Mecanismo cognitivo para control de congestión en redes ópticas definidas por software

Universidad del Cauca

Trabajo de Grado

Robinson Narváez Burbano
Mario Fernando Riascos Riascos

Director: PhD. José Giovanny López Perafán

Departamento de Telecomunicaciones
Facultad de Ingeniería Electrónica y Telecomunicaciones
Universidad del Cauca
Grupo de Investigación de Nuevas Tecnologías en Telecomunicaciones GNTT
Popayán, Cauca, 2021

Mecanismo cognitivo para control de congestión en redes ópticas definidas por software

Robinson Narváez Burbano
Mario Fernando Riascos Riascos

Trabajo de grado presentado a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la
Universidad del Cauca para obtener el título de:
Ingeniero en Electrónica y Telecomunicaciones

Director: PhD. José Giovanny López Perafán

Departamento de Telecomunicaciones
Facultad de Ingeniería Electrónica y Telecomunicaciones
Universidad del Cauca
Grupo de Investigación de Nuevas Tecnologías en Telecomunicaciones GNTT
Popayán, Cauca, 2021

Dedico este logro especialmente a mis padres, a mis hermanos y demás seres queridos, quienes me han dado todo su apoyo a lo largo de estos años, y a todos aquellos que de alguna u otra manera hicieron que esto fuera posible. Muchas gracias por todo.

Robinson Narváez Burbano.

Este logro no habría sido posible sin el acompañamiento de las personas más importantes en mi vida, mi familia, mil gracias.

Mario Fernando Riascos Riascos

Agradecimientos

Los autores del presente trabajo de grado expresan sus agradecimientos al Phd. José Giovanny López Perafan, director del proyecto, por sus importantes aportes y acompañamiento durante el proceso.

Al grupo de investigación de Nuevas Tecnologías en Telecomunicaciones - GNTT, al programa de Ingeniería Electrónica y Telecomunicaciones, a la facultad de Ingeniería Electrónica y Telecomunicaciones junto con todos sus docentes e ingenieros y a la Universidad del Cauca, muchas gracias por la formación brindada en el proceso de pregrado.

A familiares y amigos, agradecemos su apoyo y colaboración, la cual fue fundamental para la culminación de este importante logro.

Contenido

Dedicatoria **I**

Agradecimientos **III**

Lista de figuras **X**

Lista de tablas **XIV**

Lista de acrónimos **XVII**

Introducción **1**

1. Redes definidas por software, redes ópticas definidas por software y modelo óptico **7**

1.1. Software-Defined Networking (SDN) 8

1.1.1. Concepto SDN 8

1.1.2. Características de SDN 9

1.1.3. Arquitectura de SDN 9

1.1.4. Protocolo OpenFlow 11

1.2. Redes ópticas definidas por software (SDON) 12

1.2.1. Concepto de SDON 13
1.2.2. Características de SDON 13
1.2.3. Arquitectura de SDON 13
1.3. Modelo Óptico 15
1.3.1. Redes Ópticas 15

2. Características del Modelo Cognitivo y Balanceo de Carga 19
2.1. Cognición 20
2.1.1. Definición 20
2.1.2. Mecanismo Cognitivo 21
2.1.3. Ciclo Cognitivo 21
2.1.4. Redes Cognitivas 22
Definición 23
2.1.5. Trabajos Sobre Redes Cognitivas 23
FOCALE 24
CogNET 25
CHRON 25
2.1.6. Redes Cognitivas Definidas por Software 27
2.2. Balanceador de carga 29

3. Herramientas, simulación Y análisis de resultados 33
3.1. Herramientas de simulación 34
3.1.1. Entornos de Virtualización 34
VMware 34
Citrix XenServer 35

Microsoft Hyper-v 35
VirtualBox 35
3.1.2. Software de Emulación 36
Mininet 36
3.1.3. Controlador OpenFlow 38
ONOS 40
3.2. Metodología de simulación 41
3.3. Fase 1: Definición de la SDON de prueba 42
3.3.1. Caracterización de modelo óptico 42
3.3.2. Diseño de topología de red NSFNET 44
3.3.3. Flujo elefante y ratón 45
3.3.4. Parámetro de desempeño: probabilidad de bloqueo 46
3.3.5. Parámetro de desempeño: retardo extremo a extremo . . . 46
3.4. Fase 2: Implementación de la red de prueba 47
3.4.1. Simulación del modelo óptico y topología de red NSFNET . 49
3.5. Fase 3: Diseño del mecanismo cognitivo por medio de balanceo de carga 55
3.6. Fase 4: Presentación casos de estudio 58
3.6.1. Caso de estudio 1 62
3.6.2. Caso de estudio 2 63
3.7. Fase 5A: Caso de estudio 1 simulación de la SDON sin el mecanismo cognitivo 64
3.7.1. Resultados caso de estudio 1 67
Variación 1: 250Mbps 67

Variación 2: 500Mbps 68

3.8. Fase 5B: Caso de estudio 2 simulación de la SDON con el mecanismo cognitivo 69

3.8.1. Resultados caso de estudio 2: Escenario 1 76

Variación 1: 250Mbps 76

Variación 2: 500Mbps 77

3.8.2. Resultados caso de estudio 2: Escenario 2 78

Variación 1: 250Mbps 78

Variación 2: 500Mbps 79

3.8.3. Resultados caso de estudio 2: Escenario 3 80

Variación 1: 250Mbps 81

Variación 2: 500Mbps 82

3.9. Análisis de resultados de la SDON de prueba sin el mecanismo cognitivo y con el mecanismo cognitivo 83

3.9.1. Análisis de resultados caso de estudio 1 84

3.9.2. Análisis de resultados caso de estudio 2 91

Análisis de resultados escenario 1 91

Análisis de resultados escenario 2 97

Análisis de resultados escenario 3 104

3.9.3. Análisis comparativo de una arquitectura de red SDON con y sin mecanismo cognitivo 112

4. Conclusiones, recomendaciones y trabajos futuros 119

4.1. Conclusiones 119

4.2. Recomendaciones 121

4.3. Trabajos futuros 122

Bibliografía **123**

Anexos **1**

Lista de figuras

1.1. Arquitectura SDN. 10
1.2. OpenFlow en SDN. 11
1.3. Arquitectura SDON. 14
1.4. Interfaz de MiniEdit y CLI de Mininet. 15
1.5. Red WDM. 16

2.1. Ciclo cognitivo. 22
2.2. Arquitectura FOCALE. 24
2.3. Arquitectura CogNET. 25
2.4. Arquitectura CHRON. 27

3.1. Interfaz VMware Workstation. Elaboración propia 36
3.2. Comparación de controladores. 39
3.3. Arquitectura ONOS. 41
3.4. Diagrama metodología de simulación. 42
3.5. Arquitectura Clos para un switch fabric óptico. 43
3.6. OXC con conversión OEO. 44
3.7. Topología de red NSFNET. 45

3.8. Modelo WDM switch fabric creado con MiniEdit. Elaboración propia 49
3.9. Modelo óptico y topología de red NSFNET. 51
3.10. Comando **pingall** en red de prueba. 52
3.11. Red de prueba SDON para lambda 1, MiniEdit. 53
3.12. Red de prueba SDON para lambda 1, Controlador ONOS. 54
3.13. Comando **net** en topología de prueba SDON, Hosts. 54
3.14. Comando **net** en topología de prueba SDON, Nodos. 55
3.15. Método de control cognitivo para una SDON. Elaboración propia . 56
3.16. NSFNET MiniEdit. Elaboración propia 59
3.17. Diagrama casos de estudio. Elaboración propia 63
3.18. Aplicaciones instaladas en el controlador caso de estudio 1. 64
3.19. Comando pingall en topología de prueba caso 1. 65
3.20. Comando link utilizado en la SDON de prueba. 65
3.21. Comando en Ubuntu 18 para iniciar la simulación. 66
3.22. Diagrama de funcionamiento Reactive Forwarding ONOS. 66
3.23. Aplicaciones instaladas en el controlador caso de estudio 2. 69
3.24. Comando para crear una nueva aplicación en API de ONOS. . . . 70
3.25. Aplicación creada correctamente con API de ONOS. 70
3.26. Configuración properties archivo pom.xml. 71
3.27. Ruta de Archivos Java. 71
3.28. Comando para empaquetar la aplicación de ONOS. 71
3.29. Método getPortStatics API de ONOS. 73
3.30. Diagrama de funcionamiento balanceador de carga. 75
3.31. Probabilidad de bloqueo sin MCCC, variación 1. 85

3.32. Retardo extremo a extremo sin MCCC, variación 1. 86
3.33. Throughput sin MCCC, variación 1. 86
3.34. Probabilidad de bloqueo sin MCCC, variación 1. 87
3.35. Retardo extremo a extremo sin MCCC, variación 1. 87
3.36. Throughput sin MCCC, variación 1. 87
3.37. Probabilidad de bloqueo sin MCCC, variación 2. 88
3.38. Retardo extremo a extremo sin MCCC, variación 2. 89
3.39. Throughput sin MCCC, variación 2. 89
3.40. Probabilidad de bloqueo sin MCCC, variación 2. 90
3.41. Retardo extremo a extremo sin MCCC, variación 2. 90
3.42. Throughput sin MCCC, variación 2. 90
3.43. Probabilidad de bloqueo con MCCC, escenario 1, variación 1. . . . 92
3.44. Retardo extremo a extremo con MCCC, escenario 1, variación 1. . 92
3.45. Throughput con MCCC, escenario 1, variación 1. 93
3.46. Probabilidad de bloqueo con MCCC, escenario 1, variación 1. . . . 93
3.47. Retardo extremo a extremo con MCCC, escenario 1, variación 1. . 94
3.48. Throughput con MCCC, escenario 1, variación 1. 94
3.49. Probabilidad de bloqueo con MCCC, escenario 1, variación 2. . . . 95
3.50. Retardo extremo a extremo con MCCC, escenario 1, variación 2. . 96
3.51. Throughput con MCCC, escenario 1, variación 2. 96
3.52. Probabilidad de bloqueo con MCCC, escenario 1, variación 2. . . . 97
3.53. Retardo extremo a extremo con MCCC, escenario 1, variación 2. . 97
3.54. Throughput con MCCC, escenario 1, variación 2. 97
3.55. Probabilidad de bloqueo con MCCC, escenario 2, variación 1. . . . 99

3.56. Retardo extremo a extremo con MCCC, escenario 2, variación 1. . 99
3.57. Throughput con MCCC, escenario 2, variación 1. 100
3.58. Probabilidad de bloqueo con MCCC, escenario 2, variación 1. . . . 100
3.59. Retardo extremo a extremo con MCCC, escenario 2, variación 1. . 101
3.60. Throughput con MCCC, escenario 2, variación 1. 101
3.61. Probabilidad de bloqueo con MCCC, escenario 2, variación 2. . . . 102
3.62. Retardo extremo a extremo con MCCC, escenario 2, variación 2. . 102
3.63. Throughput con MCCC, escenario 2, variación 2. 103
3.64. Probabilidad de bloqueo con MCCC, escenario 2, variación 2. . . . 103
3.65. Retardo extremo a extremo con MCCC, escenario 2, variación 2. . 104
3.66. Throughput con MCCC, escenario 2, variación 2. 104
3.67. Probabilidad de bloqueo con MCCC, escenario 3, variación 1. . . . 105
3.68. Retardo extremo a extremo con MCCC, escenario 3, variación 1. . 106
3.69. Throughput con MCCC, escenario 3, variación 1. 106
3.70. Probabilidad de bloqueo con MCCC, escenario 3, variación 1. . . . 107
3.71. Retardo extremo a extremo con MCCC, escenario 3, variación 1. . 107
3.72. Throughput con MCCC, escenario 3, variación 1. 107
3.73. Probabilidad de bloqueo con MCCC, escenario 3, variación 2. . . . 108
3.74. Retardo extremo a extremo con MCCC, escenario 3, variación 2. . 109
3.75. Throughput con MCCC, escenario 3, variación 2. 109
3.76. Probabilidad de bloqueo con MCCC, escenario 3, variación 2. . . . 110
3.77. Retardo extremo a extremo con MCCC, escenario 3, variación 2. . 110
3.78. Throughput con MCCC, escenario 3, variación 2. 110
3.79. Probabilidad de bloqueo para la variación de 250Mbps. 114

3.80. Retardo extremo a extremo para la variación de 250Mbps. 114
3.81. Throughput para la variación de 250Mbps. 114
3.82. Probabilidad de bloqueo para la variación de 500Mbps. 115
3.83. Retardo extremo a extremo para la variación de 500Mbps. 115
3.84. Throughput para la variación de 500Mbps. 115
3.85. Probabilidad de bloqueo para la variación de 250Mbps. 116
3.86. Retardo extremo a extremo para la variación de 250Mbps. 116
3.87. Throughput para la variación de 250Mbps. 116
3.88. Probabilidad de bloqueo para la variación de 500Mbps. 117
3.89. Retardo extremo a extremo para la variación de 500Mbps. 117
3.90. Throughput para la variación de 500Mbps. 117

Lista de tablas

1.1. Componentes de OpenFlow. 12

2.1. Comparación de arquitecturas cognitivas. Elaboración propia . . . 28

3.1. Configuración VMware Workstation y herramientas. Elaboración propia 48

3.2. Líneas de fibra con sus lambdas y hosts. Elaboración propia . . . 50

3.3. Características de la red de prueba. Elaboración propia 50

3.4. Aplicaciones y herramientas para la construcción de la red de prueba. 52

3.5. Hosts alcanzables con su respectiva lambda. 52

3.6. Puertos de conexión *OpenFlow* nodo a nodo NSFNET. 59

3.7. Distribución de tráfico, Simulación de 12 horas. 62

3.8. Resultados caso de estudio 1, variación 1 68

3.9. Resultados caso de estudio 1, variación 2 69

3.10. Resultados caso de estudio 2, escenario 1, variación 1 77

3.11. Resultados caso de estudio 2, escenario 1, variación 2 78

3.12. Resultados caso de estudio 2, escenario 2, variación 1 79

3.13. Resultados caso de estudio 2, escenario 2, variación 2 80

3.14. Resultados caso de estudio 2, escenario 3, variación 1 82

3.15. Resultados caso de estudio 2, escenario 3, variación 2 83

Lista de acrónimos

ACO	*Ant Colony Optimization*, Optimización por Colonia de Hormigas
APIS	*Application Programming Interfaces*, Interfaz de Programación de Aplicaciones
BW	*Bandwidth*, Ancho de Banda
CDS	*Cognitive Decision System*, Sistema de Decisión Cognitivo
CHRON	*Cognitive Heterogeneous Reconfigurable Optical Network*, Red óptica Reconfigurable Heterogénea Cognitiva
CMS	*Control and Management System*, Sistema de Control y Gestión
COADM	*Configurable Optical Add-Drop Multiplexer*, Multiplexor Óptico de Adición-Descarga Configurable
CogNET	*Cognitive Complete Knowledge Network*, Red de Conocimiento Completo Cognitivo
CN	*Cognitives Networks*, Redes Cognitivas
CR	*Cognitive Radio*, Radio Cognitivo

DWDM *Dense Wavelength Division Multiplexing*, Multiplexación por División de Longitud de Onda Densamente Espaciada

FOCALE *Foundation, Observation, Comparison, Action, Learn, rEason*, Fundación, observación, comparación, acción, aprendizaje, razón

HTTP *Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto

HTTPS *Hypertext Transfer Protocol Secure*, Protocolo Seguro de Transferencia de Hipertexto

ICMP *Internet Control Message Protocol*, Protocolo de Mensajes de Control de Internet

IEEE *Institute of Electrical and Electronics Engineers*, Instituto de Ingenieros Eléctricos y Electrónicos

ITU *International Telecommunication Union*, Unión Internacional de las Telecomunicaciones

ML *Machine Learning*, Aprendizaje de Máquina

NMons *Network Monitor System*, Sistema de Monitoreo de Red

OLT *Optical Line Terminals*, Terminales de Línea Óptica

OSPF *Open Shortest Path First*, Protocolo abierto para el primer camino mas corto

QoS *Quality of Service*, Calidad del Servicio

ROADM *Reconfigurable Optical Add-Drop Multiplexer*, Multiplexor Óptico de Adición-Descarga Reconfigurable

SAEs *Software Adaptable Elements*, Elementos Adaptables de Software

SAN *Software Adaptable Network*, Redes Adaptables de Software

SDCoN *Software Defined Cognitive Networking*, Redes Cognitivas Definidas por Software

SDN *Software Defined Networks*, Redes Definidas por Software

SDR *Software Defined Radio*, Radio Definido por Software

SDON *Software Defined Optical Networks*, Redes Ópticas Definidas por Software

SNMP *Simple Network Management Protocol*, Protocolo Simple de Administración de Red

SSH *Secure Shell*

TCP *Transmission Control Protocol*, Protocolo de Control de Transmisión

TLS *Transport Layer Security*, Seguridad de la Capa de Transporte

UDP *User Datagram Protocol*, Protocolo de Datagramas de Usuario

WARPC *Wireless Open- Access Research Platform*, Plataforma de Estudio de Acceso Libre Inalámbrico

Introducción

A medida que Internet evoluciona, la velocidad de transmisión y la precisión en las rutas de red a diferentes distancias también lo hacen, se han incluido nuevas tecnologías en el procesamiento de información que se adapten a los nuevos requisitos de la red, para así mejorar la calidad del servicio (QoS, *Quality of Service*) y la experiencia del usuario [1], sin embargo, la capacidad de estos sistemas generan preocupación puesto que se puede afectar la eficiencia de la red con diferentes procesos de congestión generados por retardos, interrupción del tráfico y una degradación del rendimiento de la red [2].

Desde hace varios años se han realizado investigaciones sobre el control de la congestión en las redes y se han propuesto algunos mecanismos tales como Remy [3] , TCP-Vegas [4] y Sabul [5]; asimismo, se han propuesto otros métodos que incluyen soluciones mediante Q-Learning [6], aprendizaje de máquina (ML, *Machine Learning*) [7], optimización por colonia de hormigas (ACO, *Ant Colony Optimization*) [8], entre otros. Aunque algunos de estos mecanismos han dado resultados importantes en el desempeño de una red de comunicaciones (principalmente en redes distribuidas), el empleo de procesos enfocados en cognición puede permitir el desarrollo de alternativas de solución a la congestión en las redes ópticas definidas por software (SDON, *Software Defined Optical Networks*).

En este orden de ideas, cabe destacar que el diseño de las redes se enfoca hacia sistemas centralizados en donde se busca que, controlando algunos elementos de la red se puedan ejecutar varias operaciones, controlar distintos dispositivos y asimismo desarrollar diversas aplicaciones [9] [10] [11], que conlleven a la creación de un sistema autónomo, que permita a la red realizar tareas de manteni-

miento, programación, optimización, configuración y reconfiguración ante fallas y falta de recursos [7].

Asimismo, con el sólo hecho de contar con una red definida por software (SDN, *Software Defined Networking*) no implica que los problemas de congestión estén solucionados, al contrario, la cantidad de información que va a circular por este tipo de redes es mayor, puesto que, con un sistema centralizado se pueden tener muchas aplicaciones trabajando simultáneamente, y a causa de ello la congestión seguirá existiendo y sin duda se aumentaría ocasionando una interrupción en el tráfico [9].

Por consiguiente, se espera que los problemas de congestión que se presentan en las redes definidas por software, afecten el desempeño de la red óptica definida por software, generando inconvenientes similares que pueden ser aminorados por medio de mecanismos implementados para este tipo de problemas. Asimismo, SDON también cuenta con un plano de control que puede gestionar la funcionalidad del sistema y por medio de este plano que, además es programable, se pueden incluir mecanismos cognitivos que pueden ayudar a mitigar la congestión [10] [12].

Por los aspectos anteriormente mencionados, se planteó la realización de una propuesta de un mecanismo de control de congestión basado en el ciclo cognitivo para redes ópticas definidas por software, puesto que la mayoría de las soluciones que aparecen en la literatura científica al respecto no se enfocan en este tipo de redes. Lo anterior implicó realizar la siguiente pregunta de investigación:

¿Cómo un mecanismo cognitivo para control de congestión impacta el desempeño de una red óptica definida por software?

En consecuencia este documento propone el desarrollo de un mecanismo cognitivo que permita evaluar el desempeño de la red óptica ante una congestión, para esto se plantea una solución por medio de balanceo de carga el cual radica en la implementación de un modelo centralizado basado en el ciclo cognitivo, que tiene como propósito observar constantemente la información del entorno de la red

para luego tomar decisiones basado en los cambios que se presentan en esta. De esta forma, la solución propuesta se aplica a la red óptica de prueba dando mas inteligencia a los nodos en los cuales se destina.

En la propuesta de realización del presente trabajo de grado, se propusieron los siguientes objetivos:

Objetivo general

Implementar un mecanismo cognitivo para control de congestión en una red óptica definida por software.

Objetivos específicos

- Diseñar un mecanismo cognitivo basado en las características del ciclo cognitivo para control de congestión.
- Evaluar mediante simulación el desempeño[1] del mecanismo cognitivo en la SDON de prueba.

Organización del trabajo de grado

Los objetivos expuestos anteriormente se desarrollan de acuerdo al planteamiento de cuatro capítulos que contienen explicaciones teóricas y el desarrollo del mecanismo cognitivo, además, de la respuesta a la pregunta de investigación con sus respectivas conclusiones y los anexos. Inicialmente se necesitó elaborar una base conceptual de los temas mas importantes a tratar, en este sentido se muestran algunos modelos cognitivos que son necesarios para la elaboración de este trabajo, seguido a esto se definen las herramientas que se utilizan para la

[1] Desempeño: medida de la probabilidad de bloqueo y retardos extremo a extremo.

implementación de la red de prueba e igualmente del mecanismo cognitivo, también se define la metodología de simulación y se muestran los resultados. por último, se presentan las conclusiones, recomendaciones y trabajos futuros.

A continuación se hace una descripción detallada de cada uno de los cuatro Capítulos además de los anexos.

En el **capitulo 1** se determinan los conceptos de las redes definidas por software, las redes ópticas definidas por software y el modelo óptico, las bases conceptuales a tener en cuenta son definición, importancia y caracterísiticas principales, como lo son su arquitectura, el protocolo Openflow, el cual es el encargado de realizar la comunicación entre el plano de datos y el plano de control en el modelo SDN. Finalmente en este capitulo se hace una descripción del modelo óptico mencionando algunos términos que son importantes para la elaboración de un WDM switch fabric que se encarga de encaminar la información que se genera en las longitudes de onda implementadas en las lineas de fibra óptica.

En el **capitulo 2** se hace la caracterización del modelo cognitivo realizando un estudio sobre los diferentes modelos basados en el ciclo cognitivo que han sido utilizados en algunos trabajos a lo largo de los años, de igual forma se muestran las arquitecturas utilizadas en dichos modelos, se definen los términos de cognición, entre otros aspectos. Finalmente se hace una descripción de los balanceadores de carga mediante algunos artículos relacionados para lo cual se emplea la información de la literatura científica consultada.

El **capitulo 3** expone las herramientas, conceptos y características importantes que se emplean en el desarrollo del mecanismo cognitivo. Inicialmente, se elaboró una red de pruebas, la cual, acompañada de los diseños y modelos de simulación, proveen un ambiente adecuado para el desarrollo del mecanismo. En el cuerpo de este capítulo, se realizan las pruebas y análisis pertinentes de acuerdo a dos casos de estudio, acondicionando el respectivo mecanismo a la red y sin la inclusión de este. Finalmente, se muestran resultados de los casos de estudio, se realiza una evaluación y se determinan las conclusiones con respecto al desempeño que se obtuvo.

En el **capitulo 4** se describen las conclusiones, recomendaciones y trabajos fu-

turos que surgen a partir del desarrollo del trabajo de grado.

Por ultimo, se incluyen los **Anexos** que están compuestos de las topologías utilizadas para el desarrollo de este trabajo de grado, así como del modelo óptico propuesto, además de algunos comandos que se emplearon para generar el tráfico que circula por medio de la red de prueba.

Capítulo 1

Redes definidas por software, redes ópticas definidas por software y modelo óptico

Desde al menos una década, se reconoció que era necesario realizar cambios a las redes de nueva generación desarrollando nuevas capas de abstracción para mejorar las funciones de control de la red, esto con el fin de automatizar y simplificar la administración de la red.

Las redes definidas por software SDN iniciaron como un hito para el desarrollo de las nuevas capas de abstracción, puesto que este tipo de redes separan funciones del plano de control del plano de datos, centralizan la lógica del control y las funcionalidades de la red son programables.

La idea de SDN consiste en que la lógica del control sea centralizada y no distribuida, obteniendo una mejor gestión de la red y de todas sus funciones, generando un equilibrio entre la facilidad de realizar administración y mitigar los problemas de escalabilidad que puedan surgir.

Las redes ópticas juegan un papel importante en la tecnología moderna debido a su alta capacidad de transmisión, por lo tanto, se planteó ampliar SDN a un ámbito óptico, por lo que surge la idea de redes ópticas definidas por software

SDON, esto con el fin de aprovechar la flexibilidad del control de las redes SDN apoyadas por la infraestructura de la red óptica subyacente.

1.1. Software-Defined Networking (SDN)

En la actualidad, las redes se han convertido en un factor indispensable para el crecimiento de las comunicaciones para optimizar la transmisión de información de una forma ágil y masiva, en donde se requiere contar con elementos cuya infraestructura sea lo suficientemente idónea para la administración de datos de una forma eficiente, nace la idea de las redes controladas mediante software. [13]

Así mismo, el desarrollo de las redes ópticas ha promovido la implementación de velocidades altas de transmisión de información, ancho de banda, comunicación a larga distancia y ultra gran capacidad de transporte. Sin embargo, con la evolución del tráfico de red, las redes ópticas tradicionales no han podido soportar el transporte de tráfico hacia centros de datos y servicios basados en la nube [14], estos nuevos y emergentes casos de uso, requieren una interconexión más fiable y flexible, es por ello, que surge la necesidad de implementar un control más intuitivo en la red óptica.

Las redes definidas por software (SDN, *Software Defined Network*) son una tecnología emergente estandarizada que disgrega la arquitectura de las redes tradicionales, es decir separa el plano de datos del plano de control permitiendo que la red sea programable, aportando beneficios como la reducción de la complejidad de las infraestructuras y la vinculación de un control centralizado por medio de interfaces de desarrollo.

1.1.1. Concepto SDN

El Concepto SDN se origina de acuerdo con la complejidad e inflexibilidad de las redes tradicionales, puesto que, es difícil configurarlas de acuerdo con políticas que permitan restablecer fallos y sobrecargas, adicional a ello la integración de

las redes tradicionales es vertical con la asociación del plano de control y datos. Las redes definidas por software son un paradigma que ayuda a mejorar estos inconvenientes rompiendo la integración vertical, promoviendo la centralización del control, separando la lógica del control de la red de los enrutadores y conmutadores subyacentes e introduciendo la capacidad para programarla [15].

1.1.2. Características de SDN

La red definida por software es una tecnología emergente dinámica, administrable, rentable y adaptable, por lo que es ideal para la gestión de servicios y aplicaciones de red, sus características principales se basan en [15]:

- Directamente programable: Gracias a que las funciones de reenvío fueron desacopladas
- Ágil: Permite al administrador de red o a las aplicaciones hacer cambios dinámicos en los flujos de red.
- Gestión centralizada: El controlador centralizado mantiene una visión global de la red.
- Programación configurable: Permite a los administradores configurar, administrar, asegurar y optimizar los recursos de red rápidamente por medio de software dinámico.
- Basado en estándares abiertos y de proveedor neutral: Permite simplificar el diseño de la red y la integración de un solo protocolo para todos los fabricantes.

1.1.3. Arquitectura de SDN

La arquitectura se divide en tres planos: datos, control y gestión, como se muestra en la figura 1.1. El plano de datos va dirigido a los dispositivos de red, que son responsables del envío de datos, el plano de control determina las políticas de

conmutación y/o enrutamiento en un dispositivo de comunicación, es decir quien determina que criterios se toman en cuenta para transmitir un paquete de datos y también los criterios para la selección de las mejores rutas y finalmente, el plano de gestión incluye los servicios software, tales como el protocolo simple de administración de red (SNMP, *Simple Network Management Protocol*), siendo esta herramienta la que se utiliza para controlar de forma remota la configuración de la red [15] [16].

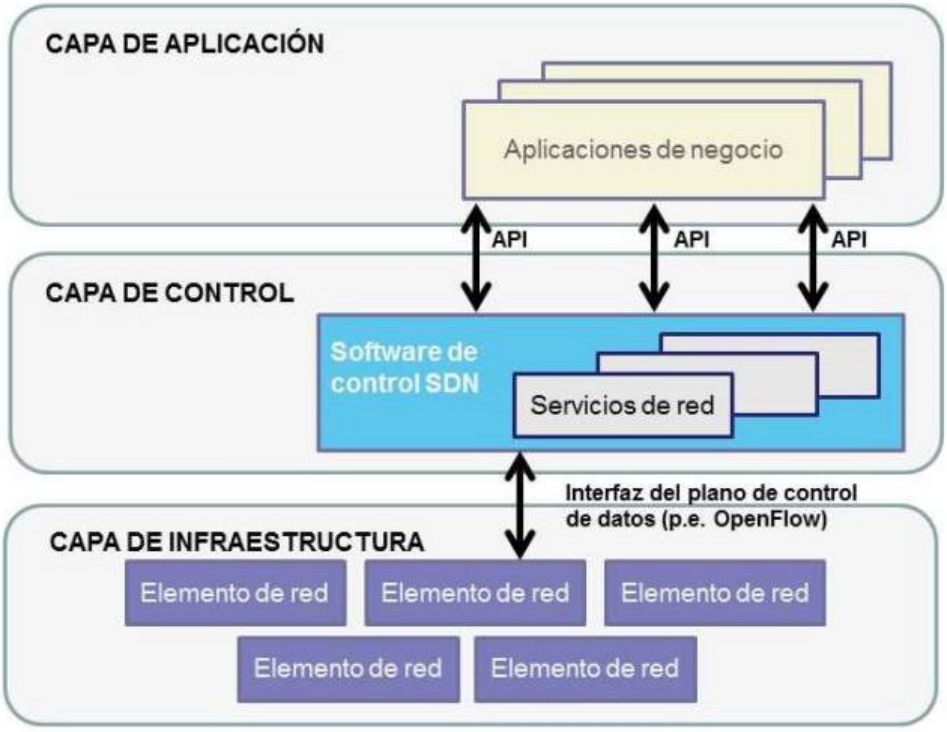

Figura 1.1: Arquitectura SDN. Tomada de [16]

El funcionamiento se describe de la siguiente forma, la política de red se define en el plano de gestión, el plano de control aplica la política, y el plano de datos la ejecuta mediante el envío de datos [17], esta modalidad ha mostrado resultados eficientes en cuanto a la velocidad de enrutamiento y al tráfico de información.

1.1.4. Protocolo OpenFlow

OpenFlow es un protocolo de bajo nivel para implementar control en los nodos de la red, permite a un servidor de software determinar el camino de reenvío de paquetes que debería seguir en una red de *switches*. Con el protocolo *OpenFlow*, una red puede ser gestionada como un todo, no como un número de dispositivos que gestionar individualmente, es el propio servidor el que dice a los *switches* dónde deben enviar los paquetes, además, como un estándar abierto busca la interoperabilidad entre distintos fabricantes. Actualmente, *OpenFlow* proporciona una interfaz estandarizada entre controladores y conmutadores SDN u otras rutas de datos [18]. A continuación en la figura 1.2 y en la tabla 1.1, se muestran los cuatro componentes: capa de mensaje, máquina de estado, interfaz del sistema y configuración, además de los siguientes términos que se refieren a *rcv* recepción, *snd* envío, *tm* transmisión y *ctrl* control [19].

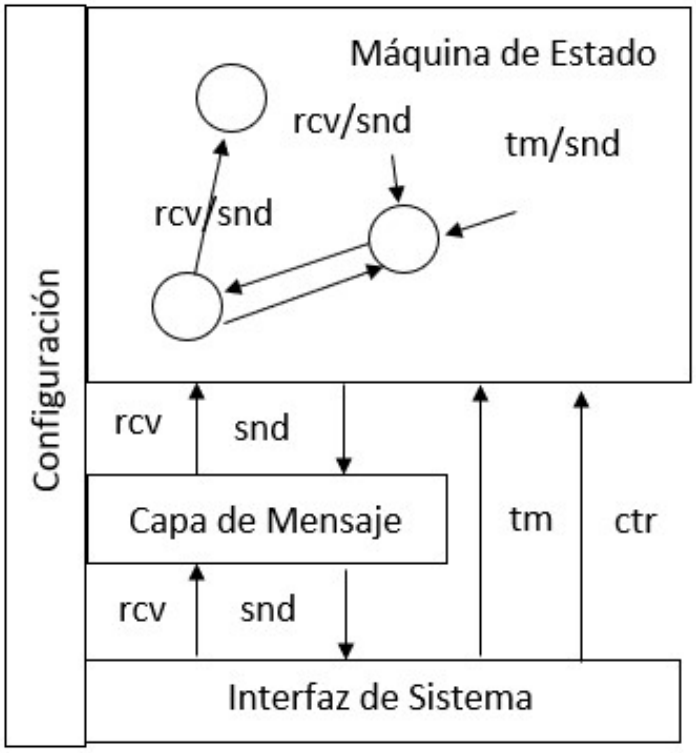

Figura 1.2: OpenFlow en SDN. Modificada a partir de [19]

COMPONENTE	DESCRIPCIÓN
Capa de mensaje	La capa de mensaje es el núcleo de la pila de protocolos. Define la estructura válida y la semántica para todos los mensajes. Una capa de mensaje típica admite la capacidad de construir, copiar, comparar, imprimir y manipular mensajes.
Máquina de estados	La máquina de estados define el comportamiento básico de bajo nivel del protocolo. Por lo general, se utiliza para describir acciones como: negociación, descubrimiento de capacidades, control de flujo, entrega, etc.
Interfaz del sistema	La interfaz del sistema define cómo interactúa el protocolo con el mundo exterior. Por lo general, identifica las interfaces necesarias y opcionales junto con su uso previsto, como TLS y TCP como canales de transporte.
Configuración	Casi todos los aspectos del protocolo tienen configuraciones o valores iniciales. La configuración puede abarcar desde tamaños de búfer predeterminados e intervalos de respuesta hasta certificados X.509.
Modelo de datos	Otra forma de considerar el protocolo OpenFlow es comprender su modelo de datos subyacente. Cada conmutador mantiene un modelo de datos relacionales que contiene los atributos para cada abstracción de OpenFlow. Estos atributos describen una capacidad de abstracción, su estado de configuración o algún conjunto de estadísticas actuales.

Tabla 1.1: Componentes de OpenFlow. Modificada a partir de [19]

1.2. Redes ópticas definidas por software (SDON)

Los proveedores de servicio de red enfrentan un problema con respecto a la administración de las redes actuales conformadas por dispositivos y protocolos complejos [20]. Las tendencias actuales de redes centradas en datos solicitan altas velocidades de transmisión y con una probabilidad de error significativamente baja en la entrega de paquetes, esta demanda se cumple cuando se fusionan los beneficios de las redes bajo conmutación de circuitos con las de conmutación de paquetes, la red óptica emplea este método para brindar gran capacidad y velocidad en sus diferentes conexiones, sin embargo, la constante evolución de las tecnologías de comunicación exigen una gestión controlada en la red que permita flexibilidad, programación y adaptación.

1.2.1. Concepto de SDON

Las redes ópticas desempeñan un papel importante debido a sus altas capacidades de transmisión, sin embargo, estas redes necesitan evolucionar hacia nuevas funciones de control para simplificar y automatizar su administración [21]. Desde hace al menos una década se reconoció que es necesario trabajar las redes ópticas bajo el concepto SDN para aprovechar las ventajas de capacidad, distancia de transmisión y bajo consumo de energía de la infraestructura de la red óptica con la flexibilidad que brinda SDN [22].

1.2.2. Características de SDON

Dado que, SDON es una evolución de las redes ópticas tradicionales con control centralizado y que este control es proporcionado por SDN, las características de SDON son similares a las de SDN, puesto que, en ellas se puede hablar de programabilidad, gestión centralizada, programación configurada, entre otras. Sin embargo, para especificar características inherentes de las redes ópticas, se examina el campo de transmisión de la señal óptica, teniendo como resultado los transceptores ópticos definidos por software que pueden configurar de manera flexible la transmisión y recepción de una amplia gama de señales ópticas [21], la multiplexación por división espacial que proporciona una vía emergente para transmisiones eficientes, la conmutación óptica controlada por SDN, que abarca los primeros elementos de conmutación y luego los paradigmas de conmutación generales y finalmente, las infraestructuras cognitivas de comunicación fotónica que monitorean la calidad de la señal óptica [22], en este orden de ideas las características más diferenciales de SDON con respecto a las redes definidas por software corresponden a los elementos de transmisión.

1.2.3. Arquitectura de SDON

La arquitectura de SDON en la que se basa el presente trabajo de grado se representa en la figura 1.3 la cual consiste en dos clientes A y B entre los cuales se

mantiene una comunicación. Estos clientes están conectados a los conmutadores de flujo abierto, que a su vez son controlados por el controlador SDN, y este, por sí solo realiza el reenvío de la comunicación [23] [24] [25] [26].

La parte óptica de la arquitectura, como se puede ver en la figura 1.3, esta compuesta de multiplexores ópticos reconfigurables de adición-extracción (ROADM, *Reconfigurable Optical Add/Drop Multiplexer*) que son una forma de multiplexor de suma y caída óptica que agrega la capacidad de conmutar de forma remota el tráfico desde un sistema de multiplexación por división de longitud de onda y multiplexores ópticos configurables de adición-extracción (COADM, *Configurable Optical Add-Drop Multiplexer*) diseñado para ofrecer una mayor funcionalidad y flexibilidad necesaria para las redes ópticas avanzadas de la actualidad; en caso de que se tenga un menor número de longitudes de onda para descartar o agregar. La comunicación entre los interruptores de flujo abierto y los interruptores *ROADM* se realiza a través de un convertidor eléctrico a óptico, puesto que, tanto los conmutadores ópticos como los de paquetes están controlados por el controlador SDN. La interacción entre el controlador y el agente o el interruptor de flujo abierto se basa en el protocolo *OpenFlow* [24] [27].

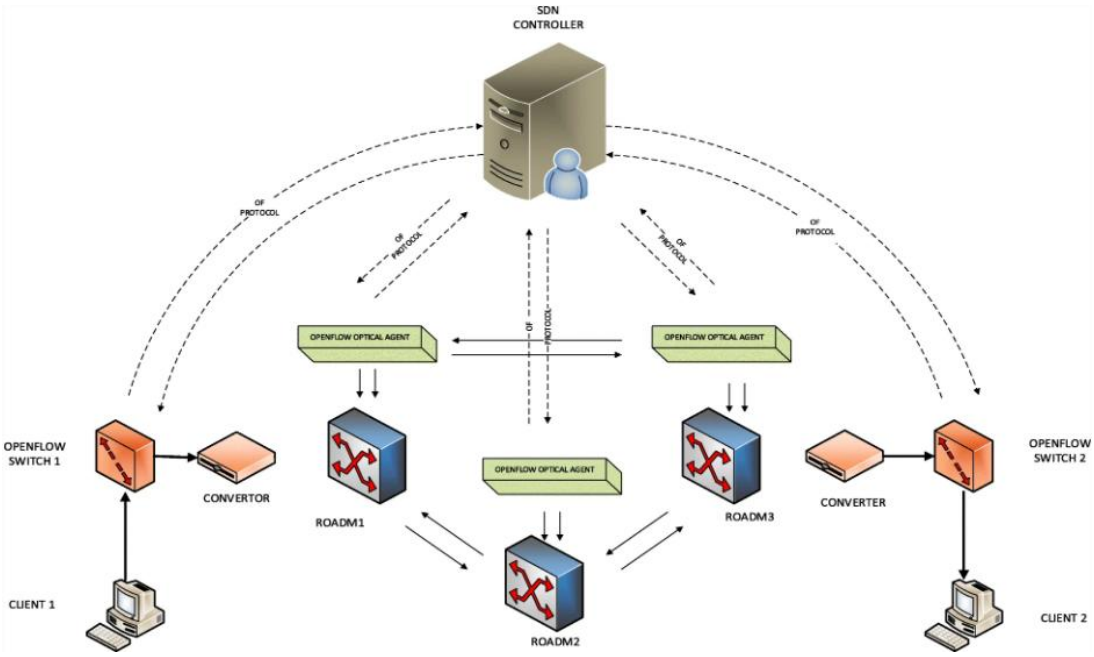

Figura 1.3: Arquitectura SDON. Tomada de [26]

1.3. Modelo Óptico

La red óptica definida por software aparece a partir de un modelo óptico, el cual es objeto de estudio para el desarrollo del presente trabajo de grado. Se podría pensar en construir una red de máquinas virtuales, sin embargo, se ha experimentado como una solución compleja; por este motivo, muchos de los investigadores han optado por utilizar el emulador *Mininet*, este emulador está desarrollado con atributos de flexibilidad y escalabilidad.

Mininet contiene una herramienta experimental de interfaz gráfica denominada MiniEdit, ver figura 1.4, esta herramienta ayuda a crear y ejecutar topologías rápidamente. Los dispositivos emulados son *switches* que no contienen capacidades ópticas pero que, se puede caracterizar por medio de *switches* que se comportan como una multiplexación por división de longitud de onda (WDM, *Wavelength Division Multiplexing*), esta topología refleja las rutas que pueden ser utilizadas por las longitudes de onda en una red de switches *WDM* donde el controlador SDN puede encontrar un camino para la comunicación entre hosts a través de este modelo de red [28].

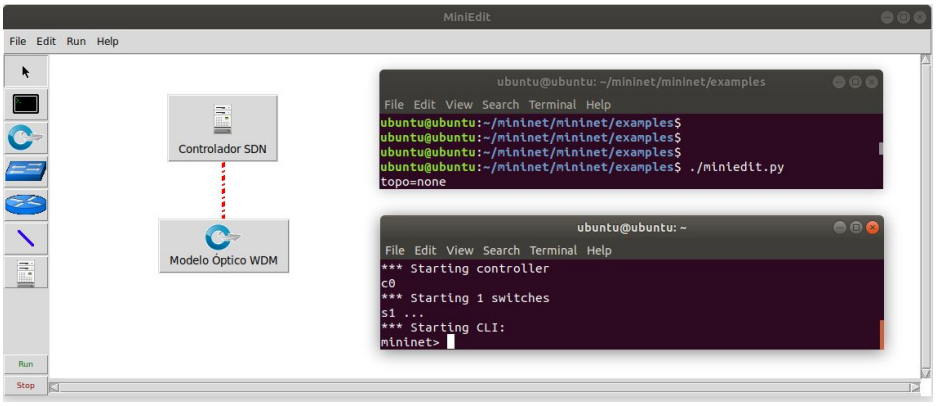

Figura 1.4: Interfaz de MiniEdit y CLI de Mininet. Elaboración propia

1.3.1. Redes Ópticas

El tráfico de datos en las redes ha tenido un crecimiento exponencial debido a la creciente demanda de servicios y aplicaciones por parte de los usuarios, llevando

a una saturación en los nodos. Por este motivo las redes ópticas han desarrollado nuevas tecnologías de multiplexación sobre los enlaces existentes para así soportar grandes requerimientos de ancho de banda (BW, *Bandwidth*). En el diseño y la implementación de las redes normalmente se busca bajar costos, como los derivados de la operación de la red y de las obras civiles de instalación. La multiplexación por división de longitud de onda (WDM) es una tecnología que se utiliza en la transmisión de varias longitudes de onda sobre una misma línea; cada longitud de onda representa un canal óptico dentro de la línea [29]. Un sistema WDM posee métodos ópticos que permiten combinar dichos canales dentro la línea y extraerlos en puntos apropiados a lo largo de la red. Al transmitir simultáneamente varios canales, se logra incrementar la capacidad del medio de transmisión. Un sistema básico utiliza un multiplexor para unir las señales y un demultiplexor para separarlas, ver figura 1.5. El sistema puede ser unidireccional o bidireccional, esto depende de la configuración del multiplexor WDM, pero en la actualidad los nuevos estándares son de tipo unidireccional [30].

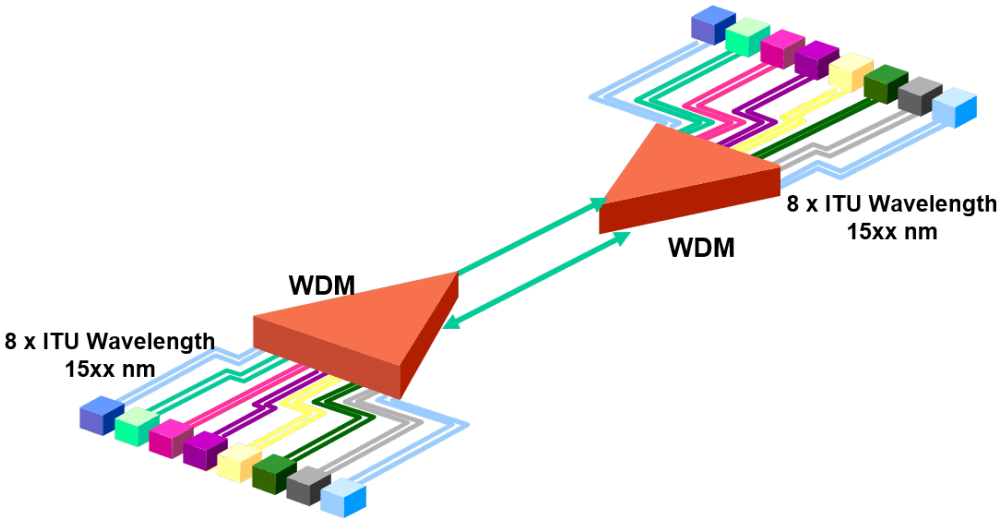

Figura 1.5: Red WDM. Tomada de [30]

Una técnica de transmisión utilizada en fibra óptica en la que se utilizan las longitudes de onda para transmitir datos en paralelo por bits o en serie por caracteres se conoce como Multiplexación por división de longitud de onda densamente espaciada (DWDM, *Dense Wavelength Division Multiplexing*). La unión internacional de las telecomunicaciones (ITU, *International Telecommunication Union*),

considera “densamente espaciados” a los sistemas WDM con más de cuatro longitudes de onda y estandariza la separación de los canales aproximadamente a un mínimo de 100 GHz, que corresponde a 0.8 nm. El aumento de la demanda de los consumidores hace que se incremente el ancho de banda, lo que es posible gracias a la DWDM. Las ventajas principales de DWDM son que las redes de fibra existentes pueden ser escaladas para soportar un mayor BW remplazando simplemente los componentes terminales. Otra ventaja que brindan las redes ópticas basadas en esta tecnología, es la posibilidad de crear topologías virtuales de red en el dominio óptico, sobre una topología que siempre es dependiente de la instalación de la fibra óptica [31]. Los sistemas DWDM pueden proporcionar centenares de longitudes de onda espaciadas menos de 1 nm dentro del BW disponible del amplificador. Por tanto, DWDM se usa en el núcleo de las redes metropolitanas y en las redes larga distancia [32] [33].

El uso de la técnica de multiplexación por división de longitudes de onda *WDM* para fines de transmisión en redes ópticas ha logrado avances importantes en lo relacionado con la conmutación y enrutamiento para que se incorporen a la parte óptica de la red. Así pues, la capa de red óptica cumple una función importante en la capa de transporte.

Los principales componentes de la red *WDM* son los terminales de línea óptica (OLTs, *Optical Line Terminals*), los multiplexores ópticos de adición/extracción *ROADM* y las conexiones cruzadas ópticas (*OXCs*) interruptores de circuitos ópticos (*OCS*). La función de las OLTs es multiplexar/desmultiplexar longitudes de onda en los extremos de la red en una señal compuesta [30], es importante resaltar que las redes ópticas descritas anteriormente y sus componentes son muy importantes en el desarrollo del modelo óptico del presente trabajo de grado.

Capítulo 2

Características del Modelo Cognitivo y Balanceo de Carga

En la actualidad los operadores de las redes de comunicaciones se tienen que adaptar a la gran cantidad de datos que circula por la red, este incremento puede ocasionar pérdida de información debido a retardos ocasionados por problemas de congestión en los nodos de la red [1]. Para afrontar los problemas de congestión se han empleado mecanismos cognitivos que han ayudado a que la red permanezca en un estado estable [1], logrando que en el caso de que se presente algún cambio brusco el mecanismo cognitivo empiece a operar haciendo que la misma red sea capaz de detectar los problemas que se están presentando en ella y así poder dar soluciones a los mismos. De esta forma las investigaciones enfocadas en buscar soluciones a estos inconvenientes mediante mecanismos cognitivos han demostrado que pueden ayudar a mejorar el desempeño de la red [34].

Teniendo en cuenta lo anterior, se hace necesario tener una base conceptual de los diferentes modelos cognitivos que se han utilizado en algunos trabajos, reali-

[1] Estado estable: De acuerdo a la literatura revisada sobre redes cognitivas y ciclo cognitivo es un punto de la red, el cual es denominado ”óptimo”, es decir que funciona con una interrupción mínima. Cuando el rendimiento de la red tiene una degradación, a causa de algún problema, se utiliza algún mecanismo o forma para retornar la red a este estado, el cual se le llama estado estable.

zando una descripción de los conceptos que ayudan a lograr la implementación del mecanismo cognitivo en una red óptica definida por software. De igual forma, se hace necesario tener una base teórica de algunos trabajos relacionados con balanceadores de carga ya que se requiere de su uso para desarrollar el mecanismo cognitivo.

2.1. Cognición

La cognición investiga cómo diferentes procesos mentales influyen en la planificación de las acciones, solución de los problemas y toma de decisiones. Varias investigaciones están enfocadas en predecir el complejo comportamiento de los seres humanos. En algunos casos se han planteado modelos para ayudar a predecir este comportamiento, pero se necesita que los modelos puedan distinguir diferentes factores de influencia que pueda tener cada persona, esto hace que predecir el comportamiento tenga un alto grado de complejidad [35].

En la actualidad los modelos cognitivos no solo son utilizados para poder analizar el comportamiento de los seres humanos, sino que también se han utilizado modelos cognitivos en diferentes áreas de la ingeniería tales como la electrónica, telecomunicaciones, sistemas, entre otras. Estos modelos han ayudado a que las redes de comunicaciones mejoren su eficiencia, se puedan corregir problemas de congestión y ayuden a los operadores de la red a tener un sistema autónomo que se auto-gestione y no requiera en su totalidad de la intervención humana, haciendo que este tipo de sistemas tomen decisiones propias en caso de que se pueda presentar una eventual falla [36].

2.1.1. Definición

En [37] lo definen como "*cualquier forma de procesamiento de información, operación mental o actividad intelectual tales como pensar, razonar, recordar, imaginar o aprender*".

Desde hace mucho tiempo se ha mencionado el término cognición en el ámbito de las telecomunicaciones, entre los primeros trabajos que se desarrollaron está el de Mitola [38] quien propone desarrollar radios cognitivos (CR, *Cognitive Radio*) con el fin de darle cierta inteligencia a los sistemas inalámbricos. Ahmad [34] realiza una prueba inicial de redes cognitivas definidas por software (SDCoN, *Software Defined Cognitive Networking*) para la asignación dinámica de espectro, ya que por medio de radios definidos por software (SDR, *Software Defined Radio*) que están conectadas a un proceso cognitivo son reconfiguradas dependiendo de los requerimientos que tenga el usuario para así realizar la nueva asignación de espectro y mejorar la experiencia de los usuarios.

2.1.2. Mecanismo Cognitivo

Se define como "*las herramientas que sirven para acelerar el proceso de la mente o el conocimiento*", que en este caso son operaciones cognitivas, de esta forma estos procesos van a ser representados por medio de un algoritmo el cual es capaz de monitorizar, analizar, planificar, ejecutar y que a su vez es el responsable de mantener la red en un estado estable, dado que, si se presenta una falla en la red, el mecanismo tiene que actuar y afrontar el problema que se esté presentando para así encontrar la mejor solución [36]. Ahora bien, en las redes tradicionales y sistemas centralizados como las redes definidas por software (SDN, *Software Defined Network*) se han empleado algunos mecanismos para el control de congestión, pero son muy pocos los trabajos que están enfocados en soluciones a redes ópticas definidas por software (SDON, *Software Optical Defined Network*), surgiendo la necesidad de realizar esta investigación.

2.1.3. Ciclo Cognitivo

Las redes para observar y analizar su entorno necesitan de procesos de control, estos procesos están determinados por una serie de fases que a su vez está compuesto por un ciclo, que en las telecomunicaciones es conocido como ciclo de cognición o ciclo cognitivo, este concepto nace hace varios años con Mitola

[39] quien propone adaptar procesos cognitivos a los sistemas de radio, con el fin de otorgarle a estos la posibilidad de tener cierto conocimiento, para apoyar el razonamiento automatizado sobre las necesidades de los usuarios. Mitola propone el uso del ciclo de cognición para observar el entorno por medio de seis fases, ver figura 2.1 [40], observación, orientación, planificación, decisión, actuación y por último aprendizaje, con el fin de poder recopilar información del entorno para después tomar una decisión que pueda mejorar el desempeño de la red.

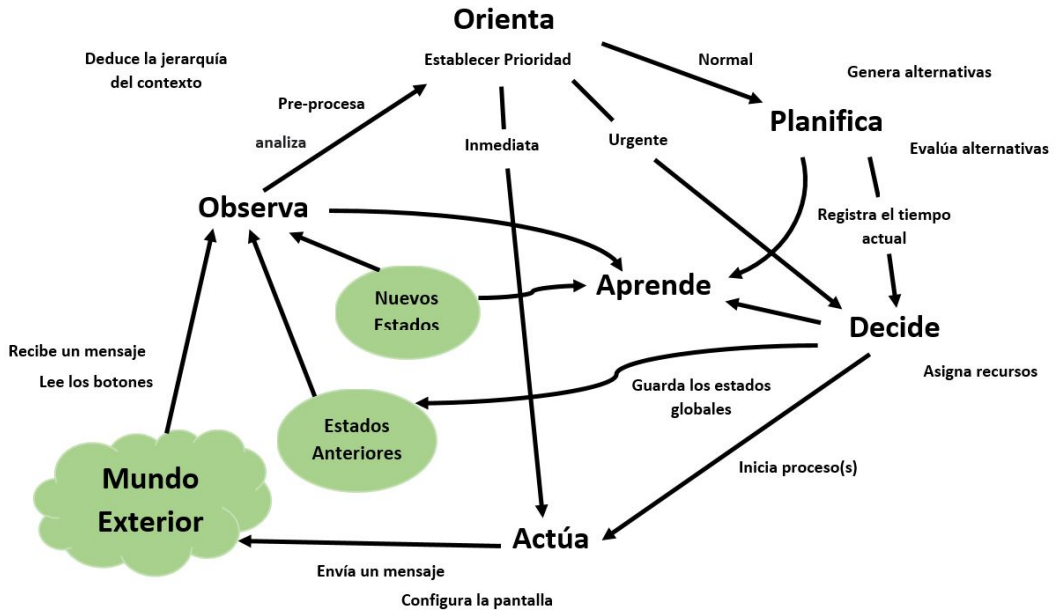

Figura 2.1: Ciclo cognitivo. Modificada a partir de [40]

2.1.4. Redes Cognitivas

Al pasar los años, las redes han tenido que experimentar varios cambios debido a la gran cantidad de información que circula en ellas, ya no es suficiente con tener sistemas de gran capacidad de procesamiento de datos si estos no son capaces de optimizar los recursos, así pues, nace la necesidad de tener sistemas que cuenten con cierta inteligencia para que la información que circula por red se pueda clasificar, ordenar, guardar, de modo que, no se deba procesar nuevamente con el fin de disminuir el gasto computacional de la red. Por esta razón nace el

termino de redes cognitivas (CN, *Cognitive Network*) las cuales son redes capaces de observar, aprender y tomar decisiones dependiendo del estado de la red, también estas decisiones pueden ser guardas en bases de datos las cuales son analizadas en caso de que la falla o evento ya haya ocurrido, el sistema puede tomar la misma decisión que tomo en esa ocasión para así disminuir los recursos de la red y hacerla más eficiente [1] [34].

Definición En DySPAN 2005 Thomas [41] Propone la siguiente definición para una red cognitiva. "*Una red cognitiva tiene un proceso cognitivo que puede percibir las condiciones actuales de la red y luego planificar, decidir y actuar sobre esas condiciones. La red puede aprender de estas adaptaciones y usarlas para tomar decisiones futuras, todo teniendo en cuenta los objetivos de extremo a extremo*".

Una red cognitiva toma decisiones bajo las condiciones de su estado actual y conociendo las decisiones de los resultados pasados, genera informes con el fin de almacenarlos en bases de datos para que puedan ser utilizados en el futuro [42]. De igual forma se han realizado otros trabajos que también mencionan las redes cognitivas como radios cognitivos [38] [39], radios inteligentes [43], antenas inteligentes [44], paquetes cognitivos [45], teniendo cada uno de estos un significado distinto, esto hace que se presente una discusión en su verdadero significado, pero el aspecto común que se presenta en cada una de las definiciones es la auto-modificación [41].

2.1.5. Trabajos Sobre Redes Cognitivas

A lo largo de los años se han propuesto algunas arquitecturas para redes cognitivas, las cuales son utilizadas en redes inalámbricas o redes ópticas, entre otras. Algunas de estas arquitecturas utilizan la información que es obtenida por medio de módulos para luego ser analizada y posteriormente poder tomar una decisión que pueda mejorar el desempeño de la red o poder ayudar a la red a que regrese a un estado óptimo. A continuación, se muestran algunos proyectos que están relacionados con redes cognitivas.

FOCALE (*Foundation, Observation, Comparison, Action, Learn, rEason*), se representa por seis elementos dado que estos describen las claves principales requeridas para soportar las redes autónomas. FOCALE es un sistema de lazo cerrado que compara el estado actual de los elementos de gestión con un estado deseado, el sistema está monitoreando constantemente los cambios que se puedan presentar en sí mismo y en su entorno, cuando detecta un cambio esta información es analizada para asegurar que los objetivos todavía se estén cumpliendo. Si es así, el sistema sigue monitoreando, pero de no ser así, hace una planificación en caso de que los objetivos puedan ser amenazados, ejecuta los cambios y observa que las acciones tomadas fueron las correctas, para asegurarse que el sistema ya no presente amenazas [46] [47].

En la figura 2.2, se puede observar la arquitectura basada en políticas de control para lograr la gestión autónoma del sistema, también se ver el modelo de información DEN-ng, el cual especifica un único modelo de información que es utilizado para derivar múltiples modelos de datos. Este modelo es importante ya que incorpora una gran diversidad de información de gestión ayudando a mejorar el manejo de los datos [46].

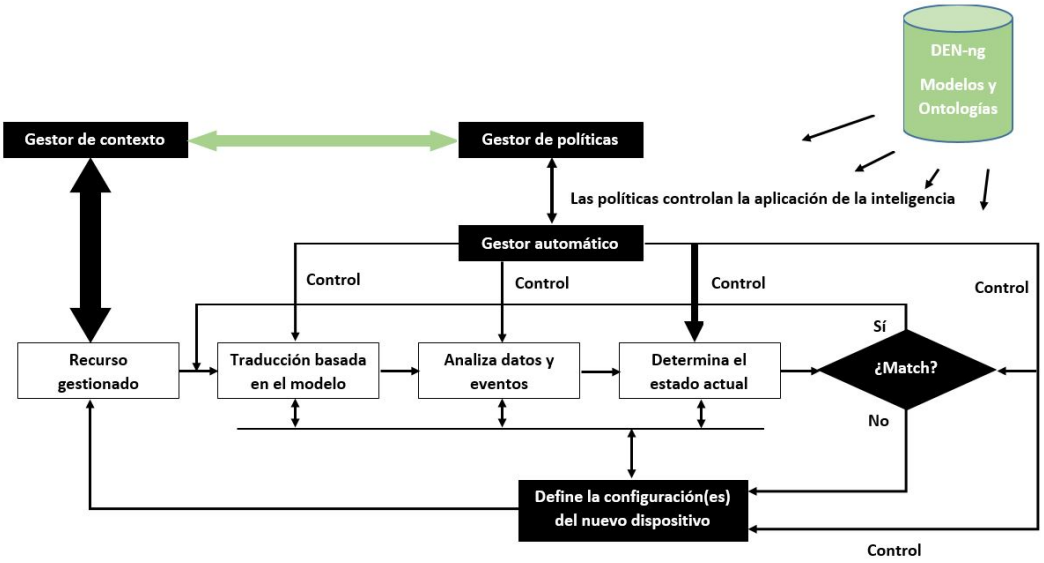

Figura 2.2: Arquitectura FOCALE. Modificada a partir de [46]

CogNET (*Cognitive Complete Knowledge Network*), es una arquitectura empleada para mejorar los recursos de la red, es propuesta para redes inalámbricas, la finalidad de CogNET es tomar la información por medio de radios cognitivos que están conectados a cada una de las últimas tres capas del modelo OSI, esta información es compartida entre las capas y a su vez está conectada a un motor cognitivo que es el encargado de procesar esta información y además es el encargado de adjuntar esta información a una base de datos interna, ver figura 2.3. La base de datos interna es la encargada de estimar la información que está siendo enviada hacia el motor cognitivo bajo cuatro estimadores, estimador de tipo de frecuencia, estimador de probabilidad de tasa de datos de cada transmisión y paquetes perdidos, finalmente el estimador de incremento del Througput. La información obtenida por cada una de los estimadores es recolectada para luego ser compartida a las otras capas, y así ser utilizada para funciones de enrutamiento [48] [49].

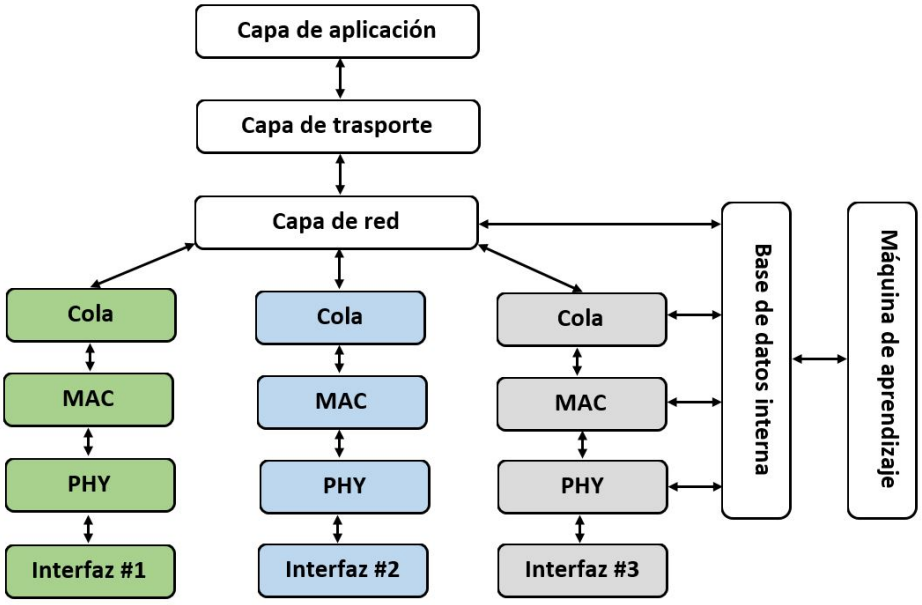

Figura 2.3: Arquitectura CogNET. Modificada a partir de [48]

CHRON (*Cognitive Heterogeneous Reconfigurable Optical Network*), es un proyecto que propone una estrategia para tener un control cognitivo de la red, sacando el mejor provecho de los recursos para luego combinarlos con tecnologías

de transmisión y conmutación. El enfoque de este proyecto es reconfigurar las redes ópticas de trasporte, donde una topología virtual reconfigurable es establecida mediante conexiones ópticas, llamadas *lightpath*. Los *lightpath* tienen la capacidad de acomodar las nuevas demandas de tráfico y dependiendo de las condiciones de la red son capaces de reconfigurar la topología virtual[2], además se pueden adicionar o remover en caso de que las demandas del usuario lo requieran [42] [50].

Para CHRON se propone una arquitectura, ver figura 2.4, que tiene como elemento principal el sistema de decisión cognitiva (CDS, *Cognitive Decision System*) quien determinan el manejo de las demandas de tráfico o los eventos de la red, para optimizar su uso y su rendimiento a partir de los estados actuales y pasados. El CDS igualmente le indica al plano de control que configure los elementos de la red. La cognición puede ser distribuida o centralizada, el proyecto está enfocado en su mayoría a una cognición centralizada. El CDS es asistido por el sistema de control y gestión (CMS, *Control and Management System*) el cual actualiza el CDS con el estado de la red y los recursos disponibles para garantizar que las decisiones tomadas por el CDS a los nodos interesados se realicen correctamente, además de observar el proceso de configuración de los dispositivos notificando cualquier anomalía o malfuncionamiento. La arquitectura también incluye los elementos adaptables de software (SAE, *Software Adaptable Element*) quienes le otorgan a la red la capacidad de modificar su configuración actual lo que la hace una red adaptativa, estos se configuran dependiendo de las decisiones del CDS y también está el sistema de monitoreo de red (NMons, *Network Monitor System*) que proporciona el estado del tráfico y las medidas del rendimiento óptico al CDS. Tanto las funcionalidades de monitoreo y adaptabilidad de cada elemento son manejadas mediante una capa física de gestión. De esta forma es como dependiendo de los estados de los elementos el CDS pueden planificar, decidir y actuar, mientras se tengan en cuenta los objetivos de extremo a extremo [51] [42].

[2]Topología virtual: Una topologia virtual es construida mediante un conjunto de lightpath como un grafo formado por nodos que corresponden a la combinación del *switch* óptico y el nodo IP.

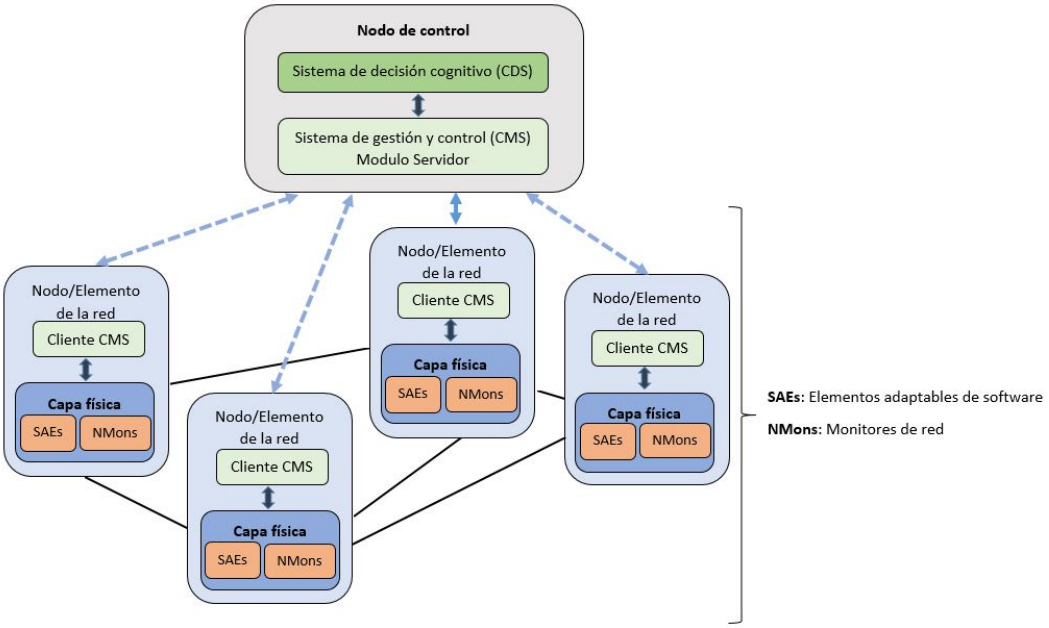

Figura 2.4: Arquitectura CHRON. Modificada a partir de [42]

Considerando los trabajos en redes cognitivas expuestos anteriormente se realiza una comparación entre ellos con el fin de definir cual es la arquitectura que mejor se adapta a la solución del presente trabajo de grado, esto se muestra en la tabla 2.1. Teniendo en cuenta el enfoque del presente trabajo de grado en las redes ópticas definidas por software la arquitectura que mejor se asemeja es la CHRON por su tipo de red y cognición.

2.1.6. Redes Cognitivas Definidas por Software

Las redes cognitivas se proponen para automatizar las funcionalidades de la red, hacer que los sistemas ya no dependan en su totalidad de un administrador, si no que el mismo sistema pueda realizar una autoconfiguración, para poder mantenerse en un estado óptimo. SDN es una arquitectura que propone la centralización de la red por medio de la separación del plano de datos y del plano de control, con el fin de brindar una inteligencia a la red, por medio de la configuración de los dispositivos que estén conectados a la red [34].

Arquitectura	Descripción	Tipo de red	Tipo de cognición	Finalidad
FOCALE	Detecta los cambios de su entorno para asegurar que sus objetivos se cumplan	No define	Distribuida	Gestión de recursos computacionales
CogNET	Utiliza la información de cuatro estimadores para funciones de enrutamiento	Redes tradicionales	Centralizada o Distribuida	Mejorar los recursos de la red
CHRON	Verifica que las condiciones de la red sean las óptimas	Redes definidas por software	Centralizada o Distribuida	Mejorar los recursos de la red

Tabla 2.1: Comparación de arquitecturas cognitivas. Elaboración propia

En *"Towards Software Defined Cognitive Networking"* [34], se propone una prueba inicial de redes cognitivas definidas por software (SDCoN, *Software Defined Cognitive Networking*) por medio de un testbed[3] para la asignación dinámica de espectro para los usuarios, donde se simula una arquitectura celular SDN en tiempo real por medio de una plataforma de estudio de acceso libre inalámbrico (WARPC, *Wireless Open-Access Research Platform*). Por otra parte, se le agregan procesos cognitivos al plano de control para mostrar su factibilidad. Para esto, se describe la arquitectura SDN con su correspondiente componente SDCoN, donde la información de la red es obtenida por medio de sensores o Interfaces de programación de aplicaciones (APIS, *Application Programming Interfaces*) a través de elementos de la red configurables llamados redes adaptables de software (SAN, *Software Adaptable Network*), para que luego pueda ser procesada por el controlador *Floodlight OpenFlow*.

Dado que se tiene una red cognitiva también se habla sobre el ciclo cognitivo, donde se nombran siete fases (entorno, registro, planificación, políticas, aprendizaje, decisión y actuación), el cual está enfocado en el aprendizaje, donde este

[3]Testbed: Plataforma para experimentación de proyectos de gran desarrollo.

se realiza por medio del registro del entorno, de esta forma cuando se detecta que la calidad del servicio de un usuario está decayendo por debajo del umbral establecido se debe de iniciar una acción. Esta acción es determinada por el perfil del usuario y las políticas del proveedor, estos pasos constituyen la planificación, políticas y aprendizaje del ciclo cognitivo. Las fases de decisión y actuación solo se aplican si se tienen los recursos disponibles (slots de frecuencia libres). Es de destacar que este artículo aporta en gran medida al desarrollo del presente trabajo de grado, por el uso del ciclo cognitivo, además de su enfoque en las redes cognitivas definidas por software.

2.2. Balanceador de carga

El balanceo de carga es un método para mejorar el rendimiento y la disponibilidad de la red, reducir al mínimo los retardos y evitar la congestión. En el presente documento se estudia el balanceo de carga como parte de la solución al mecanismo cognitivo para control de congestión. El balanceo de carga se aplica al plano de datos de la red SDN. La flexibilidad del balanceador de carga se prueba usándolo en la topología de red NSFNET, como complemento al modelo cognitivo, para que de esta forma se pueda crear un mecanismo para control de congestión eficiente.

La función básica de SDN es separar el plano de control del plano de datos mediante funciones de control basadas en una representación abstracta de la red. Las funciones de red se implementan eliminando las decisiones de control, como el enrutamiento de los dispositivos de hardware y habilitando tablas de flujo programables mediante un protocolo estandarizado, como *OpenFlow*. La abstracción de la red se logra mediante el uso de un controlador SDN centralizado lógico que define el comportamiento del plano de datos, cuando se inicia un nuevo flujo en el plano de datos y no existe una política de enrutamiento dentro de la tabla de flujo, el dispositivo de reenvío envía el primer paquete de ese flujo al controlador. El controlador entonces define la ruta de reenvío relevante para ese flujo. Generalmente, se define una política de reenvío por flujo. Dos o más políticas, pueden ser definidas con propósitos de respaldo. Por lo tanto, hay demasiados

paquetes que utilizan un flujo en particular, este puede experimentar una sobrecarga y causar retrasos reduciendo la eficiencia general de la red. Así pues, el volumen cada vez mayor del tráfico requiere métodos eficientes de ingeniería y gestión del tráfico para garantizar la disponibilidad, escalabilidad y fiabilidad de la red [52].

El equilibrio de carga es un método de gestión del tráfico entrante que consiste en distribuir y compartir la carga de manera equitativa entre los recursos de la red disponibles para mejorar la disponibilidad, reducir la latencia y la utilización del ancho de banda. En las redes tradicionales, un balanceador de carga es un dispositivo diseñado en un hardware determinado [53]. Los balanceadores de carga son costosos y difieren según el proveedor. En SDN, son códigos de programa que pueden ser fácilmente implementados en el controlador SDN para administrar eficientemente la carga de la red. La mayoría de los controladores SDN, como por ejemplo, *Onos, Ryu, Pox, floodlight*, vienen con balanceadores de carga de red estáticos incorporados con políticas de equilibrio predefinidas, como por ejemplo, Round Robin y Random.

En SDN, los balanceadores de carga determinan inteligentemente qué dispositivo dentro de toda la red es el mejor para procesar un paquete de datos entrante. Para ello, se requieren algoritmos orientados a distribuir las cargas de una manera específica. Los algoritmos varían entre ellos, dependiendo de si una carga se distribuye en la red o en la capa de aplicación. Los algoritmos seleccionados influyen en qué tan eficientes son los mecanismos de distribución de la carga y, por consiguiente, en el rendimiento y la continuidad de las actividades.

La optimización del rendimiento de la red en SDN, como en cualquier otra red, requiere la necesidad de encontrar las mejores estrategias de balanceo de carga. En la literatura se han propuesto varias técnicas para optimizar el flujo de la red y para lograr un equilibrio de carga inteligente entre los diferentes enlaces, servidores, nubes y controladores.

En la mayoría de los casos, el balanceo de carga en el controlador SDN se implementa utilizando dos enfoques, el balanceo de carga sin estado y el balanceo de carga con estado completo [54] [55]. En el enfoque sin estado, un controlador no monitoriza el estado de la red y sólo equilibrará la carga del tráfico en

un momento determinado. Este método funciona bien cuando las demandas de tráfico se conocen de antemano. En el enfoque con estado, el controlador hace un seguimiento del estado de la red y realiza el balanceo de carga según sea necesario. Aunque se prefiere el enfoque de estado completo, es costoso porque el controlador debe mantener los estados por flujo para la tabla de flujo [56]. Esto puede llevar a veces a una sobrecarga del controlador, lo que en última instancia puede causar indisponibilidad y retrasos en la red.

A continuación, se define el balanceo de carga en el caso de que el plano de control consista en varios controladores SDN. El esquema de despliegue de multicontroladores fiable y consciente de la carga (RLMD) se propone en [57]. Según los autores, el objetivo principal del RLMD es resolver la disparidad de carga del tráfico de controladores y los problemas de precisión del plano de control que surgen cuando los planos de control y de datos se dividen en la red SDN de multicontroladores. El RLMD como esquema progresivo asegura que la asignación equitativa de las cargas de tráfico de los controladores y el despliegue efectivo y fiable de los controladores estén en su lugar. Los autores también idearon un algoritmo de partición de dominio múltiple (MDP) que es capaz de hacer coincidir los controladores y conmutadores en función de la tasa de equilibrio de carga del controlador y la atracción del nodo. Cuando se probó frente a otras estrategias típicas, su algoritmo logró equilibrar mucho mejor la asignación de las cargas de tráfico del controlador, y también mejoró la fiabilidad del plano de control.

En [58] se propone una estrategia basada en la migración de los conmutadores llamada ElastiCon. ElastiCon equilibra periódicamente la carga de los controladores en un pool, y asegura un buen rendimiento de la red en todo momento independientemente de la dinámica del tráfico. La desventaja de ElastiCon es que en algunos casos lleva a una alta sobrecarga de la red. Este proceso reduce los controladores sobrecargados, evitando así la degradación del rendimiento de la red.

Los autores de [59] idearon un esquema de balanceo de carga basado en grupos de *switches* de arquitectura para controladores múltiples, con el objetivo de mejorar la eficiencia del tiempo en el proceso de balanceo de carga. El esquema propuesto utiliza un algoritmo de selección de interruptores y un algoritmo de se-

lección de controladores objetivo para resolver el problema de la oscilación de la carga entre los controladores.

Teniendo en cuenta los referentes teóricos descritos en los capítulos 1 y 2, que son necesarios para comprender como se elabora el mecanismo cognitivo para control de congestión en una red óptica definida por software se procede en el siguiente capitulo a describir las herramientas utilizadas para su elaboración así como la metodología de simulación para finalmente obtener los resultados de las simulaciones de los dos casos de estudio utilizados para la evaluación del desempeño de este mecanismo.

Capítulo 3

Herramientas, simulación Y análisis de resultados

En este capítulo se establecen las herramientas necesarias para el desarrollo del presente trabajo de grado, en el cual, se describe de forma detallada los elementos principales que se necesitaron para elaborar el mecanismo cognitivo, de igual forma se describe la elaboración de la topología basada en una red óptica definida por software por la cual se envían dos tipos de tráfico (Ratón y elefante) con características IP para así, evaluar el desempeño del mecanismo cognitivo cuando se enfrenta a un problema de congestión.

La topología de red propuesta para el desarrollo de este trabajo es la NSFNET, la cual será simulada por medio de la herramienta *Mininet*. Para el desarrollo del mecanismo cognitivo se utiliza la API de ONOS y finalmente, los flujos que se envían a través de la red se hacen por medio de Iperf, teniendo en cuenta esto, a continuación se hace una descripción de como esta compuesto este capítulo.

- Inicialmente se hace una descripción de las herramientas necesarias tanto para el desarrollo del mecanismo cognitivo, como para la creación de las topologías y la elaboración de la SDON.
- En seguida se hace una descripción de la metodología de simulación seleccionada para el desarrollo de este trabajo y así evaluar el desempeño

por medio de la probabilidad de bloqueo y el retardo extremo a extremo.

- Finalmente se realiza el análisis de los resultados (parámetros de desempeño) obtenidos para la SDON sin el mecanismo cognitivo y con el mecanismo cognitivo, para así poder evaluar el desempeño del mecanismo cognitivo para control de congestión en una red de este tipo.

El desarrollo que se elabora en este capítulo realiza un aporte definitivo a la solución de los objetivos específicos expuestos en el anteproyecto: "**Diseñar un mecanismo cognitivo basado en las características del ciclo cognitivo para control de congestión**", "**Evaluar mediante simulación el desempeño del mecanismo cognitivo en la SDON de prueba**" y por ende, dar solución al objetivo general de este trabajo de grado: "**Implementar un mecanismo cognitivo para control de congestión en una red óptica definida por software**".

3.1. Herramientas de simulación

Para el desarrollo del modelo de simulación se requiere de la integración de algunas plataformas software que son necesarias para la construcción e implementación de la SDON de prueba en la cual se implementa el mecanismo cognitivo para control de congestión. A continuación se realiza la descripción de las plataformas software que se utilizan para la elaboración de la red de prueba.

3.1.1. Entornos de Virtualización

VMware Es un software que se utiliza para la virtualización de sistemas, también abarca los recursos informáticos, la nube, la red, la seguridad y el área de trabajo digital, se utiliza para la elaboración de diversos entornos virtuales basados en Linux como Ubuntu, Debian, Cent os, entre otros, así mismo la elaboración de entornos basados en Windows y otros sistemas operativos, de igual forma en las versiones más recientes permite la creación de entornos virtuales para dispositivos móviles mediante la virtualización de Android. Este software también

permite la configuración de los componentes hardware de los dispositivos tales como discos duros, tarjetas de red, tarjetas de memoria RAM, procesadores, memorias USB, entre otros.

Citrix XenServer Citrix está basado en software de código abierto, es una plataforma de virtualización de servidores administrada, completa e integrada en el *hipervisor Xen*. La tecnología Xen proporciona aislamiento seguro, control de recursos, garantías de calidad de servicio y migración de máquinas virtuales en caliente. XenServer está diseñado para una gestión eficiente de los servidores virtuales de Windows y Linux.

Microsoft Hyper-v Es un software de virtualización que se encuentra disponible de forma nativa en los sistemas operativos versión Pro y Server. Gracias a esta herramienta se pueden virtualizar sistemas operativos con todo el hardware como si fuera máquinas reales, tal y como hacen VirtualBox y por supuesto VMware. Gracias a esta aplicación no hay necesidad de instalar software externo en el sistema Windows, una desventaja de este entorno es que no cuenta con las suficientes características para emular versiones recientes de Linux.

VirtualBox Es un software de virtualización para arquitecturas x86/amd64. Por medio de esta aplicación es posible instalar sistemas operativos adicionales, dentro de otro sistema operativo anfitrión cada uno con su propio ambiente virtual. Entre los sistemas operativos soportados (en modo anfitrión) se encuentran GNU Linux, MacOS X, OS 2 Warp, Genode, Windows y Solaris OpenSolaris, y dentro de ellos es posible virtualizar los sistemas operativos FreeBSD, GNU Linux, OpenBSD, OS 2 Warp, Windows, Solaris, MS-DOS, Genode y muchos otros.

El entorno virtual seleccionado fue VMware en su versión gratuita, tal y como se aprecia en la figura 3.1, puesto que, fue el software que más se adaptó a las configuraciones de los computadores utilizados, su desempeño fue destacable con respecto a los demás entornos de virtualización mencionados, además, cabe destacar que la aplicación tiene una buena adaptación con *mininet*, que es el

software elegido para realizar la emulación de la red SDON. VMware Workstation aloja la máquina virtual configurada bajo un sistema operativo *Linux* (Ubuntu 18).

Dentro del desarrollo del trabajo de grado se evidenció la necesidad de crear una única máquina, puesto que, todas las herramientas requeridas para el desarrollo se logran instalar en conjunto con esta máquina, de esta forma, también se aprovecha una ventaja que brinda el entorno de virtualización.

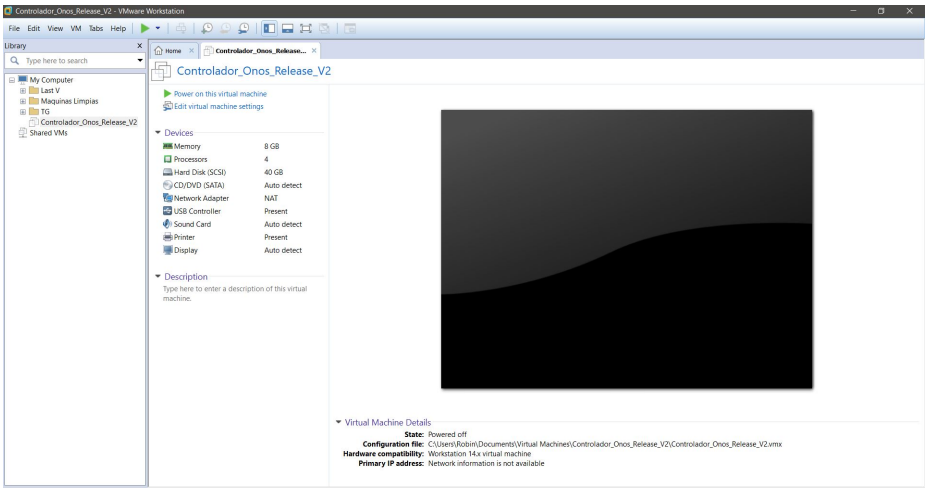

Figura 3.1: Interfaz VMware Workstation. Elaboración propia

3.1.2. Software de Emulación

Mininet *Mininet* es un software de emulación que proporciona un banco de pruebas virtual y un entorno de desarrollo para redes definidas por software (SDN). *Mininet* permite el desarrollo de SDN en cualquier PC, y los diseños de SDN pueden moverse sin problemas entre *Mininet* (permitiendo un desarrollo económico y optimizado), el hardware real que se ejecuta a velocidad de línea en implementaciones en vivo [15]. *Mininet* permite:

- Creacion rápida de prototipos de redes definidas por software
- Pruebas de topología complejas sin la necesidad de conectar una red física

- Varios desarrolladores simultáneos para trabajar de forma independiente en la misma topología

Mininet es entonces una herramienta que permite la emulación de redes definidas por software que incorpora una colección de host, *switches*, controladores y enlaces virtuales por medio del protocolo *OpenFlow*, el cual permite hacer pruebas sin tener que utilizar dispositivos reales. En los últimos años *Mininet* ha sido utilizada para la elaboración de diferentes trabajos de grado, así como de artículos relacionados con SDN por su flexibilidad, interactividad y escalabilidad que la hacen una herramienta muy atractiva y fácil de utilizar. *Mininet* al utilizar una arquitectura *OpenFlow* le permite obtener información de los *switches* que son creados bajo este protocolo (switches OpenFlow) mostrando así los diferentes flujos que tiene un nodo, su configuración, los puertos, los enlaces nodo a nodo, entre otras características [60].

Para crear topologías por medio de *Mininet* se puede hacer de dos formas, una es por medio de un script de *Python* en la cual se definen los hosts, *switches*, IP, o dirección IP del controlador, entre otras configuraciones y la otra, por medio de un simulador de interfaz gráfico que proporciona los dispositivos necesarios para elaborar la topología llamado MINIEDIT. Es de destacar que por medio del script de *Python* se pueden acceder a los recursos de *Linux* utilizando algunas de sus librerías.

Para el desarrollo de la topología NSFNET, la cual es utilizada en este trabajo de grado se define por medio de MININEDIT y se culmina su configuración a través del script de *Python*.

En la actualidad existen entornos similares a *Mininet* que ofrecen características propias para trabajar bajo SDN, sin embargo, los desarrolladores e investigadores han hecho mucho más énfasis en el programa *Mininet* puesto que, es el software recomendado por la Open Networking Foundation[1], además, la mayoría de las características compatibles están desarrolladas hacía *Mininet*.

[1]La Open Networking Foundation (ONF) es un consorcio liderado por operadores sin fines de lucro que impulsa la transformación de la infraestructura de red y los modelos comerciales de los operadores.

3.1.3. Controlador OpenFlow

Un controlador OpenFlow ofrece una interfaz de programación para los conmutadores OpenFlow de tal forma que, las aplicaciones de gestión, a través de la misma, pueden realizar tareas de gestión y ofrecer nuevas funcionalidades [61]. El controlador puede ser descrito como un software o conjunto de software que puede ofrecer:

- Gestión del estado de la red, que implica una base de datos. Estas bases de datos sirven como un repositorio para la información de los elementos de red gestionados, incluyendo el estado de la red, alguna información de configuración temporal e información sobre la topología de la red.
- Un modelo de datos de alto nivel que captura las relaciones entre los recursos gestionados, las políticas y otros servicios prestados por el controlador. En muchos casos estos modelos de datos se construyen utilizando el lenguaje de modelado Yang[2] [62].
- Un mecanismo de descubrimiento de dispositivos, topología y servicio; un sistema de cálculo de ruta y, potencialmente, otros servicios de información centrados en la red o en los recursos.
- Una sesión de control segura sobre el Protocolo de Control de Trasmisión (TCP, *Transmission Control Protocol*) entre el controlador y los agentes asociados en los elementos de la red, por ejemplo con el uso del protocolo de Seguridad en la Capa de Transporte (TLS, *Transport Layer Security*).
- Un protocolo basado en estándares (OpenFlow) para obtener el estado de la red impulsado por las aplicaciones de los elementos de red.
- Un conjunto de APIs, a menudo RESTful[3] que exponen los servicios del controlador a las aplicaciones de gestión. Esto facilita la mayor parte de la

[2]YANG es un lenguaje de modelado de datos que se utiliza para modelar la configuración y los datos de estado manipulados por el protocolo de configuración de red (NETCONF), las llamadas a procedimientos remotos NETCONF y las notificaciones NETCONF.

[3]RESTful hace referencia a un servicio web que implementa la arquitectura REST.

interacción del controlador con estas aplicaciones. Esta interfaz se representa a partir del modelo de datos que describe los servicios y funciones del controlador. En algunos casos, el controlador y su API son parte de un entorno de desarrollo que genera el código de la API a partir del modelo de datos [63].

El controlador entonces es el que permite ejecutar las operaciones y funciones de toda la red SDN. Algunos controladores de código abierto existentes son: Beacon [64], Floodlight [65], NOX [66], POX [67], Ryu [68], ONOS [69] y OpenDayLight [70]. En la figura 3.2 se muestran algunas características de estos controladores [61].

	Beacon	Floodlight	Nox	Pox	Ryu	Onos	OpenDayLight
Soporte OpenFlow	OF v1.0	OF v1.0	OF v1.0	OF v1.0	OF v1.0, v1.2, v1.3	OF v1.3	OF v1.3
Virtualización	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch
Lenguaje De Desarrollo	Java	Java	C++	Python	Python	Java	Python
Provee Rest Api	No	Si	No	No	Si	Si	Si
Interfaz Gráfica	Web	Web	Python + QT4	Python + QT4, Web	Python	Web	Web
Soporte De Plataformas	Linux, Mac Os, Windows y Android	Linux, Mac Os, Windows	Linux	Linux, Mac Os, Windows	Linux, Mac Os, Windows	Linux, Mac Os, Windows	Linux, Mac Os, Windows
Soporte De Openstack	No	Si	No	No	Si	Si	Si
Multiprocesos	Si	Si	Si	Si	Si	Si	Si
Código Abierto	Si	Si	Si	Si	Si	Si	Si
Documentación	Buena	Buena	Media	Pobre	Buena	Buena	Buena

Figura 3.2: Comparación de controladores. Modificada a partir de [61]

Al realizar la comparación de los controladores y analizar las diferentes características, se concluye que *Ryu, ONOS y OpenDayLight* son los controladores más adecuados para trabajar bajo esta línea de investigación, sin embargo, a causa de que en la exploración que se realizó se encontraron algunos trabajos relacionados con *Ryu y OpenDayLigt* en la Universidad del Cauca se toma la

decisión de utilizar el controlador *ONOS* el cual es impulsado por la Open Networking Foundation.

ONOS (*Open Network Operation System*), es un controlador de código abierto que fue creado para dar soluciones SDN, fue diseñado para la construcción de soluciones de nivel de operador que aprovechen la economía de hardware y al mismo tiempo que ofrezcan la flexibilidad para desarrollar e implementar diversos servicios de red dinámicos con interfaces programáticas simplificadas. *ONOS* permite la configuración en tiempo real de la red eliminando así la necesidad de ejecutar protocolos de control y enrutamiento dentro de la red, dado que desde el controlador se pueden hacer estas configuraciones y si en un futuro se requiere hacer alguna modificación en el controlador este es capaz de realizarla en cualquier momento.

Esta plataforma incluye:

- Una plataforma y un conjunto de aplicaciones que actúan como un controlador SDN distribuido, modular y extensible.
- Gestión simplificada, configuración e implementación del nuevo software, hardware y servicios.

Para el desarrollo del mecanismo cognitivo se utiliza la versión de API de ONOS 2.4.0 llamada Uguisu en la figura 3.3 se muestra su arquitectura. Donde se puede ver que en la parte superior (Plano de aplicaciones) están las aplicaciones que se pueden crear por medio de esta API y de igual forma se encuentran las aplicaciones que están por defecto, en la parte inferior (Plano de datos) están los elementos de la red, los protocolos y los proveedores y en la parte central se encuentra el Northbound Y Southbound que son los encargados de conectar el controlador SDN con los otros planos, en medio de estos dos se encuentra el Core (Plano de Control) [69] [71].

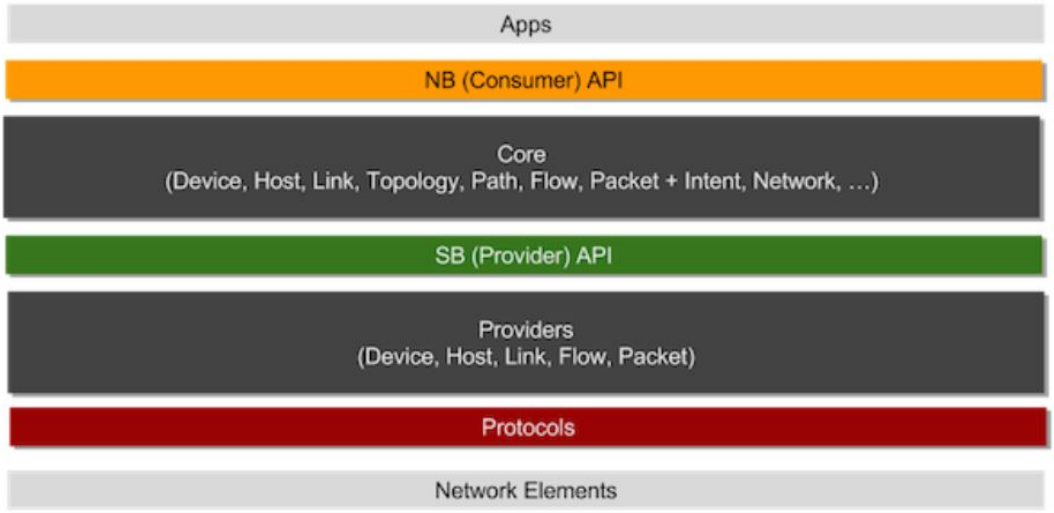

Figura 3.3: Arquitectura ONOS. Tomada de [71]

De esta forma el controlador *ONOS* se encarga de gestionar la infraestructura de la red que se utiliza en este trabajo de grado, brindando la información suficiente a la aplicación que se desarrolla por medio de la API de ONOS (Mecanismo cognitivo) y que se encuentra en el plano de aplicaciones para así poder tomar decisiones en el momento en que se presente una congestión. De igual forma el controlador toma la información de *Mininet* por medio del protocolo *OpenFlow* logrando así una visión completa del entorno de la red.

3.2. Metodología de simulación

En la figura 3.4 se puede observar el diagrama de flujo que representa la metodología de simulación la cual tiene como fin dar respuesta a la pregunta de investigación de este trabajo de grado, así mismo como a los objetivos específicos y al objetivo general [72].

Como se ve en el diagrama las fases se desarrollan de forma secuencial, pero a partir de la etapa de presentación casos de estudio, las fases siguientes se tendrán que repetir para algunos de los escenarios ya que es indispensable para el desarrollo de este trabajo de grado.

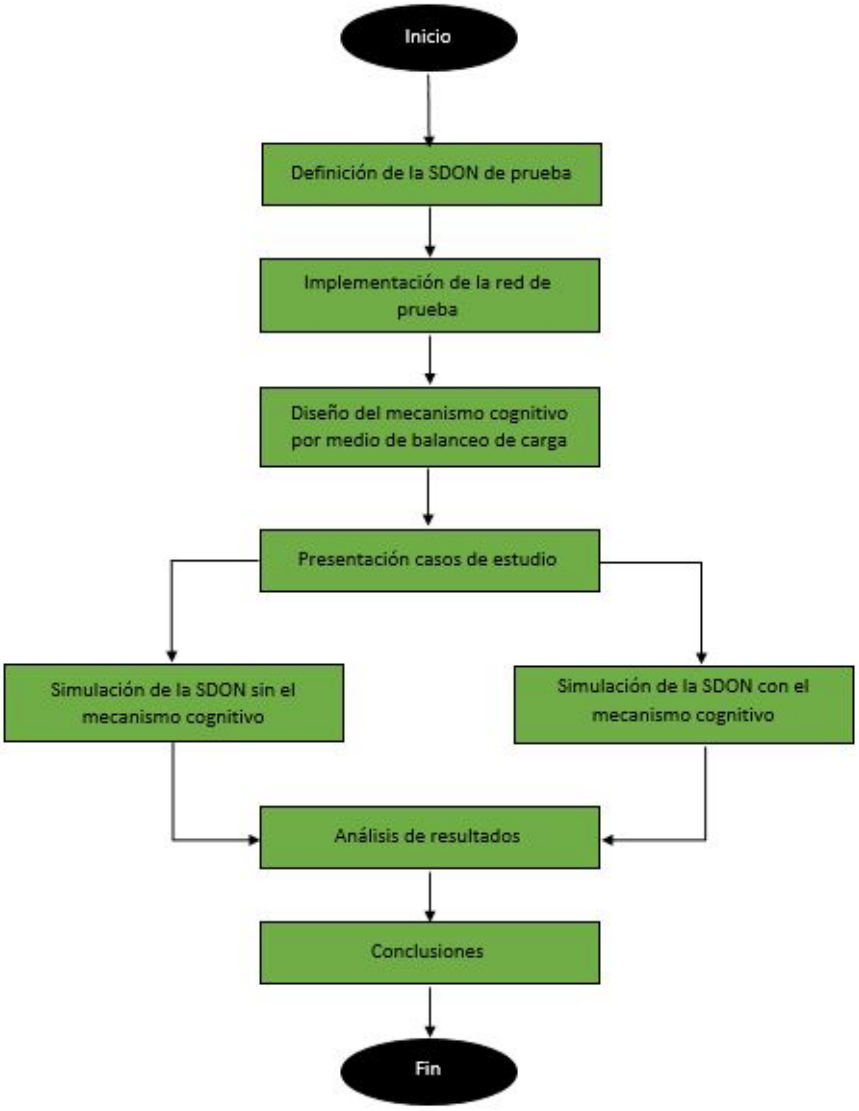

Figura 3.4: Diagrama metodología de simulación. Modificada a partir de [72]

3.3. Fase 1: Definición de la SDON de prueba

3.3.1. Caracterización de modelo óptico

Para la caracterización del modelo óptico se implementa una topología en *Mininet* que emula la estructura de un switch WDM. Esta topología define la trayectoria

que puede utilizar una longitud de onda por medio del WDM switch fabric. En la figura 3.5 se puede observar la arquitectura de este *switch*, donde se tienen tres lineas de fibra, cada linea de fibra esta acompañada de un demultiplexor seguida de tres longitudes de onda, en la parte central se encuentra el switch fabric con una topología de enmallado, que es el encargado de definir la trayectoria de cada una de las longitudes de onda, enseguida se encuentran las longitudes de onda acompañadas de un multiplexor y su respectiva linea de fibra [28].

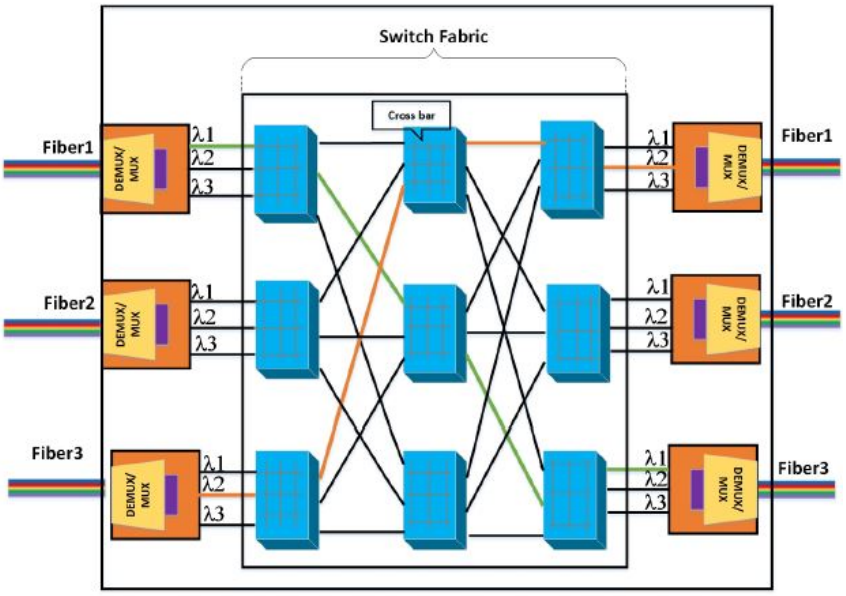

Figura 3.5: Arquitectura Clos para un switch fabric óptico. Tomada de [28]

Teniendo en cuenta que el mecanismo cognitivo es probado en la NSFNET, se utiliza un modelo OEO (Óptico-eléctrico-óptico) como se ve en la figura 3.6, los OXC son los encargados de definir las trayectorias de las longitudes de onda, seguido de esto se tiene la línea de trasmisión, lugar donde se encuentra la NSFNET que está compuesta por conmutadores eléctricos y por último se encuentra otro OXC. De esta forma se define la caracterización del modelo óptico que se implementa en el desarrollo de este trabajo de grado. Más adelante se muestran las configuraciones y su representación con los dispositivos creados en *Mininet* [28].

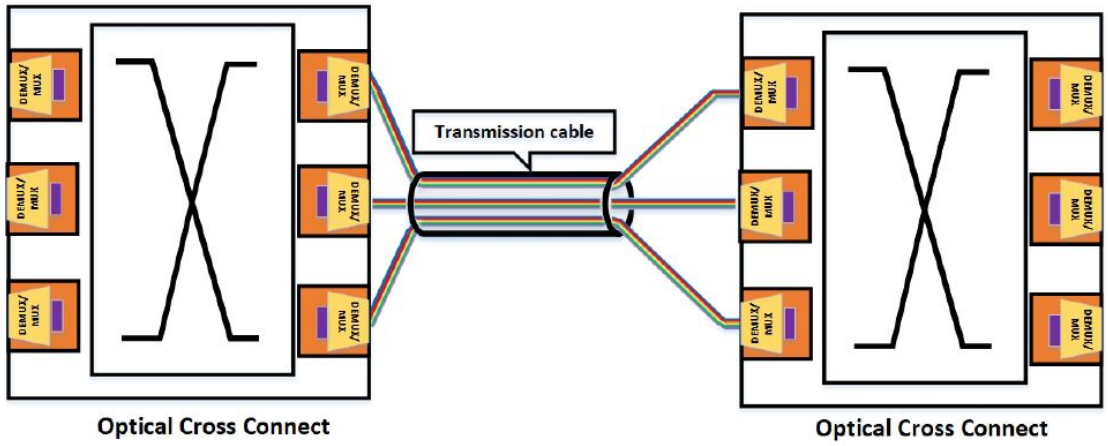

Figura 3.6: OXC con conversión OEO. Tomada de [28]

3.3.2. Diseño de topología de red NSFNET

La NSFNET (*National Science Foundation's Network*) es una red con una topología irregular dado que no tiene ninguna similitud con otro tipo de redes tales como estrella, malla, árbol, entre otras. Teniendo en cuenta las características de este trabajo de grado se utiliza esta topología de tipo WAN, la cual cuenta con 14 nodos y 21 enlaces, es de aclarar que solo se usa su topología irregular, no se consideran las capacidades de los nodos, ni las velocidades de los enlaces, ni tráfico que ha circulado por ella, entre otros aspectos. La NSFNET fue puesta en línea en 1986 y conectada a centros de súper computadoras a velocidades de 56.000 bits por segundo, debido a esto en poco tiempo la red empezó a congestionarse, en 1988 sus enlaces fueron actualizados a 1.5 megabits por segundo, varias redes de investigación y educación regionales fueron conectadas al backbone de la NSFNET, por esta razón se extendió el alcance de internet en todo Estados Unidos. En la actualidad esta red es muy utilizada para el desarrollo de trabajos de investigación, por la variedad de enlaces que se pueden utilizar para la elaboración de pruebas, en la figura 3.7 se puede observar esta topología [73][74].

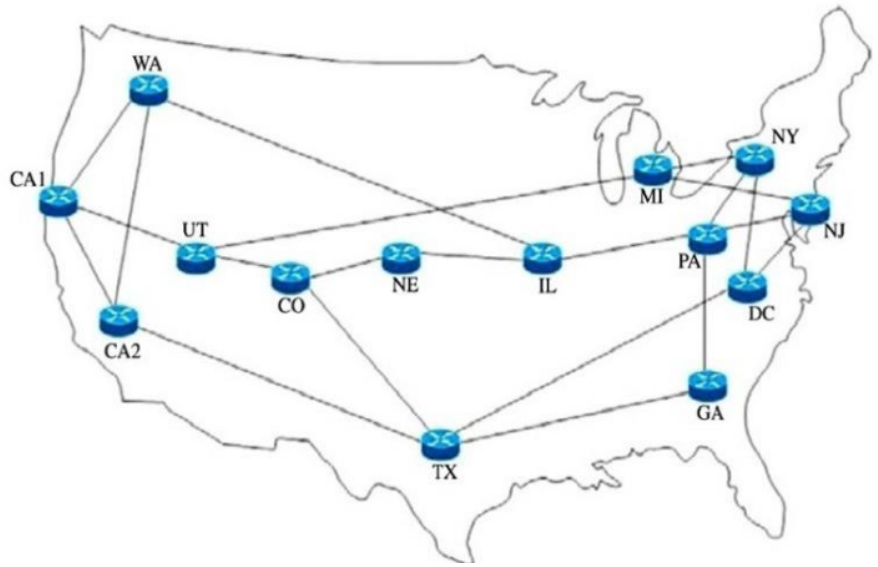

Figura 3.7: Topología de red NSFNET. Tomada de [74].

3.3.3. Flujo elefante y ratón

En los últimos años se han venido presentando varios trabajos de investigación relacionados con el tráfico elefante y ratón, se dice que más del 90 % del tráfico en una red de datacenter está compuesta por flujo ratón, el cual transporta menos del 10 % del total del número de bytes transmitidos en una red. Los flujos ratón normalmente están compuestos por tamaños por debajo de los 10KB. Los flujos elefantes son lo contrario constituyen el 10 % del tráfico, pero transportan el 90 % de los bytes transmitidos. Los flujos ratón son procesados más rápido, en cambio los flujos elefante más lento, debido a su gran tamaño pueden ocasionar retardos y por consiguiente pérdida de paquetes. Es así, como los flujos ratón se puede encontrar en tráfico HTTP, HTTPS y los flujos elefante se encuentran en tráfico de gran tamaño como por ejemplo ftp, copias de seguridad, SSH, entre otros [75][76].

Para la elaboración del tráfico que se utiliza en la SDON de prueba de este trabajo de grado se opta en usar los dos tipos de tráfico ya mencionados. El tráfico ratón se utiliza cuando la red no presenta ninguna congestión, el tamaño de estos flujos es de 1470 bytes (tamaño predeterminado por *iperf* en UDP), en cambio

para generar la congestión se envían flujos elefante con un tamaño de 65K bytes (tamaño máximo permitido por *iperf* en UDP).

A continuación se muestran los parámetros que se utilizan para evaluar el desempeño del mecanismo cognitivo. Se selecciona probabilidad de bloqueo y retardo extremo a extremo en vista de que son dos parámetros con los cuales se puede medir el desempeño de la red. Se eligen debido a que son frecuentemente utilizados en trabajos de investigación relacionados con problemas de congestión.

3.3.4. Parámetro de desempeño: probabilidad de bloqueo

La probabilidad de bloqueo para este trabajo de grado está definida como la evaluación del bloqueo entre las conexiones físicas, sin tener en cuenta la probabilidad de bloqueo de la red, esto quiere decir que se toma el parámetro a partir de la relación entre los paquetes que se envían y los que se reciben a través en un determinado enlace. Al generar un tráfico bien sea ratón o elefante se toma la cantidad de paquetes que se ha enviado o recibido tanto en la transmisión como en la recepción de estos, luego por medio de la ecuación 3.1 se calcula la probabilidad de bloqueo que es utilizada para la evaluación del desempeño de este trabajo [77].

$$BP = 1 - \frac{Pr}{Pe} \tag{3.1}$$

Donde:
Pr = Paquetes recibidos
Pe = Paquetes enviados

3.3.5. Parámetro de desempeño: retardo extremo a extremo

Al igual que se mencionó anteriormente la medida del retardo extremo a extremo se toma entre las conexiones físicas, esto quiere decir que se toma el retardo entre una fuente y un destino por medio de la ecuación 3.2.

$$R_{end\ to\ end} = N(d_{proc} + d_{queue} + d_{trans} + d_{prop}) \tag{3.2}$$

Donde:
N = Numero de enlaces entre la fuente y el destino
d_{proc} = Retardo de procesamiento
d_{queue} = Retardo de encolamiento
d_{trans} = Retardo de transmisión
d_{prop} = Retardo de propagación

- **Retardo de procesamiento**: Es el tiempo que demora en nodo o estación en procesar y preparar el paquete a enviar.
- **Retardo de encolamiento**: Es el tiempo de espera en el buffer de salida cuando el enlace está ocupado con otra transmisión.
- **Retardo de transmisión**: Es el tiempo que tarda el nodo o estación en poner todo el paquete sobre el enlace.
- **Retardo de propagación**: Es el tiempo que tarda un bit en ir de un extremo a otro del enlace.

3.4. Fase 2: Implementación de la red de prueba

En esta sección se hace una descripción de la configuración que se utiliza para la elaboración de la arquitectura SDON, además de la configuración de la topología que se usa para el modelo óptico y la configuración de la red NSFNET que es donde se evalúa el desempeño del mecanismo cognitivo.

Para evaluar el desempeño del mecanismo cognitivo para control de congestión (MCCC) se utilizan diferentes variaciones de tráfico en la NSFNET, este tráfico siempre va a estar presente en cada uno de los casos de estudio y está conformado tanto de flujo elefante como de ratón. Por otro lado, se tiene un tráfico de entrada que es el encargado de medir los parámetros de desempeño a analizar

en este trabajo de grado, este es enviado desde una de las fibras por uno de las lambdas hasta un host que se encuentra al otro extremo, el tráfico pasa a través de la NSFNET permitiendo así la evaluación del mecanismo en el caso de que no esté incluido y de igual forma evaluando el mecanismo cuando ya se implemente en la SDON.

En el caso de que el mecanismo ya se implemente en la SDON de prueba se realizan tres variaciones en su modo de operación, permitiendo así observar cómo es su comportamiento a causa de la congestión que se esté presentando, dado que, dependiendo del modo de operación del mecanismo se espera que se presentan variaciones en los parámetros de desempeño. De igual forma se configuran dos variaciones de tráfico entre los host hl1 y hl10 que pertenecen a la lambda 1, uno es de 250Mbps y el otro de 500Mbps, esto se hace con el fin de ver como es el comportamiento del mecanismo cognitivo con diferentes variaciones de tráfico, además de que estos son los encargados de tomar los reportes de los parámetros de desempeño. Los valores de trafico anteriormente mencionados se seleccionan a partir de que al realizar simulaciones de prueba en periodos cortos de tiempo (24 minutos) se observo que a velocidades bajas el mecanismo no presenta problemas, pero cuando se incrementa el trafico a valores superiores de 500Mbps aproximadamente, el mecanismo empieza a sufrir dificultades.

Los escenarios de simulación están compuestos de un solo componente virtual donde se incluyen todas las características, servidores y aplicaciones necesarios para el desarrollo de este trabajo de grado, siendo así se muestra en la tabla 3.1 la configuración del ambiente simulado en VMWare Workstation.

Nombre Servidor	CPU	RAM	Sistema Operativo	Herramientas
ServSDON	4 core	8 Gbits	Ubuntu 18	Mininet MiniEdit Controlador ONOS Python Java Linux Trafic Control

Tabla 3.1: Configuración VMware Workstation y herramientas. Elaboración propia

3.4.1. Simulación del modelo óptico y topología de red NSF-NET

Haciendo uso del modelo óptico ya mencionado se procede a realizar la emulación de los WDM switch fabric por medio de *Mininet*. En la figura 3.8 se observa el modelo creado con MiniEdit, así como también las diferentes lambdas que en este caso están representadas por un host y un *switch*, seguido a esto se encuentra el modelo WDM, que está compuesto por un enmallado de 9 dispositivos, seguido de una red de transmisión que como se ve en la figura es donde está la NSFNET, el resto de los enlaces corresponden a un nodo que representa un cable de transmisión y por último se encuentra la misma estructura mencionada anteriormente con su enmallado y la línea de fibra con sus respectivas lambdas. Cada fibra está compuesta por 3 lambdas, esto se puede ver mejor en la tabla 3.2. la tabla 3.3 muestra las características del modelo simulado.

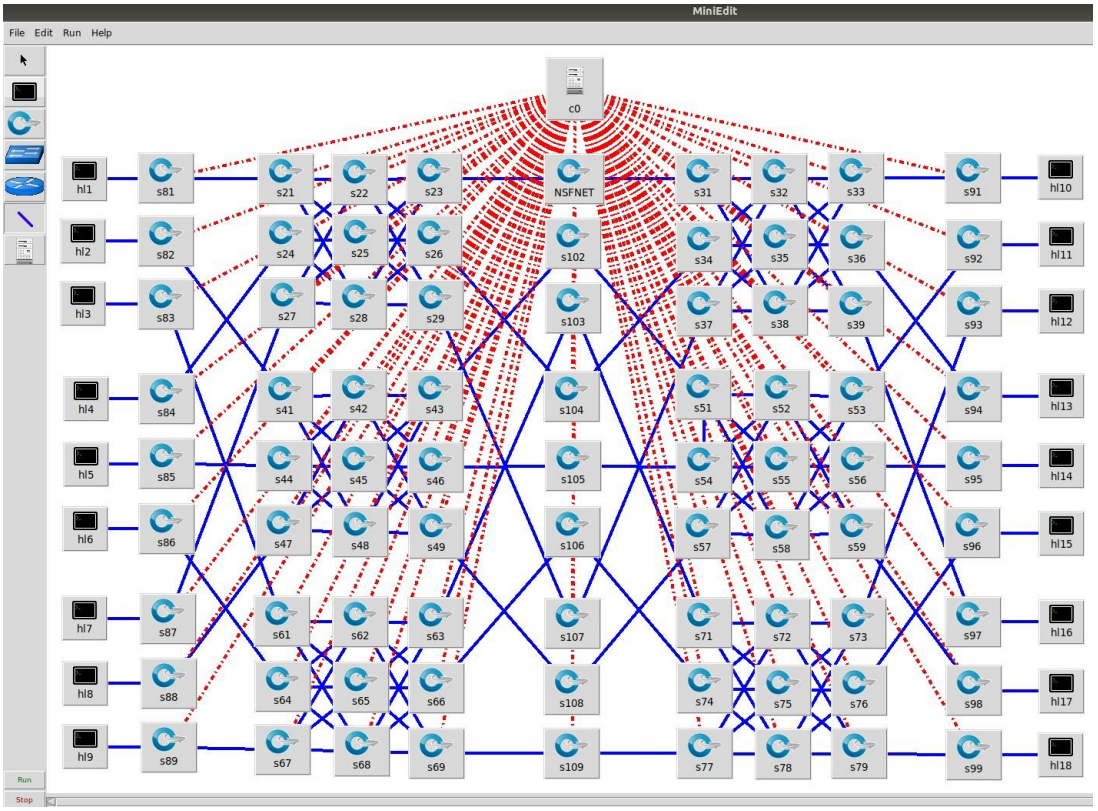

Figura 3.8: Modelo WDM switch fabric creado con MiniEdit. Elaboración propia

Fibra	Fuente/Destino	Fuente/Destino
Uno	λ_1 *Host hl1*	λ_1 *Host hl10*
	λ_2 *Host hl2*	λ_2 *Host hl11*
	λ_3 *Host hl3*	λ_3 *Host hl12*
Dos	λ_1 *Host hl4*	λ_1 *Host hl13*
	λ_2 *Host hl5*	λ_2 *Host hl14*
	λ_3 *Host hl6*	λ_3 *Host hl15*
Tres	λ_1 *Host hl7*	λ_1 *Host hl16*
	λ_2 *Host hl8*	λ_2 *Host hl17*
	λ_3 *Host hl9*	λ_3 *Host hl18*

Tabla 3.2: Líneas de fibra con sus lambdas y hosts. Elaboración propia

Característica	Modelo WDM switch fabric	NSFNET
Topología	Enmallado	Irregular
Tipo de red	WAN	WAN
Numero de nodos	80	14
Numero de enlaces	144	21
Velocidad por enlace	250Mbps/500Mbps	250Mbps/500Mbps
Protocolos soportados	TCP/UDP/ICMP	TCP/UDP/ICMP

Tabla 3.3: Características de la red de prueba. Elaboración propia

En la figura 3.9 se muestra la visión que tiene el controlador de la topología desde su interfaz de usuario, donde se observa el modelo óptico completo junto con la NSFNET que se ha configurado por medio de *Mininet*. En el controlador se muestran los nodos, los terminales y los enlaces que se están utilizando.

Para comprobar que la red está funcionando correctamente se procede a ejecutar el comando de *Mininet* que se llama pingall, el cual envía un mensaje ICMP entre todos los hosts que se encuentran en la red, esto se hace para verificar que los enlaces estén bien conectados. Pero antes de esto es necesario realizar algunas configuraciones en *ONOS* para que la red tenga un enrutamiento.

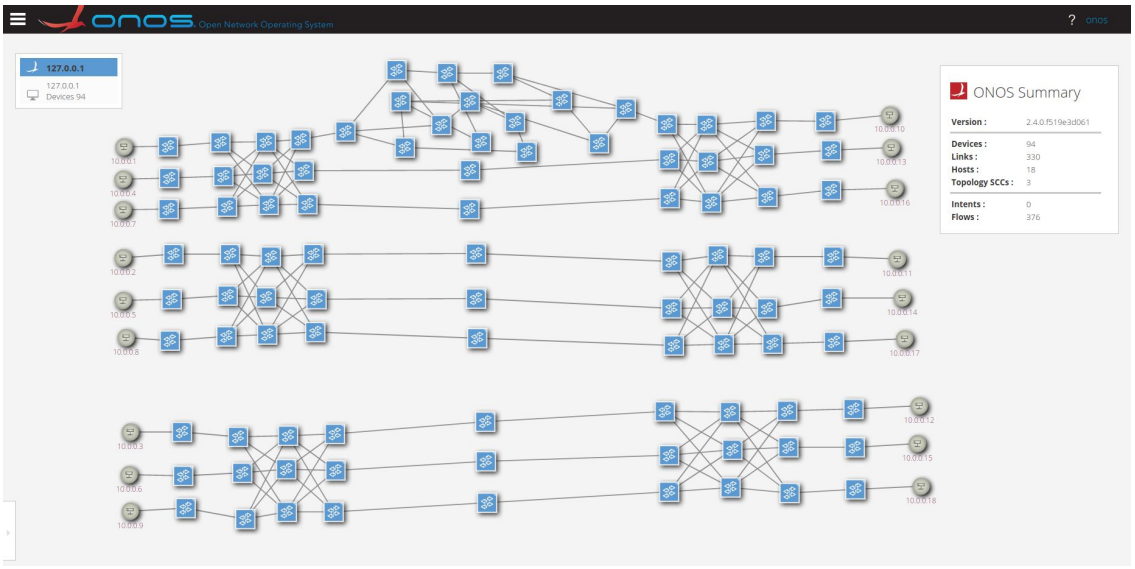

Figura 3.9: Modelo óptico y topología de red NSFNET.

Para el desarrollo de este trabajo de grado se hace necesario utilizar un paquete de aplicaciones de *ONOS* llamado OpenFlow Provider Suite el cual cuenta con las aplicaciones necesarias para que el controlador sea capaz de comunicarse con *Mininet* por medio del protocolo *OpenFlow*, además de otras funciones que son indispensables. Luego se procede a instalar un enrutamiento dinámico ya que si no se instala alguno no es posible enviar información entre los hosts, para esto se instala la aplicación llamada Reactive Forwarding que es la encargada de hacer el enrutamiento dinámico entre los diferentes enlaces de la red permitiendo el envío de tráfico a través de la red, esta aplicación utiliza el camino más corto entre un nodo de origen y un nodo de destino, se basa en el protocolo OSPF (*Open Shortest Path First*), luego de encontrar el camino establece en cada uno de los nodos seleccionados sus tablas de enrutamiento, la tabla 3.4 muestra las aplicaciones que se utilizan para la implementación de la red de prueba, además de las herramientas que se utilizan para la elaboración del modelo óptico como lo es *Mininet* y MiniEdit, también muestra a *Iperf* que es la herramientas que se utiliza para crear los flujos que circulan por la NSFNET y finalmente se muestra el *Linux Traffic Control*, que es una herramienta que trae incluida el sistema operativo *Linux* para generar perdidas en alguno de los puerto de la red por medio del comando **TC**.

Aplicaciones de ONOS	Herramientas
- Reactive Forwarding	- Mininet
- App de MCCC	- MiniEdit
- OpenFlow Provider Suite	- Iperf
* Default Drivers	- Linux Traffic Control (TC)
* Host Location Provider	
* LLDP link Provider	
* OpenFlow Base Provider	
* Optical Networks Model	

Tabla 3.4: Aplicaciones y herramientas para la construcción de la red de prueba.

Realizadas todas las configuraciones se ejecuta el comando pingall en el CLI de *Mininet* obteniendo los resultados que se muestran en la figura 3.10. En la figura se puede ver que solo son alcanzables los hosts que pertenecen a una misma lambda, en la figura 3.9 se puede observar como el controlador hace la separación de estas lambdas en tres topologías, en conclusión, la tabla 3.5 muestra cuales son los hosts alcanzables dependiendo de la lambda a la cual pertenecen.

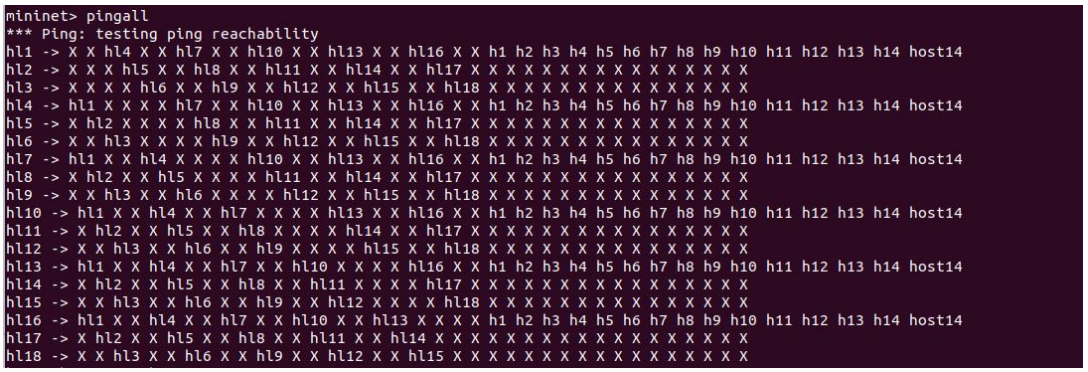

Figura 3.10: Comando **pingall** en red de prueba.

Lambda	Host alcanzables
1	*hl1, hl4, hl7, hl10, hl13, hl16*
2	*hl2, hl5, hl8, hl11, hl14, hl17*
3	*hl3, hl6, hl9, hl12, hl15, hl18*

Tabla 3.5: Hosts alcanzables con su respectiva lambda.

La figura 3.10 igualmente muestra que también son alcanzables los hosts que están conectados a cada uno de los nodos de la NSFNET, dado que se encuentran en la misma topología de lambda 1.

Realizando la simulación de los 94 dispositivos junto con todos los flujos elefante y ratón que circulan por la red, se tienen problemas utilizando un computador con un procesador Intel(R) Core(TM) i7-6700HQ con 16 GB de RAM, ocasionando que el controlador sufra algunas demoras a causa de la cantidad de dispositivos que se están utilizando, también fue probado con otro computador con un procesador AMD A12, con 16 GB de RAM, el cual tampoco soporto la simulación de esta topología, lo que quiere decir que los computadores empleados les falta capacidad para procesar los 94 dispositivos, debido a esto y teniendo en cuenta la separación de las lambdas, se toma la decisión de simular solamente los dispositivos que están conectados a la lambda 1, dado que los otros dispositivos que están en la lambda 2 y 3 no van a interactuar en la simulación y por consiguiente no van a aportar en los parámetros de desempeño para evaluar el mecanismo cognitivo, por lo tanto se eliminan estos dispositivos y se realiza la reconfiguración de la red, obteniendo la red que se puede ver en la figura 3.11 y 3.12. La reducción hace que el número de dispositivos disminuya a 40, esto permite que el controlador tenga una respuesta más rápida y se puedan realizar las pruebas. Es de tener en cuenta que el computador con el procesador AMD A12 no soportó la simulación de los 40 dispositivos, pero el computador con el procesador Intel, sí es capaz de soportar la simulación de la red de prueba.

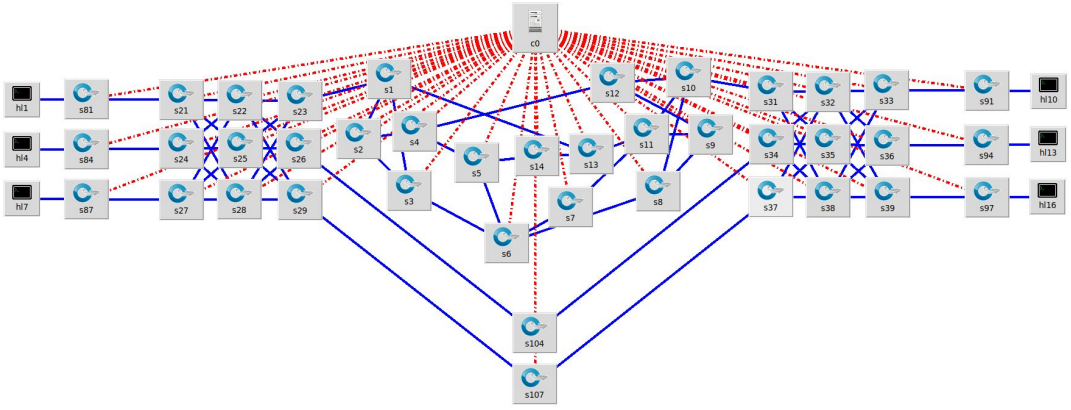

Figura 3.11: Red de prueba SDON para lambda 1, MiniEdit.

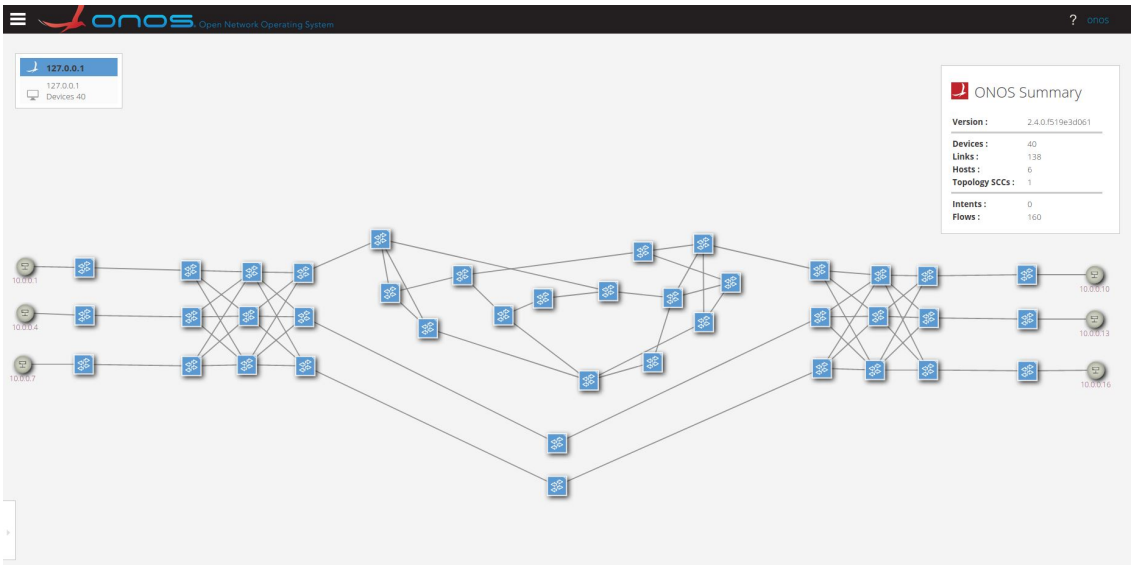

Figura 3.12: Red de prueba SDON para lambda 1, Controlador ONOS.

En la figura 3.13 y 3.14 se puede observar la configuración de los enlaces por medio del comando de *Mininet* llamado net.

En los anexos 1 y 2 se puede encontrar el script hecho en *Python* de las topologías simuladas tanto la completa como la reducida, también se puede encontrar en el anexo 3 los scripts en *Python* que se encargan de crear los flujos elefante y ratón que circulan por la NSFNET.

```
mininet> net
hl1 hl1-eth0:s81-eth1
hl4 hl4-eth0:s84-eth1
hl7 hl7-eth0:s87-eth1
hl10 hl10-eth0:s91-eth1
hl13 hl13-eth0:s94-eth1
hl16 hl16-eth0:s97-eth1
h1 h1-eth0:s1-eth2
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
h5 h5-eth0:s5-eth1
h6 h6-eth0:s6-eth1
h7 h7-eth0:s7-eth1
h8 h8-eth0:s8-eth1
h9 h9-eth0:s9-eth1
h10 h10-eth0:s10-eth2
h11 h11-eth0:s11-eth1
h12 h12-eth0:s12-eth1
h13 h13-eth0:s13-eth1
h14 h14-eth0:s14-eth1
host14 host14-eth0:s14-eth2
```

Figura 3.13: Comando **net** en topología de prueba SDON, Hosts.

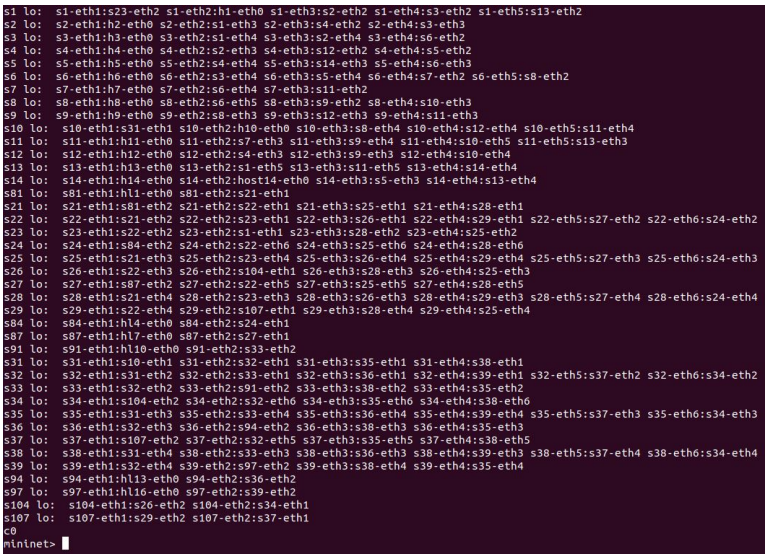

Figura 3.14: Comando **net** en topología de prueba SDON, Nodos.

3.5. Fase 3: Diseño del mecanismo cognitivo por medio de balanceo de carga

En esta sección se describe el diseño del mecanismo cognitivo que se propone como una alternativa para los problemas de congestión que sufren las redes debido a la generación de retardos, a la interrupción del trafico y a la degradación del rendimiento de la red. En este sentido se desarrolla un mecanismo cognitivo para control de congestión que tiene como propósito enfrentar los problemas de congestión que se pueden presentar en una red óptica definida por software.

Considerando que el mecanismo cognitivo se basa en las características de ciclo cognitivo se toma como referencia la arquitectura CHRON, teniendo en cuenta que esta es la que mejor se adapta a la solución de este trabajo de grado, de esta forma con la información brindada en los capítulos, 1 y 2 se procede a diseñar el mecanismo cognitivo CHRON para el control de congestión en una red óptica definida por software.

Tomando como referencia la figura 2.4 (Capítulo 2), la cual muestra la arquitectura CHRON para una cognición centralizada, se puede observar como la centrali-

zación permite que los elementos de la red puedan ser reconfigurados a partir de la información que el controlador puede ver por medio del sistema de monitoreo de la red. Esta visión general permite que en el momento en que se presente una congestión en la SDON, el CDS se dé cuenta inmediatamente de lo que está sucediendo y pueda planificar, para luego decidir cuál es la mejor opción que tiene dependiendo de la información de los otros nodos. Al final el CDS puede ejecutar la mejor decisión que en este caso está basada en la información de la probabilidad de bloqueo que tienen las otras rutas posibles para ese nodo.

La figura 3.15 muestra la implementación de mecanismo cognitivo basado en el ciclo cognitivo y una arquitectura CHRON y SDN.

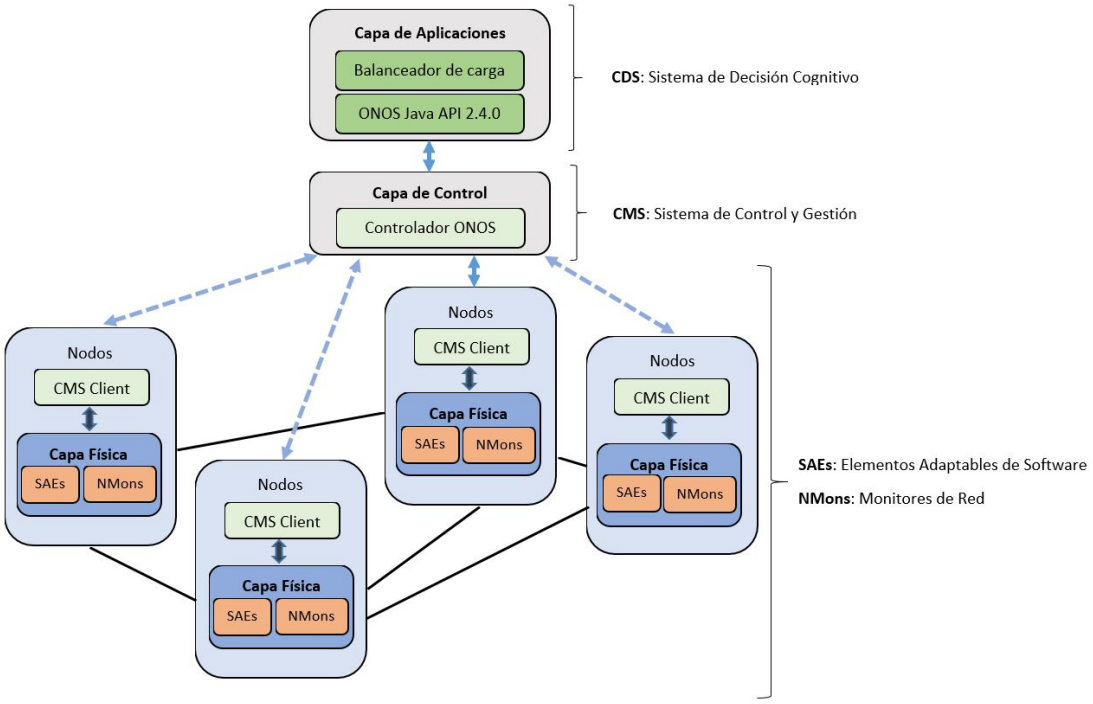

Figura 3.15: Método de control cognitivo para una SDON. Elaboración propia

Para la elaboración del mecanismo cognitivo también se hace necesario el uso de un balanceador de carga debido a que la solución al problema de congestión esta basada en el cambio de rutas que tenga disponible un nodo, por este motivo se opta por esta opción.

El balanceador de carga se encuentra alojado en la capa superior del modelo

SDN (capa de aplicaciones). En esta capa se encuentran dos módulos el balanceador de carga y la API de ONOS que está basada en lenguaje *Java*. Estos dos módulos son los que componen el CDS, que es el encargado de observar, planificar, decidir y ejecutar alguna acción dependiendo de la información que el CMS este recolectando de la red. El CDS está observando constantemente la información de los nodos, en este caso la probabilidad de bloqueo, cuando la probabilidad de bloqueo del nodo pase el umbral que se ha establecido, este le indica al balanceador de carga que cambie la dirección del flujo, para evitar que la congestión afecte los paquetes que están por llegar. El balanceador es el encargado de redirigir el tráfico por otro de los puertos que tenga disponible, para esto el CDS también está observando los otros enlaces disponibles de este nodo con el fin de enviar el tráfico por el enlace que tenga la menor probabilidad de bloqueo desde el origen hasta el destino sin tener en cuenta los nodos de borde. La API de *ONOS* es la que permite el desarrollo de la aplicación que se instala en el controlador y es la encargada de realizar todas las acciones ya mencionadas por medio de sus clases, métodos e interfaces.

En la capa de control del modelo SDN se encuentra el CMS, este es el encargado de enviar la información de los nodos al CDS para mantenerlo informado de cualquier cambio en la red, además el CMS se encarga de establecer una comunicación con los CMS CLIENT que se encuentran en cada uno de los nodos para enviar la información de las configuraciones o posibles cambios que quiera realizar el CDS. En este sentido cuando ocurre una congestión en la red el CDS le indica al CMS (Controlador ONOS) que realice una reconfiguración del nodo para redirigir el tráfico y este a su vez se comunica con el CMS CLIENT para que realice la reconfiguración de las tablas de flujo.

Por último en la capa de infraestructura o plano de datos se encuentran los últimos módulos por mencionar, el cual hace referencia a los nodos de la red y esta compuesto, primero por un módulo llamado CMS CLIENT, seguido de dos módulos el SAEs y NMons. El CMS CLIENT recibe la información del NMons para luego enviarla al CMS, para que finalmente llegue al CDS, pero de igual forma envía información que llega del controlador hacia el SAEs para que estos realicen configuraciones dependiendo de las decisiones del CDS. La información que

se envía entre el CMS CLIENT y el CMS (Controlador ONOS) se hace por medio del protocolo *OpenFlow* que es el que se encarga de la interacción entre el controlador y los *switches OpenFlow*.

El aporte cognitivo que deja el desarrollo e implementación de este mecanismo cognitivo está en el uso del ciclo cognitivo específicamente en sus fases de observar, planificar, decidir y actuar, que son los elementos claves de los cuales está compuesto el CDS para tomar las mejores decisiones haciendo que trabaje de forma autónoma sin la intervención de un administrador de red.

3.6. Fase 4: Presentación casos de estudio

En esta fase se presentan los casos de estudio que ayudan al análisis del desempeño del mecanismo cognitivo mediante la probabilidad de bloqueo y el retardo extremo a extremo. Para esto se definen tres posibles rutas a analizar, la primera es la que se toma mediante el camino más corto entre el S1 y S10 de la NSFNET y las otras dos rutas son las que existen entre S1 y S10, pero con un mayor número de saltos, esto se puede ver en la figura 3.16, de la siguiente forma:

- **Enlace A**: S1 – S13 – S11 – S10
- **Enlace B**: S1 – S2 – S4 – S12 – S10
- **Enlace C**: S1 – S3 – S6 – S8 – S10

El enlace A cuenta con cuatro saltos desde el S1 hasta el S10, el enlace B tiene 5 saltos al igual que el enlace C. La conexión entre los nodos está definida por los puertos que se pueden ver en la tabla 3.6. La tabla muestra las conexiones nodo a nodo, pero no tiene en cuenta las conexiones entre los hosts ya que todos los hosts están en el eth1 de todos los puertos exceptuando a los nodos S1 y S10 que están conectados al modelo óptico por medio de los eth1, por esta razón el S1 tiene el host en el eth2 al igual que el S10.

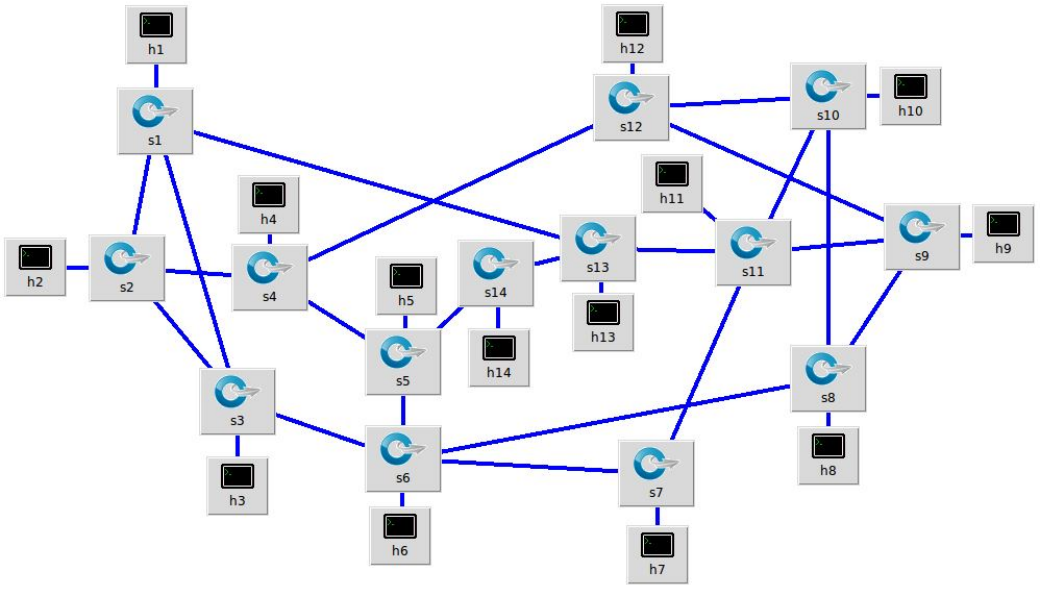

Figura 3.16: NSFNET MiniEdit. Elaboración propia

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
S1	X	eth3	eth4	X	X	X	X	X	X	X	X	X	eth5	X
S2	eth2	X	eth4	eth3	X	X	X	X	X	X	X	X	X	X
S3	eth2	eth3	X	X	X	eth4	X	X	X	X	X	X	X	X
S4	X	eth2	X	X	eth4	X	X	X	X	X	X	eth3	X	X
S5	X	X	X	eth2	X	eth4	X	X	X	X	X	X	X	eth3
S6	X	X	eth2	X	eth3	X	eth4	eth5	X	X	X	X	X	X
S7	X	X	X	X	X	eth2	X	X	X	X	eth3	X	X	X
S8	X	X	X	X	X	eth2	X	X	eth3	eth4	X	X	X	X
S9	X	X	X	X	X	X	X	eth2	X	X	eth4	eth3	X	X
S10	X	X	X	X	X	X	X	eth3	X	X	eth5	eth4	X	X
S11	X	X	X	X	X	X	eth2	X	eth3	eth4	X	X	eth5	X
S12	X	X	X	eth2	X	X	X	X	eth3	eth4	X	X	X	X
S13	eth2	X	X	X	X	X	X	X	X	X	eth3	X	X	eth4
S14	X	X	X	X	eth3	X	X	X	X	X	X	X	eth4	X

Tabla 3.6: Puertos de conexión *OpenFlow* nodo a nodo NSFNET.

Para el envío de tráfico se utiliza la herramienta llamada *Iperf* la cual necesita un host cliente y un servidor para poder enviar el tráfico. Los comandos utilizados para el envío de tráfico son los siguientes en el caso del cliente:

- Tráfico ratón: iperf –c 10.0.1.2 -u -b 25M -i 1800 -t 46920 -p 10102

- Tráfico elefante: iperf –c 10.0.1.13 -u -b 1500M -l 65k -i 1800 -t 1500 -p 20113

Por último está el tráfico que se aplica en una de las lambdas host hl1 con destino al host hl10 que es el encargado de recolectar la información de los parámetros de desempeño (Probabilidad de bloqueo y retardo extremo a extremo).

- Tráfico hl1 – hl10: iperf –c 10.0.0.10 -u -b 250M -i 1800 -t 46800 -p 10001

Para el servidor:

- Tráfico ratón: iperf -s -u -p 10102 -i 1800
- Tráfico elefante: iperf -s -u -p 20113 -l 65k -i 1800
- Tráfico hl1 – hl10: iperf -s -u -e -p 10001 -i 1800 $>$ lh1h10

Donde:

Iperf: Comando utilizado para iniciar la configuración de los flujos.
-c server_addres: Le indica a la herramienta que se va a utilizar en el cliente, seguido se coloca la Ip del servidor.
-u: Utiliza UDP en vez de TCP.
-b: Establece el ancho de banda del canal.
-i: Intervalo de tiempo entre cada uno de los reportes.
-t: Tiempo en segundos en que se ejecuta el comando, envía o recibe tráfico.
-p: Establece el puerto de escucha.
-s: Le indica a la herramienta que se va a utilizar en el servidor.
-l: Establece la lectura/escritura del tamaño o longitud del buffer (UDP default 1470).
-e: Muestra información adicional en el reporte.
$>$: Crea un archivo de reporte, con el nombre que se le indique [78].

El tráfico enviado por el enlace B y C es continuo, pero el tráfico que se envía por el enlace A cambia a medida que la simulación se va ejecutando, los cambios en

el tráfico se presentan cuando se genera una congestión, que es cambiando de flujo ratón por flujo elefante con el parámetro -l (65k), además de aumentar la velocidad a 1500Mbps, esto hace que la red aumente la pérdida de paquetes y por consiguiente se espera que la congestión sea critica, pero al realizar las pruebas los resultados indican que hay un aumento de la pérdida de paquetes pero no es lo suficientemente alto para poder medir los parámetros de desempeño así que se opta por utilizar una herramienta extra que tiene incorporada *Linux* llamada *Trafic Control*, esta herramienta puede ser utilizada desde el script de *Python* cuando se crea la topología, pero esta opción es permanente (Loss, Delay, entre otros), por esta razón fue descartada. La opción que se toma es crear un script de *Python* que es llamado desde el CLI de *Mininet* para que ejecute el comando de *Trafic Control* llamado **TC** en la ventana del nodo que se quiere aplicar la pérdida, en este caso se establece una pérdida en el S13-eth3 del enlace A, para que aumente la pérdida de paquetes en un 8 %.

Aplicando estas pérdidas en el puerto ya es posible analizar los parámetros de desempeño que sirven para evaluar el mecanismo cognitivo en la SDON de prueba y que son generados por el tráfico que se envía desde hl1 hasta hl10. Los scripts de *Python* con todos los flujos y el comando de *Trafic Control* se encuentran en el anexo 3.

El tiempo total de la simulación es de 12 horas, para el envío de tráfico elefante que es el encargado de ocasionar la congestión se toma un tiempo de 100 minutos, esto equivale al 13.88 % del total de tráfico que se genera, los reportes serán tomados cada 30 minutos para tener un total de 24 reportes. De esta forma, el tráfico elefante se distribuye en 5 bloques de congestión con diferentes intervalos de tiempo, la tabla 3.7 muestra la distribución de tráfico para las 12 horas de simulación.

Por último, se establece el enrutamiento que se utiliza en los casos de estudio, el cual es una aplicación que tiene *ONOS* por defecto que se llama Reactive Forwarding, además del paquete de aplicaciones llamado OpenFlow Provider Suite que sirve para establecer la comunicación entre el controlador y la red creada en *Mininet* por medio del protocolo *OpenFlow*.

Congestión	Tipo de tráfico	Tiempo Inicial (m)	Tiempo Final (m)	Tiempo Total (m)
No	Ratón	0	165	165
Si	Elefante/Ratón	165	190	25
No	Ratón	190	215	25
Si	Elefante/Ratón	215	250	35
No	Ratón	250	265	15
Si	Elefante/Ratón	265	280	15
No	Ratón	280	465	185
Si	Elefante/Ratón	465	480	15
No	Ratón	480	510	30
Si	Elefante/Ratón	510	520	10
No	Ratón	520	720	200

Tabla 3.7: Distribución de tráfico, Simulación de 12 horas.

Los casos de estudio que se presentan a continuación tienen como objetivo ayudar a la evaluación del desempeño del mecanismo cognitivo basado en SDN y CHRON, de esta forma se presentan dos casos de estudio, el primero consiste en evaluar el desempeño de la red sin el mecanismo cognitivo y el segundo con el mecanismo cognitivo.

3.6.1. Caso de estudio 1

En este caso se analiza la probabilidad de bloqueo y el retardo extremo a extremo sin incluir el mecanismo cognitivo, simplemente se utiliza el Reactive Forwarding para dar enrutamiento a la red y se evalúa su desempeño, esto se hace para observar cómo actúa una red sin el MCCC. En este caso lo que se espera es que en el momento en que ocurra una congestión los parámetros de desempeño tengan valores más altos que cuando ya se realice la inclusión del mecanismo cognitivo.

Para obtener estos parámetros se utiliza el reporte generado por el tráfico que se envía desde lambda 1. Este tráfico cuenta con dos variaciones 250Mbps y 500Mbps, el fin de estas variaciones es observar como es el comportamiento del mecanismo ante diferentes velocidades.

3.6.2. Caso de estudio 2

En el segundo caso de estudio se analiza la probabilidad de bloqueo y el retardo extremo a extremo en la SDON de prueba incluyendo el mecanismo cognitivo basado en CHRON con el uso de un balanceador de carga que se incorpora en uno de los nodos de la NSFNET. Los reportes al igual que en el caso 1 son tomados con el tráfico del host de lambda 1 del modelo óptico.

En este caso se realiza la inclusión del mecanismo cognitivo por medio de una aplicación desarrollada con la API 2.4.0 de ONOS. La aplicación es instalada en el controlador en lugar del Reactive Forwarding, pero el enrutamiento se sigue basando en el camino más corto entre dos nodos. Esto se debe a que el algoritmo del mecanismo cognitivo utiliza las funciones de la aplicación Reactive Forwarding para seguir dando el enrutamiento a la red.

A diferencia del caso 1, este caso cuenta con tres escenarios de prueba para el mecanismo cognitivo, su fin es ver como es la respuesta del mecanismo ante la congestión, cambiando su modo de operación. Cada uno de estos escenarios tiene una variación de tráfico de 250Mbps y 500Mbps, en la figura 3.17 se puede observar el diagrama para los dos casos de estudio.

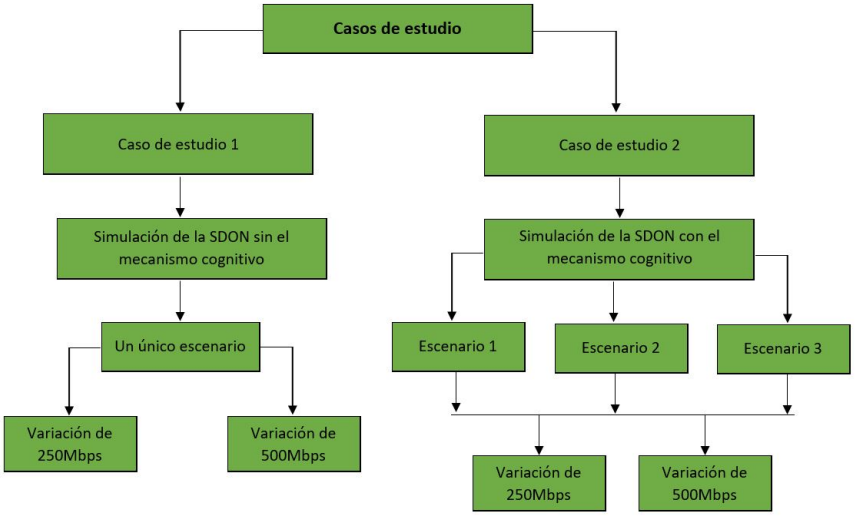

Figura 3.17: Diagrama casos de estudio. Elaboración propia

3.7. Fase 5A: Caso de estudio 1 simulación de la SDON sin el mecanismo cognitivo

En este caso de estudio, se realiza la simulación de la red de prueba basada en SDON con la topología de red NSFNET, utilizando un tráfico uniforme sobre el modelo óptico y la NSFNET. Se plantea generar congestión a ciertos intervalos de tiempo para así más adelante poder hacer una comparación entre los casos de estudio. Para iniciar con la prueba se instalan en el controlador *ONOS* las aplicaciones necesarias para realizar la prueba. la figura 3.18 muestra las aplicaciones que se utilizan en esta prueba.

	Title	App ID	Version
✓	Default Drivers	org.onosproject.drivers	2.4.0
✓	Host Location Provider	org.onosproject.hostprovider	2.4.0
✓	LLDP Link Provider	org.onosproject.lldpprovider	2.4.0
✓	ONOS GUI2	org.onosproject.gui2	2.4.0
✓	OpenFlow Base Provider	org.onosproject.openflow-base	2.4.0
✓	OpenFlow Provider Suite	org.onosproject.openflow	2.4.0
✓	Optical Network Model	org.onosproject.optical-model	2.4.0
✓	Reactive Forwarding	org.onosproject.fwd	2.4.0

Figura 3.18: Aplicaciones instaladas en el controlador caso de estudio 1.

Para comprobar que se está efectuando adecuadamente el enrutamiento se ejecuta el comando pingall desde el CLI de *Mininet*, en la figura 3.19 se pueden ver los resultados obtenidos. Donde se observa que todos los hosts que se encuentran en la lambda 1 están correctamente conectados entre sí, incluyendo los host que están conectados a cada uno de los nodos de la NSFNET.

```
mininet> pingall
*** Ping: testing ping reachability
hl1 -> hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
hl4 -> hl1 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
hl7 -> hl1 hl4 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
hl10 -> hl1 hl4 hl7 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
hl13 -> hl1 hl4 hl7 hl10 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
hl16 -> hl1 hl4 hl7 hl10 hl13 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
h1 -> hl1 hl4 hl7 hl10 hl13 hl16 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
h2 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
h3 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
h4 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
h5 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 host14
h6 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 host14
h7 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 host14
h8 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 host14
h9 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 host14
h10 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 host14
h11 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 host14
h12 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 host14
h13 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 host14
h14 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 host14
host14 -> hl1 hl4 hl7 hl10 hl13 hl16 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14
*** Results: 0% dropped (420/420 received)
```

Figura 3.19: Comando pingall en topología de prueba caso 1.

Después de verificar que el enrutamiento sea el correcto, se utiliza el comando link en la CLI de *Mininet* esto se hace debido a que en este trabajo de grado se analiza el mecanismo cognitivo en la NSFNET, sin embargo en la figura 3.12, se puede observar que la lambda 1 tiene tres enlaces de transmisión que son: primero la NSFNET, segundo es el S104 y tercero es el S107, pero como el modelo óptico utiliza el Reactive Forwarding que a su vez usa el camino más corto para definir el enrutamiento entre los nodos, esto hace que cuando se envían los flujos desde hl1 hasta hl10 para probar el mecanismo y tomar los reportes, este flujo tome el camino de S104 o S107, porque son más cortos. Por esta razón se utilizar el comando link que permite desconectar o conectar un enlace, ver anexo 3 (flows_nsfnet.sh). La figura 3.20 muestra la sentencia utilizada para desconectar estos enlaces.

```
link s26 s104 down
link s29 s107 down
```

Figura 3.20: Comando link utilizado en la SDON de prueba.

Teniendo en cuenta que son varios los flujos que se tienen que enviar a través de la NSFNET, además del flujo que toma el reporte para los parámetros de desempeño que va desde el hl1 hasta el hl10 se incluye un script de *Python* en la topología del modelo óptico y la NSFNET que llama a todos los flujos y los ejecuta al mismo tiempo incluido el comando link. De esta forma solo se debe de ejecutar

el script creado por *Mininet* y este automáticamente empieza la simulación. La figura 3.21 muestra cómo se debe de ejecutar el modelo SDON con *Python*.

```
ubuntu@ubuntu:~/mccc/toponsfnet$ sudo python nsfnet_iperf.py
```

Figura 3.21: Comando en Ubuntu 18 para iniciar la simulación.

Es importante mencionar que en este caso de estudio al iniciar la prueba la aplicación de *ONOS* Reactive Forwarding, elige como el camino más corto el enlace A, que se compone de los nodos S1, S13, S11, S10 y no cambia en el transcurso de la simulación. Esto quiere decir que observando la tabla 3.6 de la configuración de los puertos, siempre se envía el flujo por el puerto 5 del nodo S1. La figura 3.22 muestra el diagrama de funcionamiento de esta aplicación.

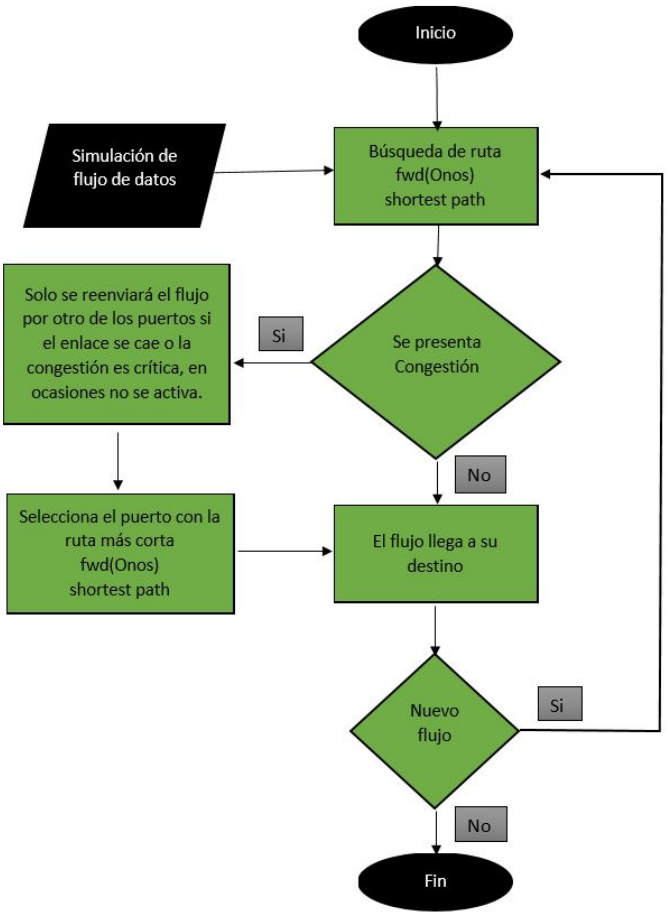

Figura 3.22: Diagrama de funcionamiento Reactive Forwarding ONOS.

3.7.1. Resultados caso de estudio 1

Como se mencionó anteriormente se tienen dos variaciones de tráfico a la entrada de la lambda debido a esto es necesario modificar el script de *Python* que genera este tráfico cambiando el parámetro del ancho de banda (b), ver anexo 4. A continuación se muestran los resultados del caso de estudio 1 y posteriormente en la sección 3.9 se muestra el análisis de los mismos.

Variación 1: 250Mbps En esta variación el valor de b es de 250M esto quiere decir que se envía un tráfico con una velocidad de 250Mbps desde el host hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un parámetro adicional (Throughput) que permite observar mejor el comportamiento del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.8 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.064	0.01	262
60	0.058	0.01	262
90	0.054	0.009	262
120	0.074	0.01	262
150	0.051	0.009	262
180	4	0.01	252
210	2.8	0.01	255
240	6.8	0.011	244
270	4.1	0.01	251
300	2.8	0.01	255
330	0.058	0.009	262
360	0.053	0.01	262
390	0.058	0.009	262
420	0.055	0.009	262
450	0.058	0.009	262
480	4.1	0.011	251
510	0.086	0.009	262
540	2.8	0.001	255

Continua en la siguiente página.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
570	0.055	0.01	262
600	0.058	0.01	262
630	0.059	0.01	262
660	0.055	0.01	262
690	0.054	0.009	262
720	0.057	0.01	262

Tabla 3.8: Resultados caso de estudio 1, variación 1

Variación 2: 500Mbps En esta variación el valor de b es de 500M esto quiere decir que se envía un tráfico con una velocidad de 500Mbps desde el terminal hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un parámetro adicional (Throughput) que permite observar mejor el comportamiento del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.9 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.37	0.008	522
60	0.38	0.008	522
90	0.38	0.008	522
120	0.36	0.008	522
150	0.36	0.008	522
180	4.6	0.009	500
210	3.2	0.009	507
240	7.3	0.01	486
270	4.6	0.009	500
300	3.2	0.009	507
330	0.36	0.008	522
360	0.37	0.008	522
390	0.37	0.008	522
420	0.37	0.008	522
450	0.37	0.008	522
480	4.5	0.09	501
510	0.38	0.008	522
540	3.1	0.009	508

Continua en la siguiente página.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
570	0.37	0.008	522
600	0.38	0.008	522
630	0.35	0.008	522
660	0.36	0.008	522
690	0.4	0.008	522
720	0.39	0.008	522

Tabla 3.9: Resultados caso de estudio 1, variación 2

3.8. Fase 5B: Caso de estudio 2 simulación de la SDON con el mecanismo cognitivo

En el segundo caso de estudio se realiza la simulación de la topología de prueba SDON, con la implementación del mecanismo cognitivo basado en CHRON y SDN. Al igual que en el caso anterior se realiza la instalación de las aplicaciones necesarias para el desarrollo de la simulación, pero a diferencia del caso anterior no se instala la aplicación Reactive Forwarding sino la aplicación que se desarrolla por medio de la API 2.4.0 de ONOS llamada Mccc-fwd App, la figura 3.23 muestra las aplicaciones instaladas en el controlador ONOS.

	Title	App ID	Version
✓	Default Drivers	org.onosproject.drivers	2.4.0
✓	Host Location Provider	org.onosproject.hostprovider	2.4.0
✓	LLDP Link Provider	org.onosproject.lldpprovider	2.4.0
✓	Mccc-fwd App	org.mccc-fwd.app	1.0.SNAPSHOT
✓	ONOS GUI2	org.onosproject.gui2	2.4.0
✓	OpenFlow Base Provider	org.onosproject.openflow-base	2.4.0
✓	OpenFlow Provider Suite	org.onosproject.openflow	2.4.0
✓	Optical Network Model	org.onosproject.optical-model	2.4.0

Figura 3.23: Aplicaciones instaladas en el controlador caso de estudio 2.

Para comprobar que se tenga un enrutamiento se realiza la misma prueba del caso anterior. La figura 3.19 muestra los resultados. Seguido a esto se utiliza nuevamente el comando de la CLI de *Mininet* link para bloquear el tráfico de los

enlaces de transmisión como se explica en el caso anterior, la figura 3.20 muestra estos comandos y la figura 3.21 el comando para ejecutar la simulación con los mismos criterios expuestos para el caso de estudio 1.

Para el desarrollo del mecanismo cognitivo basado en el ciclo cognitivo se hace necesario el uso de la API 2.4.0 de ONOS, para crear una aplicación en la que se utiliza el modelo CHRON con una arquitectura SDN, en el cual se implementa un balanceador de carga para el control de congestión. Lo primero que se debe de hacer para el diseño de este mecanismo cognitivo es crear una nueva aplicación, para esto se usa el comando que está en la figura 3.24.

```
mvn archetype:generate -DarchetypeGroupId=org.onosproject -DarchetypeArtifactId=onos-bundle-archetype
```

Figura 3.24: Comando para crear una nueva aplicación en API de ONOS.

Luego de ejecutar el comando se tienen que llenar algunos campos, seguido de esto se empieza a crear la aplicación. La figura 3.25 muestra cómo se llenaron los campos.

```
[INFO] Generating project in Interactive mode
[INFO] Archetype [org.onosproject:onos-bundle-archetype:2.4.0] found in catalog remote
Define value for property 'groupId': org.mccc.app
Define value for property 'artifactId': mccc-app
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' org.mccc.app: :
[INFO] Using property: onosVersion = 2.4.0
Confirm properties configuration:
groupId: org.mccc.app
artifactId: mccc-app
version: 1.0-SNAPSHOT
package: org.mccc.app
onosVersion: 2.4.0
 Y: : y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: onos-bundle-archetype:2.4.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: org.mccc.app
[INFO] Parameter: artifactId, Value: mccc-app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: org.mccc.app
[INFO] Parameter: packageInPathFormat, Value: org/mccc/app
[INFO] Parameter: package, Value: org.mccc.app
[INFO] Parameter: groupId, Value: org.mccc.app
[INFO] Parameter: artifactId, Value: mccc-app
[INFO] Parameter: onosVersion, Value: 2.4.0
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: /home/ubuntu/mccc-app
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

Figura 3.25: Aplicación creada correctamente con API de ONOS.

Lo siguiente que se hace es quitar los comentarios de la etiqueta properties que se ubica dentro del archivo generado y se llama pom.xml, la configuración se

coloca a criterio del programador, en la figura 3.26 se puede ver la configuración, el resto de código no se modifica, ver anexo 5.

```
<properties>
    <onos.app.name>org.mccc-fwd.app</onos.app.name>
    <onos.app.title>Mccc-fwd App</onos.app.title>
    <onos.app.category>Monitoring</onos.app.category>
    <onos.app.origin>Mccc-fwd, Inc.</onos.app.origin>
</properties>
```

Figura 3.26: Configuración properties archivo pom.xml.

En este caso los archivos de Java que ejecutan la aplicación se guardan en la ruta que se describe en la figura 3.27, ver anexo 6.

```
/home/ubuntu/mccc-app/src/main/java/org/mccc/app
```

Figura 3.27: Ruta de Archivos Java.

Por último cuando la aplicación ya está lista se debe de realizar el empaquetado para que pueda ser instalada en el controlador, se ingresar a la carpeta donde está la aplicación y se ejecuta el comando que está en la figura 3.28. Cuando termine de compilar aparece un mensaje que el proceso se desarrolló correctamente, el siguiente paso es instalar la aplicación, para esto se utiliza el archivo de extensión oar, que se encuentra en la carpeta que se acaba de crear llamada target. La figura 3.23 muestra la aplicación instalada en el controlador.

```
ubuntu@ubuntu:~/mccc-app$ mvn clean install
```

Figura 3.28: Comando para empaquetar la aplicación de ONOS.

Finalizada la instalación de las aplicaciones se procede a explicar el desarrollo del mecanismo cognitivo basado en CHRON, implementando un balanceador de carga.

BALANCEADOR DE CARGA

El mecanismo cognitivo para el control de congestión basado en CHRON aplica el balanceo de carga a solo uno de los nodos de la NSFNET, en este caso el nodo S1, debido a que implementarlo a más nodos de la red aumentaría su

complejidad en el desarrollo de la aplicación, de esta forma el balanceador de carga solo está configurado para que funcione sobre este nodo, dicho esto se procede a explicar su funcionamiento.

El CDS está constantemente observando la información que le brinda el controlador por medio del CMS, información que a su vez es obtenida de los NMons a través del CMS CLIENT. Los NMons recopilan la información de la probabilidad de bloqueo de los nodos involucrados en este trabajo de grado con el fin de ver cuál es el porcentaje de congestión de los enlaces, esto quiere decir que para el enlace A se toman los nodos S13 y S11, el enlace B S2, S4, S12 y el enlace C S3, S6, S8. Los nodos de borde S1 y S10 no son tenidos en cuenta para seleccionar el enlace menos congestionado debido a que en el momento de la congestión afectan los resultados de la probabilidad de bloqueo, pero si son tenidos en cuenta para detectar la congestión entre el nodo S1 y S10.

Cuando el CMS es informado de que la probabilidad de bloqueo entre el nodo S1 y S10 supera el umbral que se establece, en este caso 1 %, el CDS lanza una alerta inmediatamente al controlador y le indica que debe de cambiar el flujo que tenga como origen 10.0.0.1 que es el host hl1 y destino 10.0.0.10 host hl10 a otro de los puertos que tenga disponible. El CDS elige el enlace con el menor promedio de la suma de las probabilidades de bloqueo de los nodos involucrados. Como los valores se deben de dar en porcentajes se hace el siguiente cambio en la ecuación 3.1. Si existe un valor negativo esto quiere decir que se recibieron más paquetes de los enviados, pero como el mecanismo opera con valores positivos se hace necesario el uso del valor absoluto, obteniendo la ecuación 3.3.

$$BP = \left| 100 - \frac{Pr * 100}{Pe} \right| \tag{3.3}$$

Para obtener la información de los paquetes que se envían y reciben se utiliza la interfaz Device Service de la API de ONOS, que tiene un método que retorna una lista de tipo PORTSTATISTICS, enviando como argumento un objeto de tipo DeviceId (Id del nodo), en la figura 3.29 se puede ver la información que da la API. Esta lista contiene la información de los paquetes enviados, paquetes recibidos, bytes enviados, bytes recibidos, entre otros parámetros. Así es como el CDS

tiene la información de todos los nodos y por ende de los enlaces involucrados en este trabajo de grado [79].

```
List<PortStatistics>   getPortStatistics(DeviceId deviceId)
```

Figura 3.29: Método getPortStatics API de ONOS.

El CDS calcula la probabilidad de bloqueo entre el nodo S1 y S10 con la información de los puertos S1-eth1 (paquetes recibidos) y S10-eth1 (paquetes enviados), para luego aplicar la ecuación 3.3, pero al mismo tiempo el CDS calcula la probabilidad de bloqueo (Bp, *Blocking Probability*) de los otros nodos con la suma de los paquetes enviados y recibidos que obtiene de sus puertos y vuelve a aplicar la ecuación 3.3 para cada nodo, luego calcula el promedio de la Bp entre los nodos que pertenecen al enlace. Al final el CDS tiene la Bp entre el nodo S1 y S10, además de la Bp del enlace A (puerto 5), enlace B (puerto 3) y el enlace C (puerto 4), si el resultado de la Bp entre el nodo S1 y S10 es mayor o igual al 1 %, quiere decir que hay congestión sobre el enlace A, entonces el CDS ejecuta el balanceo de carga entre los dos enlaces disponibles B o C eligiendo el de menor probabilidad de bloqueo, esta elección del CDS es enviada al CMS (controlador *ONOS*) y este envía la información al CMS CLIENT hasta llegar al SAEs, indicándole que debe de cambiar el flujo por el puerto indicado.

Para realizar el balanceo de carga al puerto que indica el CDS se hacen dos cosas, la primera es borrar inmediatamente el flujo que está circulando por el enlace congestionado, en este caso el puerto 5 (Camino más corto, enlace A), seguido a esto se crean flujos temporales con la misma información del flujo anterior, pero con un puerto de salida diferente, el puerto seleccionado por el CDS. Segundo se establece correctamente la configuración de las tablas de flujo del nodo S1 para que los otros paquetes sigan este camino. Esto se hace debido a que si se cambia la prioridad de los flujos, dando una más alta al nuevo flujo, se espera que los paquetes nuevos tomen el camino de más prioridad, pero no ocurre de esta forma los paquetes siguen tomando el flujo anterior. Por esta razón, se borra el flujo anterior y se crea uno nuevo, pero al borrar los flujos los paquetes que se encuentran en el nodo no saben que puerto de salida tomar, entonces se crean flujos temporales para evitar un colapso del nodo. Cuando los

tiempos de vida de los flujos temporales terminan los nuevos paquetes que están llegando desde el host hl1 se encargan de establecer los nuevos flujos hasta que el CDS de una nueva indicación.

El CDS debe de estar sincronizado para borrar el flujo anterior y crear el nuevo flujo temporal, esto se hace por medio de la interfaz FlowRuleListener que tiene un método llamado event que recibe como parámetro un evento de tipo FlowRuleEvent, esta clase tiene una clase anidada que se llama FlowRuleEvent.Type que contiene una serie de constantes de clase, pero para el desarrollo de este trabajo de grado solo se utiliza la constante RULE_REMOVED. Entonces en el momento que se borra el flujo se ejecuta este evento y se crea inmediatamente el flujo temporal evitando que el nodo se desborde o en el peor de los casos el controlador [79].

Como el mecanismo usa las funciones de enrutamiento del Reactive Forwarding, también se utiliza la interfaz PacketProcesor, que tiene un método llamado process, que recibe como parámetro un contexto de tipo PacketContext. La clase PacketContext contiene toda la información tanto de los paquetes de entrada como de los de salida y es el encargado de bloquear el contexto si se requiere, entre otras funciones. En este caso se utiliza para detectar los paquetes con una IP de origen 10.0.0.1 (hl1) y destino 10.0.0.10 (hl10) en el nodo S1 o el nodo S10, para que se realice la configuración de las tablas de flujo, después de que el balanceador de carga cambia de puerto y los tiempos de vida de los flujos temporales se terminen. Todas las funciones que utiliza el Reactive Forwarding se siguen utilizando cuando no hay congestión o es un nodo diferente de S1 o S10 [79].

Por último, se define un tiempo de 5 segundos para que el CDS este tomado la información de la probabilidad de bloqueo entre el nodo S1 - S10 y de los otros enlaces. La razón es porque la interfaz Device Service al igual que la interfaz FlowRuleListener cuenta con un método llamado event que recibe un evento denominado DeviceEvent, que tiene una clase anidada DeviceEvent.Type que contiene varias constantes de clase, entre ellas PORT_STATS_UPDATE, que es la constante que se activa cuando las estadísticas de los puertos son actualizadas. Esta actualización la realiza cada 5 segundos [79].

En la figura 3.19 se observa que dentro de la NSFNET existe un host llamado host14, este terminal obtiene información que ayuda a calcular la probabilidad de bloqueo de los nodos.

La figura 3.30 muestra el diagrama de flujo del funcionamiento del balanceador de carga.

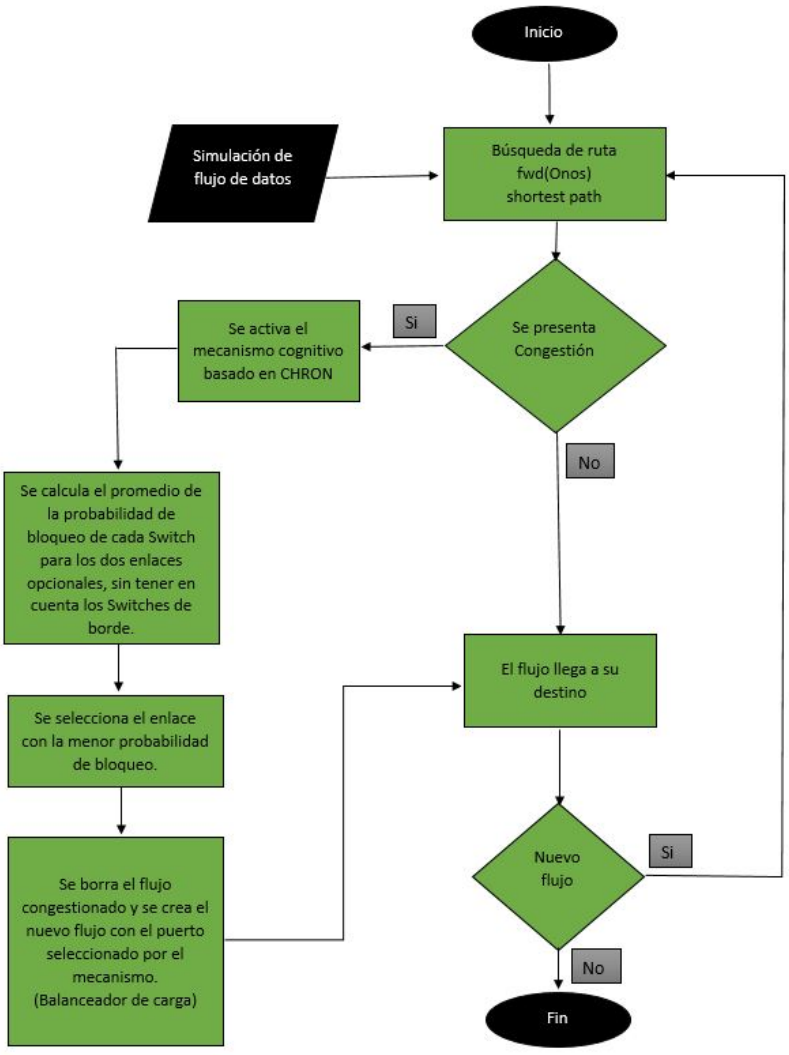

Figura 3.30: Diagrama de funcionamiento balanceador de carga.

A continuación, se describen los tres escenarios que se utilizan para probar el mecanismo cognitivo en diferentes modos de operación y sus resultados. Posteriormente en la sección 3.3 se muestra el análisis de los mismos.

3.8.1. Resultados caso de estudio 2: Escenario 1

El modo de operación en general del mecanismo cognitivo consiste en realizar un balanceo de carga en el nodo S1 por orden del CDS cuando se presenta una congestión en la SDON de prueba. En este primer escenario cuando ocurre la congestión el CDS ordena que se realice el balanceo de carga a otro puerto menos congestionado entre el enlace B o C, pero el CDS mantiene esta decisión hasta el final de la simulación. Se utiliza el mismo tráfico y los mismos comandos descritos en el caso de estudio 1 para ejecutar la simulación y de igual forma se tienen dos variaciones en el tráfico de 250Mbps y 500Mbps para la toma de los reportes.

Variación 1: 250Mbps En esta variación el valor de b es de 250M esto quiere decir que se envía un tráfico con una velocidad de 250Mbps desde el host hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un parámetro adicional (Throughput) que permite observar mejor el comportamiento del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.10 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.045	0.01	262
60	0.043	0.01	262
90	0.047	0.009	262
120	0.044	0.01	262
150	0.052	0.009	262
180	0.11	0.426	262
210	0.078	0.01	262
240	0.11	0.011	262
270	0.095	0.011	262
300	0.08	0.011	262
330	0.046	0.01	262
360	0.053	0.01	262

Continua en la siguiente página.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
390	0.045	0.01	262
420	0.056	0.01	262
450	0.054	0.01	262
480	0.082	0.011	262
510	0.058	0.01	262
540	0.074	0.01	262
570	0.046	0.01	262
600	0.052	0.01	262
630	0.046	0.01	262
660	0.051	0.01	262
690	0.042	0.01	262
720	0.047	0.01	262

Tabla 3.10: Resultados caso de estudio 2, escenario 1, variación 1

Variación 2: 500Mbps En esta variación el valor de b es de 500M esto quiere decir que se envía un tráfico con una velocidad de 500Mbps desde el terminal hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un parámetro adicional (Throughput) que permite observar mejor el comportamiento del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.11 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.42	0.009	522
60	0.41	0.009	522
90	0.4	0.008	522
120	0.41	0.009	522
150	0.39	0.008	522
180	0.73	0.199	520
210	0.58	0.009	521
240	0.92	0.01	519
270	0.84	0.009	520
300	0.58	0.009	521
330	0.4	0.009	522

Continua en la siguiente página.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
360	0.42	0.009	522
390	0.42	0.009	522
420	0.41	0.009	522
450	0.43	0.009	522
480	0.72	0.009	521
510	0.46	0.009	522
540	0.6	0.009	521
570	0.43	0.009	522
600	0.41	0.009	522
630	0.42	0.009	522
660	0.41	0.009	522
690	0.43	0.009	522
720	0.43	0.009	522

Tabla 3.11: Resultados caso de estudio 2, escenario 1, variación 2

3.8.2. Resultados caso de estudio 2: Escenario 2

En el segundo escenario, cuando ocurre la congestión el CDS ordena que se realice el balanceo de carga a otro puerto menos congestionado entre el enlace B o C, pero el CDS mantiene esta decisión por un tiempo establecido de 10 minutos, pasado este tiempo el CDS le ordena al balanceador de carga que cambie al puerto 5 (Enlace A), si todavía hay congestión, el CDS ordena nuevamente que se cambie de puerto, vuelven a pasar los 10 minutos y el CDS ordena nuevamente que se cambie al puerto 5, esto sigue ocurriendo hasta que el enlace A no este congestionado, el CDS sigue monitoreando hasta que otra congestión ocurra. Se utiliza el mismo tráfico y los mismos comandos descritos en el caso de estudio 1 para ejecutar la simulación y de igual forma se tienen dos variaciones en el tráfico de 250Mbps y 500Mbps para la toma de los reportes.

Variación 1: 250Mbps En esta variación el valor de b es de 250M esto quiere decir que se envía un tráfico con una velocidad de 250Mbps desde el host hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un

parámetro adicional (Throughput) que permite observar mejor el comportamiento del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.12 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.05	0.01	262
60	0.05	0.01	262
90	0.047	0.01	262
120	0.047	0.01	262
150	0.045	0.01	262
180	0.16	0.151	262
210	0.12	0.313	262
240	0.25	0.514	261
270	0.16	0.205	262
300	0.13	0.041	262
330	0.049	0.01	262
360	0.048	0.01	262
390	0.049	0.01	262
420	0.046	0.01	262
450	0.046	0.01	262
480	0.17	0.249	262
510	0.048	0.01	262
540	0.13	1.168	262
570	0.044	0.01	262
600	0.044	0.01	262
630	0.044	0.01	262
660	0.051	0.01	262
690	0.042	0.01	262
720	0.05	0.01	262

Tabla 3.12: Resultados caso de estudio 2, escenario 2, variación 1

Variación 2: 500Mbps En esta variación el valor de b es de 500M esto quiere decir que se envía un tráfico con una velocidad de 500Mbps desde el terminal hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un parámetro adicional (Throughput) que permite observar mejor el comportamiento

del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.13 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.39	0.009	522
60	0.41	0.009	522
90	0.39	0.008	522
120	0.41	0.009	522
150	0.4	0.008	522
180	0.79	0.467	520
210	0.65	0.174	521
240	1	0.746	519
270	0.79	0.156	520
300	0.64	0.206	521
330	0.42	0.211	522
360	0.42	0.009	522
390	0.41	0.009	522
420	0.41	0.009	522
450	0.43	0.009	522
480	0.75	0.212	520
510	0.41	0.009	522
540	0.68	0.249	521
570	0.42	0.009	522
600	0.43	0.148	522
630	0.42	0.016	522
660	0.45	0.009	522
690	0.43	0.009	522
720	0.45	0.145	522

Tabla 3.13: Resultados caso de estudio 2, escenario 2, variación 2

3.8.3. Resultados caso de estudio 2: Escenario 3

En el tercer escenario, cuando ocurre la congestión el CDS ordena que se realice el balanceo de carga a otro puerto menos congestionado entre el enlace B o C, pero el CDS mantiene esta decisión hasta que el promedio de la sumatoria de

las probabilidades de bloqueo de los nodos internos del enlace A (S13 y S11) sea menor a 0.1 % (debido a que en las pruebas realizadas el enlace sin congestión muestra valores menores a 0.1 %), esto quiere decir que el enlace ya no este congestionado, en ese momento el CDS ordena al balanceador de carga que cambie al puerto 5 (Enlace A), el CDS sigue monitoreando hasta que otra congestión ocurra. Se utiliza el mismo tráfico y los mismos comandos descritos en el caso de estudio 1 para ejecutar la simulación y de igual forma se tienen dos variaciones en el tráfico de 250Mbps y 500Mbps para la toma de los reportes.

Variación 1: 250Mbps En esta variación el valor de b es de 250M esto quiere decir que se envía un tráfico con una velocidad de 250Mbps desde el host hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un parámetro adicional (Throughput) que permite observar mejor el comportamiento del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.14 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.054	0.01	262
60	0.081	0.009	262
90	0.053	0.009	262
120	0.071	0.01	262
150	0.071	0.009	262
180	0.013	0.102	262
210	0.097	0.033	262
240	0.015	0.097	262
270	0.013	0.148	262
300	0.089	0.037	262
330	0.068	0.01	262
360	0.064	0.01	262
390	0.054	0.01	262
420	0.098	0.01	262
450	0.067	0.01	262
480	0.13	0.173	262

Continua en la siguiente página.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
510	0.051	0.01	262
540	0.1	0.245	262
570	0.051	0.01	262
600	0.063	0.01	262
630	0.059	0.01	262
660	0.06	0.01	262
690	0.062	0.01	262
720	0.054	0.01	262

Tabla 3.14: Resultados caso de estudio 2, escenario 3, variación 1

Variación 2: 500Mbps En esta variación el valor de b es de 500M esto quiere decir que se envía un tráfico con una velocidad de 500Mbps desde el terminal hl1 hasta el hl10, durante 12 horas tomando muestras cada 30 minutos con el fin de obtener la probabilidad de bloqueo y el retardo extremo a extremo, además de un parámetro adicional (Throughput) que permite observar mejor el comportamiento del retardo en la red de prueba y por lo tanto en el funcionamiento del mecanismo cognitivo. Teniendo en cuenta esto la tabla 3.15 muestra los resultados de la simulación.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
30	0.47	0.009	522
60	0.49	0.153	522
90	0.44	0.009	522
120	0.46	0.067	522
150	0.46	0.009	522
180	0.74	0.478	520
210	0.65	0.035	521
240	1	0.286	519
270	0.79	0.509	520
300	0.68	0.19	521
330	0.48	0.09	522
360	0.53	0.009	522
390	0.47	0.009	522
420	0.49	0.009	522
450	0.45	0.009	522

Continua en la siguiente página.

Tiempo (minutos)	Probabilidad de bloqueo (%)	Retardo extremo a extremo (ms)	Throughput (Mbps)
480	0.84	0.133	520
510	0.5	0.009	522
540	0.77	0.069	520
570	0.47	0.009	522
600	0.45	0.009	522
630	0.46	0.016	522
660	0.47	0.009	522
690	0.47	0.009	522
720	0.49	0.176	522

Tabla 3.15: Resultados caso de estudio 2, escenario 3, variación 2

3.9. Análisis de resultados de la SDON de prueba sin el mecanismo cognitivo y con el mecanismo cognitivo

Se describe el análisis del método establecido para el desarrollo del mecanismo cognitivo para el control de congestión, teniendo en cuenta los resultados obtenidos en las diferentes simulaciones realizadas en cada caso de estudio con el fin de analizar el desempeño del mecanismo cognitivo por medio de los parámetros de desempeño. También se analizan las dos variaciones del tráfico que se aplican sobre el modelo óptico y circulan en la NSFNET, al igual que los escenarios descritos en el segundo caso de estudio que tienen como finalidad ver cómo es el comportamiento del mecanismo en diferentes modos de operación. Sin embargo, se debe tener presente que los resultados analizados a continuación se confrontan solo entre la inclusión o no del mecanismo cognitivo para control de congestión y no se comparan con trabajos similares de otros autores. Por último, es importante resaltar que los resultados que se obtienen de las simulaciones para cada uno de los casos de estudio que se analizan a continuación, son obtenidos a partir de las limitaciones ya mencionadas como la reducción de la topología de 94 dispositivos a 40 dispositivos a causa de que no se tenía la capacidad suficiente de procesamiento en los computadores y también de las limitaciones

al generar los tráficos que circulan por la red de prueba, en el caso particular el tráfico elefante que se define con un valor de 65K bytes, pero en realidad debería de ser de al menos 100K bytes. Tomando en consideración lo anterior se procede con el análisis de los casos de estudio.

3.9.1. Análisis de resultados caso de estudio 1

En las figuras 3.31, 3.32, 3.33, se muestran las gráficas de los resultados de la simulación de la SDON de prueba realizada por un periodo de 720 minutos (12 Horas), para la variación de tráfico de 250Mbps, ver resultados en tabla 3.8. En las gráficas se observa que la falta de un mecanismo cognitivo hace que en algunos periodos de la simulación cuando se presenta congestión la probabilidad de bloqueo aumente considerablemente debido a la falta de inteligencia del nodo, alcanzando el valor más alto de 6.8 % en el intervalo de 210-240 minutos. Igualmente en la gráfica de retardo se puede ver que los puntos más altos se encuentran cuando el nodo está congestionado, como se observa en los intervalos de 210-240 y 450-480 minutos donde aumenta de 0.01ms a 0.011ms. Estos leves incrementos del retardo extremo a extremo ocasionan que el Throughput disminuya en los intervalos de congestión, alcanzando un valor mínimo de 244Mbps en el intervalo de 210-240 minutos.

A los 265 minutos de iniciada la simulación, se encuentra el primer bloque de congestión, que dura 25 minutos en este punto el tráfico ratón es cambiado por elefante, ocasionando que se aumente la probabilidad de bloqueo, pero como estos valores no son muy elevados se utiliza el *Traffic Control* para incrementar la congestión en el enlace A, el porcentaje de pérdidas utilizado en el TC es de 8 %, lo que indica en los resultados de la gráfica de probabilidad de bloqueo que a pesar de definir este porcentaje, la herramienta lo máximo que logra incrementar la congestión es de hasta 6.8 % en el intervalo de 210-240 minutos, la razón es debido a que el intervalo de tiempo que se toma es de 30 minutos y como se define congestión por 25 minutos solo se toma parte de la congestión, por ejemplo, para el caso del intervalo de 150-180 minutos que es el primer incremento que se ve en la gráfica de probabilidad de bloqueo, este reporte toma tan solo 15

minutos de congestión y el siguiente que es el de 180-210 minutos, toma los 10 minutos restantes, por eso se observa que en este intervalo la probabilidad baja de 4 % a 2.8 %. Lo mismo ocurre para los demás intervalos de congestión. Por otro lado, cuando la congestión inicia los paquetes que se están enviando por el enlace A empiezan a perderse debido al aumento del retardo que se genera por la llegada de los flujos elefante que al tener un tamaño de 65K bytes los nodos van a tardar más tiempo en procesarlos, ocasionando que los nodos del enlace empiecen a generar pérdidas por la falta de procesamiento del nodo, que a su vez ocasiona que el Throughput disminuya debido a que el retardo aumenta, haciendo que el host hl1 que es el cliente, empiece a reducir el número de paquetes que envía al host hl10 que es el servidor, para evitar que se sigan perdiendo por la congestión. De esta forma es como el nodo S1 al no tener un mecanismo que le ayude a afrontar la congestión, sigue recibiendo paquetes y los envía por el mismo enlace ocasionando que la pérdida continué hasta el final del bloque de congestión como se observa en los resultados de los parámetros de desempeño.

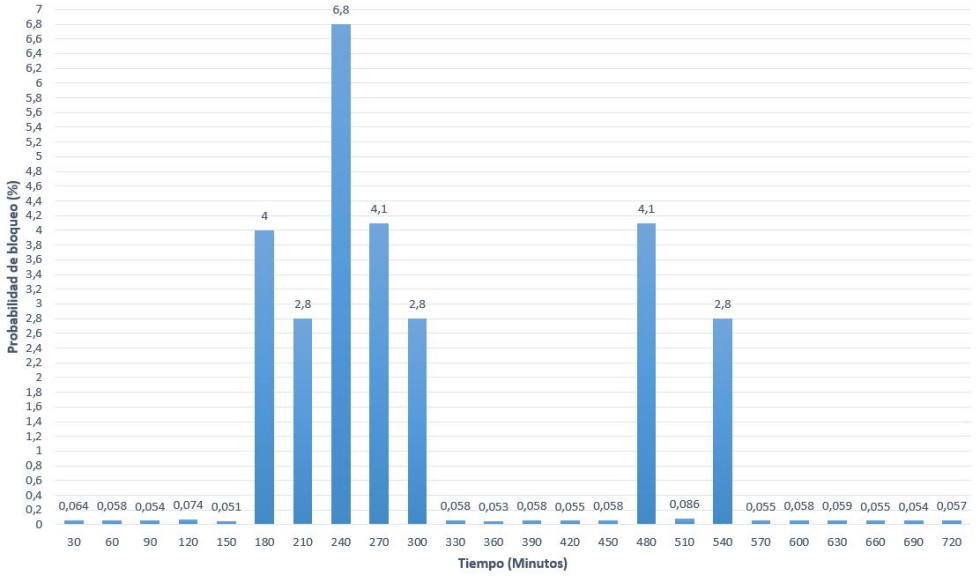

Figura 3.31: Probabilidad de bloqueo sin MCCC, variación 1.

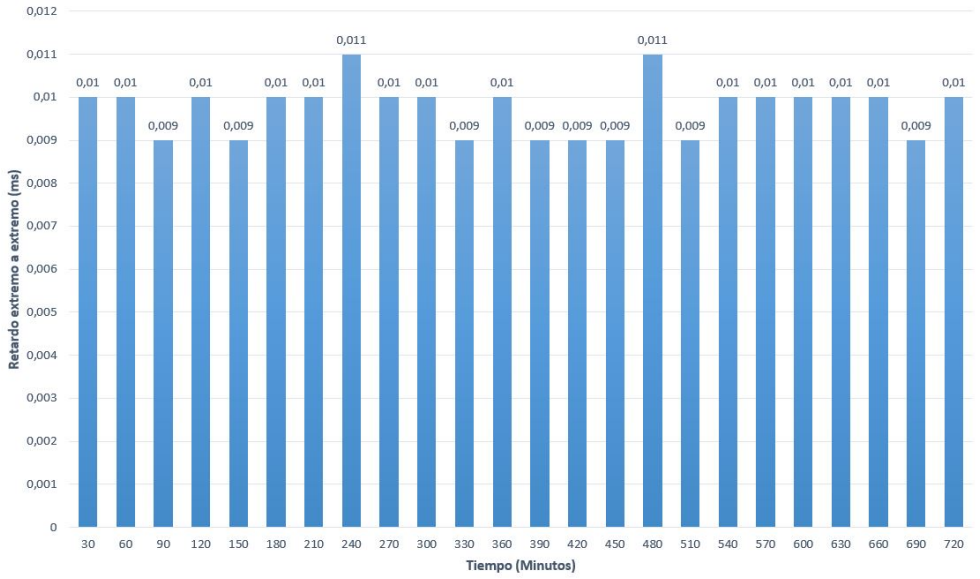

Figura 3.32: Retardo extremo a extremo sin MCCC, variación 1.

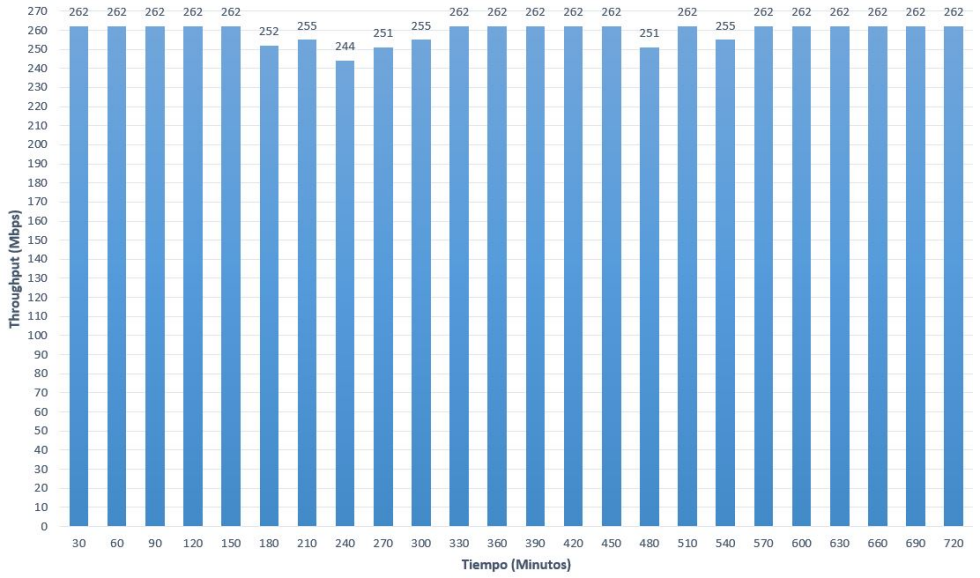

Figura 3.33: Throughput sin MCCC, variación 1.

Para una mejor visualización se muestran los resultados para la variación 1 en gráficas de líneas en las figuras 3.34, 3.35, 3.36.

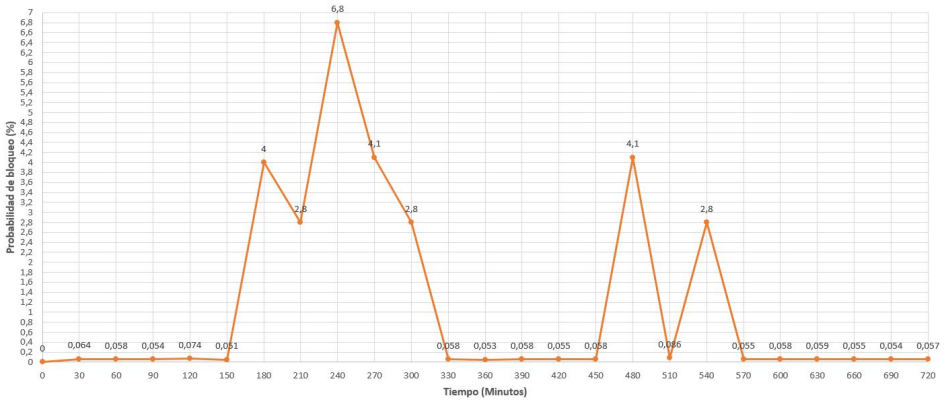

Figura 3.34: Probabilidad de bloqueo sin MCCC, variación 1.

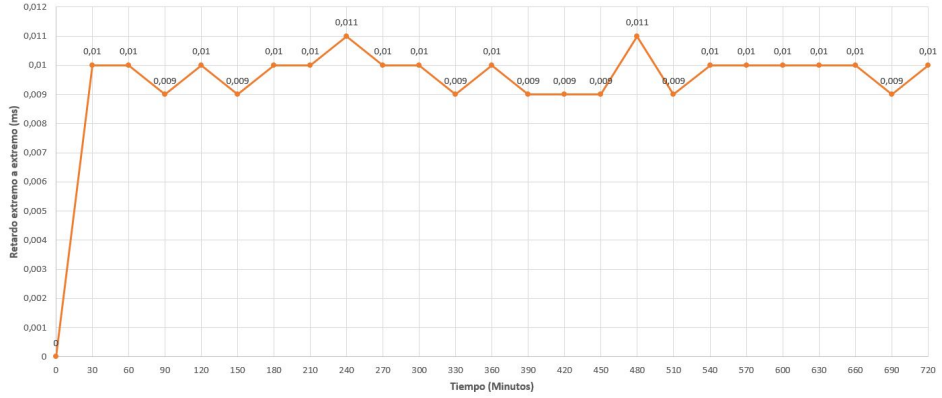

Figura 3.35: Retardo extremo a extremo sin MCCC, variación 1.

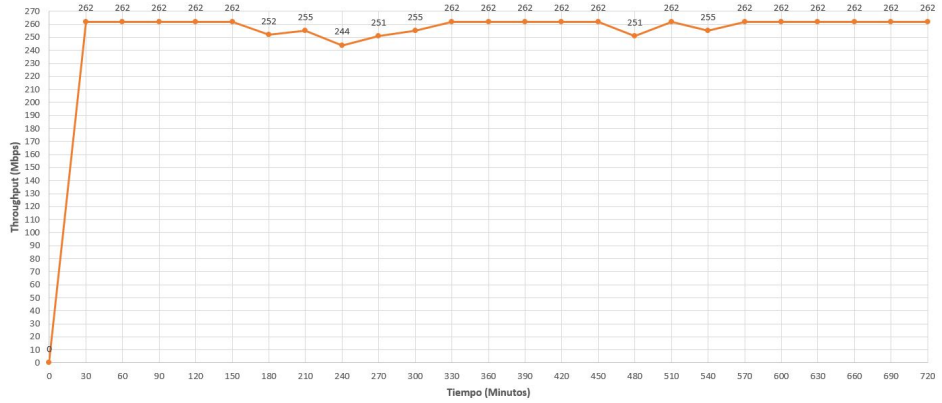

Figura 3.36: Throughput sin MCCC, variación 1.

En las figuras 3.37, 3.38, 3.39, se muestran las gráficas de los resultados de la simulación de la SDON de prueba realizada por un periodo de 720 minutos (12

Horas), para la variación de tráfico de 500Mbps, ver resultados en tabla 3.9. El aumento del tráfico sobre la red permite observar que la probabilidad de bloqueo es más alta que en la variación 1, debido a que la cantidad de paquetes que se envían desde el host hl1 es casi del doble haciendo que el enlace A se congestione más y se puedan observar valores en la probabilidad de bloqueo del 7.3 %, en el punto más crítico que es el intervalo de 210-240 minutos. En los valores del retardo también se puede observar que en este punto crítico está el valor de retardo más alto alcanzando 0.01ms y por consiguiente el valor de Throughput de 486Mbps que es el más bajo, esto quiere decir que la ausencia del mecanismo cognitivo hace que los valores de estos parámetros de desempeño aumenten en el caso de la probabilidad de bloqueo y el retardo extremo a extremo, pero en el caso del Throughput disminuya. El análisis descrito para la variación 1, sobre cómo se genera la congestión en la red se mantiene, al igual que la reducción del Throughput, que para este caso se observa que disminuye más, de hecho decae el doble, dado que para la variación 1 en el intervalo de 210-240 minutos pasa de 262Mbps que es su valor máximo a 244Mbps, disminuyendo en 18Mbps, pero para esta variación en este mismo intervalo pasa de 522Mbps a 486Mbps, disminuyendo en 36Mbps, lo cual es coherente dado que se está enviando el doble del tráfico.

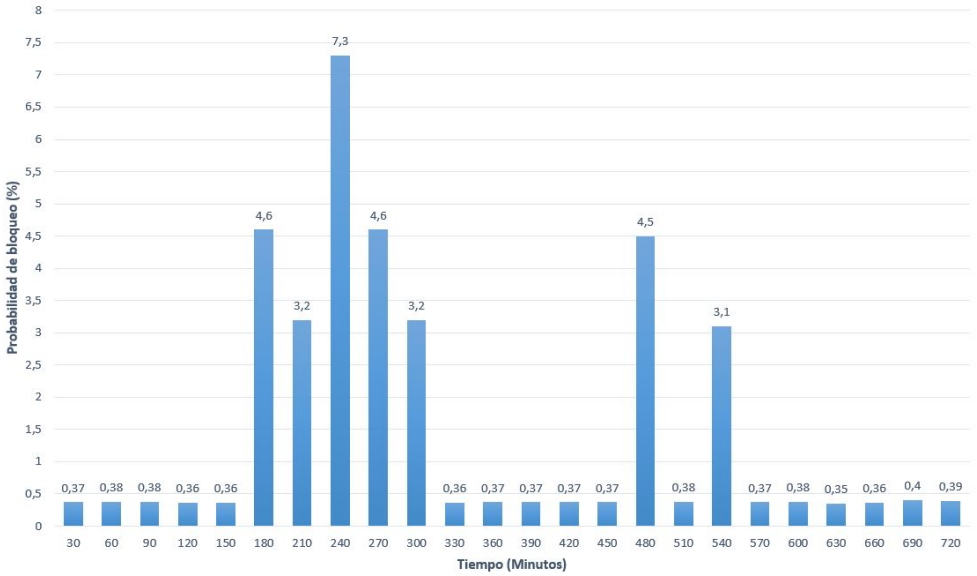

Figura 3.37: Probabilidad de bloqueo sin MCCC, variación 2.

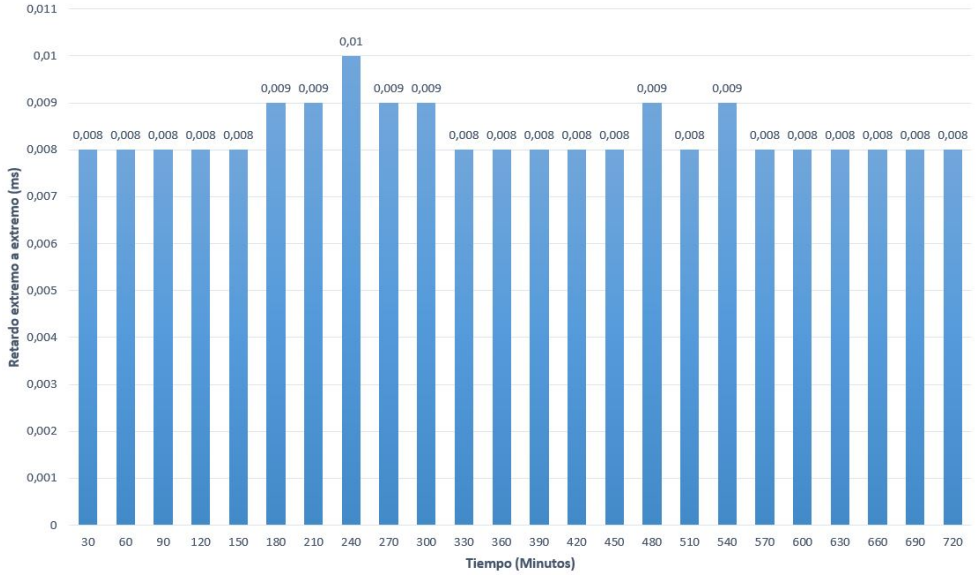

Figura 3.38: Retardo extremo a extremo sin MCCC, variación 2.

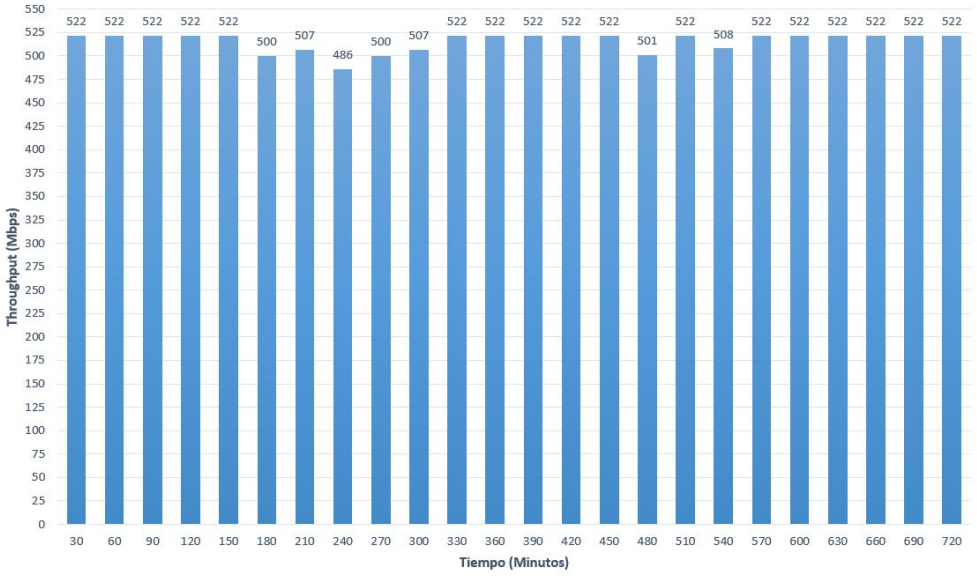

Figura 3.39: Throughput sin MCCC, variación 2.

Para una mejor visualización se muestran los resultados para la variación 2 en gráficas de líneas en las figuras 3.40, 3.41, 3.42.

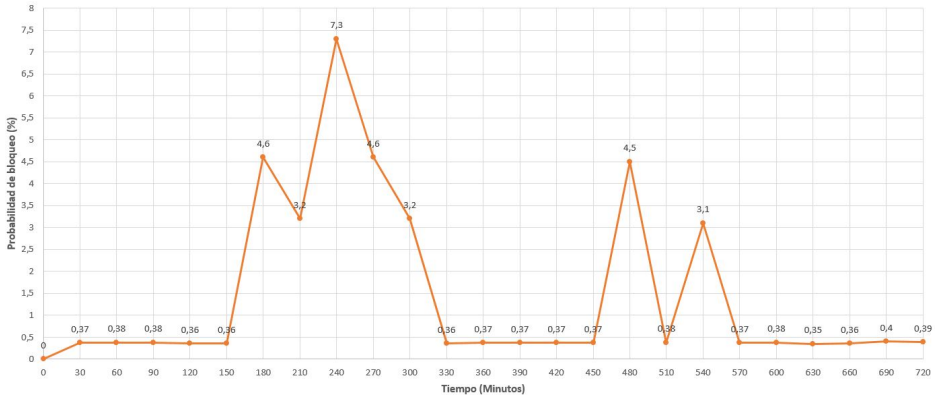

Figura 3.40: Probabilidad de bloqueo sin MCCC, variación 2.

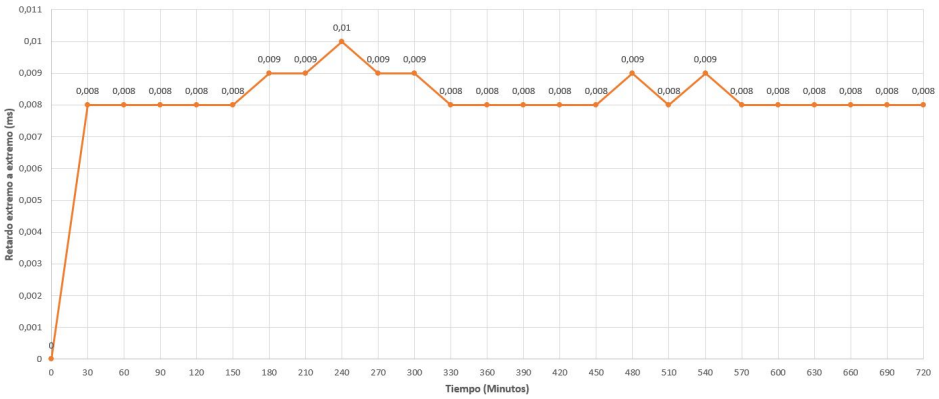

Figura 3.41: Retardo extremo a extremo sin MCCC, variación 2.

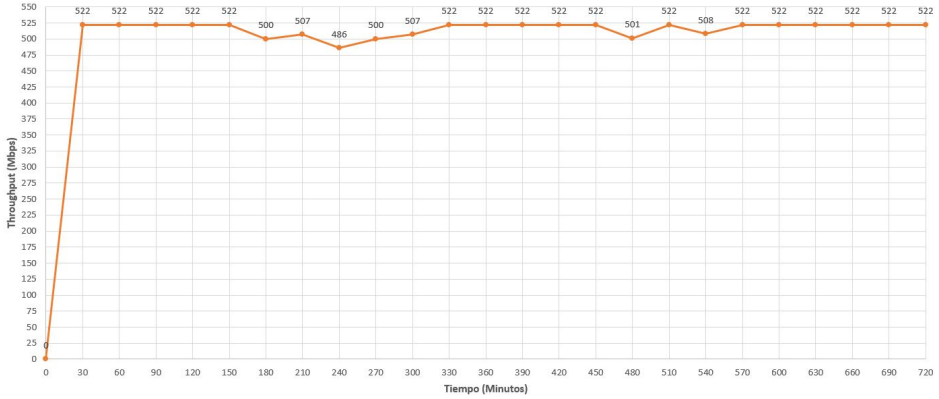

Figura 3.42: Throughput sin MCCC, variación 2.

3.9.2. Análisis de resultados caso de estudio 2

A continuación, se muestran los resultados obtenidos para cada uno de los escenarios presentados para este caso de estudio con sus dos variaciones de tráfico.

Análisis de resultados escenario 1 En las figuras 3.43, 3.44, 3.45, se muestran las gráficas de los resultados de la simulación de la SDON de prueba realizada por un periodo de 720 minutos (12 Horas), para la variación de tráfico de 250Mbps, ver resultados en tabla 3.10. Para esta variación se puede observar que los valores de la probabilidad de bloqueo no tienen aumentos considerables en los intervalos de congestión, esto se debe a que se cuenta con un mecanismo cognitivo que es capaz de evadir la congestión por medio del balanceo de carga, haciendo que el rendimiento de la red no se degrade, esto también se aprecia en los valores de retardo extremo a extremo donde se muestra un solo incremento con el valor de 0,426ms en el intervalo de 150-180 minutos, pero en los otros intervalos disminuye y se encuentra entre los valores de 0.01ms y 0.011ms, igualmente al observar la gráfica del Throughput se ve que este incremento del retardo no afecta este parámetro de desempeño, como si sucede en el caso de estudio anterior, es más se mantiene constante con el valor de 262Mbps durante toda la simulación.

A diferencia del caso de estudio 1 en el cual el nodo S1 no tenia la capacidad de tomar una decisión para afrontar la congestión, en este caso de estudio ya se cuenta con un mecanismo cognitivo, de esta forma en el momento en que las pérdidas generadas por el cambio de flujo de ratón a elefante, sumadas las pérdidas del TC empiezan a aumentar, el mecanismo cognitivo por medio del CDS detecta que se sobrepasa el umbral del 1 % que se ha definido, en ese momento el CDS le indica al balanceador de carga que cambie a otro enlace menos congestionado, para hacer esto el flujo que se está enviando por el enlace A es eliminado e inmediatamente se crea un nuevo flujo con las mismas características del anterior pero con un puerto de salida distinto (Enlace B o C), este cambio genera el retardo que se observa en el intervalo de 150-180 minutos, pero como en este escenario el mecanismo solo se activa en una sola ocasión durante toda

la simulación únicamente se observa un solo incremento del retardo extremo a extremo a causa de este efecto.

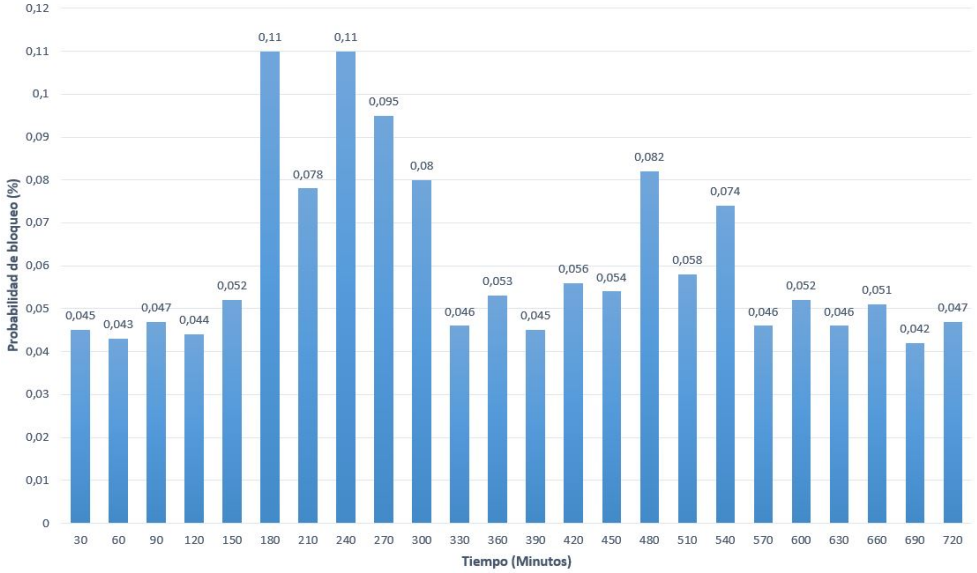

Figura 3.43: Probabilidad de bloqueo con MCCC, escenario 1, variación 1.

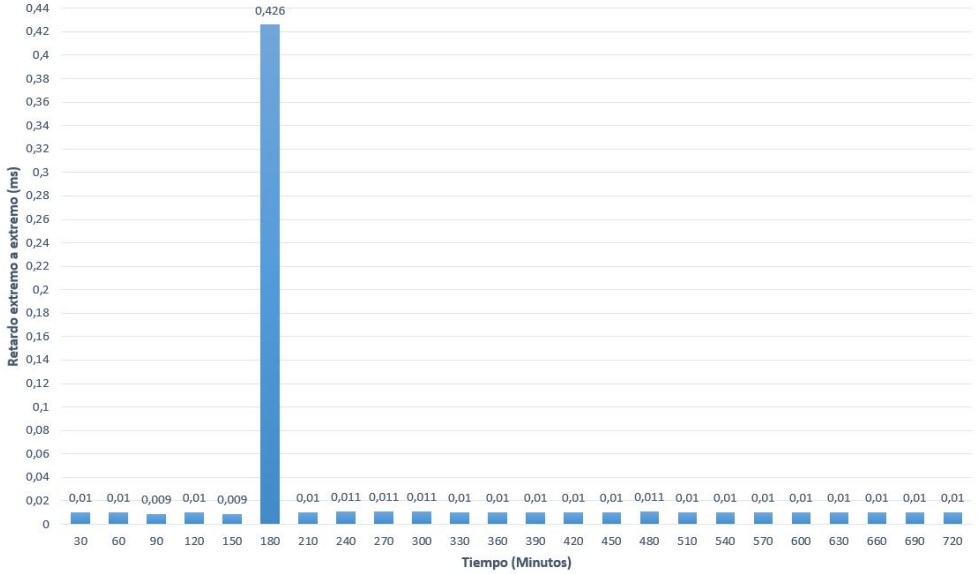

Figura 3.44: Retardo extremo a extremo con MCCC, escenario 1, variación 1.

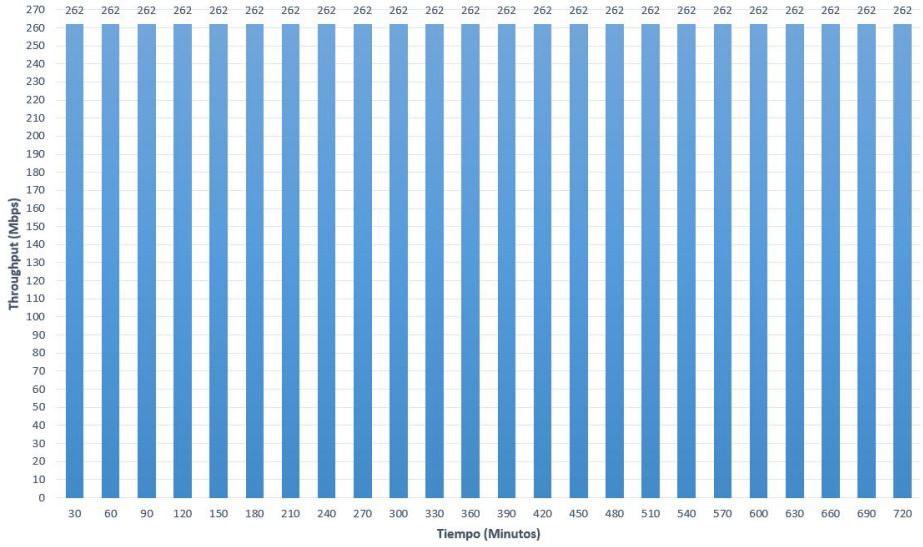

Figura 3.45: Throughput con MCCC, escenario 1, variación 1.

Para una mejor visualización se muestran los resultados para la variación 1 en gráficas de líneas en las figuras 3.46, 3.47, 3.48.

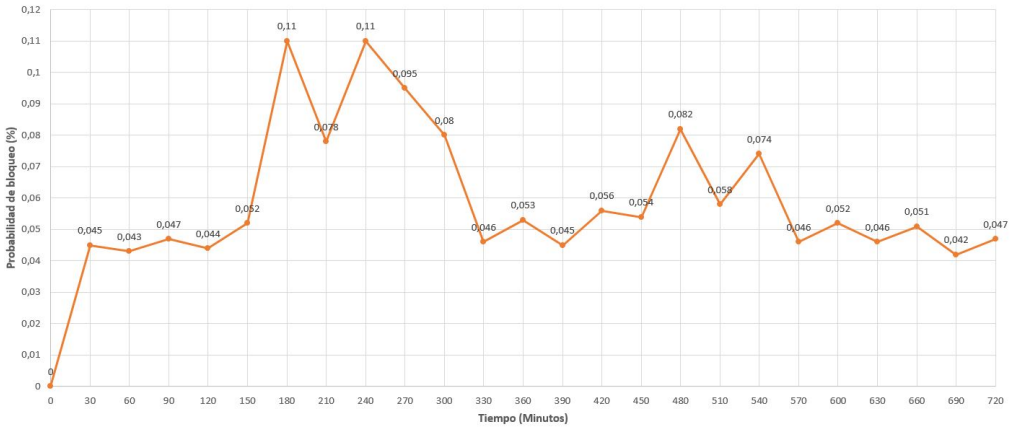

Figura 3.46: Probabilidad de bloqueo con MCCC, escenario 1, variación 1.

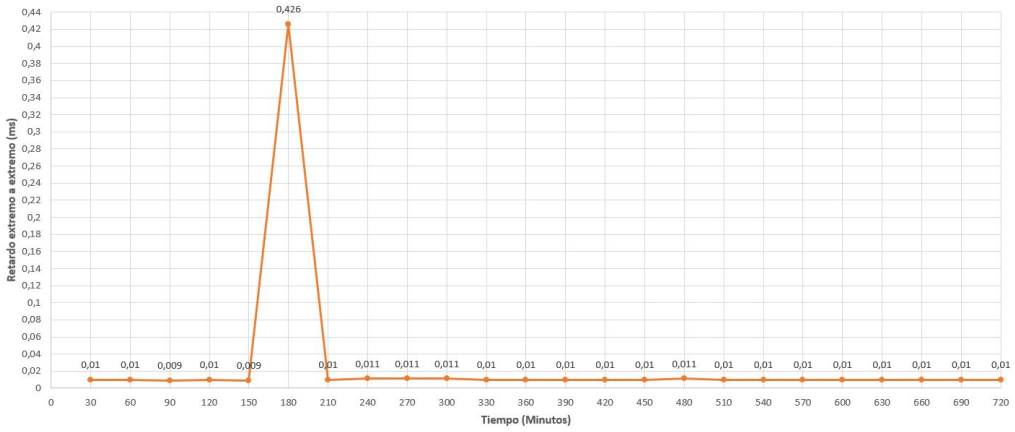

Figura 3.47: Retardo extremo a extremo con MCCC, escenario 1, variación 1.

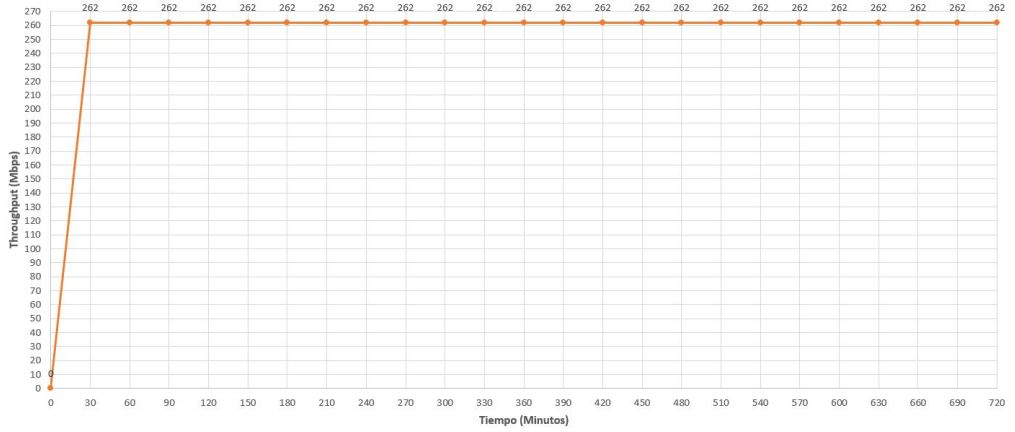

Figura 3.48: Throughput con MCCC, escenario 1, variación 1.

En las figuras 3.49, 3.50, 3.51, se muestran las gráficas de los resultados de la simulación de la SDON de prueba realizada por un periodo de 720 minutos (12 Horas), para la variación de tráfico de 500Mbps, ver resultados en tabla 3.11. Al aumentar la velocidad se puede evidenciar que los valores de los parámetros de desempeño aumentan en los intervalos de congestión, por ejemplo, en el intervalo de 210-240 minutos pasa de 0.11 % a 0.92 % en relación con la variación de 250Mbps, pero a pesar de esto, los valores en la probabilidad de bloqueo son más bajos que los valores del caso de estudio 1, esto se debe a que ya se cuenta con un mecanismo cognitivo. En el caso del retardo extremo a extremo en el intervalo de 150-180 minutos también se incrementa al igual que en la variación 1, alcanzando un valor de 0,199ms, como fue mencionado esto se debe a que

el CDS le indica al balanceador de carga que cambie de puerto a otro enlace menos congestionado, pero al igual que en la variación de 250Mbps, la caída del Throughput no están alta, como lo es en el caso de estudio 1, de hecho, se mantiene casi constante, lo cual indica que el retardo que genera el mecanismo cognitivo no está afectando este parámetro de desempeño.

A diferencia del caso de estudio 1, en el cual cuando ocurría una congestión el nodo S1 continuaba enviando paquetes por el enlace A ocasionando que el Throughput disminuyera, debido a que el host hl1 enviaba menos paquetes a causa del retardo que se generaba en la red, en este escenario no ocurre lo mismo, ya que el enlace es cambiado por uno menos congestionado, permitiendo que el host hl1 no tenga que reducir el número de paquetes que envía, como se observó en la variación 1 donde el Throughput era constante durante toda la simulación, sin embargo, para esta variación se observa que disminuye de entre 1Mbps a 3Mbps a causa del aumento de tráfico, pero a pesar de esto es mucho menor a los resultados del caso de estudio 1.

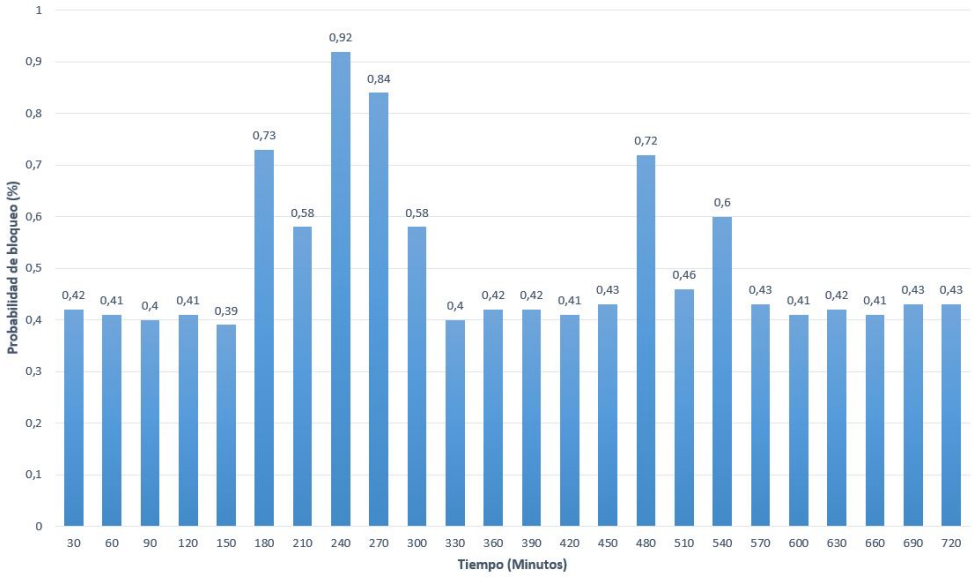

Figura 3.49: Probabilidad de bloqueo con MCCC, escenario 1, variación 2.

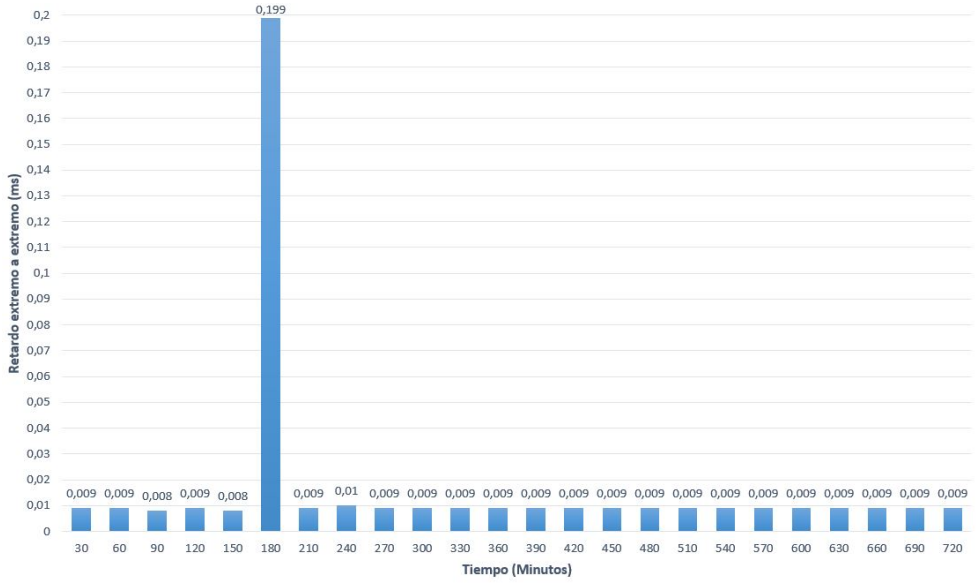

Figura 3.50: Retardo extremo a extremo con MCCC, escenario 1, variación 2.

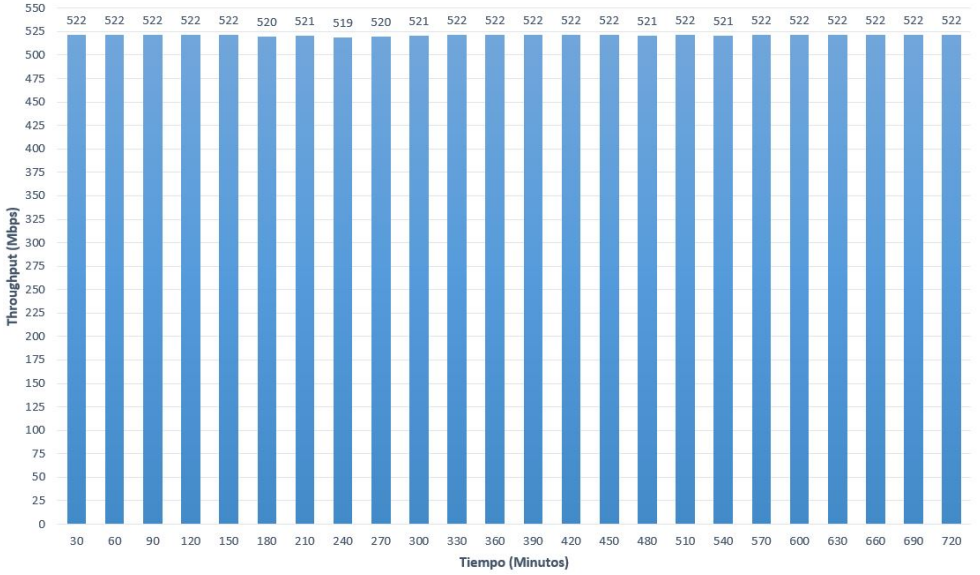

Figura 3.51: Throughput con MCCC, escenario 1, variación 2.

Para una mejor visualización se muestran los resultados para la variación 2 en gráficas de líneas en las figuras 3.52, 3.53, 3.54.

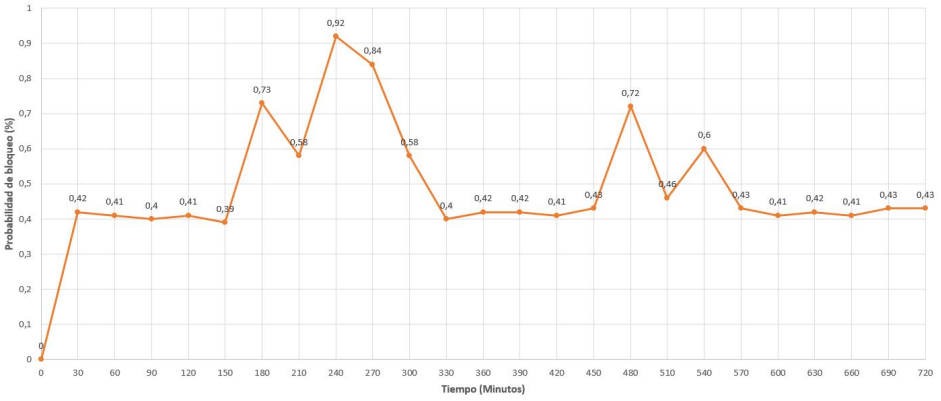

Figura 3.52: Probabilidad de bloqueo con MCCC, escenario 1, variación 2.

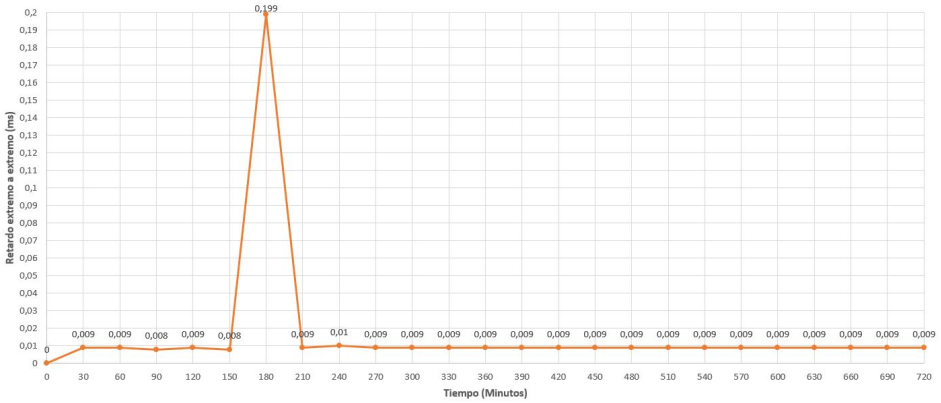

Figura 3.53: Retardo extremo a extremo con MCCC, escenario 1, variación 2.

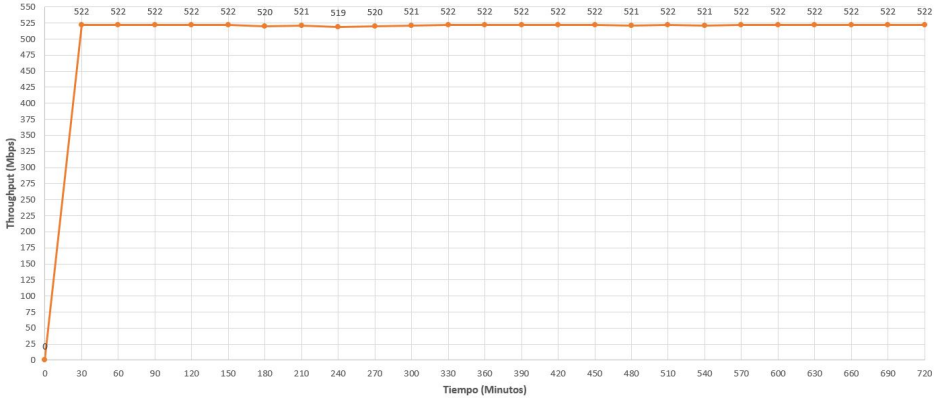

Figura 3.54: Throughput con MCCC, escenario 1, variación 2.

Análisis de resultados escenario 2 En las figuras 3.55, 3.56, 3.57, se muestran las gráficas de los resultados de la simulación de la SDON de prueba rea-

lizada por un periodo de 720 minutos (12 Horas), para la variación de tráfico de 250Mbps, ver resultados en tabla 3.12. En este escenario se observa que los valores de la probabilidad de bloqueo tienen un aumento, que no es considerable, pero sí es mayor al del escenario 1, esto se debe a que el CDS en el caso anterior le indica al balanceador de carga que realice el cambio del puerto solo una vez y este continua con está selección hasta finalizar la simulación, sin embargo, en este caso el CDS mantiene el puerto seleccionado por un tiempo de 10 minutos y luego vuelve al enlace A, pero hay que recordar que los bloques de congestión tienen periodos de tiempo entre los 10 minutos y los 35 minutos, por ejemplo, el primer bloque de congestión es de 25 minutos, lo que quiere decir que el balanceador de carga vuelve al enlace A a los 10 minutos, pero cuando este regrese se va a encontrar nuevamente con la congestión ocasionando que el CDS le indique nuevamente que cambie de puerto, esto hace que los valores aumenten, no solo de este parámetro de desempeño sino del retardo extremo a extremo que a diferencia del escenario anterior donde solo se observa un aumento en el retardo en el intervalo de 150-180 minutos, este ya presenta varios valores debido al cambio de puerto que realiza el balanceador de carga, lo que indica que cada vez que el mecanismo cognitivo detecta congestión en la red, al realizar el balanceo de carga por orden del CDS, se genera un retardo por el cambio de puerto, pese a esto, al igual que en el escenario anterior, donde el Throughput no presenta alteraciones, en este escenario solo cae 1Mbps en el intervalo de 210-240 minutos, el resto de la simulación es constante con un valor de 262Mbps, lo cual indica que los retardos generados por el mecanismo cognitivo no están afectando este parámetro de desempeño.

De igual forma, en las gráficas del retardo extremo a extremo se observa que hay un incremento en los valores de este parámetro de desempeño no solo en los intervalos de congestión sino que también aparece un incremento en el intervalo de 480-510 minutos con un valor de 0.037ms, el cual no está definido como intervalo de congestión, esto quiere decir que el mecanismo cognitivo está detectando más congestión en la red de prueba. Estos incrementos del retardo en puntos no congestionados son originados por el aumento de las funcionalidades que se le dan a la aplicación del mecanismo cognitivo, que para este escenario es cambiar el puerto del nodo S1 cada 10 minutos cuando se presenta congestión, por esta

razón es que estos valores se ven afectados.

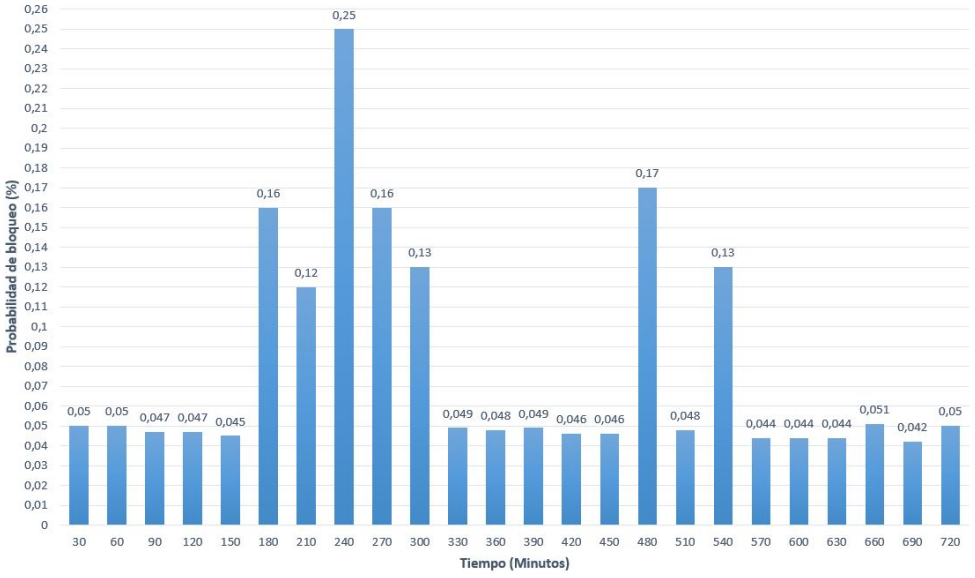

Figura 3.55: Probabilidad de bloqueo con MCCC, escenario 2, variación 1.

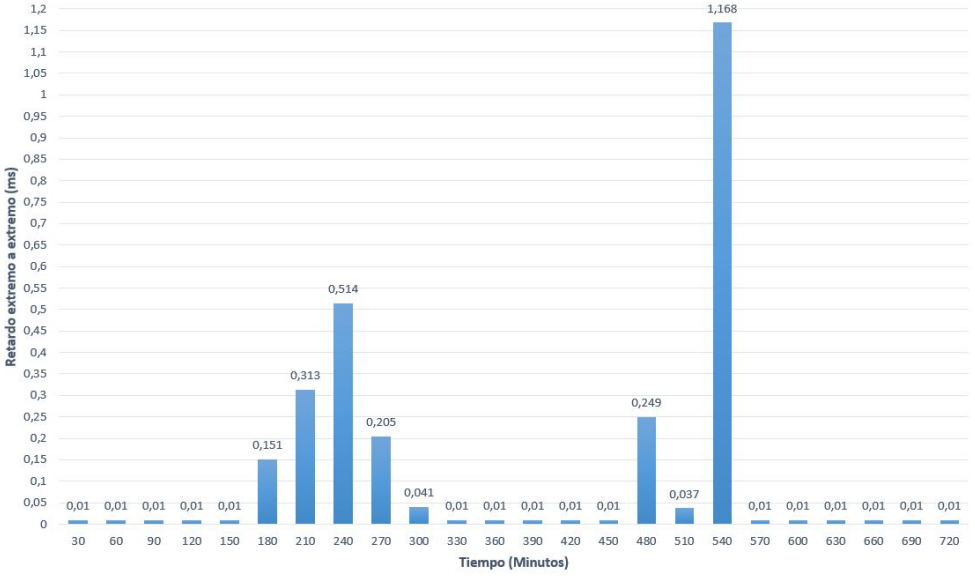

Figura 3.56: Retardo extremo a extremo con MCCC, escenario 2, variación 1.

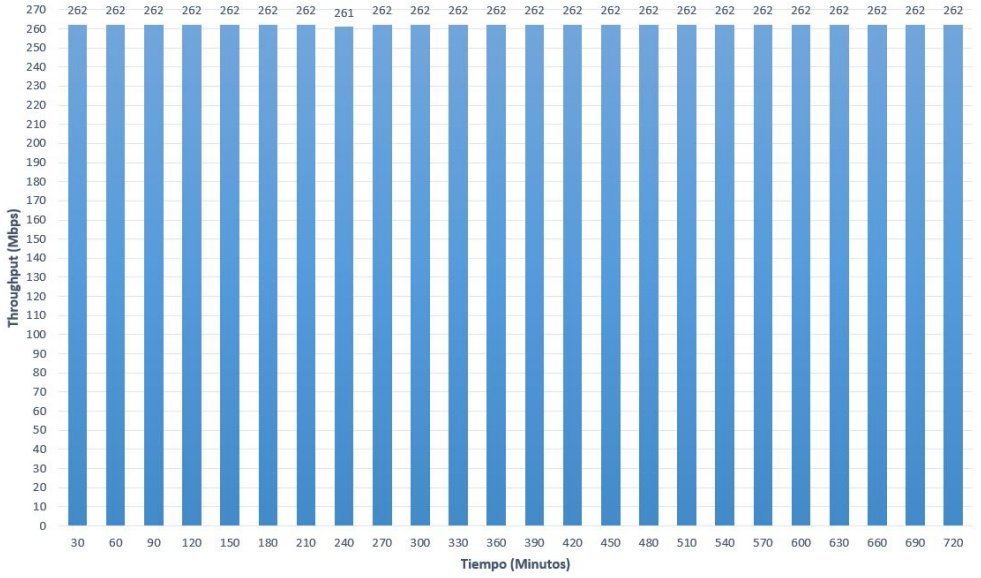

Figura 3.57: Throughput con MCCC, escenario 2, variación 1.

Para una mejor visualización se muestran los resultados para la variación 1 en gráficas de líneas en las figuras 3.58, 3.59, 3.60.

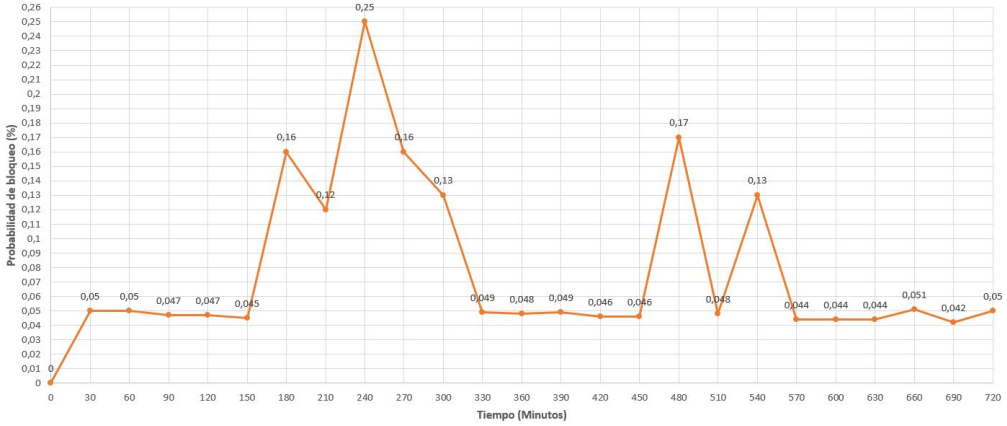

Figura 3.58: Probabilidad de bloqueo con MCCC, escenario 2, variación 1.

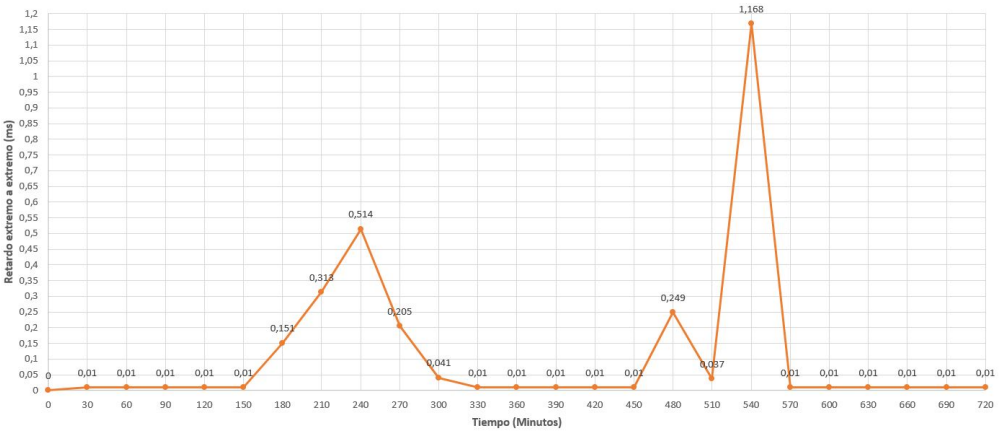

Figura 3.59: Retardo extremo a extremo con MCCC, escenario 2, variación 1.

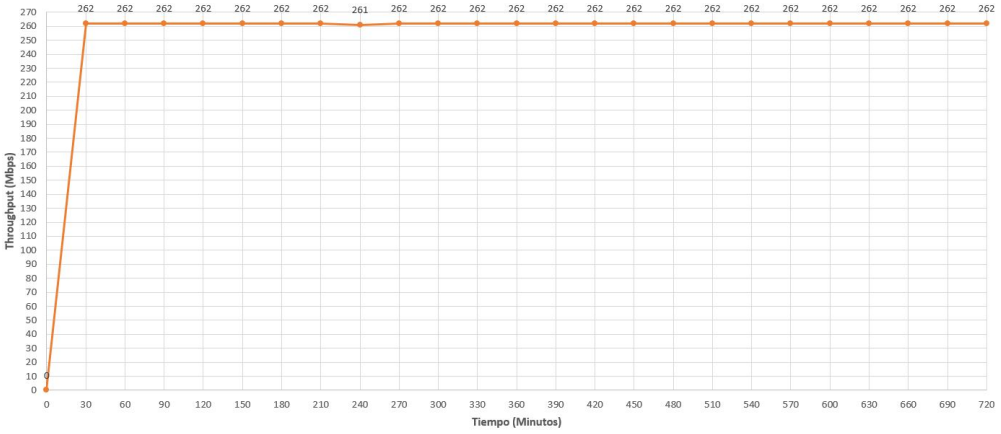

Figura 3.60: Throughput con MCCC, escenario 2, variación 1.

En las figuras 3.61, 3.62, 3.63, se muestran las gráficas de los resultados de la simulación de la SDON de prueba realizada por un periodo de 720 minutos (12 Horas), para la variación de tráfico de 500Mbps, ver resultados en tabla 3.13. En las gráficas se observa que el aumento en el tráfico ocasiona que los valores de la probabilidad de bloqueo aumenten, pero sucede lo mismo que en la variación anterior los valores no son tan grandes, pero sí mayores al los del escenario 1, lo mismo sucede con los valores del retardo extremo a extremo que también aumentan alcanzando un máximo de 0.746ms, además de que empiezan a aparecer más retardos en intervalos donde no se a definido congestión, lo cual quiere decir que el mecanismo cognitivo está detectando más congestión en la red debido al aumento de las funcionalidades que se le dan a la aplicación, que para

este caso, no es solo un aumento como sucedió en la variación anterior, sino que empiezan a aparecer más retardos pero sin afectar su rendimiento como se ve en la gráfica del Throughput que mantiene las condiciones del escenario 1, para la variación de 500Mbps.

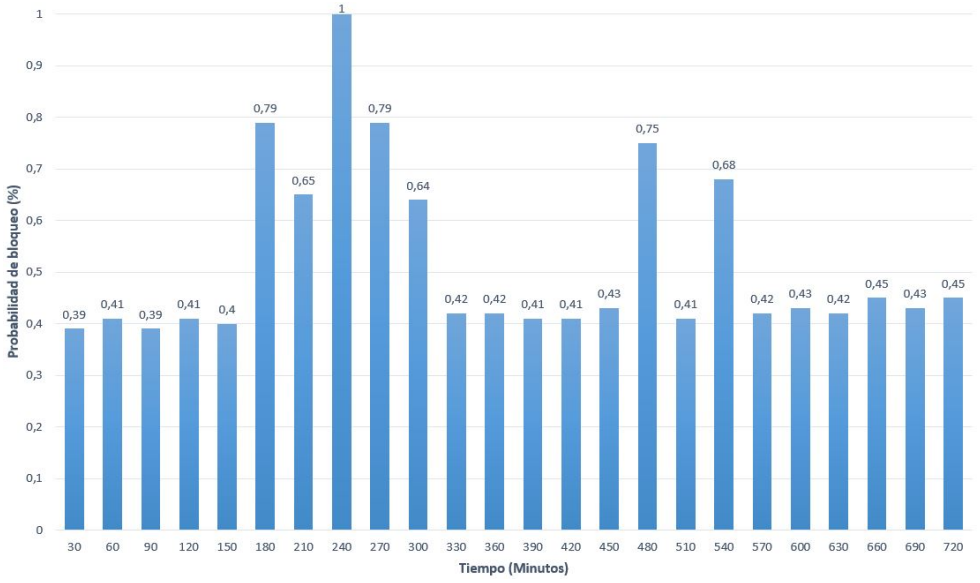

Figura 3.61: Probabilidad de bloqueo con MCCC, escenario 2, variación 2.

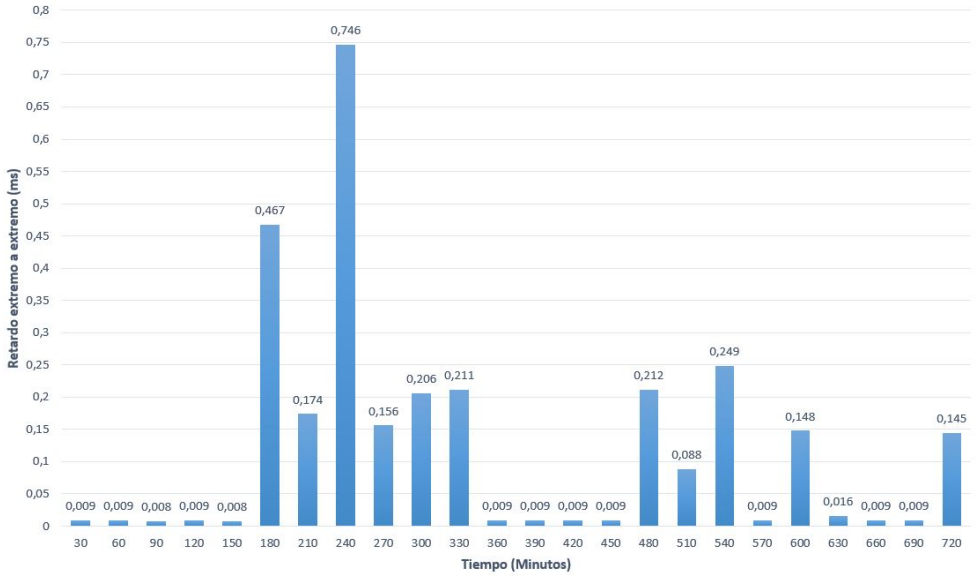

Figura 3.62: Retardo extremo a extremo con MCCC, escenario 2, variación 2.

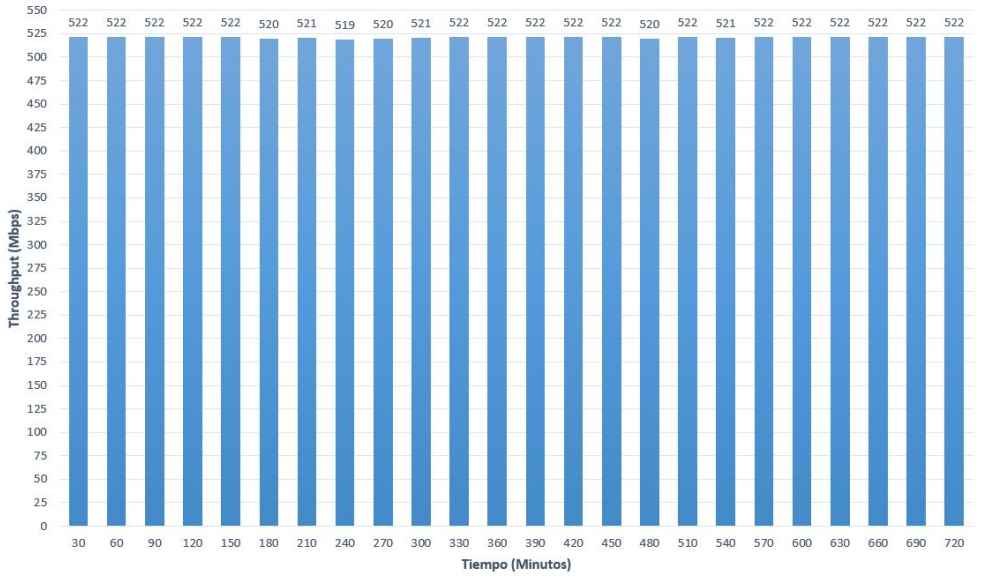

Figura 3.63: Throughput con MCCC, escenario 2, variación 2.

Para una mejor visualización se muestran los resultados para la variación 2 en gráficas de líneas en las figuras 3.64, 3.65, 3.66.

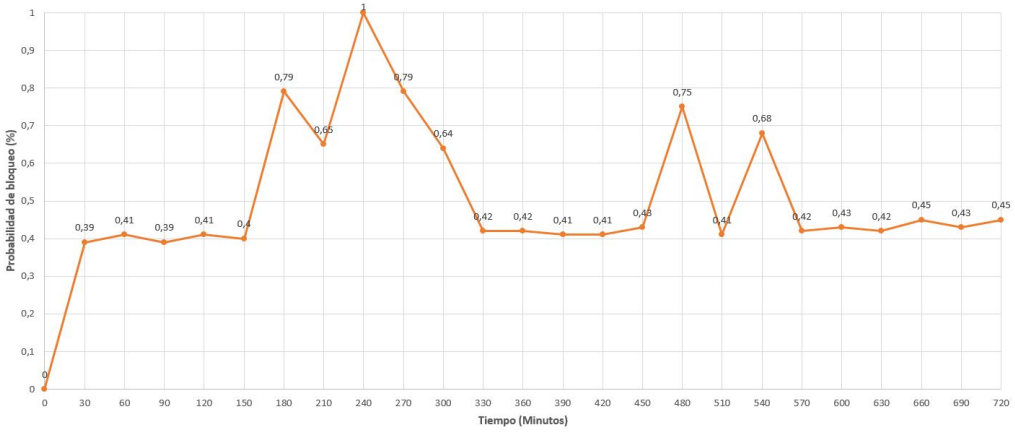

Figura 3.64: Probabilidad de bloqueo con MCCC, escenario 2, variación 2.

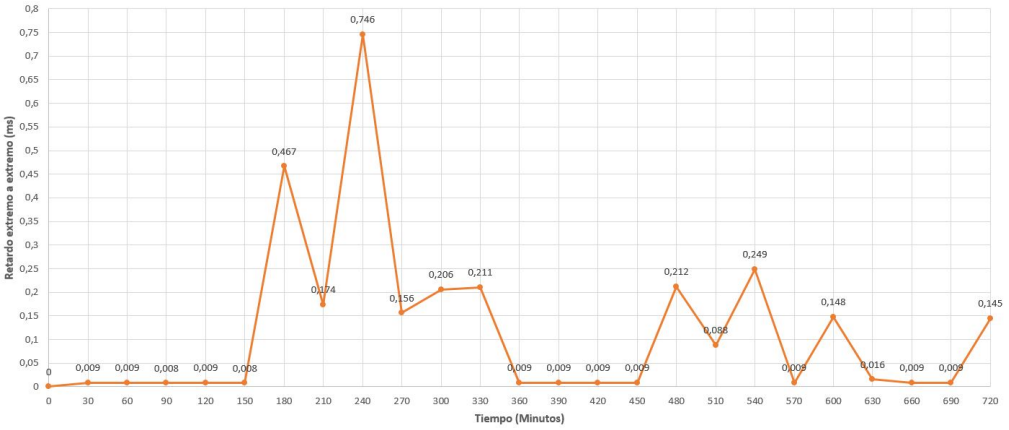

Figura 3.65: Retardo extremo a extremo con MCCC, escenario 2, variación 2.

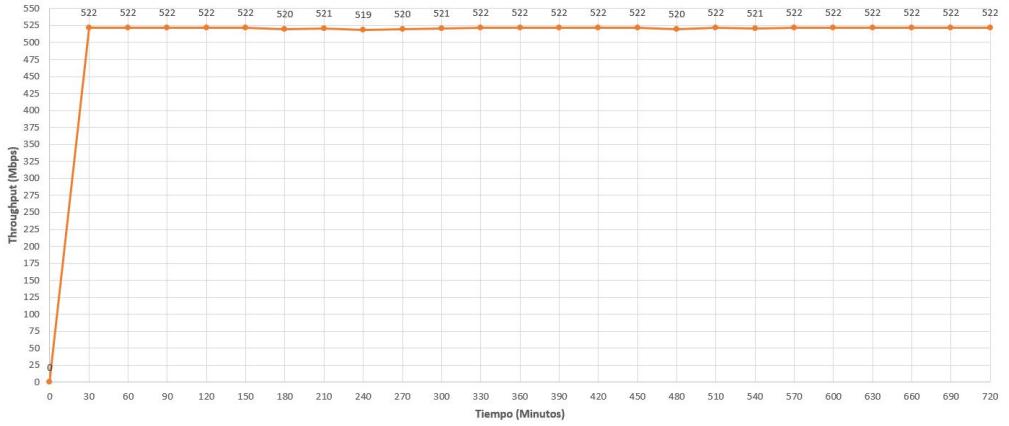

Figura 3.66: Throughput con MCCC, escenario 2, variación 2.

Análisis de resultados escenario 3 En las figuras 3.67, 3.68, 3.69, se muestran las gráficas de los resultados de la simulación de la SDON de prueba realizada por un periodo de 720 minutos (12 Horas), para la variación de tráfico de 250Mbps, ver resultados en tabla 3.14. Las gráficas muestran que los valores de los parámetros de desempeño son menores a los del escenario 2, debido a que el CDS en esta ocasión es más inteligente y solo le indica al balanceador de carga que vuelva al enlace A si este ya no presenta ninguna congestión, lo que quiere decir que, por ejemplo, para el primer bloque de congestión el CDS va a hacer retornar el flujo al enlace A a los 25 minutos y no a los 10 minutos como la hacia en el escenario 2, esto hace que el balanceador de carga no esté

cambiando el puerto, sino que solamente lo hace en el primer momento en que se presenta la congestión y solo se desactiva cuando todo el bloque de congestión termina, este comportamiento del mecanismo cognitivo hace que los valores de la probabilidad de bloqueo sean menores, de hecho, en ocasiones durante la congestión se encuentran valores por debajo de los tomados cuando no la hay, por ejemplo, en los intervalos de 150-180 y 240-270 minutos disminuyen hasta un valor de 0.013 %, pero los valores de los intervalos donde no hay congestión son mayores a cualquiera de los valores de otros escenarios, esto se debe a que en este escenario se aumentan mucho más las funcionalidades que se le da a la aplicación del mecanismo cognitivo, haciendo que ahora realice operaciones que no efectuaba en los anteriores escenarios. Este comportamiento también hace que empiecen a aparecer incrementos en el retardo en intervalos donde no se a definido congestión, por ejemplo, en el intervalo de 300-360 y de 480-510 minutos. A pesar de esto, el Throughput no se ve afectado, porque este parámetro se mantiene constante durante toda la simulación, lo que indica que los incrementos del retardo que genera el mecanismo cognitivo no afectan este parámetro de desempeño.

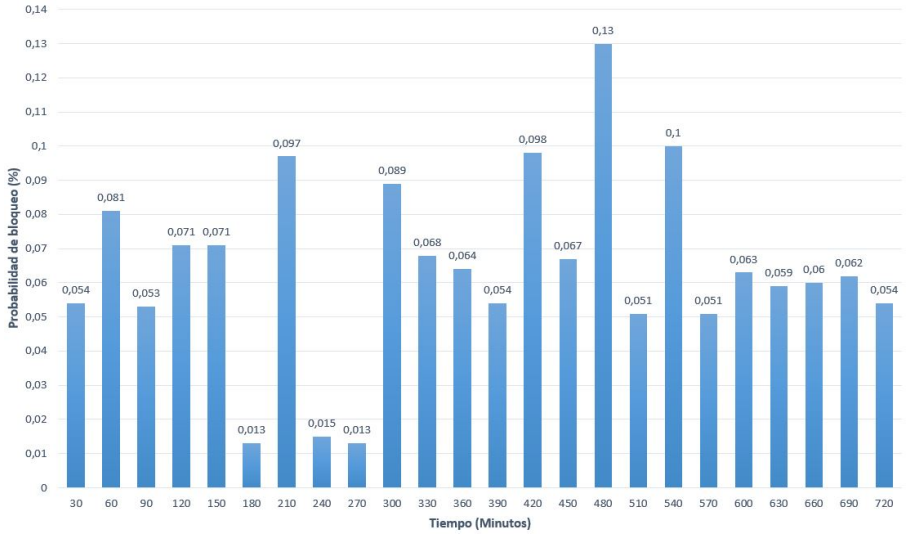

Figura 3.67: Probabilidad de bloqueo con MCCC, escenario 3, variación 1.

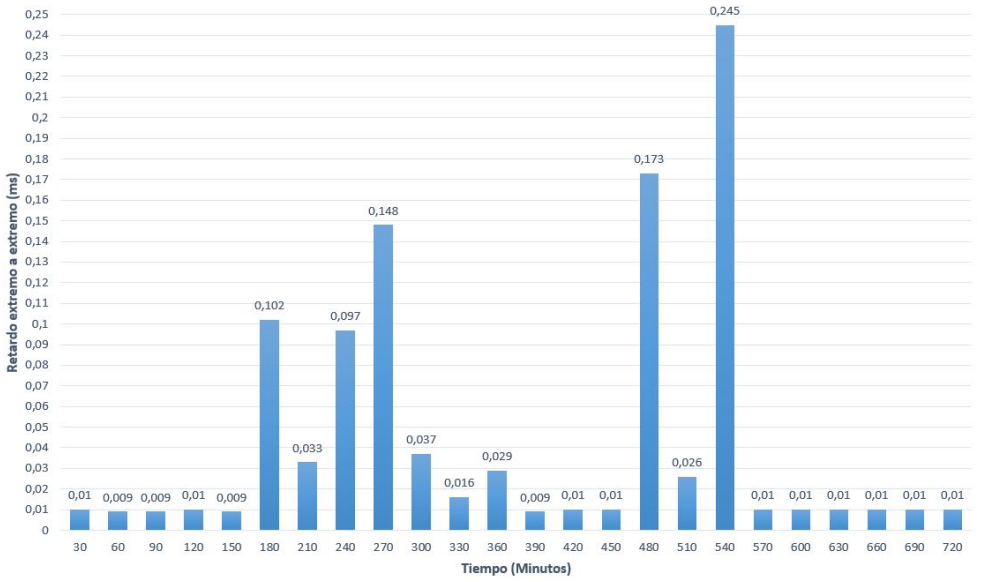

Figura 3.68: Retardo extremo a extremo con MCCC, escenario 3, variación 1.

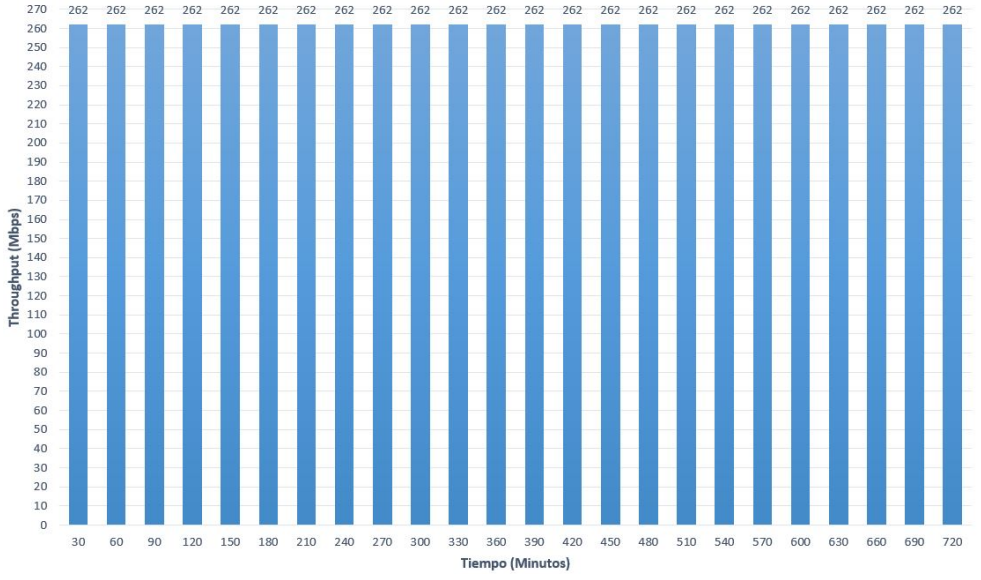

Figura 3.69: Throughput con MCCC, escenario 3, variación 1.

Para una mejor visualización se muestran los resultados para la variación 1 en gráficas de líneas en las figuras 3.70, 3.71, 3.72.

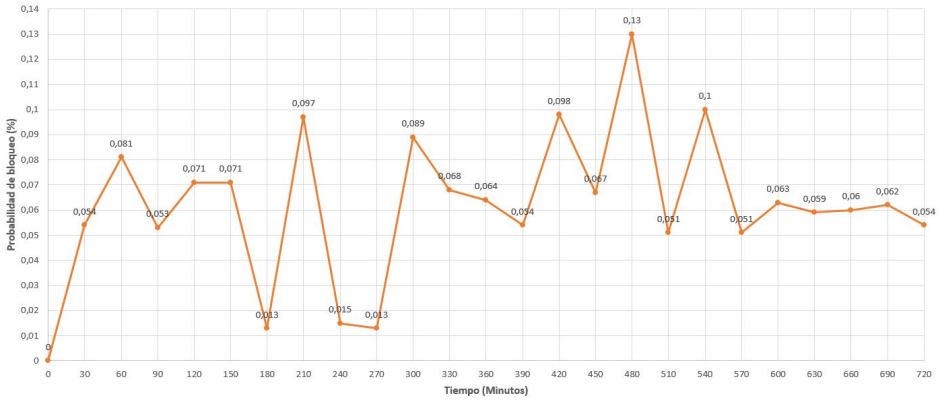

Figura 3.70: Probabilidad de bloqueo con MCCC, escenario 3, variación 1.

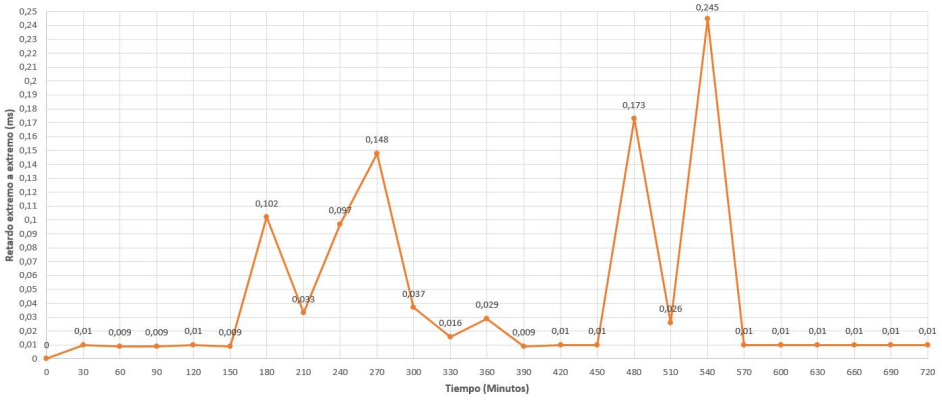

Figura 3.71: Retardo extremo a extremo con MCCC, escenario 3, variación 1.

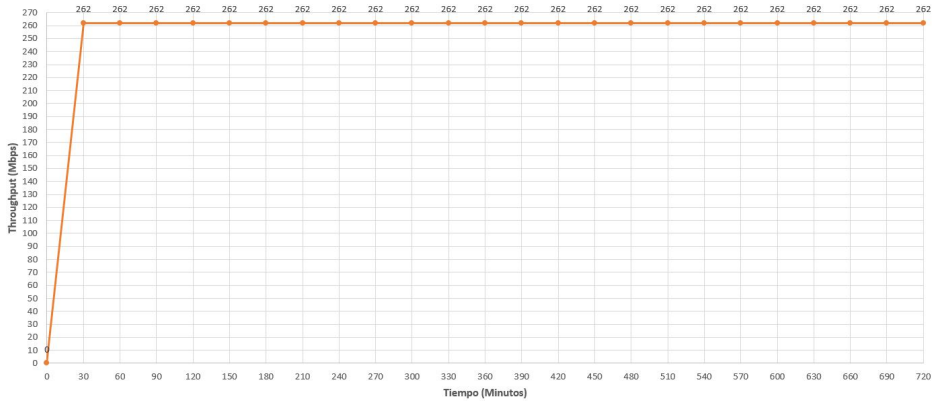

Figura 3.72: Throughput con MCCC, escenario 3, variación 1.

En las figuras 3.73, 3.74, 3.75, se muestran las gráficas de los resultados de la simulación de la SDON de prueba realizada por un periodo de 720 minutos (12 Horas), para la variación de tráfico de 500Mbps, ver resultados en tabla 3.15. En los resultados se puede ver que el valor de la probabilidad de bloqueo es muy similar a la del escenario 2, de hecho, en este caso los valores son mínimamente más elevados, pasando de 0.75 % a 0.84 % en el intervalo de 450-480 minutos y de 0.68 % a 0.77 % en 510-540 minutos, lo mismo se aprecia en el retardo extremo a extremo, donde hay momentos en que estos valores son más altos y en otros más bajos, pero al igual que la variación de 250Mbps empiezan a aparecer más retardos, lo que indica que el mecanismo cognitivo esta detectando más congestión en la red, sin embargo, a diferencia de la variación anterior en esta ocasión se puede observar como el aumento de las funcionalidades sumado el incremento del tráfico afectan más la red, ocasionando que aparezcan retardos en la mayor parte de los periodos donde no se ha definido congestión, sin embargo, en la gráfica del Throughput se puede ver que este parámetro de desempeño es casi constante durante toda la simulación, de hecho, mantiene las condiciones de los anteriores escenarios para esta misma variación, lo que indica que los retardos generados por el mecanismo cognitivo al cambiar el puerto del nodo S1 no están afectando este parámetro de desempeño.

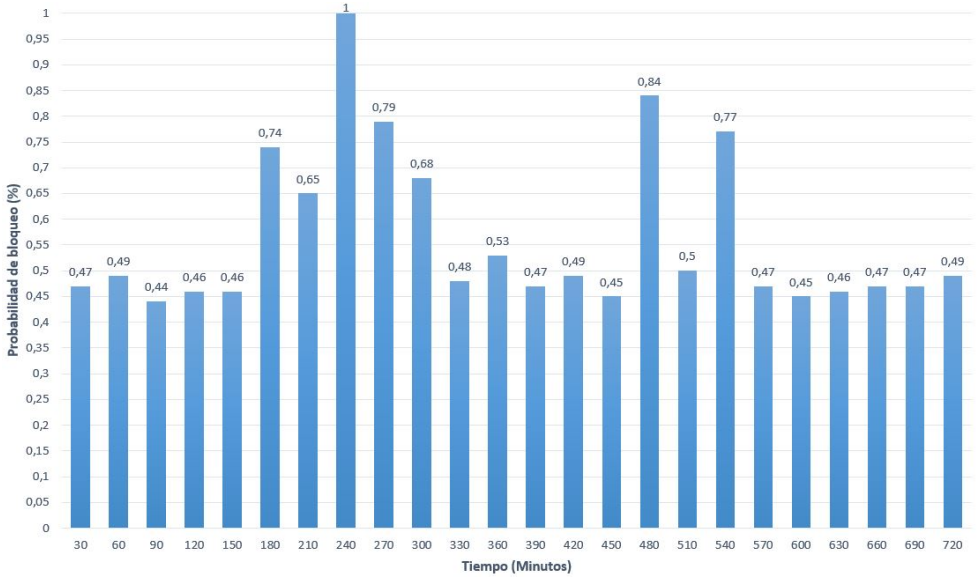

Figura 3.73: Probabilidad de bloqueo con MCCC, escenario 3, variación 2.

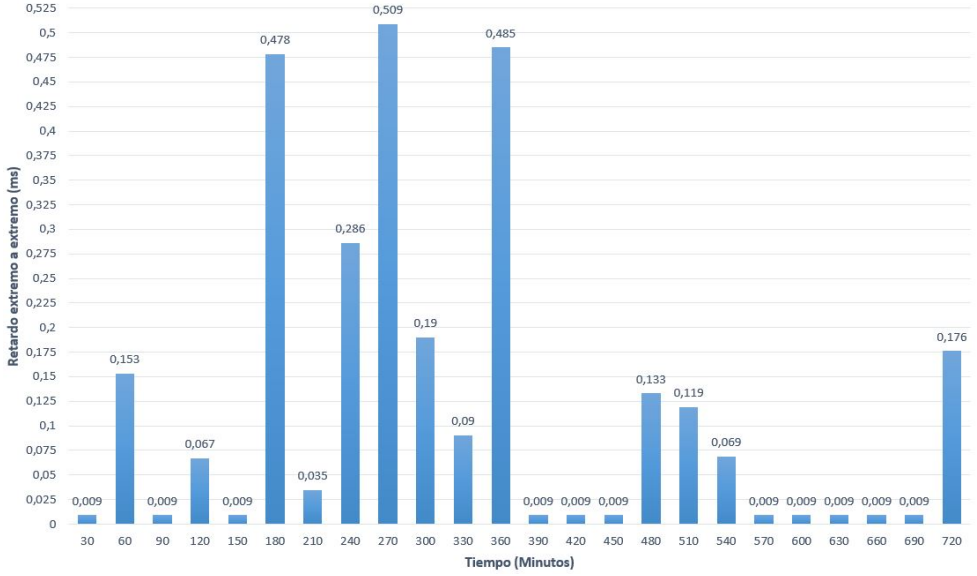

Figura 3.74: Retardo extremo a extremo con MCCC, escenario 3, variación 2.

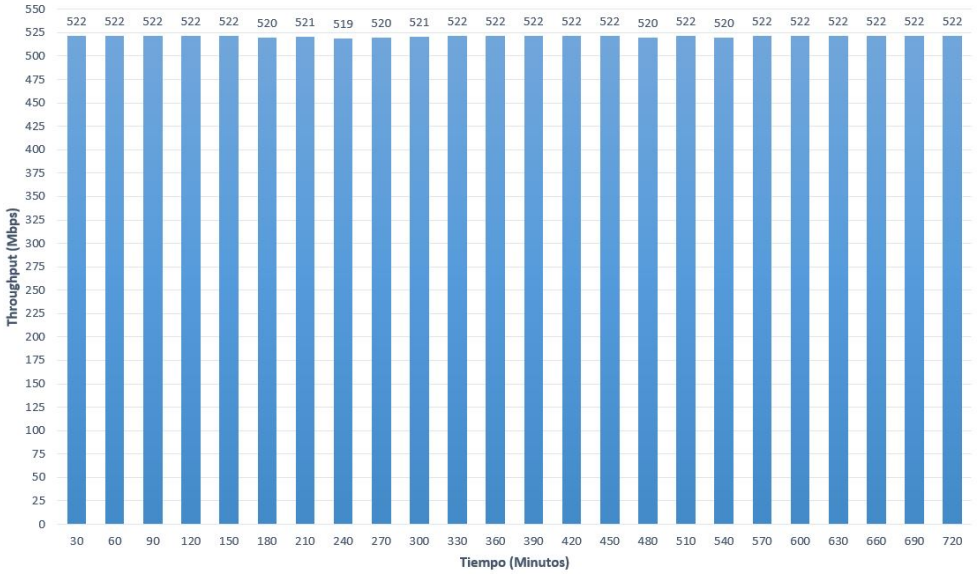

Figura 3.75: Throughput con MCCC, escenario 3, variación 2.

Para una mejor visualización se muestran los resultados para la variación 2 en gráficas de líneas en las figuras 3.76, 3.77, 3.78.

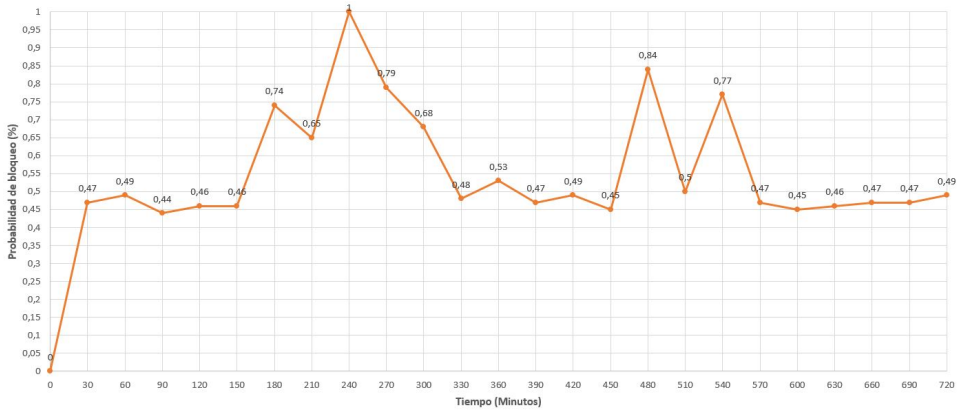

Figura 3.76: Probabilidad de bloqueo con MCCC, escenario 3, variación 2.

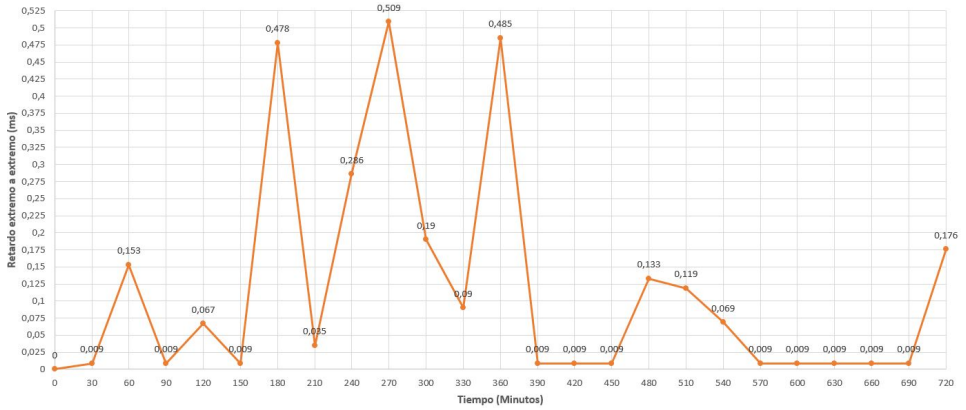

Figura 3.77: Retardo extremo a extremo con MCCC, escenario 3, variación 2.

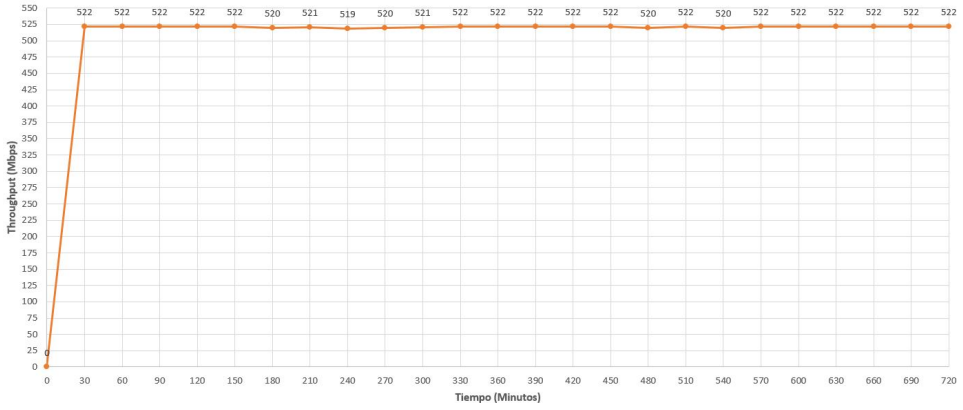

Figura 3.78: Throughput con MCCC, escenario 3, variación 2.

Al analizar todos los escenarios descritos en este caso de estudio se observa que a medida que se aumenta el tráfico en la red los valores de la probabilidad de bloqueo se van elevando, otro factor importante que se observa es que al aumentar las funciones que tiene el mecanismo cognitivo por medio de la aplicación, los valores de la probabilidad de bloqueo también van aumentando respecto a los escenarios anteriores, es decir para la variación de 250Mbps, en el escenario 1, los valores mínimos son de alrededor de 0.042 %, en el segundo escenario, este mínimo solo aparece una vez, el siguiente valor es de 0.044 % y en el tercer escenario, los valores están por encima del 0.05 %, sin tener en cuenta los valores mínimos de 0.013 % y 0.015 %, que son a causa del mecanismo cognitivo, asimismo sucede con los resultados de la variación de 500Mbps. Este efecto también se ve reflejado en el retardo extremo a extremo donde se observa que a medida que estas funcionalidades aumenten y a la vez se aumenta el tráfico en la red se observa que el mecanismo cognitivo empieza a detectar más congestión en los intervalos donde no se a definido, lo que conlleva a que el CDS le indique al balanceador de carga que cambie de puerto ocasionando este incremento. Pero como se observó en las gráficas de Throughput para cualquiera de los escenarios del caso de estudio 2, estos incrementos que genera el mecanismo cognitivo por el cambio de puerto no afectan este parámetro de desempeño.

Finalmente, es de destacarse que de entre los tres escenarios el que cuenta con los mejores resultados es el escenario 1, el motivo es que solo realiza el balanceo de carga en una sola ocasión y luego va a estar enviando el tráfico por este enlace que ya no presenta más congestión, pero dejando a un lado este escenario. En resultados se puede observar que el mejor modo de operación para el mecanismo cognitivo es el escenario 3, debido a que en este el mecanismo cognitivo es capaz de detectar la congestión y la evita durante todo el bloque de esta, en cambio, el escenario 2 no lo hace, a causa de que en este el CDS le indica al balanceador de carga que vuelva al enlace A después de un periodo de tiempo y no se tiene en cuenta si el enlace todavía está congestionado haciendo que los resultados se vean afectados por esta razón. Pero independientemente de cualquiera de estos escenarios los resultados son mejores al primer caso de estudio cuando se presenta congestión sobre la red de prueba.

3.9.3. Análisis comparativo de una arquitectura de red SDON con y sin mecanismo cognitivo

Este trabajo de grado busca aportar en el desarrollo de técnicas que permitan disminuir la congestión en redes ópticas definidas por software, por esta razón se evalúa el desempeño del mecanismo cognitivo basado en CHRON y SDN, por medio de parámetros de desempeño tales como probabilidad de bloqueo, retardo extremo a extremo y Throughput, con la finalidad de observar cómo se comporta una red que no cuenta con un mecanismo cognitivo y una red que se le incluye un mecanismo que es capaz de tomar decisiones basado en las estadísticas de su entorno.

Revisando los resultados de los parámetros de desempeño se observa que el flujo elefante utilizado para congestionar la red, genera retardos sobre el enlace A, ocasionando que la red se degrade debido a que el nodo tarda más tiempo en procesarlos, este efecto ocasiona que la probabilidad de bloqueo aumente al igual que el retardo extremo extremo y por consiguiente, el Throughput disminuya, pero cuando se incluye el mecanismo cognitivo para controlar la congestión, el nodo ya tiene la capacidad de cambiar a otro puerto menos congestionado evitando así la congestión y haciendo que los parámetros de desempeño no se vean afectados, lo cual indica que el mecanismo cognitivo si está ayudando a reducir la congestión en la red óptica definida por software.

Al analizar los resultados de los dos casos de estudio se puede observar que la falta del mecanismo cognitivo en el primer caso de estudio hace que los valores de la probabilidad de bloqueo sean más altos que en el cualquiera de las variaciones del segundo caso de estudio. Las figuras 3.79, 3.80, 3.81, 3.82, 3.83, 3.84, muestran el resumen de los parámetros de desempeño analizados en este trabajo de grado tanto para la variación de 250Mbps como de 500Mbps y es evidente que el uso del mecanismo cognitivo ayuda a mejorar el rendimiento de la red.

Todos los escenarios del segundo caso de estudio muestran una mejora evidente frente al primer caso de estudio, por ejemplo, el segundo escenario muestra los mejores resultados debido a que el balanceador de carga solo cambia de puerto

una vez, esto hace que solo se muestre un incremento en los valores del retardo en el intervalo de 150-180 minutos, el segundo escenario muestra varios incrementos al igual que el tercer escenario en este parámetro de desempeño, pero en las gráficas de Throughput se puede ver que estos retardos no afectan este parámetro, tan solo cuando no se cuenta con el mecanismo cognitivo es cuando se observa que el Throughput disminuye.

Por último, se hace referencia al aumento de los valores de la probabilidad de bloqueo, esto se puede observar en las gráficas 3.79 y 3.82 que si el tráfico es aumentado los valores de este parámetro se incrementan debido a que la cantidad de paquetes que llegan a los nodos es mayor, de esta forma la respuesta del nodo no es lo suficientemente buena como para evitar la pérdida de los paquetes, así que entre más tráfico circule por la red, más grandes serán las pérdidas. Lo anterior afecta la probabilidad de bloqueo, pero al analizar los resultados de los casos de estudio, se observa que el incremento de la probabilidad de bloqueo para el segundo caso de estudio se va incrementando a medida que las funciones que se le dan al mecanismo aumentan, por ejemplo, si se observa la figura 3.82 se espera que como la variación de tráfico es de 500Mbps para todos los casos de estudio el valor de la probabilidad de bloqueo se mantenga estable, pero a medida que se le van dando funcionalidades al mecanismo cognitivo estos valores aumentan, en el primer caso de estudio los valores se mantienen entre 0.37 % y 0.4 %, pero en el tercer escenario del segundo caso de estudio estos valores son de 0.45 % y 0.53 %, cuando no se presenta congestión, dicho lo anterior es evidente que se presenta un leve incremento debido a esta razón. De igual forma, sucede con el retardo extremo a extremo, donde se puede ver en las figuras 3.80 y 3.83 para las dos variaciones cómo a medida que se aumentan las funcionalidades del mecanismo cognitivo empiezan a aparecer más retardos en intervalos donde no se a definido congestión, siendo el escenario 3 donde más se observa este efecto.

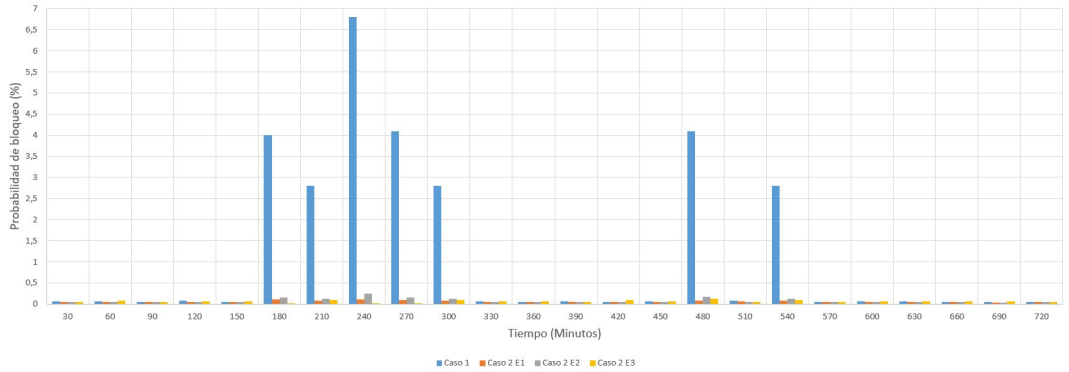

Figura 3.79: Probabilidad de bloqueo para la variación de 250Mbps.

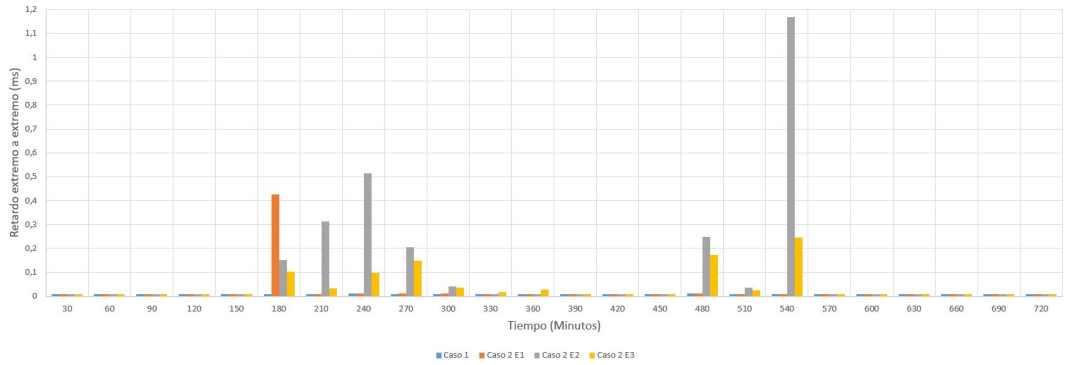

Figura 3.80: Retardo extremo a extremo para la variación de 250Mbps.

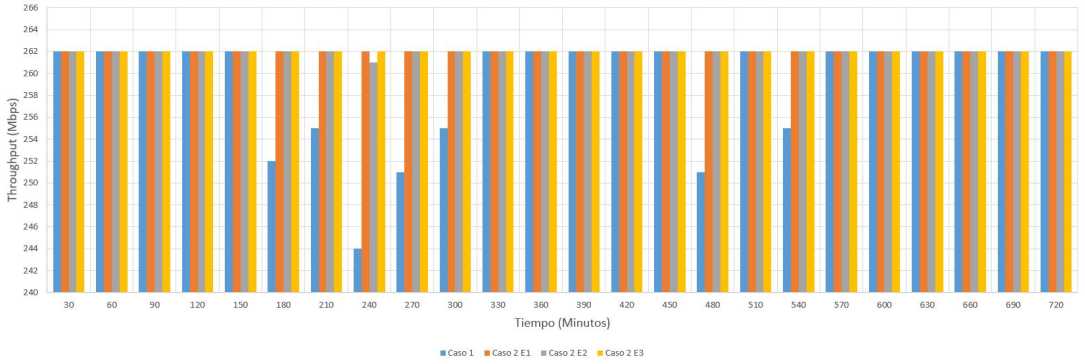

Figura 3.81: Throughput para la variación de 250Mbps.

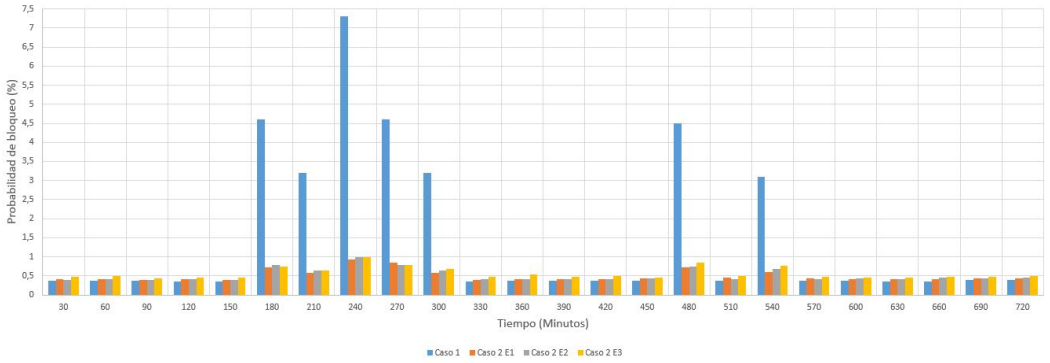

Figura 3.82: Probabilidad de bloqueo para la variación de 500Mbps.

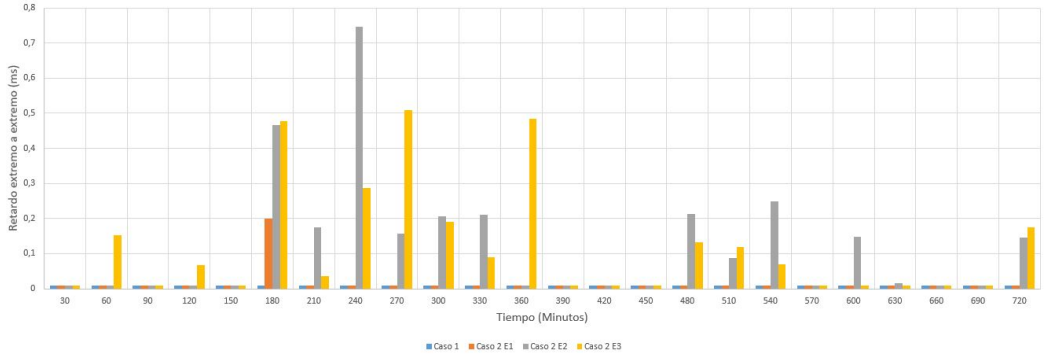

Figura 3.83: Retardo extremo a extremo para la variación de 500Mbps.

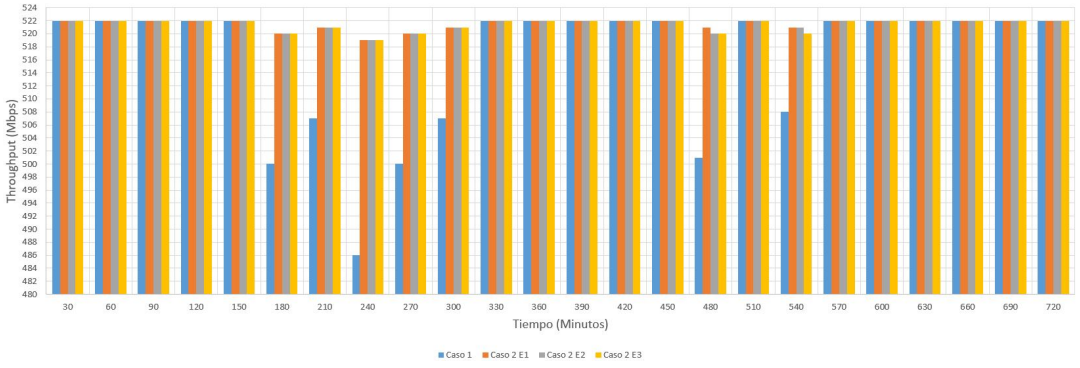

Figura 3.84: Throughput para la variación de 500Mbps.

Para una mejor visualización se muestran los resultados de los parámetros de desempeño en gráficas de líneas en las figuras 3.85, 3.86, 3.87, 3.88, 3.89, 3.90.

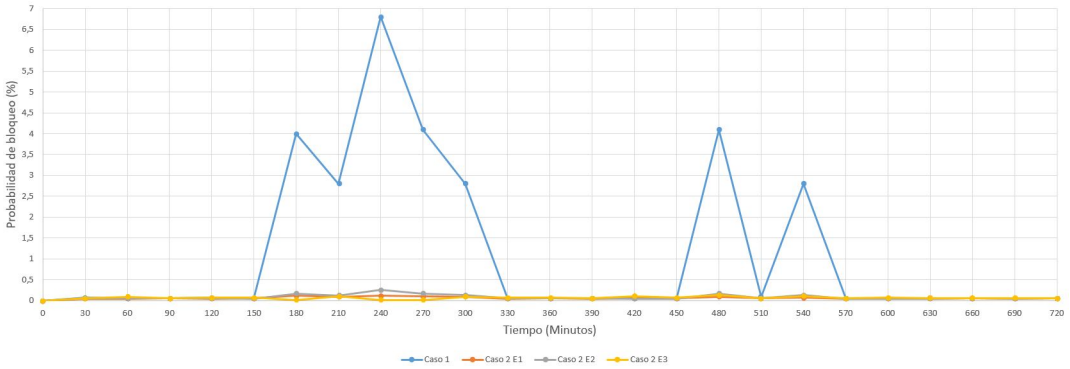

Figura 3.85: Probabilidad de bloqueo para la variación de 250Mbps.

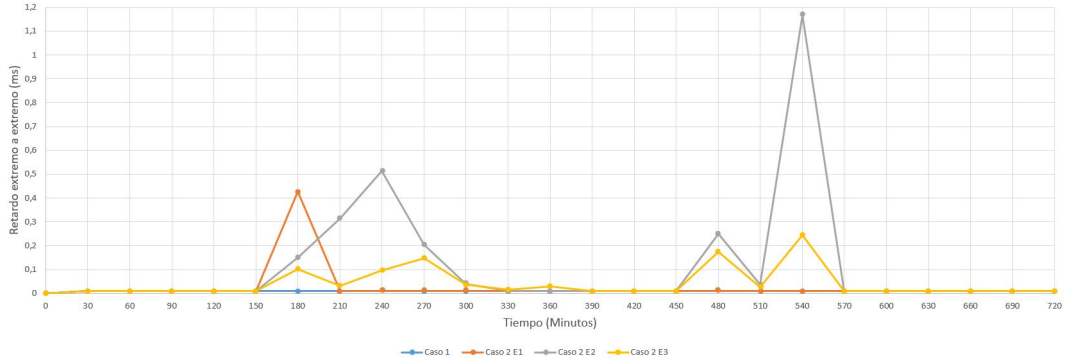

Figura 3.86: Retardo extremo a extremo para la variación de 250Mbps.

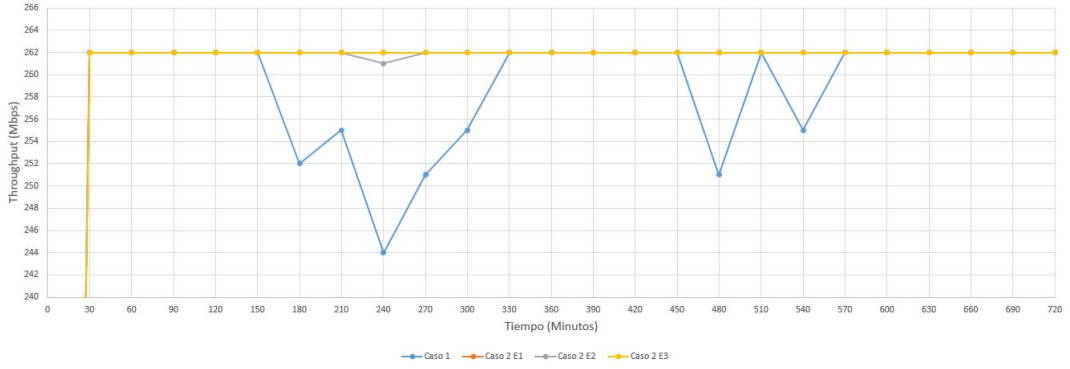

Figura 3.87: Throughput para la variación de 250Mbps.

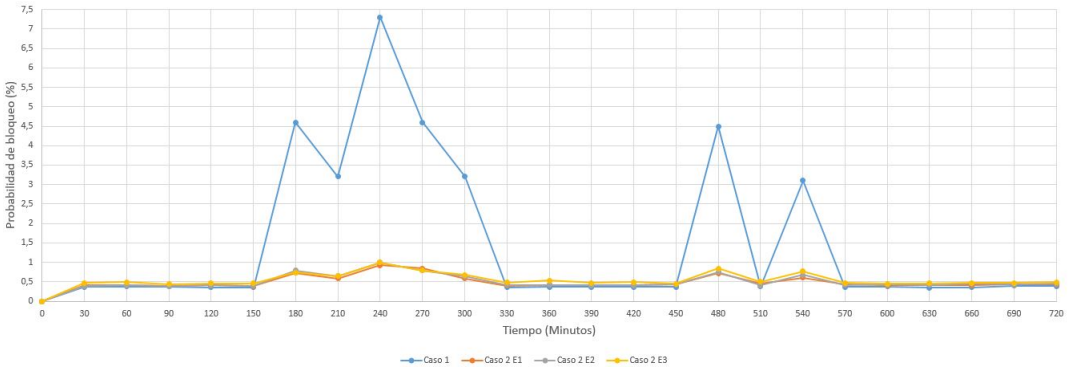

Figura 3.88: Probabilidad de bloqueo para la variación de 500Mbps.

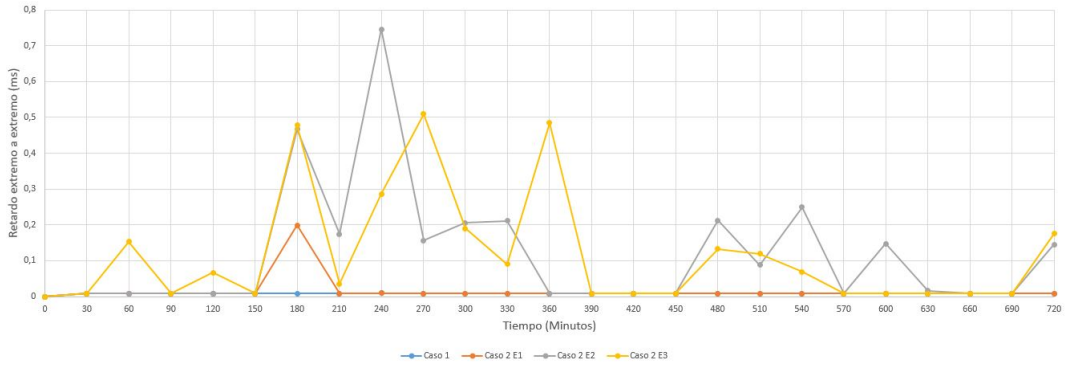

Figura 3.89: Retardo extremo a extremo para la variación de 500Mbps.

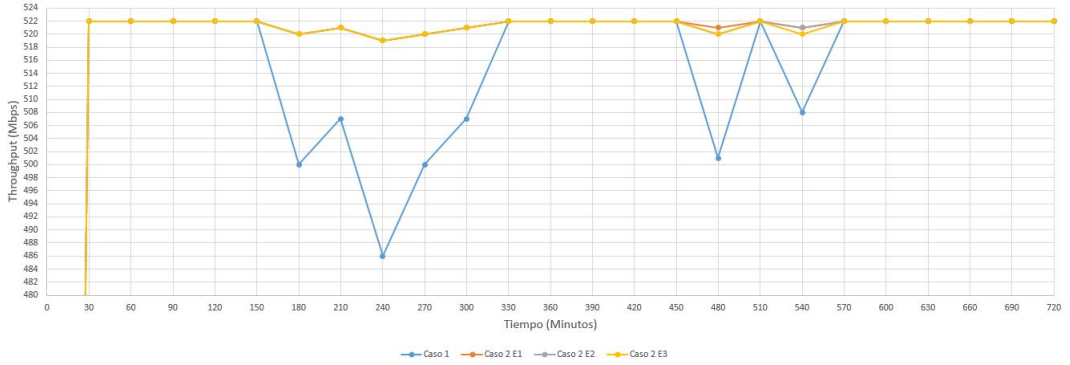

Figura 3.90: Throughput para la variación de 500Mbps.

Finalmente en el capitulo siguiente se describen las conclusiones, recomendaciones y trabajos futuros que radican a partir del desarrollo de este trabajo de grado.

Capítulo 4

Conclusiones, recomendaciones y trabajos futuros

En este capítulo se presentan las conclusiones, recomendaciones y trabajos futuros. Las conclusiones surgen a partir del desarrollo del trabajo de grado, los resultados de las simulaciones y la herramienta utilizada para desarrollar el mecanismo cognitivo. Las recomendaciones son el resultado de la metodología del trabajo y la herramienta de simulación. Los trabajos futuros se presentan respecto a la ejecución de la simulación y las redes ópticas definidas por software.

4.1. Conclusiones

- La implementación del mecanismo cognitivo basado en CHRON y SDN muestra que es posible aumentar la inteligencia de los nodos de la red por medio de balanceo de carga para enfrentar los problemas que pueda presentar la red a medida que el tráfico aumenta.
- El desarrollo del mecanismo cognitivo bajo una estructura de balanceador de carga, permite que la congestión disminuya de acuerdo a los parámetros establecidos, esto se evidencia en los resultados de la probabilidad de bloqueo la cual disminuye cuando el MCCC ya se ha implementado.

- En los resultados del segundo caso de estudio se puede evidenciar que el MCCC genera variaciones en el retardo extremo a extremo a causa de que el balanceador de carga cambia de puerto, pero es de destacarse que los retardos generados no afectan el rendimiento de la red, lo cual se ve reflejado en los resultados del throughput.

- Se puede evidenciar que el aumento de tráfico afecta el rendimiento de un nodo, en su capacidad de procesar la información, de igual forma este efecto también es ocasionado por el aumento de las funciones que se le pueda dar al nodo por medio del controlador *ONOS*.

- La centralización de una red por medio de un controlador permite el monitoreo constante de la información que al final resulta en la elaboración de aplicaciones que pueden utilizar los datos para así poder tomar decisiones con autonomía en el caso de que la red presente problemas como aumento de la probabilidad de bloqueo o del retardo extremo a extremo.

- Se comprueba que el enrutamiento de *ONOS* (*Reactive Forwarding*) no es capaz de enfrentar un problema de congestión por sí solo, sino que necesita de un mecanismo cognitivo que le permita aumentar sus capacidades de decisión y planeación para poder disminuir la congestión de la red.

- El desarrollo de este trabajo de grado permite observar que para la centralización de una red se requiere de computadores con capacidades de procesamiento muy altas y más si se hace necesario el uso de muchos dispositivos en la red.

- La topología con 40 dispositivos evidencia la eficiencia del mecanismo cognitivo en la SDON de prueba reducida con las condiciones expuestas en el desarrollo de este trabajo de grado, sin embargo, no prueba que funcione en la topología con el modelo óptico completo y la NSFNET incluidos, debido a que las limitaciones en hardware no permiten comprobar la eficiencia del mismo por la falta de capacidad de procesamiento de los computadores utilizados.

- Se evidencia que la herramienta *Mininet* puede ser utilizada para emular el funcionamiento de un *WDM switch fabric*, además de ser una herramienta muy versátil que permite crear topologías de cualquier tipo para luego ser fácilmente enlazadas con el controlador *ONOS* por medio del protocolo *OpenFlow*.

4.2. Recomendaciones

- Se recomienda que al implementar redes con alto número de dispositivos se utilicen recursos considerables cuando se virtualiza el entorno de desarrollo, debido a que este aumento puede afectar el rendimiento del entorno de trabajo.
- Es importante definir una metodología de simulación que se adapte a las necesidades que se pueden presentar en el desarrollo del trabajo de grado con el fin de evaluar de forma concisa y ordenada los resultados que se obtengan en la ejecución de la simulación.
- El uso de la API de ONOS implica tener conocimientos en *Java* debido a que es el lenguaje de programación que se utiliza para la elaboración de las aplicaciones, de igual forma se necesita tener conocimiento en *Python* debido a que *Mininet* está basado en este lenguaje.
- Cuando se elabora la aplicación en *Onos* se recomienda utilizar la última versión de la Api de Onos debido a que algunas clases pueden estar obsoletas y al momento de realizar la compilación de la aplicación pueden aparecer errores.
- Es recomendable ir documentando cada uno de los procesos que se van desarrollando a medida que se inicia la ejecución del trabajo de grado debido a que en el futuro puede servir como fundamentos teóricos del mismo.
- Se recomienda seguir desarrollando mecanismos cognitivos para control de congestión en redes ópticas definidas por software utilizando diferentes

tipos de técnicas que puedan mejorar el rendimiento de la red como aprendizaje de máquina, metaheurísticas, técnicas de clasificación, entre otras.

4.3. Trabajos futuros

- Analizar los parámetros de desempeño, probabilidad de bloqueo, retardo y Throughput con otros escenarios de prueba.
- Diseñar un mecanismo cognitivo para control de congestión basado en CHRON utilizando métodos basados en Machine Learning o el uso de una metaheurística.
- Diseñar un mecanismo cognitivo para control de congestión que realice el balanceo de carga en cualquier nodo de la red que esté congestionado.
- Utilizar otros modelos cognitivos para control de congestión con el fin de evaluar su desempeño.
- Utilizar otros controladores como *OpenDayLight, Ryu, Floodlight*, para el desarrollo de mecanismos cognitivos con el fin de evaluar su desempeño.
- Analizar el desempeño del mecanismo cognitivo basado en CHRON, generando una congestión critica en la red.

Bibliografía

[1] A. D. Valdez, C. A. Miranda, P. L. Schlesinger, J. A. Chiozza, C. V. Miranda, and A. A. Grela, "Calidad de servicio en redes de telecomunicaciones," *Extensionismo, Innovación y Transferencia Tecnológica*, vol. 4, pp. 278–293, 2018.

[2] Z. Cheng, X. Zhang, Y. Li, S. Yu, R. Lin, and L. He, "Congestion-aware local reroute for fast failure recovery in software-defined networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 11, pp. 934–944, Nov 2017.

[3] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," *Computer Communication Review*, vol. 43, no. 4, pp. 123–134, 2013.

[4] H. Jiang, Y. Luo, Q. Y. Zhang, M. Y. Yin, and C. Wu, "TCP-Gvegas with prediction and adaptation in multi-hop ad hoc networks," *Wireless Networks*, vol. 23, no. 5, pp. 1535–1548, 2017.

[5] Y. Gu and R. Grossman, "SABUL: A transport protocol for grid computing," *Journal of Grid Computing*, vol. 1, no. 4, pp. 377–386, 2003.

[6] S. Kim, J. Son, A. Talukder, and C. S. Hong, "Congestion prevention mechanism based on q-leaning for efficient routing in sdn," in *2016 International Conference on Information Networking (ICOIN)*, Jan 2016, pp. 124–128.

[7] Y. Wang and M. Valipour, "A Cognitive Machine Learning System for Phrases Composition and Semantic Comprehension," *2018 IEEE International*

Conference on Systems, Man, and Cybernetics (SMC), no. Icic, pp. 24–29, 2018.

[8] D. Pal, P. Verma, D. Gautam, and P. Indait, "Improved optimization technique using hybrid aco-pso," *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, pp. 277–282, Oct 2016.

[9] A. Srikanth, P. Varalakshmi, V. Somasundaram, and P. Ravichandiran, "Congestion control mechanism in software defined networking by traffic rerouting," in *2018 Second International Conference on Computing Methodologies and Communication (ICCMC)*, Feb 2018, pp. 55–58.

[10] T. Jain, D. S. Kushwaha, and S. Mallick, "Congestion control in software defined network," *2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering, UPCON 2018*, pp. 1–5, 2018.

[11] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, *Software Defined Optical Networks (SDONs): A Comprehensive Survey*, 2016, vol. 18, no. 4.

[12] P. Geurts, I. El Khayat, and G. Leduc, "A machine learning approach to improve congestion control over wireless computer networks," *Proceedings - Fourth IEEE International Conference on Data Mining, ICDM 2004*, pp. 383–386, 2004.

[13] A. F. Ruíz, Master's thesis, Universidad católica de Pereira, 2014.

[14] W. Li, Z. Yang, W. Zhao, Z. Qi, and F. Liu, "A Review of Research on Software-Defined Optical Network," *Proceedings - 2019 International Conference on Intelligent Transportation, Big Data and Smart City, ICITBS 2019*, pp. 155–160, 2019.

[15] "Software-Defined Networking (SDN) Definition," [Web; accedido el 04-11-2019]. [Online]. Available: https://www.opennetworking.org/sdn-definition/

[16] R. J. M. Tejedor, “Sdn: el futuro de las redes inteligentes,” *Conectrónica, teconlogía y elementos de conexión y conectividad*, vol. 2, no. 179, pp. 24–27, Sep 2014.

[17] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[18] O. N. F. Tr, “OpenFlow ONF SDN Evolution,” pp. 1–47, 2016.

[19] “SDN / OpenFlow | Flowgrammable,” [Web; accedido el 04-11-2019]. [Online]. Available: http://flowgrammable.org/sdn/openflow/

[20] R. K. Jha and B. N. M. Llah, “Software Defined Optical Networks (SDON): proposed architecture and comparative analysis,” *Journal of the European Optical Society*, vol. 15, no. 1, 2019.

[21] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, “B4: Experience with a globally-deployed software defined WAN,” *Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[22] W. W. Yongli Zhao, Yuqiao Wang and X. Yu, “Software-defined optical networking (sdon):principles and applications,” *Worldś largest Science , Technology & Medicine Open Access book publisher Portopulmonary Hypertension*, pp. 3–17, 2017.

[23] S. Singh and R. K. Jha, “Sdown: A novel algorithm and comparative performance analysis of underlying infrastructure in software defined heterogeneous network,” *Fiber and Integrated Optics*, vol. 38, no. 1, pp. 43–75, 2019.

[24] S. Singh and R. K. Jha, “A survey on software defined networking: Architecture for next generation network,” *Journal of Network and Systems Management*, vol. 25, no. 2, pp. 321–374, 2017.

[25] S. Azodolmolky, R. Nejabati, E. Escalona, R. Jayakumar, N. Efstathiou, and D. Simeonidou, “Integrated openflow - gmpls control plane: An overlay model

for software defined packet over optical networks," *Optics express*, vol. 19, pp. B421–8, 12 2011.

[26] R. K. Jha and B. N. M. Llah, "Software defined optical networks (sdon): proposed," *Journal of the European Optical Society-Rapid Publications*, vol. 15, no. 16, Jun 2019.

[27] Y. Li, J. Li, L. Zong, S. K. Bose, and G. Shen, "Upgrading nodes with colorless, directionless, and/or contentionless roadms in an optical transport network," in *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, 2020, pp. 1–4.

[28] S. Qureshi and R. Braun, "Mininet topology: Mirror of the optical switch fabric," in *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, 2019, pp. 1–6.

[29] "Introduction to dwdm technology," [Web; accedido el 15-11-2020]. [Online]. Available: https://www.cisco.com/c/dam/global/de_at/assets/docs/dwdm.pdf

[30] S. Kaneko, T. Yoshida, S. Kimura, and N. Yoshimoto, "Reliable λ-tuning olt-protection method based on backup-wavelength pre-assignment and discovery process for resilient wdm/tdm-pons," *Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1617–1622, 2015.

[31] J. A. Giron Salazar and L. H. Mora Saavedra, "Criterios para el diseño de una red de paquetes ip sobre una red óptica dwdm (ipodwdm)," *(Trabajo de grado) Universidad del Cauca, Popayán, Colombia*, 2003.

[32] A. Escallon Portilla and J. R. Barrios Lis, "Criterios para el monitoreo del canal óptico y la incidencia de los parámetros que afectan la calidad de la señal óptica en el desempeño de una red metropolitana wdm," *(Trabajo de grado) Universidad del Cauca, Popayán, Colombia*, 2008.

[33] A. Escallon Portilla and V. H. Ruiz Guacheta, "Evaluación del desempeño a nivel físico de un sistema ftth-gpon para servicios quad play al integrar un módulo rof," *(Trabajo de Maestría) Universidad del Cauca, Popayán, Colombia*, 2018.

[34] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Towards software defined cognitive networking," *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, July 2015.

[35] S. Prezenski, A. Brechmann, S. Wolff, and N. Russwinkel, "A cognitive modeling approach to strategy formation in dynamic decision making," *Frontiers in Psychology*, vol. 8, p. 1335, 2017.

[36] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, Jan 2018.

[37] C. Wessinger and E. Clapham, "Cognitive neuroscience: An overview," *Encyclopedia of Neuroscience*, pp. 1117–1122, December 2009.

[38] J. Mitola, "Cognitive radio an integrated agent architecture for software defined radio," *Royal Institute of technology, KTH, Kista*, 2000.

[39] J. Mitola and G. Q. Maguire, "Cognitive radio: making software radios more personal," *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, Aug 1999.

[40] J. Mitola, "Cognitive radio architecture evolution," *Proceedings of the IEEE*, vol. 97, no. 4, pp. 626–641, April 2009.

[41] R. W. Thomas, L. A. DaSilva, and A. B. MacKenzie, "Cognitive networks," *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, DySPAN 2005.*, pp. 352–360, Nov 2005.

[42] R. Borkowski, R. J. Duran, C. Kachris, D. Siracusa, A. Caballero, N. Fernandez, D. Klonidis, A. Francescon, T. Jimenez, J. C. Aguado, I. de Miguel, E. Salvadori, I. Tomkos, R. M. Lorenzo, and I. T. Monroy, "Cognitive optical network testbed: Eu project chron," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 2, pp. 344–355, February 2015.

[43] R. Berezdivin, R. Breinig, and R. Topp, "Next-generation wireless communications concepts and technologies," *IEEE Communications Magazine*, vol. 40, no. 3, pp. 108–116, March 2002.

[44] A. Alexiou and M. Haardt, "Smart antenna technologies for future wireless systems: trends and challenges," *IEEE Communications Magazine*, vol. 42, no. 9, pp. 90–97, Sep. 2004.

[45] E. Gelenbe, R. Lent, and Z. Xu, "Design and performance of cognitive packet networks," *Performance Evaluation*, vol. 46, no. 2, pp. 155 – 176, 2001.

[46] Q. Mahmoud, *Cognitive Networks: Towards Self-Aware Networks*. John Wiley & Sons, Ltd, June 2007.

[47] J. Strassner, "Context-aware, policy-based seamless mobility using the focale autonomic architecture," *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, pp. 568–573, May 2009.

[48] S. Ju and J. B. Evans, "Modeling and analysis of cognet architecture for cognitive radio networks," *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, pp. 953–958, Dec 2011.

[49] B. S. Manoj, R. R. Rao, and M. Zorzi, "Cognet: a cognitive complete knowledge network system," *IEEE Wireless Communications*, vol. 15, no. 6, pp. 81–88, December 2008.

[50] Y. Xin, M. Shayman, R. La, and S. Marcus, "Reconfiguration of survivable ip over wdm networks," *Optical Switching and Networking*, vol. 21, 11 2015.

[51] A. Caballero, R. Borkowski, I. de Miguel, R. J. Duran, J. C. Aguado, N. Fernandez, T. Jimenez, I. Rodriguez, D. Sanchez, R. M. Lorenzo, D. Klonidis, E. Palkopoulou, N. P. Diamantopoulos, I. Tomkos, D. Siracusa, A. Francescon, E. Salvadori, Y. Ye, J. Lopez Vizcaino, F. Pittala, A. Tymecki, and I. Tafur Monroy, "Cognitive, heterogeneous and reconfigurable optical networks: The chron project," *Journal of Lightwave Technology*, vol. 32, no. 13, pp. 2308–2323, July 2014.

[52] M. C. Nkosi, A. A. Lysko, and S. Dlamini, "Multi-path load balancing for SDN data plane," *2018 International Conference on Intelligent and Innovative Computing Applications, ICONIC 2018*, pp. 1–6, 2019.

[53] W. K. Soo, T.-C. Ling, A. H. Maw, and S. T. Win, "Survey on load-balancing methods in 802.11 infrastructure mode wireless networks for improving quality of service," *ACM Comput. Surv.*, vol. 51, no. 2, Feb. 2018. [Online]. Available: https://doi.org/10.1145/3172868

[54] S. Ejaz, Z. Iqbal, P. Azmat Shah, B. H. Bukhari, A. Ali, and F. Aadil, "Traffic load balancing using software defined networking (sdn) controller as virtualized network function," *IEEE Access*, vol. 7, pp. 46 646–46 658, 2019.

[55] J. S. Ganesh, M. Arisoylu, P. Anand, and N. M. Sawant, "Methods of forwarding data packets using transient tables and related load balancers," Apr. 11 2017, uS Patent 9,621,642.

[56] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *Communications Magazine, IEEE*, vol. 51, pp. 136–141, 02 2013.

[57] T. Hu, P. Yi, J. Zhang, and J. Lan, "Reliable and load balance-aware multi-controller deployment in sdn," *China Communications*, vol. 15, no. 11, pp. 184–198, 2018.

[58] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "Elasticon; an elastic distributed sdn controller," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2014, pp. 17–27.

[59] Y. Zhou, Y. Wang, J. Yu, J. Ba, and S. Zhang, "Load balancing for multiple controllers in sdn based on switches group," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2017, pp. 227–230.

[60] B. Valencia, S. Santacruz, L. Becerra, and J. Padilla, “Mininet: una herramienta versátil para emulación y prototipado de Redes Definidas por Software,” *Entre Ciencia e Ingeniería*, no. 17, pp. 62–70, 2015.

[61] A. Centeno, C. M. Rodriguez Vergel, C. Anías Calderón, F. Camilo, C. Bondarenko, and Uci, “Controladores sdn, elementos para su selección y evaluación,” *Telemática*, vol. 13, pp. 10–20, 11 2014.

[62] M. Bjørklund, *RFC 6020 (Proposed Standar)*, vol. 6020, 2010.

[63] I. GUIS, “The sdn gold rush to the northbound api,” 2012, [Web; accedido el 26-01-2021]. [Online]. Available: https://www.sdxcentral.com/articles/contributed/the-sdn-gold-rush-to-the-northbound-api/2012/11/

[64] D. Erickson, “Beacon,” [Web; accedido el 26-01-2021]. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home.html

[65] K. J. Srinivasan Ramasubramanian, Rob Adams, “Floodlight,” [Web; accedido el 26-01-2021]. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/HOME/overview?mode=global

[66] S. Rao, “Nox, the original openflow controller,” [Web; accedido el 26-01-2021]. [Online]. Available: https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/

[67] S. Kaur, J. Singh, and N. Ghumman, “Network programmability using pox controller,” 08 2014.

[68] R. S. F. Community, “component-based software defined networking framework build sdn agilely,” [Web; accedido el 26-01-2021]. [Online]. Available: https://ryu-sdn.org/

[69] “Open network operation system - onos),” [Web; accedido el 15-11-2020]. [Online]. Available: https://opennetworking.org/onos/

[70] L. OpenDaylight Project a Series of LF Projects, “Opendaylight,” [Web; accedido el 26-01-2021]. [Online]. Available: https://www.opendaylight.org/

[71] "Onos java api (2.4.0))," [Web; accedido el 15-11-2020]. [Online]. Available: http://api.onosproject.org/2.4.0/apidocs/

[72] J. L. Rivera Hurtado, "Método de control cognitivo aplicado al enrutamiento de una red bajo arquitectura sdn/nfv," *(Tesis Maestría) Universidad del Cauca, Popayán, Colombia*, 2018.

[73] "National science foundation's network - nsfnet," [Web; accedido el 15-11-2020]. [Online]. Available: https://www.nsf.gov/news/news_summ.jsp?cntn_id=103050#:~:text=NSFNET%20went%20online%20in%201986,to%201.5%20megabits%20per%20second.&text=Creation%20of%20NSFNET%20was%20an%20intellectual%20leap.

[74] P. Sayyad Khodashenas, J. Comellas, S. Spadaro, J. Perello, and G. Junyent, "Using spectrum fragmentation to better allocate time-varying connections in elastic optical networks," *Optical Communications and Networking, IEEE/OSA Journal of*, vol. 6, pp. 433–440, 05 2014.

[75] T. Slattery, "A story of mice and elephants: Dynamic packet prioritization)," [Web; accedido el 15-11-2020]. [Online]. Available: https://www.nojitter.com/story-mice-and-elephants-dynamic-packet-prioritization

[76] M. Hamdan, B. Abdalla, U. Humayun, A. Abdelaziz, S. Khan, M. Ali, M. Imran, and M. N. Marsono, "Flow-aware elephant flow detection for software-defined networks," *IEEE Access*, vol. PP, pp. 1–1, 04 2020.

[77] V. Gond and A. Goel, "Performance evaluation of wavelength routed optical network with wavelength conversion," *Journal of Telecommunications*, vol. 2, 2010.

[78] "Iperf," [Web; accedido el 15-11-2020]. [Online]. Available: https://www.mankier.com/1/iperf

[79] "Onos java api (2.4.0)," [Web; accedido el 16-11-2020]. [Online]. Available: http://api.onosproject.org/2.4.0/apidocs/

Mecanismo cognitivo para control de congestión en redes ópticas definidas por software

Universidad
del Cauca

ANEXOS

Trabajo de grado

Robinson Narváez Burbano
Mario Fernando Riascos Riascos

Director: PhD. José Giovanny López Perafán

Departamento de Telecomunicaciones
Facultad de Ingeniería Electrónica y Telecomunicaciones
Universidad del Cauca
Grupo de Investigación de Nuevas Tecnologías en Telecomunicaciones GNTT
Popayán, Cauca, 2021

Anexos

Anexo 1: Modelo óptico con topología de red NSF-NET completa

nsfnet.py

```
#!/usr/bin/python

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

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6633)
```

```
info( '*** Add switches\n')

#NSFNET SWITCH
s1 = net.addSwitch('s1', cls=OVSKernelSwitch, protocols='OpenFlow13')
s2 = net.addSwitch('s2', cls=OVSKernelSwitch, protocols='OpenFlow13')
s3 = net.addSwitch('s3', cls=OVSKernelSwitch, protocols='OpenFlow13')
s4 = net.addSwitch('s4', cls=OVSKernelSwitch, protocols='OpenFlow13')
s5 = net.addSwitch('s5', cls=OVSKernelSwitch, protocols='OpenFlow13')
s6 = net.addSwitch('s6', cls=OVSKernelSwitch, protocols='OpenFlow13')
s7 = net.addSwitch('s7', cls=OVSKernelSwitch, protocols='OpenFlow13')
s8 = net.addSwitch('s8', cls=OVSKernelSwitch, protocols='OpenFlow13')
s9 = net.addSwitch('s9', cls=OVSKernelSwitch, protocols='OpenFlow13')
s10 = net.addSwitch('s10', cls=OVSKernelSwitch, protocols='OpenFlow13')
s11 = net.addSwitch('s11', cls=OVSKernelSwitch, protocols='OpenFlow13')
s12 = net.addSwitch('s12', cls=OVSKernelSwitch, protocols='OpenFlow13')
s13 = net.addSwitch('s13', cls=OVSKernelSwitch, protocols='OpenFlow13')
s14 = net.addSwitch('s14', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 1A Lambda 1, SWITCH
s81 = net.addSwitch('s81', cls=OVSKernelSwitch, protocols='OpenFlow13')
s82 = net.addSwitch('s82', cls=OVSKernelSwitch, protocols='OpenFlow13')
s83 = net.addSwitch('s83', cls=OVSKernelSwitch, protocols='OpenFlow13')
s21 = net.addSwitch('s21', cls=OVSKernelSwitch, protocols='OpenFlow13')
s22 = net.addSwitch('s22', cls=OVSKernelSwitch, protocols='OpenFlow13')
s23 = net.addSwitch('s23', cls=OVSKernelSwitch, protocols='OpenFlow13')
s24 = net.addSwitch('s24', cls=OVSKernelSwitch, protocols='OpenFlow13')
s25 = net.addSwitch('s25', cls=OVSKernelSwitch, protocols='OpenFlow13')
s26 = net.addSwitch('s26', cls=OVSKernelSwitch, protocols='OpenFlow13')
s27 = net.addSwitch('s27', cls=OVSKernelSwitch, protocols='OpenFlow13')
s28 = net.addSwitch('s28', cls=OVSKernelSwitch, protocols='OpenFlow13')
s29 = net.addSwitch('s29', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 2A Lambda 2, SWITCH
s84 = net.addSwitch('s84', cls=OVSKernelSwitch, protocols='OpenFlow13')
s85 = net.addSwitch('s85', cls=OVSKernelSwitch, protocols='OpenFlow13')
s86 = net.addSwitch('s86', cls=OVSKernelSwitch, protocols='OpenFlow13')
s41 = net.addSwitch('s41', cls=OVSKernelSwitch, protocols='OpenFlow13')
s42 = net.addSwitch('s42', cls=OVSKernelSwitch, protocols='OpenFlow13')
s43 = net.addSwitch('s43', cls=OVSKernelSwitch, protocols='OpenFlow13')
s44 = net.addSwitch('s44', cls=OVSKernelSwitch, protocols='OpenFlow13')
s45 = net.addSwitch('s45', cls=OVSKernelSwitch, protocols='OpenFlow13')
s46 = net.addSwitch('s46', cls=OVSKernelSwitch, protocols='OpenFlow13')
s47 = net.addSwitch('s47', cls=OVSKernelSwitch, protocols='OpenFlow13')
s48 = net.addSwitch('s48', cls=OVSKernelSwitch, protocols='OpenFlow13')
s49 = net.addSwitch('s49', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 3A Lambda 3, SWITCH
s87 = net.addSwitch('s87', cls=OVSKernelSwitch, protocols='OpenFlow13')
s88 = net.addSwitch('s88', cls=OVSKernelSwitch, protocols='OpenFlow13')
s89 = net.addSwitch('s89', cls=OVSKernelSwitch, protocols='OpenFlow13')
```

```
s61 = net.addSwitch('s61', cls=OVSKernelSwitch, protocols='OpenFlow13')
s62 = net.addSwitch('s62', cls=OVSKernelSwitch, protocols='OpenFlow13')
s63 = net.addSwitch('s63', cls=OVSKernelSwitch, protocols='OpenFlow13')
s64 = net.addSwitch('s64', cls=OVSKernelSwitch, protocols='OpenFlow13')
s65 = net.addSwitch('s65', cls=OVSKernelSwitch, protocols='OpenFlow13')
s66 = net.addSwitch('s66', cls=OVSKernelSwitch, protocols='OpenFlow13')
s67 = net.addSwitch('s67', cls=OVSKernelSwitch, protocols='OpenFlow13')
s68 = net.addSwitch('s68', cls=OVSKernelSwitch, protocols='OpenFlow13')
s69 = net.addSwitch('s69', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 1B Lambda 1, SWITCH
s91 = net.addSwitch('s91', cls=OVSKernelSwitch, protocols='OpenFlow13')
s92 = net.addSwitch('s92', cls=OVSKernelSwitch, protocols='OpenFlow13')
s93 = net.addSwitch('s93', cls=OVSKernelSwitch, protocols='OpenFlow13')
s31 = net.addSwitch('s31', cls=OVSKernelSwitch, protocols='OpenFlow13')
s32 = net.addSwitch('s32', cls=OVSKernelSwitch, protocols='OpenFlow13')
s33 = net.addSwitch('s33', cls=OVSKernelSwitch, protocols='OpenFlow13')
s34 = net.addSwitch('s34', cls=OVSKernelSwitch, protocols='OpenFlow13')
s35 = net.addSwitch('s35', cls=OVSKernelSwitch, protocols='OpenFlow13')
s36 = net.addSwitch('s36', cls=OVSKernelSwitch, protocols='OpenFlow13')
s37 = net.addSwitch('s37', cls=OVSKernelSwitch, protocols='OpenFlow13')
s38 = net.addSwitch('s38', cls=OVSKernelSwitch, protocols='OpenFlow13')
s39 = net.addSwitch('s39', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 2B Lambda 1, SWITCH
s94 = net.addSwitch('s94', cls=OVSKernelSwitch, protocols='OpenFlow13')
s95 = net.addSwitch('s95', cls=OVSKernelSwitch, protocols='OpenFlow13')
s96 = net.addSwitch('s96', cls=OVSKernelSwitch, protocols='OpenFlow13')
s51 = net.addSwitch('s51', cls=OVSKernelSwitch, protocols='OpenFlow13')
s52 = net.addSwitch('s52', cls=OVSKernelSwitch, protocols='OpenFlow13')
s53 = net.addSwitch('s53', cls=OVSKernelSwitch, protocols='OpenFlow13')
s54 = net.addSwitch('s54', cls=OVSKernelSwitch, protocols='OpenFlow13')
s55 = net.addSwitch('s55', cls=OVSKernelSwitch, protocols='OpenFlow13')
s56 = net.addSwitch('s56', cls=OVSKernelSwitch, protocols='OpenFlow13')
s57 = net.addSwitch('s57', cls=OVSKernelSwitch, protocols='OpenFlow13')
s58 = net.addSwitch('s58', cls=OVSKernelSwitch, protocols='OpenFlow13')
s59 = net.addSwitch('s59', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 3B Lambda 1, SWITCH
s97 = net.addSwitch('s97', cls=OVSKernelSwitch, protocols='OpenFlow13')
s98 = net.addSwitch('s98', cls=OVSKernelSwitch, protocols='OpenFlow13')
s99 = net.addSwitch('s99', cls=OVSKernelSwitch, protocols='OpenFlow13')
s71 = net.addSwitch('s71', cls=OVSKernelSwitch, protocols='OpenFlow13')
s72 = net.addSwitch('s72', cls=OVSKernelSwitch, protocols='OpenFlow13')
s73 = net.addSwitch('s73', cls=OVSKernelSwitch, protocols='OpenFlow13')
s74 = net.addSwitch('s74', cls=OVSKernelSwitch, protocols='OpenFlow13')
s75 = net.addSwitch('s75', cls=OVSKernelSwitch, protocols='OpenFlow13')
s76 = net.addSwitch('s76', cls=OVSKernelSwitch, protocols='OpenFlow13')
s77 = net.addSwitch('s77', cls=OVSKernelSwitch, protocols='OpenFlow13')
s78 = net.addSwitch('s78', cls=OVSKernelSwitch, protocols='OpenFlow13')
```

```
s79 = net.addSwitch('s79', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Tx, SWITCH
s102 = net.addSwitch('s102', cls=OVSKernelSwitch, protocols='OpenFlow13')
s103 = net.addSwitch('s103', cls=OVSKernelSwitch, protocols='OpenFlow13')
s104 = net.addSwitch('s104', cls=OVSKernelSwitch, protocols='OpenFlow13')
s105 = net.addSwitch('s105', cls=OVSKernelSwitch, protocols='OpenFlow13')
s106 = net.addSwitch('s106', cls=OVSKernelSwitch, protocols='OpenFlow13')
s107 = net.addSwitch('s107', cls=OVSKernelSwitch, protocols='OpenFlow13')
s108 = net.addSwitch('s108', cls=OVSKernelSwitch, protocols='OpenFlow13')
s109 = net.addSwitch('s109', cls=OVSKernelSwitch, protocols='OpenFlow13')

info( '*** Add hosts\n')
#Fiber Host
hl1 = net.addHost('hl1', cls=Host, ip='10.0.0.1', defaultRoute=None,
mac='00:00:00:00:00:01')
hl2 = net.addHost('hl2', cls=Host, ip='10.0.0.2', defaultRoute=None,
mac='00:00:00:00:00:02')
hl3 = net.addHost('hl3', cls=Host, ip='10.0.0.3', defaultRoute=None,
mac='00:00:00:00:00:03')
hl4 = net.addHost('hl4', cls=Host, ip='10.0.0.4', defaultRoute=None,
mac='00:00:00:00:00:04')
hl5 = net.addHost('hl5', cls=Host, ip='10.0.0.5', defaultRoute=None,
mac='00:00:00:00:00:05')
hl6 = net.addHost('hl6', cls=Host, ip='10.0.0.6', defaultRoute=None,
mac='00:00:00:00:00:06')
hl7 = net.addHost('hl7', cls=Host, ip='10.0.0.7', defaultRoute=None,
mac='00:00:00:00:00:07')
hl8 = net.addHost('hl8', cls=Host, ip='10.0.0.8', defaultRoute=None,
mac='00:00:00:00:00:08')
hl9 = net.addHost('hl9', cls=Host, ip='10.0.0.9', defaultRoute=None,
mac='00:00:00:00:00:09')
hl10 = net.addHost('hl10', cls=Host, ip='10.0.0.10', defaultRoute=None,
mac='00:00:00:00:00:10')
hl11 = net.addHost('hl11', cls=Host, ip='10.0.0.11', defaultRoute=None,
mac='00:00:00:00:00:11')
hl12 = net.addHost('hl12', cls=Host, ip='10.0.0.12', defaultRoute=None,
mac='00:00:00:00:00:12')
hl13 = net.addHost('hl13', cls=Host, ip='10.0.0.13', defaultRoute=None,
mac='00:00:00:00:00:13')
hl14 = net.addHost('hl14', cls=Host, ip='10.0.0.14', defaultRoute=None,
mac='00:00:00:00:00:14')
hl15 = net.addHost('hl15', cls=Host, ip='10.0.0.15', defaultRoute=None,
mac='00:00:00:00:00:15')
hl16 = net.addHost('hl16', cls=Host, ip='10.0.0.16', defaultRoute=None,
mac='00:00:00:00:00:16')
hl17 = net.addHost('hl17', cls=Host, ip='10.0.0.17', defaultRoute=None,
mac='00:00:00:00:00:17')
hl18 = net.addHost('hl18', cls=Host, ip='10.0.0.18', defaultRoute=None,
mac='00:00:00:00:00:18')
```

```
#NSFNET Host
h1 = net.addHost('h1', cls=Host, ip='10.0.1.1', defaultRoute=None,
mac='00:00:00:00:01:01')
h2 = net.addHost('h2', cls=Host, ip='10.0.1.2', defaultRoute=None,
mac='00:00:00:00:01:02')
h3 = net.addHost('h3', cls=Host, ip='10.0.1.3', defaultRoute=None,
mac='00:00:00:00:01:03')
h4 = net.addHost('h4', cls=Host, ip='10.0.1.4', defaultRoute=None,
mac='00:00:00:00:01:04')
h5 = net.addHost('h5', cls=Host, ip='10.0.1.5', defaultRoute=None,
mac='00:00:00:00:01:05')
h6 = net.addHost('h6', cls=Host, ip='10.0.1.6', defaultRoute=None,
mac='00:00:00:00:01:06')
h7 = net.addHost('h7', cls=Host, ip='10.0.1.7', defaultRoute=None,
mac='00:00:00:00:01:07')
h8 = net.addHost('h8', cls=Host, ip='10.0.1.8', defaultRoute=None,
mac='00:00:00:00:01:08')
h9 = net.addHost('h9', cls=Host, ip='10.0.1.9', defaultRoute=None,
mac='00:00:00:00:01:09')
h10 = net.addHost('h10', cls=Host, ip='10.0.1.10', defaultRoute=None,
mac='00:00:00:00:01:10')
h11 = net.addHost('h11', cls=Host, ip='10.0.1.11', defaultRoute=None,
mac='00:00:00:00:01:11')
h12 = net.addHost('h12', cls=Host, ip='10.0.1.12', defaultRoute=None,
mac='00:00:00:00:01:12')
h13 = net.addHost('h13', cls=Host, ip='10.0.1.13', defaultRoute=None,
mac='00:00:00:00:01:13')
h14 = net.addHost('h14', cls=Host, ip='10.0.1.14', defaultRoute=None,
mac='00:00:00:00:01:14')
host14 = net.addHost('host14', cls=Host, ip='10.0.2.14',
defaultRoute=None, mac='00:00:00:00:02:14')

info( '*** Add links\n')
#Links Fiber 1A
net.addLink(hl1, s81, 0, 1)
net.addLink(hl2, s82, 0, 1)
net.addLink(hl3, s83, 0, 1)
net.addLink(s81, s21, 2, 1)
net.addLink(s82, s41, 2, 1)
net.addLink(s83, s61, 2, 1)
net.addLink(s21, s22, 2, 1)
net.addLink(s21, s25, 3, 1)
net.addLink(s21, s28, 4, 1)
net.addLink(s22, s23, 2, 1)
net.addLink(s22, s26, 3, 1)
net.addLink(s22, s29, 4, 1)
net.addLink(s23, s1, 2, 1)
net.addLink(s24, s22, 2, 6)
net.addLink(s24, s25, 3, 6)
```

```
net.addLink(s24, s28, 4, 6)
net.addLink(s25, s23, 2, 4)
net.addLink(s25, s26, 3, 4)
net.addLink(s25, s29, 4, 4)
net.addLink(s26, s104, 2, 1)
net.addLink(s27, s22, 2, 5)
net.addLink(s27, s25, 3, 5)
net.addLink(s27, s28, 4, 5)
net.addLink(s28, s23, 2, 3)
net.addLink(s28, s26, 3, 3)
net.addLink(s28, s29, 4, 3)
net.addLink(s29, s107, 2, 1)

#Links Fiber 2A
net.addLink(hl4, s84, 0, 1)
net.addLink(hl5, s85, 0, 1)
net.addLink(hl6, s86, 0, 1)
net.addLink(s84, s24, 2, 1)
net.addLink(s85, s44, 2, 1)
net.addLink(s86, s64, 2, 1)
net.addLink(s41, s42, 2, 1)
net.addLink(s41, s45, 3, 1)
net.addLink(s41, s48, 4, 1)
net.addLink(s42, s43, 2, 1)
net.addLink(s42, s46, 3, 1)
net.addLink(s42, s49, 4, 1)
net.addLink(s43, s102, 2, 1)
net.addLink(s44, s42, 2, 6)
net.addLink(s44, s45, 3, 6)
net.addLink(s44, s48, 4, 6)
net.addLink(s45, s43, 2, 4)
net.addLink(s45, s46, 3, 4)
net.addLink(s45, s49, 4, 4)
net.addLink(s46, s105, 2, 1)
net.addLink(s47, s42, 2, 5)
net.addLink(s47, s45, 3, 5)
net.addLink(s47, s48, 4, 5)
net.addLink(s48, s43, 2, 3)
net.addLink(s48, s46, 3, 3)
net.addLink(s48, s49, 4, 3)
net.addLink(s49, s108, 2, 1)

#Links Fiber 3A
net.addLink(hl7, s87, 0, 1)
net.addLink(hl8, s88, 0, 1)
net.addLink(hl9, s89, 0, 1)
net.addLink(s87, s27, 2, 1)
net.addLink(s88, s47, 2, 1)
net.addLink(s89, s67, 2, 1)
net.addLink(s61, s62, 2, 1)
```

```
net.addLink(s61, s65, 3, 1)
net.addLink(s61, s68, 4, 1)
net.addLink(s62, s63, 2, 1)
net.addLink(s62, s66, 3, 1)
net.addLink(s62, s69, 4, 1)
net.addLink(s63, s103, 2, 1)
net.addLink(s64, s62, 2, 6)
net.addLink(s64, s65, 3, 6)
net.addLink(s64, s68, 4, 6)
net.addLink(s65, s63, 2, 4)
net.addLink(s65, s66, 3, 4)
net.addLink(s65, s69, 4, 4)
net.addLink(s66, s106, 2, 1)
net.addLink(s67, s62, 2, 5)
net.addLink(s67, s65, 3, 5)
net.addLink(s67, s68, 4, 5)
net.addLink(s68, s63, 2, 3)
net.addLink(s68, s66, 3, 3)
net.addLink(s68, s69, 4, 3)
net.addLink(s69, s109, 2, 1)

#Links Fiber 1B
net.addLink(hl10, s91, 0, 1)
net.addLink(hl11, s92, 0, 1)
net.addLink(hl12, s93, 0, 1)
net.addLink(s10, s31, 1, 1)
net.addLink(s104, s34, 2, 1)
net.addLink(s107, s37, 2, 1)
net.addLink(s31, s32, 2, 1)
net.addLink(s31, s35, 3, 1)
net.addLink(s31, s38, 4, 1)
net.addLink(s32, s33, 2, 1)
net.addLink(s32, s36, 3, 1)
net.addLink(s32, s39, 4, 1)
net.addLink(s33, s91, 2, 2)
net.addLink(s34, s32, 2, 6)
net.addLink(s34, s35, 3, 6)
net.addLink(s34, s38, 4, 6)
net.addLink(s35, s33, 2, 4)
net.addLink(s35, s36, 3, 4)
net.addLink(s35, s39, 4, 4)
net.addLink(s36, s94, 2, 2)
net.addLink(s37, s32, 2, 5)
net.addLink(s37, s35, 3, 5)
net.addLink(s37, s38, 4, 5)
net.addLink(s38, s33, 2, 3)
net.addLink(s38, s36, 3, 3)
net.addLink(s38, s39, 4, 3)
net.addLink(s39, s97, 2, 2)
```

```
#Links Fiber 2B
net.addLink(hl13, s94, 0, 1)
net.addLink(hl14, s95, 0, 1)
net.addLink(hl15, s96, 0, 1)
net.addLink(s102, s51, 2, 1)
net.addLink(s105, s54, 2, 1)
net.addLink(s108, s57, 2, 1)
net.addLink(s51, s52, 2, 1)
net.addLink(s51, s55, 3, 1)
net.addLink(s51, s58, 4, 1)
net.addLink(s52, s53, 2, 1)
net.addLink(s52, s56, 3, 1)
net.addLink(s52, s59, 4, 1)
net.addLink(s53, s92, 2, 2)
net.addLink(s54, s52, 2, 6)
net.addLink(s54, s55, 3, 6)
net.addLink(s54, s58, 4, 6)
net.addLink(s55, s53, 2, 4)
net.addLink(s55, s56, 3, 4)
net.addLink(s55, s59, 4, 4)
net.addLink(s56, s95, 2, 2)
net.addLink(s57, s52, 2, 5)
net.addLink(s57, s55, 3, 5)
net.addLink(s57, s58, 4, 5)
net.addLink(s58, s53, 2, 3)
net.addLink(s58, s56, 3, 3)
net.addLink(s58, s59, 4, 3)
net.addLink(s59, s98, 2, 2)

#Links Fiber 3B
net.addLink(hl16, s97, 0, 1)
net.addLink(hl17, s98, 0, 1)
net.addLink(hl18, s99, 0, 1)
net.addLink(s103, s71, 2, 1)
net.addLink(s106, s74, 2, 1)
net.addLink(s109, s77, 2, 1)
net.addLink(s71, s72, 2, 1)
net.addLink(s71, s75, 3, 1)
net.addLink(s71, s78, 4, 1)
net.addLink(s72, s73, 2, 1)
net.addLink(s72, s76, 3, 1)
net.addLink(s72, s79, 4, 1)
net.addLink(s73, s93, 2, 2)
net.addLink(s74, s72, 2, 6)
net.addLink(s74, s75, 3, 6)
net.addLink(s74, s78, 4, 6)
net.addLink(s75, s73, 2, 4)
net.addLink(s75, s76, 3, 4)
net.addLink(s75, s79, 4, 4)
net.addLink(s76, s96, 2, 2)
```

```
net.addLink(s77, s72, 2, 5)
net.addLink(s77, s75, 3, 5)
net.addLink(s77, s78, 4, 5)
net.addLink(s78, s73, 2, 3)
net.addLink(s78, s76, 3, 3)
net.addLink(s78, s79, 4, 3)
net.addLink(s79, s99, 2, 2)

#Links NSFNET
net.addLink(h2, s2, 0, 1)
net.addLink(h1, s1, 0, 2)
net.addLink(h3, s3, 0, 1)
net.addLink(h4, s4, 0, 1)
net.addLink(h5, s5, 0, 1)
net.addLink(h6, s6, 0, 1)
net.addLink(h7, s7, 0, 1)
net.addLink(h8, s8, 0, 1)
net.addLink(h9, s9, 0, 1)
net.addLink(h10, s10, 0, 2)
net.addLink(h11, s11, 0, 1)
net.addLink(h12, s12, 0, 1)
net.addLink(h13, s13, 0, 1)
net.addLink(h14, s14, 0, 1)
net.addLink(host14, s14, 0, 2)
net.addLink(s1, s2, 3, 2)
net.addLink(s1, s3, 4, 2)
net.addLink(s1, s13, 5, 2)
net.addLink(s2, s4, 3, 2)
net.addLink(s2, s3, 4, 3)
net.addLink(s3, s6, 4, 2)
net.addLink(s4, s12, 3, 2)
net.addLink(s4, s5, 4, 2)
net.addLink(s5, s14, 3, 3)
net.addLink(s5, s6, 4, 3)
net.addLink(s6, s7, 4, 2)
net.addLink(s6, s8, 5, 2)
net.addLink(s7, s11, 3, 2)
net.addLink(s8, s9, 3, 2)
net.addLink(s8, s10, 4, 3)
net.addLink(s9, s12, 3, 3)
net.addLink(s9, s11, 4, 3)
net.addLink(s10, s12, 4, 4)
net.addLink(s10, s11, 5, 4)
net.addLink(s11, s13, 5, 3)
net.addLink(s13, s14, 4, 4)

info( '*** Starting network\n')
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
```

```
    controller.start()

info( '*** Starting switches\n')

#NSFNET
net.get('s7').start([c0])
net.get('s13').start([c0])
net.get('s2').start([c0])
net.get('s8').start([c0])
net.get('s14').start([c0])
net.get('s3').start([c0])
net.get('s1').start([c0])
net.get('s9').start([c0])
net.get('s4').start([c0])
net.get('s10').start([c0])
net.get('s5').start([c0])
net.get('s11').start([c0])
net.get('s6').start([c0])
net.get('s12').start([c0])

#Fiber 1A
net.get('s81').start([c0])
net.get('s82').start([c0])
net.get('s83').start([c0])
net.get('s21').start([c0])
net.get('s22').start([c0])
net.get('s23').start([c0])
net.get('s24').start([c0])
net.get('s25').start([c0])
net.get('s26').start([c0])
net.get('s27').start([c0])
net.get('s28').start([c0])
net.get('s29').start([c0])

#Fiber 2A
net.get('s84').start([c0])
net.get('s85').start([c0])
net.get('s86').start([c0])
net.get('s41').start([c0])
net.get('s42').start([c0])
net.get('s43').start([c0])
net.get('s44').start([c0])
net.get('s45').start([c0])
net.get('s46').start([c0])
net.get('s47').start([c0])
net.get('s48').start([c0])
net.get('s49').start([c0])


#Fiber 3A
```

```
net.get('s87').start([c0])
net.get('s88').start([c0])
net.get('s89').start([c0])
net.get('s61').start([c0])
net.get('s62').start([c0])
net.get('s63').start([c0])
net.get('s64').start([c0])
net.get('s65').start([c0])
net.get('s66').start([c0])
net.get('s67').start([c0])
net.get('s68').start([c0])
net.get('s69').start([c0])

#Fiber 1B
net.get('s91').start([c0])
net.get('s92').start([c0])
net.get('s93').start([c0])
net.get('s31').start([c0])
net.get('s32').start([c0])
net.get('s33').start([c0])
net.get('s34').start([c0])
net.get('s35').start([c0])
net.get('s36').start([c0])
net.get('s37').start([c0])
net.get('s38').start([c0])
net.get('s39').start([c0])

#Fiber 2B
net.get('s94').start([c0])
net.get('s95').start([c0])
net.get('s96').start([c0])
net.get('s51').start([c0])
net.get('s52').start([c0])
net.get('s53').start([c0])
net.get('s54').start([c0])
net.get('s55').start([c0])
net.get('s56').start([c0])
net.get('s57').start([c0])
net.get('s58').start([c0])
net.get('s59').start([c0])

#Fiber 3B
net.get('s97').start([c0])
net.get('s98').start([c0])
net.get('s99').start([c0])
net.get('s71').start([c0])
net.get('s72').start([c0])
net.get('s73').start([c0])
net.get('s74').start([c0])
net.get('s75').start([c0])
```

```
net.get('s76').start([c0])
net.get('s77').start([c0])
net.get('s78').start([c0])
net.get('s79').start([c0])

#Tx
net.get('s102').start([c0])
net.get('s103').start([c0])
net.get('s104').start([c0])
net.get('s105').start([c0])
net.get('s106').start([c0])
net.get('s107').start([c0])
net.get('s108').start([c0])
net.get('s109').start([c0])

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

CLI(net)
net.stop()
```

Anexo 2: Modelo óptico con topología de red NSF-NET reducida, para lambda 1

nsfnet_iperf.py

```
#!/usr/bin/python

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

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')

    #NSFNET SWITCH
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s9 = net.addSwitch('s9', cls=OVSKernelSwitch, protocols='OpenFlow13')
    s10 = net.addSwitch('s10', cls=OVSKernelSwitch, protocols='OpenFlow13')
```

```
s11 = net.addSwitch('s11', cls=OVSKernelSwitch, protocols='OpenFlow13')
s12 = net.addSwitch('s12', cls=OVSKernelSwitch, protocols='OpenFlow13')
s13 = net.addSwitch('s13', cls=OVSKernelSwitch, protocols='OpenFlow13')
s14 = net.addSwitch('s14', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 1A Lambda 1, SWITCH
s81 = net.addSwitch('s81', cls=OVSKernelSwitch, protocols='OpenFlow13')
s21 = net.addSwitch('s21', cls=OVSKernelSwitch, protocols='OpenFlow13')
s22 = net.addSwitch('s22', cls=OVSKernelSwitch, protocols='OpenFlow13')
s23 = net.addSwitch('s23', cls=OVSKernelSwitch, protocols='OpenFlow13')
s24 = net.addSwitch('s24', cls=OVSKernelSwitch, protocols='OpenFlow13')
s25 = net.addSwitch('s25', cls=OVSKernelSwitch, protocols='OpenFlow13')
s26 = net.addSwitch('s26', cls=OVSKernelSwitch, protocols='OpenFlow13')
s27 = net.addSwitch('s27', cls=OVSKernelSwitch, protocols='OpenFlow13')
s28 = net.addSwitch('s28', cls=OVSKernelSwitch, protocols='OpenFlow13')
s29 = net.addSwitch('s29', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 2A Lambda 2, SWITCH
s84 = net.addSwitch('s84', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 3A Lambda 3, SWITCH
s87 = net.addSwitch('s87', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 1B Lambda 1, SWITCH
s91 = net.addSwitch('s91', cls=OVSKernelSwitch, protocols='OpenFlow13')
s31 = net.addSwitch('s31', cls=OVSKernelSwitch, protocols='OpenFlow13')
s32 = net.addSwitch('s32', cls=OVSKernelSwitch, protocols='OpenFlow13')
s33 = net.addSwitch('s33', cls=OVSKernelSwitch, protocols='OpenFlow13')
s34 = net.addSwitch('s34', cls=OVSKernelSwitch, protocols='OpenFlow13')
s35 = net.addSwitch('s35', cls=OVSKernelSwitch, protocols='OpenFlow13')
s36 = net.addSwitch('s36', cls=OVSKernelSwitch, protocols='OpenFlow13')
s37 = net.addSwitch('s37', cls=OVSKernelSwitch, protocols='OpenFlow13')
s38 = net.addSwitch('s38', cls=OVSKernelSwitch, protocols='OpenFlow13')
s39 = net.addSwitch('s39', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 2B Lambda 1, SWITCH
s94 = net.addSwitch('s94', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Fiber 3B Lambda 1, SWITCH
s97 = net.addSwitch('s97', cls=OVSKernelSwitch, protocols='OpenFlow13')

#Tx, SWITCH
s104 = net.addSwitch('s104', cls=OVSKernelSwitch, protocols='OpenFlow13')
s107 = net.addSwitch('s107', cls=OVSKernelSwitch, protocols='OpenFlow13')

info( '*** Add hosts\n')
#Fiber Host
hl1 = net.addHost('hl1', cls=Host, ip='10.0.0.1', defaultRoute=None,
mac='00:00:00:00:00:01')
hl4 = net.addHost('hl4', cls=Host, ip='10.0.0.4', defaultRoute=None,
```

```
mac='00:00:00:00:00:04')
hl7 = net.addHost('hl7', cls=Host, ip='10.0.0.7', defaultRoute=None,
mac='00:00:00:00:00:07')
hl10 = net.addHost('hl10', cls=Host, ip='10.0.0.10', defaultRoute=None,
mac='00:00:00:00:00:10')
hl13 = net.addHost('hl13', cls=Host, ip='10.0.0.13', defaultRoute=None,
mac='00:00:00:00:00:13')
hl16 = net.addHost('hl16', cls=Host, ip='10.0.0.16', defaultRoute=None,
mac='00:00:00:00:00:16')

#NSFNET Host
h1 = net.addHost('h1', cls=Host, ip='10.0.1.1', defaultRoute=None,
mac='00:00:00:00:01:01')
h2 = net.addHost('h2', cls=Host, ip='10.0.1.2', defaultRoute=None,
mac='00:00:00:00:01:02')
h3 = net.addHost('h3', cls=Host, ip='10.0.1.3', defaultRoute=None,
mac='00:00:00:00:01:03')
h4 = net.addHost('h4', cls=Host, ip='10.0.1.4', defaultRoute=None,
mac='00:00:00:00:01:04')
h5 = net.addHost('h5', cls=Host, ip='10.0.1.5', defaultRoute=None,
mac='00:00:00:00:01:05')
h6 = net.addHost('h6', cls=Host, ip='10.0.1.6', defaultRoute=None,
mac='00:00:00:00:01:06')
h7 = net.addHost('h7', cls=Host, ip='10.0.1.7', defaultRoute=None,
mac='00:00:00:00:01:07')
h8 = net.addHost('h8', cls=Host, ip='10.0.1.8', defaultRoute=None,
mac='00:00:00:00:01:08')
h9 = net.addHost('h9', cls=Host, ip='10.0.1.9', defaultRoute=None,
mac='00:00:00:00:01:09')
h10 = net.addHost('h10', cls=Host, ip='10.0.1.10', defaultRoute=None,
mac='00:00:00:00:01:10')
h11 = net.addHost('h11', cls=Host, ip='10.0.1.11', defaultRoute=None,
mac='00:00:00:00:01:11')
h12 = net.addHost('h12', cls=Host, ip='10.0.1.12', defaultRoute=None,
mac='00:00:00:00:01:12')
h13 = net.addHost('h13', cls=Host, ip='10.0.1.13', defaultRoute=None,
mac='00:00:00:00:01:13')
h14 = net.addHost('h14', cls=Host, ip='10.0.1.14', defaultRoute=None,
mac='00:00:00:00:01:14')
host14 = net.addHost('host14', cls=Host, ip='10.0.2.14',
defaultRoute=None, mac='00:00:00:00:02:14')

info( '*** Add links\n')
#Links Fiber 1A
net.addLink(hl1, s81, 0, 1)
net.addLink(s81, s21, 2, 1)
net.addLink(s21, s22, 2, 1)
net.addLink(s21, s25, 3, 1)
net.addLink(s21, s28, 4, 1)
net.addLink(s22, s23, 2, 1)
```

```
net.addLink(s22, s26, 3, 1)
net.addLink(s22, s29, 4, 1)
net.addLink(s23, s1, 2, 1)
net.addLink(s24, s22, 2, 6)
net.addLink(s24, s25, 3, 6)
net.addLink(s24, s28, 4, 6)
net.addLink(s25, s23, 2, 4)
net.addLink(s25, s26, 3, 4)
net.addLink(s25, s29, 4, 4)
net.addLink(s26, s104, 2, 1)
net.addLink(s27, s22, 2, 5)
net.addLink(s27, s25, 3, 5)
net.addLink(s27, s28, 4, 5)
net.addLink(s28, s23, 2, 3)
net.addLink(s28, s26, 3, 3)
net.addLink(s28, s29, 4, 3)
net.addLink(s29, s107, 2, 1)

#Links Fiber 2A
net.addLink(hl4, s84, 0, 1)
net.addLink(s84, s24, 2, 1)

#Links Fiber 3A
net.addLink(hl7, s87, 0, 1)
net.addLink(s87, s27, 2, 1)

#Links Fiber 1B
net.addLink(hl10, s91, 0, 1)
net.addLink(s10, s31, 1, 1)
net.addLink(s104, s34, 2, 1)
net.addLink(s107, s37, 2, 1)
net.addLink(s31, s32, 2, 1)
net.addLink(s31, s35, 3, 1)
net.addLink(s31, s38, 4, 1)
net.addLink(s32, s33, 2, 1)
net.addLink(s32, s36, 3, 1)
net.addLink(s32, s39, 4, 1)
net.addLink(s33, s91, 2, 2)
net.addLink(s34, s32, 2, 6)
net.addLink(s34, s35, 3, 6)
net.addLink(s34, s38, 4, 6)
net.addLink(s35, s33, 2, 4)
net.addLink(s35, s36, 3, 4)
net.addLink(s35, s39, 4, 4)
net.addLink(s36, s94, 2, 2)
net.addLink(s37, s32, 2, 5)
net.addLink(s37, s35, 3, 5)
net.addLink(s37, s38, 4, 5)
net.addLink(s38, s33, 2, 3)
net.addLink(s38, s36, 3, 3)
```

```
net.addLink(s38, s39, 4, 3)
net.addLink(s39, s97, 2, 2)

#Links Fiber 2B
net.addLink(hl13, s94, 0, 1)

#Links Fiber 3B
net.addLink(hl16, s97, 0, 1)

#Links NSFNET
net.addLink(h2, s2, 0, 1)
net.addLink(h1, s1, 0, 2)
net.addLink(h3, s3, 0, 1)
net.addLink(h4, s4, 0, 1)
net.addLink(h5, s5, 0, 1)
net.addLink(h6, s6, 0, 1)
net.addLink(h7, s7, 0, 1)
net.addLink(h8, s8, 0, 1)
net.addLink(h9, s9, 0, 1)
net.addLink(h10, s10, 0, 2)
net.addLink(h11, s11, 0, 1)
net.addLink(h12, s12, 0, 1)
net.addLink(h13, s13, 0, 1)
net.addLink(h14, s14, 0, 1)
net.addLink(host14, s14, 0, 2)
net.addLink(s1, s2, 3, 2)
net.addLink(s1, s3, 4, 2)
net.addLink(s1, s13, 5, 2)
net.addLink(s2, s4, 3, 2)
net.addLink(s2, s3, 4, 3)
net.addLink(s3, s6, 4, 2)
net.addLink(s4, s12, 3, 2)
net.addLink(s4, s5, 4, 2)
net.addLink(s5, s14, 3, 3)
net.addLink(s5, s6, 4, 3)
net.addLink(s6, s7, 4, 2)
net.addLink(s6, s8, 5, 2)
net.addLink(s7, s11, 3, 2)
net.addLink(s8, s9, 3, 2)
net.addLink(s8, s10, 4, 3)
net.addLink(s9, s12, 3, 3)
net.addLink(s9, s11, 4, 3)
net.addLink(s10, s12, 4, 4)
net.addLink(s10, s11, 5, 4)
net.addLink(s11, s13, 5, 3)
net.addLink(s13, s14, 4, 4)

info( '*** Starting network\n')
net.build()
info( '*** Starting controllers\n')
```

```
for controller in net.controllers:
    controller.start()

info( '*** Starting switches\n')

#NSFNET
net.get('s7').start([c0])
net.get('s13').start([c0])
net.get('s2').start([c0])
net.get('s8').start([c0])
net.get('s14').start([c0])
net.get('s3').start([c0])
net.get('s1').start([c0])
net.get('s9').start([c0])
net.get('s4').start([c0])
net.get('s10').start([c0])
net.get('s5').start([c0])
net.get('s11').start([c0])
net.get('s6').start([c0])
net.get('s12').start([c0])

#Fiber 1A
net.get('s81').start([c0])
net.get('s21').start([c0])
net.get('s22').start([c0])
net.get('s23').start([c0])
net.get('s24').start([c0])
net.get('s25').start([c0])
net.get('s26').start([c0])
net.get('s27').start([c0])
net.get('s28').start([c0])
net.get('s29').start([c0])

#Fiber 2A
net.get('s84').start([c0])

#Fiber 3A
net.get('s87').start([c0])

#Fiber 1B
net.get('s91').start([c0])
net.get('s31').start([c0])
net.get('s32').start([c0])
net.get('s33').start([c0])
net.get('s34').start([c0])
net.get('s35').start([c0])
net.get('s36').start([c0])
net.get('s37').start([c0])
net.get('s38').start([c0])
net.get('s39').start([c0])
```

```
    #Fiber 2B
    net.get('s94').start([c0])

    #Fiber 3B
    net.get('s97').start([c0])

    #Tx
    net.get('s104').start([c0])
    net.get('s107').start([c0])

    info( '*** Post configure switches and hosts\n')
    sleep(30)
    info( '*** Start Test Iperf\n')
    myscript = "flows_nsfnet.sh"
    CLI(net, script=myscript)
    info( '*** Finish Test Iperf\n')

    CLI(net)
    net.stop()

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

Anexo 3: Scripts de trafico NSFNET con Iperf

Todos los scripts que se muestran a continuación deben de estar en la misma carpeta junto con los scripts del anexo 2 y 4.

flows_nsfnet.sh

```
link s26 s104 down
link s29 s107 down
s13 python traficos13tc.py &
hl1 python tfh1h10.py &
h1 python traficoh1.py &
h1 python traficoh1_13.py &
h2 python traficoh2.py &
h3 python traficoh3.py &
h4 python traficoh4.py &
h6 python traficoh6.py &
h8 python traficoh8.py &
h11 python traficoh11.py &
h12 python traficoh12.py &
h13 python traficoh13.py &
h13 python traficoh13_11.py &
h2 python traficoh2server.py &
h3 python traficoh3server.py &
h4 python traficoh4server.py &
h6 python traficoh6server.py &
h8 python traficoh8server.py &
h10 python traficoh10server.py &
h11 python traficoh11server.py &
h12 python traficoh12server.py &
h13 python traficoh13server.py &
hl10 python tfh1h10server.py
```

traficos13tc.py

```
import os
import sys
from time import time, sleep

sleep(120)

#Simulacion

sleep(9900)

sleep(1800)

print(os.system("tc qdisc add dev s13-eth3 root netem loss 8%"))

sleep(1500)

print(os.system("tc qdisc del dev s13-eth3 root"))

sleep(1500)

print(os.system("tc qdisc add dev s13-eth3 root netem loss 8%"))

sleep(2100)

print(os.system("tc qdisc del dev s13-eth3 root"))

sleep(900)

print(os.system("tc qdisc add dev s13-eth3 root netem loss 8%"))

sleep(900)

print(os.system("tc qdisc del dev s13-eth3 root"))

sleep(11100)

print(os.system("tc qdisc add dev s13-eth3 root netem loss 8%"))

sleep(900)

print(os.system("tc qdisc del dev s13-eth3 root"))

sleep(1800)

print(os.system("tc qdisc add dev s13-eth3 root netem loss 8%"))
```

```
sleep(600)

print(os.system("tc qdisc del dev s13-eth3 root"))
```

traficoh1.py

```
import os
import sys
from time import time, sleep

sleep(30)

#Simulacion

print(os.system("iperf -c 10.0.1.2 -u -b 25M -i 1800 -t 46920 -p 10102 &"))
print(os.system("iperf -c 10.0.1.3 -u -b 25M -i 1800 -t 46920 -p 10103"))
```

traficoh1_13.py

```
import os
import sys
from time import time, sleep

sleep(60)

#Simulacion

print(os.system("iperf -c 10.0.1.13 -u -b 25M -i 1800 -t 11760 -p 10113"))

print(os.system("iperf -c 10.0.1.13 -u -b 1500M -l 65k -i 1800 -t 1500 -p 20113"))

print(os.system("iperf -c 10.0.1.13 -u -b 25M -i 1800 -t 1500 -p 10113"))

print(os.system("iperf -c 10.0.1.13 -u -b 1500M -l 65k -i 1800 -t 2100 -p 20113"))

print(os.system("iperf -c 10.0.1.13 -u -b 25M -i 1800 -t 900 -p 10113"))

print(os.system("iperf -c 10.0.1.13 -u -b 1500M -l 65k -i 1800 -t 900 -p 20113"))

print(os.system("iperf -c 10.0.1.13 -u -b 25M -i 1800 -t 11100 -p 10113"))
```

```
print(os.system("iperf -c 10.0.1.13 -u -b 1500M -l 65k -i 1800 -t 900 -p 20113"))

print(os.system("iperf -c 10.0.1.13 -u -b 25M -i 1800 -t 1800 -p 10113"))

print(os.system("iperf -c 10.0.1.13 -u -b 1500M -l 65k -i 1800 -t 600 -p 20113"))

print(os.system("iperf -c 10.0.1.13 -u -b 25M -i 1800 -t 13830 -p 10113"))
```

traficoh2.py

```
import os
import sys
from time import time, sleep

#Simulacion

print(os.system("iperf -c 10.0.1.4 -u -b 25M -i 1800 -t 46950 -p 10204"))
```

traficoh3.py

```
import os
import sys
from time import time, sleep

#Simulacion

print(os.system("iperf -c 10.0.1.6 -u -b 25M -i 1800 -t 46950 -p 10306"))
```

traficoh4.py

```
import os
import sys
from time import time, sleep

#Simulacion

print(os.system("iperf -c 10.0.1.12 -u -b 25M -i 1800 -t 46950 -p 10412"))
```

traficoh6.py

```
import os
import sys
from time import time, sleep

#Simulacion

print(os.system("iperf -c 10.0.1.8 -u -b 25M -i 1800 -t 46950 -p 10608"))
```

traficoh8.py

```
import os
import sys
from time import time, sleep

sleep(30)

#Simulacion

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 46920 -p 10810"))
```

traficoh11.py

```
import os
import sys
from time import time, sleep

sleep(60)

#Simulacion

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 11760 -p 11110"))

print(os.system("iperf -c 10.0.1.10 -u -b 1500M -l 65k -i 1800 -t 1500 -p 21110"))

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 1500 -p 11110"))
```

```
print(os.system("iperf -c 10.0.1.10 -u -b 1500M -l 65k -i 1800 -t 2100 -p 21110"))

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 900 -p 11110"))

print(os.system("iperf -c 10.0.1.10 -u -b 1500M -l 65k -i 1800 -t 900 -p 21110"))

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 11100 -p 11110"))

print(os.system("iperf -c 10.0.1.10 -u -b 1500M -l 65k -i 1800 -t 900 -p 21110"))

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 1800 -p 11110"))

print(os.system("iperf -c 10.0.1.10 -u -b 1500M -l 65k -i 1800 -t 600 -p 21110"))

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 13830 -p 11110"))
```

traficoh12.py

```
import os
import sys
from time import time, sleep

sleep(30)

#Simulacion

print(os.system("iperf -c 10.0.1.10 -u -b 25M -i 1800 -t 46920 -p 11210"))
```

traficoh13.py

```
import os
import sys
from time import time, sleep

sleep(60)

#Simulacion

print(os.system("iperf -c 10.0.1.11 -u -b 50M -i 1800 -t 11760 -p 11311"))

sleep(1500)
```

```
print(os.system("iperf -c 10.0.1.11 -u -b 50M -i 1800 -t 1500 -p 11311"))

sleep(2100)

print(os.system("iperf -c 10.0.1.11 -u -b 50M -i 1800 -t 900 -p 11311"))

sleep(900)

print(os.system("iperf -c 10.0.1.11 -u -b 50M -i 1800 -t 11100 -p 11311"))

sleep(900)

print(os.system("iperf -c 10.0.1.11 -u -b 50M -i 1800 -t 1800 -p 11311"))

sleep(600)

print(os.system("iperf -c 10.0.1.11 -u -b 50M -i 1800 -t 13830 -p 11311"))
```

traficoh13_11.py

```
import os
import sys
from time import time, sleep

sleep(60)

#Simulacion

sleep(11760)

print(os.system("iperf -c 10.0.1.11 -u -b 1500M -i 1800 -t 1500 -p 21311"))

sleep(1500)

print(os.system("iperf -c 10.0.1.11 -u -b 1500M -i 1800 -t 2100 -p 21311"))

sleep(900)

print(os.system("iperf -c 10.0.1.11 -u -b 1500M -i 1800 -t 900 -p 21311"))

sleep(11100)

print(os.system("iperf -c 10.0.1.11 -u -b 1500M -i 1800 -t 900 -p 21311"))

sleep(1800)
```

```
print(os.system("iperf -c 10.0.1.11 -u -b 1500M -i 1800 -t 600 -p 21311"))
```

traficoh2server.py

```
import os
import sys

print(os.system("iperf -s -u -p 10102 -i 1800"))
```

traficoh3server.py

```
import os
import sys

print(os.system("iperf -s -u -p 10103 -i 1800"))
```

traficoh4server.py

```
import os
import sys

print(os.system("iperf -s -u -p 10204 -i 1800"))
```

traficoh6server.py

```
import os
import sys

print(os.system("iperf -s -u -p 10306 -i 1800"))
```

traficoh8server.py

```
import os
import sys

print(os.system("iperf -s -u -p 10608 -i 1800"))
```

traficoh10server.py

```
import os
import sys

print(os.system("iperf -s -u -p 11110 -i 1800 &"))
print(os.system("iperf -s -u -p 11210 -i 1800 &"))
print(os.system("iperf -s -u -p 10810 -i 1800 &"))

print(os.system("iperf -s -u -p 21110 -l 65k -i 1800"))
```

traficoh11server.py

```
import os
import sys

print(os.system("iperf -s -u -p 11311 -i 1800 &"))

print(os.system("iperf -s -u -p 21311 -i 1800"))
```

traficoh12server.py

```
import os
import sys

print(os.system("iperf -s -u -p 10412 -i 1800"))
```

traficoh13server.py

```
import os
import sys

print(os.system("iperf -s -u -p 10113 -i 1800 &"))

print(os.system("iperf -s -u -p 20113 -l 65k -i 1800"))
```

Anexo 4: Script de trafico hl1-hl10 para tomar los reportes con Iperf

Variación 1: 250Mbps

tfh1h10.py

```
import os
import sys
from time import time, sleep

sleep(120)

#Simulacion

print(os.system("iperf -c 10.0.0.10 -u -b 250M -i 1800 -t 46800 -p 10001"))
```

Variación 2: 500Mbps

tfh1h10.py

```
import os
import sys
from time import time, sleep

sleep(120)

#Simulacion

print(os.system("iperf -c 10.0.0.10 -u -b 500M -i 1800 -t 46800 -p 10001"))
```

tfh1h10server.py

```
import os
import sys

print(os.system("iperf -s -u -e -p 10001 -i 1800 > lh1h10"))
```

Anexo 5: Archivo pom.xml

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  ~ Copyright 2020 Open Networking Foundation
  ~
  ~ Licensed under the Apache License, Version 2.0 (the "License");
  ~ you may not use this file except in compliance with the License.
  ~ You may obtain a copy of the License at
  ~
  ~     http://www.apache.org/licenses/LICENSE-2.0
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the License is distributed on an "AS IS" BASIS,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  ~ See the License for the specific language governing permissions and
  ~ limitations under the License.
  -->
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.onosproject</groupId>
        <artifactId>onos-dependencies</artifactId>
        <version>2.2.1-b2</version>
    </parent>

    <groupId>org.mccc.app</groupId>
    <artifactId>mccc-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>bundle</packaging>

    <description>Our Mccc Application</description>
    <url>http://onosproject.org</url>

    <properties>
        <onos.app.name>org.mccc-fwd.app</onos.app.name>
        <onos.app.title>Mccc-fwd App</onos.app.title>
        <onos.app.category>Monitoring</onos.app.category>
        <onos.app.origin>Mccc-fwd, Inc.</onos.app.origin>
    </properties>
```

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

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

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

        <dependency>
            <groupId>org.onosproject</groupId>
            <artifactId>onos-api</artifactId>
            <version>${onos.version}</version>
            <scope>test</scope>
            <classifier>tests</classifier>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.onosproject</groupId>
                <artifactId>onos-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Anexo 6: Código (Mccc-fwd App)

ReactiveForwarding.java

```
package org.mccc.app;

import com.google.common.collect.ImmutableSet;
import org.onlab.packet.Ethernet;
import org.onlab.packet.ICMP;
import org.onlab.packet.ICMP6;
import org.onlab.packet.IPv4;
import org.onlab.packet.IPv6;
import org.onlab.packet.Ip4Prefix;
import org.onlab.packet.Ip6Prefix;
import org.onlab.packet.MacAddress;
import org.onlab.packet.TCP;
import org.onlab.packet.TpPort;
import org.onlab.packet.UDP;
import org.onlab.packet.VlanId;
import org.onlab.util.KryoNamespace;
import org.onlab.util.Tools;
import org.onosproject.cfg.ComponentConfigService;
import org.onosproject.core.ApplicationId;
import org.onosproject.core.CoreService;
import org.onosproject.event.Event;
import org.onosproject.net.ConnectPoint;
import org.onosproject.net.provider.ProviderId;
import org.onosproject.net.device.DeviceEvent;
import org.onosproject.net.utils.ForwardingDeviceService;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.DeviceId;
import org.onosproject.net.Host;
import org.onosproject.net.HostId;
import org.onosproject.net.Link;
import org.onosproject.net.Path;
import org.onosproject.net.PortNumber;
import org.onosproject.net.flow.DefaultTrafficSelector;
import org.onosproject.net.flow.DefaultTrafficTreatment;
import org.onosproject.net.flow.FlowEntry;
import org.onosproject.net.flow.FlowId;
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.DefaultFlowRule;
import org.onosproject.net.flow.FlowRuleService;
import org.onosproject.net.flow.FlowRuleListener;
import org.onosproject.net.flow.FlowRuleEvent;
import org.onosproject.net.flow.TrafficSelector;
import org.onosproject.net.flow.TrafficTreatment;
import org.onosproject.net.flow.criteria.Criterion;
import org.onosproject.net.flow.criteria.EthCriterion;
import org.onosproject.net.flow.criteria.PortCriterion;
import org.onosproject.net.flow.instructions.Instruction;
import org.onosproject.net.flow.instructions.Instructions;
import org.onosproject.net.flowobjective.DefaultForwardingObjective;
import org.onosproject.net.flowobjective.FlowObjectiveService;
import org.onosproject.net.flowobjective.ForwardingObjective;
import org.onosproject.net.host.HostService;
import org.onosproject.net.link.LinkEvent;
import org.onosproject.net.packet.InboundPacket;
import org.onosproject.net.packet.OutboundPacket;
import org.onosproject.net.packet.PacketContext;
import org.onosproject.net.packet.PacketPriority;
```

```
import org.onosproject.net.packet.PacketProcessor;
import org.onosproject.net.packet.PacketService;
import org.onosproject.net.topology.TopologyEvent;
import org.onosproject.net.topology.TopologyListener;
import org.onosproject.net.topology.TopologyService;
import org.onlab.util.KryoNamespace.Builder;
import org.onosproject.store.service.EventuallyConsistentMap;
import org.onosproject.store.service.MultiValuedTimestamp;
import org.onosproject.store.service.StorageService;
import org.onosproject.store.service.WallClockTimestamp;
import org.osgi.service.component.ComponentContext;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Modified;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.slf4j.Logger;

import java.util.Collection;
import java.util.Dictionary;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.concurrent.ExecutorService;

import java.nio.ByteBuffer;

import static java.util.concurrent.Executors.newSingleThreadExecutor;
import static org.onlab.util.Tools.groupedThreads;
import static org.mccc.app.OsgiPropertyConstants.FLOW_PRIORITY;
import static org.mccc.app.OsgiPropertyConstants.FLOW_PRIORITY_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.FLOW_TIMEOUT;
import static org.mccc.app.OsgiPropertyConstants.FLOW_TIMEOUT_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.IGNORE_IPV4_MCAST_PACKETS;
import static org.mccc.app.OsgiPropertyConstants.IGNORE_IPV4_MCAST_PACKETS_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.IPV6_FORWARDING;
import static org.mccc.app.OsgiPropertyConstants.IPV6_FORWARDING_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_DST_MAC_ONLY;
import static org.mccc.app.OsgiPropertyConstants.MATCH_DST_MAC_ONLY_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_ICMP_FIELDS;
import static org.mccc.app.OsgiPropertyConstants.MATCH_ICMP_FIELDS_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV4_ADDRESS;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV4_ADDRESS_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV4_DSCP;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV4_DSCP_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV6_ADDRESS;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV6_ADDRESS_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV6_FLOW_LABEL;
import static org.mccc.app.OsgiPropertyConstants.MATCH_IPV6_FLOW_LABEL_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_TCP_UDP_PORTS;
import static org.mccc.app.OsgiPropertyConstants.MATCH_TCP_UDP_PORTS_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.MATCH_VLAN_ID;
import static org.mccc.app.OsgiPropertyConstants.MATCH_VLAN_ID_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.PACKET_OUT_OFPP_TABLE;
import static org.mccc.app.OsgiPropertyConstants.PACKET_OUT_OFPP_TABLE_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.PACKET_OUT_ONLY;
import static org.mccc.app.OsgiPropertyConstants.PACKET_OUT_ONLY_DEFAULT;
import static org.mccc.app.OsgiPropertyConstants.RECORD_METRICS;
import static org.mccc.app.OsgiPropertyConstants.RECORD_METRICS_DEFAULT;
import static org.slf4j.LoggerFactory.getLogger;

import static org.onosproject.net.flow.FlowRuleEvent.Type.RULE_REMOVED;
import static org.onosproject.net.flow.FlowRuleEvent.Type.RULE_ADDED;
import static org.onosproject.net.flow.FlowRuleEvent.Type.RULE_UPDATED;
import static org.onosproject.net.flow.criteria.Criterion.Type.ETH_SRC;
```

```
import static org.onosproject.net.flow.criteria.Criterion.Type.ETH_DST;
import static org.onosproject.net.flow.criteria.Criterion.Type.IN_PORT;

import com.google.common.collect.Lists;
/**
 * Sample reactive forwarding application.
 */
@Component(
        immediate = true,
        service = ReactiveForwarding.class,
        property = {
                PACKET_OUT_ONLY + ":Boolean=" + PACKET_OUT_ONLY_DEFAULT,
                PACKET_OUT_OFPP_TABLE + ":Boolean=" + PACKET_OUT_OFPP_TABLE_DEFAULT,
                FLOW_TIMEOUT + ":Integer=" + FLOW_TIMEOUT_DEFAULT,
                FLOW_PRIORITY + ":Integer=" + FLOW_PRIORITY_DEFAULT,
                IPV6_FORWARDING + ":Boolean=" + IPV6_FORWARDING_DEFAULT,
                MATCH_DST_MAC_ONLY + ":Boolean=" + MATCH_DST_MAC_ONLY_DEFAULT,
                MATCH_VLAN_ID + ":Boolean=" + MATCH_VLAN_ID_DEFAULT,
                MATCH_IPV4_ADDRESS + ":Boolean=" + MATCH_IPV4_ADDRESS_DEFAULT,
                MATCH_IPV4_DSCP + ":Boolean=" + MATCH_IPV4_DSCP_DEFAULT,
                MATCH_IPV6_ADDRESS + ":Boolean=" + MATCH_IPV6_ADDRESS_DEFAULT,
                MATCH_IPV6_FLOW_LABEL + ":Boolean=" + MATCH_IPV6_FLOW_LABEL_DEFAULT,
                MATCH_TCP_UDP_PORTS + ":Boolean=" + MATCH_TCP_UDP_PORTS_DEFAULT,
                MATCH_ICMP_FIELDS + ":Boolean=" + MATCH_ICMP_FIELDS_DEFAULT,
                IGNORE_IPV4_MCAST_PACKETS + ":Boolean=" + IGNORE_IPV4_MCAST_PACKETS_DEFAULT,
                RECORD_METRICS + ":Boolean=" + RECORD_METRICS_DEFAULT
        }
)
public class ReactiveForwarding {

    protected final Logger log = getLogger(getClass());

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected TopologyService topologyService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected PacketService packetService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected HostService hostService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected FlowRuleService flowRuleService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected FlowObjectiveService flowObjectiveService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected CoreService coreService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected ComponentConfigService cfgService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected StorageService storageService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected DeviceService deviceService;

    private ReactivePacketProcessor processor = new ReactivePacketProcessor();

    private FlowRuleListener flowRuleListener = new InternalFlowListener();

    private  EventuallyConsistentMap<MacAddress, ReactiveForwardMetrics> metrics;

    private ApplicationId appId;

    /** Enable packet-out only forwarding; default is false. */
```

```
private boolean packetOutOnly = PACKET_OUT_ONLY_DEFAULT;

/** Enable first packet forwarding using OFPP_TABLE port instead of PacketOut with actual port; default is false. */
private boolean packetOutOfppTable = PACKET_OUT_OFPP_TABLE_DEFAULT;

/** Configure Flow Timeout for installed flow rules; default is 10 sec. */
private int flowTimeout = FLOW_TIMEOUT_DEFAULT;

/** Configure Flow Priority for installed flow rules; default is 10. */
private int flowPriority = FLOW_PRIORITY_DEFAULT;

/** Enable IPv6 forwarding; default is false. */
private boolean ipv6Forwarding = IPV6_FORWARDING_DEFAULT;

/** Enable matching Dst Mac Only; default is false. */
private boolean matchDstMacOnly = MATCH_DST_MAC_ONLY_DEFAULT;

/** Enable matching Vlan ID; default is false. */
private boolean matchVlanId = MATCH_VLAN_ID_DEFAULT;

/** Enable matching IPv4 Addresses; default is false. */
private boolean matchIpv4Address = MATCH_IPV4_ADDRESS_DEFAULT;

/** Enable matching IPv4 DSCP and ECN; default is false. */
private boolean matchIpv4Dscp = MATCH_IPV4_DSCP_DEFAULT;

/** Enable matching IPv6 Addresses; default is false. */
private boolean matchIpv6Address = MATCH_IPV6_ADDRESS_DEFAULT;

/** Enable matching IPv6 FlowLabel; default is false. */
private boolean matchIpv6FlowLabel = MATCH_IPV6_FLOW_LABEL_DEFAULT;

/** Enable matching TCP/UDP ports; default is false. */
private boolean matchTcpUdpPorts = MATCH_TCP_UDP_PORTS_DEFAULT;

/** Enable matching ICMPv4 and ICMPv6 fields; default is false. */
private boolean matchIcmpFields = MATCH_ICMP_FIELDS_DEFAULT;

/** Ignore (do not forward) IPv4 multicast packets; default is false. */
private boolean ignoreIPv4Multicast = IGNORE_IPV4_MCAST_PACKETS_DEFAULT;

/** Enable record metrics for reactive forwarding. */
private boolean recordMetrics = RECORD_METRICS_DEFAULT;

private final TopologyListener topologyListener = new InternalTopologyListener();

private ExecutorService blackHoleExecutor;

protected PortStatsListener portStatsListener = null;

/** Caso de estudio 2 con mccc */

/** Escenario 1 */
private boolean Case_2 = false;

/** Escenario 2, predeterminado */

/** Escenario 3 */
private boolean Case_4 = false;

/** One point represent 5 seconds */
private  int DefaultTimeStopMCCC = 120;
private  int TimeStopMCCC = DefaultTimeStopMCCC;

private boolean Congestion_flag_1 = false;
private boolean Congestion_flag_2 = false;
private boolean CongestionControlLink = false;
private boolean StopMCCC = false;
```

```
private boolean ClearFlowsRulesCongestion_flag = false;

private long pReceived = 0;

private long pSend = 0;

private PortNumber PortNumberStatics;
private DeviceId DeviceIdStatics;

private PortNumber CongestionportNumberClient;
private PortNumber CongestionportNumberServer;

@Activate
public void activate(ComponentContext context) {
    KryoNamespace.Builder metricSerializer = KryoNamespace.newBuilder()
            .register(ReactiveForwardMetrics.class)
            .register(MultiValuedTimestamp.class);
    metrics =  storageService.<MacAddress, ReactiveForwardMetrics>eventuallyConsistentMapBuilder()
            .withName("metrics-mccc")
            .withSerializer(metricSerializer)
            .withTimestampProvider((key, metricsData) -> new
                    MultiValuedTimestamp<>(new WallClockTimestamp(), System.nanoTime()))
            .build();

    blackHoleExecutor = newSingleThreadExecutor(groupedThreads("onos/app/fwd",
            "black-hole-fixer",
            log));

    cfgService.registerProperties(getClass());
    appId = coreService.registerApplication("org.onosproject.mccc");

    packetService.addProcessor(processor, PacketProcessor.director(2));
    //start
    portStatsListener = new PortStatsListener(this);
    deviceService.addListener(portStatsListener);
    flowRuleService.addListener(flowRuleListener);
    //end
    topologyService.addListener(topologyListener);
    readComponentConfiguration(context);
    requestIntercepts();

    log.info("Started", appId.id());
}

@Deactivate
public void deactivate() {
    cfgService.unregisterProperties(getClass(), false);
    withdrawIntercepts();
    flowRuleService.removeFlowRulesById(appId);
    packetService.removeProcessor(processor);
    flowRuleService.removeListener(flowRuleListener);
    topologyService.removeListener(topologyListener);
    blackHoleExecutor.shutdown();
    //inicio
    if (portStatsListener != null) {
        deviceService.removeListener(portStatsListener);
        portStatsListener = null;
    }
    Congestion_flag_1 = false;
    Congestion_flag_2 = false;
    CongestionControlLink = false;
    ClearFlowsRulesCongestion_flag = false;
    //fin
    blackHoleExecutor = null;
    processor = null;
    log.info("Stopped");
}
```

```
@Modified
public void modified(ComponentContext context) {
    readComponentConfiguration(context);
    requestIntercepts();
}

/**
 * Request packet in via packet service.
 */
private void requestIntercepts() {
    log.info("Hola-requestIntercepts");
    TrafficSelector.Builder selector = DefaultTrafficSelector.builder();
    selector.matchEthType(Ethernet.TYPE_IPV4);
    packetService.requestPackets(selector.build(), PacketPriority.REACTIVE, appId);

    selector.matchEthType(Ethernet.TYPE_IPV6);
    if (ipv6Forwarding) {
        packetService.requestPackets(selector.build(), PacketPriority.REACTIVE, appId);
    } else {
        packetService.cancelPackets(selector.build(), PacketPriority.REACTIVE, appId);
    }
}

/**
 * Cancel request for packet in via packet service.
 */
private void withdrawIntercepts() {
    TrafficSelector.Builder selector = DefaultTrafficSelector.builder();
    selector.matchEthType(Ethernet.TYPE_IPV4);
    packetService.cancelPackets(selector.build(), PacketPriority.REACTIVE, appId);
    selector.matchEthType(Ethernet.TYPE_IPV6);
    packetService.cancelPackets(selector.build(), PacketPriority.REACTIVE, appId);
}

/**
 * Extracts properties from the component configuration context.
 *
 * @param context the component context
 */
private void readComponentConfiguration(ComponentContext context) {
    Dictionary<?, ?> properties = context.getProperties();

    Boolean packetOutOnlyEnabled =
            Tools.isPropertyEnabled(properties, PACKET_OUT_ONLY);
    if (packetOutOnlyEnabled == null) {
        log.info("Packet-out is not configured, " +
                "using current value of {}", packetOutOnly);
    } else {
        packetOutOnly = packetOutOnlyEnabled;
        log.info("Configured. Packet-out only forwarding is {}",
                packetOutOnly ? "enabled" : "disabled");
    }

    Boolean packetOutOfppTableEnabled =
            Tools.isPropertyEnabled(properties, PACKET_OUT_OFPP_TABLE);
    if (packetOutOfppTableEnabled == null) {
        log.info("OFPP_TABLE port is not configured, " +
                "using current value of {}", packetOutOfppTable);
    } else {
        packetOutOfppTable = packetOutOfppTableEnabled;
        log.info("Configured. Forwarding using OFPP_TABLE port is {}",
                packetOutOfppTable ? "enabled" : "disabled");
    }

    Boolean ipv6ForwardingEnabled =
            Tools.isPropertyEnabled(properties, IPV6_FORWARDING);
    if (ipv6ForwardingEnabled == null) {
```

```
    log.info("IPv6 forwarding is not configured, " +
            "using current value of {}", ipv6Forwarding);
} else {
    ipv6Forwarding = ipv6ForwardingEnabled;
    log.info("Configured. IPv6 forwarding is {}",
            ipv6Forwarding ? "enabled" : "disabled");
}

Boolean matchDstMacOnlyEnabled =
        Tools.isPropertyEnabled(properties, MATCH_DST_MAC_ONLY);
if (matchDstMacOnlyEnabled == null) {
    log.info("Match Dst MAC is not configured, " +
            "using current value of {}", matchDstMacOnly);
} else {
    matchDstMacOnly = matchDstMacOnlyEnabled;
    log.info("Configured. Match Dst MAC Only is {}",
            matchDstMacOnly ? "enabled" : "disabled");
}

Boolean matchVlanIdEnabled =
        Tools.isPropertyEnabled(properties, MATCH_VLAN_ID);
if (matchVlanIdEnabled == null) {
    log.info("Matching Vlan ID is not configured, " +
            "using current value of {}", matchVlanId);
} else {
    matchVlanId = matchVlanIdEnabled;
    log.info("Configured. Matching Vlan ID is {}",
            matchVlanId ? "enabled" : "disabled");
}

Boolean matchIpv4AddressEnabled =
        Tools.isPropertyEnabled(properties, MATCH_IPV4_ADDRESS);
if (matchIpv4AddressEnabled == null) {
    log.info("Matching IPv4 Address is not configured, " +
            "using current value of {}", matchIpv4Address);
} else {
    matchIpv4Address = matchIpv4AddressEnabled;
    log.info("Configured. Matching IPv4 Addresses is {}",
            matchIpv4Address ? "enabled" : "disabled");
}

Boolean matchIpv4DscpEnabled =
        Tools.isPropertyEnabled(properties, MATCH_IPV4_DSCP);
if (matchIpv4DscpEnabled == null) {
    log.info("Matching IPv4 DSCP and ECN is not configured, " +
            "using current value of {}", matchIpv4Dscp);
} else {
    matchIpv4Dscp = matchIpv4DscpEnabled;
    log.info("Configured. Matching IPv4 DSCP and ECN is {}",
            matchIpv4Dscp ? "enabled" : "disabled");
}

Boolean matchIpv6AddressEnabled =
        Tools.isPropertyEnabled(properties, MATCH_IPV6_ADDRESS);
if (matchIpv6AddressEnabled == null) {
    log.info("Matching IPv6 Address is not configured, " +
            "using current value of {}", matchIpv6Address);
} else {
    matchIpv6Address = matchIpv6AddressEnabled;
    log.info("Configured. Matching IPv6 Addresses is {}",
            matchIpv6Address ? "enabled" : "disabled");
}

Boolean matchIpv6FlowLabelEnabled =
        Tools.isPropertyEnabled(properties, MATCH_IPV6_FLOW_LABEL);
if (matchIpv6FlowLabelEnabled == null) {
    log.info("Matching IPv6 FlowLabel is not configured, " +
            "using current value of {}", matchIpv6FlowLabel);
```

```
    } else {
        matchIpv6FlowLabel = matchIpv6FlowLabelEnabled;
        log.info("Configured. Matching IPv6 FlowLabel is {}",
                matchIpv6FlowLabel ? "enabled" : "disabled");
    }

    Boolean matchTcpUdpPortsEnabled =
            Tools.isPropertyEnabled(properties, MATCH_TCP_UDP_PORTS);
    if (matchTcpUdpPortsEnabled == null) {
        log.info("Matching TCP/UDP fields is not configured, " +
                "using current value of {}", matchTcpUdpPorts);
    } else {
        matchTcpUdpPorts = matchTcpUdpPortsEnabled;
        log.info("Configured. Matching TCP/UDP fields is {}",
                matchTcpUdpPorts ? "enabled" : "disabled");
    }

    Boolean matchIcmpFieldsEnabled =
            Tools.isPropertyEnabled(properties, MATCH_ICMP_FIELDS);
    if (matchIcmpFieldsEnabled == null) {
        log.info("Matching ICMP (v4 and v6) fields is not configured, " +
                "using current value of {}", matchIcmpFields);
    } else {
        matchIcmpFields = matchIcmpFieldsEnabled;
        log.info("Configured. Matching ICMP (v4 and v6) fields is {}",
                matchIcmpFields ? "enabled" : "disabled");
    }

    Boolean ignoreIpv4McastPacketsEnabled =
            Tools.isPropertyEnabled(properties, IGNORE_IPV4_MCAST_PACKETS);
    if (ignoreIpv4McastPacketsEnabled == null) {
        log.info("Ignore IPv4 multi-cast packet is not configured, " +
                "using current value of {}", ignoreIPv4Multicast);
    } else {
        ignoreIPv4Multicast = ignoreIpv4McastPacketsEnabled;
        log.info("Configured. Ignore IPv4 multicast packets is {}",
                ignoreIPv4Multicast ? "enabled" : "disabled");
    }
    Boolean recordMetricsEnabled =
            Tools.isPropertyEnabled(properties, RECORD_METRICS);
    if (recordMetricsEnabled == null) {
        log.info("IConfigured. Ignore record metrics  is {} ," +
                "using current value of {}", recordMetrics);
    } else {
        recordMetrics = recordMetricsEnabled;
        log.info("Configured. record metrics  is {}",
                recordMetrics ? "enabled" : "disabled");
    }

    flowTimeout = Tools.getIntegerProperty(properties, FLOW_TIMEOUT, FLOW_TIMEOUT_DEFAULT);
    log.info("Configured. Flow Timeout is configured to {} seconds", flowTimeout);

    flowPriority = Tools.getIntegerProperty(properties, FLOW_PRIORITY, FLOW_PRIORITY_DEFAULT);
    log.info("Configured. Flow Priority is configured to {}", flowPriority);
}

/**
 * Packet processor responsible for forwarding packets along their paths.
 */

private class ReactivePacketProcessor implements PacketProcessor {

    @Override
    public void process(PacketContext context) {
        // Stop processing if the packet has been handled, since we
        // can't do any more to it.

        if (context.isHandled()) {
```

```
    return;
}

InboundPacket pkt = context.inPacket();
Ethernet ethPkt = pkt.parsed();

if (ethPkt == null) {
    return;
}

MacAddress macAddress = ethPkt.getSourceMAC();
ReactiveForwardMetrics macMetrics = null;
macMetrics = createCounter(macAddress);
inPacket(macMetrics);

// Bail if this is deemed to be a control packet.
if (isControlPacket(ethPkt)) {
    droppedPacket(macMetrics);
    return;
}

// Skip IPv6 multicast packet when IPv6 forward is disabled.
if (!ipv6Forwarding && isIpv6Multicast(ethPkt)) {
    droppedPacket(macMetrics);
    return;
}

HostId id = HostId.hostId(ethPkt.getDestinationMAC(), VlanId.vlanId(ethPkt.getVlanID()));
// Do not process LLDP MAC address in any way.
if (id.mac().isLldp()) {
    droppedPacket(macMetrics);
    return;
}

// Do not process IPv4 multicast packets, let mfwd handle them
if (ignoreIPv4Multicast && ethPkt.getEtherType() == Ethernet.TYPE_IPV4) {
    if (id.mac().isMulticast()) {
        return;
    }
}

// Do we know who this is for? If not, flood and bail.
Host dst = hostService.getHost(id);
if (dst == null) {
    flood(context, macMetrics);
    return;
}

// Are we on an edge switch that our destination is on? If so,
// simply forward out to the destination and bail.
if (pkt.receivedFrom().deviceId().equals(dst.location().deviceId())) {
    if (!context.inPacket().receivedFrom().port().equals(dst.location().port())) {
        installRule(context, dst.location().port(), macMetrics);
    }
    return;
}
// Otherwise, get a set of paths that lead from here to the
// destination edge switch.
Set<Path> paths =
        topologyService.getPaths(topologyService.currentTopology(),
                pkt.receivedFrom().deviceId(),
                dst.location().deviceId());
if (paths.isEmpty()) {
    // If there are no paths, flood and bail.
    flood(context, macMetrics);
    return;
}
```

```
        // Otherwise, pick a path that does not lead back to where we
        // came from; if no such path, flood and bail.
        Path path = pickForwardPathIfPossible(paths, pkt.receivedFrom().port());
        if (path == null) {
            log.warn("Don't know where to go from here {} for {} -> {}",
                    pkt.receivedFrom(), ethPkt.getSourceMAC(), ethPkt.getDestinationMAC());
            flood(context, macMetrics);
            return;
        }

        // Otherwise forward and be done with it.
        //log.info("Hola-Location, DeviceID {}", pkt.receivedFrom().deviceId());

        installRule(context, path.src().port(), macMetrics);
    }

}

private class InternalFlowListener implements FlowRuleListener {

    @Override
    public void event(FlowRuleEvent event) {
        FlowRule rule = event.subject();
        if (event.type() == RULE_REMOVED && rule.appId() == appId.id()) {
            Criterion criterion_src = rule.selector().getCriterion(ETH_SRC);
            MacAddress src = ((EthCriterion) criterion_src).mac();
            Criterion criterion_dst = rule.selector().getCriterion(ETH_DST);
            MacAddress dst = ((EthCriterion) criterion_dst).mac();
            Criterion criterion_InPort = rule.selector().getCriterion(IN_PORT);
            PortNumber InPort = ((PortCriterion) criterion_InPort).port();
            DeviceId deviceId = rule.deviceId();
            int priority = rule.priority();
            boolean matchesSrc = false, matchesDst = false;
            if (Congestion_flag_1) {
                if (deviceId.equals(DeviceId.deviceId("of:0000000000000001")) && (priority == 10)) {

                    if (src.equals(MacAddress.valueOf("00:00:00:00:00:01")) &&
                            dst.equals(MacAddress.valueOf("00:00:00:00:00:10"))) {
                        TemporaryFlowsOne(deviceId, CongestionportNumberClient, rule, 15);
                    }
                }

                if (deviceId.equals(DeviceId.deviceId("of:000000000000000a")) && (priority == 10)){

                    if (src.equals(MacAddress.valueOf("00:00:00:00:00:01")) &&
                            dst.equals(MacAddress.valueOf("00:00:00:00:00:10"))) {
                        TemporaryFlowsTwo(deviceId, CongestionportNumberServer, src, dst, rule, 15);
                    }
                }
            } else {

                if(ClearFlowsRulesCongestion_flag){

                    if (deviceId.equals(DeviceId.deviceId("of:0000000000000001")) && (priority == 15)) {

                        if (src.equals(MacAddress.valueOf("00:00:00:00:00:01")) &&
                                dst.equals(MacAddress.valueOf("00:00:00:00:00:10"))) {
                            TemporaryFlowsOne(deviceId, PortNumber.portNumber("5"), rule, 10);
                        }
                    }

                    if (deviceId.equals(DeviceId.deviceId("of:000000000000000a")) && (priority == 15)){

                        if (src.equals(MacAddress.valueOf("00:00:00:00:00:01")) &&
                                dst.equals(MacAddress.valueOf("00:00:00:00:00:10"))) {
                            TemporaryFlowsTwo(deviceId, PortNumber.portNumber("5"), src, dst, rule, 10);
                        }
                    }
```

```
                }
            }
        }
    }
}

private void TemporaryFlowsOne(DeviceId deviceId, PortNumber Out_Port, FlowRule rule, int priority) {

    TrafficTreatment treatment = DefaultTrafficTreatment.builder()
            .setOutput(Out_Port)
            .build();

    FlowRule CongestionFlow = DefaultFlowRule.builder()
            .withSelector(rule.selector())
            .withTreatment(treatment)
            .forDevice(deviceId)
            .fromApp(appId)
            .makeTemporary(flowTimeout)
            //.makePermanent()
            .withPriority(priority)
            .build();

    flowRuleService.applyFlowRules(CongestionFlow);
}

private void TemporaryFlowsTwo(DeviceId deviceId, PortNumber In_Port, MacAddress src,
            MacAddress dst, FlowRule rule, int priority) {

    TrafficSelector.Builder selectorBuilder = DefaultTrafficSelector.builder();

    selectorBuilder.matchInPort(In_Port)
            .matchEthSrc(src)
            .matchEthDst(dst);

    FlowRule CongestionFlow = DefaultFlowRule.builder()
            .withSelector(selectorBuilder.build())
            .withTreatment(rule.treatment())
            .forDevice(deviceId)
            .fromApp(appId)
            .makeTemporary(flowTimeout)
            //.makePermanent()
            .withPriority(priority)
            .build();

    flowRuleService.applyFlowRules(CongestionFlow);
}

// Indicates whether this is a control packet, e.g. LLDP, BDDP
private boolean isControlPacket(Ethernet eth) {
    short type = eth.getEtherType();
    return type == Ethernet.TYPE_LLDP || type == Ethernet.TYPE_BSN;
}

// Indicated whether this is an IPv6 multicast packet.
private boolean isIpv6Multicast(Ethernet eth) {
    return eth.getEtherType() == Ethernet.TYPE_IPV6 && eth.isMulticast();
}

// Selects a path from the given set that does not lead back to the
// specified port if possible.
private Path pickForwardPathIfPossible(Set<Path> paths, PortNumber notToPort) {
    for (Path path : paths) {
        if (!path.src().port().equals(notToPort)) {
            return path;
        }
    }
    return null;
}
```

```
// Floods the specified packet if permissible.
private void flood(PacketContext context, ReactiveForwardMetrics macMetrics) {
    log.info("Hola-Flood");
    if (topologyService.isBroadcastPoint(topologyService.currentTopology(),
            context.inPacket().receivedFrom())) {
        packetOut(context, PortNumber.FLOOD, macMetrics);
    } else {
        context.block();
    }
}

// Sends a packet out the specified port.
private void packetOut(PacketContext context, PortNumber portNumber, ReactiveForwardMetrics macMetrics) {
    replyPacket(macMetrics);
    context.treatmentBuilder().setOutput(portNumber);
    context.send();
}

// Install a rule forwarding the packet to the specified port.
private void installRule(PacketContext context, PortNumber portNumber, ReactiveForwardMetrics macMetrics) {
    //
    // We don't support (yet) buffer IDs in the Flow Service so
    // packet out first.
    //

    Ethernet inPkt = context.inPacket().parsed();
    InboundPacket pkt = context.inPacket();
    DeviceId deviceId = context.inPacket().receivedFrom().deviceId();
    PortNumber inport = context.inPacket().receivedFrom().port();
    // If PacketOutOnly or ARP packet than forward directly to output port

    if (packetOutOnly || inPkt.getEtherType() == Ethernet.TYPE_ARP) {
        packetOut(context, portNumber, macMetrics);
        return;
    }

    IPv4 ipv4Packett = (IPv4) inPkt.getPayload();

    if (inPkt.getEtherType() == Ethernet.TYPE_LLDP) {
    }

    if (ipv4Packett.getProtocol() == IPv4.PROTOCOL_UDP || ipv4Packett.getProtocol() == IPv4.PROTOCOL_ICMP){
    }

    if (pkt.receivedFrom().deviceId().equals(DeviceId.deviceId("of:0000000000000001"))) {
        if (inPkt.getDestinationMAC().equals(MacAddress.valueOf("00:00:00:00:00:10"))
                && Congestion_flag_1 == true) {
            CongestionFlows(CongestionportNumberClient, context, macMetrics);
            Congestion_flag_2 = true;
            packetOut(context, CongestionportNumberClient, macMetrics);
            return;
        }
    }

    if (pkt.receivedFrom().deviceId().equals(DeviceId.deviceId("of:000000000000000a"))) {
        if (inPkt.getDestinationMAC().equals(MacAddress.valueOf("00:00:00:00:00:10"))
                && (Congestion_flag_1 == true)) {
            CongestionFlows(portNumber, context, macMetrics);
            packetOut(context, portNumber, macMetrics);
            return;
        }
    }

    if (pkt.receivedFrom().deviceId().equals(DeviceId.deviceId("of:000000000000000a"))) {
        if (inPkt.getDestinationMAC().equals(MacAddress.valueOf("00:00:00:00:00:01"))
                && (Congestion_flag_1 == true || Congestion_flag_2 == true)) {
            CongestionFlows(CongestionportNumberServer, context, macMetrics);
```

```
            packetOut(context, portNumber, macMetrics);
            return;
        }
    }

    if (pkt.receivedFrom().deviceId().equals(DeviceId.deviceId("of:0000000000000001"))) {
        if (inPkt.getDestinationMAC().equals(MacAddress.valueOf("00:00:00:00:00:01"))
                && (Congestion_flag_1 == true || Congestion_flag_2 == true)) {
            CongestionFlows(portNumber, context, macMetrics);
            packetOut(context, portNumber, macMetrics);
            Congestion_flag_2 = false;
        }
    }

    TrafficSelector.Builder selectorBuilder = DefaultTrafficSelector.builder();
    //
    // If matchDstMacOnly
    //    Create flows matching dstMac only
    // Else
    //    Create flows with default matching and include configured fields
    //
    if (matchDstMacOnly) {
        selectorBuilder.matchEthDst(inPkt.getDestinationMAC());
        log.info("Hola-matchDstMacOnly");
    } else {
        selectorBuilder.matchInPort(context.inPacket().receivedFrom().port())
                .matchEthSrc(inPkt.getSourceMAC())
                .matchEthDst(inPkt.getDestinationMAC());

        // If configured Match Vlan ID
        if (matchVlanId && inPkt.getVlanID() != Ethernet.VLAN_UNTAGGED) {
            selectorBuilder.matchVlanId(VlanId.vlanId(inPkt.getVlanID()));
        }

        //
        // If configured and EtherType is IPv4 - Match IPv4 and
        // TCP/UDP/ICMP fields
        //
        if (matchIpv4Address && inPkt.getEtherType() == Ethernet.TYPE_IPV4) {
            log.info("Hola-IPV4");
            IPv4 ipv4Packet = (IPv4) inPkt.getPayload();
            byte ipv4Protocol = ipv4Packet.getProtocol();
            Ip4Prefix matchIp4SrcPrefix =
                    Ip4Prefix.valueOf(ipv4Packet.getSourceAddress(),
                            Ip4Prefix.MAX_MASK_LENGTH);
            Ip4Prefix matchIp4DstPrefix =
                    Ip4Prefix.valueOf(ipv4Packet.getDestinationAddress(),
                            Ip4Prefix.MAX_MASK_LENGTH);
            selectorBuilder.matchEthType(Ethernet.TYPE_IPV4)
                    .matchIPSrc(matchIp4SrcPrefix)
                    .matchIPDst(matchIp4DstPrefix);
            if (matchIpv4Dscp) {
                log.info("Hola-IPV4Dscp");
                byte dscp = ipv4Packet.getDscp();
                byte ecn = ipv4Packet.getEcn();
                selectorBuilder.matchIPDscp(dscp).matchIPEcn(ecn);
            }
            if (matchTcpUdpPorts && ipv4Protocol == IPv4.PROTOCOL_TCP) {
                log.info("Hola-TCP");
                TCP tcpPacket = (TCP) ipv4Packet.getPayload();
                selectorBuilder.matchIPProtocol(ipv4Protocol)
                        .matchTcpSrc(TpPort.tpPort(tcpPacket.getSourcePort()))
                        .matchTcpDst(TpPort.tpPort(tcpPacket.getDestinationPort()));
            }
            if (matchTcpUdpPorts && ipv4Protocol == IPv4.PROTOCOL_UDP) {
                log.info("Hola-UDP");
                UDP udpPacket = (UDP) ipv4Packet.getPayload();
                selectorBuilder.matchIPProtocol(ipv4Protocol)
```

```
                    .matchUdpSrc(TpPort.tpPort(udpPacket.getSourcePort()))
                    .matchUdpDst(TpPort.tpPort(udpPacket.getDestinationPort()));
            }
            if (matchIcmpFields && ipv4Protocol == IPv4.PROTOCOL_ICMP) {
                log.info("Hola-ICMP");
                ICMP icmpPacket = (ICMP) ipv4Packet.getPayload();
                selectorBuilder.matchIPProtocol(ipv4Protocol)
                        .matchIcmpType(icmpPacket.getIcmpType())
                        .matchIcmpCode(icmpPacket.getIcmpCode());
            }
        }

        //
        // If configured and EtherType is IPv6 - Match IPv6 and
        // TCP/UDP/ICMP fields
        //
        if (matchIpv6Address && inPkt.getEtherType() == Ethernet.TYPE_IPV6) {
            IPv6 ipv6Packet = (IPv6) inPkt.getPayload();
            byte ipv6NextHeader = ipv6Packet.getNextHeader();
            Ip6Prefix matchIp6SrcPrefix =
                    Ip6Prefix.valueOf(ipv6Packet.getSourceAddress(),
                            Ip6Prefix.MAX_MASK_LENGTH);
            Ip6Prefix matchIp6DstPrefix =
                    Ip6Prefix.valueOf(ipv6Packet.getDestinationAddress(),
                            Ip6Prefix.MAX_MASK_LENGTH);
            selectorBuilder.matchEthType(Ethernet.TYPE_IPV6)
                    .matchIPv6Src(matchIp6SrcPrefix)
                    .matchIPv6Dst(matchIp6DstPrefix);

            if (matchIpv6FlowLabel) {
                selectorBuilder.matchIPv6FlowLabel(ipv6Packet.getFlowLabel());
            }

            if (matchTcpUdpPorts && ipv6NextHeader == IPv6.PROTOCOL_TCP) {
                TCP tcpPacket = (TCP) ipv6Packet.getPayload();
                selectorBuilder.matchIPProtocol(ipv6NextHeader)
                        .matchTcpSrc(TpPort.tpPort(tcpPacket.getSourcePort()))
                        .matchTcpDst(TpPort.tpPort(tcpPacket.getDestinationPort()));
            }
            if (matchTcpUdpPorts && ipv6NextHeader == IPv6.PROTOCOL_UDP) {
                UDP udpPacket = (UDP) ipv6Packet.getPayload();
                selectorBuilder.matchIPProtocol(ipv6NextHeader)
                        .matchUdpSrc(TpPort.tpPort(udpPacket.getSourcePort()))
                        .matchUdpDst(TpPort.tpPort(udpPacket.getDestinationPort()));
            }
            if (matchIcmpFields && ipv6NextHeader == IPv6.PROTOCOL_ICMP6) {
                ICMP6 icmp6Packet = (ICMP6) ipv6Packet.getPayload();
                selectorBuilder.matchIPProtocol(ipv6NextHeader)
                        .matchIcmpv6Type(icmp6Packet.getIcmpType())
                        .matchIcmpv6Code(icmp6Packet.getIcmpCode());
            }
        }
    }
    TrafficTreatment treatment = DefaultTrafficTreatment.builder()
            .setOutput(portNumber)
            .build();

    FlowRule newFlow = DefaultFlowRule.builder()
            .withSelector(selectorBuilder.build())
            .withTreatment(treatment)
            .forDevice(context.inPacket().receivedFrom().deviceId())
            .fromApp(appId)
            .makeTemporary(flowTimeout)
            //.makePermanent()
            .withPriority(flowPriority)
            .build();

    flowRuleService.applyFlowRules(newFlow);
```

```
        forwardPacket(macMetrics);
        //
        // If packetOutOfppTable
        //  Send packet back to the OpenFlow pipeline to match installed flow
        // Else
        //  Send packet direction on the appropriate port
        //
        if (packetOutOfppTable) {
            packetOut(context, PortNumber.TABLE, macMetrics);
        } else {
            packetOut(context, portNumber, macMetrics);
        }
    }

    private void CongestionFlows(PortNumber Out_Port, PacketContext context, ReactiveForwardMetrics macMetrics) {

        Ethernet inPkt = context.inPacket().parsed();
        DeviceId deviceId = context.inPacket().receivedFrom().deviceId();

        TrafficSelector.Builder selectorBuilder = DefaultTrafficSelector.builder();

        selectorBuilder.matchInPort(context.inPacket().receivedFrom().port())
                .matchEthSrc(inPkt.getSourceMAC())
                .matchEthDst(inPkt.getDestinationMAC());

        TrafficTreatment treatment = DefaultTrafficTreatment.builder()
                .setOutput(Out_Port)
                .build();

        FlowRule CongestionFlow = DefaultFlowRule.builder()
                .withSelector(selectorBuilder.build())
                .withTreatment(treatment)
                .forDevice(deviceId)
                .fromApp(appId)
                .makeTemporary(flowTimeout)
                //.makePermanent()
                .withPriority(flowPriority+5)
                .build();

        flowRuleService.applyFlowRules(CongestionFlow);
        forwardPacket(macMetrics);
    }

    private class InternalTopologyListener implements TopologyListener {
        @Override
        public void event(TopologyEvent event) {
            List<Event> reasons = event.reasons();
            if (reasons != null) {
                reasons.forEach(re -> {
                    if (re instanceof LinkEvent) {
                        LinkEvent le = (LinkEvent) re;
                        if (le.type() == LinkEvent.Type.LINK_REMOVED && blackHoleExecutor != null) {
                            blackHoleExecutor.submit(() -> fixBlackhole(le.subject().src()));
                        }
                    }
                });
            }
        }
    }

    private void fixBlackhole(ConnectPoint egress) {
        Set<FlowEntry> rules = getFlowRulesFrom(egress);
        Set<SrcDstPair> pairs = findSrcDstPairs(rules);

        Map<DeviceId, Set<Path>> srcPaths = new HashMap<>();

        for (SrcDstPair sd : pairs) {
```

```
        // get the edge deviceID for the src host
        Host srcHost = hostService.getHost(HostId.hostId(sd.src));
        Host dstHost = hostService.getHost(HostId.hostId(sd.dst));
        if (srcHost != null && dstHost != null) {
            DeviceId srcId = srcHost.location().deviceId();
            DeviceId dstId = dstHost.location().deviceId();
            log.trace("SRC ID is {}, DST ID is {}", srcId, dstId);

            cleanFlowRules(sd, egress.deviceId());

            Set<Path> shortestPaths = srcPaths.get(srcId);
            if (shortestPaths == null) {
                shortestPaths = topologyService.getPaths(topologyService.currentTopology(),
                        egress.deviceId(), srcId);
                srcPaths.put(srcId, shortestPaths);
            }
            backTrackBadNodes(shortestPaths, dstId, sd);
        }
    }
}

// Backtracks from link down event to remove flows that lead to blackhole
private void backTrackBadNodes(Set<Path> shortestPaths, DeviceId dstId, SrcDstPair sd) {
    for (Path p : shortestPaths) {
        List<Link> pathLinks = p.links();
        for (int i = 0; i < pathLinks.size(); i = i + 1) {
            Link curLink = pathLinks.get(i);
            DeviceId curDevice = curLink.src().deviceId();

            // skipping the first link because this link's src has already been pruned beforehand
            if (i != 0) {
                cleanFlowRules(sd, curDevice);
            }

            Set<Path> pathsFromCurDevice =
                    topologyService.getPaths(topologyService.currentTopology(),
                            curDevice, dstId);
            if (pickForwardPathIfPossible(pathsFromCurDevice, curLink.src().port()) != null) {
                break;
            } else {
                if (i + 1 == pathLinks.size()) {
                    cleanFlowRules(sd, curLink.dst().deviceId());
                }
            }
        }
    }
}

// Removes flow rules off specified device with specific SrcDstPair
private void cleanFlowRules(SrcDstPair pair, DeviceId id) {
    log.trace("Searching for flow rules to remove from: {}", id);
    log.trace("Removing flows w/ SRC={}, DST={}", pair.src, pair.dst);
    for (FlowEntry r : flowRuleService.getFlowEntries(id)) {
        boolean matchesSrc = false, matchesDst = false;
        for (Instruction i : r.treatment().allInstructions()) {
            if (i.type() == Instruction.Type.OUTPUT) {
                // if the flow has matching src and dst
                for (Criterion cr : r.selector().criteria()) {
                    if (cr.type() == Criterion.Type.ETH_DST) {
                        if (((EthCriterion) cr).mac().equals(pair.dst)) {
                            matchesDst = true;
                        }
                    } else if (cr.type() == Criterion.Type.ETH_SRC) {
                        if (((EthCriterion) cr).mac().equals(pair.src)) {
                            matchesSrc = true;
                        }
                    }
                }
```

```
            }
        }
        if (matchesDst && matchesSrc) {
            log.trace("Removed flow rule from device: {}", id);
            flowRuleService.removeFlowRules((FlowRule) r);
        }
    }

}

// Returns a set of src/dst MAC pairs extracted from the specified set of flow entries
private Set<SrcDstPair> findSrcDstPairs(Set<FlowEntry> rules) {
    ImmutableSet.Builder<SrcDstPair> builder = ImmutableSet.builder();
    for (FlowEntry r : rules) {
        MacAddress src = null, dst = null;
        for (Criterion cr : r.selector().criteria()) {
            if (cr.type() == Criterion.Type.ETH_DST) {
                dst = ((EthCriterion) cr).mac();
            } else if (cr.type() == Criterion.Type.ETH_SRC) {
                src = ((EthCriterion) cr).mac();
            }
        }
        builder.add(new SrcDstPair(src, dst));
    }
    return builder.build();
}

private ReactiveForwardMetrics createCounter(MacAddress macAddress) {
    ReactiveForwardMetrics macMetrics = null;
    if (recordMetrics) {
        macMetrics = metrics.compute(macAddress, (key, existingValue) -> {
            if (existingValue == null) {
                return new ReactiveForwardMetrics(0L, 0L, 0L, 0L, macAddress);
            } else {
                return existingValue;
            }
        });
    }
    return macMetrics;
}

private void  forwardPacket(ReactiveForwardMetrics macmetrics) {
    if (recordMetrics) {
        macmetrics.incrementForwardedPacket();
        metrics.put(macmetrics.getMacAddress(), macmetrics);
    }
}

private void inPacket(ReactiveForwardMetrics macmetrics) {
    if (recordMetrics) {
        macmetrics.incrementInPacket();
        metrics.put(macmetrics.getMacAddress(), macmetrics);
    }
}

private void replyPacket(ReactiveForwardMetrics macmetrics) {
    if (recordMetrics) {
        macmetrics.incremnetReplyPacket();
        metrics.put(macmetrics.getMacAddress(), macmetrics);
    }
}

private void droppedPacket(ReactiveForwardMetrics macmetrics) {
    if (recordMetrics) {
        macmetrics.incrementDroppedPacket();
        metrics.put(macmetrics.getMacAddress(), macmetrics);
    }
}
```

```
public EventuallyConsistentMap<MacAddress, ReactiveForwardMetrics> getMacAddress() {
    return metrics;
}

public void printMetric(MacAddress mac) {
    System.out.println("-----------------------------------------------------------------------------------------------------");
    System.out.println(" MACADDRESS \t\t\t\t\t Metrics");
    if (mac != null) {
        System.out.println(" " + mac + " \t\t\t " + metrics.get(mac));
    } else {
        for (MacAddress key : metrics.keySet()) {
            System.out.println(" " + key + " \t\t\t " + metrics.get(key));
        }
    }
}

private Set<FlowEntry> getFlowRulesFrom(ConnectPoint egress) {
    ImmutableSet.Builder<FlowEntry> builder = ImmutableSet.builder();
    flowRuleService.getFlowEntries(egress.deviceId()).forEach(r -> {
        if (r.appId() == appId.id()) {
            r.treatment().allInstructions().forEach(i -> {
                if (i.type() == Instruction.Type.OUTPUT) {
                    if (((Instructions.OutputInstruction) i).port().equals(egress.port())) {
                        builder.add(r);
                    }
                }
            });
        }
    });

    return builder.build();
}

// Wrapper class for a source and destination pair of MAC addresses
private final class SrcDstPair {
    final MacAddress src;
    final MacAddress dst;

    private SrcDstPair(MacAddress src, MacAddress dst) {
        this.src = src;
        this.dst = dst;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        SrcDstPair that = (SrcDstPair) o;
        return Objects.equals(src, that.src) &&
                Objects.equals(dst, that.dst);
    }

    @Override
    public int hashCode() {
        return Objects.hash(src, dst);
    }
}

public void setCongestionParameters(long pReceived, long pSend, DeviceId DeviceIdStatics,
            PortNumber PortNumberStatics) {
    this.pReceived = pReceived;
    this.pSend = pSend;
    this.DeviceIdStatics = DeviceIdStatics;
```

```
        this.PortNumberStatics = PortNumberStatics;
        CongestionEstadistics();
    }

    public void ObserverParameter() {
        FixParametersPorts();
        BlockProbabilityDevice();
        MCCC();
        if (Congestion_flag_1) {
            ClearFlowsRulesCongestion_flag = true;
            ClearRulesCongestion(DeviceId.deviceId("of:0000000000000001"), 10);
            ClearRulesCongestion(DeviceId.deviceId("of:000000000000000a"), 10);
        } else {
            if(ClearFlowsRulesCongestion_flag) {
                ClearRulesCongestion(DeviceId.deviceId("of:0000000000000001"), 15);
                ClearRulesCongestion(DeviceId.deviceId("of:000000000000000a"), 15);
            }
        }
    }

    public void CongestionEstadistics () {
        //Switch S1
        if (DeviceIdStatics.equals(DeviceId.deviceId("of:0000000000000001"))) {
            if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
                S1_Port1_SendP = pSend;
                S1_Port1_RecevP = pReceived;
                S1_Port1_RecevPDelta = S1_Port1_RecevP - S1_Port1_RecevPDeltaTemp;
                S1_Port1_RecevPDeltaTemp = S1_Port1_RecevP;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
                S1_Port2_SendP = pSend;
                S1_Port2_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
                S1_Port3_SendP = pSend;
                S1_Port3_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
                S1_Port4_SendP = pSend;
                S1_Port4_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("5"))){
                S1_Port5_SendP = pSend;
                S1_Port5_RecevP = pReceived;
            }
        }
        //Switch S2
        if (DeviceIdStatics.equals(DeviceId.deviceId("of:0000000000000002"))) {
            if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
                S2_Port1_SendP = pSend;
                S2_Port1_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
                S2_Port2_SendP = pSend;
                S2_Port2_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
                S2_Port3_SendP = pSend;
                S2_Port3_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
                S2_Port4_SendP = pSend;
                S2_Port4_RecevP = pReceived;
            }
        }
        //Switch S3
        if (DeviceIdStatics.equals(DeviceId.deviceId("of:0000000000000003"))) {
            if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
                S3_Port1_SendP = pSend;
                S3_Port1_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
                S3_Port2_SendP = pSend;
                S3_Port2_RecevP = pReceived;
            } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
                S3_Port3_SendP = pSend;
```

```
            S3_Port3_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
            S3_Port4_SendP = pSend;
            S3_Port4_RecevP = pReceived;
        }
    }
    //Switch S4
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:0000000000000004"))) {
        if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
            S4_Port1_SendP = pSend;
            S4_Port1_RecevP = pReceived;
        }  else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
            S4_Port2_SendP = pSend;
            S4_Port2_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
            S4_Port3_SendP = pSend;
            S4_Port3_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
            S4_Port4_SendP = pSend;
            S4_Port4_RecevP = pReceived;
        }
    }
    //Switch S6
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:0000000000000006"))) {
        if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
            S6_Port1_SendP = pSend;
            S6_Port1_RecevP = pReceived;
        }  else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
            S6_Port2_SendP = pSend;
            S6_Port2_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
            S6_Port3_SendP = pSend;
            S6_Port3_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
            S6_Port4_SendP = pSend;
            S6_Port4_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("5"))){
            S6_Port5_SendP = pSend;
            S6_Port5_RecevP = pReceived;
        }
    }
    //Switch S8
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:0000000000000008"))) {
        if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
            S8_Port1_SendP = pSend;
            S8_Port1_RecevP = pReceived;
        }  else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
            S8_Port2_SendP = pSend;
            S8_Port2_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
            S8_Port3_SendP = pSend;
            S8_Port3_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
            S8_Port4_SendP = pSend;
            S8_Port4_RecevP = pReceived;
        }
    }
    //Switch S10
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:000000000000000a"))) {
        if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
            S10_Port1_SendP = pSend;
            S10_Port1_RecevP = pReceived;
            S10_Port1_SendPDelta = S10_Port1_SendP - S10_Port1_SendPDeltaTemp;
            S10_Port1_SendPDeltaTemp = S10_Port1_SendP;
        }  else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
            S10_Port2_SendP = pSend;
            S10_Port2_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
```

```
            S10_Port3_SendP = pSend;
            S10_Port3_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
            S10_Port4_SendP = pSend;
            S10_Port4_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("5"))){
            S10_Port5_SendP = pSend;
            S10_Port5_RecevP = pReceived;
        }
    }
    //Switch s11
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:000000000000000b"))) {
        TotalPacketsSendS11 = TotalPacketsSendS11 + pSend;
        TotalPacketsRecevedS11 = TotalPacketsRecevedS11 + pReceived;
    }
    //Switch S12
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:000000000000000c"))) {
        if(PortNumberStatics.equals(PortNumber.portNumber("1"))){
            S12_Port1_SendP = pSend;
            S12_Port1_RecevP = pReceived;
        }  else if (PortNumberStatics.equals(PortNumber.portNumber("2"))){
            S12_Port2_SendP = pSend;
            S12_Port2_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("3"))){
            S12_Port3_SendP = pSend;
            S12_Port3_RecevP = pReceived;
        } else if (PortNumberStatics.equals(PortNumber.portNumber("4"))){
            S12_Port4_SendP = pSend;
            S12_Port4_RecevP = pReceived;
        }
    }
    //Switch s13
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:000000000000000d"))) {
        TotalPacketsSendS13 = TotalPacketsSendS13 + pSend;
        TotalPacketsRecevedS13 = TotalPacketsRecevedS13 + pReceived;
    }
    //Switch s14
    if (DeviceIdStatics.equals(DeviceId.deviceId("of:000000000000000e"))) {
        if(PortNumberStatics.equals(PortNumber.portNumber("2"))) {
            S14_Port2_SendP = pSend;
            S14_Port2_RecevP = pReceived;
        }
    }
}

public void ClearRulesCongestion (DeviceId deviceId, int deletePriority){

    if (deviceId.equals(DeviceId.deviceId("of:0000000000000001")) ||
            deviceId.equals(DeviceId.deviceId("of:000000000000000a"))) {
        for (FlowEntry r : flowRuleService.getFlowEntries(deviceId)) {
            int priority = r.priority();
            boolean matchesSrc = false, matchesDst = false;
            for (Instruction i : r.treatment().allInstructions()) {
                if (i.type() == Instruction.Type.OUTPUT) {
                    for (Criterion cr : r.selector().criteria()) {
                        if (cr.type() == Criterion.Type.ETH_DST) {
                            if (priority == deletePriority && (((EthCriterion) cr).mac().equals(
                                    MacAddress.valueOf("00:00:00:00:00:10")) ||
                                    ((EthCriterion) cr).mac().equals(MacAddress.valueOf("00:00:00:00:00:01")))) {
                                matchesDst = true;
                            }
                        } else if (cr.type() == Criterion.Type.ETH_SRC) {
                            if (priority == deletePriority && (((EthCriterion) cr).mac().equals(
                                    MacAddress.valueOf("00:00:00:00:00:01")) ||
                                    ((EthCriterion) cr).mac().equals(MacAddress.valueOf("00:00:00:00:00:10")))) {
                                matchesSrc = true;
                            }
                        }
```

```
                }
            }
        }
        if (matchesDst && matchesSrc) {
            log.trace("Removed flow rule from device: {}", deviceId);
            flowRuleService.removeFlowRules((FlowRule) r);
        }
    }
  }
}

public void MCCC(){

    float result = 0;

    log.info("Congestion Control MCCC, Congestion Control MCCC, Congestion Control MCCC");

    if (S1_Port1_RecevPDelta > 10 && S10_Port1_SendPDelta > 10){

        if(Case_2 && Congestion_flag_1 && (S1_Port1_RecevP > 500000 && S1_Port1_RecevP < 1000000)){
            Congestion_flag_1 = false;
        }
        result = Math.abs(100 - ((float)(S10_Port1_SendPDelta * 100) / (float)(S1_Port1_RecevPDelta)));
        log.info("El total de paquetes Enviados por el Host 111111 es : {} and Delta {}",
                S1_Port1_RecevP, S1_Port1_RecevPDelta);
        log.info("El total de paquetes Recividos por el Host 101010 es : {} and Delta {} ",
                S10_Port1_SendP, S10_Port1_SendPDelta);
        log.info("La Probabilidad de bloqueo H1 to H10 es :             {}", result);

        if((result >= 1 && CounterPacketsCongestion == 0) || Congestion_flag_1){

            if(!CongestionControlLink) {
                if ((BlockProbaLink_B <= BlockProbaLink_C)) {
                    CongestionportNumberClient = PortNumber.portNumber("3");
                    CongestionportNumberServer = PortNumber.portNumber("4");
                    CongestionControlLink = true;
                } else {
                    CongestionportNumberClient = PortNumber.portNumber("4");
                    CongestionportNumberServer = PortNumber.portNumber("3");
                    CongestionControlLink = true;
                }
            }
            Congestion_flag_1 = true;
            if(Case_2)TimeStopMCCC=99999;
            if (CounterPacketsCongestion >= TimeStopMCCC || StopMCCC){
                if(Case_4 && BlockProbaDeltaLink_A > 0.1){
                    TimeStopMCCC = TimeStopMCCC + DefaultTimeStopMCCC;
                    CounterPacketsCongestion++;
                    log.info("The Congestion Counter isssss:                {}", CounterPacketsCongestion);
                } else {
                    if(Case_4){
                        if(StopMCCC)StopMCCC = false;
                        TimeStopMCCC = DefaultTimeStopMCCC;
                    }
                    Congestion_flag_1 = false;
                    CongestionControlLink = false;
                }
            } else {
                CounterPacketsCongestion++;
                log.info("The Congestion Counter is:                {}", CounterPacketsCongestion);
            }

            if(CongestionportNumberClient != null) {
                log.info("Congestion Control MCCC Activated Sending Packets from port: {}",
                        CongestionportNumberClient);
            }

        } else {
```

```
            Congestion_flag_1 = false;
            CounterPacketsCongestion = 0;
        }
    } else {
        CounterPacketsCongestion = 0;
        Congestion_flag_1 = false;
        CongestionControlLink = false;
    }
}

public void FixParametersPorts (){

    //Switch S1
    S1_Port2_SendP = S1_Port2_SendP - S14_Port2_SendP;
    S1_Port2_RecevP = S1_Port2_RecevP - S14_Port2_RecevP;

    //Switch S2
    S2_Port1_SendP = S2_Port1_SendP - S14_Port2_SendP;
    S2_Port1_RecevP = S2_Port1_RecevP - S14_Port2_RecevP;

    //Switch S3
    S3_Port1_SendP = S3_Port1_SendP - S14_Port2_SendP;
    S3_Port1_RecevP = S3_Port1_RecevP - S14_Port2_RecevP;

    //Switch S4
    S4_Port1_SendP = S4_Port1_SendP - S14_Port2_SendP;
    S4_Port1_RecevP = S4_Port1_RecevP - S14_Port2_RecevP;

    //Switch S6
    S6_Port1_SendP = S6_Port1_SendP - S14_Port2_SendP;
    S6_Port1_RecevP = S6_Port1_RecevP - S14_Port2_RecevP;

    //Switch S8
    S8_Port1_SendP = S8_Port1_SendP - S14_Port2_SendP;
    S8_Port1_RecevP = S8_Port1_RecevP - S14_Port2_RecevP;

    //Switch S10
    S10_Port2_SendP = S10_Port2_SendP - S14_Port2_SendP;
    S10_Port2_RecevP = S10_Port2_RecevP - S14_Port2_RecevP;

    //Switch S11
    TotalPacketsSendS11 = TotalPacketsSendS11 - S14_Port2_SendP;
    TotalPacketsRecevedS11 = TotalPacketsRecevedS11 - S14_Port2_RecevP;

    //Switch S12
    S12_Port1_SendP = S12_Port1_SendP - S14_Port2_SendP;
    S12_Port1_RecevP = S12_Port1_RecevP - S14_Port2_RecevP;

    //Switch S13
    TotalPacketsSendS13 = TotalPacketsSendS13 - S14_Port2_SendP;
    TotalPacketsRecevedS13 = TotalPacketsRecevedS13 - S14_Port2_RecevP;
}

public void BlockProbabilityDevice (){

    long TotalPacketsSendS1 = 0;
    long TotalPacketsRecevedS1 = 0;
    long TotalPacketsSendS2 = 0;
    long TotalPacketsRecevedS2 = 0;
    long TotalPacketsSendS3 = 0;
    long TotalPacketsRecevedS3 = 0;
    long TotalPacketsSendS4 = 0;
    long TotalPacketsRecevedS4 = 0;
    long TotalPacketsSendS6 = 0;
    long TotalPacketsRecevedS6 = 0;
    long TotalPacketsSendS8 = 0;
    long TotalPacketsRecevedS8 = 0;
    long TotalPacketsSendS10 = 0;
```

```
long TotalPacketsRecevedS10 = 0;
long TotalPacketsSendS12 = 0;
long TotalPacketsRecevedS12 = 0;

//Switch S1
TotalPacketsSendS1 = S1_Port1_SendP + S1_Port2_SendP + S1_Port3_SendP + S1_Port4_SendP + S1_Port5_SendP;
TotalPacketsRecevedS1 = S1_Port1_RecevP + S1_Port2_RecevP + S1_Port3_RecevP + S1_Port4_RecevP +
                S1_Port5_RecevP;

BlockProba_S1 = BlockProbability(TotalPacketsSendS1, TotalPacketsRecevedS1);

//Switch S2
TotalPacketsSendS2 = S2_Port1_SendP + S2_Port2_SendP + S2_Port3_SendP + S2_Port4_SendP;
TotalPacketsRecevedS2 = S2_Port1_RecevP + S2_Port2_RecevP + S2_Port3_RecevP + S2_Port4_RecevP;

BlockProba_S2 = BlockProbability(TotalPacketsSendS2, TotalPacketsRecevedS2);

//Switch S3
TotalPacketsSendS3 = S3_Port1_SendP + S3_Port2_SendP + S3_Port3_SendP + S3_Port4_SendP;
TotalPacketsRecevedS3 = S3_Port1_RecevP + S3_Port2_RecevP + S3_Port3_RecevP + S3_Port4_RecevP;

BlockProba_S3 = BlockProbability(TotalPacketsSendS3, TotalPacketsRecevedS3);

//Switch S4
TotalPacketsSendS4 = S4_Port1_SendP + S4_Port2_SendP + S4_Port3_SendP + S4_Port4_SendP;
TotalPacketsRecevedS4 = S4_Port1_RecevP + S4_Port2_RecevP + S4_Port3_RecevP + S4_Port4_RecevP;

BlockProba_S4 = BlockProbability(TotalPacketsSendS4, TotalPacketsRecevedS4);

//Switch S6
TotalPacketsSendS6 = S6_Port1_SendP + S6_Port2_SendP + S6_Port3_SendP + S6_Port4_SendP + S6_Port5_SendP;
TotalPacketsRecevedS6 = S6_Port1_RecevP + S6_Port2_RecevP + S6_Port3_RecevP + S6_Port4_RecevP +
                S6_Port5_RecevP;

BlockProba_S6 = BlockProbability(TotalPacketsSendS6, TotalPacketsRecevedS6);

//Switch S8
TotalPacketsSendS8 = S8_Port1_SendP + S8_Port2_SendP + S8_Port3_SendP + S8_Port4_SendP;
TotalPacketsRecevedS8 = S8_Port1_RecevP + S8_Port2_RecevP + S8_Port3_RecevP + S8_Port4_RecevP;

BlockProba_S8 = BlockProbability(TotalPacketsSendS8, TotalPacketsRecevedS8);

//Switch S10
TotalPacketsSendS10 = S10_Port1_SendP + S10_Port2_SendP + S10_Port3_SendP + S10_Port4_SendP + S10_Port5_SendP;
TotalPacketsRecevedS10 = S10_Port1_RecevP + S10_Port2_RecevP + S10_Port3_RecevP + S10_Port4_RecevP +
                S10_Port5_RecevP;

BlockProba_S10 = BlockProbability(TotalPacketsSendS10, TotalPacketsRecevedS10);

//Switch S11
BlockProba_S11 = BlockProbability(TotalPacketsSendS11, TotalPacketsRecevedS11);

//Switch S12
TotalPacketsSendS12 = S12_Port1_SendP + S12_Port2_SendP + S12_Port3_SendP + S12_Port4_SendP;
TotalPacketsRecevedS12 = S12_Port1_RecevP + S12_Port2_RecevP + S12_Port3_RecevP + S12_Port4_RecevP;

BlockProba_S12 = BlockProbability(TotalPacketsSendS12, TotalPacketsRecevedS12);

//Switch S13
BlockProba_S13 = BlockProbability(TotalPacketsSendS13, TotalPacketsRecevedS13);

log.info("El total de paquetes Enviados en S11111111 es : {}", TotalPacketsSendS1);
log.info("El total de paquetes Recividos en S11111111 es : {}", TotalPacketsRecevedS1);
log.info("La Probabilidad de bloqueo en S11111111 es : {}", BlockProba_S1);

log.info("El total de paquetes Enviados en S1010101010 es : {}", TotalPacketsSendS10);
log.info("El total de paquetes Recividos en S1010101010 es : {}", TotalPacketsRecevedS10);
log.info("La Probabilidad de bloqueo en S1010101010 es : {}", BlockProba_S10);
```

```
    //Link B S2 S4 S12
    BlockProbaLink_A =(BlockProba_S11 + BlockProba_S13)/2;
    log.info("La Probabilidad del Enlace AAAAAAAA es : {}", BlockProbaLink_A);
    BlockProbaLink_B = (BlockProba_S2 + BlockProba_S4 + BlockProba_S12)/3;
    log.info("La Probabilidad del Enlace BBBBBBBB es : {}", BlockProbaLink_B);
    BlockProbaLink_C = (BlockProba_S3 + BlockProba_S6 + BlockProba_S8)/3;
    log.info("La Probabilidad del Enlace CCCCCCCC es : {}", BlockProbaLink_C);
    if(Case_4) {
        //BlockProbabilityDeltaLink_A
        BlockProbabilityDeltaLink_A();
        if (Congestion_flag_1) {
            log.info("La Probabilidad del Enlace Delta AAAA es : {}", BlockProbaDeltaLink_A);
            if (BlockProbaDeltaLink_A < 0.1) {
                CounterPacketsCongestionLink_A++;
                if (CounterPacketsCongestionLink_A == 3) {
                    StopMCCC = true;
                    CounterPacketsCongestionLink_A = 0;
                }
            } else {
                CounterPacketsCongestionLink_A = 0;
            }
        }
    }
}

public void BlockProbabilityDeltaLink_A (){
    float BlockProbaDelta_S11 = 0;
    float BlockProbaDelta_S13 = 0;

    long TotalPacketsSendDeltaS11 = 0;
    long TotalPacketsRecevedDeltaS11 = 0;

    long TotalPacketsSendDeltaS13 = 0;
    long TotalPacketsRecevedDeltaS13 = 0;

    //BlockProbabilityDeltaS11
    //Send
    TotalPacketsSendDeltaS11 = TotalPacketsSendS11 - TotalPacketsSendDeltaS11Temp;
    TotalPacketsSendDeltaS11Temp = TotalPacketsSendS11;
    //Received
    TotalPacketsRecevedDeltaS11 = TotalPacketsRecevedS11 - TotalPacketsRecevedDeltaS11Temp;
    TotalPacketsRecevedDeltaS11Temp = TotalPacketsRecevedS11;

    BlockProbaDelta_S11 = BlockProbability(TotalPacketsSendDeltaS11, TotalPacketsRecevedDeltaS11);
    log.info("La Probabilidad del Enlace Delta AAAA S11 es : {}", BlockProbaDelta_S11);

    //BlockProbabilityDeltaS11
    //Send
    TotalPacketsSendDeltaS13 = TotalPacketsSendS13 - TotalPacketsSendDeltaS13Temp;
    TotalPacketsSendDeltaS13Temp = TotalPacketsSendS13;
    //Received
    TotalPacketsRecevedDeltaS13 = TotalPacketsRecevedS13 - TotalPacketsRecevedDeltaS13Temp;
    TotalPacketsRecevedDeltaS13Temp = TotalPacketsRecevedS13;

    BlockProbaDelta_S13 = BlockProbability(TotalPacketsSendDeltaS13, TotalPacketsRecevedDeltaS13);
    log.info("La Probabilidad del Enlace Delta AAAA S13 es : {}", BlockProbaDelta_S13);

    //BlockProbabilityDeltaLink_A
    BlockProbaDeltaLink_A = (BlockProbaDelta_S11 + BlockProbaDelta_S13)/2;

    TotalPacketsSendS11 = 0;
    TotalPacketsRecevedS11 = 0;
    TotalPacketsSendS13 = 0;
    TotalPacketsRecevedS13 = 0;

}
```

```
public float BlockProbability (double TotalPacketsSend, double TotalPacketsReceved){

    float BlockProba = 0;

    if ((Math.abs(TotalPacketsSend - TotalPacketsReceved)) < 16){
        return 0;
    } else {
        BlockProba = Math.abs(100 - ((float)(TotalPacketsReceved * 100) / (float)(TotalPacketsSend)));
        return  BlockProba;
    }
}

//port variables
private int CounterPacketsCongestion = 0;
private int CounterPacketsCongestionLink_A = 0;

private long S1_Port1_RecevPDelta = 0;
private long S10_Port1_SendPDelta = 0;
private long S1_Port1_RecevPDeltaTemp = 0;
private long S10_Port1_SendPDeltaTemp = 0;

private long TotalPacketsSendS11 = 0;
private long TotalPacketsRecevedS11 = 0;

private long TotalPacketsSendDeltaS11Temp = 0;
private long TotalPacketsRecevedDeltaS11Temp = 0;

private long TotalPacketsSendS13 = 0;
private long TotalPacketsRecevedS13 = 0;

private long TotalPacketsSendDeltaS13Temp = 0;
private long TotalPacketsRecevedDeltaS13Temp = 0;

private float BlockProbaLink_A = 0;
private float BlockProbaDeltaLink_A = 0;
private float BlockProbaLink_B = 0;
private float BlockProbaLink_C = 0;

private float BlockProba_S1 = 0;
private float BlockProba_S2 = 0;
private float BlockProba_S3 = 0;
private float BlockProba_S4 = 0;
private float BlockProba_S6 = 0;
private float BlockProba_S8 = 0;
private float BlockProba_S10 = 0;
private float BlockProba_S11 = 0;
private float BlockProba_S12 = 0;
private float BlockProba_S13 = 0;

private long S1_Port1_SendP = 0;
private long S1_Port1_RecevP = 0;
private long S1_Port2_SendP = 0;
private long S1_Port2_RecevP = 0;
private long S1_Port3_SendP = 0;
private long S1_Port3_RecevP = 0;
private long S1_Port4_SendP = 0;
private long S1_Port4_RecevP = 0;
private long S1_Port5_SendP = 0;
private long S1_Port5_RecevP = 0;

private long S2_Port1_SendP = 0;
private long S2_Port1_RecevP = 0;
private long S2_Port2_SendP = 0;
private long S2_Port2_RecevP = 0;
private long S2_Port3_SendP = 0;
private long S2_Port3_RecevP = 0;
private long S2_Port4_SendP = 0;
private long S2_Port4_RecevP = 0;
```

```
    private long S3_Port1_SendP = 0;
    private long S3_Port1_RecevP = 0;
    private long S3_Port2_SendP = 0;
    private long S3_Port2_RecevP = 0;
    private long S3_Port3_SendP = 0;
    private long S3_Port3_RecevP = 0;
    private long S3_Port4_SendP = 0;
    private long S3_Port4_RecevP = 0;

    private long S4_Port1_SendP = 0;
    private long S4_Port1_RecevP = 0;
    private long S4_Port2_SendP = 0;
    private long S4_Port2_RecevP = 0;
    private long S4_Port3_SendP = 0;
    private long S4_Port3_RecevP = 0;
    private long S4_Port4_SendP = 0;
    private long S4_Port4_RecevP = 0;

    private long S6_Port1_SendP = 0;
    private long S6_Port1_RecevP = 0;
    private long S6_Port2_SendP = 0;
    private long S6_Port2_RecevP = 0;
    private long S6_Port3_SendP = 0;
    private long S6_Port3_RecevP = 0;
    private long S6_Port4_SendP = 0;
    private long S6_Port4_RecevP = 0;
    private long S6_Port5_SendP = 0;
    private long S6_Port5_RecevP = 0;

    private long S8_Port1_SendP = 0;
    private long S8_Port1_RecevP = 0;
    private long S8_Port2_SendP = 0;
    private long S8_Port2_RecevP = 0;
    private long S8_Port3_SendP = 0;
    private long S8_Port3_RecevP = 0;
    private long S8_Port4_SendP = 0;
    private long S8_Port4_RecevP = 0;

    private long S10_Port1_SendP = 0;
    private long S10_Port1_RecevP = 0;
    private long S10_Port2_SendP = 0;
    private long S10_Port2_RecevP = 0;
    private long S10_Port3_SendP = 0;
    private long S10_Port3_RecevP = 0;
    private long S10_Port4_SendP = 0;
    private long S10_Port4_RecevP = 0;
    private long S10_Port5_SendP = 0;
    private long S10_Port5_RecevP = 0;

    private long S12_Port1_SendP = 0;
    private long S12_Port1_RecevP = 0;
    private long S12_Port2_SendP = 0;
    private long S12_Port2_RecevP = 0;
    private long S12_Port3_SendP = 0;
    private long S12_Port3_RecevP = 0;
    private long S12_Port4_SendP = 0;
    private long S12_Port4_RecevP = 0;

    private long S14_Port2_SendP = 0;
    private long S14_Port2_RecevP = 0;
}
```

PortStatsListener.java

```
package org.mccc.app;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.TimeUnit;

import org.onlab.packet.Ip6Address;
import org.onlab.packet.IpAddress;
import org.onosproject.net.ConnectPoint;
import org.onosproject.net.Device;
import org.onosproject.net.PortNumber;
import org.onosproject.net.DeviceId;
import org.onosproject.net.MastershipRole;
import org.onosproject.net.device.DeviceEvent;
import org.onosproject.net.device.DeviceListener;
import org.onosproject.net.device.PortStatistics;

import com.google.common.primitives.Longs;

/**
 * Internal PortStats Listener.
 * Exports IPFIX statistics when PortStats are updated.
 */
public class PortStatsListener implements DeviceListener {

    private ReactiveForwarding reactiveForwarding;

    private Long inPackets = null;
    private Long outPackets = null;

    //***************************************
    private boolean flag_S1 = false;
    private boolean flag_S2 = false;
    private boolean flag_S3 = false;
    private boolean flag_S4 = false;
    private boolean flag_S5 = false;
    private boolean flag_S6 = false;
    private boolean flag_S7 = false;
    private boolean flag_S8 = false;
    private boolean flag_S9 = false;
    private boolean flag_S10 = false;
    private boolean flag_S11 = false;
    private boolean flag_S12 = false;
    private boolean flag_S13 = false;
    private boolean flag_S14 = false;
    //***************************************

    /**
     * Internal PortStats Listener.
     * Exports IPFIX statistics when PortStats are updated.
     *
     * @param ipfixManager ipfix manager instance
     */
    public PortStatsListener(ReactiveForwarding reactiveForwarding) {
        this.reactiveForwarding = reactiveForwarding;
    }

    @Override
    public void event(DeviceEvent event) {
```

```
Device device = event.subject();
DeviceId deviceId = device.id();
PortNumber portnumber = null;

switch (event.type()) {
    case PORT_STATS_UPDATED:
        if (reactiveForwarding.deviceService.getRole(device.id()) == MastershipRole.MASTER) {

            for (PortStatistics stat : reactiveForwarding.deviceService.getPortStatistics(device.id())) {

                final String format = "PortStatsListener Stats: port={}, pktRx={}, pktTx={},"
                        + "bytesRx={}, bytesTx={}, pktRxDrp={}, pktTxDrp={}, Dur={}.{}";
                reactiveForwarding.log.trace(format, stat.portNumber(), stat.packetsReceived(),
                                stat.packetsSent(),
                        stat.bytesReceived(), stat.bytesSent(),
                        stat.packetsRxDropped(), stat.packetsTxDropped(),
                        stat.durationSec(), (stat.durationNano() / 1000000));

                portnumber = stat.portNumber();

                inPackets = stat.packetsReceived();
                outPackets = stat.packetsSent();
                long endTime = System.currentTimeMillis();
                long startTime = endTime - (stat.durationSec() * TimeUnit.SECONDS.toMillis(1) +
                        stat.durationNano() / TimeUnit.MILLISECONDS.toNanos(1));

                //MCCC
                reactiveForwarding.setCongestionParameters(inPackets, outPackets, deviceId, portnumber);
            }
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000001"))) {flag_S1 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000002"))) {flag_S2 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000003"))) {flag_S3 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000004"))) {flag_S4 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000005"))) {flag_S5 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000006"))) {flag_S6 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000007"))) {flag_S7 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000008"))) {flag_S8 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:0000000000000009"))) {flag_S9 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:000000000000000a"))) {flag_S10 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:000000000000000b"))) {flag_S11 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:000000000000000c"))) {flag_S12 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:000000000000000d"))) {flag_S13 = true;}
            if(deviceId.equals(DeviceId.deviceId("of:000000000000000e"))) {flag_S14 = true;}
        }
        break;

    default:
        break;
}

if(flag_S1 && flag_S2 && flag_S3 && flag_S4 && flag_S5 && flag_S6 && flag_S7 && flag_S8 && flag_S9 &&
        flag_S10 && flag_S11 && flag_S12 && flag_S13 && flag_S14){

    flag_S1 = false;
    flag_S2 = false;
    flag_S3 = false;
    flag_S4 = false;
    flag_S5 = false;
    flag_S6 = false;
    flag_S7 = false;
    flag_S8 = false;
    flag_S9 = false;
    flag_S10 = false;
    flag_S11 = false;
    flag_S12 = false;
    flag_S13 = false;
    flag_S14 = false;
    reactiveForwarding.log.info("AllFlags Is True, AllFlags Is True, AllFlags Is True, AllFlags Is True");
```

```
            reactiveForwarding.ObserverParameter();
        }
    }
}
```

OsgiPropertyConstants.java

```
package org.mccc.app;

public final class OsgiPropertyConstants {
    private OsgiPropertyConstants() {
    }

    static final String PACKET_OUT_ONLY = "packetOutOnly";
    static final boolean PACKET_OUT_ONLY_DEFAULT  = false;

    static final String  PACKET_OUT_OFPP_TABLE  = "packetOutOfppTable";
    static final boolean PACKET_OUT_OFPP_TABLE_DEFAULT = false;

    static final String FLOW_TIMEOUT = "flowTimeout";
    static final int FLOW_TIMEOUT_DEFAULT = 10;

    static final String FLOW_PRIORITY = "flowPriority";
    static final int FLOW_PRIORITY_DEFAULT = 10;

    static final String IPV6_FORWARDING = "ipv6Forwarding";
    static final boolean IPV6_FORWARDING_DEFAULT = false;

    static final String MATCH_DST_MAC_ONLY = "matchDstMacOnly";
    static final boolean MATCH_DST_MAC_ONLY_DEFAULT = false;

    static final String MATCH_VLAN_ID = "matchVlanId";
    static final boolean MATCH_VLAN_ID_DEFAULT = false;

    static final String MATCH_IPV4_ADDRESS = "matchIpv4Address";
    static final boolean MATCH_IPV4_ADDRESS_DEFAULT = false;

    static final String MATCH_IPV4_DSCP = "matchIpv4Dscp";
    static final boolean MATCH_IPV4_DSCP_DEFAULT = false;

    static final String MATCH_IPV6_ADDRESS = "matchIpv6Address";
    static final boolean MATCH_IPV6_ADDRESS_DEFAULT = false;

    static final String MATCH_IPV6_FLOW_LABEL = "matchIpv6FlowLabel";
    static final boolean MATCH_IPV6_FLOW_LABEL_DEFAULT = false;

    static final String MATCH_TCP_UDP_PORTS = "matchTcpUdpPorts";
    static final boolean MATCH_TCP_UDP_PORTS_DEFAULT = false;

    static final String MATCH_ICMP_FIELDS = "matchIcmpFields";
    static final boolean MATCH_ICMP_FIELDS_DEFAULT = false;

    static final String IGNORE_IPV4_MCAST_PACKETS = "ignoreIPv4Multicast";
    static final boolean IGNORE_IPV4_MCAST_PACKETS_DEFAULT = false;

    static final String RECORD_METRICS = "recordMetrics";
    static final boolean RECORD_METRICS_DEFAULT = false;
}
```

ReactiveForwardMetrics.java

```
package org.mccc.app;

import org.onlab.packet.MacAddress;
import static com.google.common.base.MoreObjects.toStringHelper;

/**
 * Sample reactive forwarding application.
 */
public class ReactiveForwardMetrics {
    private Long replyPacket = null;
    private Long inPacket = null;
    private Long droppedPacket = null;
    private Long forwardedPacket = null;
    private MacAddress macAddress;

    ReactiveForwardMetrics(Long replyPacket, Long inPacket, Long droppedPacket,
                           Long forwardedPacket, MacAddress macAddress) {
        this.replyPacket = replyPacket;
        this.inPacket = inPacket;
        this.droppedPacket = droppedPacket;
        this.forwardedPacket = forwardedPacket;
        this.macAddress = macAddress;
    }

    public void incremnetReplyPacket() {
        replyPacket++;

    }

    public void incrementInPacket() {
        inPacket++;
    }

    public void incrementDroppedPacket() {
        droppedPacket++;
    }

    public void incrementForwardedPacket() {
        forwardedPacket++;
    }

    public MacAddress getMacAddress() {
        return macAddress;
    }

    @Override
    public String toString() {
        return toStringHelper(this)
                .add("inpktCounter ", inPacket)
                .add("replypktCounter ", replyPacket)
                .add("forwardpktCounter ", forwardedPacket)
                .add("droppktCounter ", droppedPacket).toString();
    }
}
```