el lenguaje *Python* que presenta la finalidad de leer un archivo JSON, que contiene la especificación de una determinada topología con hosts, conmutadores, enlaces y direcciones, y que será básico para crear una emulación de una red que contenga todo lo especificado. Además, se iniciará una instancia del programa BMv2 por cada conmutador incluido en la topología.

En la siguiente página, se muestra un diagrama de actividad que especifica el comportamiento dinámico del script mencionado, junto a un ejemplo de archivo JSON.

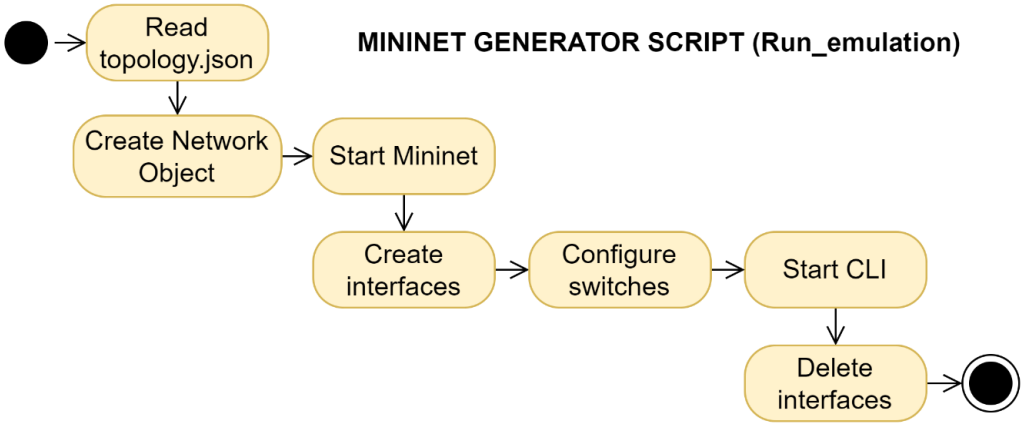


Figura 97. Diagrama de actividad del generador de Mininet

```
{ "hosts": {
        "h1": {
            "ip": "203.0.113.2/24",
            "mac": "08:00:00:00:00:01",
            "commands": [
                "route add default gw 203.0.113.1 dev eth0",
                "arp -i eth0 -s 203.0.113.1 08:00:00:00:01:00"
            ]
        },
        "h2": {
            "ip": "10.0.0.2/24",
            "mac": "08:00:00:00:00:02",
            "commands": [
                "route add default gw 10.0.0.1 dev eth0",
                "arp -i eth0 -s 10.0.0.1 08:00:00:00:01:00"
            ]
        },
        "h3": {
            "ip": "192.168.1.2/24",
            "mac": "08:00:00:00:00:03",
            "commands": [
                "route add default gw 192.168.1.1 dev eth0",
                "arp -i eth0 -s 192.168.1.1 08:00:00:00:01:00"
            ]
        }
    },
    "switches": {"R1": {}},
    "links": [["h1", "R1-p1"], ["h2", "R1-p2"], ["h3", "R1-p3"]]}
```

Figura 98. Ejemplo de topología especificada en JSON para Mininet

7.3. Desarrollo y validación de servicios de red

Una vez ha sido introducido el entorno de BMv2, y contextualizado su uso en el emulador Mininet, estamos en disposición de comentar el proceso de desarrollo de los servicios que han sido previamente ideados y modelados. Por cada servicio, se incluirá un diagrama de Pipeline adaptado a la arquitectura que se está abordando, para que pueda verse con claridad su estructura. Además. Serán expuestos diferentes escenarios de prueba, con topologías y configuraciones específicas, que permite ejecutar pruebas para validar el comportamiento en todos los casos posibles. En cada escenario de prueba se define una o varias topologías, y una representación gráfica de los elementos del pipeline configurados, que son configurados por el controlador que se emplea.

7.3.1. Conmutador con aprendizaje

Para abordar el desarrollo de este servicio, se ha empleado lógicamente la arquitectura V1model, como base del programa que se implementa. Entre los aspectos más destacados de este caso particular se puede comentar en primer lugar el uso del bloque programable *Egress*, que permite descartar adecuadamente un paquete clonado por difusión que pretende salir del dispositivo por el puerto por el que ingresó, en segundo lugar, el uso de un mensaje *digest* con los campos que son objeto de aprendizaje, y en tercer lugar la configuración del temporizador en la tabla *eth_timeout*.

MENSAJE DIGEST	TEMPORIZADOR DE CADUCIDAD
`digest<MacLearnDigest_t>(LEARN_DIGEST_ID, {headers.ethernet.src, std_meta.ingress_port});`	`table eth_timeout { key = { headers.ethernet.src : exact; } actions = { decideLearn; } size = TABLE_SIZE; support_timeout = true; }`

Tabla 17. Elementos destacados del desarrollo del conmutador con aprendizaje

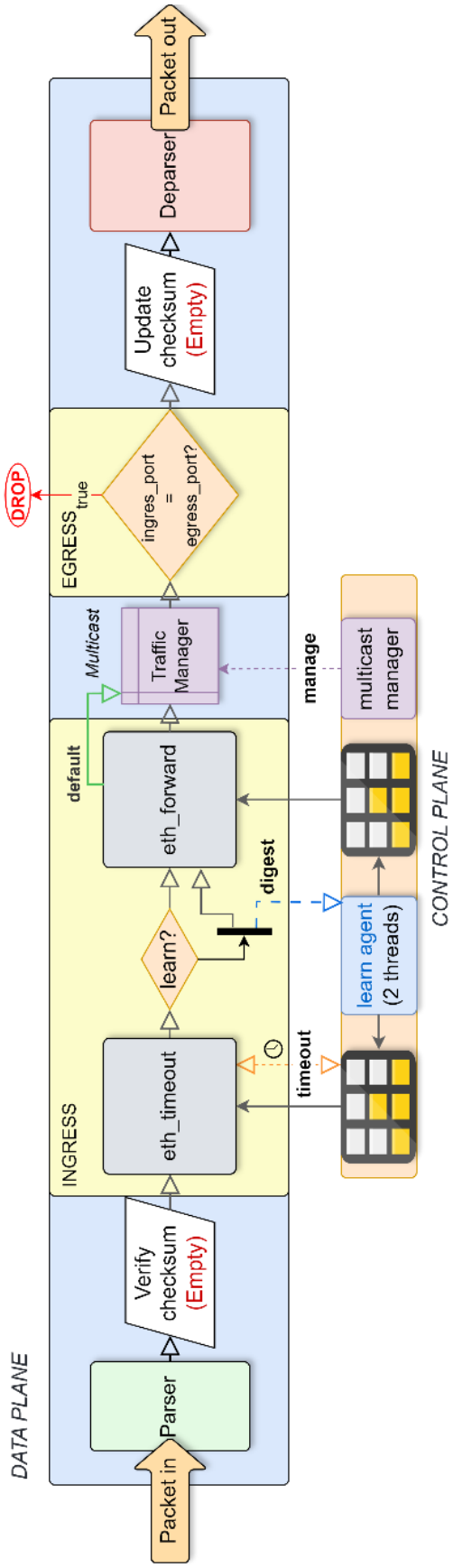


Figura 99. Pipeline del conmutador con aprendizaje desarrollado en BMv2

1. PRIMER ESCENARIO DE PRUEBA

En este primer escenario, se ha elegido una topología con cuatro equipos con direcciones privadas correspondientes a una misma subred, debido a que el conmutador con aprendizaje opera en la capa de enlace de la torre de protocolos.

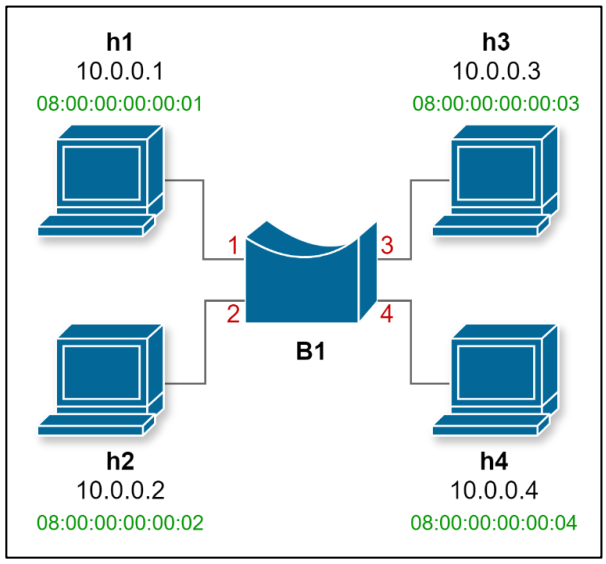


Figura 100. Primera topología de prueba del conmutador con aprendizaje

Los principales objetivos propuestos en esta primera emulación de prueba son:

- Validar el borrado de direcciones MAC obsoletas.
- Observar que todos los equipos se comunican adecuadamente.

Para garantizar una correcta ejecución del servicio, debemos en primer lugar establecer un grupo de difusión, con identificador **1** y al que pertenecen todos los puertos del conmutador conectados, que pueden ser apreciados en la imagen.

Switch: B1	
MULTICAST GROUPS	
ID	**PORTS**
1	1, 2, 3, 4

Figura 101. Grupo multicast del primer escenario de prueba

PRUEBA DE FUNCIONAMIENTO DEL TEMPORIZADOR

Tiempo de validez del aprendizaje no renovado: 30 segundos.

En esta prueba se emitirán paquetes que contengan únicamente la cabecera *Ethernet*, para mostrar fielmente el comportamiento del temporizador sin interferencias de paquetes ARP que se generan al emitir paquetes con la cabecera *IP*, y podrá observarse la emisión de difusión inicial al no conocer el puerto de salida que conduce al destino. Pasado un tiempo prudencial (ligeramente mayor al establecido), el temporizador se agotará, y volverá a repetirse nuevamente el proceso de aprendizaje.

En la siguiente página se muestra la captura Wireshark resultante de emitir tráfico desde el equipo *h1* al equipo *h2*.

Para poder reflejar el aprendizaje fielmente, se han emitido una sucesión de dos paquetes correspondientes a los dos sentidos de comunicación (h1-h2 y h2-h1).

No.	Time	Source	Destination	Interf	Protoco
1	0.000000000	08:00:00:00:00:01	08:00:00:00:00:02	2	LOOP
3	-0.0034280…	08:00:00:00:00:01	08:00:00:00:00:02	0	LOOP
5	-0.0006876…	08:00:00:00:00:01	08:00:00:00:00:02	1	LOOP
7	-0.0006706…	08:00:00:00:00:01	08:00:00:00:00:02	3	LOOP
9	2.089389209	08:00:00:00:00:01	08:00:00:00:00:02	1	LOOP
11	2.088666983	08:00:00:00:00:01	08:00:00:00:00:02	0	LOOP
13	35.1977640…	08:00:00:00:00:01	08:00:00:00:00:02	2	LOOP
15	35.1961389…	08:00:00:00:00:01	08:00:00:00:00:02	0	LOOP
17	35.1977918…	08:00:00:00:00:01	08:00:00:00:00:02	3	LOOP
19	35.1977786…	08:00:00:00:00:01	08:00:00:00:00:02	1	LOOP
21	37.2896524…	08:00:00:00:00:01	08:00:00:00:00:02	1	LOOP
22	37.2887346…	08:00:00:00:00:01	08:00:00:00:00:02	0	LOOP

Figura 102. Captura del tráfico de prueba del temporizador

Nótese que en la captura empieza la indexación de interfaces por 0, que se corresponde con el puerto 1 del conmutador. Esto aplica de aquí en adelante, en todos los casos de prueba.

En la imagen, se destacan cuatro zonas bien diferenciadas:

- **El primer envío de difusión**, que refleja que el conmutador no conoce el puerto donde se encuentra la dirección MAC 08:00:00:00:00:02.
- **El segundo envío unidireccional**, que refleja el aprendizaje al salir por el puerto 2.
- **El tercer envío de difusión**, que refleja la caducidad de la dirección MAC, realizando nuevamente el aprendizaje.
- **El cuarto envío unidireccional**, que refleja nuevamente el correcto envío por el puerto asociado a la dirección MAC aprendida.

PRUEBA DE INTERCONEXIÓN DE LA RED

En esta segunda prueba se emitirán paquetes ICMP que prueben la interconexión de la red, desde *h1* a *h2, h3* y *h4*. La elección de este protocolo viene motivada por la generación de respuestas tras enviar una solicitud de ping, por lo que permite observar la comunicación en los dos sentidos.

```
    5 0.001806643 10.0.0.2          10.0.0.1          2 ICMP    42 Echo (ping) reply    id=0x0000,
    6 0.043221721 10.0.0.1          10.0.0.3          0 ICMP    42 Echo (ping) request  id=0x0000,
    7 0.044437221 08:00:00:00:00:03 Broadcast         0 ARP     42 Who has 10.0.0.1? Tell 10.0.0.3
    8 0.044446476 08:00:00:00:00:01 08:00:00:00:00:03 0 ARP     42 10.0.0.1 is at 08:00:00:00:00:01
    9 0.045235459 10.0.0.3          10.0.0.1          0 ICMP    42 Echo (ping) reply    id=0x0000,
   10 0.099015802 10.0.0.1          10.0.0.4          0 ICMP    42 Echo (ping) request  id=0x0000,
   11 0.100621715 08:00:00:00:00:04 Broadcast         0 ARP     42 Who has 10.0.0.1? Tell 10.0.0.4
   12 0.100629859 08:00:00:00:00:01 08:00:00:00:00:04 0 ARP     42 10.0.0.1 is at 08:00:00:00:00:01
   13 0.101603420 10.0.0.4          10.0.0.1          0 ICMP    42 Echo (ping) reply    id=0x0000,
   14 0.043884241 10.0.0.1          10.0.0.3          2 ICMP    42 Echo (ping) request  id=0x0000,
   15 0.043901934 08:00:00:00:00:03 Broadcast         2 ARP     42 Who has 10.0.0.1? Tell 10.0.0.3
   16 0.044923422 08:00:00:00:00:01 08:00:00:00:00:03 2 ARP     42 10.0.0.1 is at 08:00:00:00:00:01
   17 0.044926501 10.0.0.3          10.0.0.1          2 ICMP    42 Echo (ping) reply    id=0x0000,
   18 0.100589451 10.0.0.1          10.0.0.4          2 ICMP    42 Echo (ping) request  id=0x0000,
   19 0.100599372 08:00:00:00:00:04 Broadcast         2 ARP     42 Who has 10.0.0.1? Tell 10.0.0.4
   20 0.101042341 08:00:00:00:00:01 08:00:00:00:00:04 2 ARP     42 10.0.0.1 is at 08:00:00:00:00:01
   21 0.101614929 10.0.0.4          10.0.0.1          2 ICMP    42 Echo (ping) reply    id=0x0000,
   22 0.001167276 10.0.0.1          10.0.0.2          3 ICMP    42 Echo (ping) request  id=0x0000,
   23 0.001861322 10.0.0.2          10.0.0.1          3 ICMP    42 Echo (ping) reply    id=0x0000,
   24 0.043716392 10.0.0.1          10.0.0.3          3 ICMP    42 Echo (ping) request  id=0x0000,
   25 0.044201786 08:00:00:00:00:03 Broadcast         3 ARP     42 Who has 10.0.0.1? Tell 10.0.0.3
   26 0.044768202 08:00:00:00:00:01 08:00:00:00:00:03 3 ARP     42 10.0.0.1 is at 08:00:00:00:00:01
   27 0.045218939 10.0.0.3          10.0.0.1          3 ICMP    42 Echo (ping) reply    id=0x0000,
   28 0.001044781 10.0.0.2          10.0.0.1          1 ICMP    42 Echo (ping) reply    id=0x0000,
   29 0.043704022 10.0.0.1          10.0.0.3          1 ICMP    42 Echo (ping) request  id=0x0000,
```

Figura 103. Captura de prueba de interconexión en la primera topología de prueba del conmutador con aprendizaje

2. SEGUNDO ESCENARIO DE PRUEBA

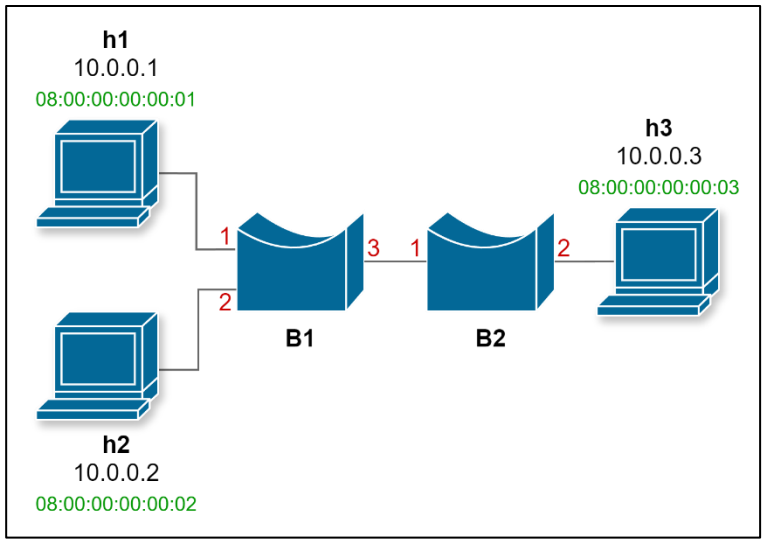


Figura 104. Segunda topología de prueba del conmutador con aprendizaje

En esta ocasión, una vez validado el correcto comportamiento del conmutador con aprendizaje de forma unitaria, se realizará una prueba de interconexión de la red, cuyo objetivo es validar la correcta integración de dos conmutadores que ejecuta el mismo programa.

Switch: B1	
MULTICAST GROUPS	
ID	**PORTS**
1	1, 2, 3

Switch: B2	
MULTICAST GROUPS	
ID	**PORTS**
1	1, 2

Figura 105. Grupos de difusión del segundo escenario de prueba

PRUEBA DE INTERCONEXIÓN DE LA RED

En esta prueba se emite tráfico ICMP desde el equipo *h1* a los equipos *h2* y *h3*. Existen dos casos bien diferenciados:

- **La emisión de paquetes entre *h1* y *h2***, que posibilita nuevamente comprobar el aprendizaje y posterior reenvío unidireccional.
- **La emisión de paquetes entre *h1* y *h3***, que involucra a dos conmutadores que tendrán que realizar el aprendizaje.

En esta última captura de prueba en el *conmutador con aprendizaje* se reflejan las evidencias finales del correcto funcionamiento de este servicio.

1	0.0000000…	10.0.0.1	10.0.0.2	1	ICMP	42 Echo (ping) request id=0x0000,
2	0.0031262…	08:00:00:00:00…	Broadcast	1	ARP	42 Who has 10.0.0.1? Tell 10.0.0.2
3	0.0031337…	08:00:00:00:00…	08:00:00:00:00:02	1	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
4	0.0050634…	10.0.0.2	10.0.0.1	1	ICMP	42 Echo (ping) reply id=0x0000,
5	0.0528299…	10.0.0.1	10.0.0.3	1	ICMP	42 Echo (ping) request id=0x0000,
6	0.0562416…	08:00:00:00:00…	Broadcast	1	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3
7	0.0017046…	10.0.0.1	10.0.0.2	0	ICMP	42 Echo (ping) request id=0x0000,
8	0.0033807…	08:00:00:00:00…	Broadcast	0	ARP	42 Who has 10.0.0.1? Tell 10.0.0.2
9	0.0039984…	08:00:00:00:00…	08:00:00:00:00:02	0	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
10	0.0050777…	10.0.0.2	10.0.0.1	0	ICMP	42 Echo (ping) reply id=0x0000,
11	0.0539870…	10.0.0.1	10.0.0.3	0	ICMP	42 Echo (ping) request id=0x0000,
12	0.0554100…	08:00:00:00:00…	Broadcast	0	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3
13	0.0567949…	08:00:00:00:00…	08:00:00:00:00:03	0	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
14	0.0580626…	10.0.0.3	10.0.0.1	0	ICMP	42 Echo (ping) reply id=0x0000,
15	0.0031748…	10.0.0.1	10.0.0.2	3	ICMP	42 Echo (ping) request id=0x0000,
16	0.0562487…	08:00:00:00:00…	08:00:00:00:00:03	1	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
17	0.0588694…	10.0.0.3	10.0.0.1	1	ICMP	42 Echo (ping) reply id=0x0000,
18	0.0047670…	08:00:00:00:00…	Broadcast	3	ARP	42 Who has 10.0.0.1? Tell 10.0.0.2
19	0.0047853…	08:00:00:00:00…	08:00:00:00:00:02	3	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
20	0.0055154…	10.0.0.2	10.0.0.1	3	ICMP	42 Echo (ping) reply id=0x0000,
21	0.0545529…	10.0.0.1	10.0.0.3	3	ICMP	42 Echo (ping) request id=0x0000,
22	0.0545681…	08:00:00:00:00…	Broadcast	3	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3
23	0.0572653…	08:00:00:00:00…	08:00:00:00:00:03	3	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
24	0.0572684…	10.0.0.3	10.0.0.1	3	ICMP	42 Echo (ping) reply id=0x0000,
25	0.0017027…	10.0.0.1	10.0.0.2	4	ICMP	42 Echo (ping) request id=0x0000,
26	0.0033792…	08:00:00:00:00…	Broadcast	4	ARP	42 Who has 10.0.0.1? Tell 10.0.0.2
27	0.0039974…	08:00:00:00:00…	08:00:00:00:00:02	4	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
28	0.0050771…	10.0.0.2	10.0.0.1	4	ICMP	42 Echo (ping) reply id=0x0000,
29	0.0539855…	10.0.0.1	10.0.0.3	4	ICMP	42 Echo (ping) request id=0x0000,
30	0.0554115…	08:00:00:00:00…	Broadcast	4	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3
31	0.0567934…	08:00:00:00:00…	08:00:00:00:00:03	4	ARP	42 10.0.0.1 is at 08:00:00:00:00:01
32	0.0580638…	10.0.0.3	10.0.0.1	4	ICMP	42 Echo (ping) reply id=0x0000,
33	0.0024129…	10.0.0.1	10.0.0.2	2	ICMP	42 Echo (ping) request id=0x0000,
34	0.0024291…	08:00:00:00:00…	Broadcast	2	ARP	42 Who has 10.0.0.1? Tell 10.0.0.2
35	0.0040666…	08:00:00:00:00…	08:00:00:00:00:02	2	ARP	42 10.0.0.1 is at 08:00:00:00:00:01

Figura 106. Captura de prueba de interconexión en la segunda topología de prueba del conmutador con aprendizaje

7.3.2. Router con firewall de filtrado

El proceso de desarrollo seguido en este servicio es simular al anterior, se toma de referencia la arquitectura, y se desarrolla el programa con el flujo algorítmico modelado.

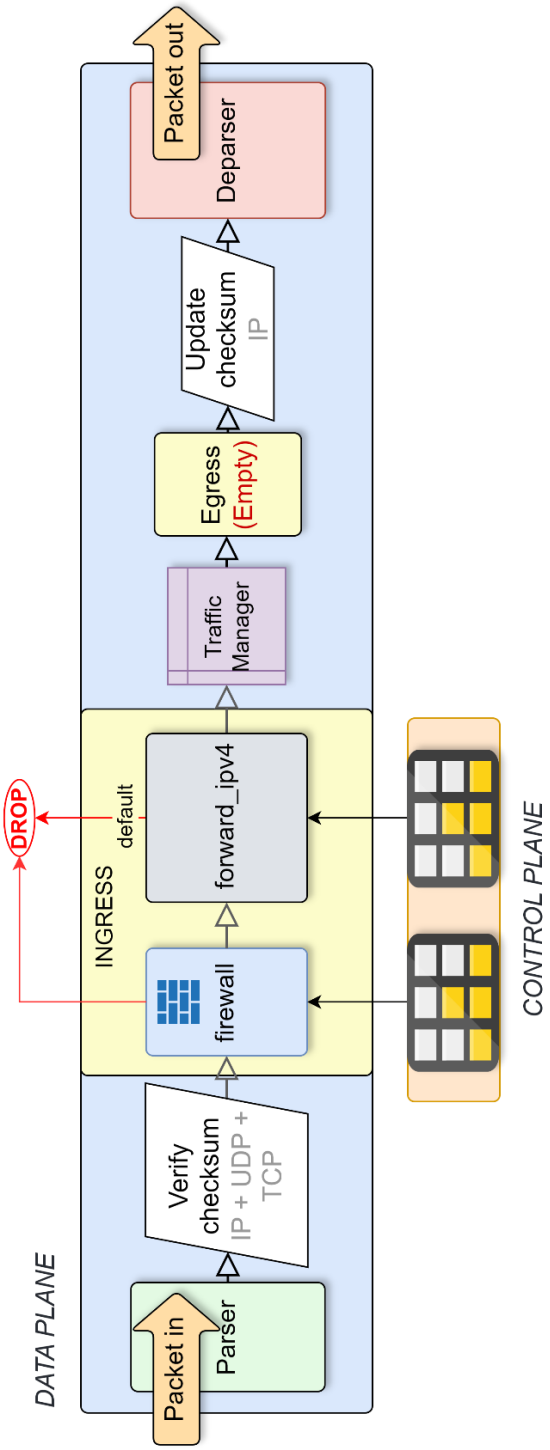


Figura 107.Pipeline del router con firewall desarrollado en BMv2

1. PRIMER ESCENARIO DE PRUEBA

En este escenario se ha ideado una topología en la que se conectan cuatro equipos al conmutador. Se probará el correcto encaminamiento del protocolo *IPv4*, el funcionamiento del firewall para una configuración preestablecida, y el correcto mapeo de direcciones MAC, definido en la tabla *Port-MAC*. Es importante tener en cuenta que para probar el tráfico entre todos los equipos es necesario mantener la tabla firewall sin entradas.

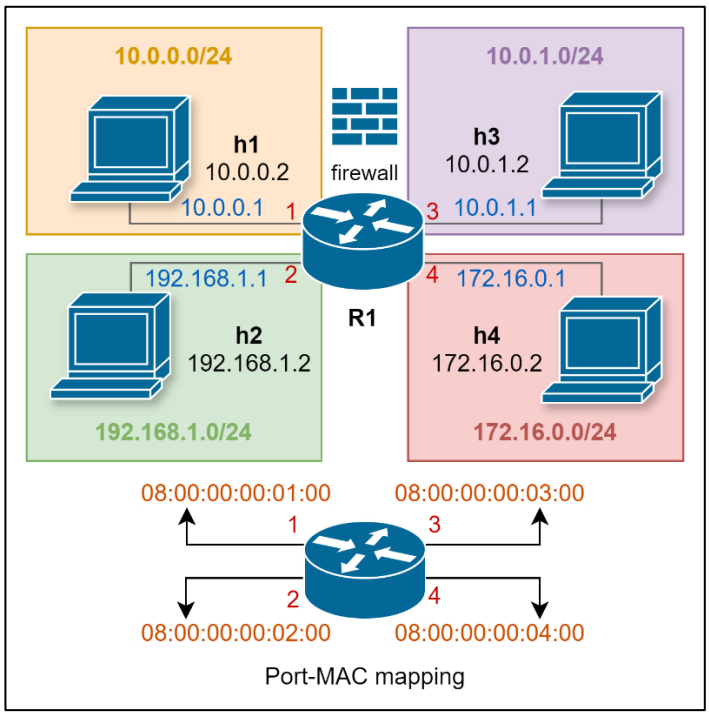


Figura 108. Primera topología de prueba del router con firewall en BMv2

Se establece en primer lugar, el valor de la tabla de encaminamiento presente en el conmutador:

Switch: R1		
forward_ipv4		
Key	*Action*	Parameters
10.0.0.2	forward	08:00:00:00:00:01 1
192.168.1.2	forward	08:00:00:00:00:02 2
10.0.1.2	forward	08:00:00:00:00:03 3
172.16.0.2	forward	08:00:00:00:00:04 4
missed	*drop*	

Figura 109. Estado de la tabla de encaminamiento en el primer escenario de prueba del router con firewall en BMv2

Se definen a continuación todos los casos de prueba.

PRUEBA DE CORRECTA INTERPRETACIÓN DEL CAMPO “OPTIONS”

Las cabeceras *IPv4*, pueden presentar al final un campo de longitud variable con diversas opciones. En ese sentido, dado en que este servicio se analizan cabeceras *TCP* y *UDP*, se hace necesario analizarlo correctamente, para que ello no repercuta en el posterior análisis (por parte del parser) de estas dos últimas cabeceras.

Para probarlo, se ha decidido emitir tráfico *UDP* entre dos hosts (Se han seleccionado *h1* y *h2*):

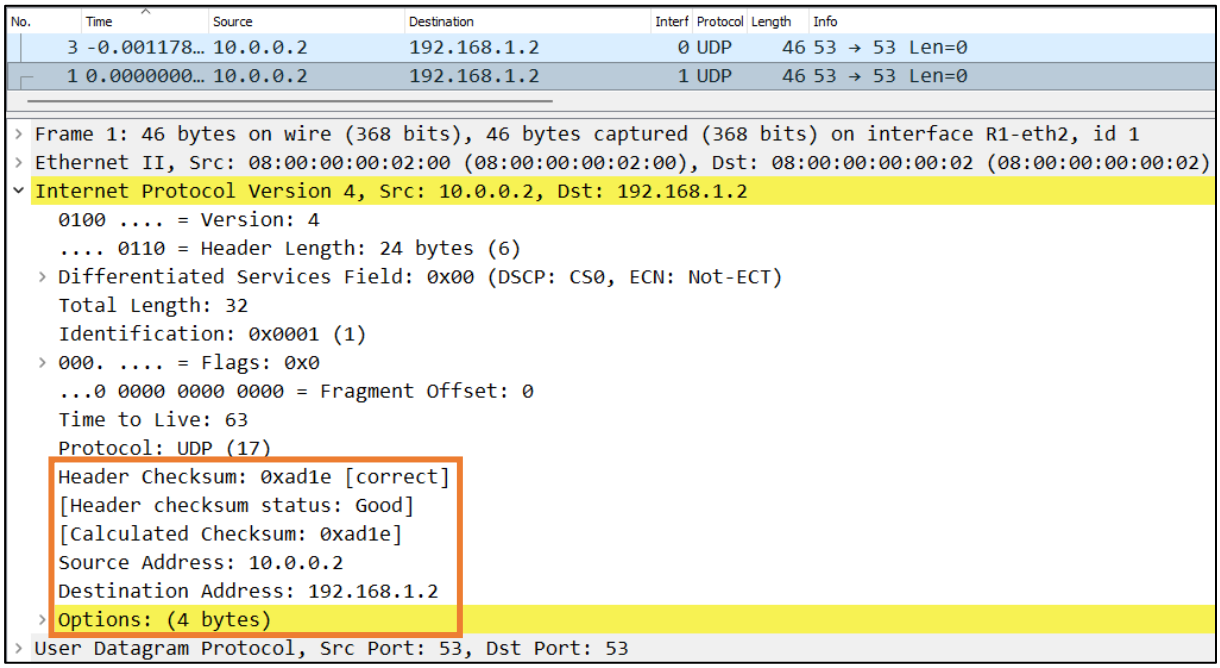


Figura 110. Captura de la prueba del campo "options" del protocolo IPv4 en BMv2

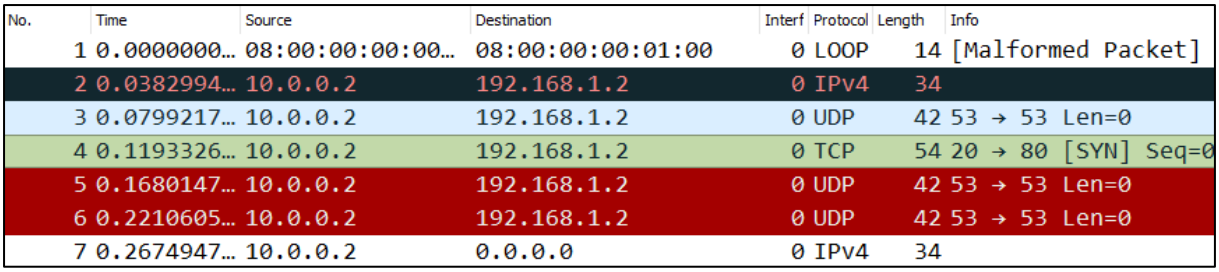
Como se puede apreciar, el paquete saliente del dispositivo presenta una correcta suma de comprobación (si este campo está presente, se toma en cuenta el valor en la suma), y el campo de opciones como ingresó, por lo que este servicio analiza adecuadamente este campo variable. El comportamiento se considera **validado.**

PRUEBA DE DESCARTE DE PAQUETES MAL FORMADOS

En la implementación del programa se ha incluido un mecanismo de detección de errores en tráfico, así como el descarte de paquetes que no posean la cabecera *IPv4*, para cumplir con las especificaciones de este último protocolo, y las funcionalidades establecidas en los requisitos funcionales del presente servicio.

Particularmente, se prueban los siguientes aspectos:

- **El rechazo de paquetes que no tengan el protocolo IPv4**: Se ha emitido un paquete que contiene únicamente la cabecera *Ethernet*.
- **El rechazo de paquetes con suma de verificación errónea**: Se han emitido dos, uno TCP y otro UDP.
- **El rechazo de paquetes con un valor TTL <= 1**: Se han emitido dos paquetes, uno con el valor puesto a 0 (no llegará al conmutador) y el otro con 1 (llegará al conmutador, pero será eliminado en la primera etapa). Esto elimina paquetes que han agotado los saltos de red disponibles.
- **El rechazo de paquetes cuya dirección IPv4 de destino no sea alcanzable**, tras no encontrar entrada coincidente en la tabla de encaminamiento (forward IPv4).

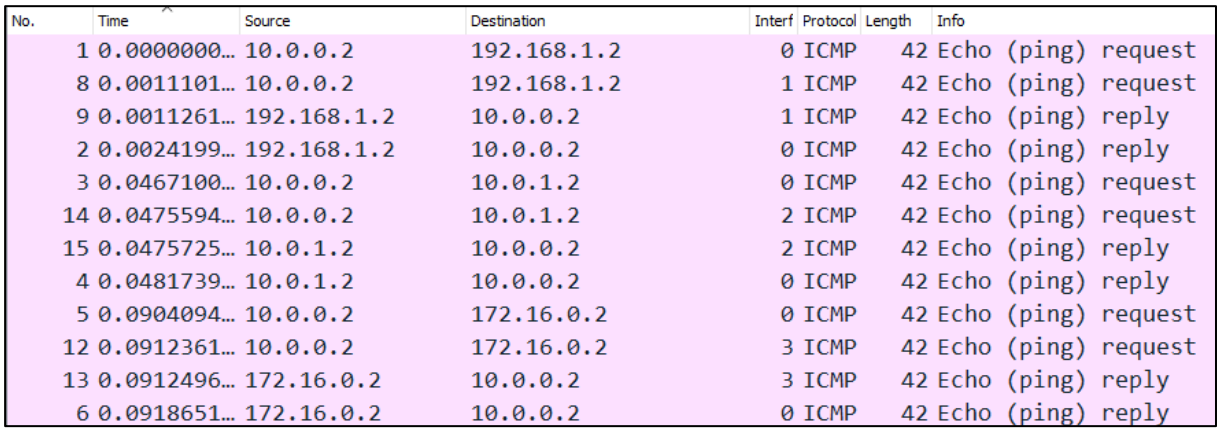


No.	Time	Source	Destination	Interf	Protocol	Length	Info
1	0.0000000…	08:00:00:00:00…	08:00:00:00:01:00	0	LOOP	14	[Malformed Packet]
2	0.0382994…	10.0.0.2	192.168.1.2	0	IPv4	34	
3	0.0799217…	10.0.0.2	192.168.1.2	0	UDP	42	53 → 53 Len=0
4	0.1193326…	10.0.0.2	192.168.1.2	0	TCP	54	20 → 80 [SYN] Seq=0
5	0.1680147…	10.0.0.2	192.168.1.2	0	UDP	42	53 → 53 Len=0
6	0.2210605…	10.0.0.2	192.168.1.2	0	UDP	42	53 → 53 Len=0
7	0.2674947…	10.0.0.2	0.0.0.0	0	IPv4	34	

Figura 111. Captura de la prueba de descarte de paquetes mal formados en BMv2

PRUEBA DE ENCAMINAMIENTO EN RED

En esta prueba se emiten paquetes ICMP desde *h1* hasta *h2, h3* y *h4*, para probar que se realiza correctamente el encaminamiento.



No.	Time	Source	Destination	Interf	Protocol	Length	Info
1	0.0000000…	10.0.0.2	192.168.1.2	0	ICMP	42	Echo (ping) request
8	0.0011101…	10.0.0.2	192.168.1.2	1	ICMP	42	Echo (ping) request
9	0.0011261…	192.168.1.2	10.0.0.2	1	ICMP	42	Echo (ping) reply
2	0.0024199…	192.168.1.2	10.0.0.2	0	ICMP	42	Echo (ping) reply
3	0.0467100…	10.0.0.2	10.0.1.2	0	ICMP	42	Echo (ping) request
14	0.0475594…	10.0.0.2	10.0.1.2	2	ICMP	42	Echo (ping) request
15	0.0475725…	10.0.1.2	10.0.0.2	2	ICMP	42	Echo (ping) reply
4	0.0481739…	10.0.1.2	10.0.0.2	0	ICMP	42	Echo (ping) reply
5	0.0904094…	10.0.0.2	172.16.0.2	0	ICMP	42	Echo (ping) request
12	0.0912361…	10.0.0.2	172.16.0.2	3	ICMP	42	Echo (ping) request
13	0.0912496…	172.16.0.2	10.0.0.2	3	ICMP	42	Echo (ping) reply
6	0.0918651…	172.16.0.2	10.0.0.2	0	ICMP	42	Echo (ping) reply

Figura 112. Captura de la prueba de encaminamiento de la primera topología del router con firewall de filtrado en BMv2

En primer lugar, valoraremos a partir de la captura tres aspectos fundamentales. Concretamente nos centraremos en el **primer paquete** que aparece, donde se puede observar su estado anterior y posterior a transitar por el conmutador:

- Que se decrementa el TTL: Se cumple.

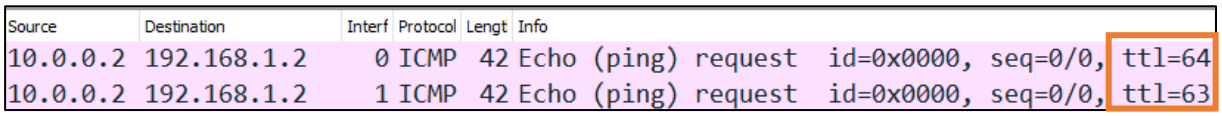


Figura 113. Evidencia de decremento del valor de TTL

- Que se establece una MAC origen especificada en Port-MAC al salir del conmutador: Se cumple.

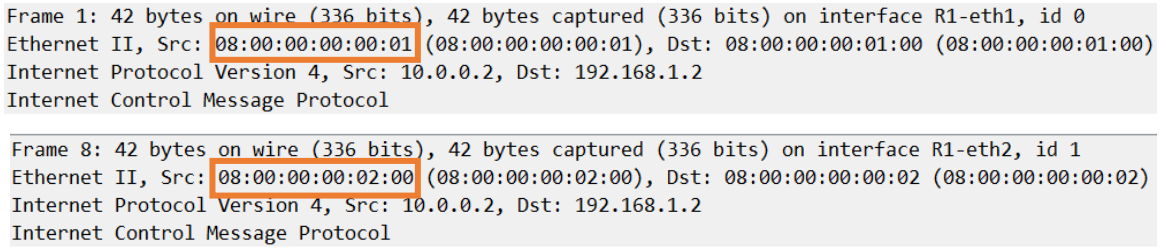


Figura 114. Evidencia del establecimiento de una nueva dirección MAC origen

- Que se actualiza la suma de comprobación, tras modificar campos de la cabecera IPv4 (TTL): Se cumple.

ESTADO EN EL INGRESO	ESTADO EN EL EGRESO
Header Checksum: 0xaf34 [correct] [Header checksum status: Good]	Header Checksum: 0xb034 [correct] [Header checksum status: Good]

Tabla 18. Variación del valor de la suma de comprobación

Además, tras observar el comportamiento general de la captura se puede observar que se cumplen con todos los requisitos del encaminamiento IPv4, por lo que este comportamiento quedaría debidamente **validado**.

PRUEBA DE FILTRADO DEL FIREWALL

En esta prueba partiremos con entradas insertadas la tabla *Firewall*, que han sido elegidas para cubrir todas las posibles combinaciones de filtros existentes (clave de la tabla). Para ello se incluye una representación del contenido de la tabla, junto a una sucesión de pruebas de tráfico que deben mostrar el comportamiento correspondiente tras aplicar la misma tabla (descarte o aceptación de paquetes).

Switch: R1			
firewall			
Key		*Action*	Priority
Origin	Destination		
10.0.0.1	10.0.1.2	drop	1
10.0.0.1	172.16.0.2 : 80	drop	3
192.168.1.2	10.0.0.0/24	drop	1
172.16.0.0/24	any : 50	drop	2
any : 50	172.16.0.0/24	noAction	1

Tabla 19. Estado de la tabla Firewall en la primera topología del router con firewall en BMv2

No.	Time	Source	Destination	Interf	Protocol	Lengt	Info
3	-0.042114…	10.0.0.1	10.0.1.2	0	ICMP	42	Echo (ping) request id=
4	-0.001886…	10.0.0.2	10.0.1.2	0	ICMP	42	Echo (ping) request id=
1	0.0000000…	10.0.0.2	10.0.1.2	2	ICMP	42	Echo (ping) request id=
2	0.0000748…	10.0.1.2	10.0.0.2	2	ICMP	42	Echo (ping) reply id=
5	0.0012652…	10.0.1.2	10.0.0.2	0	ICMP	42	Echo (ping) reply id=
9	0.0339881…	10.0.0.1	172.16.0.2	0	UDP	42	53 → 80 Len=0
10	0.0779271…	10.0.0.1	172.16.0.2	0	TCP	54	20 → 80 [SYN] Seq=0 Win=
11	0.1106090…	10.0.0.1	172.16.0.2	0	UDP	42	53 → 30 Len=0
12	0.1128928…	10.0.0.1	172.16.0.2	3	UDP	42	53 → 30 Len=0
13	0.1132759…	172.16.0.2	10.0.0.1	3	ICMP	70	Destination unreachable
6	0.1418356…	192.168.1.2	10.0.0.2	1	ICMP	42	Echo (ping) request id=
14	0.1738699…	172.16.0.2	10.0.0.2	3	UDP	42	53 → 50 Len=0
15	0.2140066…	172.16.0.2	192.168.1.2	3	UDP	42	53 → 50 Len=0
16	0.2548598…	172.16.0.2	192.168.1.2	3	UDP	42	53 → 40 Len=0
7	0.2571278…	172.16.0.2	192.168.1.2	1	UDP	42	53 → 40 Len=0
8	0.2573625…	192.168.1.2	172.16.0.2	1	ICMP	70	Destination unreachable
17	0.2582048…	192.168.1.2	172.16.0.2	3	ICMP	70	Destination unreachable
20	0.3108644…	10.0.0.2	172.16.0.2	0	UDP	42	50 → 53 Len=0
18	0.3120349…	10.0.0.2	172.16.0.2	3	UDP	42	50 → 53 Len=0
19	0.3121115…	172.16.0.2	10.0.0.2	3	ICMP	70	Destination unreachable
21	0.3131643…	172.16.0.2	10.0.0.2	0	ICMP	70	Destination unreachable
22	0.3580665…	192.168.1.2	172.16.0.2	0	UDP	42	53 → 53 Len=0
24	0.3589580…	192.168.1.2	172.16.0.2	3	UDP	42	53 → 53 Len=0
25	0.3590514…	172.16.0.2	192.168.1.2	3	ICMP	70	Destination unreachable
23	0.3603004…	172.16.0.2	192.168.1.2	1	ICMP	70	Destination unreachable

Figura 115. Captura de la prueba de filtrado de la primera topología del router con firewall de filtrado en BMv2

Para verificar que el firewall cumple con los objetivos marcados, se han analizado todas las emisiones de paquetes si existe una entrada que coincida con algunas de las direcciones y puertos que conforman la clave. El tráfico ha sido elegido de tal manera que se verifique el comportamiento definido en la tabla (descarte o aceptación de paquetes). Por ejemplo, si tomamos la primera entrada de la tabla *Firewall* y el primer paquete mostrado en la anterior captura, se puede visualizar que este es bloqueado porque así lo especifica la tabla con la acción *drop.* Tras un análisis exhaustivo, se ha comprobado que se comporta como se espera, por lo que se considera **validado**. Nótese que, al emplear paquetes con protocolos de la capa de transporte, se retorna un paquete ICMP que indica que el puerto de destino no es accesible. Este comportamiento es esperado dado que en el host experimental empleado no existe ningún servicio activo con el puerto de destino presente. Es más, en este caso constituye una evidencia de que el paquete ha llegado a su destino, que es el principal objetivo de las pruebas de tráfico.

2. SEGUNDO ESCENARIO DE PRUEBA

Una vez validados de forma unitaria los comportamientos y funcionalidades del conmutador cuando ejecuta el programa del router, se hace necesario introducir una topología con tres conmutadores que permita observar el correcto encaminamiento y el filtrado de paquetes especificado en la tabla *Firewall*.

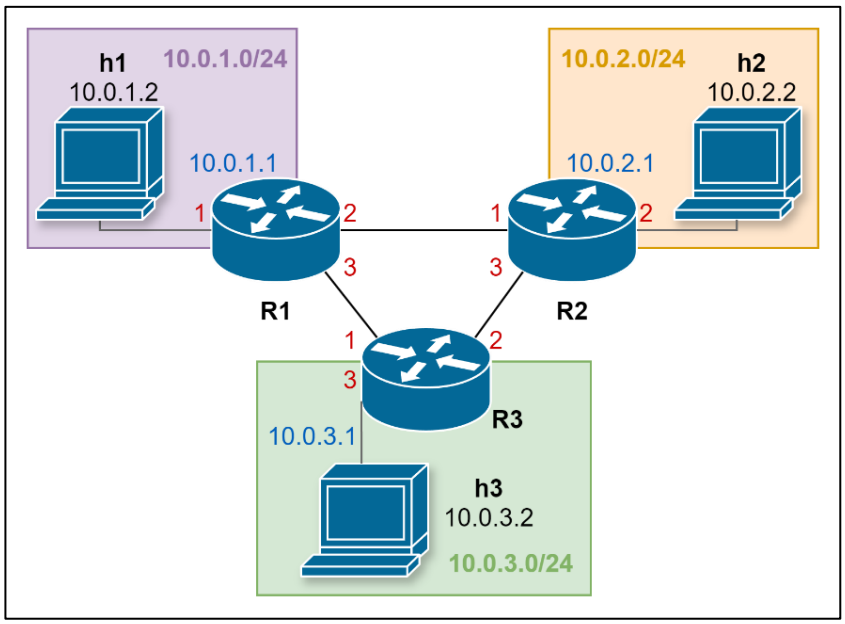


Figura 116. Segunda topología de prueba del router con firewall en BMv2

Se exponen a continuación los valores de las tablas para cada conmutador (R1, R2 y R3).

Switch: R1		
forward_ipv4		
Key	*Action*	Parameters
10.0.1.2	forward	08:00:00:00:00:01 1
10.0.2.0/24	forward	08:00:00:00:02:00 2
10.0.3.0/24	forward	08:00:00:00:03:00 3
missed	*drop*	

Switch: R2		
forward_ipv4		
Key	*Action*	Parameters
10.0.1.0/24	forward	08:00:00:00:01:00 1
10.0.2.2	forward	08:00:00:00:00:02 2
10.0.3.0/24	forward	08:00:00:00:03:00 3
missed	*drop*	

Switch: R3		
forward_ipv4		
Key	*Action*	Parameters
10.0.1.0/24	forward	08:00:00:00:01:00 1
10.0.2.0/24	forward	08:00:00:00:02:00 2
10.0.3.2	forward	08:00:00:00:00:03 3
missed	*drop*	

Figura 117. Estado de las tablas de encaminamiento en el segundo escenario de prueba del router con firewall en BMv2

Una vez establecido es estado de las tablas para cada uno de los conmutadores presentes, se muestra el resultado de cada una de las pruebas ejecutadas, mostrando así comportamientos similares a las pruebas del escenario anterior.

PRUEBA DE ENCAMINAMIENTO EN RED

En este caso se emiten paquetes ICMP con origen en *h1* y destinos *h2* y *h3.* El objetivo de esta prueba es trasladar aquello validado en la primera topología de prueba a la segunda, y así validar correctamente que el comportamiento previamente validado se repite en una situación más compleja.

Para demostrar la validez del tráfico capturado, se han observado las tablas de encaminamiento, y cómo fluye el tráfico a través de los distintos conmutadores presentes para alcanzar el destino final establecido. Podemos observar esto en la captura de la siguiente página:

Time	Source	Destination	Interf	Protocol	Lengt	Info
-0.041915…	10.0.1.2	10.0.2.2	2	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64
-0.040324…	10.0.1.2	10.0.2.2	3	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=63
-0.040322…	10.0.1.2	10.0.2.2	0	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=63
-0.039067…	10.0.1.2	10.0.2.2	5	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=62
-0.039052…	10.0.2.2	10.0.1.2	5	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
-0.038278…	10.0.2.2	10.0.1.2	0	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
-0.038277…	10.0.2.2	10.0.1.2	3	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
-0.037792…	10.0.2.2	10.0.1.2	2	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=62
-0.002602…	10.0.1.2	10.0.3.2	2	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64
-0.000933…	10.0.1.2	10.0.3.2	6	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=63
-0.000930…	10.0.1.2	10.0.3.2	1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=63
0.0000000…	10.0.1.2	10.0.3.2	8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=62
0.0000155…	10.0.3.2	10.0.1.2	8	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
0.0007198…	10.0.3.2	10.0.1.2	1	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
0.0007213…	10.0.3.2	10.0.1.2	6	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
0.0014700…	10.0.3.2	10.0.1.2	2	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=62

Figura 118. Captura de la prueba de encaminamiento de la primera topología del router con firewall de filtrado en BMv2

Tras realizar un análisis del tráfico en detalle se ha validado de nuevo el comportamiento correcto de cada dispositivo, y su integración en una red en la que convive con otros que presentan un comportamiento lógico equivalente. La conclusión obtenida es la **validación de esta prueba**.

PRUEBA DE FILTRADO DEL FIREWALL

Para contextualizar adecuadamente el desarrollo de las pruebas de tráfico, se incluyen en primer lugar los valores de las tablas firewall de este escenario.

Switch: R1

firewall

Key		*Action*	Priority
Origin	Destination		
10.0.1.2	any : 49052	noAction	2
any	10.0.1.2	noAction	2
default		drop	1

Switch: R2

firewall

Key		*Action*	Priority
Origin	Destination		
any	10.0.1.0/24	noAction	2
any	10.0.2.0/24	noAction	2
any	10.0.3.0/24	noAction	2
10.0.2.0/24 : 80	10.0.3.0/24	drop	3
default		drop	1

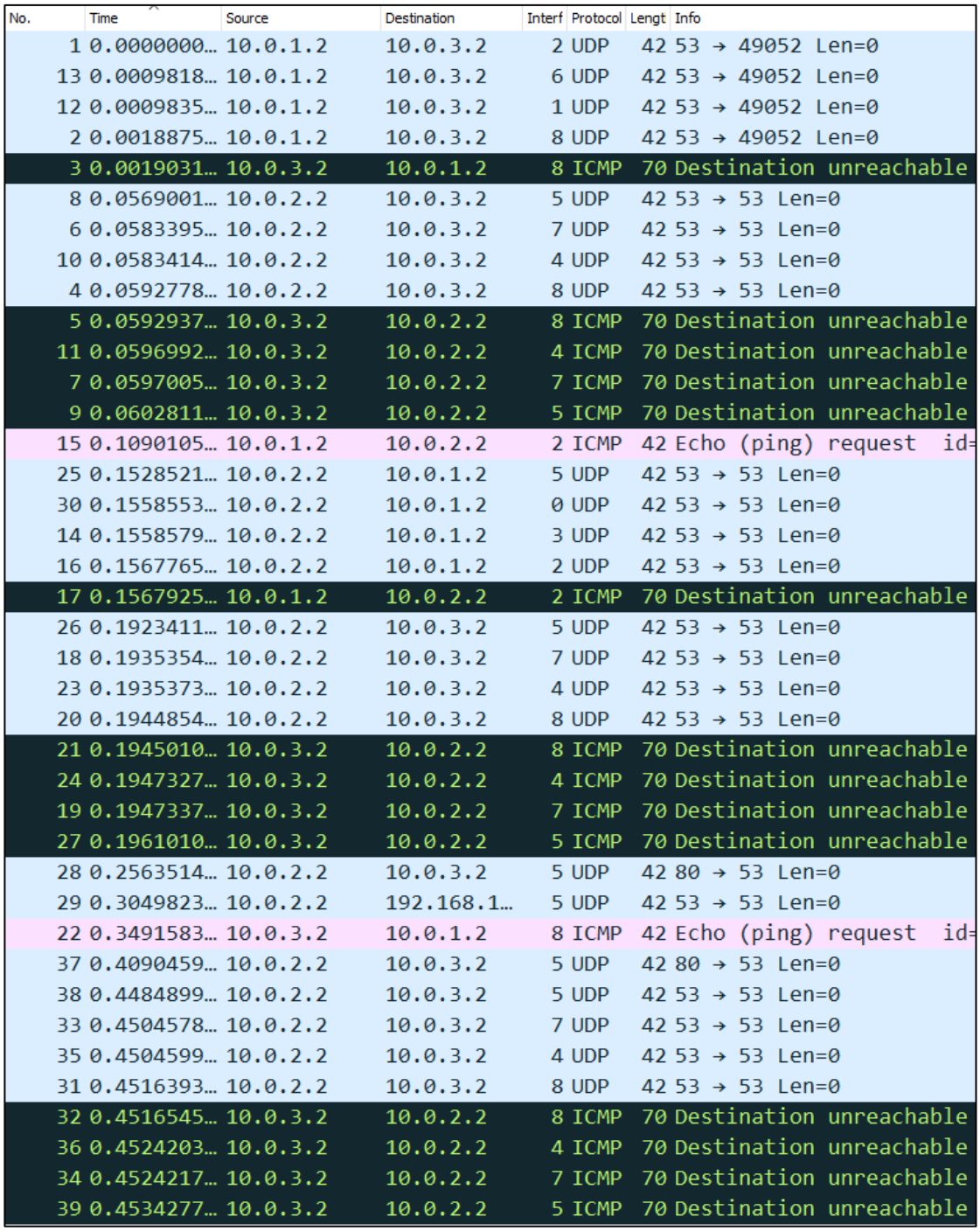
Switch: R3

firewall

Key		*Action*	Priority
Origin	Destination		
any	10.0.1.0/24	drop	1
any : 80	10.0.3.2	drop	1

Figura 119. Estado de las tablas Firewall en la segunda topología del router con firewall en BMv2

Para probar el correcto funcionamiento de las tablas, se han emitido paquetes desde todos los equipos, que han sido elegidos para ejecutar todas las acciones de las entradas contenidas.



No.	Time	Source	Destination	Interf	Protocol	Lengt	Info
1	0.0000000…	10.0.1.2	10.0.3.2	2	UDP	42	53 → 49052 Len=0
13	0.0009818…	10.0.1.2	10.0.3.2	6	UDP	42	53 → 49052 Len=0
12	0.0009835…	10.0.1.2	10.0.3.2	1	UDP	42	53 → 49052 Len=0
2	0.0018875…	10.0.1.2	10.0.3.2	8	UDP	42	53 → 49052 Len=0
3	0.0019031…	10.0.3.2	10.0.1.2	8	ICMP	70	Destination unreachable
8	0.0569001…	10.0.2.2	10.0.3.2	5	UDP	42	53 → 53 Len=0
6	0.0583395…	10.0.2.2	10.0.3.2	7	UDP	42	53 → 53 Len=0
10	0.0583414…	10.0.2.2	10.0.3.2	4	UDP	42	53 → 53 Len=0
4	0.0592778…	10.0.2.2	10.0.3.2	8	UDP	42	53 → 53 Len=0
5	0.0592937…	10.0.3.2	10.0.2.2	8	ICMP	70	Destination unreachable
11	0.0596992…	10.0.3.2	10.0.2.2	4	ICMP	70	Destination unreachable
7	0.0597005…	10.0.3.2	10.0.2.2	7	ICMP	70	Destination unreachable
9	0.0602811…	10.0.3.2	10.0.2.2	5	ICMP	70	Destination unreachable
15	0.1090105…	10.0.1.2	10.0.2.2	2	ICMP	42	Echo (ping) request id=
25	0.1528521…	10.0.2.2	10.0.1.2	5	UDP	42	53 → 53 Len=0
30	0.1558553…	10.0.2.2	10.0.1.2	0	UDP	42	53 → 53 Len=0
14	0.1558579…	10.0.2.2	10.0.1.2	3	UDP	42	53 → 53 Len=0
16	0.1567765…	10.0.2.2	10.0.1.2	2	UDP	42	53 → 53 Len=0
17	0.1567925…	10.0.1.2	10.0.2.2	2	ICMP	70	Destination unreachable
26	0.1923411…	10.0.2.2	10.0.3.2	5	UDP	42	53 → 53 Len=0
18	0.1935354…	10.0.2.2	10.0.3.2	7	UDP	42	53 → 53 Len=0
23	0.1935373…	10.0.2.2	10.0.3.2	4	UDP	42	53 → 53 Len=0
20	0.1944854…	10.0.2.2	10.0.3.2	8	UDP	42	53 → 53 Len=0
21	0.1945010…	10.0.3.2	10.0.2.2	8	ICMP	70	Destination unreachable
24	0.1947327…	10.0.3.2	10.0.2.2	4	ICMP	70	Destination unreachable
19	0.1947337…	10.0.3.2	10.0.2.2	7	ICMP	70	Destination unreachable
27	0.1961010…	10.0.3.2	10.0.2.2	5	ICMP	70	Destination unreachable
28	0.2563514…	10.0.2.2	10.0.3.2	5	UDP	42	80 → 53 Len=0
29	0.3049823…	10.0.2.2	192.168.1…	5	UDP	42	53 → 53 Len=0
22	0.3491583…	10.0.3.2	10.0.1.2	8	ICMP	42	Echo (ping) request id=
37	0.4090459…	10.0.2.2	10.0.3.2	5	UDP	42	80 → 53 Len=0
38	0.4484899…	10.0.2.2	10.0.3.2	5	UDP	42	53 → 53 Len=0
33	0.4504578…	10.0.2.2	10.0.3.2	7	UDP	42	53 → 53 Len=0
35	0.4504599…	10.0.2.2	10.0.3.2	4	UDP	42	53 → 53 Len=0
31	0.4516393…	10.0.2.2	10.0.3.2	8	UDP	42	53 → 53 Len=0
32	0.4516545…	10.0.3.2	10.0.2.2	8	ICMP	70	Destination unreachable
36	0.4524203…	10.0.3.2	10.0.2.2	4	ICMP	70	Destination unreachable
34	0.4524217…	10.0.3.2	10.0.2.2	7	ICMP	70	Destination unreachable
39	0.4534277…	10.0.3.2	10.0.2.2	5	ICMP	70	Destination unreachable

Figura 120. Captura de la prueba de filtrado de la segunda topología del router con firewall de filtrado en BMv2

Queda esta prueba por lo tanto debidamente **validada**.

7.3.3. Router con traducción NAT

Como se ha realizado en los casos anteriores, se muestra la estructura del programa que implementa el servicio que pretende ofrecer un router con traducción NAT.

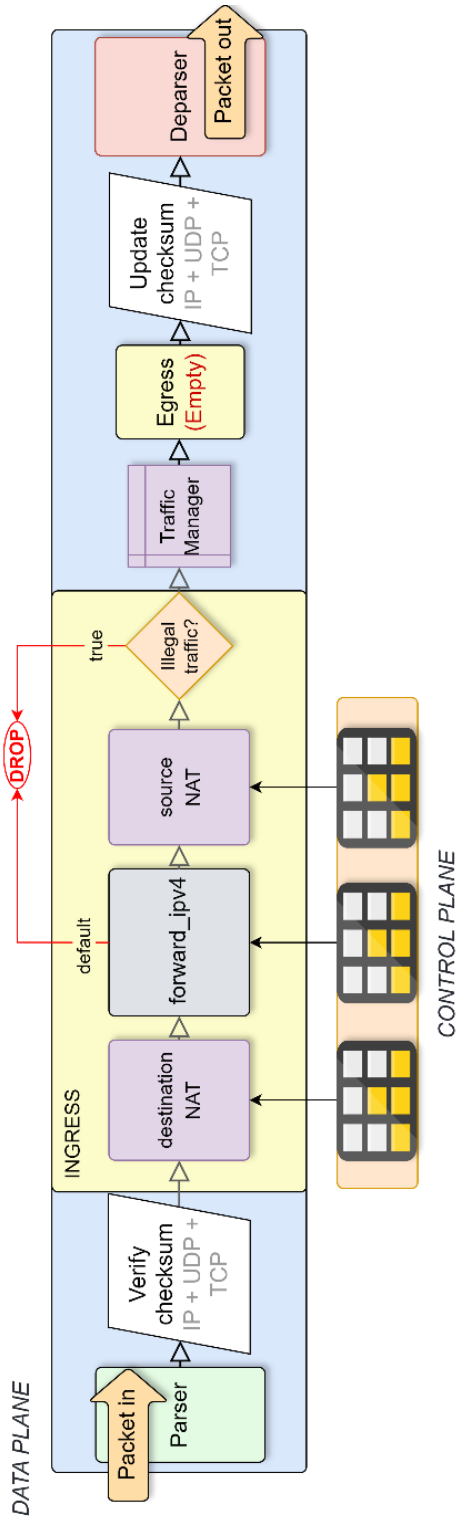


Figura 121. Pipeline del router NAT desarrollado en BMv2

Derivado de la funcionalidad que se desea probar, la traducción de direcciones privadas y la comunicación entre redes públicas y privadas, se ha estimado una única topología que permita confirmar el correcto funcionamiento del mecanismo definido.

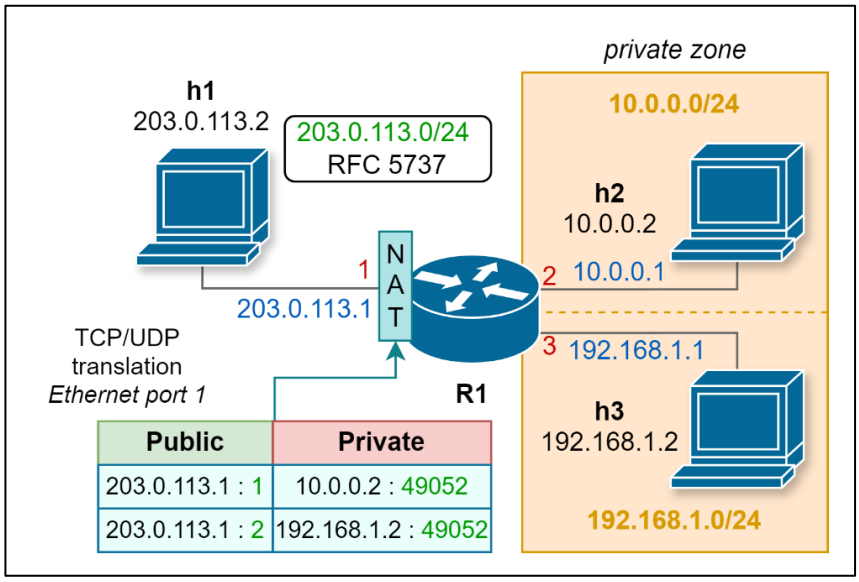


Figura 122. Topología de prueba del router NAT en BMv2

Se observa un equipo (*h1*) situado en una red que presenta direccionamiento público. Cabe destacar que la dirección asignada al mismo es una IP pública reservada para casos de documentación [38], y así evitar conflictos con una que se encuentre asignada. Por otra parte, se mantienen dos subredes privadas, con espacios de direccionamiento privados [33]. Por su parte, la tabla NAT mostrada en el diagrama de la topología se divide en las dos que se muestra a continuación, para cumplir con los requisitos y el diseño del servicio que está definido en su correspondiente apartado.

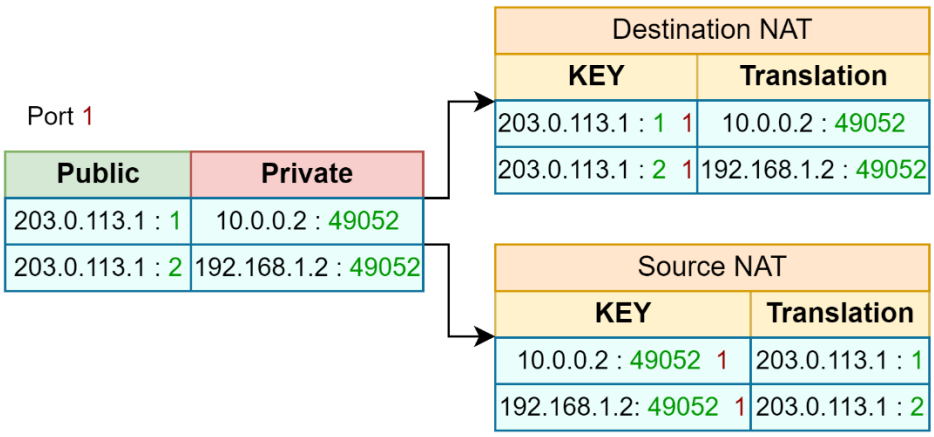


Figura 123. Estructura lógica con valores de la tabla NAT en BMv2

También se define la tabla de encaminamiento para este caso, teniendo en cuenta que su comportamiento ya se validó en el servicio anterior.

Switch: R1		
forward_ipv4		
Key	*Action*	Parameters
203.0.113.2	forward	08:00:00:00:00:01 1
10.0.0.2	forward	08:00:00:00:00:02 2
172.16.0.2	forward	08:00:00:00:00:03 3
missed	*drop*	

Figura 124. Tabla de encaminamiento del router NAT en BMv2

PRUEBA DE TRADUCCIÓN

En este caso se ha generado tráfico que permita validar el bloqueo de comunicaciones entre redes privadas y públicas cuando no se define una traducción y la correcta traducción de paquetes TCP y UDP.

No.	Time	Source	Destination	Interf	Protocol	Lengt	Info
1	0.00000…	203.0.113.2	10.0.0.2	0	ICMP	42	Echo (ping) request
2	0.08488…	192.168.1.2	203.0.113.2	2	ICMP	42	Echo (ping) request
8	0.13260…	203.0.113.2	203.0.113.1	0	UDP	42	53 → 1 Len=0
16	0.13380…	203.0.113.2	10.0.0.2	1	UDP	42	53 → 49052 Len=0
17	0.13382…	10.0.0.2	203.0.113.2	1	ICMP	70	Destination unreacha
18	0.17056…	10.0.0.2	203.0.113.2	1	UDP	42	49052 → 53 Len=0
9	0.17460…	203.0.113.1	203.0.113.2	0	UDP	42	1 → 53 Len=0
10	0.17462…	203.0.113.2	203.0.113.1	0	ICMP	70	Destination unreacha
19	0.21209…	10.0.0.2	203.0.113.2	1	TCP	54	49052 → 80 [SYN] Seq
11	0.21367…	203.0.113.1	203.0.113.2	0	TCP	54	1 → 80 [SYN] Seq=0 W
12	0.21368…	203.0.113.2	203.0.113.1	0	TCP	54	80 → 1 [RST, ACK] Se
20	0.21446…	203.0.113.2	10.0.0.2	1	TCP	54	80 → 49052 [RST, ACK
13	0.27311…	203.0.113.2	203.0.113.1	0	UDP	42	53 → 2 Len=0
3	0.27391…	203.0.113.2	192.168.1.2	2	UDP	42	53 → 49052 Len=0
4	0.27393…	192.168.1.2	203.0.113.2	2	ICMP	70	Destination unreacha
5	0.31203…	192.168.1.2	203.0.113.2	2	UDP	42	49052 → 53 Len=0
14	0.31324…	203.0.113.1	203.0.113.2	0	UDP	42	2 → 53 Len=0
15	0.31326…	203.0.113.2	203.0.113.1	0	ICMP	70	Destination unreacha
21	0.35224…	10.0.0.2	192.168.1.2	1	ICMP	42	Echo (ping) request
6	0.35311…	10.0.0.2	192.168.1.2	2	ICMP	42	Echo (ping) request
7	0.35312…	192.168.1.2	10.0.0.2	2	ICMP	42	Echo (ping) reply
22	0.35368…	192.168.1.2	10.0.0.2	1	ICMP	42	Echo (ping) reply

Figura 125. Captura de la prueba de traducción en BMv2

Si analizamos con detenimiento la captura observaremos que:

- La traducción se realiza de manera exitosa en ambos sentidos.

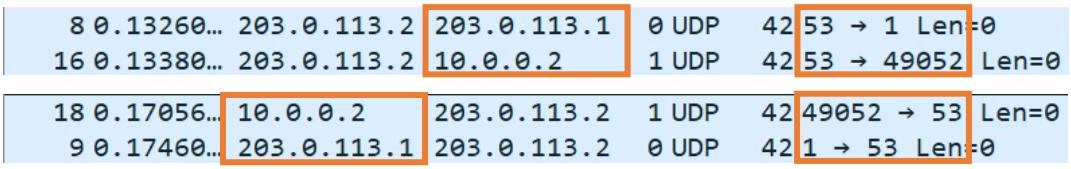


Figura 126. Evidencia de traducción NAT bidireccional

- Se permite una comunicación entre redes privadas.

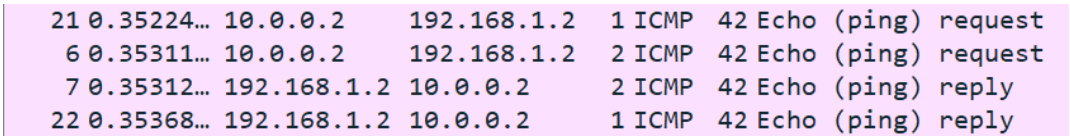


Figura 127. Evidencia de comunicación entre redes privadas

- Se bloquea el tráfico no permitido cuando no aplica la traducción (público a privado y viceversa).

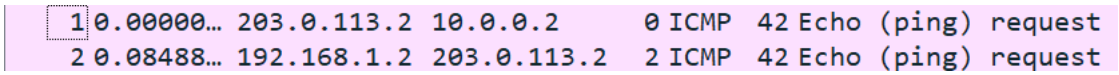


Figura 128. Evidencia de bloqueo de tráfico no permitido

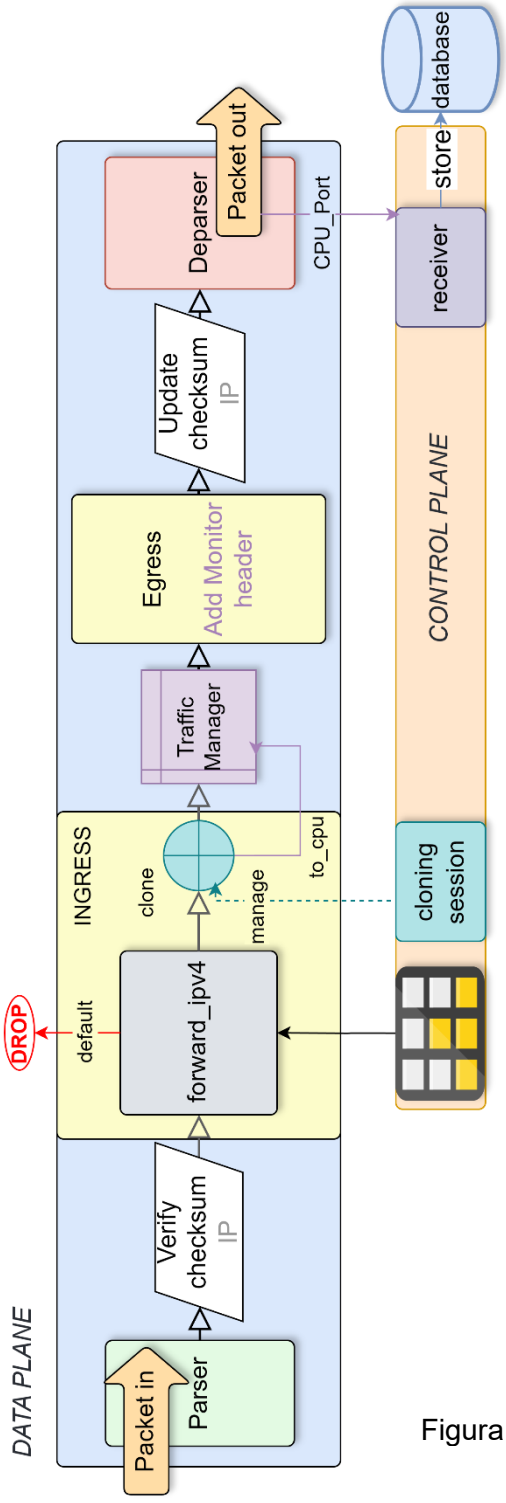
Para determinar la privacidad de una dirección, se emplean las tablas P4:

```
table private_src {
        key = { headers.ipv4.src : ternary; }
        actions = {
            NoAction;
        }
        const entries = {
            (0x0A000000 &&& 0xFF000000) : NoAction; // 10.0.0.0/8
            (0xAC100000 &&& 0xFFF00000) : NoAction; // 172.16.0.0/12
            (0xC0A80000 &&& 0xFFFF0000) : NoAction; // 192.168.0.0/16
        }
    }
    table private_dest {
        key = { headers.ipv4.dest : ternary; }
        actions = {
            NoAction;
        }
        const entries = {
            (0x0A000000 &&& 0xFF000000) : NoAction; // 10.0.0.0/8
            (0xAC100000 &&& 0xFFF00000) : NoAction; // 172.16.0.0/12
            (0xC0A80000 &&& 0xFFFF0000) : NoAction; // 192.168.0.0/16
        }
    }
```

Figura 129. Tablas P4 que determinan la privacidad de direcciones IPv4

Por lo que se ha demostrado que la implementación desarrollada en BMv2 del router con traducción NAT funciona como se esperaba, y por lo tanto está debidamente validada.

7.3.4. Router con monitorizacion de tráfico



En el desarrollo del router con monitorización de tráfico, ha sido necesario establecer una sesión de **clonación** de paquetes al puerto de CPU (directo al controlador), una operación primitiva presente en la definición de la arquitectura *V1model*, y que permite crear una copia de cada paquete que entra en la línea de procesamiento. Esa copia resultante de la clonación ejecutará el bloque *Egress*, precisamente para añadir una nueva cabecera de tipo *Monitor*, que como recordará el lector fue definida en el modelo del servicio de monitorización (consultar Figura 73). En definitiva, el principal objetivo de esta implementación es mostrar este importante modelo de comunicación plano de datos – controlador.

Figura 130. Pipeline del router monitor desarrollado en BMv2

En este caso, se ha establecido una topología simple con dos hosts conectados al conmutador.

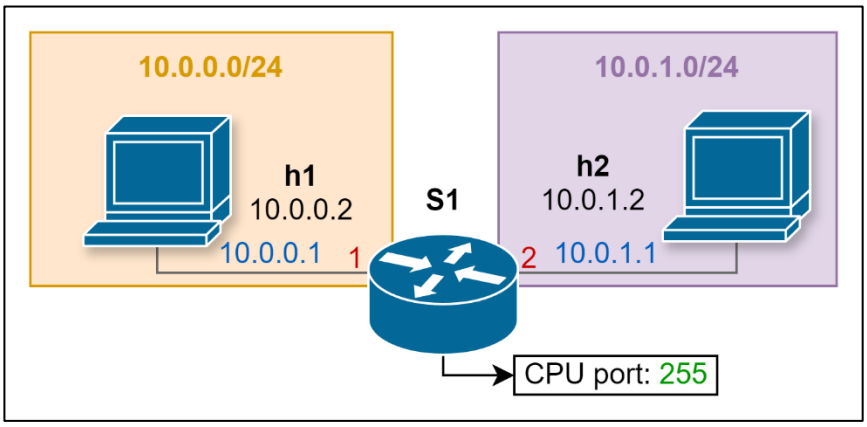


Figura 131. Topología de prueba del router monitor en BMv2

Por otro lado, como ya se ha mencionado, ha sido necesario establecer una sesión de clonación, en la que se ha establecido como puerto de clonado el *255* (como se muestra en la topología). Este valor se ha especificado al ejecutar Mininet, para adoptarlo como puerto de CPU.

Switch: S1	
CLONNING SESSIONS	
ID	**PORT**
1	255

Figura 132. Sesión de clonación del monitor

Adicionalmente, y para garantizar el correcto encaminamiento de paquetes como en pruebas anteriores, se ha establecido las entradas de la tabla que figuran en el diagrama:

Switch: S1		
forward_ipv4		
Key	*Action*	Parameters
10.0.0.2	forward	08:00:00:00:00:01 1
10.0.1.2	forward	08:00:00:00:00:02 2
missed	*drop*	

Figura 133. Tabla de encaminamiento del router monitor en BMv2

Veremos en la siguiente tabla que muestra correctamente su funcionamiento.

PRUEBA DE MONITORIZACIÓN

Este caso de prueba es relativamente sencillo, pues trabajamos sobre un router ya validado, pero que implementa el mecanismo de emisión de paquetes especificado. Para generar tráfico de prueba simplemente se ha implementado un generador con *Scapy* que emita constantemente tráfico ICMP entre los dos equipos presentes en la topología. En ese sentido, la siguiente captura muestra parte del tráfico generado:

No.	Time	Source	Destination	Interf	Protocol	Lengt	Info
1	0.000000	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
2	0.002577	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
3	0.001864	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
4	0.001878	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
5	1.002643	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
6	1.006316	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
7	1.005633	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
8	1.005647	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
9	2.007402	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
10	2.009378	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
11	2.008562	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
12	2.008573	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
13	3.010977	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
14	3.013722	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
15	3.013264	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
16	3.013275	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
17	4.013546	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
18	4.014920	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
19	4.014438	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
20	4.014461	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
21	5.015597	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
22	5.017262	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
23	5.016880	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
24	5.016894	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
25	6.016541	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
26	6.018401	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
27	6.017711	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
28	6.017723	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
29	7.018116	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
30	7.020081	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
31	7.019362	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
32	7.019374	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply
33	8.020672	10.0.0.2	10.0.1.2		ICMP	42	Echo (ping) request
34	8.026621	10.0.1.2	10.0.0.2		ICMP	42	Echo (ping) reply

Figura 134. Captura del tráfico de prueba monitorizado

Para validar que llegan los paquetes al controlador, y, que por lo tanto insertan correctamente las entradas en la base de datos, se ha incluido una salida por pantalla en la ejecución del programa controlador, donde aparecen los paquetes clonados:

```
(.venv) p4@p4:~/TFG/TFG-DesarrolloSDN-P4/src/P4-project/BMv2-version/SDN_services/Monitor-switch$ make control
python control.py
Packet from 10.0.0.2 to 10.0.1.2
Ingress: 38985521, Egress: 38986227, process_time = 706, queue_time = 207, time_unit = MICROSECONDS
Packet from 10.0.1.2 to 10.0.0.2
Ingress: 38986537, Egress: 38990493, process_time = 3956, queue_time = 3616, time_unit = MICROSECONDS
Packet from 10.0.0.2 to 10.0.1.2
Ingress: 39986862, Egress: 39987422, process_time = 560, queue_time = 179, time_unit = MICROSECONDS
Packet from 10.0.1.2 to 10.0.0.2
Ingress: 39988297, Egress: 39989812, process_time = 1515, queue_time = 1211, time_unit = MICROSECONDS
Packet from 10.0.0.2 to 10.0.1.2
Ingress: 40989443, Egress: 40990219, process_time = 776, queue_time = 239, time_unit = MICROSECONDS
Packet from 10.0.1.2 to 10.0.0.2
Ingress: 40990884, Egress: 40991666, process_time = 782, queue_time = 376, time_unit = MICROSECONDS
Packet from 10.0.0.2 to 10.0.1.2
Ingress: 41990404, Egress: 41991181, process_time = 777, queue_time = 304, time_unit = MICROSECONDS
Packet from 10.0.1.2 to 10.0.0.2
Ingress: 41991722, Egress: 41992912, process_time = 1190, queue_time = 365, time_unit = MICROSECONDS
Packet from 10.0.0.2 to 10.0.1.2
```

Figura 135. Salida por consola del controlador del router monitor

Las estadísticas temporales recibidas en la cabecera *Monitor* se procesan en el controlador, y son las siguientes:

- **Tiempo de procesamiento**: Diferencia absoluta entre los campos *ingress_time* y *egress_time*.
- **Tiempo en cola del Traffic Manager**: Valor del campo *deq_time*.

Una vez comprobada la correcta comunicación por el puerto de CPU, nos desplazaremos a la interfaz de la base de datos *InfluxDB*, donde se ha configurado una consulta a los datos en tiempo real, pudiendo visualizar una gráfica generada a partir de los paquetes que circulan en un instante. Observamos con detección las mediciones mostradas, teniendo en cuenta que el tiempo de procesamiento siempre será lógicamente mayor que el tiempo en cola, puesto que el primero incluye al segundo si tenemos en cuenta la organización del pipeline, pues el Traffic Manager se sitúa entre las etapas *Ingress* y *Egress*.

En la siguiente imagen podemos observar dos gráficas mostradas cuando está transitando el tráfico, debido a que cada una de ellas se corresponde con un sentido de circulación:

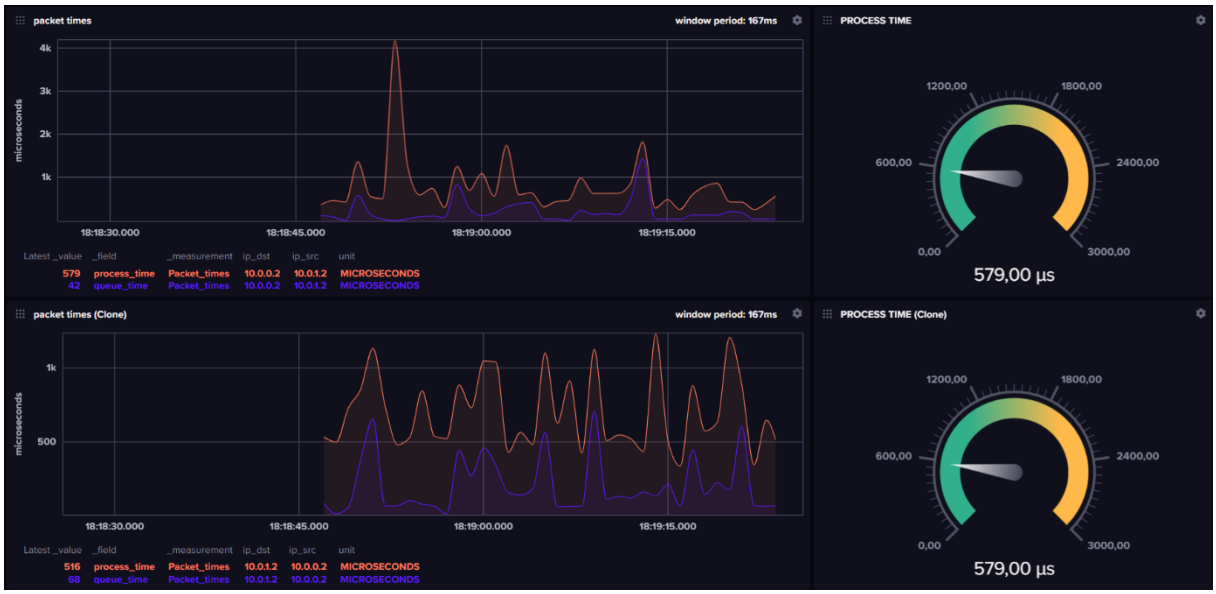


Figura 136. Gráficas temporales de monitorización de tiempos de procesamiento

Podemos realizar las representaciones que se estimen, a partir de los datos almacenados.

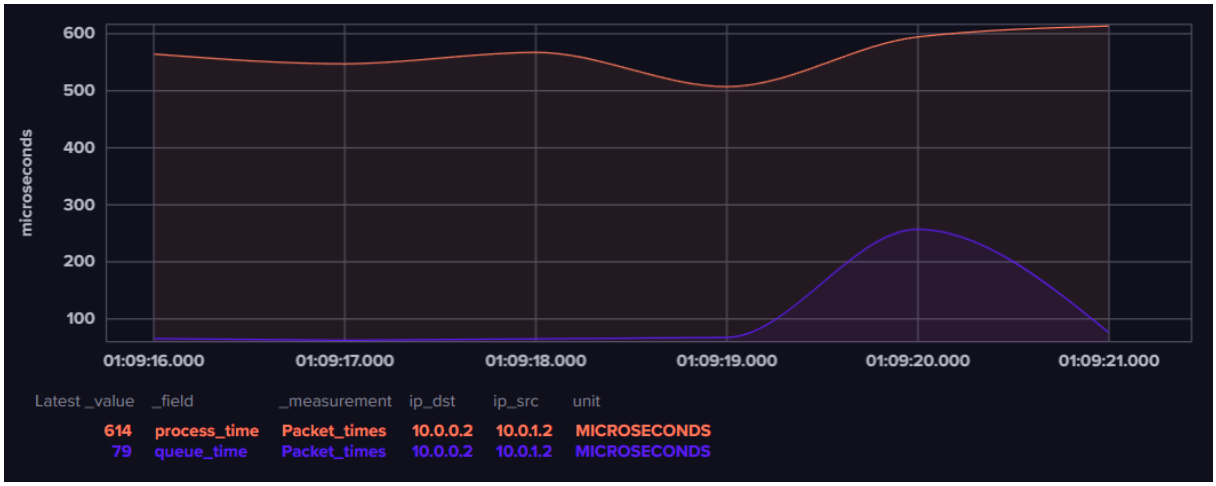


Figura 137. Detalle aumentado de la gráfica de monitorización

Como conclusión, queda **validado** el router monitor.

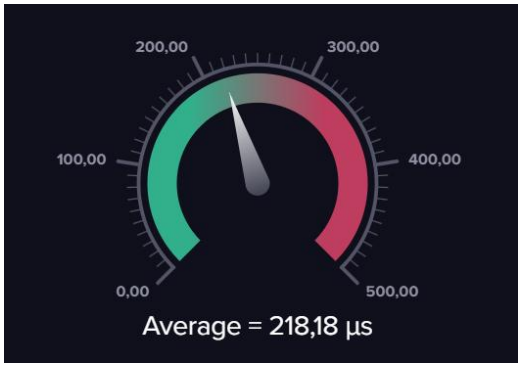


Figura 138. Diagrama de rueda generado a partir de estadísticas temporales

7.3.5. Router con políticas de calidad de servicio

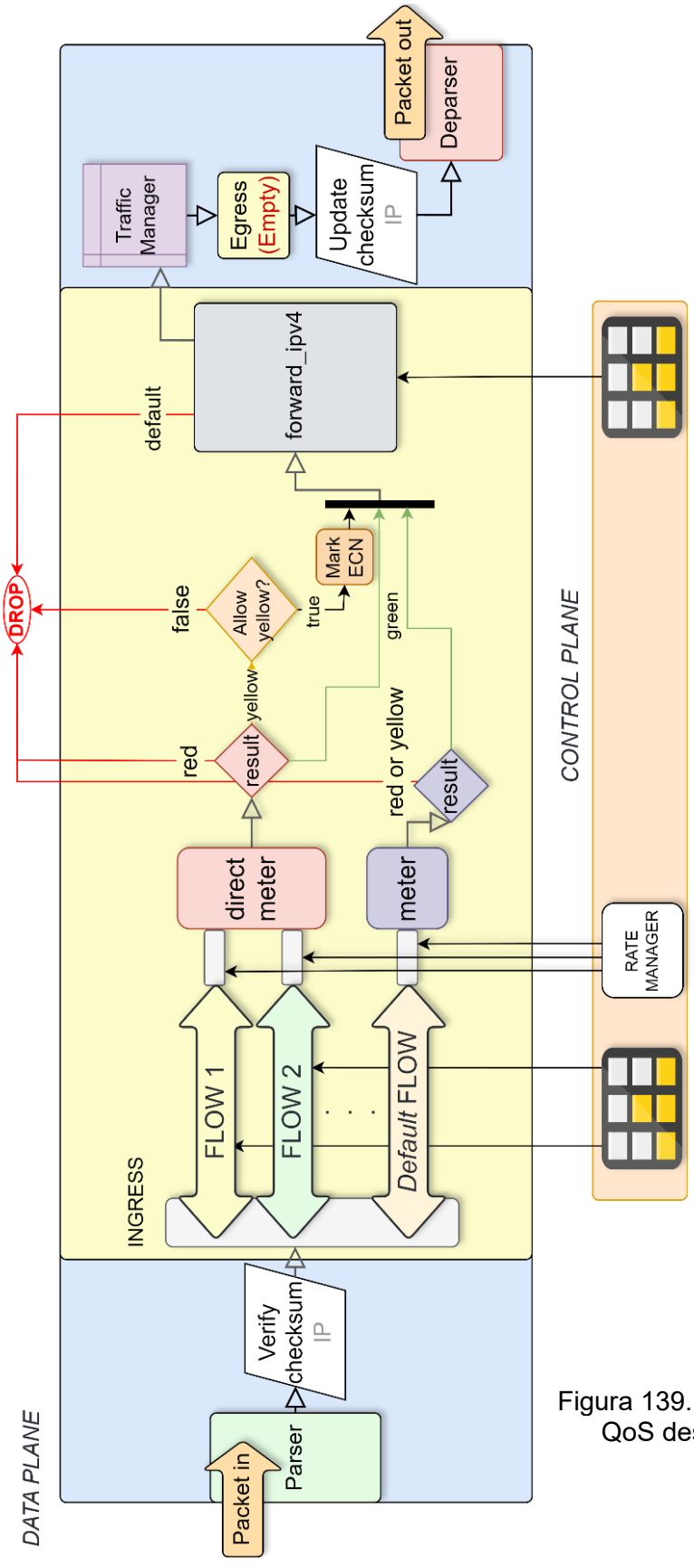


Figura 139. Pipeline del router con QoS desarrollado en BMv2

Llegamos al último servicio desarrollado en BMv2, el router con QoS. En este caso se aceptan paquetes que tengan la cabecera *Flow* ya definida anteriormente. La implementación realizada hace uso de los meter disponibles en esta arquitectura, donde se mantendrá uno asociado a la tabla de flujos definida, y otro que medirá los paquetes que no posean flujo especificado.

En este caso, como queremos validar el correcto funcionamiento de los flujos de comunicación, emplearemos la misma topología del servicio anterior:

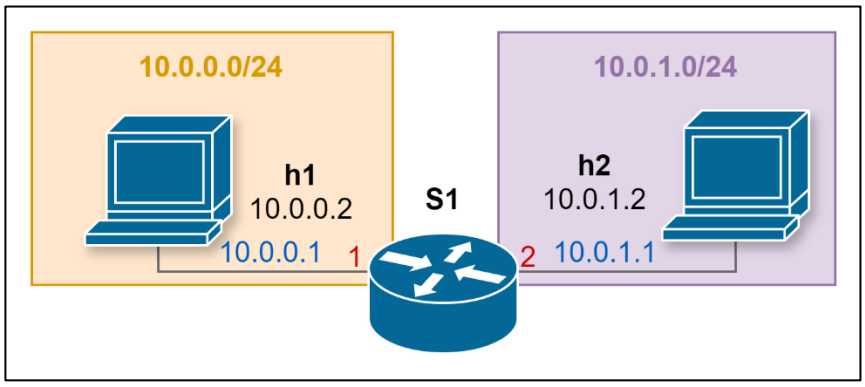


Figura 140. Topología de prueba para el router con QoS en BMv2

Y definimos también la ya conocida tabla de encaminamiento:

Switch: S1		
forward_ipv4		
Key	*Action*	Parameters
10.0.0.2	forward	08:00:00:00:00:01 1
10.0.1.2	forward	08:00:00:00:00:02 2
missed	*drop*	

Figura 141. Tabla de encaminamiento del router con QoS en BMv2

Para poder probar el comportamiento de los flujos ante diferentes situaciones de uso del ancho de banda del que disponen, se ha estimado la siguiente distribución de los medidores:

Switch: S1						
flow_meter						
Key	*Action*	*CIR (Bytes per second)*	*Cburst (Bytes)*	*PIR (Bytes per second)*	*Pburst (Bytes)*	*Allow yellow*
1	apply_meter	125	1500	250	3000	true
2	apply_meter	125	1500	250	3000	false

Switch: S1				
default meter				
Index	*CIR (Bytes per second)*	*Cburst (Bytes)*	*PIR (Bytes per second)*	*Pburst (Bytes)*
0	125	1500	250	3000

Figura 142. Estado de los meters en BMv2

Los valores de las tasas han sido elegidos para que sean fácilmente alcanzables. Nótese que la tabla *flow_meter* es una tabla P4 que presenta medidores asociados a cada una de las entradas. Para poder visualizar la diferencia de comportamiento entre el flujo 1, que admite mediciones con resultado AMARILLO, y el flujo 2 que no lo permite, se les ha asignado el mismo valor de tasas del medidor, para que sea posible notar la diferencia de comportamiento bajo las mismas condiciones.

PRUEBA DE MEDICIÓN

Para acometer esta prueba, se ha implementado un generador de tráfico que emite paquetes con una determinada frecuencia establecida y de forma constante, desde *h1* hasta *h2*. Estos han sido configurados con el tamaño necesario para superar las tasas de tráfico, y poder experimentar todos los resultados posibles. Veremos tres casos:

1. COMPORTAMIENTO DE UN PAQUETE SIN FLUJO

Time	Source	Destination	Interf	Protocol	Length	Info
-0.000913…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.0000000…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321 Len=158
0.0000195…	10.0.1.2	10.0.0.2	1	ICMP	228	Destination unreachabl
0.0003457…	10.0.1.2	10.0.0.2	0	ICMP	228	Destination unreachabl
0.0718576…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.0734703…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321 Len=158
0.0734863…	10.0.1.2	10.0.0.2	1	ICMP	228	Destination unreachabl
0.0743561…	10.0.1.2	10.0.0.2	0	ICMP	228	Destination unreachabl
0.1243640…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.1258905…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321 Len=158
0.1259817…	10.0.1.2	10.0.0.2	1	ICMP	228	Destination unreachabl
0.1267190…	10.0.1.2	10.0.0.2	0	ICMP	228	Destination unreachabl
0.1683222…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.1692106…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321 Len=158
0.1692269…	10.0.1.2	10.0.0.2	1	ICMP	228	Destination unreachabl
0.2044917…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.2517505…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.2986118…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.3442758…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.3759242…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.4321314…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.4721316…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.5165971…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.5601541…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158
0.5919454…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321 Len=158

Figura 143. Captura de paquetes sin flujo en BMv2

La comprobación del correcto funcionamiento del programa tras la aplicación del medidor por defecto es rápida y sencilla. Hay que observar la zona resaltada, en la que el paquete emitido se retransmite con total normalidad; pero hasta llegar a la zona no resaltada, donde se ha excedido la tasa máxima permitida, y el conmutador no permite el tránsito por él. Por lo que queda totalmente **validado**. Para poder volver a recuperar la correcta comunicación, habrá que dejar pasar un tiempo hasta que el dispositivo determine un ancho de banda correcto al realizar la medición.

2. COMPORTAMIENTO DE UN PAQUETE CON FLUJO 1

Time	Source	Destination	Interf	Protocol	Length	Info
1.2519382…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.2537997…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.3237890…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.3247680…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.3743252…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.3762299…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.4241353…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.4252188…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.4601892…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.4611948…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.4997928…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.5007759…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.5519243…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.5528139…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.5851849…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.5862376…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.6321315…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.6332747…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.6642199…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.6651969…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.7079110…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.7089305…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.7401329…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.7420444…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
1.7800739…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.8126692…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
1.8478900…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321

Figura 144. Captura del flujo 1 de QoS en BMv2

En este caso presentamos tres regiones bien diferenciadas, que han sido señaladas con el color devuelto por el medidor. Si observamos bien, veremos que la comunicación quedaría bloqueada exclusivamente en la zona de color rojo, por lo que efectivamente se está aceptando el tráfico marcado como amarillo.

Veamos dos ejemplos en detalle; un paquete marcado como verde y otro como amarillo:

- Paquete de la zona verde:

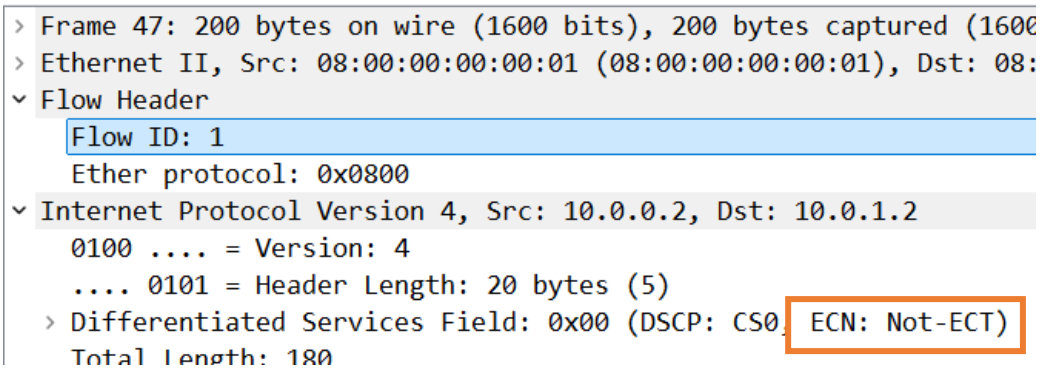


Figura 145. Paquete del flujo 1 marcado como verde

- Paquete de la zona amarilla:

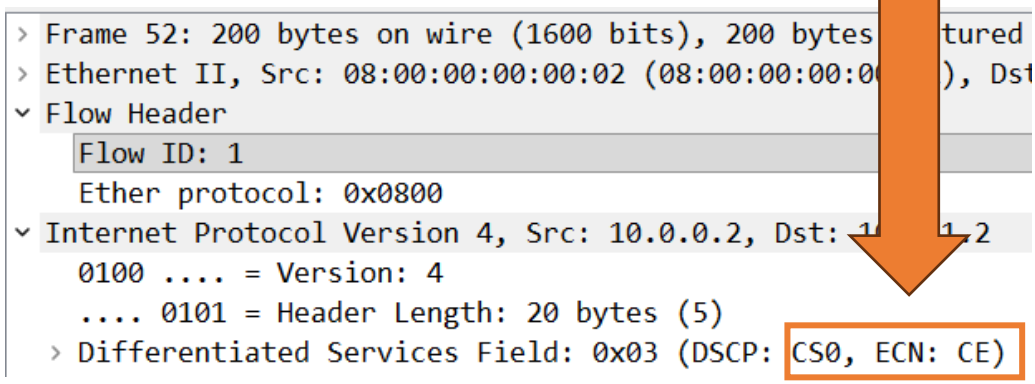


Figura 146. Paquete del flujo 1 marcado como amarillo

En el primero se indica que no hay congestión, mientras que en el segundo se notifica el flag ECN (Explicit Congestion Notification). Esto nos indica el correcto funcionamiento de la lógica aplicada al paquete tras la medición, y podemos concluir que el funcionamiento de este flujo queda **validado**.

3. COMPORTAMIENTO DE UN PAQUETE CON FLUJO 2

Time	Source	Destination	Interf	Protocol	Length	Info
2.1698127…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
2.2118275…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.2126836…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
2.2523356…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.2531513…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
2.3093003…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.3099671…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
2.3717553…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.3722472…	10.0.0.2	10.0.1.2	1	UDP	200	12345 → 54321
2.4080268…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.4806586…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.5242120…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.5807032…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.6241086…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.6721376…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.7239454…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.7745433…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.8406144…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321
2.8806141…	10.0.0.2	10.0.1.2	0	UDP	200	12345 → 54321

Figura 147.Captura del flujo 2 de QoS en BMv2

En este caso únicamente se muestran dos zonas diferenciadas:

- zona verde, donde los paquetes llegan a su destino.
- zona roja, donde se supera el ancho de banda acumulado, y el conmutador bloquea y descarta los paquetes. Nótese que cuando el paquete es marcado con amarillo, también pertenece a este grupo en el presente flujo.

Queda por lo tanto este último flujo **validado**, y por extensión todo el servicio de calidad de servicio, al cumplir con los requisitos definidos.

8

ADAPTACIÓN A LA PLATAFORMA TOFINO

En este último capítulo de desarrollo, introduciremos ligeramente el conmutador físico de laboratorio con el que se ha trabajado, y las adaptaciones de servicios que se han llevado a cabo. El principal objetivo de su empleo es mostrar el proceso de adaptación de programas implementados en BMv2 a una arquitectura de producción.

8.1. La arquitectura PSA

La arquitectura PSA (*Portable Switch Architecture*) está definida en las especificaciones del lenguaje P4 [39], y define una línea de procesamiento renovada y a la vez más compleja con respecto a la ya conocida *V1model*. Su objetivo es permitir que sea interoperable entre los distintos conmutadores programables con P4, sin importar el fabricante.

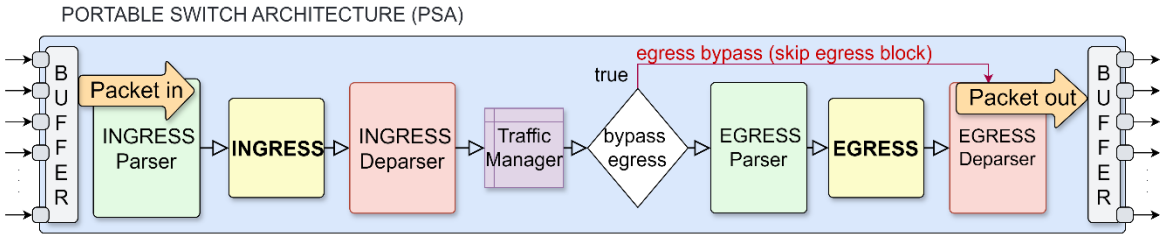


Figura 148. Pipeline de la arquitectura PSA

Como se aprecia en el diagrama de la línea de procesamiento, ya no existe un parser y un deparser global, sino que tanto el bloque de ingreso (ingress) como el de egreso (egress) presentan los suyos. Esto implica que cuando un paquete termina el procesamiento en el primer bloque, deberá ser ensamblado y posteriormente

analizado si tiene que ejecutar el segundo bloque. En ese sentido, se proporciona un mecanismo para evitar el bloque de egreso, y poder ser retransmitido, lo cual puede presentar un impacto notable. El resto de la línea de procesamiento es bastante similar, y las operaciones y externs disponibles son similares y ofrecen un comportamiento lógico equivalente.

La arquitectura de *Intel Tofino* sigue un modelo similar, pero equivalente a nivel estructural, y su archivo de librería P4 se especifica en un repositorio público [31].

8.2. Entorno de desarrollo de Tofino

En esta plataforma se ha trabajado con un entorno de desarrollo y despliegue ofrecido por el fabricante de este conmutador. Su acceso se encuentra restringido a personas autorizadas y grupos de investigación, y ofrece un entorno centralizado y ordenado de desarrollo y validación de programas de plano de datos implementados bajo su arquitectura. En él se localizan el compilador de este dispositivo, y demás herramientas que permiten depurar correctamente el código desarrollado.

Debemos tener clara la naturaleza de Tofino, que es un conmutador hardware desarrollado por la empresa estadounidense Intel [40]. Su estructura principal es una placa de procesamiento montada en un circuito integrado, que contiene una línea de procesamiento física programable, donde se ejecutan los programas P4 desarrollados para esta arquitectura. El revestimiento externo y la caja la integran fabricantes asociados; como ocurre en los ordenadores de uso personal.

Por su parte, el entorno de desarrollo ofrece un simulador de características similares a BMv2, que permite realizar pruebas tempranas en desarrollos, y observar el flujo de tráfico que circula por sus puertos. Esto se realiza así debido a que cuando se carga un programa en la placa física, no es posible depurar su comportamiento, ya que encontramos un hardware configurable, pero dedicado totalmente a ofrecer el procesamiento definido, y no puede observarse de forma clara el comportamiento interno. Por lo tanto, cuando se desarrolla un programa para este conmutador, se hace necesario llevar a cabo su validación previamente en el simulador. En el diagrama se muestra su estructura conceptual a alto nivel.

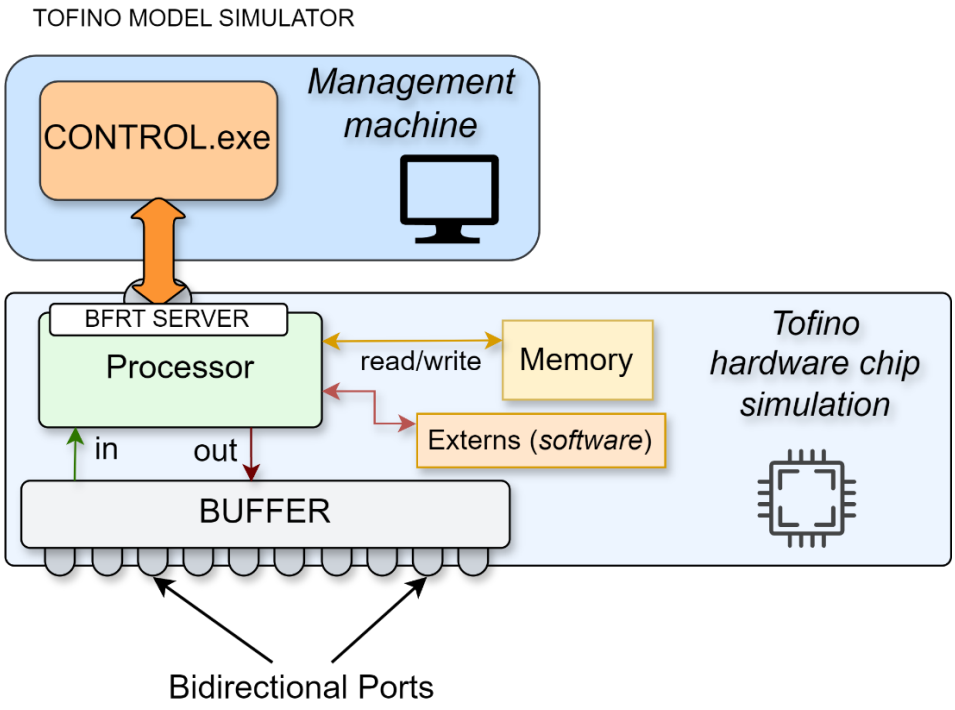


Figura 149. Estructura conceptual del simulador de Tofino

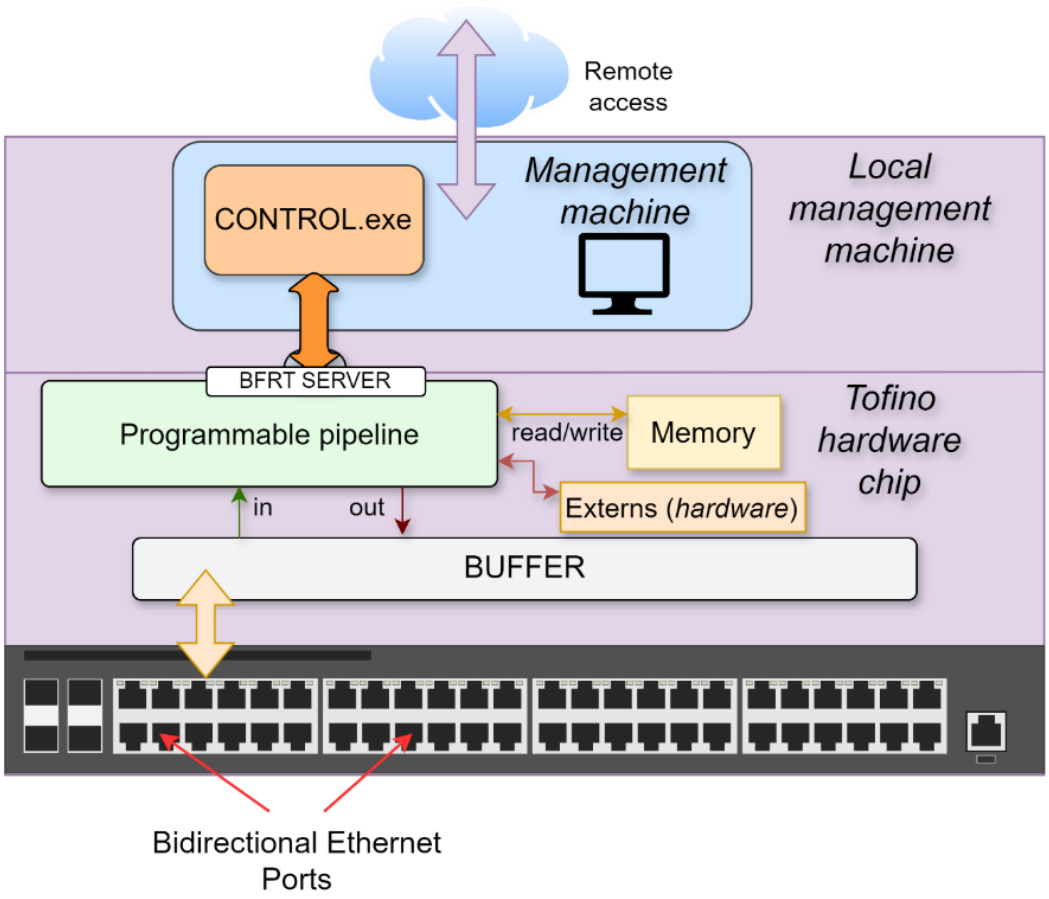


Figura 150. Estructura conceptual a alto nivel del hardware de Tofino

Típicamente, los fabricantes de conmutadores que integran a la unidad de procesamiento de Tofino establecen un modelo de control local, de tal manera que el dispositivo que comercializan dispone de un equipo con sistema operativo definible por parte del usuario, que permite establecer la conexión a la placa para ejecutar controladores P4.

En la actualidad el entorno de Tofino pretende convertirse en abierto para atraer talento, y así mejorar las soluciones que se están implementado. En la actualidad hay una versión reducida del simulador que es de libre acceso, pero aún está en proceso de adopción [41]. La apertura de esta plataforma hace vislumbrar un futuro prometedor de la programación de esta arquitectura, que será un cambio revolucionario en el mapa de las redes SDN.

8.3. Adaptación de servicios y verificación

Para mostrar el procedimiento de adaptación de servicios SDN a arquitecturas de producción, se ha elegido una selección de aquellos que resultan más interesantes en el contexto de este conmutador hardware, teniendo en cuenta también de que sólo se disponen dos equipos conectados al conmutador.

Partimos de programas de plano de datos debidamente validados, y a los que sólo será necesario aplicar una pequeña adaptación a las particularidades establecidas en este nuevo conmutador. En este contexto, el fabricante aporta un conmutador de manipulación de entidades del plano de datos, que ha sido empleado para realizar las mismas tareas de control ya desempeñadas en BMv2.

Ha de tenerse en cuenta que no es una labor sencilla, pues la documentación presente es escasa, y estamos hablando de un dispositivo que se encuentra en proceso de adopción, y sobre el que se está investigando en la actualidad; pero algo que aporta valor a los servicios que se han implementado anteriormente, y constituye una conclusión adecuada a este proceso de desarrollo. El flujo de trabajo ha incluido el uso del simulador, y posteriormente la puesta en ejecución en el hardware. Veamos a partir de la siguiente página las características y resultados obtenidos para cada uno de ellos.

8.3.1. Router con firewall de filtrado

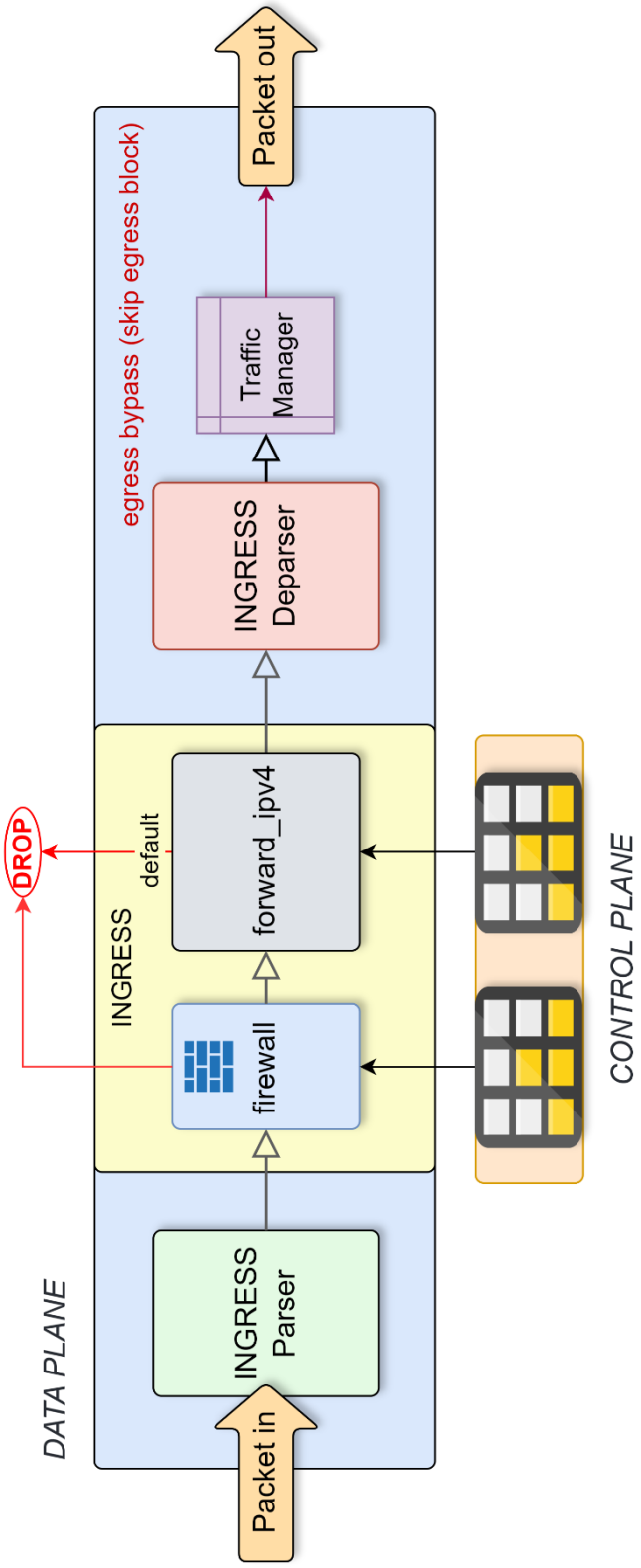


Figura 151. Pipeline del router con firewall adaptado a Tofino

Partiendo del programa P4 desarrollado en el conmutador anterior, la adaptación efectuada ha consistido en realizar una modificación del parser del bloque *Ingress*, que requiere de analizar metadatos propios de la arquitectura.

Otras de las diferencias clave con respecto al anterior es la presencia de *egress_bypass* que ha permitido saltar la etapa *Egress*, que no es necesaria en este programa.

A continuación, exponemos los escenarios de prueba establecidos, junto a los resultados, que serán equivalentes a BMv2 desde un punto de vista funcional.

El escenario de prueba es sencillo, debido a que se disponen dos equipos conectados al conmutador Tofino:

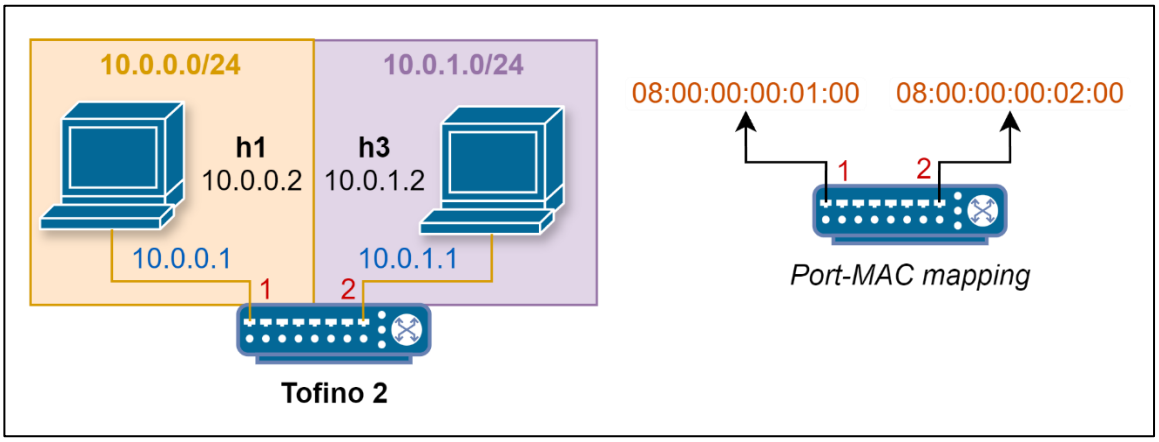


Figura 152. Topología del router con firewall en Tofino

En base a esa estructura, el controlador establece los siguientes valores para las dos tablas presentes en el Pipeline:

forward_IPv4		
Key	*Action*	Parameters
10.0.0.2	forward	08:00:00:00:00:01 1
10.0.1.2	forward	08:00:00:00:00:02 2
missed	*drop*	

firewall			
Key		*Action*	Priority
Origin	Destination		
10.0.0.2	any : 49052	noAction	3
any	10.0.1.2	noAction	2
any:80	any:80	noAction	3
default		drop	1

Figura 153. Estado de las tablas del router con firewall en Tofino

Se definen a continuación los casos de prueba junto a los resultados obtenidos en cada caso.

PRUEBA DE CORRECTA INTERPRETACIÓN DEL CAMPO “OPTIONS”

Al igual que en BMv2, verificaremos que el conmutador procesa este campo variable de IPv4. Para probar este aspecto, se ha emitido un paquete desde *h1* a *h2*.

```
> Ethernet II, Src: 08:00:00:00:02:00 (08:00:00:00:02:00), Dst: 08:00:00:00:00:02
v Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.1.2
    0100 .... = Version: 4
    .... 0110 = Header Length: 24 bytes (6)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 32
    Identification: 0x0001 (1)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 63
    Protocol: UDP (17)
    Header Checksum: 0x63c7 [correct]
    [Header checksum status: Good]
    [Calculated Checksum: 0x63c7]
    Source Address: 10.0.0.2
    Destination Address: 10.0.1.2
  > Options: (4 bytes)
```

Figura 154. Captura de la prueba del campo "options" del protocolo IPv4 en Tofino

Queda **validado**.

PRUEBA DE ENCAMINAMIENTO EN RED

Consiste en la emisión de tráfico UDP entre ambos equipos de la red.

No.	Time	Source	Destination	Interf	Protocol
1	0.0000000…	10.0.0.2	10.0.1.2	0	UDP
2	0.0260383…	10.0.1.2	10.0.0.2	1	UDP
3	0.2374970…	10.0.1.2	10.0.0.2	0	UDP
4	0.2369589…	10.0.0.2	10.0.1.2	1	UDP

Figura 155. Captura de la prueba de encaminamiento del router con firewall de filtrado en Tofino

Se puede observar que:

- Se decrementa el TTL: Se cumple.

No.	Time	Source	Destination	Interf	Protocol	Length	Time to Live	Info
1	0.0000000…	10.0.0.2	10.0.1.2	0	UDP	42	64	53 → 53 Len=0
4	0.2369589…	10.0.0.2	10.0.1.2	1	UDP	60	63	53 → 53 Len=0

Figura 156. Evidencia de decremento del valor de TTL en Tofino

- Se establece una MAC origen especificada en Port-MAC al salir del conmutador: Se cumple.

```
Ethernet II, Src: 08:00:00:00:00:01 (08:00:00:00:00:01), Dst: 08:00:00:00:01:00
Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.1.2
User Datagram Protocol, Src Port: 53, Dst Port: 53
```

```
Ethernet II, Src: 08:00:00:00:02:00 (08:00:00:00:02:00), Dst: 08:00:00:00:00:02
Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.1.2
User Datagram Protocol, Src Port: 53, Dst Port: 53
```

Figura 157. Evidencia del establecimiento de una nueva dirección MAC origen en Tofino

- Que se actualiza la suma de comprobación, tras modificar campos de la cabecera IPv4 (TTL): Se cumple.

ESTADO EN EL INGRESO	ESTADO EN EL EGRESO
Header Checksum: 0x65cd [correct] [Header checksum status: Good]	Header Checksum: 0x66cd [correct] [Header checksum status: Good]

Figura 158. Variación del valor de la suma de comprobación en Tofino

Por lo tanto, el encaminamiento se considera **validado**.

PRUEBA DE FILTRADO DEL FIREWALL

En este caso se emite tráfico en ambos sentidos que permite validar el funcionamiento del firewall declarado en el plano de datos. El firewall definido en este escenario permite el tráfico de los paquetes que coinciden con alguna de las entradas, descartando cualquier otro tráfico.

Time	Source	Destination	Interf	Protocol	Length	Time to Live	Info
0.0000000…	10.0.0.2	10.0.1.2	0	UDP	42	64	53 → 49052 Len=0
0.1095666…	10.0.0.2	10.0.1.2	1	UDP	60	63	53 → 49052 Len=0
0.5299120…	10.0.0.2	10.0.1.2	0	UDP	42	64	53 → 53 Len=0
0.5422719…	10.0.0.2	10.0.1.2	1	UDP	60	63	53 → 53 Len=0
1.0572639…	10.0.1.2	10.0.0.2	0	UDP	60	63	80 → 80 Len=0
1.0450736…	10.0.1.2	10.0.0.2	1	UDP	42	64	80 → 80 Len=0
1.5649551…	10.0.1.2	10.0.0.2	1	UDP	42	64	53 → 53 Len=0

Figura 159. Captura de la prueba de filtrado del router con firewall de filtrado en Tofino

En la captura se muestran 4 paquetes que circulan por ambas interfaces del conmutador, a excepción del último, que no presenta entrada en la tabla de firewall, y es descartado. Con esta prueba, queda **validada** la adaptación del router a Tofino.

8.3.2. Router con traducción NAT

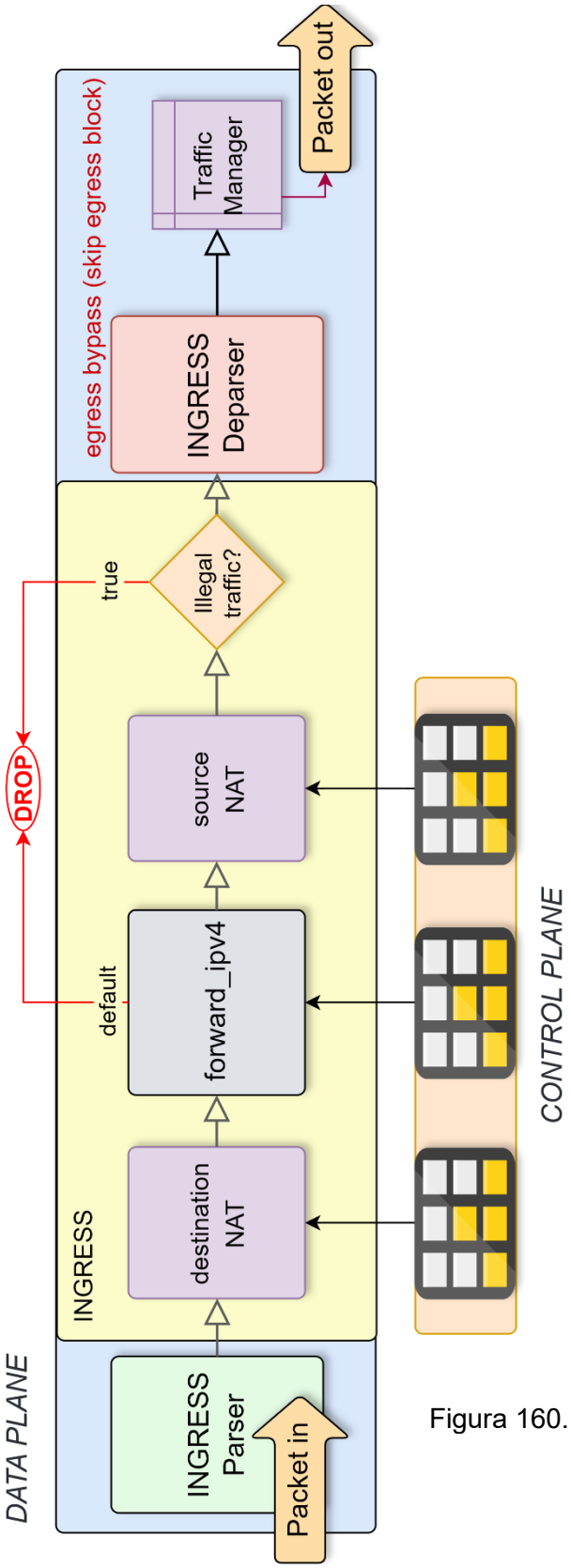


Figura 160. Pipeline del router NAT adaptado a Tofino

Para acometer la adaptación del router con traducción de direcciones privadas, se ha trabajado con optimizaciones necesarias a nivel de arquitectura, pero el programa en sí presenta una estructura muy similar con respecto al implementado en BMv2. En este caso, como en el servicio anterior, no se ejecuta el bloque *Egress*.

El escenario de prueba establecido para este caso está formado por dos redes bien diferenciadas, una pública y otra privada, de tal manera que pueda probarse la traducción entre ambas.

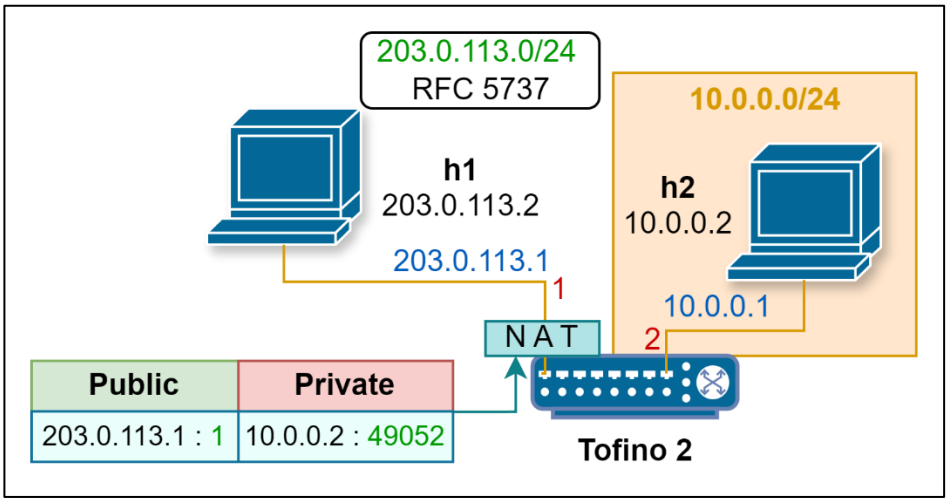


Figura 161. Topología del router NAT en Tofino

Las tablas del plano de datos presentan los siguientes valores:

forward_IPv4		
Key	*Action*	Parameters
203.0.113.2	forward	08:00:00:00:00:01 1
10.0.0.2	forward	08:00:00:00:00:02 2
missed	*drop*	

Port 1

Public	Private
203.0.113.1 : 1	10.0.0.2 : 49052

Destination NAT	
KEY	**Translation**
203.0.113.1 1 1	10.0.0.2 : 49052

Source NAT	
KEY	**Translation**
10.0.0.2 49052 1	203.0.113.1 : 1

Figura 162. Estado de las tablas del router NAT en Tofino

En la siguiente página se muestra el caso de prueba propuesto para este caso.

PRUEBA DE TRADUCCIÓN

Se ha generado tráfico entre ambos sentidos del conmutador, para mostrar dos sentidos de traducción. En este caso sólo tenemos una red privada, por lo que sólo podrá haber tráfico traducido, restringiendo los protocolos a TCP y UDP.

Time	Source	Destination	Interf	Protocol	Length	Time to Live	Info
0.0000000…	203.0.113.2	10.0.0.2	0	UDP	42	64	53 → 53 Len
0.5230044…	203.0.113.2	203.0.113.1	0	UDP	42	64	53 → 1 Len=0
0.6237672…	203.0.113.2	10.0.0.2	1	UDP	60	63	53 → 49052
1.0431936…	10.0.0.2	203.0.113.2	1	UDP	42	64	49052 → 53
1.0717219…	203.0.113.1	203.0.113.2	0	UDP	60	63	1 → 53 Len=0
1.5786874…	203.0.113.1	203.0.113.2	0	TCP	60	63	1 → 80 [SYN
1.5591881…	10.0.0.2	203.0.113.2	1	TCP	54	64	49052 → 80

Figura 163. Captura de la prueba de traducción en Tofino

Veamos en detalle los casos más importantes:

- El primer paquete sale del equipo *h1* y entra por el puerto 1 al conmutador, pero como tiene un destino privado, es descartado. Recordemos que el tráfico entre redes públicas y privadas no está permitido, salvo en escenarios de traducción.

203.0.113.2	10.0.0.2	0	UDP	42	64	53 → 53 Len=0

Figura 164. Detalle ampliado del paquete descartado en el router NAT en Tofino

- Cuando se envía el segundo paquete desde h1 a la dirección pública y el puerto de transporte virtual (1), se realiza la traducción del destino a de público a privado, y el origen permanece público.

0.5230044…	203.0.113.2	203.0.113.1	0	UDP	42	64 53 → 1 Len=0
0.6237672…	203.0.113.2	10.0.0.2	1	UDP	60	63 53 → 49052 Len=0

Figura 165. Detalle ampliado del paquete que se transmite de h1 a h2 en NAT de Tofino

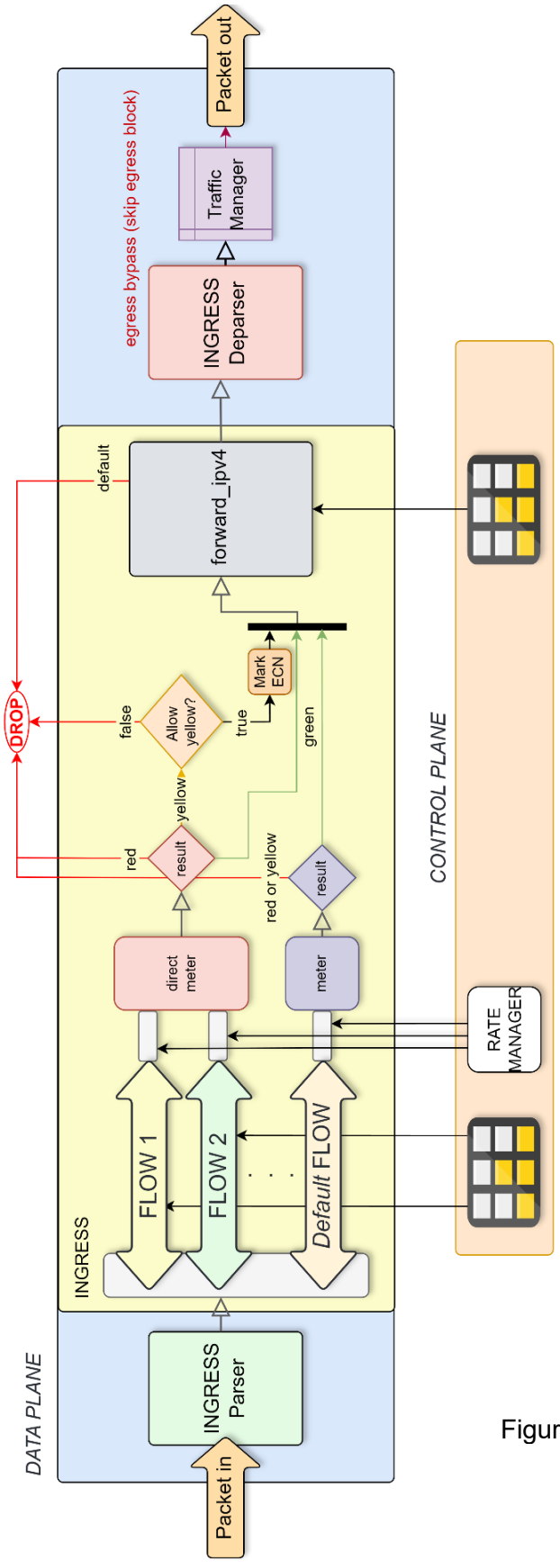
- En el sentido contrario, de h2 a h1, se realiza la traducción de la dirección de origen, de privada a pública.

10.0.0.2	203.0.113.2	1	UDP	42	64	49052 → 53 Len
203.0.113.1	203.0.113.2	0	UDP	60	63	1 → 53 Len=0

Figura 166. Detalle ampliado del paquete que se transmite de h2 a h1 en NAT de Tofino

Con esto queda **validado** el comportamiento del servicio para la arquitectura Tofino.

8.3.3. Router con políticas de calidad de servicio



En esta última adaptación a Tofino, se ha mantenido la estructura del bloque *Ingress*, pero utilizando medidores de tráfico propios de la arquitectura del conmutador. Uno de los detalles más importantes

En cuanto a los medidores, presentan un formato diferente a los de BMv2, sobre todo en las tasas, que emplean la unidad bits por segundo en lugar de bytes por segundo. No obstante, la tabla que se mostrará tendrá las mismas unidades que en el otro conmutador para garantizar coherencia.

Esto anterior indica que cada fabricante establece determinadas especificaciones, sobre todo en el plano de control, pero el funcionamiento lógico se mantiene.

Figura 167. Pipeline del router con QoS adaptado a Tofino

Para probar el correcto funcionamiento de este servicio, se empleará la siguiente topología:

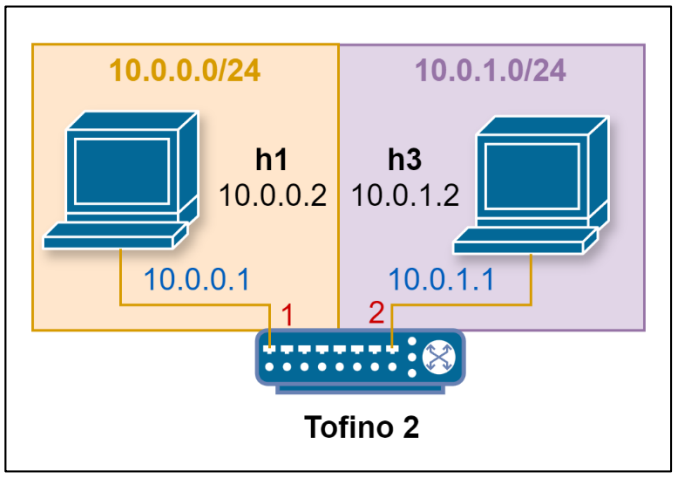


Figura 168. Topología del router con QoS en Tofino

Los valores de la tabla de encaminamiento y de los medidores son:

forward_IPv4		
Key	*Action*	Parameters
10.0.0.2	forward	08:00:00:00:00:01 1
10.0.1.2	forward	08:00:00:00:00:02 2
missed	*drop*	

flow_meter						
Key	*Action*	*CIR (Bytes per second)*	*Cburst (Bytes)*	*PIR (Bytes per second)*	*Pburst (Bytes)*	*Allow yellow*
1	apply_meter	125	1500	250	3000	true
2	apply_meter	125	1500	250	3000	false

default meter				
Index	*CIR (Bytes per second)*	*Cburst (Bytes)*	*PIR (Bytes per second)*	*Pburst (Bytes)*
0	125	1500	250	3000

Figura 169. Estado de las tablas del router con QoS en Tofino

En este caso, emplearemos la misma prueba de la versión de BMv2, para así poder validar el comportamiento del programa tras ejecutar el medidor.

PRUEBA DE MEDICIÓN

Distinguimos los tres casos posibles: paquete sin flujo registrado, paquete con flujo 1, y paquete con flujo 2.

1. COMPORTAMIENTO DE UN PAQUETE SIN FLUJO

10.0.0.2	10.0.1.2	1 UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	1 UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1 UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1 UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	1 UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1 UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0 UDP	200	64 12345 → 54321

Figura 170. Captura de paquetes sin flujo en Tofino

En la zona resaltada en naranja se muestran los paquetes que fueron marcados con verde, y por lo tanto pudieron circular con normalidad por el conmutador, frente al resto que fueron descartados. Podemos concluir que el medidor por defecto está **validado** en Tofino.

2. COMPORTAMIENTO DE UN PAQUETE CON FLUJO 1

10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1	UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1	UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1	UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1	UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1	UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1	UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	1	UDP	200	63 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321
10.0.0.2	10.0.1.2	0	UDP	200	64 12345 → 54321

Figura 171. Captura de paquetes con flujo 1 de QoS en Tofino

```
Flow Header
  Flow ID: 1
  Ether protocol: 0x0800
Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.1.2
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
› Differentiated Services Field: 0x03 (DSCP: CS0, ECN: CE)
```

Figura 172. Paquete del flujo 1 marcado como amarillo en Tofino

En este caso, obtenemos el mismo escenario que en las pruebas ejecutadas en BM2, destacando tres zonas bien diferenciadas, la de los paquetes marcados con verde, que pudieron transitar por el conmutador (en este caso uno), los que son marcados con amarillo, que muestran el valor ECN en la cabecera IPv4, y los que son marcados con rojo, que son descartados en el dispositivo.

Este caso también queda **validado**.

3. COMPORTAMIENTO DE UN PAQUETE CON FLUJO 2

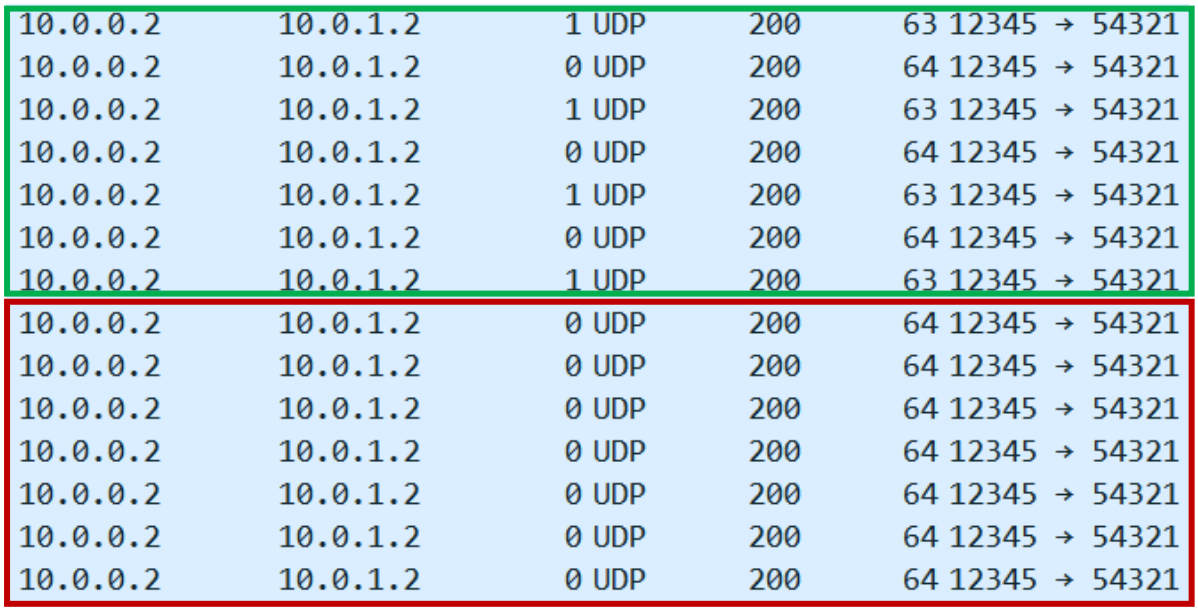


Figura 173. Captura de paquetes con flujo 2 de QoS en Tofino

En este último caso no hay marcado de ECN, sólo hay dos situaciones:

- Zona verde: Incluye a los paquetes que obtuvieron el color verde al ejecutar el medidor.
- Zona roja: Incluye a los paquetes que obtuvieron los colores amarillo o rojo, al ejecutar el medidor, que son descartados.

Con esta última prueba, queda **validado** el comportamiento del flujo 2 en Tofino, y por lo tanto también esta adaptación del servicio.

9

CONCLUSIONES Y LÍNEAS FUTURAS

Al finalizar un proyecto de gran intensidad y trabajo, surgen muchos resultados de aprendizaje, y con ellos muchas conclusiones que permiten tomar nuevas perspectivas acerca de la temática estudiada. Tras realizar un profundo análisis, que detecte las aportaciones más importantes que surgen tras haber enfrentado una temática innovadora y estrechamente ligada al área de la Ingeniería Telemática, podría valorarse en primer lugar la apertura de mente que ha supuesto enfrentarse a nuevos conceptos que están a la vanguardia, y, en segundo lugar, aplicar correctamente la forma correcta de trabajar interiorizada durante el desarrollo de los estudios.

Al volver a consultar el anteproyecto inicial, se ha podido observar el cambio de visión experimentado frente a un mundo que era nuevo por descubrir, donde se planteaba la existencia de un lenguaje que podía dirigir el comportamiento de una red, y multitud de conceptos. El empleo de una metodología bien estructurada, el conocimiento y detección de las limitaciones que existían a priori, y una documentación y formación relajada y a la vez de calidad, ha permitido tener un trabajo ordenado con una organización eficiente.

Si nos centramos en el resultado obtenido, es sin lugar a duda un éxito de ejecución, porque ha permitido cumplir los objetivos iniciales que se querían cubrir, reforzarlos porque el dominio de la tecnología era incremental y finalmente ampliarlos, debido a que han surgido nuevos horizontes no contemplados inicialmente.

El producto final obtenido, reúne las buenas prácticas de la programación en general, constituye una gran profundización en conocimientos telemáticos, con una visión fiel de la infraestructura de dispositivos, y especialmente conmutadores, que componen las redes que se utilizan a diario, que cada vez cobran más importancia, y tienen una gran repercusión en tareas diarias, que podrían ser paralizadas si no existe un empleo razonable de la infraestructura existente. En este escenario se enmarca a la perfección el servicio desarrollado que aplica políticas de calidad de servicio, que ha demandado conocer adecuadamente los distintos parámetros de ancho de banda disponible, y realizar una asignación eficiente de la capacidad disponible en transmisiones, para priorizar tráfico de red crítico, que puede repercutir por ejemplo en la calidad de comunicaciones de una determinada situación excepcional que así lo requiera, disminuyendo las posibles interferencias de paquetes menos prioritarios.

El conocimiento intenso de las tecnologías que operan a un nivel físico, como pueden ser los cables de conexión al conmutador, y el modelo de procesamiento de este, presenta un valor incuestionable, en cuanto al entendimiento de lo que sucede en una determinada comunicación, y que es verdaderamente importante para fomentar un uso de calidad de la infraestructura desde los programas y aplicaciones usuarios de sus conexiones.

Además, este trabajo ha servido para explorar nuevos horizontes de posibilidades que presenta el desarrollo de una carrera tecnológica. Particularmente, se ha podido conocer con más cercanía las labores de un grupo de investigación de la universidad, que me ha permitido observar el estado actual de las investigaciones que se están realizando en esta área, y que ha sido tomado como una experiencia formativa adicional, que permite contextualizar mejor el trabajo realizado, y aprender buenas prácticas para entender nuevos conceptos que permiten la actualización constante del ingeniero en cuanto a su formación.

Siguiendo en la misma línea, todo esto ha permitido validar el correcto funcionamiento de una organización eficaz de un desarrollo dirigido por modelos, que ha permitido documentar y transmitir los comportamientos que debía de tener un conmutador para ofrecer un servicio en su entorno, y finalmente tener desarrollos que son trasladables a un ámbito real.

Si observamos esto desde una perspectiva de ahorro de recursos, la programación SDN puede suponer un ahorro bastante significativo en las inversiones que se llevan a cabo para suplir a la red de nuevas funcionalidades, que a menudo son costosas y no respetan el medio ambiente. Disponer de dispositivos programables con unidades dedicadas a la función que se le asigna y que no queden rápidamente obsoletos puede tener una repercusión positiva en un mejor aprovechamiento de tarjetas de red y de la circuitería existente en el dispositivo, lo cual deriva en una práctica que favorece el fomento de la conocida como *economía verde*, que promulga precisamente un aprovechamiento de los recursos que se emplean. Por todo esto, esta tecnología tiene una repercusión positiva en ese ámbito.

En definitiva, explorar y explotar las posibilidades ofrecidas por P4 es una oportunidad de dar visibilidad a una tecnología prometedora, y que supondrá un cambio de paradigma de red en un futuro cercano, lo cual es especialmente útil para un incremento en la velocidad de adopción de nuevas tecnologías software que requieran de una red más flexible y con funciones avanzadas fácilmente configurables, y que no requieren desplazamientos innecesarios para gestionar un determinado dispositivo, ya que propone la idea de cambio de configuración remoto.

Cuando se ha realizado un correcto balance de lo que se ha realizado en el trabajo, que repercute en una exploración profunda del plano de datos y el plano de control introducido por SDN, surgen multitud de posibilidades y ampliaciones que podrían llevarse a cabo con la adquisición de los conocimientos adquiridos. En ese sentido, surgen diferentes **líneas de trabajo** que pueden ser consideradas oportunidades de ampliación. Vamos a expandir las que se han considerado más prometedoras en base a todo lo realizado:

1. **Desarrollo de controladores con comportamiento dinámico**: En una primera toma de contacto con la estructura del conmutador, y una posterior profundización de las funciones desempeñadas por el plano de datos, se ha entendido la necesidad derivada de un control y operaciones robustas y seguras para llevarlo a cabo. Dadas las características del proyecto, que estaba centrado principalmente en el desarrollo con P4, los controles han presentado un comportamiento estático que suple las funciones básicas de arbitraje. Tomando beneficio de las nuevas técnicas

de inteligencia artificial, que permiten detectar diferentes patrones en paquetes que circulan por la red, puede ser una excelente aplicación tomar beneficio de ello para mejorar la seguridad en las redes con firewalls avanzados que inhiban un determinado tráfico malintencionado. De esto se extrae la necesidad de presencia de controladores responsivos, que se apoyen en operaciones básicas de control para comunicarse con el plano de datos, pero que presente a su vez un procesamiento adicional en su ámbito.

2. **Mejora de técnicas de ciberseguridad con P4**: La programación de dispositivos SDN con P4 permite definir el comportamiento del procesamiento de conmutadores intermedios, además de otros que también lo adoptan. El conocimiento y dominio de la infraestructura existente presenta un carácter diferenciador, y si se combinan conceptos de seguridad con funcionalidades avanzadas implementadas en el plano de datos con apoyo de un procesamiento complejo en el plano de control, permitirá la adopción de algoritmos de seguridad que permita ofrecer un entorno fiable y seguro. En ese sentido pueden ser aplicadas técnicas criptográficas que garanticen comunicaciones confidenciales.
3. **Desarrollo de un componente intermedio para un modelo de compilación P4 universal**: Una de las conclusiones más importantes que se obtienen tras trabajar en el proyecto, es la necesidad del empleo de una arquitectura de referencia, como lo es *V1model*, que permite aislar el algoritmo lógico de tratamiento de paquetes de la naturaleza física del dispositivo con el que se trabaja. El análisis de eficiencia algorítmica es ya suficientemente complejo y requiere de focalización en él, por lo que cualquier configuración relacionada con el dispositivo requiere de tiempo adicional que puede desviarnos del objetivo principal. Por esa problemática y por la complejidad existente en una determinada arquitectura lógica, se ha pensado en el desarrollo de una extensión del compilador p4c, que fuera un mid-end existente entre el compilador de referencia un backend existente de las extensiones insertadas por los distintos fabricantes, por ejemplo, Tofino de Intel. Esto permitiría mantener compiladores especializados según la plataforma, pero con programas P4 escritos en la arquitectura de referencia [30]. Sería una oportunidad para

garantizar la universalidad de un programa, en una arquitectura que permite profundizar en su complejidad algorítmica, y al mismo tiempo, delegar en el mencionado componente objeto de desarrollo, las adaptaciones del código al ejecutable final especificado por el fabricante. En esencia, es crear un software que tienda un puente lógico entre la arquitectura de referencia de P4, pero con compilación y optimizaciones subyacentes propias de la plataforma en la que se programa, y proporcionar una plantilla de trabajo para garantizar la universalidad total de código P4. De esta forma, las optimizaciones de parámetros de la arquitectura pueden hacerse de forma más eficiente, y ofrecer un marco de trabajo común que garantice una interfaz común de programación, pero optimizando al máximo el uso de la arquitectura. Esto sólo sería aplicable en conmutadores, y no en otros dispositivos programables con P4.

4. **Programación de otros dispositivos de red con P4**: La programación con este lenguaje no se reduce a los conmutadores, y permite ser empleado en otros dispositivos conectados a la red, como es el caso de las *tarjetas de red*. En ese contexto ya se está trabajando en una arquitectura lógica [32] que permita programarlas, y aplicar funciones avanzadas para un determinado equipo, como puede ser NAT para la traducción de determinadas direcciones. Aplicar el conocimiento del lenguaje puede permitir trabajar en otros ámbitos y mejorar soluciones existentes, que pueden ayudar a interactuar con los servicios de conmutadores que ya se tienen desarrollados.

Para concluir con el desarrollo de este último capítulo, hay que indicar la satisfacción existente por trabajar en algo novedoso, que además de haber aportado resultados muy importantes en el contexto de programación de redes, ha permitido la especialización en un área particular, y poner en práctica para afianzar las diferentes competencias adquiridas a lo largo de los estudios.

Finalmente, y como evidencia de profundización en distintas áreas de conocimiento que han sido abordadas a lo largo del grado, se incluye en una tabla de la siguiente página aquellas que se han considerado destacables.

ÁREAS TECNOLÓGICAS DEL GRADO CUBIERTAS
• **Arquitectura de computadores**: Un conmutador programable físico presenta los mismos elementos hardware estudiados en este módulo. • **Sistemas Operativos**: Para poder emplear emisores de tráfico, configurar interfaces y trabajar con un controlador remoto accesible, es necesario dominar adecuadamente los fundamentos de un sistema tipo Unix (*Linux*). • **Redes informáticas**: Se ha profundizado en el significado de los protocolos incluidos en *TCP/IP*, su definición estandarizada y su utilización en una red. Para ello también ha sido necesario comprender e interiorizar la infraestructura física presente, el establecimiento de enlaces, y la declaración de espacios de direccionamiento. • **Desarrollo con APIS**: Trabajar con *GRPC* Y la definición de *P4Runtime*, ha permitido estudiar un nuevo formato de conexiones entre clientes y servidores. • **Estudio de compiladores**: Entender la estructura interna de un compilador, como es P4C, permite estudiar otros modelos de compilación orientados a un dispositivo particular. • **Administración de bases de datos**: El trabajo que se ha realizado con *InfluxDB* ha permitido conocer un nuevo paradigma de almacenamiento de datos, orientado específicamente a series temporales. • **Software basado en modelos**: Idear los servicios de red que han sido expuestos ha sido esencial para documentar detalladamente su comportamiento. • **Programación orientada a objetos**: El uso de buenas prácticas de encapsulamiento de métodos, y reducción de acoplamiento de clases ha permitido obtener un conector debidamente jerarquizado. • **Programación concurrente**: Se ha profundizado en la gestión de diferentes hebras de ejecución, para evitar conflictos entre ellas, y aportar un conector con funcionalidades sólidas.

Tabla 20. Áreas tecnológicas del grado cubiertas

Referencias

[1] The P4 Language Consortium, «P416 Language Specification,» 11 octubre 2024. [En línea]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html. [Último acceso: 5 mayo 2025].

[2] F. A. Marín, «Introducción a la telemática. Redes y servicios telemáticos,» de *Teleformación: Diseño para todos*, Universitat de València, 1999.

[3] P4 Language, «BEHAVIORAL MODEL (bmv2),» [En línea]. Available: https://github.com/p4lang/behavioral-model. [Último acceso: agosto 2025].

[4] IBM, «concepts tcpip,» 5 abril 2023. [En línea]. Available: https://www.ibm.com/docs/en/zos/3.1.0?topic=concepts-tcpip. [Último acceso: agosto 2025].

[5] IEEE, «IEEE 802.3 - Standard for Ethernet,» [En línea]. Available: https://ieeexplore.ieee.org/document/9844436. [Último acceso: agosto 2025].

[6] J. Postel, «RFC 791 - Internet Protocol,» septiembre 1981. [En línea]. Available: https://www.rfc-editor.org/info/rfc791. [Último acceso: agosto 2025].

[7] W. Eddy, «RFC 9293 - Transmission Control Protocol (TCP),» agosto 2022. [En línea]. Available: https://www.rfc-editor.org/info/rfc9293. [Último acceso: agosto 2025].

[8] J. Postel, «RFC 768 - User Datagram Protocol,» agosto 1980. [En línea]. Available: https://www.rfc-editor.org/info/rfc768. [Último acceso: agosto 2025].

[9] P. Göransson y C. Black, Software Defined Networks. A Comprehensive Approach, Elsevier, 2014.

[10] Open Networking Foundation, «OpenFlow Switch Specification,» 26 marzo 2015. [En línea]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. [Último acceso: agosto 2025].

[11] Open Networking Foundation, «NG-SDN,» [En línea]. Available: https://opennetworking.org/ng-sdn/. [Último acceso: agosto 2025].

[12] D. D. G. G. M. I. N. M. J. R. C. S. D. T. A. V. G. V. D. W. Pat Bosshart, «P4: Programming Protocol-Independent,» 2014.

[13] P4, «P4 ORG,» [En línea]. Available: https://p4.org/. [Último acceso: agosto 2025].

[14] P4 Language, «P4C - P4_16 reference compiler,» [En línea]. Available: https://github.com/p4lang/p4c. [Último acceso: agosto 2025].

[15] The P4.org API Working Group, «P4Runtime Specification,» 21 octubre 2024. [En línea]. Available: https://p4.org/wp-content/uploads/2024/10/P4Runtime-Spec-v1.4.1.html. [Último acceso: agosto 2025].

[16] «Protocol Buffers,» [En línea]. Available: https://protobuf.dev/. [Último acceso: agosto 2025].

[17] «GRPC,» [En línea]. Available: https://grpc.io/. [Último acceso: agosto 2025].

[18] «Draw.io,» [En línea]. Available: https://www.drawio.com/. [Último acceso: agosto 2025].

[19] «Github,» [En línea]. Available: https://github.com/. [Último acceso: agosto 2025].

[20] Influx data, «InfluxDB,» [En línea]. Available: https://www.influxdata.com/. [Último acceso: agosto 2025].

[21] «Lua, the programming language,» [En línea]. Available: https://www.lua.org/. [Último acceso: agosto 2025].

[22] GNU, «GNU make,» 26 febrero 2023. [En línea]. Available: https://www.gnu.org/software/make/manual/make.html. [Último acceso: agosto 2025].

[23] «Mininet,» [En línea]. Available: https://mininet.org/. [Último acceso: agosto 2025].

[24] Mobatek, «MobaXterm,» [En línea]. Available: https://mobaxterm.mobatek.net/. [Último acceso: agosto 2025].

[25] «Python,» [En línea]. Available: https://www.python.org/. [Último acceso: agosto 2025].

[26] «Trello,» [En línea]. Available: https://trello.com/es. [Último acceso: agosto 2025].

[27] Oracle, «VirtualBox,» [En línea]. Available: https://www.virtualbox.org/. [Último acceso: agosto 2025].

[28] Microsoft, «Visual Studio Code,» [En línea]. Available: https://code.visualstudio.com/. [Último acceso: agosto 2025].

[29] Wireshark, [En línea]. Available: https://www.wireshark.org/. [Último acceso: agosto 2025].

[30] «XTERM,» [En línea]. Available: https://xterm.dev/. [Último acceso: agosto 2025].

[31] Barefoot Networks, «Open Tofino,» [En línea]. Available: https://github.com/barefootnetworks/Open-Tofino. [Último acceso: agosto 2025].

[32] OMG, Unified Modeling Language, 2017.

[33] «RFC 1918 - Private Address Space,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc1918#section-3. [Último acceso: agosto 2025].

[34] J. Heinanen y R. Guerin, «RFC 2698 - A Two Rate Three Color Marker,» septiembre 1999. [En línea]. Available: https://www.rfc-editor.org/info/rfc2698. [Último acceso: agosto 2025].

[35] Refactoring Guru, «Strategy Pattern,» [En línea]. Available: https://refactoring.guru/design-patterns/strategy. [Último acceso: mayo 2025].

[36] Python, «Asyncio - Asynchronous I/O AC,» [En línea]. Available: https://docs.python.org/es/3/library/asyncio.html. [Último acceso: agosto 2025].

[37] P4 Language, «v1model,» [En línea]. Available: https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4. [Último acceso: agosto 2025].

[38] J. Arkko, E. M. Cotton, L. Vegoda y I. , «RFC 5737 - IPv4 Address Blocks Reserved for Documentation,» enero 2010. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc5737. [Último acceso: agosto 2025].

[39] The P4.org Architecture Working Group, «P416 Portable Switch Architecture (PSA),» 22 diciembre 2022. [En línea]. Available: https://p4.org/p4-spec/docs/PSA-v1.2.html. [Último acceso: agosto 2025].

[40] Intel, «Intel® Tofino™ 2,» [En línea]. Available: https://www.intel.la/content/www/xl/es/products/details/network-io/intelligent-fabric-processors/tofino-2.html. [Último acceso: agosto 2025].

[41] P4 Language, «Open P4Studio,» [En línea]. Available: https://github.com/p4lang/open-p4studio. [Último acceso: agosto 2025].

[42] The P4.org Architecture Working Group, «P4 Portable NIC Architecture (PNA),» 22 diciembre 2022. [En línea]. Available: https://p4.org/p4-spec/docs/PNA-v0.7.html. [Último acceso: agosto 2025].

[43] R. Braden, D. Borman y C. Partridge, «RFC 1071 - Computing the Internet Checksum,» septiembre 1988. [En línea]. Available: https://www.rfc-editor.org/info/rfc1071. [Último acceso: agosto 2025].

[44] A. Rijsinghani, «RFC 1624 - Computation of the Internet Checksum via Incremental Update,» mayo 1994. [En línea]. Available: https://www.rfc-editor.org/info/rfc1624. [Último acceso: agosto 2025].

[45] IANA, «Números de procolo para IEEE 802.3,» [En línea]. Available: https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml. [Último acceso: agosto 2025].

[46] IANA, «Números de protocolo para IPv4,» [En línea]. Available: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml. [Último acceso: agosto 2025].

[47] Baeldung, «network-interface-configure,» [En línea]. Available: https://www.baeldung.com/linux/network-interface-configure. [Último acceso: agosto 2025].

[48] IBM, «Protocol tcpip network interfaces,» [En línea]. Available: https://www.ibm.com/docs/en/aix/7.1.0?topic=protocol-tcpip-network-interfaces. [Último acceso: agosto 2025].

Apéndice A:

Estructura del proyecto

El **proyecto desarrollado** para englobar cada uno de los módulos implementados en este trabajo, presenta una estructura modularizada y jerarquizada, que favorece la reutilización de componentes comunes a cada uno de los programas presentes, la facilidad de ejecución y una distribución lógica ordenada. Todo ello permite diferenciar claramente las zonas de trabajo relativas a las dos plataformas de switches que han sido empleados; manteniendo al mismo tiempo aislamiento e integración de definiciones comunes a ambas secciones. Además, constituye una plantilla fácilmente reutilizable.

Los principales objetivos del presente apéndice son dejar constancia de aquellos detalles importantes que justifiquen la organización de archivos seguida, y documentar adecuadamente el procedimiento a seguir para poder instalar y emplear el proyecto. Además, para aportar claridad, todos los niveles del proyecto presentan descripciones detalladas de lo que contienen en un archivo README, y los artefactos software con su código fuente debidamente identificado y documentado, de tal manera que se facilite su consulta y la navegación por el mismo.

A.1: DESCRIPCIÓN DE LA ESTRUCTURA

Para comenzar adecuadamente, vamos a profundizar en los elementos contenidos en la jerarquía de directorios. Nos vamos a centrar en todo lo que se contiene dentro del directorio “src”, que es la raíz del proyecto desarrollado.

En la próxima página, se puede visualizar una representación gráfica detallada de los directorios y ficheros presentes, de tal forma que iremos detallando cada uno de los niveles, a la vez que se presenta el diagrama, para una visualización rápida y detallada.

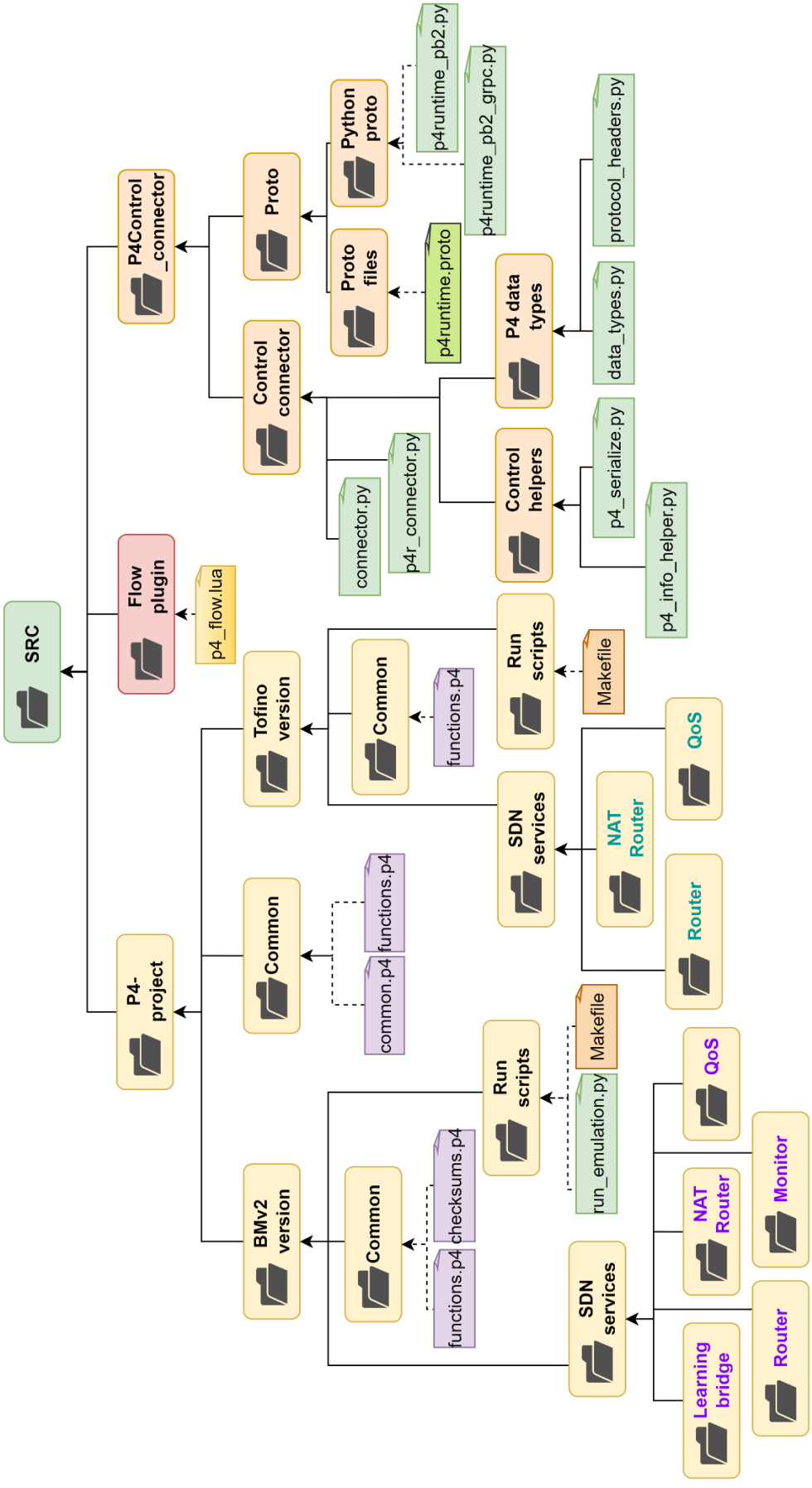


Figura 174. Jerarquía general del proyecto

La división lógica de la jerarquía viene motivada por los artefactos software que se incluyen, y que se dividen en los siguientes niveles:

- **Proyecto P4 (P4-project)**: Sección que contiene el código fuente P4, donde se incluyen por una parte algunos directorios con ficheros con definiciones y funciones comunes, y por otra las secciones relativas a los dos conmutadores empleados, BMv2 y Tofino. Para cada uno de los dispositivos, y como es apreciable en el diagrama, se tienen definiciones y funciones específicas de su respectiva arquitectura, scripts que automatizan el lanzamiento y ejecución de servicios de red contenidos, y directorios que organizan los distintos servicios contenidos, con el propio código fuente de los planos de datos implementados y scripts de control.

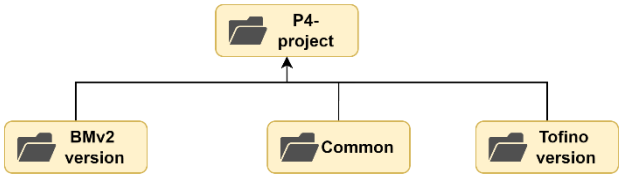


Figura 175. Sección del proyecto titulada "P4-project"

- **Conector de control (P4Control_connector)**: Sección que contiene el código Python que implementa un conector que actúa como cliente *P4Runtime*. Se distinguen dos divisiones importantes: una que contiene el proyecto del conector en sí, y otra que presenta las definiciones proto que son empleadas por el mencionado proyecto.

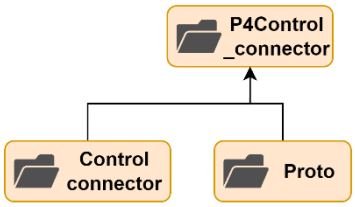


Figura 176. Sección del proyecto titulada "P4Control_connector"

- **Flow Plugin**: Directorio simple que contiene el archivo fuente relativo a una extensión para Wireshark desarrollada en el lenguaje *Lua*. Su principal objetivo es permitir la correcta visualización de la cabecera *Flow,* que ha sido definida en el servicio que aplica políticas de calidad de servicio. Además, contiene unas instrucciones detalladas para su correcta integración en el análisis de paquetes.

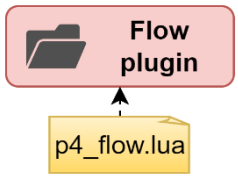


Figura 177. Sección del proyecto titulada "Flow plugin"

A partir de esos tres principales directorios, se contienen el resto de los archivos y subdirectorios que permite mantener una clasificación clara.

A.2: INSTALACIÓN DEL PROYECTO

Los artefactos software desarrollados en este proyecto (programas P4, controladores, pruebas de tráfico…) que conforman el proyecto técnico, se encuentran localizados en un repositorio. Tener acceso al mismo es necesario para descargarlo en un determinado equipo.

En la siguiente tabla se recogen los requisitos necesarios para poder ejecutar las secciones del proyecto:

REQUISITOS PARA LA INSTALACIÓN
• Tener una máquina con distribución Linux (Ubuntu, Lubuntu, Debian…). • Emulador de redes **Mininet** [9]. • Emulador de terminal **Xterm** [18]. • Intérprete del lenguaje **Python** [12] (con librerías especificadas en *requirements.txt*, archivo presente en el directorio raíz del repositorio). • Switch software **BMv2** [3]. • Compilador oficial de P4 **p4c** [11] (incluye uno extendido para BMv2). • Base de datos de series temporales **InfluxDB** [6]. • Analizador de tráfico de red **Wireshark** [13]. • Plug-in para leer cabeceras *Flow* en Wireshark (presente en el repositorio con pasos para su instalación).

Tabla 21. Requisitos de instalación

Se recomienda encarecidamente recurrir a la virtualización, mediante la creación de una máquina virtual que contenga todo lo anterior, pues al tratar con configuraciones de redes y requisitos muy específicos, el tener un entorno aislado nos proporciona una capa adicional de seguridad. En cuanto al empleo del lenguaje Python, lo más fiable será mantener un entorno virtual creado específicamente para ser empleado en el proyecto.

Cuando se hayan instalado todos los requisitos presentes en la tabla, estaremos en disposición de instalar debidamente el proyecto para su utilización. Seguiremos los pasos especificados en la siguiente página:

PASOS PARA UNA CORRECTA INSTALACIÓN

1) Se **clonará/copiará** el directorio raíz del repositorio en la ubicación deseada.

2) Una vez asentado el proyecto, entraremos en el **primer nivel** del directorio y allí encontraremos el directorio “src”, entre otros elementos que se encuentran en el mismo nivel.

Figura 178. Contenido del primer nivel del repositorio

3) Debemos garantizar que las todas las **librerías** especificadas en el archivo *requirements.txt* estén debidamente instaladas, con sus respectivas versiones.

```
protobuf==5.29.4
grpcio==1.71.0
grpcio-tools==1.71.0
...
```

Figura 179. Formato de un archivo de requisitos de Python

4) Seguidamente, **instalaremos de forma local librerías Python** definidas en *P4Control_connector*, donde la primera contiene los archivos “proto” compilados para el lenguaje Python, que definen el servicio de GRPC *P4Runtime*, mientras que la segunda alberga el conector que implementa una interfaz de control:

a. **Definiciones proto**: Nos desplazaremos al directorio del proyecto: src/p4Control_connector/proto/python_proto_files (ver la Figura 174), y allí abriremos un terminal para insertar el comando `make install`. Esto desencadenará la instalación de la librería. Nótese que el comando *make* tiene definida la regla de instalación en un archivo **Makefile** [8], además de una variable denominada *PYTHON*, que permite especificar el intérprete específico que se desea emplear.

b. **Conector**: Nos desplazaremos ahora a src/p4Control_connector, y ejecutaremos el mismo comando, `make install`.

Es importante hacerlo en este orden, debido a que la segunda librería depende de la primera.

5) Cada servicio contenido en la sección del conmutador BMv2, requiere de variables de entorno que necesitan ser consumidas por los controladores. Ello viene justificado de la necesidad de indicar la ruta absoluta del archivo ejecutable, y del archivo P4info, que indica la codificación de parámetros empleados en la comunicación de control. Habrá que definir por este motivo un archivo denominado *.env*, con las variables que se muestran:

```
P4_INFO_PATH = /TFG/TFG-DesarrolloSDN-P4/src/P4-
project/BMv2-version/SDN services/Learning-
bridge/build/bridging.p4.p4info.txtpb

JSON_PATH = /TFG/TFG-DesarrolloSDN-P4/src/P4-
project/BMv2-version/SDN services/Learning-
bridge/build/bridging.json
```

Figura 180. Variables de entorno para el controlador

Este ejemplo se corresponde con el Conmutador con aprendizaje (*Learning-bridge*), y deberá ser definido para los cinco servicios contenidos.

6) Una vez completados los pasos anteriores de forma exitosa, hemos instalado adecuadamente el proyecto y está en disposición de ser utilizado.

A.3: GUÍA DE EJECUCIÓN

Instalado el proyecto, es importante conocer cómo ejecutar un determinado servicio en el conmutador virtual, lanzar un controlador que gestione los recursos presentes en el plano de datos, y poder emitir tráfico de prueba. Será explicado de forma exclusiva para el conmutador *BMv2* [3], dado que todo el software de gestión del dispositivo Tofino es confidencial.

Antes de comenzar con la explicación paso a paso, es preciso incidir en el modelo de ejecución que ha sido diseñado para el presente conmutador. Para ello, debemos

navegar al directorio src/P4-project/BMv2-version/run_scripts. Allí dentro encontraremos dos archivos: *run_emulation.py*, que ejecuta una determinada topología en el emulador mininet, y makefile que define reglas (targets) Makefile para su reutilización en todos los servicios. Son destacables *build*, *run* y *clean*, que permiten la compilación de un programa P4, la ejecución de una topología y la limpieza de los artefactos compilados respectivamente. Todo ello definiendo una estructura de directorios creadas al realizar la labor de compilación, que permite tener una organización limpia en los directorios de cada servicio.

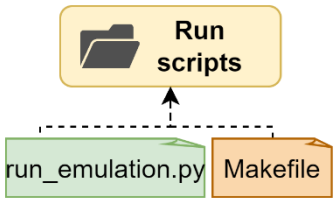


Figura 181. Contenido del directorio "Run scripts"

Emplearemos el servicio “Conmutador con aprendizaje” para entender con claridad cómo activar de forma adecuada cada uno de los componentes. Para ello nos desplazaremos al directorio src/P4-project/BMv2-version/SDN_services/Learning-bridge y observaremos con detenimiento los archivos que se encuentran dentro:

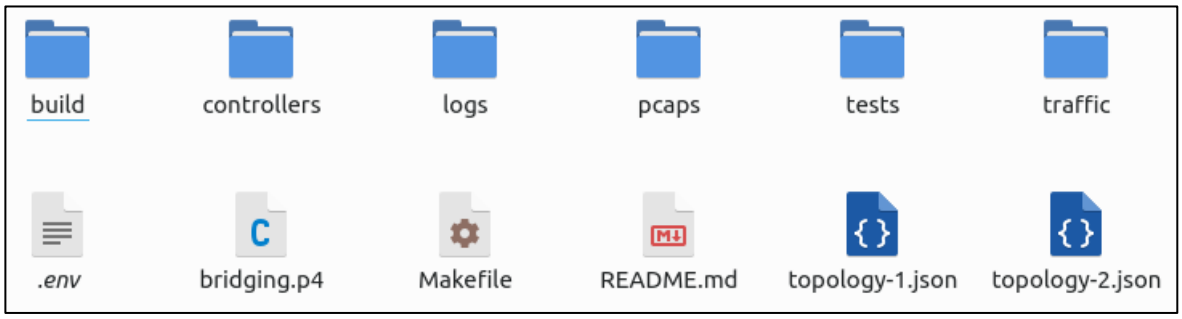


Figura 182. Contenido del directorio "Learning-bridge"

Se destacan los siguientes elementos:

- Archivos de **topologías:** Contienen la especificación JSON de los diferentes nodos (hosts y conmutadores) junto a identificadores, y enlaces que componen la red, con una asignación de direcciones MAC e IPv4.
- Directorio **controllers**: Contiene dos controladores para cada una de las topologías (uno para cada una).
- Archivo **bridging.p4**: Contiene el código fuente P4 del plano de datos del conmutador con aprendizaje.
- Archivo **Makefile**: Contiene una extensión del archivo común citado anteriormente, y presente en la Figura 181. En él se indican valores de

variables del ámbito del comando *make* y sus reglas, tales como el programa a compilar o la topología ejecutada por defecto.

Cabe destacar que esta estructura ha sido seguida en cada uno de los servicios, para garantizar una homogeneidad en la ejecución. Es importante entender contenido del archivo Makefile:

```
BMV2_SWITCH_EXE = simple_switch_grpc
TOPO = topology-1.json # Default topology.
DEFAULT_PROG = bridging.p4
include ../../run_scripts/Makefile

topo-2:
    make TOPO=topology-2.json
control1:
    python controllers/control_t1.py
control2:
    python controllers/control_t2.py
test-idle:
    sudo python tests/test_idle.py
test1:
    sudo python tests/test_t1.py
test2:
    sudo python tests/test_t2.py
```

Figura 183. Contenido del archivo makefile del conmutador con aprendizaje

PASOS PARA LA EJECUCIÓN

1) Teniendo en cuenta esto, debemos abrir un terminal en el directorio del servicio, e iniciaremos la emulación con el comando:

```
bash$ make
```

2) Ello iniciará el script de lanzamiento de Mininet con la topología por defecto, que está especificada en la variable `TOPO` del archivo de la Figura 183. Se abrirá un terminal en el que se puede interactuar con cualquier equipo presente en la red emulada. Es importante destacar que es posible emplear el emulador de terminales *Xterm* [18] para acceder a un host (por ejemplo, h1) en esa misma línea de comandos de mininet con el comando:

```
mininet> xterm h1
```

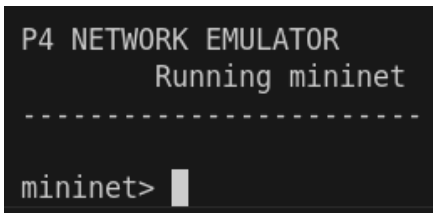


Figura 184. Prompt de Mininet

3) Para especificar la topología, se ha de ejecutar:

```
bash$ make TOPO=<topology>.json
```

Y de forma alternativa para ejecutar la segunda topología:

```
bash$ make topo-2
```

7) Para ejecutar los controladores presentes simplemente habrá que acompañar el comando *make* de la regla deseada en otro terminal diferente (`control1` o `control2`). Algo similar es lo que permite lanzar los distintos tests de tráfico encapsulados con las reglas, que se encargan de ejecutar programas Python que emplean la librería de generación de paquetes de red *Scapy*.

Figura 185. Ejecución simultánea de Mininet y un controlador en dos terminales

Para finalizar, cabe mencionar que se dispone de un controlador con línea de comandos incluido en la instalación del conmutador. Simplemente habrá que invocarlo por un terminal por su nombre: *simple_switch_CLI*, y en él podremos visualizar los diferentes objetos del pipeline del dispositivo, así como realizar manipulaciones sencillas. Es una herramienta útil para monitorizar el estado de las tablas presentes.

Siguiendo todos estos pasos, estamos en disposición de trabajar con los diferentes servicios programados para el conmutador BMv2, así como emplear el entorno que nos garantiza la emulación.

Apéndice B:

La suma de comprobación de Internet

La **suma de comprobación de Internet** (Internet Checksum) o suma de complemento de 16 bits, es un mecanismo de soporte para la verificación de integridad de datos ampliamente adoptado por los protocolos de comunicación. Se emplea principalmente para sumar una selección de campos del protocolo usuario, y transmitir el resultado como un campo adicional en su cabecera, para que el receptor del paquete pueda comprobar su originalidad. Ello permite mejorar la seguridad de las comunicaciones frente a fallos o alteraciones que se pueden producir a lo largo del camino recorrido en las transmisiones, o frente a amenazas malintencionadas que comprometen la fiabilidad de estas.

Todos los protocolos de nivel superior al 2 (enlace) empleados en este trabajo la emplean como mecanismo de redundancia, a excepción de los protocolos personalizados. En cualquier caso, viene siempre especificado en el correspondiente estándar.

Téngase en cuenta que la suma no se encuentra estandarizada, pero existen documentos RFC (Request For Comments) que especifican todo lo relativo a su cálculo y actualización, además de propiedades avanzadas, incluyendo cómo se puede trasladar su procesamiento a unidades hardware. A continuación, profundizaremos sobre los aspectos más importantes y su traslación a un plano de datos programado con P4.

B.1: CÁLCULO Y VERIFICACIÓN

El **cálculo** de la suma presenta un procedimiento sencillo con bajo costo computacional, y viene definido en el documento RFC 1041 [19]. La suma se computa agrupando octetos binarios en palabras de 16 bits, que son sumadas en secuencia, y finalmente se aplica el complemento a uno al resultado obtenido, que no es más que cambiar los bits por su valor contrario (0 por 1 y 1 por 0).

Debido a la división de bloques que se efectúa con un ancho de 16 bits, la **verificación** del cálculo de forma automática es muy sencilla, pues para comprobar que un valor v es veraz, bastará con recalcular la operación, obteniendo v', y si al sumar el resultado con el anterior con el proceso de la suma de comprobación el valor resulta 0 (ancho de 16 bits), entonces se puede afirmar su igualdad, y por tanto su corrección.

B.2: ACTUALIZACIÓN INCREMENTAL

La actualización incremental es una técnica de actualización del valor de la suma, que aporta un incremento de rapidez en el procesamiento. Viene especificada en el documento RFC 1624 [20].

Si se tiene en cuenta la propia definición de la operación de la suma de comprobación, podrá notarse que verifica las propiedades asociativa y conmutativa, y de ello puede extraerse que al restar un valor v, y sumar un nuevo valor v', obtenemos el mismo resultado que si se hubiese calculado de nuevo la operación teniendo en cuenta el segundo valor. Nótese que la sustracción complemento a 16 sigue el mismo procedimiento de complementar a uno el resultado, pero aplicando la operación de resta.

Supongamos que C es el valor anterior de la suma, C' su nuevo valor, v es el antiguo valor del campo actualizado, y v' su nuevo valor, y que S es el valor base de la suma sin tener en cuenta el mismo campo:

$$S = C' - v' = C - v$$
$$C' = C + v' - v = S + v'$$

Figura 186. Propiedad de actualización incremental de una suma de comprobación

Así quedaría expresada la ecuación para verificar la propiedad como declara RFC 1624. Esto presenta gran aplicación en diversos casos donde sólo se alteran algunos campos de la cabecera un determinado protocolo, debido a que recalcular nuevamente la suma desde cero supondría un mayor costo computacional. En el proyecto ha sido empleado en los protocolos *TCP* y *UDP*.

B.3: VERIFICACIÓN Y ACTUALIZACIÓN EN P4

Dado que el lenguaje P4 nos permite programar el flujo de información de determinados bloques programables, ofrecidos por la arquitectura del dispositivo SDN con el que se trabaja, las operaciones lógico-matemáticas avanzadas se delegan en otras unidades, los externs, que pueden ser tanto software como hardware, y cuya interfaz viene declarada por la especificación de la misma arquitectura. En esta sección veremos cómo llevar a cabo la actualización del valor de la suma de comprobación, para los dispositivos BMv2 y Tofino. Esto supone que el formato de la interfaz de operaciones depende del dispositivo con el que se trabaje.

Para comprender cómo invocar el cómputo desde un programa P4, debemos tener en cuenta que este lenguaje está ideado para tratar exclusivamente los campos de la cabecera que se analiza, excluyendo la zona de datos o carga útil, y es por ello por lo que podemos realizar una distinción de dos situaciones según los campos del protocolo involucrados en la suma:

- **Suma de comprobación exclusiva sobre la cabecera**: Es aquella suma que se computa exclusivamente sobre campos de la cabecera del protocolo que la emplea. Es el caso de IPv4.
- **Suma de comprobación sobre la cabecera y carga útil**: Es aquella suma que se computa sobre campos de la cabecera del protocolo que la emplea, los datos del paquete, y opcionalmente sobre campos de cabeceras de niveles inferiores.

Ha de tenerse en cuenta la anterior clasificación para elegir de manera adecuada las operaciones ofrecidas por una determinada arquitectura. A continuación, se incluyen en una tabla ejemplos de cómputo para las arquitecturas V1model [22] y Tofino [23]:

SUMAS DE COMPROBACIÓN EN P4

CÁLCULO EN V1MODEL

```
update_checksum_with_payload(udp.isValid(), // Only updated when
valid
           {
                ipv4.src,
                ipv4.dest,
                8w0,
                ipv4.l4_protocol,
                udp.length,
                udp.src,
                udp.dest,
                udp.length,
                16w0
           },
            udp.checksum, // Checksum field
            HashAlgorithm.csum16); // Checksum algorithm
```

CÁLCULO EN TOFINO

```
Checksum() ipv4_checksum; // Extern
    apply{
        if(ipv4.isValid()){
            ipv4.header_checksum = ipv4_checksum.update(
                {
                    ipv4.version,
                    ipv4.header_length,
                    ipv4.diffserv,
                    ipv4.ecn,
                    ipv4.length,
                    ipv4.id,
                    ipv4.flags,
                    ipv4.frag_offset,
                    ipv4.ttl,
                    ipv4.l4_protocol,
                    ipv4.src,
                    ipv4.dest,
                    options.options
                }
            );
        }
    }
```

Tabla 22. Ejemplos de manipulación de la suma de comprobación en P4

Apéndice C:

Declaración de cabeceras en P4

En este apéndice se presenta una relación de todos los protocolos empleados en este trabajo. Cada uno contiene una representación gráfica de su cabecera, junto al mapeo realizado en el lenguaje P4, para su correcta interpretación. Se encuentran ordenados por su capa de operación en la torre de protocolos TCP/IP [13], y con desempate alfabético para aquellos del mismo nivel.

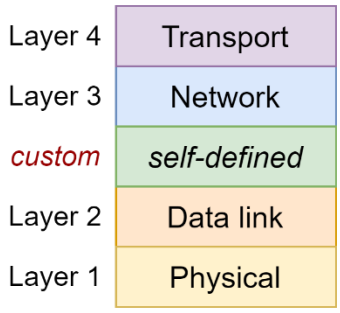


Figura 187.Torre de protocolos TCP/IP con capa personalizada

Una vez definidos todos, se muestran los tipos de datos definidos para referirse a campos de direccionamiento, así como los identificadores de protocolos para las capas *custom*, 3 y 4; que son imprescindibles para la correcta interpretación de paquetes entrantes desde el plano de datos.

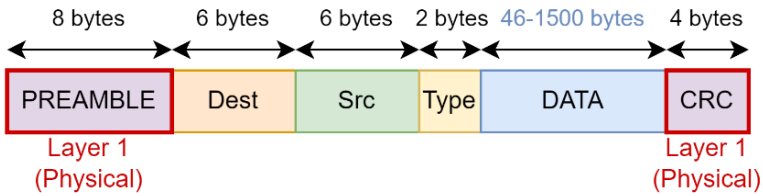
IMPORTANTE

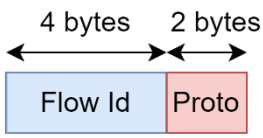
Nótese que la capa denominada “custom” agrupa cabeceras personalizadas y experimentales (no estandarizadas), y que se sitúa entre las capas 2 y 3. Su emplazamiento se ha elegido por simplicidad en la interpretación de los datos que contienen, debido a que constituyen una extensión de información sobre el paquete. Además, en P4 sólo se definen los campos necesarios para el procesamiento, y por ello se omite la carga útil (payload del paquete).

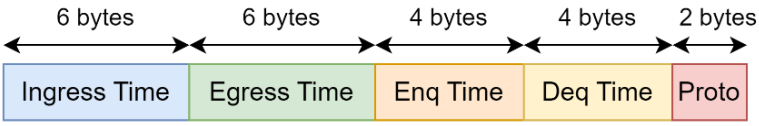
Para declarar el mapeo de una cabecera en P4, se emplea un tipo estructurado con la palabra reservada `header`. Es importante mencionar que, debido al agnosticismo existente respecto a los protocolos reconocidos, existe libertad de agrupación de campos que no son de interés en uno sólo, sin tener consecuencia alguna en el procesamiento del paquete, pues éste es interpretado por su naturaleza a bajo nivel, esto es, una cadena binaria con una determinada longitud.

En la siguiente tabla, el lector podrá apreciar con claridad cada uno de los protocolos empleados, junto a su respectivo número de registro y una representación gráfica de su cabecera, que acompaña a la definición en P4:

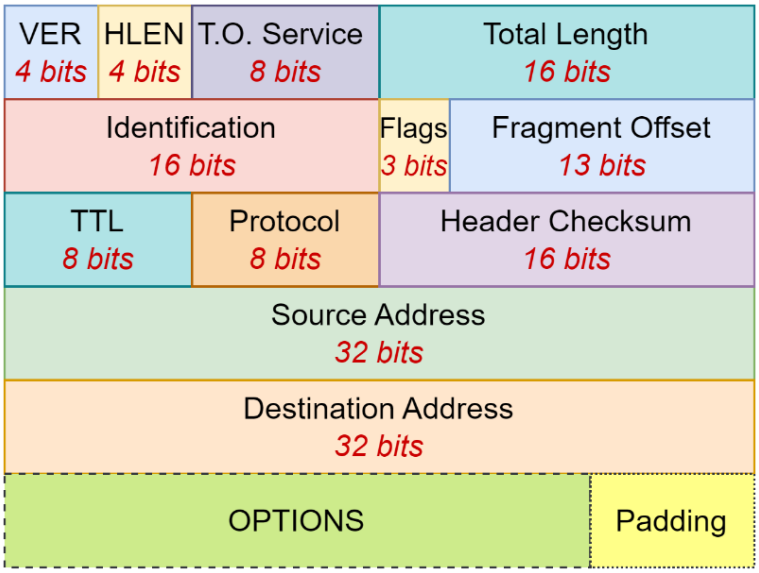
RELACIÓN DE CABECERAS DE PROTOCOLOS

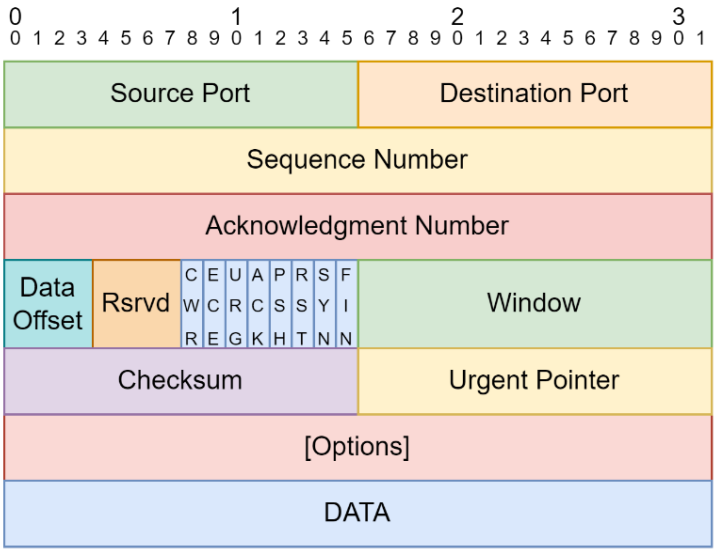
<table>
<tr><th>CAPA</th><th>PROTOCOLO</th><th>MAPEO</th></tr>
<tr><td>2 - Enlace</td><td>ETHERNET
(IEEE 802.3)
[14]</td><td> 

<pre>header ethernet_h { // IEEE 802.3
    // Preamble is not delivered to L2
    MacAddress_t dest;
    MacAddress_t src;
    EtherType_t ether_type; // L3 protocol
}</pre></td></tr>
<tr><td rowspan="2">Custom</td><td>FLOW
(Experimental)</td><td> 

<pre>header flow_h {
    bit<32> flow_id;
    EtherType_t protocol;
}</pre></td></tr>
<tr><td>MONITOR
(Experimental)</td><td> 

<pre>header monitor_h {
    bit<48> ingress_time;
    bit<48> egress_time;
    bit<32> enq_timestamp;
    bit<32> deq_timedelta;
    EtherType_t protocol;
}</pre></td></tr>
</table>

3 - Red	IPv4
	(RFC 791)
	[15]
		
		
		`header ipv4_h { // RFC 791`
		`  bit<4> version; // Set to 4`
		`  bit<4> header_length;`
		`  bit<6> diffserv;`
		`  bit<2> ecn;`
		`  bit<16> length;`
		`  bit<16> id;`
		`  bit<3> flags;`
		`  bit<13> frag_offset;`
		`  bit<8> ttl;`
		`  L4Protocol_t l4_protocol;`
		`  bit<16> header_checksum;`
		`  IpV4Address_t src;`
		`  IpV4Address_t dest;`
		`}`
		
		`header ipv4_options_h { // Makes parser easier`
		`  varbit<((IP_MAX_HEADER_OCTETS_LENGTH - IP_FIXED_HEADER_OCTETS_LENGTH) * 8)> options;`
		`}`

4 – Trans-porte	TCP (RFC 9293) [16]	
		`header tcp_h { // RFC 9293`
		`  L4Port_t src;`
		`  L4Port_t dest;`
		`  bit<32> seq_number;`
		`  bit<32> ack_number;`
		`  bit<4> data_offset;`
		`  bit<4> res;`
		`  bit<8> flags;`
		`  bit<16> window;`
		`  bit<16> checksum;`
		`  bit<16> urgent_ptr;`
		`}`

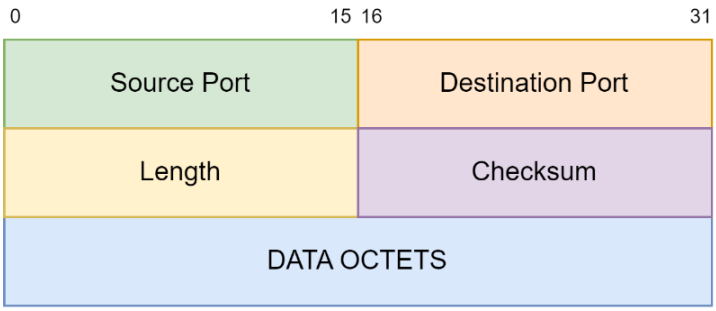
	UDP (RFC 768) [17]	
		`header udp_h { // RFC 768`
		`  L4Port_t src;`
		`  L4Port_t dest;`
		`  bit<16> length;`
		`  bit<16> checksum;`
		`}`

Tabla 23. Relación de mapeo de cabeceras de protocolos

Para facilitar la declaración de direcciones IP y otros campos especiales de las cabeceras que se han abordado, ha sido preciso definir constantes y tipos de datos, mostrados en la siguiente tabla:

DEFINICIÓN DE DATOS	
DEFINICIÓN	DESCRIPCIÓN
`typedef bit<16> EtherType_t;`	Declaración del campo “ether_type” definido en la cabecera *Ethernet*. Permite identificar al protocolo superior.
`typedef bit<8> L4Protocol_t;`	Campo con función similar al anterior, pero perteneciente a la cabecera *IPv4*.

`#define IP_MAX_HEADER_OCTETS_LENGTH 60`
 	
`#define IP_FIXED_HEADER_OCTETS_LENGTH 20`
	Longitudes máxima y mínima de la cabecera *IPv4*. Especialmente útil para calcular la longitud de la cabecera auxiliar variable que mapea el campo de opciones del protocolo.

`typedef bit<48> MacAddress_t;`	Tipo de dato que representa una dirección MAC de *Ethernet*.
`typedef bit<32> IpV4Address_t;`	Tipo de dato que representa una dirección *IPv4*.
`typedef bit<16> L4Port_t;`	Tipo de dato que representa un puerto de la *capa de transporte* (protocolos *TCP* y *UDP*).

Tabla 24. Datos definidos en P4

Una vez definidas ya las cabeceras y los tipos de datos auxiliares de los protocolos empleados en el proyecto, se muestran los identificadores numéricos que permiten detectar el protocolo empleado en la capa superior. Ellos están asignados por la IANA, para los campos “Type” de *Ethernet* [18] y “Protocol” de IPv4 [19], que contiene asignaciones de identificador para protocolos que se encuentran estandarizados y algunos valores libres para uso experimental.

Se muestran a continuación los identificadores de los protocolos que han sido abordados, distinguiendo los que se encuentran estandarizados de los experimentales.

IDENTIFICADORES DE PROTOCOLOS		
PROTOCOLO	IDENTIFICADOR	CAMPOS INVOLUCRADOS
IPv4	0x0800 (hexadecimal)	• "Type" de *Ethernet.* • "Protocol" de *Flow*. • "Protocol" de *Monitor*.
TCP	6 (decimal)	• "Protocol" de *IPv4*.
UDP	17 (decimal)	• "Protocol" de *IPv4*.
FLOW	0x88B6 (hexadecimal) Identificador Experimental	• "Type" de *Ethernet.*
MONITOR	0x88B5 (hexadecimal) Identificador Experimental	• "Type" de *Ethernet.*

Tabla 25. Identificadores de protocolos

Tanto las definiciones de las cabeceras, como las constantes y tipo de datos mostrados en este apéndice, se encuentran ubicados en una librería P4 desarrollada para poder ser importada desde cualquier programa de este lenguaje. De esta forma es posible que sean reutilizables sin importar la arquitectura de conmutador con la que se trabaja.

Apéndice D:

Las interfaces de red

Las interfaces de red son componentes que permiten gestionar la conexión existente entre un dispositivo y una red. Pueden ser tanto físicas (hardware) como lógicas (software), y constituyen un componente esencial para establecer conexiones en redes [20].

En este proyecto se han empleado tanto interfaces físicas, con las conexiones externas del switch hardware Tofino 2, como virtuales, para la gestión de las interfaces creadas en el emulador de redes *Mininet* y para la conexión de los hosts de pruebas que forman parte de las topologías que se han empleado para la validación de servicios de red. Una diferencia clave entre ambos tipos de interfaces es el grado de flexibilidad que presentan, debido a que mientras las físicas presentan un direccionamiento fijo, las virtuales son modificables.

Si nos aproximamos al contexto de la torre de protocolos TCP/IP [21], cuando disponemos de una conexión con un medio físico, habitualmente el direccionamiento del protocolo Ethernet (nivel de enlace) se llevará a cabo en la tarjeta de red del equipo en cuestión, mientas que el resto de direcciones y parámetros estarán gestionados por una interfaz software asociada a la citada tarjeta; todo ello permitiendo una correcta interacción del sistema con el medio externo y favoreciendo siempre una adaptación rápida a posibles modificaciones de características requeridas por protocolos superiores al nivel 2, como son IP, TCP o UDP. En redes emuladas el nivel físico de protocolos también sería emulado por software, presentando en ese caso una interfaz completamente virtual.

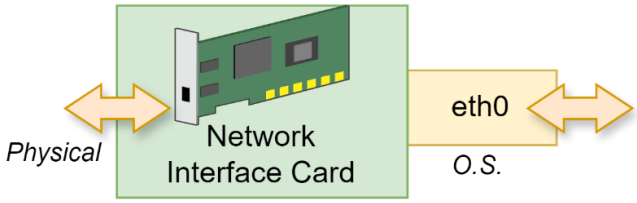


Figura 188. Conexión entre las interfaces física y lógica

La relación existente entre tarjetas de red e interfaces lógicas está definida de la siguiente forma: una interfaz lógica está asociada a una tarjeta de red, pero una determinada tarjeta puede tener asociadas múltiples interfaces lógicas (relación 1:N). De todo esto se encarga el kernel de Linux, que, mediante la carga de drivers, permite la asociación físico-lógica planteada.

El comando ifconfig nos permite consultar el estado de las interfaces (lógicas y virtuales) de una máquina linux. En la siguiente salida por consola, se puede apreciar la ejecución del comando, que muestra una interfaz lógica de conexión a una red, y otra virtual que gestiona la dirección de bucle. Dentro de una determinada interfaz es posible observar cada una de las direcciones y parámetros que gestiona una interfaz virtual. Entre ellos se puede destacar el MTU, que limita y restringe el tamaño máximo de los paquetes que circulan por una determinada interfaz.

```
$ ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.2  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 08:00:00:00:00:01  txqueuelen 1000  (Ethernet)
        RX packets 4  bytes 356 (356.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 90 (90.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        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 189. Ejecución del comando ifconfig en Linux

Las interfaces de un sistema operativo pueden ser asociadas tanto a un medio externo, como a otra interfaz virtual del mismo sistema, constituyendo lo que se conoce como tubería de interfaces virtuales, que son clave en la emulación de conexiones; a excepción de la interfaz de bucle (loopback o lo), que es única, y cuya función es la emisión de paquetes en la misma dirección de recepción.

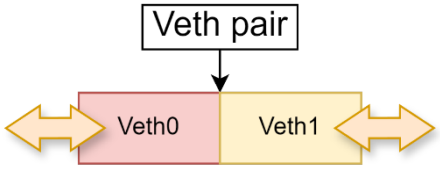


Figura 190. Tubería de interfaces virtuales

En el caso concreto de los conmutadores SDN, las interfaces pueden ser gestionadas por el controlador asociado al plano de datos, mediante diferentes APIs o protocolos; destacando entre ellos el protocolo GNMI. La capa máxima de operación de un conmutador de red tradicional es la *de enlace* (nivel 2), algo recurrente también en el direccionamiento de las interfaces SDN, lo que permite una adaptación rápida a protocolos de orden superior, puesto que se pueden emplear direcciones virtuales desde la capa 3 en adelante, delegando así la responsabilidad en el plano de datos.

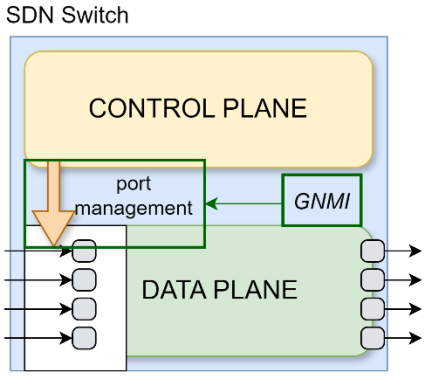


Figura 191. Gestión de interfaces en un switch SDN

Todos los fundamentos que se han abordado en esta sección permiten obtener al lector una visión más nítida de la gestión del enlace de red, imprescindible para tener una idea sólida de la interacción de un conmutador programable con el entorno en el que se sitúa.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga