



FACULTAD DE INGENIERIA, UNIVERSIDAD
DE LA REPUBLICA

Estudio de alternativas de Redes de Contenido
para la Red Académica Uruguaya
Tesis de Ingeniería en computación

Mathias Duarte
Tutor: Eduardo Grampin
Co-Tutor: Martin Giachino

Mayo 2017

Contenidos

Contenidos	2
Lista de Figuras	3
Lista de Tablas	5
1. Introducción	6
2. Objetivos	8
3. Resultados Esperados	9
4. RAU	10
5. Estado del Arte - SDN	11
5.1. ¿Que es SDN?	11
5.2. Arquitectura	13
5.2.1. Capas de la arquitectura	13
5.2.2. OpenFlow	14
5.2.2.1. Switch OpenFlow	14
5.2.3. Controladores SDN	18
6. Estado del Arte - CDN	19
6.1. ¿Que es CDN? Conceptos Básicos	19
6.2. Evolución Historica	22
6.2.1. Primera Generación	24
6.2.2. Segunda Generación	25
6.2.3. Tercera Generación	26
6.3. Arquitectura	27
6.3.1. Sistema de distribución CDN	29
6.3.1.1. Ubicación de surrogates	30
6.3.1.2. Selección de Contenido y Distribución	31
6.3.1.3. Externalización de Contenido	32
6.3.1.4. Organización de Caché	33
6.3.2. Sistema de encaminamiento CDN	36
6.3.2.1. Algoritmos de encaminamiento	37
6.3.2.2. Mecanismos de encaminamiento	39
6.3.2.2.1. Balanceo global de carga (GSLB)	39
6.3.2.2.2. Redirección basado en DNS	40
6.3.2.2.3. Redirección HTTP	41
6.3.2.2.4. URL rewriting	41
6.3.2.2.5. Anycasting	41
6.3.2.2.6. CDN Peering	42

7. Arquitectura SDN - CDN	43
7.1. Trabajos relacionados	43
7.2. Arquitectura propuesta	49
7.3. Arquitectura propuesta vs CDN Tradicional	54
8. Implementación arquitectura propuesta	56
8.1. MiniNet	56
8.2. Implementación de componentes	59
8.2.1. Tecnologías	59
8.2.1.1. Node.js	59
8.2.1.2. mongoDB	59
8.2.1.3. AngularJS	59
8.2.1.4. Ryu Controller	60
8.2.2. Modelo de datos	60
8.2.3. Base de datos	62
8.2.4. Ryu App	62
8.2.4.1. Implementación para una red ethernet	62
8.2.4.2. Implementación para una red IP	67
8.2.4.2.1. Configuración BGP	68
8.2.4.3. Servicio RESTful	68
8.2.5. Algoritmo de selección surrogate	70
8.2.6. Controlador CDN	71
8.2.7. Administración Web	72
8.2.8. Otras configuraciones	73
8.3. Despliegue Arquitectura propuesta en RAU	73
9. Conclusiones	74
Bibliografía	76
A. Pruebas de concepto	80
A.1. Prueba 1	80
A.2. Prueba 2	85
A.3. Prueba 3	91
A.4. Prueba 4	94
A.5. Prueba 5	97
B. Códigos Fuente	107

Lista de Figuras

1. Comparación arquitecturas	12
2. Arquitectura SDN	13
3. Componentes OpenFlow Switch	15
4. Campos entrada tabla de flujos	16
5. Tratamiento de flujos entrantes	17

6.	Distribución tradicional vs CDN	20
7.	Esquema básico CDN	21
8.	Generaciones CDN	24
9.	Relacion Componentes CDN	28
10.	Taxonomía CDN	29
11.	Clasificación sistema de distribución	30
12.	Estrategias ubicación surrogates	30
13.	Clasificación distribución	31
14.	Clasificación tecnicas de caching	34
15.	Clasificación actualización caché	35
16.	Clasificación algoritmos de encaminamiento	37
17.	Clasificación mecanismos de encaminamiento	39
18.	Arquitectura CDN-SDN 1	44
19.	Flujo de mensajes, contenido en hash	45
20.	Flujo de mensajes, no contenido en hash	45
21.	Arquitectura CDN-SDN 2	46
22.	Arquitectura CDN-SDN 4	48
23.	Arquitectura Propuesta	50
24.	Flujo Administrador	52
25.	Flujo Cliente	53
26.	Mininet	57
27.	Estructura MiniNet	58
28.	Modelo de datos	60
29.	Inicio Ryu App	64
30.	Captura paquete Ryu App	65
31.	Prueba Concepto 1	80
32.	Inicio topología 1	82
33.	Tabla de flujos inicial	82
34.	HTTP Request rechazado	82
35.	Captura HTTP Request rechazado	83
36.	Tabla de flujos s1	83
37.	HTTP Request OK	84
38.	Captura HTTP Request OK	84
39.	Captura HTTP Request OK 2	84
40.	Captura HTTP Request OK 3	85
41.	Prueba Concepto 2	86
42.	Inicio topología 2	87
43.	Capturas prueba 2	87
44.	Match Flow Add	88
45.	Actions Flow Add	88
46.	Capturas prueba 2	89
47.	Match Flow Add	90
48.	Actions Flow Add	90
49.	Tabla de flujos s1	91
50.	Prueba Concepto 3	92
51.	Prueba 3	93

52.	Prueba 3	93
53.	r3 modifica ip destino	96
54.	Administración CDN	98
55.	Agregar Origin	99
56.	Agregar Surrogate	99
57.	Eliminar Origin	100
58.	Eliminar Surrogate	100
59.	CDN sitio web	101
60.	Topología prueba 5	102
61.	Capturas prueba 5	103
62.	Capturas prueba 5	103
63.	Tablas flujo prueba 5	104
64.	Capturas prueba 5	104
65.	Capturas prueba 5	105
66.	Tablas flujo prueba 5	105
67.	Capturas prueba 5	106
68.	Capturas prueba 5	106
69.	Tablas flujo prueba 5	106

Lista de Tablas

1.	Métodos Controlador SDN.	68
2.	Métodos Controlador CDN.	71

1. Introducción

El tráfico académico cuenta con peculiaridades que lo distinguen de las aplicaciones comerciales, y que se vinculan con el tráfico de grandes volúmenes de datos de investigación (por ejemplo, de radiotelescopios, imágenes médicas, entre muchos otros) para su procesamiento remoto, así como las aulas virtuales y servicios de videoconferencia multipunto en general. Estos ejemplos entran dentro de la categoría de Redes de Distribución de Contenido (CDN), donde es fundamental que las aplicaciones dispongan de estrategias de uso adecuado de la red para entregar el contenido en forma óptima y haciendo un uso equilibrado de los recursos. Existe una cantidad creciente de aplicaciones que combinan el uso intensivo de cómputo con el acceso a gigantescos volúmenes de información; cuando el origen de los datos está ligado a multifuentes y no son muy estructurados (sensores de todo tipo, imágenes, datos de genómica, datos abiertos, entre otros), se suele hablar de “BigData”. Los dominios de esas aplicaciones son de los más variados, pasando por biología, meteorología, logística, comercio, economía o finanzas. Algunas de las aplicaciones que se beneficiarían de un despliegue adecuado de una CDN académica, que permita hacer un uso racional de los recursos incluyen, por ejemplo, el procesamiento en forma automática y transparente sobre el cluster de la Facultad de Ingeniería, de los grandes volúmenes de datos generados por los exámenes PET realizados por el Centro Uruguayo de Imagenología Molecular (CUDIM), o de una secuenciación de ADN en el Pasteur, o de los datos meteorológicos que permiten predecir inundaciones o planificar cuáles son las alternativas energéticas más convenientes en el horizonte próximo. Aplicaciones como las anteriormente referidas requieren la implementación de servicios de comunicación con valor añadido, sobre una red de contenido apropiada y con interfaces adecuadas para que distintos miembros de la RAU (investigadores, médicos, organismos públicos) puedan hacer uso transparente y eficiente de los servicios de cómputo. Además de desarrollar software y tener a disposición una red de alta velocidad, el despliegue de una CDN requiere un grado adicional de integración entre la infraestructura de la red y los componentes informáticos, que permita explotar en forma coordinada el máximo de recursos disponibles en el sistema. Esto es inusual en las aplicaciones comúnmente implementadas sobre Internet, donde los usuarios tienen un mínimo control de su tráfico dentro de la red, y además la contribución puntual de tráfico de cada usuario es insignificante en el total. Este nivel de integración es una tendencia creciente; a nivel comercial es usado por grandes y contadas corporaciones (como Google o Netflix), y han tenido una explosión de uso bajo el nombre de “Cloud Computing”, es decir, una abstracción donde los datos se transfieren a sitios con la capacidad de cómputo adecuada de forma transparente al usuario y con tiempos de respuesta muy aceptables si la infraestructura de red es la adecuada. El concepto general de infraestructura como servicio (Infraestructure as a Service - IaaS), y sus instancias de software como servicio (Software as a Service - SaaS) y red como servicio (Network as a Service - NaaS), es un camino de evolución de servicios comerciales, y es de interés estratégico

para el país que sea investigado en la UdelaR. En este sentido, se ha avanzado en la implementación de un prototipo de RAU de Altas Prestaciones basado en el paradigma de las redes definidas por software (Software Defined Networks - SDN), y es viable seguir explorando estas ideas para la implementación de una plataforma transversal e integrada de servicio académicos.

2. Objetivos

El objetivo principal de este proyecto es evaluar alternativas de despliegue de una CDN en el ámbito de la RAU. Se plantea utilizar el prototipo basado en SDN como base del trabajo, y herramientas de simulación/emulación que hagan posible pruebas de escala que permitan sacar conclusiones válidas para un potencial despliegue operativo.

A partir de lo expresado anteriormente, los objetivos principales del proyecto son: (i) resumir el estado del arte en redes de distribución de contenido, (ii) relevar los requerimientos específicos para la RAU, (iii) proponer una arquitectura en base a estos requerimientos, y (iv) construir una prueba de concepto en ambiente de simulación/emulación.

Eventualmente se puede extender a desplegar un prototipo funcional, de acuerdo al desarrollo del proyecto.

3. Resultados Esperados

Se espera contar con un buen documento de relevamiento de estado del arte, y con una prueba de concepto que evalúe diversos aspectos de un posible despliegue. Asimismo, se espera que el proyecto se difunda mediante una página web y se elabore un reporte técnico en formato de artículo de investigación.

4. RAU

La Red Académica Uruguaya (RAU) es un emprendimiento de la Universidad de la República, administrado por el Servicio Central de Informática Universitario (SeCIU) que opera desde el año 1988.

Reúne a las Facultades, Escuelas, Institutos y Servicios de la Universidad de la República y a numerosas entidades de educación e investigación del país. (ver Recursos de Información de la RAU).

Está al servicio de todos los actores académicos del país y de la sociedad uruguaya en su conjunto.

Atenta a las pautas establecidas en las instancias regionales y mundiales de reflexión sobre la misión de las Redes Académicas Universitarias, la RAU, busca ser un ámbito de integración, comunicación y discusión, al servicio de los objetivos de la educación, la investigación y las transformaciones de la sociedad.

La RAU juega un papel muy importante como herramienta de difusión, intercambio y acceso a los centros de información nacionales, regionales e internacionales, así como en la ejecución y defensa de las políticas e intereses de la comunidad académica en estos temas (ver Documentos Marco).

Los nodos de la Red Académica Uruguaya son un total de 153, pudiéndose discriminar entre los pertenecientes a la Universidad de la República 31 TCP/IP, 82 UUCP y los de otras entidades Académicas y de Investigación .25 TCP/IP, 14 UUCP.

Los recursos de información de la RAU se pueden medir en:

- Número de instituciones que la integran: 37
- Número de servidores WWW: 62
- Número de servidores FTP anónimos: 10

Dentro de los nodos de la Universidad de la República, cada servicio ha sido conectado desde mediados de 1996, en el marco de un proyecto central auspiciado por la RAU y financiado por la Comisión Sectorial de Investigación Científica (CSIC), recibiendo equipos, conectividad sin costo alguno y apoyo técnico.

La RAU está al servicio de 31 Facultades, Institutos y Escuelas, 6.516 docentes, 1.065 técnicos, y 60.000 estudiantes.

Su sitio WWW oficial, / expone 2.500 páginas, es visitado 390.000 veces por un promedio de 7.700 hosts distintos al mes, enlaza más de 2.100 páginas y 600 e-mails.

Por más información consultar [1].

En cuanto al prototipo de RAU basado en SDN, se puede observar su documentación en [2].

5. Estado del Arte - SDN

En esta sección se presentan las redes definidas por software, su arquitectura y los principales componentes de la misma. La presentación de esta tecnología se realiza para contextualizar el trabajo que se realizará en este documento, ya que se ha avanzado en la implementación de un prototipo de RAU de Altas Prestaciones basado en el paradigma de las redes definidas por software (Software Defined Networks - SDN). Un estudio más detallado de esta tecnología fue realizado por mi en un trabajo de tesis de licenciatura, en la carrera de licenciatura en computación en el perfil de redes de computadoras, el mismo se encuentra en [3].

5.1. ¿Que es SDN?

Las redes actuales cuentan cada vez más con configuraciones estáticas y complejas (lo que impide flexibilizarlas a los continuos cambios necesarios en los data center), esto hace que a la hora de crear una nueva red, se aborde su creación con una filosofía distinta a la que se usaba tradicionalmente. Los distintos proveedores y consumidores de servicios en la red, tienen cada vez más necesidades tecnológicas, las cuales son cada vez más difíciles de satisfacer por las redes tradicionales. Es decir, las redes tradicionales no son suficientemente ágiles como para reprogramarse, reconfigurarse y reajustarse a los despliegues de nuevos servicios. Las redes tradicionales generalmente presentan una arquitectura jerárquica, que están orientadas a tráfico norte-sur (cliente-servidor), lo cual, genera un importante problema, ya que las aplicaciones de hoy en día requieren de altas necesidades de tráfico este-oeste (tráfico entre servidores, no jerárquico), este tráfico supone la mayor carga de trabajo para la red. En el 2014, el 80 % de tráfico en el data center fue este-oeste. Por estos motivos, se necesita un cambio de arquitectura de red que se acerque a las necesidades actuales, que sea capaz de lograr flexibilidad, escalabilidad, automatización y control. Las redes definidas por software (SDN - software defined networking) son una nueva forma de abordar la creación de redes, en la cual el control se desprende del hardware y se le otorga a una aplicación de software, llamada controlador SDN. SDN presenta una arquitectura de red donde el plano del control se separa del plano de datos (logrando independencia en cada uno de los planos), lo cual permite obtener redes más programables, automatizables y flexibles. En definitiva, esta arquitectura le otorga al controlador SDN el control de la red. En la figura 1 se ilustran las arquitecturas mencionadas.

Para entender un poco más cómo funciona SDN, veamos que pasa cuando un paquete llega a un switch. En una red tradicional, las reglas de protocolo integradas al firmware propietario del switch le indican a dónde mover el paquete. El switch envía cada paquete al mismo destino por la misma trayectoria (y trata a todos los paquetes de la misma manera). En una SDN, un administrador de red puede manejar el tráfico de red de forma centralizada sin tener que configurar switches de manera individuales. El administrador puede cambiar cualquier regla de los switches cuando sea necesario, dando o quitando priori-

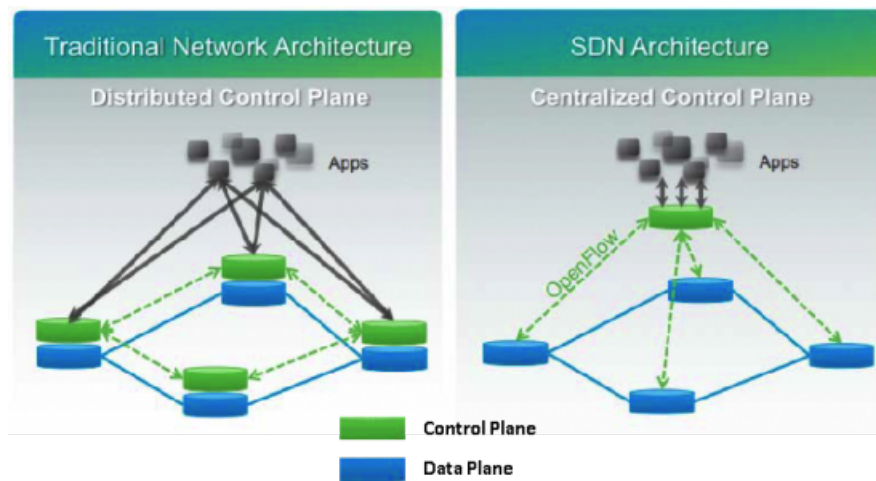


Figura 1: Comparación arquitecturas [4]

dad, o hasta bloqueando tipos específicos de paquetes con un nivel de control muy detallado. En definitiva con SDN se virtualiza la red independizándola de la infraestructura física subyacente y creando redes lógicas con objeto de cumplir los requisitos de rendimiento, escalabilidad y agilidad necesarios en modelos de cloud computing.[5]

5.2. Arquitectura

En esta sección se presenta la arquitectura de las redes definidas por software.

5.2.1. Capas de la arquitectura

Como se comentó anteriormente la arquitectura SDN, separa el plano de control, del plano de datos. Por esto existe una capa de infraestructura, la cual contiene a los dispositivos de red que son los encargados de realizar la conmutación y el encaminamiento de los paquetes. Además de la capa de infraestructura, se tiene la capa de control y la capa de aplicaciones, un esquema de la arquitectura se puede observar en la figura 2.

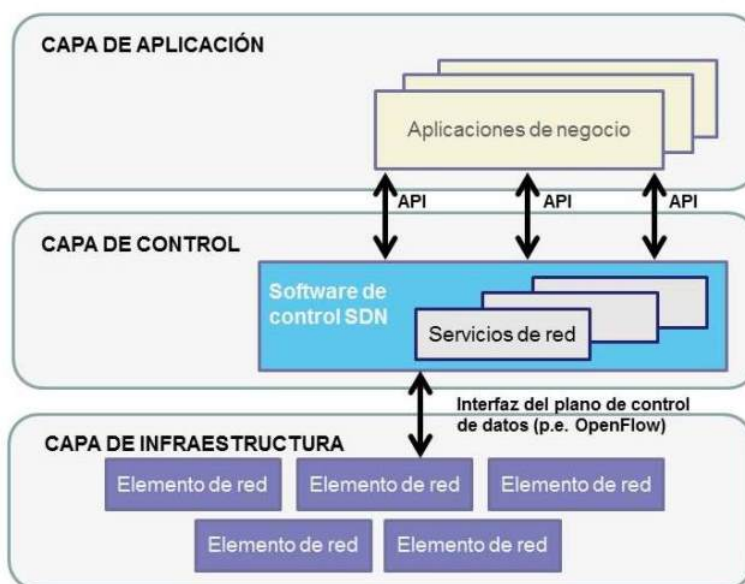


Figura 2: Arquitectura SDN [6]

- **Capa de infraestructura:** En la capa de infraestructura se encuentran los dispositivos de red, son los encargados de la conmutación de los paquetes.
- **Capa de control:** En la capa de control, se encuentra el controlador SDN, el cual, es una entidad de software que tiene control exclusivo sobre el conjunto abstracto de recursos de plano de control, es decir, es la entidad que controla y configura los dispositivos de red para dirigir correctamente los flujos de tráfico. El controlador SDN elimina la inteligencia de conmutación y encaminamiento de datos de los dispositivos, la cual en las redes tradicionales estaba en dichos dispositivos, de esta forma el controlador SDN es quien toma esas decisiones y selecciona el mejor camino para

el tráfico. La capa de control se comunica con la capa de infraestructura a través de una API, llamada southbound. Mediante esta interfaz, el controlador SDN, logra interactuar con los dispositivos de red. Logrando así, descubrir la topología de red y realizar cambios en los flujos de tráfico para satisfacer las demandas que surgen en tiempo real. Una especificación de la interfaz southbound, es el protocolo OpenFlow. Pero existen otras como Cisco OpFlex, Extensible Messaging and Presence Protocol (XMPP), Network Configuration Protocol (Netconf), OpenStack de Rackspace y la NASA.

- Capa de aplicación: A través de la capa de aplicaciones, se puede programar la capa de control, esto se realiza utilizando una API northbound. Por lo tanto, la capa de control y la capa de aplicación se comunican a través de la API northbound. Esta característica es muy importante para SDN, ya que al añadir esta tercera capa al diseño, es posible programar la red de la misma manera que las aplicaciones software, de modo que se pueden conseguir implementaciones que se integren ágilmente con el resto de componentes IT, por ejemplo, haciendo que la red se adapte automáticamente a los movimientos de máquinas virtuales entre Data Centers, permitiendo grandes mejoras frente a los diseños tradicionales.

5.2.2. OpenFlow

Se puede definir OpenFlow como un protocolo, un mecanismo que permite a un servidor (controlador SDN) gestionar e interactuar con los dispositivos de red. Dicho de otra forma, es un protocolo que comunica los dispositivos OpenFlow con el controlador SDN. OpenFlow es open source y comenzó a desarrollarse en 2007, surgiendo como resultado de la colaboración de los sectores académico y empresarial. Fueron las universidades de Stanford y California en Berkeley quienes llevaron las riendas en primera instancia. En la actualidad, la Open Networking Foundation (ONF) se encarga de la definición del estándar, esta fundación, es un consorcio industrial que está a cargo de apoyar activamente a los avances de la SDN y la estandarización de OpenFlow. Este consorcio es una asociación de empresas como Google, Microsoft, Facebook y Yahoo con Verizon y Deutsche Telekom. Ha sido muy implementado por los fabricantes de dispositivos de red, lo que lo convierte en un referente de interfaz southbound. Por este motivo, nos tomamos un punto aparte para su estudio. La versión actual del protocolo OpenFlow es la 1.5.1 (Marzo 2015).

5.2.2.1 Switch OpenFlow

Para que el controlador pueda comunicarse con los dispositivos de la capa de infraestructura, estos deben admitir API Southbound (OpenFlow), llamaremos a estos dispositivos switch OpenFlow. Un switch OpenFlow se compone de un conjunto de tablas de flujo (las cuales indican cómo procesar la información que llega al dispositivo), un canal seguro, el cual es el encargado de comunicar el dispositivo con el controlador SDN, mediante el protocolo OpenFlow. El canal

OpenFlow es usualmente encriptado utilizando TLS (Transport Layer Security), pero este canal puede correr directamente en TCP (Transmission Control Protocol). También se compone de una group table, la cual contiene acciones que afectan a un grupo de flujos y una meter table, la cual consiste en una serie de entradas que definen un conjunto de métricas por flujo que facilitan el control de la tasa de paquetes asignadas a ellas. Permiten implementar operaciones simples de calidad de servicio, como pueden ser limitaciones en el tráfico, o pueden ser combinadas con colas para implementar configuraciones de calidad de servicio más complejas como DiffServ. En la figura 3 se puede observar la organización de estos componentes en el switch OpenFlow.

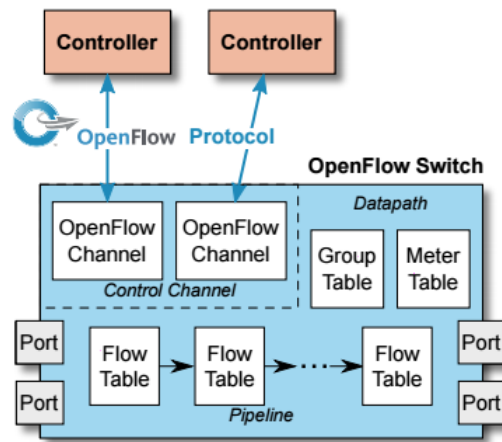


Figura 3: Componentes OpenFlow Switch [7]

OpenFlow utiliza las tablas de flujos que contienen los routers y switches Ethernet modernos, estas tablas son generalmente distintas en dispositivos de distintos fabricantes, pero contienen un conjunto de funcionalidades en común. Las tablas de flujos contienen una serie de entradas, con una acción asociada a cada una de estas entradas, de esta forma el switch sabe cómo manejar los distintos flujos de entrada.

Los campos de una entrada en la tabla de flujos, son los siguientes:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags
--------------	----------	----------	--------------	----------	--------	-------

Figura 4: Campos entrada tabla de flujos [7]

- **Match Fields:** Definen un cierto flujo mediante el establecimiento de un conjunto de campos que se comparan con los paquetes recibidos en el switch.
- **Counters:** Muestran estadísticas sobre el flujo correspondiente a una cierta entrada, como por ejemplo el número de paquetes o bytes identificados.
- **Instructions:** Son acciones a realizar cuando se encuentra una coincidencia entre un paquete y una tabla de flujos (por ejemplo: modificar el conjunto de acciones o el procesamiento del pipeline).
- **Priority:** Orden de preferencia de “matching” del flujo de entrada.
- **Cookie:** Es una información añadida por el controlador que permite la identificación de la entrada de flujo.
- **Timeouts:** Permiten la eliminación de una cierta entrada de forma automática transcurrido un cierto periodo de tiempo.
- **Flags:** Estas flags alteran la forma en que se manejan los flujos de entrada.

Como se observa en la figura 5, el flujo de entrada pasa por varias tablas de flujos.

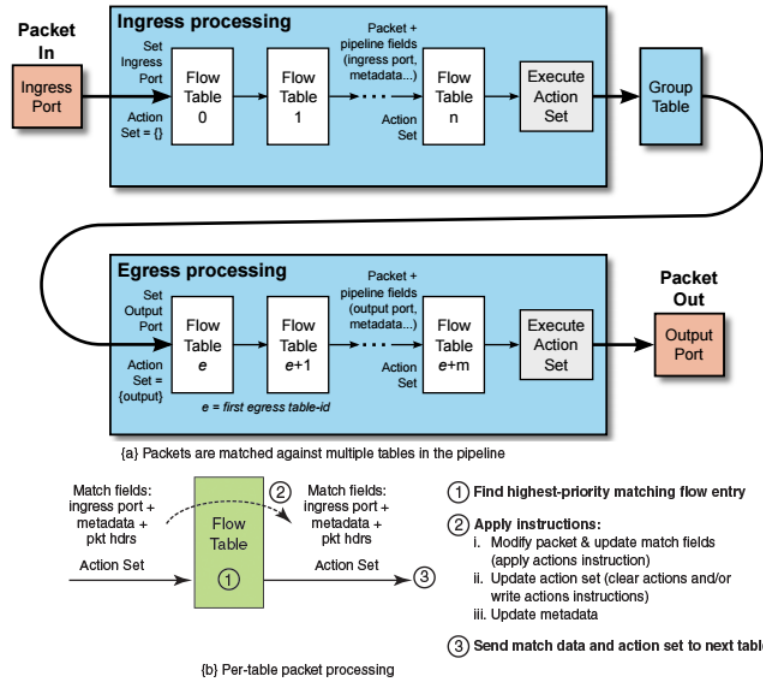


Figura 5: Tratamiento de flujos entrantes [7]

En el pipeline de openflow se realiza el proceso de decisión, donde, cada switch tiene una o varias tablas, en las cuales se realiza el siguiente proceso:

1. En primer lugar se busca el paquete entrante coincidente con la mayor prioridad.
2. Una vez detectado se aplican las distintas instrucciones.
 - a. Modificación del paquete y actualización de los campos coincidentes.
 - b. Actualización del conjunto de acciones.
 - c. Actualización del metadata.
3. Por último, se envía el dato coincidente a la siguiente tabla junto con el conjunto de acciones.

En caso de no hallar coincidencias en las tablas de flujo, el flujo de entrada será devuelto al Controlador SDN, que indicará al switch qué hacer con él. Como se mencionó anteriormente, cada entrada en la tabla de flujos tiene una acción asociada. En OpenFlow hay tres tipos de acciones básicas que todos los dispositivos deben soportar:

- Reenvío de los paquetes de un flujo particular a un determinado puerto (o conjunto de puertos). Esto permite que los paquetes sean enrutados a través de la red.

- Encapsular y reenviar los paquetes de un flujo a un controlador. El paquete se entrega al canal seguro, donde se encapsula y se envía a un controlador. Normalmente se utiliza para el primer paquete en un flujo nuevo, para que el controlador pueda decidir si el flujo debe ser añadido a la tabla de flujos. Alternativamente, se podría utilizar como sniffer para transmitir todos los paquetes a un controlador para su procesamiento.
- Eliminar paquetes de un flujo. De esta forma, el controlador SDN podría actuar como un firewall, y bloquear paquetes sospechosos.

5.2.3. Controladores SDN

En el centro de la arquitectura de las SDN se encuentra la capa de control con el controlador SDN, que es quien gestiona los flujos de entrada. El controlador es el cerebro de la red, este controla todas las comunicaciones entre las aplicaciones y los dispositivos. El controlador SDN se encarga de traducir las necesidades o requisitos de la capa Aplicación a los elementos de red, y de proporcionar información relevante a las aplicaciones SDN, pudiendo incluir estadísticas y eventos. Existen múltiples controladores SDN OpenSource, como por ejemplo: NOX, POX, Ryu, Beacon, Floodlight, FlowER, Maestro, NodeFlow, OpenContrail, ONOS, OpenDaylight, Trema. Algunos de estos han sido desarrollados por la comunidad, y otros que han sido desarrollados por empresas que han liberado el código. También existen varios controladores comerciales, como por ejemplo: Big Switch, Cisco XNC, HP VAN SDN Controller Software, y los de las empresas NEC y Juniper. Como mencionamos anteriormente OpenFlow no es la única alternativa para una southbound API, pero si es la más fácil de explotar y sobre la que más información hay disponible en este momento para abordar las redes definidas por software, por lo que elegir un controlador con soporte OpenFlow ofrece muchas ventajas. El rendimiento es una de las características importantes a evaluar en un controlador, la cual se evalúa mediante el establecimiento de flujos. La medida que se utiliza en este caso es el tiempo de configuración de un flujo, y el número de flujos por segundo que el controlador puede configurar. Se debe garantizar que el rendimiento del controlador no produzca un cuello de botella en la red implantada. La configuración de flujos puede ser de dos formas: Proactiva o Reactiva (esto se define cuando se diseña la sdn).

Un controlador SDN debe manejar un gran número de switches, por lo que la escalabilidad a la hora de seleccionar un controlador es muy importante. Dependiendo de las necesidades actuales y futuras de la red, pero un solo controlador SDN debería ser capaz de administrar al menos 100 switches sin problemas de rendimiento. Algunas de las otras características que se deben tener en cuenta a la hora de evaluar un controlador son: Soporte a la virtualización de la red, soporte a ejecución en multiplataforma, seguridad en la red (soportar autenticación y autorización), entre otros. Otra diferenciación que tienen los controladores SDN, es el lenguaje en el que fueron programados, algunos de los lenguajes utilizados son: C/C++, Java, Python o Ruby entre otros.

6. Estado del Arte - CDN

El primer objetivo de este trabajo es resumir el estado del arte en redes de distribución de contenido, por lo que en esta sección se realiza dicho estudio.

6.1. ¿Que es CDN? Conceptos Básicos

Una red de entrega de contenidos (CDN) es un conjunto de servidores (denominados surrogates o réplicas) que contienen copias de datos, colocados en varios puntos de una red con el fin de maximizar el ancho de banda para el acceso a los datos de la red, para realizar una entrega transparente, eficaz, escalable y rápida de la información. La idea es que un cliente acceda a una copia de la información, ubicada en un servidor cercano geográficamente a él. Si no se cuenta con una red de entrega de contenidos, todos los clientes accederán al mismo servidor central para obtener la información. Lo que puede generar un cuello de botella en el acceso a esa información. La primera definición de CDN nació en un principio, de los documentos de Akamai [8] a finales de los años 90. Actualmente el uso de CDN está muy difundido ya que es utilizado por grandes empresas de transmisión de vídeo tanto en vivo como bajo demanda, como pueden ser Netflix o Youtube, y proveen al usuario de beneficios como:

- Disminución de los tiempos de respuesta en consultas ya que, al buscar el servidor más cercano para descargar el contenido, disminuye el número de saltos que deben dar los datos entre nodos.
- Mayor escalabilidad al permitir la conexión de un gran número de usuarios simultáneamente.
- Menor costo de distribución.
- Seguridad: protección del contenido a entregar a un tercero.
- Fiabilidad, rendimiento y mejor capacidad de respuesta pues la probabilidad de perder información o de indisponibilidad es menor al ser una red tolerante a fallas.
- Obtención de estadísticas de uso de los usuarios.
- Proyección de crecimiento al tener una alta tendencia de uso futuro.

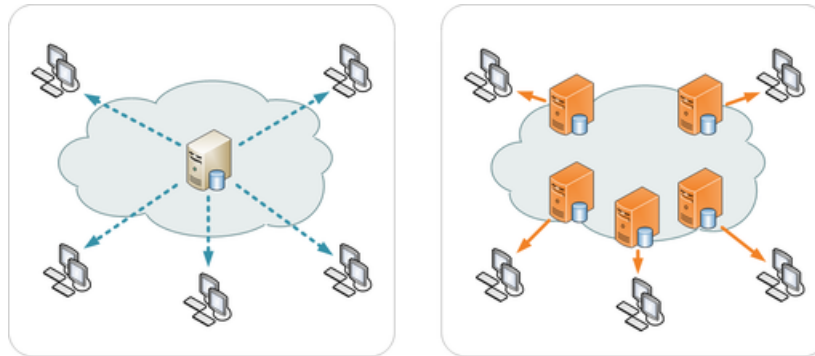


Figura 6: Distribución tradicional vs CDN [9]

Las funcionalidades básicas de una CDN son:

- **Mecanismo de redirección:** Para dirigir una solicitud al surrogate más cercano, utilizando mecanismos para evitar la congestión.
- **Servicios de distribución de contenido:** Para replicar o cachear contenido desde un servidor origen a los surrogates dispersos en Internet.
- **Servicios de negociación de contenido:** Para cubrir los requerimientos concretos de usuarios específicos (o grupos de usuarios).
- **Servicios de gestión:** Para administrar los componentes de red, gestionar la contabilidad y monitorizar y reportar el uso y acceso al contenido.

La Figura 7 muestra un esquema básico CDN donde los surrogates se encuentran repartidos en los extremos de las redes de Internet (dispersas por todo el mundo) a través de los cuales se conectan los usuarios finales. La CDN es la encargada de distribuir el contenido a cada uno de estos surrogates remotos, de tal forma que el contenido final llega al usuario de forma fiable y en unos tiempos aceptables. El contenido puede ser replicado bajo demanda, cuando los usuarios lo solicitan, o bien con antelación, dependiendo de la estimación del tráfico y contenido demandado. Los usuarios finales, de forma transparente, terminan contactando con un surrogate en lugar del servidor origen, pero el contenido es servido con las mismas garantías y con un retardo menor.

La infraestructura de una CDN proporciona típicamente los siguientes servicios y funcionalidades: almacenamiento y gestión del contenido, distribución de contenido entre surrogates remotos, gestión de caching, distribución de contenido estático, dinámico y streaming, soluciones de backup y recuperación ante fallos, monitorización, medidas de rendimientos y reporte de uso.

De lo presentado anteriormente se desprende que una CDN está dirigida esencialmente a los proveedores de contenido o clientes que desean asegurar la QoS a sus usuarios finales al acceder a su contenido Web. Por lo que una CDN se

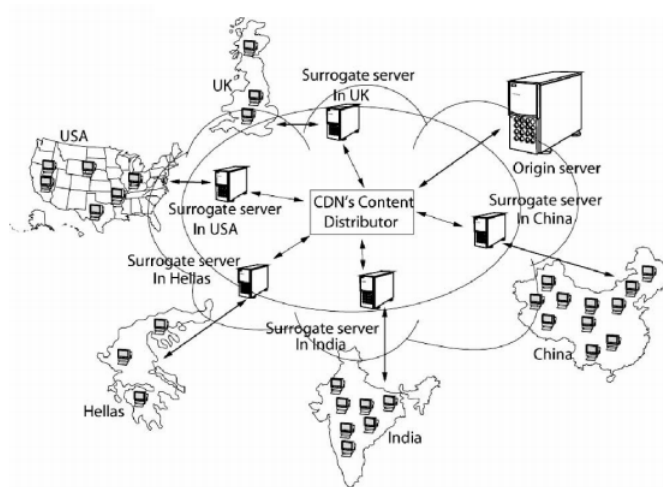


Figura 7: Esquema básico CDN [10]

debe enfocar en los siguientes objetivos: escalabilidad, seguridad, sensibilidad, confiabilidad y performance [11].

- **Escalabilidad:** Parte de la importancia de las CDN son sus prestaciones en cuanto a alta velocidad y gestión de enrutamiento de grandes volúmenes de información.
- **Seguridad:** Debe ofrecer también mitigación de posibles ataques DDoS. Un ataque de denegación de servicio, también llamado ataque DDoS (por sus siglas en inglés), es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad con la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema atacado.
- **Sensibilidad:** La sensibilidad de una red se mide por la cantidad de tiempo que le toma a la misma gestionar cambios en su topología lógica por lo que debe contar con mecanismos de rápida propagación.
- **Confiabilidad:** Una CDN debe proveer alta disponibilidad, con un diseño sin puntos únicos de falla, cumpliendo Acuerdos de Nivel de Servicio (SLA, por sus siglas en inglés) entre 99.9 y 99.999%. Para lograrlo se elaboran procesos de mantenimiento de hardware y software, además de utilizar conexiones con múltiples carriers.
- **Performance:** Una CDN busca, entre otras cosas, minimizar la latencia lo que significa optimizar la conectividad colocando los surrogates en tantas intersecciones de la red principal como sea posible o, mejorar las facilidades

físicas, instalándolos en centros de datos importantes donde los backbones de proveedores vecinos y nuestro proveedor de CDN hayan establecido acuerdos de vecindad con otras CDN y carriers disminuyendo los tiempos de ida y vuelta de los datos.

Algunas de las principales CDN comerciales son: Akamai Technologies [8], EdgeStream [12], Limelight Networks [13] y Mirror Image [14]. Mientras que algunas CDN académicas son: CodeeN [15], Coral [16], Globule [17], FCAN (Flash Crowd Alleviation Network) [18] y COMODIN [19]

6.2. Evolución Historica

En los últimos 20 años el servicio web ha evolucionado desde una simple aplicación de Internet, que era utilizada por científicos e investigadores hasta convertirse en un fenómeno comercial del cual hoy en día son dependientes muchas empresas millonarias. Sin embargo este gran crecimiento en la red, la rápida evolución de las velocidades de conexión, el incremento en la complejidad de los sistemas y contenidos de hoy en día representan un gran desafío a la hora de gestionar y entregar contenido a los usuarios. Cualquier reducción de calidad del servicio, o un elevado retardo en el acceso de contenidos multimedia produce frustración en el usuario y provocar un abandono del servicio o del sitio web en particular. Esto es lo que se trata de evitar en los nuevos mercados web, donde se incorporan aplicaciones de red nuevas, herramientas software así como nuevos tipos de servicios de red. Cuando todas estas tecnologías se usan conjuntamente se crea un nuevo tipo de red denominada content network. Por lo que existen varias redes de contenido que intentan abordar el problema de rendimiento mediante el uso de diferentes mecanismos para mejorar la QoS:

- Un enfoque inicial es modificar la arquitectura Web tradicional mejorando el hardware del servidor web añadiendo un procesador de alta velocidad, mayor cantidad de memoria y mayor espacio de disco, o incluso un sistema con varios procesadores. Este enfoque inicial no es flexible, ya que pequeñas mejoras no son posibles y en algún momento, el servidor podría tener que ser reemplazado.
- La implementación de proxy caché por parte de un ISP puede ser beneficioso para los usuarios con poco ancho de banda que acceden a Internet, ya que para mejorar el rendimiento y reducir la utilización del ancho de banda, los proxies caché se despliegan cerca de los usuarios. Los proxies caché también pueden estar equipados con tecnologías para detectar un fallo del servidor y maximizar el uso eficiente de los recursos proxy caché. Los usuarios a menudo configuran sus navegadores para enviar su solicitud web a través de estos cachés en lugar de enviarlos directamente a los servidores de origen. Cuando esta configuración se realiza correctamente, toda la sesión de navegación del usuario pasa por un proxy caché específico. Por lo tanto, las cachés contienen el contenido más popular visto por todos los usuarios de los proxies, los cuales están almacenados en caché.

- Un proveedor también puede implementar diferentes niveles de cachés locales, regionales e internacionales distribuidos geográficamente. Dicha disposición se denomina almacenamiento en caché jerárquico. Esto puede proporcionar mejoras de rendimiento adicionales y ahorros de ancho de banda. El establecimiento de granjas de servidores es una solución más escalable que ha estado en uso generalizado durante varios años. Una granja de servidores comprende varios servidores web, los cuales responden las solicitudes del mismo sitio web. También hace uso de un conmutador de Capa 4-7 (conmutación inteligente basada en información tal como: URL solicitada, tipo de contenido y nombre de usuario), conmutador Web o El conmutador de contenido que examina las solicitudes de contenido y las distribuye entre el grupo de servidores. Una granja de servidores también se puede construir con surrogates en lugar de un conmutador. Este enfoque es más flexible y muestra una mejor escalabilidad. Además, proporciona el beneficio inherente de la tolerancia a fallos. La implementación y el crecimiento de las granjas de servidores avanza con la actualización de vínculos de red que conecta los sitios web a Internet.
- Aunque las granjas de servidores y el almacenamiento en caché jerárquico a través de los proxies caché son técnicas útiles para solucionar el problema de rendimiento de la web, tienen limitaciones. En el primer caso, puesto que los servidores se despliegan cerca del servidor de origen, hacen poco para mejorar el rendimiento de la red debido a la congestión de la misma. Procesar caché puede ser beneficioso en este caso. Pero los objetos de caché se basan en las demandas del cliente. Esto puede obligar a los proveedores de contenido con una fuente de contenido popular a invertir en granjas de servidores grandes, balanceo de carga y conexiones de alto ancho de banda para mantenerse al día con la demanda. Para abordar estas limitaciones, otro tipo de redes de contenido se ha desplegado a finales de los años noventa. Esto se denomina red de entrega de contenido, que es un sistema de computadoras conectadas en red a través de Internet para cooperar de forma transparente para entregar contenido a los usuarios finales.

Con la introducción de CDN, los proveedores de contenido comenzaron a poner sus sitios web bajo una CDN. Pronto se dieron cuenta de su utilidad al recibir una mayor fiabilidad y escalabilidad sin necesidad de mantener una infraestructura cara. Es por esta razón, que poco a poco se fueron dando varias iniciativas, para emprender aún más el desarrollo de la infraestructura CDN tal como la conocemos hoy en día. Con el pasar de los años, varios investigadores como por ejemplo los del MIT quienes en su afán de combatir el problema del flash-crowd y entregar una solución de fondo al grave problema de caching que sufrieron varios sitios web que fueron inundados por peticiones durante algún evento de gran proporción, crearon lo que hoy en día es la red CDN más grande conocida como Akamai Technologies y así varios grupos más de científicos han venido desarrollando un conjunto de algoritmos de vanguardia para el enrutamiento inteligente y la replicación del contenido a través de una amplia red de servidores distribuidos por todo el mundo, logrando así que varias empresas se

convirtieran en especialistas en proveer una rápida y confiable distribución de contenido, logrando así que CDN sea un enorme mercado para generar grandes ingresos.

Existen tres generaciones de CDNs [20]

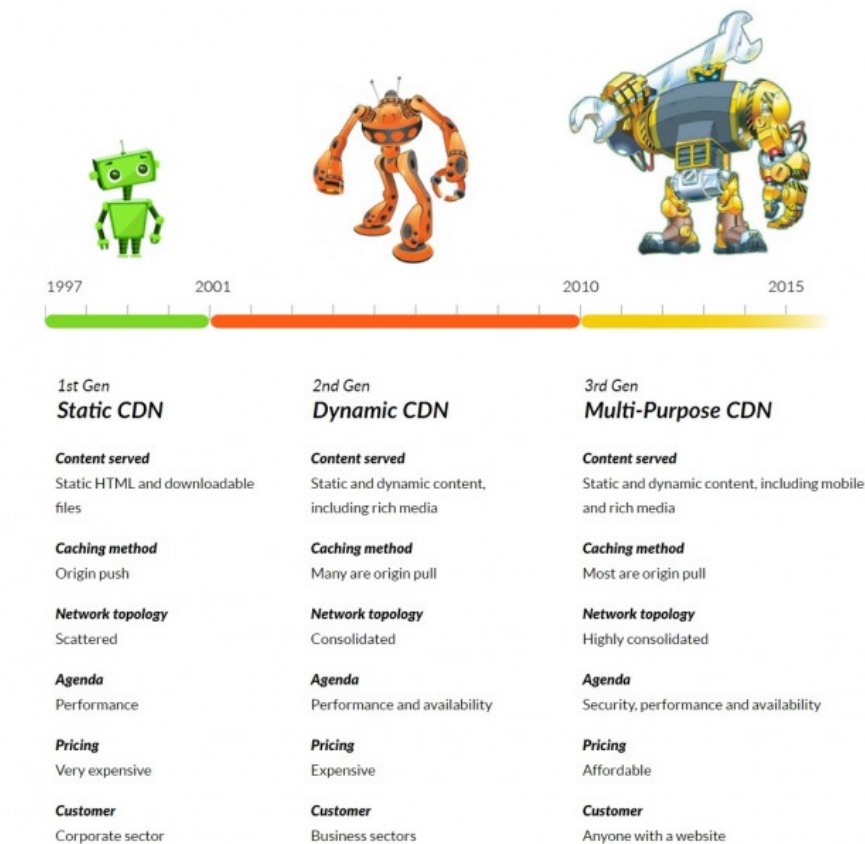


Figura 8: Generaciones CDN [11]

6.2.1. Primera Generación

La primera generación de CDN surgió hace más de una década para que los sitios de Internet puedan mantener el ritmo ante el crecimiento desmedido del uso de Internet y mayor ancho de banda. Esto llevó al desarrollo del concepto de “web acceleration” y eran CDNs que almacenaban frecuentemente peticiones de contenido en servidores distribuidos lo más cerca de los puntos de consumo, para disminuir la demanda y reducir la latencia. Dentro de esta generación ya se introdujeron conceptos claves, como el despliegue del intelligent routing y el ed-

ge computation, y llevado al crecimiento de otro de los grandes actores globales en CDN conocido como Limelight, que desarrolló en gran medida un fenómeno estadounidense debido al enorme liderazgo de ese país en desarrollos sobre Internet. En esta generación a más de realizar solamente procesos de caching, realmente se introdujo la capacidad de manejar un caching del contenido más veloz y dinámico, así como el del material estático tradicional mediante técnicas de “pushing” sobre los objetos con datos pesados para que sean distribuidos en tiempo real a los diferentes cachés. Aquí se utilizaron técnicas emergentes como la desarrollada por las herramientas de Linux basadas en Squid, el cual basó su funcionamiento en el no tan sofisticado proxy caching que se conoce hoy en día, para automatizar el frecuente proceso de descarga de páginas web de los servidores distribuidos. Principalmente, la primera generación de CDN se centró sobre el uso de los documentos Web estáticos o dinámicos.

6.2.2. Segunda Generación

Esta generación de CDN empezó a surgir cuando el vídeo comenzó a ser integrado dentro de los sitios web, eso sí, en un principio el contenido solamente era pre-grabado y entregado bajo demanda. En cierto sentido, esto significaba que el vídeo era solamente un gran archivo de datos. En el uso del contenido pre-grabado no se conocía en detalle si el usuario deseaba dejar de ver el archivo de video en cualquier momento. Por lo tanto, cuando surgió la necesidad y la demanda de manipular el video, fue cuando se requirió del streaming, y esto estimuló el desarrollo de varios CDNs optimizados para la entrega de este servicio, como algunos de los grandes referentes dentro de los servicios over-the-top y que lo son Hulu y el popular Netflix, los cuales están desarrollando su propia infraestructura CDN, pero que en su gran mayoría aún continúan utilizando la infraestructura proporcionada por un proveedor especializado en CDN. Luego llegó la creciente demanda de transmisión de video en vivo, y en un principio parecía que este sería el final del camino para el uso del caching. En definitiva, se concluyó que el tráfico de video en vivo no podía ser almacenado en caché como se lo hizo previamente para el contenido grabado, y que por lo tanto, la infraestructura de la CDN tenía que ser modificada para dar cabida a las elevadas tasas de transferencia en la transmisión punto a punto entre la fuente de origen del contenido y el usuario final. El problema fue que los costos involucrados en la construcción y el mantenimiento de la transmisión en vivo para eventos populares serían extremadamente costosos, así como técnicamente exigentes. Esta segunda generación de CDN se centró en el Video on-Demand (VoD), news on-demand, la transmisión de audio y video con alta interactividad para el usuario, así como también dedicaron recursos de investigación sobre los métodos de entrega de contenido para los dispositivos móviles.

6.2.3. Tercera Generación

Esta siguiente generación de CDN se ha venido anticipando durante el transcurso de estos últimos años y trae gran cantidad de desarrollo e investigación, los expertos asumen que podría ser conocida como community-based CDN's y que permitiría un poco más de control para el usuario promedio, por lo cual estos tienen que ser investigados antes de que Wireless Mesh Network (WMN) se convierta en una parte completamente integral de las redes de contenido. Se prevé que las redes multihomed integradas sean funcionales a finales de la década, basado en el progreso que ha traído la investigación en WMN, la selección de red y otras cuestiones pendientes de investigación relacionadas, teniendo en cuenta que solo las infraestructuras altamente heterogéneas basadas en diferentes tipos de redes serán capaces de hacer frente a esto y proporcionar el apoyo necesario. Los operadores se están alejando paulatinamente de las redes cerradas como el IPTV, ya que todos los nuevos conceptos que trae esta generación, muestran una clara oportunidad de reutilizar la infraestructura existente; debido a la fuerte presión para agregar más y más canales, hace que esto siga creciendo junto con el concepto de TV-Everywhere (incluida la movilidad de dispositivos), por lo que el tráfico unicast aumentará inevitablemente. Como por ejemplo, el concepto de "Multicast To The Home" que está siendo desarrollado por Time Warner Cable (TWC) que es una de las operadoras más avanzadas en servicio de cable, a través del proceso de Adaptive Bit-Rate (ABR) sobre un Single Program Transport Stream (SPTS) para tomar ventaja de la infraestructura multicast existente, fortalecerla y obtener beneficios. Un aspecto importante de la generación CDN 3.0 es que será el surgimiento de las federated CDN's.

6.3. Arquitectura

Dada la diversidad de contenidos que se entregan dentro de CDN, se pueden adoptar diversas arquitecturas y tecnologías para diseñarla y desarrollarla, lo que requiere de un conjunto de servicios de soporte y ciertas capacidades, que colaboren junto con la distribución que realizan los surrogates en el borde de la red. Con el afán de que el servicio sea eficaz para un número significativo de usuarios y un área considerablemente amplia, los surrogates deben ser desplegados en miles de redes y dentro de diferentes lugares separadas geográficamente. Lograr un buen rendimiento y una confiabilidad adecuada dependerá de la granularidad con que se distribuyan los surrogates.

Con lo anteriormente dicho, se puede definir la arquitectura general de una CDN, como la relación de siete componentes principales [21]: clientes, surrogates, servidor origen, sistema de tarificación, sistema de encaminamiento, sistema de distribución de contenidos y sistema de contabilidad.

A continuación se presentan los componentes mencionados:

- **Clientes:** Son los que se interesan en acceder a la información distribuida por el servidor origen a través de la CDN.
- **Surrogates:** Replican toda o parte de la contenido del servidor origen.
- **Servidor origen:** Es el propietario de los contenidos.
- **Sistema de tarificación:** Son servidores que contabilizan y cuadran los contenidos entre los clientes, CDN y los proveedores de contenidos.
- **Sistema de encaminamiento:** Es una de las partes más importantes de la CDN, se encarga de recibir las peticiones de los clientes, procesarlas y enviarlas al servidor de caché más adecuado (según un algoritmo de encaminamiento) de una manera suficientemente rápida para que los clientes perciben una latencia menor.
- **Sistema de distribución de contenidos:** Se encarga de trasladar los contenidos desde los servidores origen a los servidores de caché.
- **Sistema de contabilidad:** Se encarga de controlar y resumir la información de la monitorización en estadísticas.

En la figura 9 se observan las relaciones entre componentes presentados.

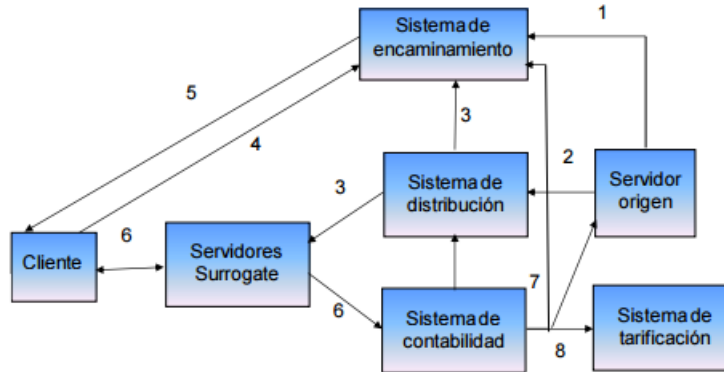


Figura 9: Relacion Componentes CDN [22]

Estas relaciones se definen de la siguiente manera:

- El servidor origen delega su espacio de nombres (URIs) al sistema de encaminamiento (1).
- El servidor origen publica contenido web que debe ser distribuido por la CDN a través del sistema de distribución (2).
- El sistema de distribución traslada el contenido a los surrogates. Adicionalmente, interacciona con el sistema de encaminamiento para colaborar en la selección del surrogate más adecuado (3).
- El cliente solicita un contenido de lo que él percibe como servidor origen, pero la solicitud es realmente redirigida al sistema de encaminamiento (4).
- El sistema de encaminamiento encamina la solicitud a un surrogate óptimo de la CDN, utilizando su algoritmo de encaminamiento. (5).
- El surrogate seleccionado sirve el contenido al cliente, e interactúa con el sistema de contabilidad (6).
- El sistema de contabilidad procesa y sintetiza la información obtenida en estadísticas y detalles de uso por contenido, facilitándoselos al servidor origen y al sistema de tarificación o facturación. Las estadísticas también se mandan a modo de feedback al sistema de encaminamiento (7).
- El sistema de tarificación emplea los registros detallados de contenido para liquidar cuentas con cada una de las partes involucradas en la distribución del contenido. Téngase en cuenta que el sistema de tarificación tiene sentido en las CDNs comerciales (8).

En 2008 Buyya define una clasificación de CDN [10], tomando en cuenta sus componentes y funcionalidades principales, la cual se muestra en la figura 10:

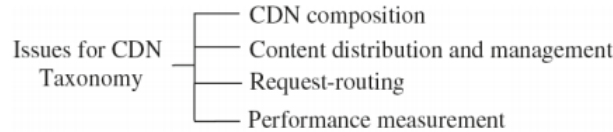


Figura 10: Taxonomía CDN [10]

Los cuatro aspectos que se tienen en cuenta en la clasificación son:

- **Composición:** Hace referencia a los aspectos de organización y arquitectura, clasificando las CDNs en base a sus atributos estructurales.
- **Distribución de contenido y gestión:** Se refiere a la ubicación de servidores, selección y distribución de contenido, externalización de contenido y organización de cachés y réplicas.
- **Encaminamiento y redirección:** Hace referencia a los mecanismos y algoritmos de redirección de clientes para encaminar las peticiones hacia los surrogates más cercanos.
- **Rendimiento:** Se refiere a las metodologías de evaluación de prestaciones de una CDN.

Para los objetivos de este trabajo, los componentes de la arquitectura más importantes son el sistema de encaminamiento y el sistema de distribución, ya que estos sistemas tendrán que comunicarse con el control de la SDN para redirigir las peticiones de los clientes y para mover contenidos en la red. Estos sistemas se estudiarán con más detalle en los siguientes puntos.

6.3.1. Sistema de distribución CDN

Este sistema se encarga de mover el contenido desde el servidor de origen hacia los surrogates y asegurar la consistencia de información del contenido. La distribución y la administración del contenido son muy importantes para todo el desempeño del sistema CDN, ya que implica la reducción del tiempo de respuesta que percibe un usuario. La distribución del contenido incluye: técnicas basadas en el posicionamiento estratégico para que un surrogate esté lo más cerca al usuario y para esto se emplea el placement of surrogates. Se tiene también el content selection y delivery, que se basa en el comportamiento del tipo y frecuencia con que se realizan las peticiones del usuario, y finalmente se tiene el método de content outsourcing, que es utilizado para decidir qué tipo de metodología de externalización se seguirá. La gestión de contenido depende en gran parte de las técnicas conocidas como cache organization, las cuales

incluye: caching techniques, cache maintenance y cache update. En la figura 11 se muestra la clasificación del sistema de distribución según Buyya.

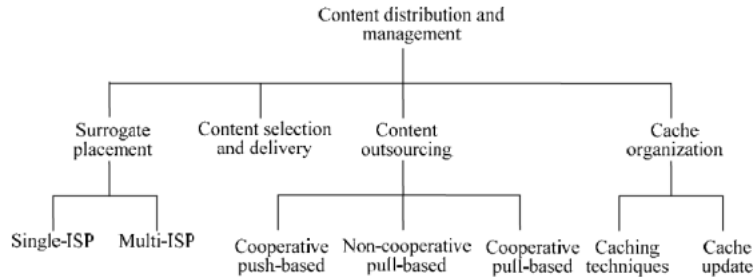


Figura 11: Clasificación sistema de distribución [10]

6.3.1.1 Ubicación de surrogates

La ubicación de los surrogates está profundamente relacionada con el proceso de distribución de contenido, por lo que se debe seleccionar cuidadosamente la mejor ubicación para cada surrogate. El objetivo en la ubicación óptima de surrogates es reducir la latencia percibida por el usuario al acceder a contenido, así como reducir el ancho de banda global en la transferencia de datos desde los surrogates a los clientes. La optimización de estas dos métricas conlleva una reducción en los costes de infraestructura y comunicación del proveedor de CDN, lo que permite ofrecer servicios de mejor calidad a un coste menor [23]. Las estrategias que se utilizan para la ubicación de surrogates se pueden observar en la figura 12.

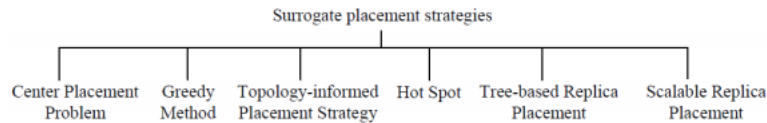


Figura 12: Estrategias ubicación surrogates [10]

6.3.1.2 Selección de Contenido y Distribución

Este aspecto representa uno de los parámetros más importantes que determinan la eficiencia en la distribución de contenido, ya que implica la reducción en el tiempo de respuesta percibido por los clientes. El contenido que se debe seleccionar para su replicación, presente en el servidor origen, se puede distribuir hacia los surrogates de forma total o parcial, como se muestra en la figura 13.

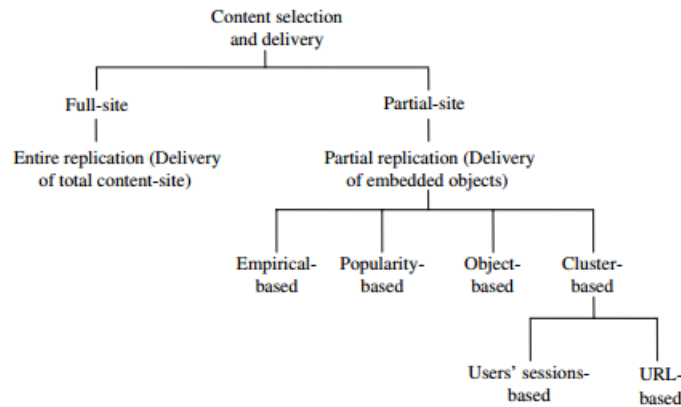


Figura 13: Clasificación distribución [10]

La selección y distribución total (Full-site) es la aproximación más sencilla donde los surrogates replican todo el contenido del servidor origen. En este esquema el proveedor de contenido configura su servicio DNS de tal forma que todas las peticiones son resueltas por un servidor de la CDN (surrogate) quien distribuye el contenido. La gran ventaja de este esquema es su sencillez. Sin embargo, dentro del diseño no representaría una solución práctica, ya que se debe considerar el crecimiento dinámico de Internet y la enorme colección que se puede formar de los objetos web. Mientras que en la selección y distribución parcial (partial-site), los surrogates realizan una especie de replicación parcial para distribuir únicamente objetos embebidos como pueden ser por ejemplo, imágenes desde la CDN, de esta manera el cambio del contenido embebido es menos frecuente, por lo que este método resulta tener un mejor rendimiento. Este a su vez se subdivide en modalidad empírica, de objetos y basada en popularidad del contenido o cluster. De esta forma, un proveedor de contenido modifica su contenido de tal forma que los enlaces a objetos específicos embebidos referencian nombres de host de los cuales el proveedor de CDN es autoritativo. Pero también se requiere de una estrategia de gestión adecuada para replicar dicho contenido web. Así, la página web HTML básica se obtiene del origen server, mientras que los objetos embebidos se obtienen de los proxy caches o servidores cache de CDN. Siendo la más compleja de las modalidades la de Cluster, que se

realiza o bien fijando el número de clústeres o fijando el diámetro máximo del clúster, lo que permite reducir el tiempo de descarga y la carga en el servidor, ya que los objetos web más populares son replicados en unidades de clústeres donde la distancia de correlación entre cada par de URLs está basada en una métrica de correlación determinada. Por otro lado, el clústering de contenido puede estar basado en la sesión de usuario o en la URL.

6.3.1.3 Externalización de Contenido

La externalización de contenido (content outsourcing), se deriva una vez que se ha seleccionado correctamente el posicionamiento de los surrogates servers y se ha seleccionado el contenido a ser distribuido dentro de la infraestructura CDN, haciendo referencia al mecanismo de distribución desde el origin server hacia los surrogates, Existen tres mecanismos de distribución:

- **Push Cooperativo:** En este sistema el contenido se envía desde el servidor origen hacia los surrogates en un primer momento y, posteriormente, estos surrogates cooperan entre ellos para reducir costes de replicación y actualizaciones. En este esquema, la CDN mantiene un mapeo entre el contenido y los surrogates, y cada una de las peticiones de clientes se encaminan al surrogate más cercano o (en última instancia) al servidor origen. Desde esta perspectiva, resulta adecuado emplear un algoritmo de tipo greedy para realizar las decisiones de replicación entre surrogates cooperativos. En cualquier caso se trata de un esquema teórico puesto que no se emplea por ninguna CDN comercial.
- **Pull no Cooperativo:** En este caso, las peticiones de los clientes son dirigidas hacia sus surrogates más cercanos. Si se produce un fallo (cache miss), los surrogates obtienen el contenido del servidor origen. La mayor parte de las CDNs comerciales (Akamai, Mirror Image) emplean este mecanismo. El principal inconveniente de este esquema es que no siempre se toma un surrogate óptimo para servir contenido. Sin embargo, la mayoría de CDNs comerciales estiman que es una solución acertada mientras el esquema push cooperativo se encuentre en una etapa experimental.
- **Pull Cooperativo:** La diferencia con el esquema anterior es que, en este caso, los surrogates cooperan entre ellos para obtener el contenido en caso de un fallo (cache miss). Mediante el empleo de un índice distribuido, los surrogates son capaces de localizar contenido en surrogates cercanos y almacenarlos en caché. Este esquema es reactivo desde el punto de vista de que un objeto es cacheado sólo cuando un cliente lo solicita. La CDN académica Coral emplea este esquema de externalización de contenido.

Dentro del enfoque de la externalización de contenido, resulta extremadamente importante dentro de los problemas que se puedan presentar en la distribución de contenido, el determinar en cuales surrogates debe replicar qué

contenido. Se pueden encontrar varios trabajos dentro de la literatura científica que demuestran la efectividad de diversas estrategias de replicación del contenido externalizado. En [24] se describen cuatro métodos heurísticos, como son aleatorio (random), basado en popularidad (popularity), greedy-single y greedy-global. Mientras que [25] presenta otras estrategias de tipo greedy donde el contenido se distribuye balanceando la carga y el tamaño de la capacidad de los surrogates. En [26] se describe un algoritmo autoajustable sin parámetros (de entrada) denominado latcdn para ubicar de forma óptima el contenido en una CDN, este algoritmo emplea la latencia de los objetos para tomar las decisiones de replicación. La latencia de un objeto se define como el retardo experimentado desde que se solicita un objeto hasta que se obtiene completamente. Una mejora sobre este algoritmo la constituye [27], que ubica el contenido en los surrogates dependiendo de la latencia y la carga de los objetos.

6.3.1.4 Organización de Caché

La organización de Caché está compuesto por las técnicas de almacenamiento en caché (caching) utilizados y la frecuencia de actualización, para asegurar la frescura de la información, la disponibilidad y la fiabilidad de los contenidos. Una adecuada gestión del contenido es fundamental para el buen rendimiento y está principalmente basado en el esquema de organización de caché empleado dentro de una CDN. Además, es posible el uso de forma integrada en la infraestructura de la CDN, a través de políticas de caching y replicación con la finalidad de introducir mejoras potenciales en la latencia percibida, hit ratio y byte hit ratio. Es más, el uso conjunto de ambas técnicas permite aumentar la robustez del sistema frente a eventos de tipo flash-crowd. Para esto, Stamos [28] propone un método heurístico no paramétrico que integra ambas técnicas de Web caching con content replication y evalúa el nivel de integración, a lo cual se lo conoce como Similarity Replication Caching (SRC). Otro ejemplo, denominado Hybrid [29] el cual combina static replication con web caching, usando un modelo analítico basado en LRU. La integración híbrida llena gradualmente los cachés de los surrogate servers con contenido estático en cada interacción, tanto tiempo como contribuya a la optimización de los tiempos de respuesta. Una clasificación de las técnicas de caching se muestra en la Figura 14, donde se aprecian las dos opciones básicas, inter-clúster e intra-clúster.

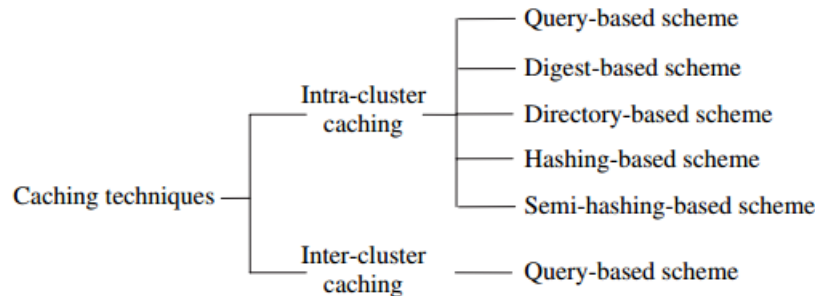


Figura 14: Clasificación técnicas de caching [10]

En un esquema basado en consultas (query), un surrogate de la CDN realiza una operación de broadcast a otros surrogates (con los que coopera) si se produce un fallo (cache miss). El problema principal de este esquema es el exceso de tráfico en la red durante la petición, así como el retardo incurrido, ya que el surrogate debe esperar hasta la última contestación de fallo para determinar cuándo ninguno de sus peers tiene el contenido solicitado. En un esquema de tipo digest, cada uno de los surrogates dispone de un resumen (también denominado digest) con la información disponible en el resto de surrogates cooperativos, que se actualiza periódicamente. Consultando este resumen (digest), un surrogate sabe a qué otro surrogate cooperativo debe solicitar el contenido. La principal desventaja de este método es precisamente la sobrecarga de tráfico en las actualizaciones para garantizar que los surrogates cooperativos disponen de la información correcta. El esquema basado en directorio (directory) no es más que una versión centralizada del esquema digest, donde un servidor centralizado almacena la información de localización de contenido de todos los surrogates cooperativos.

Los surrogates sólo notifican al servidor de directorio cuando se producen actualizaciones locales, y lo consultan siempre que se produce un fallo local (cache miss). Este mecanismo representa un cuello de botella potencial y un punto centralizado de fallo. En un esquema basado en hashing, los surrogates cooperativos emplean la misma función hash. Cada uno de los surrogates almacena un cierto contenido en base a la URL del contenido, las direcciones IP de los surrogates y la función hash. Los esquemas basados en funciones hash son los más eficientes al tener menor sobrecarga de implementación, pero no escala bien con peticiones locales y contenido multimedia ya que las peticiones pueden ser encaminadas a un surrogate lejano (en lugar de uno local y cercano). Para solventar este problema, se puede emplear un esquema semi-hashing. En este caso, un surrogate divide su espacio de almacenamiento en dos zonas: una primera zona donde se aloja el contenido más popular para sus usuarios locales (incrementando el hit

rate local), y una segunda zona donde se aloja el contenido indicado por la función hash en el modo cooperativo. En la Figura 14 se puede ver que las técnicas inter-clúster sólo disponen de esquemas basados en consultas (query). Un sistema hashing no resultaría apropiado para el caching cooperativo inter-clúster ya que los surrogates están típicamente distribuidos geográficamente y el impacto en la latencia sería considerable. Los esquemas de tipo digest y directorio tampoco son adecuados porque el tamaño de la información a almacenar puede ser considerable entre tantos surrogates dispersos en diferentes clústeres. Por ello se suele emplear un esquema basado en consultas para un caching inter-clúster. En este caso, cuando un clúster no es capaz de servir un cierto contenido, realiza una petición a un clúster cercano. Si no consigue el contenido de este clúster, puede preguntar a otro clúster vecino. Los surrogates dentro del clúster suelen emplear un esquema de tipo hashing, y el servidor representativo de ese clúster (front end) solamente contacta con el surrogate designado por la función hash. En la figura 15 se puede observar una clasificación de los procesos de actualización del caché.

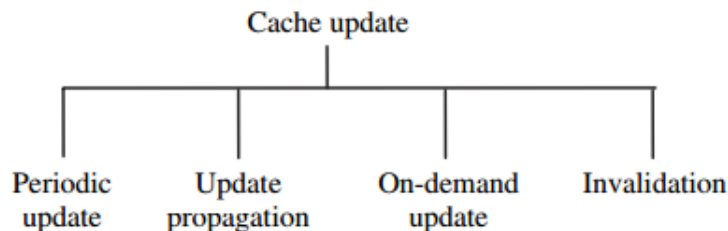


Figura 15: Clasificación actualización caché [10]

El método más común para actualizar la caché son las actualizaciones periódicas. Para garantizar la consistencia del contenido, el proveedor de contenido configura su servidor origen y proporciona instrucciones a las cachés acerca de qué contenido es cacheable, durante cuánto tiempo un contenido se considera válido, cuándo se debe interrogar al servidor origen por contenido actualizado, etc. De esta forma, las cachés se actualizan de forma periódica. Sin embargo, este mecanismo incurre en niveles significativos de tráfico generado en cada intervalo de actualización. En un esquema de propagación la actualización sólo se produce cuando hay un cambio en el contenido, y se notifica a los surrogates (pushing) desde el servidor origen. El problema en este caso es que en una situación de cambio frecuente de contenido en el servidor origen, se genera demasiado tráfico de actualización de datos hacia los surrogates. En una actualización bajo demanda sólo se actualiza el contenido en los surrogates cuando es solicitado explícitamente por éste. La desventaja de este mecanismo es el

tráfico generado entre surrogate y servidor origen para garantizar que el objeto solicitado por el cliente es el último actualizado. Finalmente, otro método de actualización lo constituye la invalidación, donde se envía un mensaje de invalidación a todas las cachés cuando se realiza un cambio en el servidor origen, y los surrogates bloquean dicho contenido hasta que no se obtiene nuevamente del servidor. Generalmente, las CDNs comerciales ofrecen a los proveedores de contenido el control sobre el grado de actualidad (freshness) y garantizan la consistencia entre todos los múltiples sites de la CDN. Por otro lado, los proveedores de contenido pueden crear sus propias políticas de caching específicas. En este caso, el proveedor de contenido debe especificar sus políticas de caching en un formato único de cada proveedor de CDN, quien propaga el conjunto de reglas a sus surrogates.

6.3.2. Sistema de encaminamiento CDN

Este sistema es responsable de enrutar las peticiones de los clientes al surrogate server más apropiado para entregar el contenido. El sistema consta básicamente de dos partes: el algoritmo de encaminamiento y el mecanismo de encaminamiento. El algoritmo de encaminamiento especifica cómo seleccionar un surrogate adecuado en respuesta a un requerimiento y se invoca cada vez que se recibe una petición. Mientras que el mecanismo de encaminamiento es una forma de notificar al cliente el surrogate que debe contactar.

Cuando un cliente necesita algún contenido de la red, se disponen de diferentes mecanismos para seleccionar el servidor al que el cliente le debe realizar la solicitud. En el caso de una arquitectura con una distribución tradicional, supone una decisión sencilla, ya que se tiene un único servidor con el contenido solicitado. En una CDN, por el contrario, los usuarios deben ser vinculados con el contenido solicitado en un lugar adecuado, el surrogate más cercano no siempre es el mejor servidor para satisfacer al cliente. Por lo que la decisión acerca de cuál es el servidor más adecuado puede tomarse en base a diferentes métricas:

- **Proximidad de red:** La distancia entre clientes y surrogates se calcula habitualmente en términos de saltos de red con una sencilla aplicación (traceroute). Sin embargo, los saltos de red no tienen en cuenta el tráfico de red y las posibles congestiones.
- **Tiempo de respuesta:** El tiempo de ida y vuelta (RTT, Round Trip Time) medido con una aplicación omnipresente (ping) aporta información sobre la latencia en la respuesta de cada servidor. Sin embargo, si se toma como única métrica en la decisión, puede conducir a decisiones de servidores desafortunadas, puesto que el tiempo de ida y vuelta es altamente variable en un entorno WAN (Internet) como es el que suele dar soporte a la infraestructura de una CDN.
- **Carga del servidor:** Existen dos mecanismos básicos de medida de la carga de los servidores; en el primero de ellos, los servidores envían información periódica a ciertos agentes, mientras que en el segundo son los

agentes quienes directamente solicitan información. Existe un compromiso entre la frecuencia de solicitud para una mayor precisión y el tráfico de red en que se incurre al mandar paquetes de solicitud de información de estado.

- **Usuarios:** Los usuarios pueden ser clasificados según el contrato establecido con el proveedor. Por ejemplo, clientes especiales que hayan pagado por un mejor servicio pueden ser redirigidos a servidores diferentes que los asociados a clientes convencionales.

En estas métricas se basan los algoritmos de encaminamiento.

6.3.2.1 Algoritmos de encaminamiento

Los algoritmos de encaminamiento, según Buyya se dividen en dos categorías:

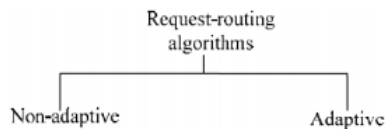


Figura 16: Clasificación algoritmos de encaminamiento [10]

- **Algoritmos No Adaptativos:** Emplean ciertas heurísticas para tomar la decisión en lugar de considerar las condiciones del sistema en tiempo real.
- **Algoritmos Adaptativos:** Consideran las condiciones en tiempo real de la red y/o los servidores para decidir el surrogate más adecuado a partir de las métricas que se mencionaron anteriormente (congestión en los enlaces, carga de los servidores, etc.)

Un estudio realizado en [22] dice que los algoritmos no adaptativos son más fáciles de implementar, dado que los algoritmos adaptativos suelen ser más complejos ya que deben adaptarse a las condiciones cambiantes de la red y/o de los surrogates en tiempo real. Los algoritmos adaptativos suelen ser más robustos que los no adaptativos, especialmente en los eventos de tipo flash-crowd, los algoritmos no adaptativos sólo son eficientes en condiciones óptimas de la red.

El algoritmo de encaminamiento no adaptativo más común y sencillo es uno de tipo round robin, que distribuye todas las peticiones a los surrogates de la CDN balanceando la carga entre ellos, asumiendo la misma capacidad para

todos los servidores. Estos algoritmos sencillos resultan eficientes para clústeres, donde los servidores están ubicados físicamente en una misma localización. Sin embargo, no son adecuados en un entorno WAN donde los servidores están ubicados en lugares distintos, y un cliente podría ser redirigido a un surrogate lejano, con una latencia percibida claramente mejorable. Por otro lado, el objetivo principal del balanceo de carga puede no quedar satisfecho ya que cada petición puede ser diferente y requerir cargas computacionales distintas.

En otro ejemplo de algoritmo no adaptativo, los surrogates se ordenan dependiendo de la carga esperada en cada uno de ellos. Dicha predicción está basada en el número de peticiones que cada servidor ha satisfecho, y considera también la distancia cliente-servidor. De esta forma, se balancea la carga entre los servidores en base a estos dos parámetros. Si bien el algoritmo resulta eficiente, la latencia percibida por el cliente se puede mejorar.

Sin embargo, en la mayoría de estos algoritmos se puede producir una redirección a un servidor sobrecargado, lo que degrada el rendimiento percibido por el usuario. En [30] se propone un algoritmo que se adapta a efectos flash-crowd, basado en el cálculo de una función hash considerando la URL del contenido. Este cálculo determina un encaminamiento eficiente a un anillo lógico de servidores caché. Algunas variaciones de este algoritmo se han empleado en contextos de caching intra-clúster y sistemas P2P de compartición de ficheros.

Globule (CDN Académica) emplea un algoritmo de encaminamiento adaptativo que selecciona la réplica más cercana a los clientes en términos de proximidad de red. La estimación de la métrica en Globule se basa en la longitud del trayecto (path) que se actualiza de manera continua. El servicio de estimación de métrica en Globule es pasivo, con lo cual no se introduce tráfico adicional en la red. Sin embargo, en [31] se muestra que dicha estimación de la métrica no es muy exacta.

Otros algoritmos de encaminamiento adaptativos están basados en la distancia cliente servidor. En estos, se tiene en cuenta o bien los logs de acceso de clientes o medidas pasivas de latencia en los servidores. De esta forma, se encamina una petición al surrogate que reporta la menor latencia. Si bien estos algoritmos resultan eficientes, se requiere el mantenimiento de una base de datos central de medidas, lo que limita la escalabilidad del sistema.

Akamai (CDN Comercial) emplea un algoritmo complejo de encaminamiento adaptativo muy enfocado a evitar flash-crowds. El algoritmo considera una serie de métricas, como son la carga de los surrogates, la fiabilidad de las cargas entre cliente y surrogates y el ancho de banda disponible en cada surrogate. El algoritmo es propietario de Akamai y los detalles tecnológicos no están disponibles.

Cisco Distributed Director ha implementado un algoritmo adaptativo de en-

rutamiento de solicitudes que tiene en cuenta una combinación de tres medidas (que tienen distintos pesos) que son: la distancia entre sistemas autónomos, de distancia al interior de los sistemas autónomos y el retardo extremo a extremo. Dicho algoritmo es costoso debido a la necesidad de implementar la monitorización en cada servidor y se genera un tráfico en la red.

6.3.2.2 Mecanismos de encaminamiento

Los mecanismos de encaminamiento indican al cliente la selección de surrogate llevada a cabo por el algoritmo de encaminamiento. Estos mecanismos pueden ser clasificados dependiendo de varios criterios, aunque típicamente se tiene en cuenta la forma de procesar la petición. La Figura 17 muestra esta clasificación que realizó Buyya.

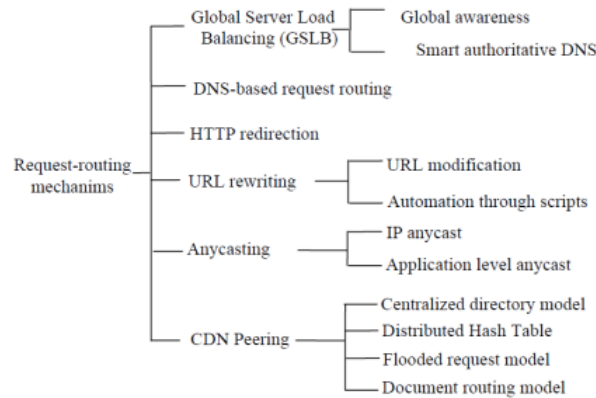


Figura 17: Clasificación mecanismos de encaminamiento [10]

6.3.2.2.1. Balanceo global de carga (GSLB)

Se dispone de un switch global y un conjunto de surrogates desplegados por todo el mundo. Existen dos capacidades especiales de los surrogates que les permiten soportar balanceo de carga global. La primera de ellas es el conocimiento global (global awareness) y la segunda un DNS autoritativo inteligente. En el balanceo de carga local, cada nodo (servidor) es consciente del estado y rendimiento de los servidores directamente conectados. En GSLB, existe un nodo (switch) que es consciente de la información disponible en los otros nodos e incluye sus IPs virtuales en su lista de servidores conocidos. En cada site de la CDN se dispone de un switch que tiene un conocimiento global del resto de switches así como de su información de rendimiento. Para hacer uso de este conocimiento global, los switches GSLB actúan a modo de servidores DNS autoritativos para ciertos dominios. La gran ventaja del mecanismo GSLB es que

cada nodo (switch) dispone de un conocimiento global del resto de nodos, por lo que es capaz de seleccionar el surrogate más adecuado para cada petición, ya sea local (conectados directamente al switch) o global (conectados a un switch remoto). Otra ventaja significativa de GSLB es que el administrador de red puede añadir capacidades GSLB sin necesidad de introducir dispositivos adicionales en la red. El mayor inconveniente de GSLB es que típicamente implica una configuración manual de los switches.

6.3.2.2.2. Redirección basado en DNS

Los servicios de distribución de contenido se basan en servidores DNS modificados capaces de realizar un mapeo entre el nombre de un surrogate y su correspondiente dirección IP. Normalmente, un dominio dispone de múltiples direcciones IP asociadas. Ante la llegada de una petición de un cliente, el servidor DNS devuelve una lista de IPs de servidores (surrogates) que disponen de una réplica del contenido solicitado. El resolver DNS en la parte del cliente debe seleccionar un surrogate de dicha lista. Para tomar esta decisión, el resolver DNS puede mandar mensajes de prueba hacia los surrogates y seleccionar aquel cuyo tiempo de respuesta es menor. También podría tomar información histórica de los clientes basada en el acceso previo a dichos surrogates. Los proveedores de servicios CDN, tanto globales (full-site) como parciales (partial-site) emplean redirección DNS. El rendimiento y la efectividad del encaminamiento basado en DNS se han examinado en varios estudios. La ventaja de este mecanismo es la transparencia, ya que los servicios y contenidos quedan referenciados por sus nombres DNS, y no por sus direcciones IP. El esquema DNS es extremadamente popular debido a su simplicidad, independencia y ubicuidad, puesto que está incorporado en el servicio de resolución de nombres, y puede ser empleado por cualquier aplicación de Internet. La gran desventaja de la redirección DNS es que se incrementa la latencia de red porque se requiere un tiempo adicional en la resolución de nombres. Los administradores de CDNs abordan este problema dividiendo el servicio DNS en dos niveles (alto y bajo nivel) para la distribución de carga. Otra limitación del sistema DNS es que en la resolución se dispone de la dirección IP del servidor DNS local, y no de la dirección IP del cliente, por lo que se puede producir una mala resolución si el cliente y el servidor DNS local no están próximos. Otro aspecto negativo en la redirección DNS es que no pueden controlar todas las peticiones debido al sistema de caching interno, tanto en el ISP como en el cliente. Desgraciadamente, en ocasiones el control puede reducirse únicamente a un 5 % de las peticiones, y resulta muy conveniente evitar el caching en el cliente ya que en caso de error el sistema puede no responder. Pese a que los mensajes de DNS incorporan un campo de tiempo de vida (TTL, Time To Live), no existe un valor claro para fijar. Un valor elevado del TTL reduce la capacidad de responder dinámicamente frente a cambios en servidores y red, mientras que un valor reducido incrementa el tráfico de red.

6.3.2.2.3. Redirección HTTP

En este esquema se toma la decisión en el servidor origen mediante el mecanismo de redirección integrado en el protocolo HTTP, de forma que el cliente es redirigido a otra URL ubicada en un surrogate. Este mecanismo es considerado poco eficiente y lento, y supone carga extra de proceso para el servidor origen. La mayor ventaja es la fina granularidad conseguida en comparación con otros mecanismos que contemplan el sitio web (website) entero como unidad de redirección.

6.3.2.2.4. URL rewriting

Pese a que la mayoría de las CDNs actuales usan DNS como esquema de enrutamiento, algunos sistemas emplean la modificación de URLs (URL rewriting o navigation hyperlink). Normalmente se emplea en la modalidad de replicación parcial donde los objetos embebidos se envían con una URL modificada. En este esquema, el servidor origen redirige a los clientes a diferentes surrogates al modificar de manera dinámica la URL de los enlaces de los objetos embebidos (la página principal la sirve el propio servidor origen). Para automatizar el proceso, las CDNs actuales proporcionan scripts especiales que parsean de forma transparente contenido web y modifican las URLs embebidas. El mecanismo de modificación de URLs puede ser proactivo o reactivo. En la modalidad proactiva, se crean las URLs de los objetos embebidos de la página principal antes de cargar el contenido en el servidor origen. En la modalidad reactiva, las URLs se modifican cuando la petición del cliente llega al servidor origen. La gran ventaja de la modificación de URLs es que los clientes no están asociados a un surrogate únicamente, ya que las URLs contienen nombres DNS que apuntan a un grupo de surrogates. Además, se puede conseguir un elevado nivel de granularidad, ya que cada objeto embebido se puede gestionar de forma independiente. La gran desventaja de este mecanismo es el retardo adicional debido al parsing de URLs, así como el posible cuello de botella que puede experimentar alguno de estos elementos embebidos.

6.3.2.2.5. Anycasting

El mecanismo de anycasting puede ser dividido en anycast en capa de red y anycast de nivel de aplicación. La técnica de anycast en la capa de red consiste en un conjunto de servidores que están asociados a una dirección anycast, a la que un cliente envía su solicitud. Cada router contiene una ruta al servidor anycast más cercano en base a una cierta métrica, por lo que encaminarán cada cliente con su surrogate o site más cercano. Nótese que mientras una comunicación multicast supone que un paquete llegará a todos los servidores (realmente a todo el grupo multicast), un paquete anycast sólo llegará a un servidor. Pese a que esto pueda parecer un mecanismo de enrutamiento adecuado para una CDN, cuenta con algunos inconvenientes. En primer lugar, no existe un espacio de direcciones para alojar las direcciones anycast (al menos en IPv4). Algunos

sugieren el uso de direcciones multicast, donde sólo un servidor del grupo multicast responda a la solicitud. En cualquier caso, IP anycast requiere soporte en los routers, lo cual resulta imposible en Internet. Además, la decisión de encaminamiento es tomada a nivel de red sin considerar otros parámetros como la carga de la red, el usuario o el tipo de contenido. Debido a los problemas anteriormente descritos en la redirección DNS, la comunidad científica propuso nuevos mecanismos avanzados de encaminamiento basados en contenido. Nótese que las políticas de redirección mediante DNS son propietarias de cada proveedor de CDN, ya que emplean distintos mecanismos y diferentes métricas en sus decisiones. Aunque una CDN ofrece una configuración distribuida de contenido, no ofrece un método de localización de contenido con un mecanismo global de encaminamiento mediante DNS. Una de las alternativas fue desarrollada por científicos de la universidad de Stanford, quienes proponían un protocolo de encaminamiento basado en el nombre como parte del proyecto TRIAD. Dado que los clientes realmente desean un cierto contenido y no conectividad a un servidor, se propone un nuevo nivel, denominado nivel de contenido. Éste está basado en una nueva entidad denominada CR (Content Router) que actúa tanto de router IP como de servidor DNS. La resolución de nombres se soporta mediante un nuevo protocolo propuesto denominado INRP (Internet Name Resolution Protocol), compatible hacia atrás con el mecanismo de DNS, pero con la futura posibilidad de incorporar la URL entera y no únicamente la porción del servidor, de forma que el CR sea consciente del contenido solicitado. De forma similar a un router IP, un CR dispone de una tabla donde se asocia un nombre con el siguiente salto, de forma que se encamina la solicitud INRP hacia el servidor de contenido óptimo.

6.3.2.2.6. CDN Peering

Cuando se habla de las redes de contenido basadas en tecnología peer-to-peer, se entiende a las redes que están formadas por conexiones simétricas entre hosts de computadoras (servidores) y que distribuyen contenido entre sus peers. En estos casos, una CDN puede alcanzar a un mayor número de usuarios si emplea servidores y forward proxies cercanos de otra CDN aliada con las que puede realizar interconexiones hacia otras CDNs (peering CDNs), de cierta manera similar a los puntos neutros entre ISPs. Las CDNs con funcionalidad de peering, son más tolerante a fallos debido a que la información de la red de recuperación necesaria puede ser desarrollada por los mismos miembros del peering en lugar de depender de una infraestructura dedicada como lo hacían las CDN tradicionales. Para localizar el contenido en una CDN de tipo peering, se puede emplear centralized directory model, un esquema DHT (Distributed Hash Table), flooded request model o un document routing model.

7. Arquitectura SDN - CDN

En esta sección se estudiarán las arquitecturas existentes para redes de entrega de contenido sobre redes definidas por software y se analizará y propondrá la mejor opción para este trabajo. Hay varios beneficios de la adopción de SDN en la infraestructura de CDN. En primer lugar, SDN tiene el potencial de hacer la arquitectura de red más flexible. SDN facilita a los proveedores de CDN la gestión y actualización de su infraestructura de red subyacente. En segundo lugar, SDN proporciona interfaces para recopilar estadísticas en tiempo real de los enlaces de la red. En tercer lugar, los protocolos SDN como OpenFlow proporcionan interfaces para modificar encabezados de paquetes, lo que permite redirigir las solicitudes de los usuarios a diferentes surrogates en tiempo real. El controlador SDN tiene un conocimiento global de la red, por lo que es capaz de redirigir la petición de un cliente al surrogate con menos carga, al surrogate más cercano al cliente, etc. De esta forma se pueden tomar decisiones inteligentes en el momento adecuado.

7.1. Trabajos relacionados

El trabajo [32] propone una red de entrega de contenidos utilizando una red definida por software, para esto plantea una Peer-Assisted CDN, en donde todos los clientes de la red colaboran para brindar los contenidos. La arquitectura propuesta, plantea que los contenidos de la red se encuentran distribuidos entre los clientes de la misma y existe un servidor central, el cual es una aplicación que se nutre de los conocimientos de la red del controlador SDN, y se conecta con él a través de la northbound api, también maneja la información de que cliente almacena qué contenido. Un diagrama de la arquitectura se puede observar en la figura 18.

Esta arquitectura consta de tres componentes:

- **Controller/SDN Server:** Este componente es el encargado de administrar la ubicación de los contenidos. Es una aplicación que se comunica con el controlador SDN a través de la northbound API. Para administrar la ubicación de los contenidos, mantiene una tabla de ruteo llamada hash table/map, la cual se ordena por nombre de contenido y contiene la localización del contenido (dirección IP y directorio). Cuando un cliente desea un contenido el controlador busca en su tabla de hash, si existe coincidencia en la tabla, se contacta con el cliente que tiene el contenido y se lo sirve al solicitante. En caso contrario contactará al servidor de origen para obtener el contenido deseado. El servidor controlador se nutre de la información de la topología de red, que es conocida por el controlador SDN.
- **Origin server:** Es el servidor que se encarga de originar el contenido.
- **Client:** Son los usuarios de la CDN-SDN, ellos solicitan contenidos, pero también son los encargados de almacenar contenidos. El controlador tiene

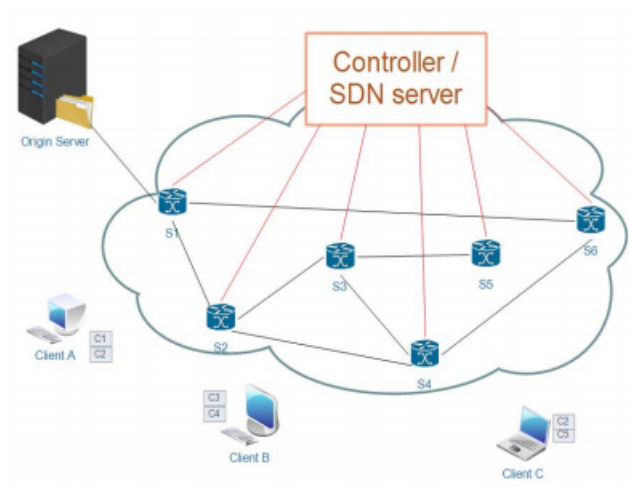


Figura 18: Arquitectura CDN-SDN 1 [32]

la responsabilidad de conocer la ubicación de cada uno de ellos.

En esta arquitectura, se por sentado que cualquier solicitud va dirigida al servidor controlador, que cada cliente ejecuta código de servidor (para poder compartir contenidos) y que el servidor controlador tiene conocimiento de todos los contenidos de la red.

En la figura 19 se puede observar el flujo de mensajes cuando un cliente solicita un contenido, y éste contenido se encuentra en la tabla de hash del controlador. El controlador solicita el contenido al cliente que está asociado al mismo en la tabla de hash y luego se lo envía al cliente solicitante. Mientras que la figura 20 muestra cuando el contenido no está en ninguno de los clientes, por lo que el controlador se lo debe solicitar al servidor de origen.

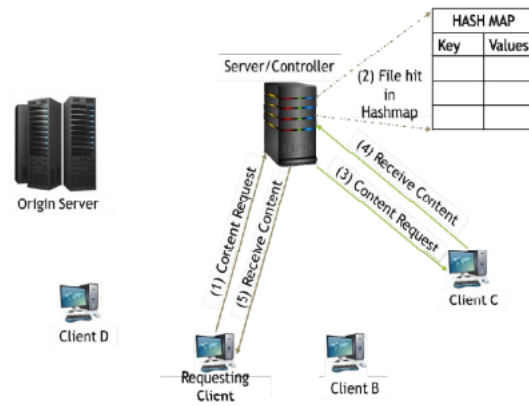


Figura 19: Flujo de mensajes, contenido en hash [32]

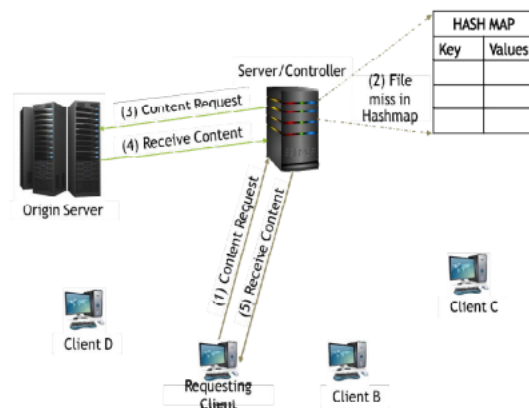


Figura 20: Flujo de mensajes, no contenido en hash [32]

Cada uno de los clientes de esta arquitectura debe levantar un servidor para que el controlador se pueda conectar a él en caso de tener que solicitarle un contenido. Lo mismo que el servidor de origen. En la arquitectura propuesta todas las conexiones se realizan a través de sockets.

Esta arquitectura no aplicaría para los fines de este trabajo, ya que la idea es que estudiantes, docentes y funcionarios accedan a los contenidos de la RAU, sin la necesidad de instalar ningún software que haga de servidor para compartir contenidos.

Por otro lado, en [33] se propone una arquitectura CDN - SDN para dar soporte a vídeo streaming de alta calidad. Lo que plantea este trabajo es que dado que videos y otros contenidos multimedia se vuelven cada vez más populares entre los usuarios de Internet de hoy en día, se da una mejora en la infraestructura subyacente de Internet, por lo que los usuarios pueden obtener contenidos de vídeo con una calidad mucho mayor a la que se obtenía en la década pasada. Los contenidos de vídeo, especialmente de streaming de vídeos de alta calidad, ocupan una parte importante del tráfico de Internet. Los algoritmos de balanceo de carga de una CDN tradicional, son los encargados de balancear las cargas de tráfico, pero estos algoritmos desconocen los requerimientos de ancho de banda del usuario. Esto puede hacer que las necesidades de los usuarios no se satisfagan. Por lo que, la arquitectura propuesta en el trabajo mencionada intenta satisfacer esas necesidades integrando una SDN a una infraestructura de CDN.

El trabajo propone una arquitectura CDN flexible con dispositivos de red habilitados para OpenFlow, de esta forma los proveedores de CDN pueden centralizar la administración de la infraestructura subyacente y acortar el proceso de actualización de su arquitectura de red. También propone combinar algoritmos tradicionales de balanceo de carga con controladores SDN para lograr una visión global de los CDN y tomar mejores decisiones. El diseño de la arquitectura se puede observar en la figura 21.

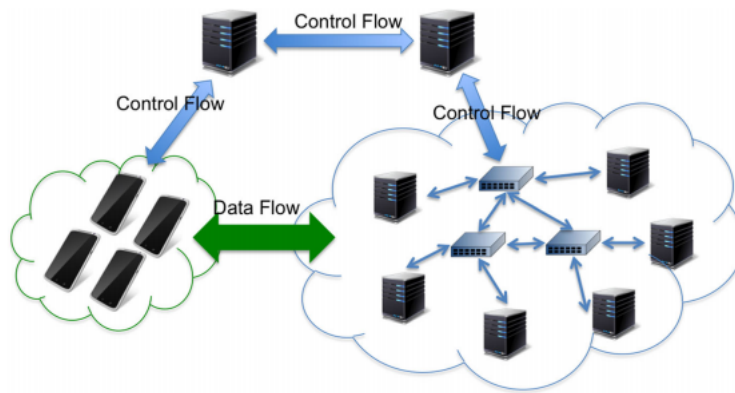


Figura 21: Arquitectura CDN-SDN 2 [33]

Entre los componentes de la arquitectura se encuentra el controlador SDN, el cual desempeña un papel clave ya que es el único punto de administración centralizado de la arquitectura propuesta. El controlador es responsable de supervisar toda red y tomar decisiones de enrutamiento para las solicitudes entrantes de los hosts. Los controladores SDN son capaces de tomar decisiones de enrutamiento de manera más eficiente ya que normalmente se ejecutan en servidores de alto rendimiento. El otro componente importante de la arquitectura es el balanceador de carga, el cual es una aplicación a la cual se conectarán los clientes para solicitar el contenido deseado, esta aplicación se conecta con el controlador SDN a través de la northbound api. En los enfoques de CDN tradicionales, los balanceadores de carga reconocen el conjunto de recursos en la CDN. Cuando un balanceador de carga recibe solicitudes de recursos por parte de los clientes, examina los recursos disponibles en la red, por ejemplo, las posibles conexiones a contenidos de vídeo específicos. Si se encuentran recursos adecuados, el balanceador de carga reenvía al cliente al surrogate donde residen los recursos. Pero en este enfoque tradicional los balanceadores de carga no son totalmente conscientes de los requisitos de ancho de banda de los clientes. Por lo que en este trabajo el balanceador de carga se beneficia de las poderosas funciones soportadas por SDN. En lugar de responder directamente a los clientes utilizando algún algoritmo de balanceo de carga, el balanceador de carga en primer lugar se comunica con el controlador SDN para obtener estadísticas de enlaces en tiempo real, para ver cuáles satisfacen los requerimientos necesarios. El balanceador de carga calcula las mejores conexiones para cumplir con los requisitos de calidad de vídeo de los clientes. Cuando finaliza el cálculo, el balanceador de carga devuelve una o varias conexiones a clientes.

Por otro lado el trabajo [34] se propone un desarrollo de servicios de distribución de contenidos multimedia en redes corporativas con tecnología SDN. Para esto plantea la sustitución de los algoritmos y mecanismos de encaminamiento (redirección DNS, es el que reemplaza en el trabajo) de CDN por un controlador SDN. Por lo tanto en el trabajo antes mencionado se utiliza el controlador SDN para controlar y dirigir los contenidos. Cuando un cliente solicita un contenido, el controlador SDN, se encarga de seleccionar el surrogate más próximo al cliente e informárselo.

Este trabajo propone la sustitución mencionada ya que los servidores de DNS tienen una sobrecarga muy alta, un rendimiento bajo, elevados retardos, y además con un número grande de servidores de DNS es difícil controlar la red y enfrentarse a los fallos. Para esto se desarrolla el algoritmo de dijkstra en el controlador SDN, el cual es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista. El trabajo utiliza el controlador SDN POX, en el cual se puede diseñar un método de Python en el script de controlador para realizar el algoritmo de dijkstra. Con el método de dijkstra se puede sacar una ruta entre los dos puntos con menor coste.

En resumen lo que plantea este trabajo es utilizar el controlador SDN, con el algoritmo dijkstra y de esta forma encontrar el surrogate de la CDN más cer-

cano a los clientes.

En [35] se realiza un estudio sobre el uso de una red definida por software para mejorar el encaminamiento de peticiones en una CDN que colabora con el ISP. Se propone un esquema bastante parecido al del trabajo mencionado anteriormente [34], la idea es sustituir el redireccionamiento DNS en la CDN, por dos componentes que son gobernados por el controlador SDN. La arquitectura se puede observar en la imagen 25.

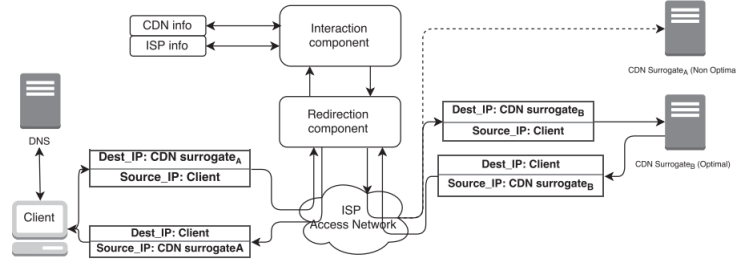


Figura 22: Arquitectura CDN-SDN 4 [35]

Esta arquitectura propone la implementación de dos componentes: componente de interacción y componente de redirección, estos componentes son los encargados de sustituir al servidor DNS, el cual se encargaba de dirigir a los clientes al surrogate adecuado. Los pasos para que un cliente obtenga un contenido son los siguientes:

- 1. Un cliente solicita contenido de un proveedor de contenido realizando una solicitud DNS para una url determinada.
- 2. La resolución DNS devuelve una dirección IP correspondiente a un surrogate de la CDN.
- 3. El cliente inicia una sesión TCP con la dirección IP proporcionada y realiza las solicitudes HTTP GET como parte de una sesión HTTP
- 4. El surrogate devuelve el contenido solicitado al cliente

Las contribuciones de esta solución se aplican en los pasos 2, 3 y 4. Cuando el cliente inicia una conexión TCP al surrogate indicado por su DNS, el componente de redirección consulta con el componente de interacción, el cual con ayuda de los conocimientos de la CDN y del ISP obtiene el surrogate apropiado para el cliente y se lo informa al componente de redirección. Una vez que el componente de redirección obtiene el surrogate adecuado, cambia la dirección ip destino del paquete (DNAT) y envía la petición al surrogate. Cuando el surrogate responde con el contenido, el componente de redirección hace el proceso

inverso (SNAT) y cambia la dirección ip origen del paquete.

El componente de redirección está formado por switches OpenFlow, los cuales hacen uso del action set_field para cambiar los encabezados del mensaje IP. El componente de interacción es el controlador SDN (el cual debe soportar OpenFlow).

El trabajo [36] estudia cómo optimizar el enrutamiento y la selección de un surrogate en una CDN utilizando redes definidas por software. Al igual que el trabajo mencionado anteriormente, se propone una arquitectura donde el controlador SDN es el encargado de monitorear el estado de la red en tiempo real y en base a esto se encarga de redirigir las peticiones al surrogate adecuado.

Los trabajos presentados tienen en común el aprovechamiento de redes definidas por software para conocer el estado de la red en tiempo real y así tomar decisiones de encaminamiento.

7.2. Arquitectura propuesta

En este punto se detalla la arquitectura CDN - SDN propuesta para la RAU. En la arquitectura propuesta por este trabajo, se sustituirán los algoritmos y mecanismos de encaminamiento de CDN (sistema de encaminamiento), por el controlador SDN. El controlador SDN utilizará su conocimiento global de los nodos y el estado de enlaces de la red para seleccionar en tiempo real el mejor surrogate para cada solicitud de los clientes. El cliente realizará sus solicitudes a los servidores de origen, y los dispositivos de red serán los encargados de modificar los cabezales IP (TCP/UDP) de esas solicitudes y redirigirlas según lo disponga el controlador SDN. Los contenidos de la red se replicarán de forma completa en los surrogates (full-site). Se seleccionó esta forma de replicar los contenidos debido a su sencillez. Se contará con un controlador CDN, el cual será el encargado de gestionar los surrogates de la red, según lo disponga el administrador de red. El controlador CDN nutrirá una base de datos con el mapeo de servidores que generan contenido y sus surrogates asociados, esta base de datos también será consultada por el controlador SDN.

En resumen, para cada servidor de origen (ip servidor de origen, puerto del servicio que provee los contenidos a replicar) se tendrá un conjunto de surrogates (ip surrogate, puerto del servicio que provee los contenidos replicados), en cada solicitud a un servidor de origen los dispositivos de red OpenFlow consultarán al controlador para seleccionar para cada origen su mejor surrogate (según el cliente que realice el pedido), una vez que obtienen la respuesta, realizan el cambio de ip, puerto (si es que difieren los puerto del origen y el surrogate) para redirigir la petición al surrogate seleccionado por el controlador SDN.

Un esquema de la arquitectura propuesta se puede observar en la figura 23.

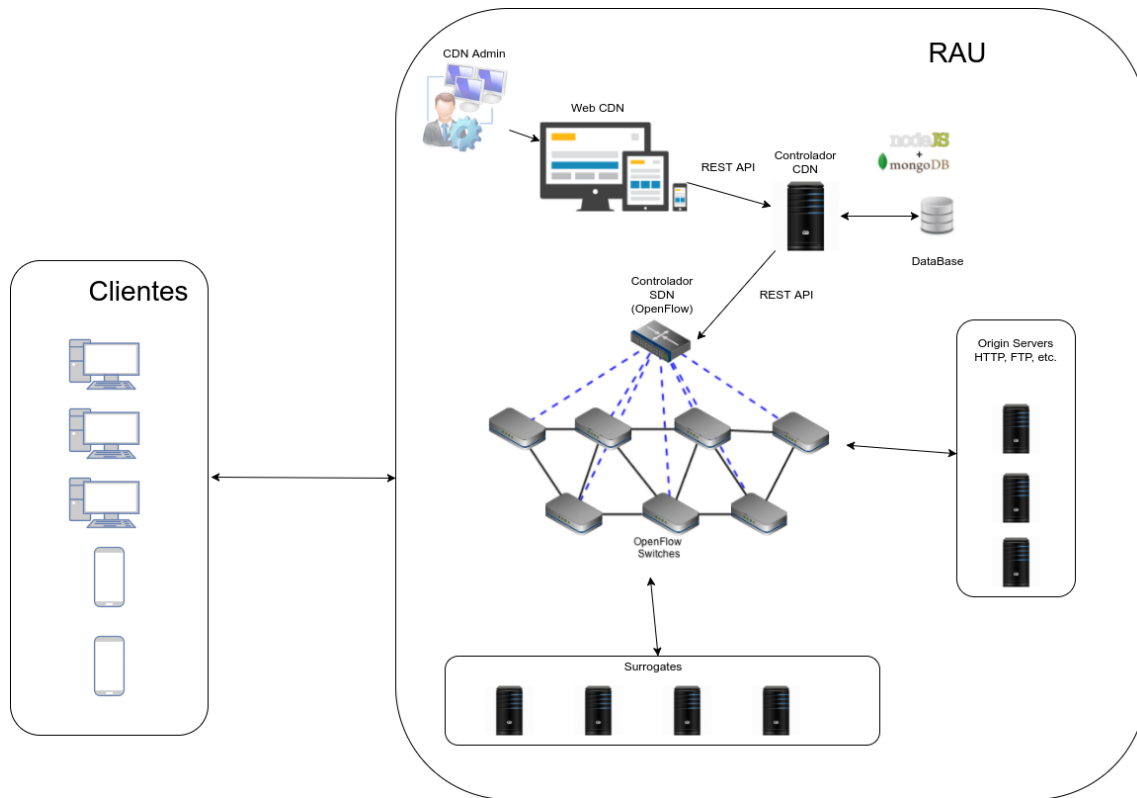


Figura 23: Arquitectura Propuesta

A continuación se detallan los componentes presentados en la arquitectura de la figura 23:

- **OpenFlow Switches:** Son los switches de la RAU, y tienen soporte para OpenFlow.
- **Controlador SDN:** El controlador SDN a utilizar será Ryu, la elección se debió a que la aplicación RAUFlow ejecuta una aplicación Ryu. Este controlador tiene una visión global del estado de la red en tiempo real, es el encargado de instalar flujos en los switches OpenFlow, según la mejor opción de surrogate para las solicitudes de los clientes, para esto utilizará el algoritmo de seleccion de surrogate, el cual se describira mas adelante. El controlador SDN, expone una northboud API (mediante servicios RESTful) la cual será consumida por el controlador CDN. Más adelante se detallan los métodos de la northbound API. El controlador SDN carga en su inicio los datos de la base de datos correspondientes a los mapeos de surrogate, servidor de origen. Como ya se mencionó la southbound api a utilizar será OpenFlow.

- **Controlador CDN:** Está construido por encima del controlador SDN, en la capa de aplicación. Será el encargado de indicar al controlador SDN que se agregue/elimine un surrogate y que se agregue/elimine un servidor de origen. Es el encargado de gestionar los servidores de origen y surrogates, según lo ordene el administrador de la CDN, para esto expone servicios RESTful, los cuales serán consumidos por el administrador de CDN a través de una página web (desarrollada para este trabajo). El controlador será desarrollado en node.js y realizará la gestión de la CDN utilizando un base de datos mongoDB.
- **Origin Servers:** Serán los encargados de originar contenido en la RAU, servidores HTTP, FTP, etc.
- **Surrogates:** Serán quienes tienen las réplicas de contenidos de los Origin Servers.
- **DataBase:** La base de datos de la arquitectura guardará el mapeo entre servidor de origen de contenidos y los surrogates que cachean esos contenidos. Se utilizará mongoDB, ya que trabaja muy bien con node.js y tiene una muy buena performance.
- **Admin CDN:** Es el encargado de comunicar al controlador CDN sobre surrogates y Origin servers. La comunicación la realiza a través de la página web de administración.
- **Web CDN:** Será una web desarrollada con Angular 2 y html5, a la cual ingresará el admin CDN, para comunicarse con el controlador CDN, enviará las solicitudes del admin CDN al controlador CDN, consumiendo los servicios RESTful expuestos por el controlador CDN.
- **Clientes:** Son quienes consumen los contenidos de la RAU, los cuales pueden ser estudiantes, docentes, funcionarios, etc.

En la figura 24 se muestra el flujo de mensajes cuando el administrador CDN, quiere agregar o eliminar un surrogate a la red.

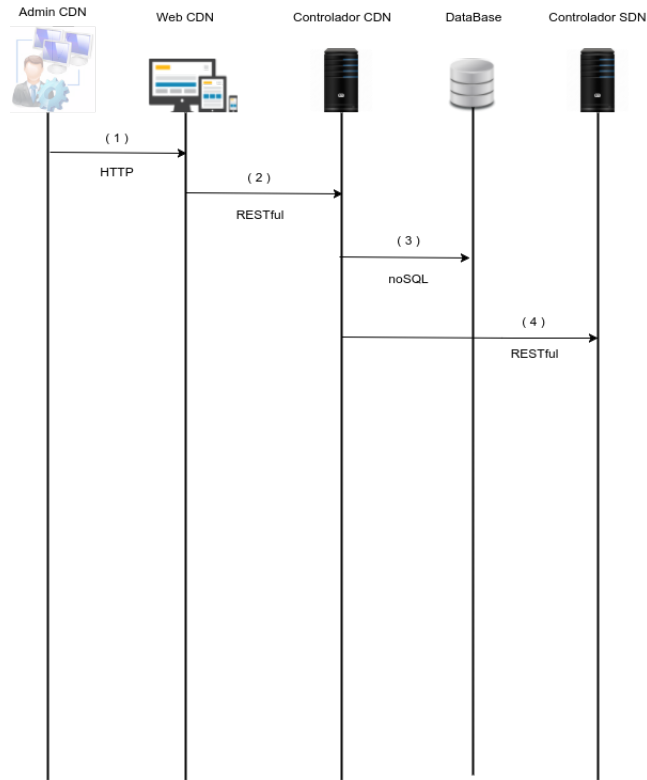


Figura 24: Flujo Administrador

Los pasos que se siguen son los siguientes:

- 1. El administrador CDN ingresa al sitio web de administración CDN, indica que se agregó/elimino un surrogate de la cdn y pasa la dirección ip y el puerto del surrogate junto con la dirección ip y puerto del origin server desde el cual se cacheara el contenido, también indica si el protocolo de capa de transporte será udp o tcp.
- 2. El sitio web consume los servicios RESTful expuestos por el controlador CDN y utiliza los métodos de eliminar o agregar surrogate pasándole la información ingresado por el administrador de la CDN.
- 3. El controlador CDN guarda/elimina en la mongoDB el mapeo ipOriginServer, ipSurrogate.
- 4. El controlador CDN le indica al controlador SDN a través de la north-bound API, que debe realizar el mapeo ipOriginServer, ipSurrogate.

- 5. El controlador SDN guarda en memoria el mapeo indicado por el controlador CDN.

Cuando el Administrador quiere agregar un nuevo servidor de origen en la CDN, los pasos son similares, en la base de datos se guardará el nuevo servidor de origen sin ningún surrogate asociado (por lo tanto las peticiones irán directamente al origen) y el controlador guardará en memoria este nuevo servidor de origen.

En la figura 25 se muestra el flujo de mensajes cuando cliente requiere un contenido de la RAU.

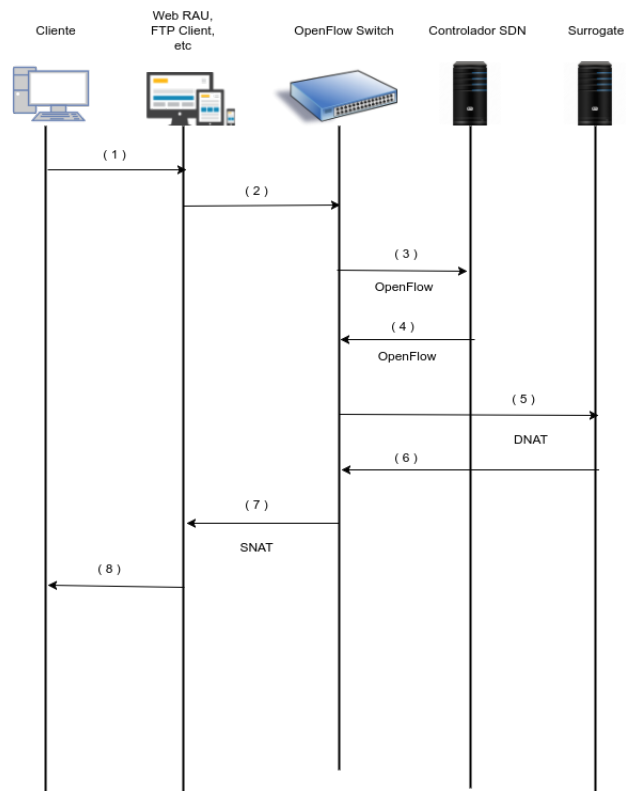


Figura 25: Flujo Cliente

Los pasos que se siguen son los siguientes:

- 1. El cliente accede a su cliente Web/cliente FTP, etc y solicita un contenido de la rau indicando la url del servidor de origen o la dirección ip. En caso de ingresar la url del servidor de origen, el servidor DNS del usuario le indicará la dirección IP del servidor en cuestión.
- 2. La aplicación cliente (cliente web, cliente ftp, etc) se intentará conectar al servidor origen, ingresando a la RAU.
- 3. Cuando el flujo de datos de la conexión al servidor de origen llegue a los dispositivos de la capa de infraestructura de la RAU (RAUSwitches), estos le consultarán al controlador SDN que hacer con él.
- 4. El controlador SDN aplicará un algoritmo de encaminamiento para obtener el mejor surrogate (mas adelante se describe el algoritmo implementado) para la petición del cliente y le pasara la información al dispositivo OpenFlow.
- 5. El dispositivo OpenFlow aplicará DNAT cambiando la dirección IP destino de la petición y el puerto destino (si es que son diferentes puerto servidor Origen - puerto surrogate), por último la redirigirá al surrogate seleccionado.
- 6. El surrogate enviará el contenido solicitado por el cliente.
- 7. Los dispositivos de red aplicarán SNAT para cambiar la dirección IP origen del contenido y el puerto origen (si es que son diferentes puerto servidor Origen - puerto surrogate), luego lo redirigirá al cliente solicitante.
- 8. La aplicación cliente le sirve el contenido al cliente.

7.3. Arquitectura propuesta vs CDN Tradicional

Como se mencionó anteriormente la arquitectura propuesta busca sustituir los algoritmos y mecanismos de encaminamiento de CDN tradicional por el controlador SDN. Cuando se estudio el estado de arte de las CDN tradicionales, se presentaron los diferentes mecanismos de encaminamiento que existían, estos mecanismos cuentan con algunas desventajas, las cuales son contrarrestadas con el controlador SDN. A continuación se presentan las desventajas mencionadas y como el controlador SDN ayuda a contrarrestarlas.

Uno de los mecanismos presentados fue el GSLB, para implementar este mecanismo es necesario disponer de un switch central, el cual estará conectado a todos los surrogates de la red y un servidor DNS autoritativo inteligente. El switch central se encargará de recolectar información sobre cada uno de los switches, por lo que cada uno de los surrogates informan al servidor central acerca de su estado de carga, de esta forma el switch central cuenta con información global de la red. En este mecanismo cuando se quiere acceder a un recurso de

un servidor de origen, el servidor DNS autoritativo inteligente recibe el pedido y segun la informacion de carga de la red con la que cuenta el switch central, el servidor DNS redirigirá la petición al mejor surrogate. En la arquitectura propuesta no es necesario tener un servidor DNS autoritativo inteligente, ya que los pedidos irán dirigidos al servidor de origen y el controlador SDN, será el encargado de cambiar los cabezales ip de los pedidos, según lo disponga el algoritmo de encaminamiento implementado (más adelante se describe este algoritmo). Si se quisiera implementar un algoritmo que tenga en cuenta la carga de cada uno de los surrogates, es necesario configurar en cada uno de los surrogates y en el controlador una aplicacion de gestion de red, que utilice por ejemplo el protocolo SNMP. En este caso se deberían configurar manualmente cada uno de los surrogates y el controlador, esta es una de las principales desventajas del mecanismo GSLB. Por lo que la principal ventaja del controlador SDN, si se quisiera implementar en vez de usar el mecanismo GSLB es que no es necesario contar con un servidor DNS autoritativo inteligente.

Otro de los mecanismos presentados fue la redirección basada en DNS, este mecanismo es uno de los más usados por los proveedores de CDN. Pero cuenta con algunas desventajas, las cuales fueron presentadas anteriormente, estas desventajas no aplican en la arquitectura propuesta ya que no es necesario contar con un servidor DNS para redirigir los pedidos. La ventaja de este mecanismo es la transparencia, ya que los servicios y contenidos quedan referenciados por sus nombres DNS, y no por sus direcciones IP, en la arquitectura propuesta también se tiene una total transparencia ya que todos los pedidos son dirigidos por el cliente al servidor de origen y es el controlador el que se encarga de realizar la redirección, esto es transparente para el cliente, el cual recibirá el contenido solicitado en nombre del servidor de origen.

La gran ventaja del mecanismo de redirección HTTP es su flexibilidad y simplicidad, además este mecanismo permite gestionar la redirección con una granularidad fina, esto en la arquitectura propuesta no es posible, ya que todo lo que se solicite por parte de un cliente a un servidor de origen, será servido por el mejor surrogate elegido, de esta forma no se podrán redirigir peticiones que tengan un mismo servidor de origen como destino a varios surrogates.

La principal desventaja que tiene la arquitectura propuesta es que se cuenta con un único controlador SDN, por lo que se tiene un único punto de falla. De esta forma si hay un problema en el controlador SDN, la red se quedara sin poder entregar contenidos.

8. Implementación arquitectura propuesta

En este punto se detallará cómo se implementarán los distintos componentes de la arquitectura propuesta en la sección anterior (tanto como para una red ip, como para una red ethernet), esta implementación es a modo de prueba de concepto. Para lograr la implementación de estos componentes se utiliza como base el desarrollo del entorno virtual (extensión de mininet) realizado en [37]. Se realizaron diversas pruebas de concepto más pequeñas, para probar algunos de los componentes más importantes, las cuales se detallan en los anexos del informe. En la siguiente sub sección se resume cómo funciona el emulador mininet. La extensión de este emulador fue construida para poder realizar pruebas funcionales y de escala sobre la arquitectura RAUFlow, la cual es propuesta por el prototipo para la RAU 2, el cual fue construido en el proyecto RRAP [2]. También se presentan las distintas tecnologías usadas para la implementación de la arquitectura.

8.1. MiniNet

Es una plataforma de emulación de red, la cual es open source. Se pueden crear redes definidas por software (cuyas dimensiones pueden ser de hasta cientos de nodos, según la configuración deseada, dependiendo del HW donde se corra la plataforma) utilizando un único kernel linux. MiniNet permite crear, interactuar, personalizar y compartir de forma rápida un prototipo de red definido por software al mismo tiempo proporcionar un camino fácilmente adaptable para la migración a hardware. La primer versión de Mininet fue desarrollado por Bob Lantz y Brandon Heller, basado en el prototipo original creado por Bob Lantz en el 2009. La versión actual es 2.2.1 y corre sobre un sistema operativo Ubuntu 14.04 LTS. Mininet se ejecuta sobre una sola máquina y emula solo enlaces cableados, ofreciendo una virtualización parcial que limita su funcionalidad, ya que no puede manejar diferentes kernels del sistema operativo simultáneamente, por lo que todos los host comparten el mismo sistema de archivos. MiniNet permite emular los siguientes componentes:

- **Links:** Es un enlace de Ethernet virtual con la función de actuar como un cable de red conectando dos interfaces virtuales, pudiendo enviar paquetes desde una interfaz a la otra, el comportamiento de este link es como el de un puerto de Ethernet real.
- **Hosts:** En Mininet un host es un proceso simple de Shell situado dentro de su propio espacio de nombres de red, cada host posee su propio enlace virtual.
- **Switches:** Son dispositivos con software OpenFlow que tienen el mismo comportamiento como un dispositivo real.
- **Controladores:** Un controlador puede estar en cualquier lugar de la red, tiene que tener conectividad ip con otros dispositivos (switch) cuando estos estén en ejecución.

La instalación de esta herramienta se puede realizar de manera muy sencilla, ya que desde su página web oficial[38], se puede descargar una máquina virtual preconfigurada (archivo ovf, para ser usado con VirtualBox). MiniNet cuenta con una gran documentación, muy detallada sobre su funcionamiento e instalación, lo cual es de gran utilidad para los usuarios. Una gran ventaja que posee esta herramienta, es que se pueden crear determinadas topologías, de forma muy rápida, utilizando un único comando.

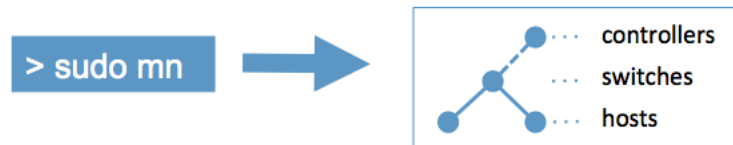


Figura 26: Mininet [38]

Algunas de las topologías que permite crear la herramienta son:

- Single: Un switch, N hosts

```
# mn --topo=single ,N
```

- Linear: N switches, cada uno conectado a un único host

```
# mn --topo=linear ,N
```

- Tree: Árbol, de profundidad N

```
# mn --topo=tree , depth=N
```

- Definida por el usuario: A través de un archivo python el usuario puede crear su topología

```
# mn --custom mytopo.py --topo mytopo
```

MiniNet por defecto soporta la versión 1.0 de OpenFlow, pero se puede activar el soporte para OpenFlow 1.1, 1.2 y 1.3. Cuenta con OpenVSwitch integrado.

```
# mn -switch ovs,protocols=OpenFlow13
```

La estructura básica de un modelo de red en esta plataforma, se muestra en la figura 27.

La plataforma permite configurar el controlador SDN a utilizar, se puede utilizar un controlador gestionado por MiniNet o se puede utilizar un controlador gestionado por el usuario, indicando su dirección ip y su puerto.

```
# mn --controller=remote , ip=127.0.0.1 , port=6653
```

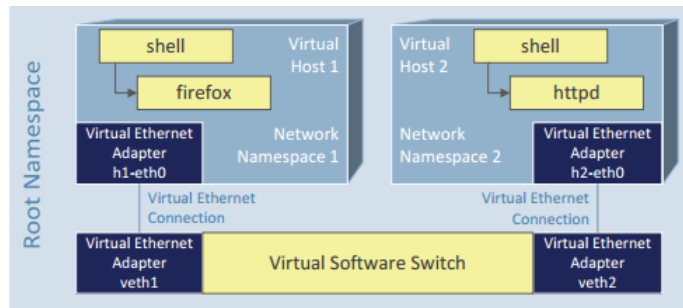


Figura 27: Estructura MiniNet [39]

Si no se elige un controlador, Mininet utiliza el controlador que incorpora por defecto. La distribución OpenFlow de referencia incorpora un controlador que actúa como un switch clásico. En lenguaje OpenFlow, se le denomina Ethernet Learning Switch. Su funcionamiento es el de un switch tradicional: según vaya recibiendo tramas irá creando una tabla donde se asociarán direcciones MAC con puertos. Éste controlador básicamente al asociar una MAC con un puerto, enviará un mensaje OpenFlow al switch, instaurando una nueva entrada en la tabla de flujo.

La plataforma cuenta con una GUI, llamada Miniedit, la cual se agregó desde la versión 2.2.0.1, está desarrollada en python, trayendo incorporados los elementos básicos de una red, como lo son, hosts, switches, controlador, links (cable virtual), routers, entre otros, permitiendo al usuario poder crear la topología escogiendo los dispositivos que desea. También permite poder exportar en python una topología creada para ejecutarla desde la línea de comandos de MiniNet.

La extensión usada en este trabajo agrega cuatro clases de entidades a emular, los cuales son:

- **RAUSwitch:** La clase RAUSwitch es el núcleo del entorno virtual. Dado que Mininet es una herramienta de prototipado para SDN puro, no está pensado para un esquema híbrido, de esta forma los switches emulados por Mininet no se pueden comportar como switches OpenFlow y al mismo tiempo como routers tradicionales. Es por este motivo que la extensión utiliza la clase Host y se la extiende para cumplir los requerimientos del RAUSwitch. Cada RAUSwitch está en su propio network namespace (igual que la clase Host) y ejecuta su propia instancia de Quagga y Open vSwitch.
- **RAUController:** Host virtual, que ejecute la aplicación de RAUFlow y se comunique con los switches a través de enlaces virtuales.
- **QuaggaRouter:** Es una clase similar al RAUSwitch pero sin Open vSwitch, es decir, sólo usa Quagga. Apunta a representar el router CE (Customer Edge) que utilizaría una subred cliente para conectarse a la red SDN. Siempre debería estar conectado a un RAUSwitch de borde.

- RAUHost: Representa a los hosts que serán clientes de la red.

Para más información sobre estas clases y su modo de uso leer [37]. Los RAUSwitches y el RAUController de la extensión antes mencionada ejecutan el algoritmo de enrutamiento OSPF como IGP a través del software de ruteo Quagga. De esta forma los RAUSwitches y el RAUController nutren sus tablas de ruteo con las rutas dentro de la SDN.

8.2. Implementación de componentes

En esta sección se detalla la implementación de cada uno de los componentes que forman parte de la arquitectura presentada.

8.2.1. Tecnologías

En esta sección se presentan las tecnologías utilizadas en la implementación de la arquitectura propuesta.

8.2.1.1 Node.js

Como se mencionó en la sección anterior, para el desarrollo del servidor web que contendrá el sitio web de administración de la arquitectura se utilizara Node.js [40]. Este servidor web oficiará de controlador CDN exponiendo los servicios RESTful para la administración de surrogates.

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web. Se utilizará en su versión 7.6.0.

8.2.1.2 mongoDB

Es una base de datos multiplataforma open source orientada a documentos [41], esto quiere decir que en lugar de guardar los datos en registro, los guarda en documentos. que son almacenados en BSON (formato ligero para intercambio de datos). Forma parte de la familia S.B.P NOSQL. Tiene la capacidad de realizar consultas utilizando javascript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutada. Se utilizará en su versión 2.6.12.

8.2.1.3 AngularJS

Se utilizará AngularJS 2 para el desarrollo del sitio web de administración. AngularJS [42] es un framework de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de

Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

8.2.1.4 Ryu Controller

Como controlador SDN se utilizará Ryu, como se mencionó anteriormente Ryu Está escrito en Python. Dispone de un conjunto de APIs bien definidas, que permite el desarrollo de nuevas aplicaciones de gestión de red, y de varios protocolos para la gestión de los dispositivos de red, como OpenFlow y Netconf. Está escrito en Python soporta completamente desde la versión 1.0 hasta la 1.5 de OpenFlow. Su versión actual es la 3.6. En la arquitectura de este trabajo se utilizará ryu con soporte para OpenFlow 1.3.

8.2.2. Modelo de datos

El modelo de datos para esta implementación de arquitectura es bastante simple, se modelaron por un lado los servidores de origen y sus surrogates asociados y por otro lado se modelaron dos relaciones, una para la implementación para una red de capa 2 (ethernet) y otra para la implementación de una red de capa 3 (ip). La relación para la implementación en una red ethernet es la de RAUSwitch de borde, mac de QuaggaRouter conectado a él, esto se debe a las conclusiones a las que se llegaron en la prueba de concepto 3. Para la implementación sobre una red ip se modelo el mapeo entre RAUSwitches de borde y número de puerto de su interfaz física de borde. Por lo tanto el modelo es de la siguiente forma:

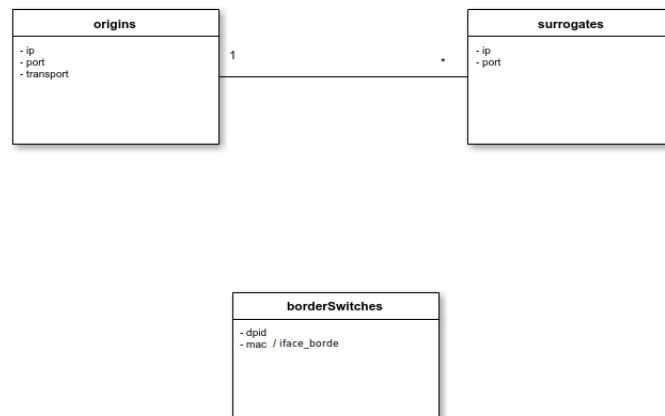


Figura 28: Modelo de datos

- origins:

- ip: Dirección ip del servidor origen.
 - port: Puerto donde escucha el servidor de origen.
 - transport: Protocolo de capa de transporte utilizado por el servicio en el servidor de origen.
- surrogates:
 - ip: Dirección ip del surrogate.
 - port: Puerto donde escucha el surrogate.
 - borderSwitches:
 - dpid: Identificador del RAUSwitch de borde.
 - mac (solo implementación capa2): Dirección mac del QuaggaRouter de borde al que se conecta el RAUSwitch.
 - iface_borde (solo implementación capa3): Número de puerto físico de borde del RAUSwitch de borde.

Cuando se utiliza un modelo NOSQL basado en documentos, como es el caso del modelado de datos en MongoDB, la forma de modelar las relaciones entre las entidades difiere de los modelos ER. Por lo tanto para modelar la relación One-to-Many entre origen y surrogates se utiliza una estrategia llamada Embedding. En este caso se tiene una sola colección de documentos que será la de los servidores de origen. Por cada una de las entradas de orígenes se tendrán anidados tantos surrogates como se hayan configurado, un ejemplo de esta relación se puede observar en el siguiente código.

```
{
  ip: "10.10.11.1",
  port: 80,
  transport: "tcp",
  surrogates:
  [
    {ip: "10.10.12.1", port:80},
    {ip: "10.10.13.2", port:8080},
    {ip: "10.10.18.1", port:80}
  ]
}
```

Mientras que un ejemplo del documento que contiene la relación entre los RAUSwitches de borde y la mac del QuaggaRouter al que se encuentra conectado (implementación capa 2) es el siguiente:

```
{
  switches:
  [
```

```

        { dpid: 1, mac: "00:00:00:00:00:01" },
        { dpid: 2, mac: "00:00:00:00:00:02" },
        { dpid: 3, mac: "00:00:00:00:00:03" }
    ]
}

```

Un ejemplo del documento que contiene la relacion entre los RAUSwitches de borde y su puerto fisico de borde (implementación capa 3) es el siguiente:

```

{
    switches:
    [
        { dpid: 1, port: 3 },
        { dpid: 2, port: 3 },
        { dpid: 3, port: 3 }
    ]
}

```

8.2.3. Base de datos

Se utilizaran dos documentos de mongoDB, uno llamado 'origins' y el otro llamado 'borderSwitches'. El primero contendrá los servidores de origen y sus correspondientes surrogates, mientras que el otro guardará el mapeo entre RAUSwitches y macs de los QuaggaRouters de borde, para el caso de la implementación sobre una red de capa 2 y para la implementación de una red de capa 3, guardara el mapeo RAUSwitch de borde, número de puerto de su interfaz física de borde. La base de datos se instala y corre en la máquina encargada de levantar la VM que contiene la extensión de MiniNet.

8.2.4. Ryu App

En esta sección se detallan las implementaciones de las aplicaciones ryu, tanto para una red ethernet como para una red ip.

8.2.4.1 Implementación para una red ethernet

Esta implementación se realizó para un red ethernet en la cual el controlador se debe encargar de realizar los ruteos hacia surrogates y servidores de origen, además de realizar los cambios de cabezales ip (tcp/udp). Se toma como base la aplicación RAUFlow (la cual es ejecutada por el RAUController) desarrollada en [37] llamada

"gui_topology.py" (su codigo fuente y el de sus dependencias se encuentran en [43]). La aplicación ryu de esta implementación se anexa al trabajo (controller_app_implementacion_eth.py). Para que la aplicación del controlador pueda interactuar con la base de datos, se utiliza un plugin de mongoDB para python, llamado pymongo [44], en su versión 3.4. En el controlador se tendrán estructuras donde se guardará la información referente a servidores de origen

(ip, puerto), surrogates (ip, puerto) y RAUSwitches de borde.

Al controlador se le agregan dos funciones, una encargada de agregar flujos a los switches y la otra encargada de capturar paquetes entrantes al switch, mediante la suscripción al evento 'EventOFPPacketIn'. El controlador al iniciar llenará estas estructuras con los datos existentes en la base mongoDB. Luego por cada switch que se conecta al controlador, este recorre el listado de servidores de origen con sus respectivos surrogates y agrega para cada puerto y protocolo de transporte (tcp o udp) dos flujos, los cuales indican que flujos con la dupla puerto y protocolo de transporte sean enviados al controlador para su análisis. Por ejemplo si se tiene un servidor HTTP de origen escuchando en el puerto 80 y un surrogate de ese servidor escuchando en el puerto 8080, el controlador agregara en cada switch cuatro flujos, uno para paquetes con puerto tcp destino 80, otro con puerto tcp origen 80, otro con puerto tcp destino 8080 y uno más con puerto tcp origen 8080. De esta forma cuando un cliente realiza un pedido al servidor de origen al puerto 80, el switch enviará a analizar ese paquete al controlador, el cual seleccionara el mejor surrogate para ese pedido. Un diagrama del inicio del controlador se puede observar en la figura 29.

Una vez instalados los flujos iniciales, el controlador analizara los paquetes entrantes a los switches, que cumplan con los flujos instalados. El primer chequeo que realiza el controlador es fijarse que destino tiene el paquete recibido, si el paquete tiene como destino un servidor de origen (ip, puerto, protocolo de transporte), aplica el algoritmo para seleccionar el mejor surrogate para ese origen (más adelante se entrará en detalle sobre el algoritmo de selección) y le indicará al switch que para ese paquete se debe de aplicar un cambio de ip destino y de puerto destino (en caso de diferir los puertos entre el surrogate y el origen), instalando un flujo. Este flujo tendrá un timeout por inactividad, el cual se seteo a modo de ejemplo en un minuto. Si el servidor de origen al que está destinado el paquete no tiene ningún surrogate asociado, el paquete será destinado al servidor de origen, pero en ese caso no se instalará ningún flujo.

Si el paquete no tiene como destino un servidor de origen, el controlador se fijará si tiene como origen un surrogate, en caso de ser así, le indicará al switch que debe realizar el cambio de ip origen y puerto de origen (en caso de diferir puerto entre origen y surrogate), instalando un nuevo flujo. Este flujo tendrá un timeout por inactividad, el cual se seteo también a modo de ejemplo en un minuto.

Si no se cumple ninguno de los chequeos anteriores, el controlador se fijará si el paquete tiene como destino u origen, un surrogate o un servidor de origen, de ser así le indicará al switch por qué puerto debe reenviar el paquete. Estos casos refieren a switches intermedios, o switches de borde la RAU, en caso de ser switches de borde, por la limitante mostrada en la prueba de concepto 3, se cambia la dirección mac destino del paquete, seteando la dirección mac del QuaggaRouter al que se encuentra conectado el RAUSwitch.

Si no se cumple ninguno de los chequeos anteriores el controlador le indicará al switch que debe descartar el paquete. En la figura 30 se puede observar un diagrama de la toma de decisiones del controlador.

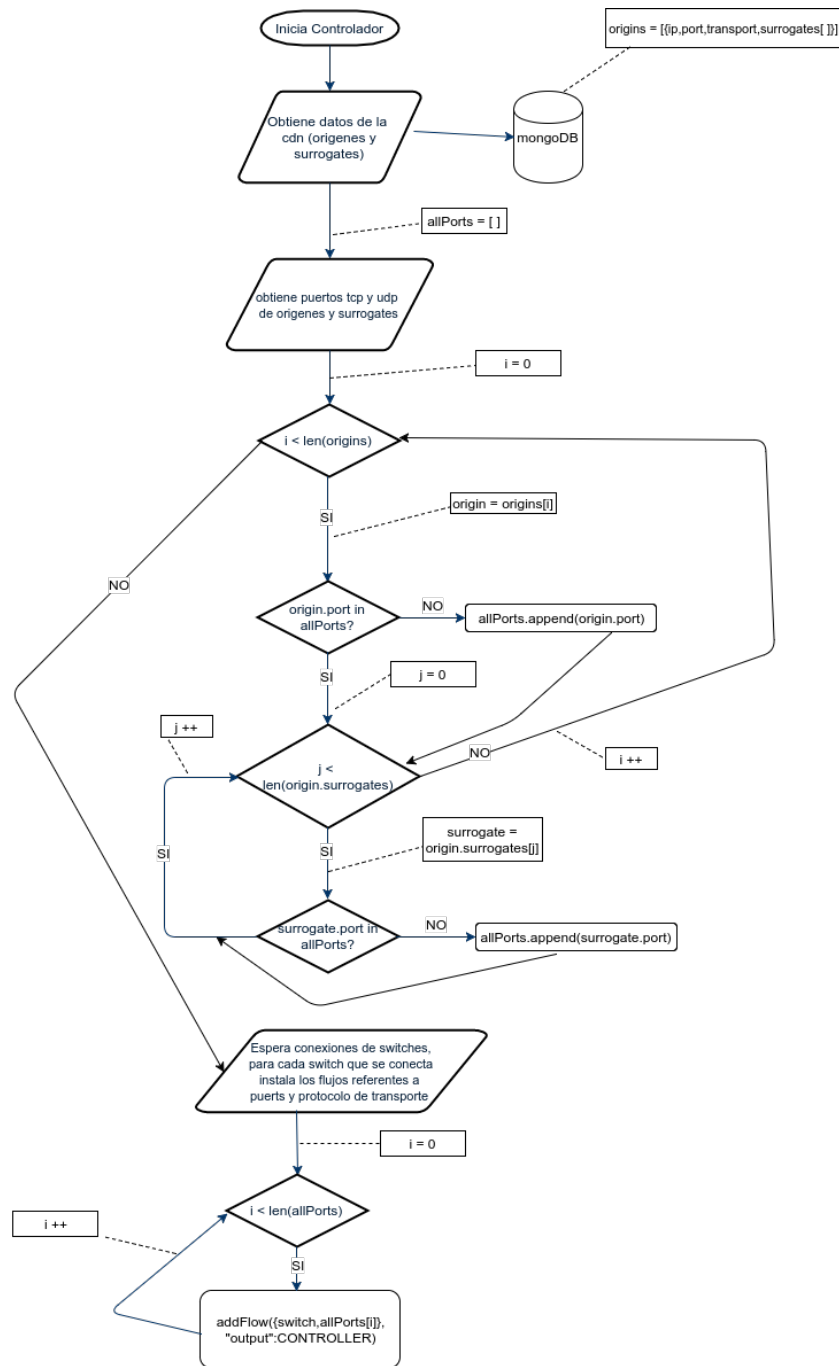


Figura 29: Inicio Ryu App

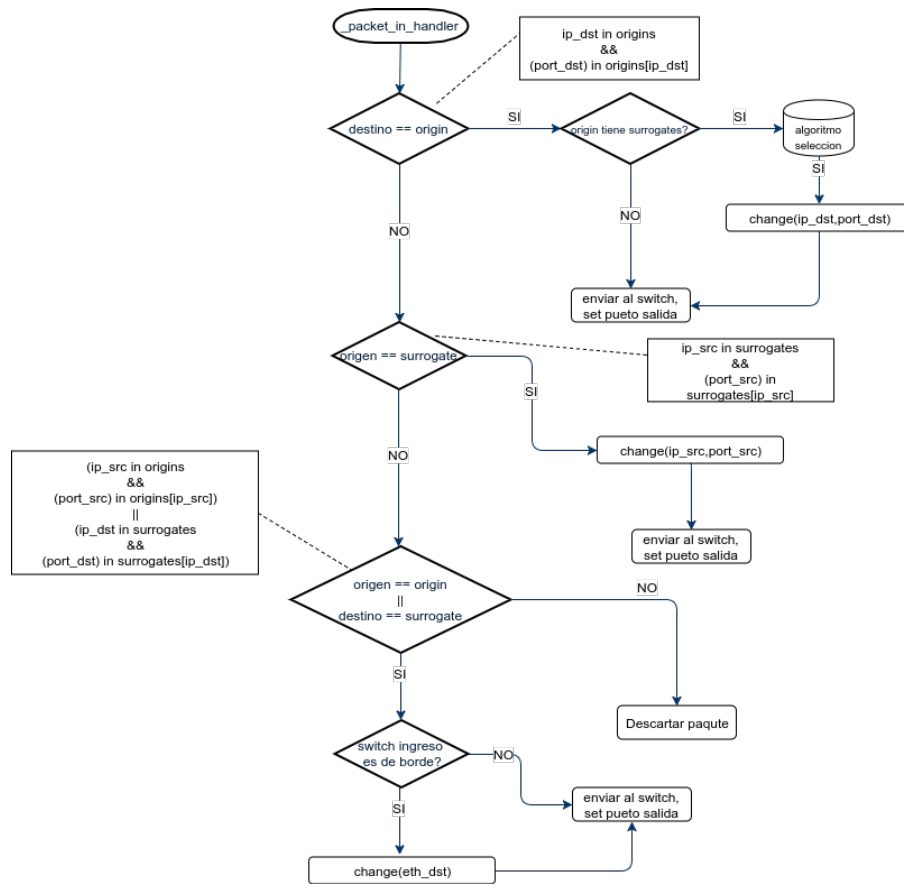


Figura 30: Captura paquete Ryu App

A continuación se mencionan algunas de las decisiones que se tomaron a la hora de la implementación:

- Para evitar bucles en las topologías, se incorpora la librería stp de ryu a la aplicación del controlador. El protocolo STP (Spanning Tree Protocol, protocolo del árbol de expansión) es un protocolo de Capa 2 que se ejecuta en bridges y switches. La especificación para STP se denomina IEEE 802.1D. El principal objetivo de STP es garantizar que se impida la creación de bucles en trayectos redundantes en la red.
- El controlador guardará para cada switch un mapeo entre dirección ip origen de un paquete y el puerto de entrada de ese paquete, esto se realiza, para que cuando se tenga que reenviar un paquete a esa dirección ip, el controlador sepa por qué puerto debe indicarle al switch que debe reenviar el paquete.
- Los Switches de borde de la SDN deben conocer la dirección mac de la interfaz del QuaggaRouter al que están conectados. De este modo cada vez que un RAUSwitch de borde debe reenviar un paquete (en los casos que no realiza cambios de ip, ni de origen ni de destino) setea como dirección mac destino, la dirección mac del QuaggaRouter al que está conectado. Esto se hizo así, ya que si no se realiza el cambio de dirección mac, cuando el QuaggaRouter recibe el paquete no lo tendrá en cuenta ya que no tiene como destino su dirección mac. Esto sucede ya que los RAUSwitches no funcionan como routers tradicionales cuando manejan flujos de la CDN.
- Como se mencionó anteriormente el controlador guarda por RAUSwitch un mapeo entre dirección ip origen y puerto de entrada. Cuando un switch debe reenviar un paquete el controlador se fija si para la ip destino del paquete tiene mapeado un puerto, si es así, le indica al switch por qué puerto debe reenviar el paquete. En caso contrario le indica que debe realizar un flood con ese paquete.
- Debido a las últimas dos decisiones descritas, si un RAUSwitch de borde debe reenviar un paquete a una dirección ip destino que no tiene mapeada con ningún puerto, este realizará un flood del paquete y le cambiará la dirección mac de destino. De esta forma si el destino del paquete no es conocido por el QuaggaRouter al que está conectado el RAUSwitch de borde, este reenviará el paquete a través de su default gateway, el cual es el RAUSwitch. De este modo el RAUSwitch volverá a recibir el paquete por un puerto de entrada distinto, para evitar estos problemas, se decidió que cuando un RAUSwitch de borde recibe un paquete desde el QuaggaRouter al que está conectado, chequea si la ip de origen del paquete está incluida en los mapeos que este tiene, si es así, se compara el puerto que tiene mapeado a la ip origen, con el puerto por el cual volvió a ingresar el paquete, si esos puertos son distintos el paquete es descartado.

- Como se mencionó anteriormente, cuando se instala un flujo se le setea un timeout de inactividad, en el caso de esta implementación se setea en 30 segundos a modo de ejemplo. Hay que estudiar más detalladamente el tiempo a setear, porque puede traer complicaciones con conexiones persistentes establecidos entre el cliente y un surrogate. Puede pasar que haya una conexión establecida entre un cliente y un surrogate, pero esa conexión quede inactiva por más tiempo que el tiempo de vida establecido para el flujo que conecta a ambos, y si el cliente activa la conexión y el flujo no existe, puede pasar que el controlador derive el cliente a otro surrogate y se pierda la conexión establecida.

8.2.4.2 Implementación para una red IP

A diferencia de la implementación para una red ethernet, en esta implementación el controlador solo se encargara de cambiar cabezales ip (tcp/udp). Para lograr esto (que el controlador no se encargue del enrutamiento), se debe utilizar un protocolo de enrutamiento externo (EGP), por esto se utiliza BGP, de esta forma los RAUSwitches tendran informacion sobre las rutas ip de servidores de origen, surrogates y clientes, de esta forma el controlador no tomará decisiones de enrutamiento. Al utilizar un protocolo de enrutamiento EGP, ya no es necesario que el controlador conozca la dirección mac de cada QuaggaRouter conectado a cada RAUSwitch de borde. Igual que en la implementación anterior se toma como base la aplicación RAUFlow (la cual es ejecutada por el RAU-Controller) desarrollada en [37] llamada "gui_topology.py". La aplicación ryu de esta implementación se anexa al trabajo (controller_app_implementacion_ip.py). Para que se pueda realizar la selección de mejor surrogate y aplicar los cambios de cabezal ip y los cambios de cabezal tcp/udp, se le debe indicar a los RAUSwitches de borde que el tráfico que ingrese por la interfaz física de borde sea enviada al controlador para ver si el tráfico entrante está dirigido a un servidor de origen. Una vez que el controlador analiza el tráfico, realiza los cambios de cabezales ip (DNAT) y envía el paquete a la interfaz virtual de borde para que realice el ruteo del tráfico según lo indica su tabla de ruteo. Debido a los inconvenientes que se encontraron en la prueba de concepto 4 (filtro rp cuando se realiza SNAT), el encargado de realizar SNAT será el RAUSWitch de borde al que se encuentra conectado el cliente (en la implementación para una red de capa 2, esto lo realizaba el RAUSWitch de borde al que estaba conectado el surrogate). Por esto es necesario que el controlador instale en cada RAUSwitch de borde un flujo que indique que todo lo que ingrese por la interfaz virtual de borde y tenga como origen algún puerto en donde algún servidor de la CDN este escuchando sea enviado al controlador. De esta forma el controlador analizara si el paquete tiene como origen un surrogate, si es así, instalará en el RAUSwitch el flujo que cambia la dirección ip origen, seteando la del servidor de origen al que se conectó el cliente originalmente. Para lograr los pasos anteriores, como se comentó en la presentación del modelo de datos, se utilizara un documento de base de datos que contiene a los RAUSwitches de borde y el número de puerto de la interfaz física de borde. De esta forma

cuando un RAUSwitch de borde se conecta al controlador se le instalaran los flujos mencionados anteriormente (todo lo que ingrese por la interfaz física de borde y tenga como destino u origen tcp /udp un puerto en que algún servidor de la cdn está escuchando, es enviado al controlador, este analiza lo que recibe y si va destinado a un servidor de origen o viene desde un surrogate, realiza los cambios en los cabezales ip y lo envía a la interfaz virtual de borde, para que esta lo rutee adecuadamente). De esta forma, el controlador SDN solo analiza paquetes entrantes a los RAUSwitches de borde por sus interfaces de borde (física y virtual).

8.2.4.2.1. Configuración BGP

Para que los RAUSwitches de la SDN tengan información de ruteo sobre servidores de origen, surrogates y clientes, es necesario configurar un protocolo de enrutamiento externo, para esto se configura el protocolo BGP a través del software de ruteo Quagga[45]. Para lograr configurar BGP, es necesario realizar modificaciones en las clases RAUSwitch y QUaggaRouter. Estas clases ahora recibirán como parámetro a la hora de su creación a que AS (Autonomous System) pertenecen y cuál es el AS al que están conectados (Para los RAUSwitchs, esto aplica solo si es de borde), de esta forma en los scripts de creación de la topología, es necesario pasar dicha información. También se modifican ambas clases para realizar la configuración de bgp en cada RAUSwitch y QuaggaRouter, y ejecutar los comandos para iniciar la ejecución de bgp a través de quagga. Se anexan estas modificaciones en el archivo rau_nodes.py”. Las rutas a servidores de origen, surrogates y clientes obtenidas por los RAUSwitches de borde a través de BGP, son distribuidas en la SDN con OSPF, de esta forma los RAUSwitchs que no son de borde y el RAUController tienen información sobre estas rutas.

8.2.4.3 Servicio RESTful

Por otro lado el controlador expone una northbound api (independientemente de si se implementa para una red ethernet o para una red ip), a través de un servicio RESTful, a través del cual el controlador CDN se pueda comunicar con el. Los métodos del servicio se pueden observar en la tabla 1.

Metodo	Entrada	Salida
addSurrogate	ipOrigin, ipSurrogate, portOrigin, portSurrogate, transport	boolean
deleteSurrogate	ipOrigin, ipSurrogate, portOrigin, portSurrogate, transport	boolean
addOrigin	ipOrigin, portOrigin, transport	boolean
deleteOrigin	ipOrigin, portOrigin, transport	boolean

Tabla 1: Métodos Controlador SDN.

Métodos:

- **addOrigin:** A través de este método se le indica al controlador SDN que fue agregado un nuevo servidor de origen a la RAU, se debe indicar la ip del nuevo servidor de origen, el puerto donde está escuchando y el protocolo de capa de transporte (udp o tcp). El resultado de este método es un booleano que indica si el controlador agregó con éxito el nuevo servidor de origen. El controlador además de agregar en memoria el nuevo servidor de origen, agrega de ser necesarios en cada switch, flujos que matchen con los puertos donde está levantado el servidor de origen. Este es un método POST y la url para su acceso es http://ip_controlador:8080/addOrigin.
- **deleteOrigin:** A través de este método se le indica al controlador SDN que fue eliminado un servidor de origen, el controlador elimina todo lo que tenga en memoria referente al servidor de origen, así como también los surrogates asociados, en caso de volver a agregar el servidor eliminado, se deberán volver a agregar todos los surrogates que tenga asociados. Este método recibe como entrada la ip del servidor de origen a eliminar, el puerto donde está escuchando y el protocolo de capa de transporte (udp o tcp). El resultado de este método es un booleano que indica si el controlador eliminó con éxito el servidor de origen indicado. Este es un método POST y la url para su acceso es http://ip_controlador:8080/deleteOrigin.
- **addSurrogate:** A través de este método se le indica al controlador SDN que fue agregado un nuevo surrogate a la RAU, se debe indicar la ip del nuevo surrogate, el puerto donde está escuchando, la dirección ip del servidor de origen que está replicando, el puerto del servidor de origen y el protocolo de capa de transporte (udp o tcp) del servicio a replicar. El resultado de este método es un booleano que indica si el controlador agregó con éxito el nuevo surrogate. El controlador además de agregar en memoria el nuevo surrogate y asociarlo a el servidor de origen correspondiente, agrega de ser necesarios en cada switch, flujos que matchen con los puertos donde está levantado el nuevo surrogate. Este es un método POST y la url para su acceso es http://ip_controlador:8080/addSurrogate.
- **deleteSurrogate:** A través de este método se le indica al controlador SDN que fue eliminado un surrogate, el controlador elimina todo lo que tenga en memoria referente al surrogate y lo desasocia del servidor de origen correspondiente. Este método recibe como entrada la ip del surrogate a eliminar, el puerto donde está escuchando, la dirección ip del servidor de origen que está replicando, el puerto del servidor de origen y el protocolo de capa de transporte (udp o tcp) del servicio replicado. El resultado de este método es un booleano que indica si el controlador eliminó con éxito el surrogate indicado. Este es un método POST y la url para su acceso es http://ip_controlador:8080/deleteSurrogate.

8.2.5. Algoritmo de selección surrogate

En esta implementación se eligió implementar un algoritmo no adaptativo (dado que es más sencilla su implementación, y se realizará a modo de ejemplo), en donde se seleccionara el mejor surrogate dependiendo de la distancia (cantidad de saltos) entre el cliente y los surrogates. Para esto se utilizará un algoritmo que calcule el camino más corto, dependiendo de los saltos entre el cliente y los surrogates. Se utilizará el algoritmo de Dijkstra para el cálculo del camino más corto, se adapta la versión del algoritmo desarrollado en [46], para que sea compatible con este trabajo. Como el foco de este trabajo no era el algoritmo de selección se eligió el algoritmo antes mencionado, por tal motivo no se profundizó en el desarrollo de un mejor algoritmo. Se podría destinar todo un trabajo a realizar un estudio detallado del algoritmo de selección.

Esta implementación hace uso de las tablas de ruteo del controlador, para ubicar los switches de borde conectados a clientes y surrogates. Como se explica en el trabajo que implementa el entorno virtual [37], es necesario ejecutar el comando "python telnetRouters.py" desde el nodo controlador. Este script Python es el que se invocaría automáticamente si se detectara un cambio en la topología (en el entorno virtual se lo llama manualmente) y se encarga de consultar la base de datos topológica de OSPF mediante Telnet, parsear la información y enviarla a la aplicación RAUFlow. Luego de recibir la topología, la aplicación RAUFlow todavía necesita la siguiente información de cada RAUSwitch: nombres de interfaces, direcciones IP y direcciones MAC. Para obtener esta información, se conecta automáticamente con el nodo que tiene levantado el Web Service que hace disponible la información de cada nodo. De esta forma la aplicación RAUFlow tiene información de cada RAUSwitch de la red (ips, interfaces, etc.).

La aplicación RAUFlow ejecutará un algoritmo de descubrimiento de la topología de la SDN, utilizando el protocolo LLDP, de esta forma, el controlador es capaz de conocer todos los links existentes en la SDN y así, puede determinar el mejor surrogate.

Para que la aplicación RAUFlow obtenga información de rutas de la red, es necesario instalar el módulo de python "pyroute2". Para obtener información sobre el módulo y su forma de instalación acceder a [47]. Como la implementación corre el protocolo BGP, el controlador conoce rutas a clientes y switches. Gracias a esto el controlador conoce el gateway para acceder a clientes y surrogates, este gateway es el RAUSwitch de borde que conecta a un cliente o a un surrogate a la red de la implementación. De esta forma, cuando llega a la aplicación RAUFlow un pedido a un servidor de origen, está obtendrá todos los RAUSwitches de borde que conectan a los surrogates asociados a ese servidor de origen y el RAUSwitch de borde que conecta al cliente que realiza la petición. Con todos los RAUSwitches de borde obtenidos la aplicación RAUFlow aplicará ejecutará el algoritmo dijkstra para determinar el surrogate más cercano y obtendrá el mejor RAUSwitch de borde al que debe redirigir el pedido. Con

ese RAUSwitch de borde se obtiene el surrogate elegido (surrogate conectado al RAUSwitch de borde) y se realiza el cambio de cabezales ip.

En la prueba de concepto 5 se muestra el funcionamiento del algoritmo de selección.

8.2.6. Controlador CDN

El controlador CDN, se encarga de administrar los distintos servidores de la CDN, expone un servicio RESTful que es utilizado por la web de administración para dar de alta servidores de origen y surrogates, así como también eliminarlos. Se encarga de nutrir la base de datos mongoDB según lo disponga el administrador de la CDN a través del sitio web. Es el encargado de consumir la northbound api expuesta por el controlador SDN, para informarle de cambios en la CDN. El controlador CDN fue desarrollado en node.JS y es ejecutado fuera del entorno virtual, se ejecuta en la máquina encargada de levantar la VM que contiene la extensión de MiniNet. Todo su código fuente se encuentra en los anexos del trabajo.

En la tabla 2 se pueden observar los métodos del servicio RESTful expuestos por el controlador CDN, que serán consumidos por la página web de administración.

Metodo	Entrada	Salida
addSurrogate	ipOrigin, ipSurrogate, portOrigin, portSurrogate, transport	boolean
deleteSurrogate	ipOrigin, ipSurrogate, portOrigin, portSurrogate, transport	boolean
addOrigin	ipOrigin, portOrigin, transport	boolean
deleteOrigin	ipOrigin, portOrigin, transport	boolean
getCDN		JSON

Tabla 2: Métodos Controlador CDN.

Métodos:

- **addOrigin:** A través de este método se le indica al controlador CDN que se va a agregar un nuevo servidor de origen a la RAU, se debe indicar la ip del nuevo servidor de origen, el puerto donde está escuchando y el protocolo de capa de transporte (udp o tcp). El resultado de este método es un booleano que indica si el servidor de origen fue agregado con éxito. El controlador CDN agrega a la base de datos la información del nuevo servidor de origen y a través de la northbound api le informa al controlador SDN que se agregó un nuevo servidor de origen. Este método es invocado por el administrador CDN a través de la página web de administración de la CDN.
- **deleteOrigin:** A través de este método se le indica al controlador CDN que se va a eliminar un servidor de origen de la RAU, se debe indicar la ip del nuevo servidor de origen, el puerto donde está escuchando y el

protocolo de capa de transporte (udp o tcp). El resultado de este método es un booleano que indica si el servidor de origen fue eliminado con éxito. El controlador CDN elimina de la base de datos la información del servidor de origen en cuestión y a través de la northbound api le informa al controlador SDN que se eliminó un servidor de origen. Este método es invocado por el administrador CDN a través de la página web de administración de la CDN.

- **addSurrogate:** A través de este método se le indica al controlador CDN que se va a agregar un nuevo surrogate a la RAU, se debe indicar la ip del nuevo surrogate, el puerto donde está escuchando, la dirección ip del servidor de origen que está replicando, el puerto del servidor de origen y el protocolo de capa de transporte (udp o tcp) del servicio a replicar. El resultado de este método es un booleano que indica si el controlador agregó con éxito el nuevo surrogate. El controlador además de agregar en la base de datos el nuevo surrogate y asociarlo a el servidor de origen correspondiente, le informa al controlador SDN a través de la northbound api que se agregó un nuevo surrogate. Este método es invocado por el administrador CDN a través de la página web de administración de la CDN.
- **deleteSurrogate:** A través de este método se le indica al controlador CDN que se va a eliminar un surrogate de la RAU, se debe indicar la ip del surrogate, el puerto donde está escuchando, la dirección ip del servidor de origen que está replicando, el puerto del servidor de origen y el protocolo de capa de transporte (udp o tcp) del servicio a replicar. El resultado de este método es un booleano que indica si el controlador eliminó con éxito el surrogate. El controlador además de eliminar en la base de datos el surrogate, le informa al controlador SDN a través de la northbound api que se eliminó un surrogate. Este método es invocado por el administrador CDN a través de la página web de administración de la CDN.
- **getCDN:** A través de este modo el controlador CDN proporciona información sobre servidores de origen y sus correspondientes surrogates.

8.2.7. Administración Web

Para la administración de la CDN, se desarrolla un sitio web, a través del cual se puede consultar servidores de origen y sus surrogates asociados, se puede dar de alta y eliminar servidores de origen, y se puede dar de alta y eliminar surrogates. Para realizar estas operaciones se conecta con el controlador CDN, a través de los servicios RESTful que este expone. Fue desarrollado utilizando AngularJS 2, y es ejecutado en la máquina encargada de levantar el entorno virtual. En la prueba de concepto 5 se muestra su forma de uso, así como también la forma de instalar, compilar y levantar el sitio web. Todo su código fuente se encuentra en los anexos del trabajo.

8.2.8. Otras configuraciones

Para poder realizar pruebas sobre las arquitecturas implementadas fueron necesarias realizar algunas configuraciones extras, las cuales se deben tener en cuenta a la hora de ejecutar el entorno virtual. Como se comentó anteriormente, la base de datos mongoDB, el servidor web que hostea el sitio de administración y el controlador CDN, están fuera del entorno virtual, corriendo en la máquina que lo ejecuta. Por este motivo es necesario mapear a al controlador SDN una interfaz de red física, externa al entorno virtual. De esta forma el controlador SDN se podrá comunicar con la base de datos y el controlador CDN podrá consumir la northbound api expuesta por el controlador SDN. Para lograr lo antes mencionado es necesario modificar el script que se encarga de levantar la topología (start.py), desarrollado en [37]. El script obtiene el primer host de la lista de hosts (RAUController es una extensión de la clase host) y realiza el mapeo. Por esto es necesario, a la hora de crear la topología y asignar nombres de hosts(hosts, RAUSwitches, QueaggaRouters, RAUController), que el RAUController sea el primero en orden alfabético.

8.3. Despliegue Arquitectura propuesta en RAU

Para lograr el despliegue de la arquitectura propuesta en la RAU es necesario aplicar los siguientes puntos:

- Se debe configurar y ejecutar OSPF en cada uno de los RAUSwitches y en el RAUController
- SE debe configurar y ejecutar BGP en cada uno de los RAUSwitches de borde
- Se deben instalar todos los componentes para poder administrar servidores de origen y surrogates (web server, mongoDB, hosteo del sitio web).
- Se debe cargar y ejecutar en el RAUController la aplicación ryu desarrollada en este trabajo.
- Se deben registrar RAUSwitches de borde y el número de puerto de la interfaz de borde de cada uno de estos RAUSwitches de borde en la base de datos de la arquitectura.

Con la aplicación de los puntos mencionados ya es posible poner en marcha la arquitectura propuesta por este trabajo.

9. Conclusiones

El objetivo principal de este trabajo era estudiar distintas alternativas para el despliegue de un CDN en la RAU, basándose en el prototipo SDN de la misma.

Se estudiaron distintos trabajos relacionados, en los cuales el denominador común es el aprovechamiento del controlador SDN para realizar el encaminamiento al surrogate más cercano. En todas estas soluciones estudiadas los clientes se comunican con algún componente central de la arquitectura el cual le informa al cliente el mejor surrogate. A diferencia de las soluciones existentes estudiadas, la arquitectura propuesta por este trabajo, logra que el cliente no deba consultar por el mejor surrogate asociado a un servidor de origen, el cliente dirigirá todos sus pedidos al servidor de origen y la red (gracias a la visión global del controlador SDN) será la encargada de redirigir esos pedidos al surrogate adecuado de forma transparente para el cliente.

Como se mencionó anteriormente, el controlador SDN de la arquitectura propuesta por este trabajo, suplanta a los mecanismos y algoritmos de encaminamiento de una CDN tradicional, en el punto 7.3 se presentaron las ventajas de utilizar el controlador SDN en vez de los mecanismos tradicionales. Con esta solución no es necesario contar con servicios extras en la red para lograr determinar el mejor surrogate para un pedido determinado.

También se presentó la principal desventaja de la solución desarrollada, la cual muestra un único punto de falla en la arquitectura, para lograr solucionar esta desventaja, se podría implementar una arquitectura con múltiples controlador SDN y así distribuir la carga entre ellos. Esto también mejoraría la performance de la solución, ya que con la solución propuesta un único controlador se encarga de procesar todos los pedidos de los clientes de la red.

En este trabajo no se estudio el rendimiento de la solución propuesta, lo cual se tendría que realizar en futuros trabajos sobre esta arquitectura, también se deberá comparar resultados de performance, con la performance de los mecanismos de encaminamiento de una CDN tradicional.

Otra conclusión interesante a la que se puede llegar viendo lo presentado en el punto 8.3, es que esta solución es fácil de desplegar en una red SDN existente.

Otro objetivo del trabajo era relevar requerimientos específicos para la RAU, lo cual fue muy difícil de conseguir, ya que no se logró concretar reuniones con los distintos encargados de generadores de contenidos de la RAU, solo se pudo realizar una reunión con el encargado del proyecto OpenFING (Fernando Carpani), el cual es un proyecto estudiantil que pretende crear una biblioteca multimedia colaborativa, con videos de los cursos dictados en la facultad. Por estos motivos, se decidió proponer una arquitectura genérica CDN-SDN, sin tener mucho en cuenta los requerimientos reales de la RAU.

Respecto a la administración de la arquitectura, si se quisiera poner en producción lo propuesto en este trabajo, se deben realizar mejoras en la página web administrativa, ya que la versión entregada en este trabajo carece de algunos controles a la hora de agregar surrogates o servidores de origen, como por

ejemplo chequeo de existencia de surrogates y servidores de origen.

Bibliografía

- [1] “Red Académica Uruguay.” <http://www.rau.edu.uy>.
- [2] Viotti, E. and Amaro, R, *Routers Reconfigurables de Altas Prestaciones*. 2015.
- [3] Mathias Duarte, *Herramientas de simulación/emulación SDN*. 2016.
- [4] Joe Silver, *SDN 101: What It Is, Why It Matters, and How to Do It*. Cisco Blogs, 2013.
- [5] thegreek80, “Investigación de redes sdn.” <http://principletechnologica.com/category/principia-technologica/redes/sdn>, 2014.
- [6] Ramón Jesús Millán Tejedor, *SDN: el futuro de las redes inteligentes*. Conectronica n° 179, GM2 Publicaciones Técnicas, 2014.
- [7] Open Networking Foundation, *OpenFlow Switch Specification*. Open Networking Foundation, 2014.
- [8] “Akamai.” <http://www.akamai.com>.
- [9] Kanoha, “Single server (left) versus Content Delivery Network (CDN) (right).” https://upload.wikimedia.org/wikipedia/commons/f/f9/NCDN_CDN.png.
- [10] R. Buyya, M. Pathan, and A. Vakali, *Content Delivery Networks*. Springer Publishing Company, Incorporated, 1st ed., 2008.
- [11] Incapsula, “The Essential CDN Guide.” <https://www.incapsula.com/cdn-guide>.
- [12] “EdgeStream.” <http://www.edgestream.com>.
- [13] “Limelight Networks.” <http://www.limelight.com>.
- [14] “Mirror Image.” <http://www.mirror-image.com/site>.
- [15] “CodeeN.” <http://codeen.cs.princeton.edu>.
- [16] “Coral.” <http://www.coralcdn.org>.
- [17] “Globule.” <http://www.globule.org>.
- [18] “FCAN.” https://en.wikipedia.org/wiki/Flash_crowds_alleviation_network.
- [19] Giancarlo Fortino, Carlos E Palau, Wilma Russo, Manuel Esteve, “The comodin system: A cdn-based platform for cooperative media on-demand on the internet,” Jan. 2004.

- [20] Gálvez Huerta, Santiago David, “Diseño de un nodo con tecnología cloud-oriented content delivery network, utilizando la integración con modelos de servicio de cloud computing, para el datacenter de un proveedor de servicios de contenido,” May 2016.
- [21] G. Peng, “Cdn: Content distribution network,” 2008.
- [22] Benjamín Molina Moreno, *Estudio, análisis y desarrollo de una red de distribución de contenido y su algoritmo de redirección de usuarios para servicios web y streaming*. Universidad politecnica de Valencia, 2013.
- [23] S. Sivasubramanian, G. Pierre, M. Steen, and G. Alonso, “G.: Analysis of caching and replication strategies for web applications,” *IEEE Internet Computing*, vol. 11, pp. 60–66, 2007.
- [24] J. Kangasharju, J. Roberts, and K. W. Ross, “Object replication strategies in content distribution networks,” *Comput. Commun.*, vol. 25, pp. 376–383, Mar. 2002.
- [25] S. S. H. Tse, “Approximate algorithms for document placement in distributed web servers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 489–496, June 2005.
- [26] G. Pallis, A. Vakali, K. Stamos, A. Sidiropoulos, D. Katsaros, and Y. Manolopoulos, “A latency-based object placement approach in content distribution networks,” in *Third Latin American Web Congress (LA-WEB’2005)*, pp. 8 pp.–, Oct 2005.
- [27] G. Pallis, K. Stamos, A. Vakali, D. Katsaros, and A. Sidiropoulos, “Replication based on objects load under a content distribution network,” in *22nd International Conference on Data Engineering Workshops (ICDEW’06)*, pp. 53–53, 2006.
- [28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 149–160, Aug. 2001.
- [29] S. Bakiras and T. Loukopoulos, “Combining replica placement and caching techniques in content distribution networks,” *Comput. Commun.*, vol. 28, pp. 1062–1073, June 2005.
- [30] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, “Web caching with consistent hashing,” in *Proceedings of the Eighth International Conference on World Wide Web, WWW ’99*, (New York, NY, USA), pp. 1203–1213, Elsevier North-Holland, Inc., 1999.
- [31] B. Huffaker, M. Fomenkov, D. Plummer, D. Moore, and k. claffy, “Distance Metrics in the Internet,” in *IEEE International Telecommunications Symposium (ITS)*, (Brazil), pp. 200–202, IEEE, Sep 2002.

- [32] J. . . . Chandrakanth, P. . . . Chollangi, and C.-H. . . . Lung, “Content distribution networks using software defined networks,” in *Proceedings - 2nd International Conference on Trustworthy Systems and Their Applications, TSA 2015*, no. Proceedings - 2nd International Conference on Trustworthy Systems and Their Applications, TSA 2015, ((1)Electrical and Computer Engineering, University of Ottawa), pp. 44–50, 2015.
- [33] L. Chen, M. Qiu, W. Dai, and N. Jiang, “Supporting high-quality video streaming with sdn-based cdns,” *Journal of Supercomputing*, pp. 1–15, 2016.
- [34] T. Yuan, “Desarrollo de servicios de distribución de contenidos multimedia en redes corporativas con tecnología sdn.” 2015.
- [35] J. Lai, Q. Fu, and T. Moors, “Using sdn and nfv to enhance request re-routing in isp-cdn collaborations,” *Computer Networks*, vol. 113, pp. 176 – 187, 2017.
- [36] F. Qin, Z. Zhao, and H. Zhang, “Optimizing routing and server selection in intelligent sdn-based cdn,” 2016. cited By 0.
- [37] Santiago Vidal, *Escalabilidad de redes definidas por software en la red académica*. 2016.
- [38] “MiniNet.” <http://mininet.org/>. [Sitio web oficial].
- [39] Te-Yuan Huang, Vimalkumar Jeyakumar Bob Lantz, Brian O’Connor Nick Feamster Keith Winstein, Anirudh Sivaraman, “Teaching computer networking with mininet,” *ACM SIGCOMM 2014*.
- [40] Ryan Lienhart Dahl, “Node.js.” <https://nodejs.org>.
- [41] mongoDB, “mongoDB.” <https://www.mongodb.com>.
- [42] Google, “AngularJS.” <https://angularjs.org/>.
- [43] Santiago Vidal, “App RAUFlow, .^{Escalabilidad de Redes Definidas por Software en la Red Académica}.” <https://github.com/santiagovidal/LiveCode>.
- [44] mongoDB, “Python MongoDB Drivers.” <https://docs.mongodb.com/ecosystem/drivers/python/>.
- [45] Paul Jakma, “Quagga Routing Suite.” <http://www.nongnu.org/quagga/>.
- [46] “Algoritmo Dijkstra Python.” <https://gist.github.com/kachayev/5990802>.
- [47] “pyroute2.” <https://github.com/svinota/pyroute2>.
- [48] “*rpfilter*.” <https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt>.

- [49] “Instalación mongoDB.” <https://www.digitalocean.com/community/tutorials/como-instalar-mongodb-en-ubuntu-16-04-es>.
- [50] “Anexos Códigos fuente.” <https://github.com/elchobo5/tesisIngenieria>.

A. Pruebas de concepto

A.1. Prueba 1

La primer prueba de concepto consiste en aplicar cambios en la direcciones ip y mac destino a un request HTTP GET y direcciones ip y mac origen en la respuesta. Para esto se utilizara mininet como emulador, en el cual se emula una topología single (un único switch conectado a todos los hosts), la cual consiste en un switch openflow (OpenVSwitch, s1), el controlador sdn de referencia de mininet y tres hosts, de los cuales uno oficiara de cliente (h3), otro de surrogate (h2) y el último de servidor de origen (h1). En el host que oficiara de servidor de origen, no se levantará ningún servidor web, mientras que en host surrogate se levantará un servidor web, en el cual su index.html tendrá la palabra "SURROGATE". Esto para verificar que el cliente está obteniendo el contenido del surrogate. Si el contenido se fuera a obtener del origen, no se debería poder establecer la conexión HTTP, ya que no habrá ningún servidor web levantado. La topología en esta prueba representa un entorno LAN, la prueba se realiza para verificar el cambio de campos en los cabezales ip y ethernet. Se instalarán manualmente los flujos para aplicar los cambios de direcciones ip y mac.

La topología de la red es la siguiente:

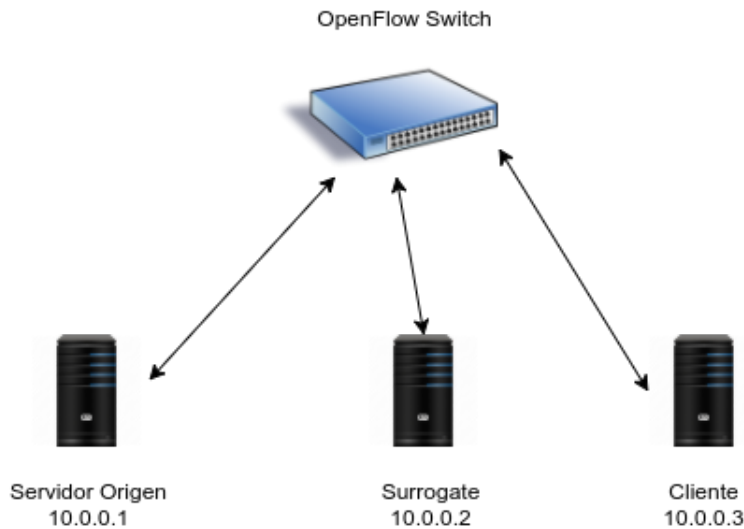


Figura 31: Prueba Concepto 1

Los pasos que se siguen en las pruebas son los siguientes:

- 1. Se levanta la topología indicada en la figura 32 con mininet utilizando el siguiente comando:

```
sudo mn --topo=single,3 --switch ovs,protocols=
OpenFlow10 --controller ref --mac
```


Se utiliza `-mac` para que tener direcciones mac correlativas y no rándómic-
as en las interfaces del switch.

- 2. Se instalan en el switch openflow los siguientes flujos:

```
$ sudo ovs-ofctl add-flow s1 "table=0,priority  
=65535,dl_dst=00:00:00:00:00:01,nw_dst  
=10.0.0.1,nw_tos=0,tcp,tp_dst=80_actions=  
mod_nw_dst=10.0.0.2,mod_dl_dst  
=00:00:00:00:00:02,output:2"
```

```
$ sudo ovs-ofctl add-flow s1 "table=0,priority  
=65535,dl_src=00:00:00:00:00:02,nw_src  
=10.0.0.2,nw_tos=0,tcp,tp_src=80_actions=  
mod_nw_src=10.0.0.1,mod_dl_src  
=00:00:00:00:00:01,output:3"
```

El primero indica que si se recibe un flujo con direccion ip de destino 10.0.0.1 (origen) y puerto de destino 80, se apliquen acciones de cambio de cabezal ip, con dirección destino 10.0.0.2 (surrogate), cambio de mac destino a 00:00:00:00:00:02 (surrogate) y se setea el puerto de salida 2 para esos paquetes, ya que es el puerto al que está conectado el surrogate en el switch. El segundo indica que si se recibe un flujo con direccion ip de origen 10.0.0.2 (surrogate) y puerto de origen 80, se apliquen acciones de cambio de cabezal ip, con dirección origen 10.0.0.1 (origen), cambio de mac origen a 00:00:00:00:00:01 (origen) y se setea el puerto de salida 3 para esos paquetes, ya que es el puerto al que está conectado el cliente en el switch.

- 3. Se crea el archivo `index.html` en el surrogate con el contenido indicado mas arriba.
- 4. Se levanta el servidor web en mininet en el host que oficia de surrogate con el siguiente comando:

```
mininet> h2 python -m SimpleHTTPServer 80 &
```

- 5. El cliente (h3) realiza un HTTP REQUEST al servidor de origen, utilizando el comando `wget` desde mininet, de la siguiente manera:

```
mininet> h3 wget -O - h1
```

- 6. El switch openflow aplica la acción referida al flujo con ip destino 10.0.0.1, puerto destino 80 y realiza el cambio de ip destino y mac destino.
- 7. El surrogate recibe el request y sirve el contenido `index.html` al cliente.
- 8. El switch openflow aplica la acción referida al flujo con ip origen 10.0.0.2, puerto origen 80 y realiza el cambio de ip origen y mac origen.

- 9. El cliente obtiene el contenido.

Pruebas:

Se levanta la topología y se consultan los flujos configurados en el switch, donde se observa que la tabla de flujos esta inicialmente vacia.

```
mininet@mininet-vm:~$ sudo mn --topo=single,3 --switch ovs,protocols=OpenFlow10 --controller ref --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figura 32: Inicio topología 1

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
mininet@mininet-vm:~$
```

Figura 33: Tabla de flujos inicial

Una vez levantada la topologia, se levanta el servidor web en el surrogate (h2 - 10.0.0.2), pero se intenta realizar un HTTP GET desde el cliente (h3 - 10.0.0.3) al servidor de origen (h1 - 10.0.0.1), este request es rechazado ya que no existe ningun servidor web levantado en ese host (servidor origen).

```
mininet> h3 wget -O - h1
--2017-02-15 19:34:49-- http://10.0.0.1/
Connecting to 10.0.0.1:80... failed: Connection refused
mininet>
```

Figura 34: HTTP Request rechazado

949	8.488137000	00:00:00 00:00:03		ARP	44 Who has 10.0.0.1? Tell 10.0.0.3
950	8.488382000	00:00:00 00:00:03	Broadcast	OF 1.0	128 of packet in
951	8.489297000	127.0.0.1	127.0.0.1	OF 1.0	92 of packet out
952	8.489318000	127.0.0.1	127.0.0.1	TCP	68 60315 > 6633 [ACK] Seq=69 Ack=33 Win=86 Len=0 TSval=284711 T
953	8.489408000	00:00:00 00:00:03		ARP	44 Who has 10.0.0.1? Tell 10.0.0.3
954	8.489413000	00:00:00 00:00:03		ARP	44 Who has 10.0.0.1? Tell 10.0.0.3
955	8.489415000	00:00:00 00:00:03		ARP	44 Who has 10.0.0.1? Tell 10.0.0.3
956	8.489430000	00:00:00 00:00:01		ARP	44 10.0.0.1 is at 00:00:00:00:00:01
957	8.489657000	00:00:00 00:00:01	00:00:00 00:00:03	OF 1.0	128 of packet in
958	8.490178000	127.0.0.1	127.0.0.1	OF 1.0	148 of flow add
959	8.490279000	00:00:00 00:00:01		ARP	44 10.0.0.1 is at 00:00:00:00:00:01
960	8.490288000	10.0.0.3	10.0.0.1	TCP	76 39095 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
961	8.490653000	10.0.0.3	10.0.0.1	OF 1.0	160 [TCP Out-of-Order] of packet in
962	8.491078000	127.0.0.1	127.0.0.1	OF 1.0	148 of flow add
963	8.491174000	10.0.0.3	10.0.0.1	TCP	76 [TCP Out-of-Order] 39095 > http [SYN] Seq=0 Win=29200 Len=0
964	8.491193000	10.0.0.1	10.0.0.3	TCP	56 http > 39095 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
965	8.491537000	10.0.0.1	10.0.0.3	OF 1.0	140 of packet in
966	8.491984000	127.0.0.1	127.0.0.1	OF 1.0	148 of flow add
967	8.492078000	10.0.0.1	10.0.0.3	TCP	56 http > 39095 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figura 35: Captura HTTP Request rechazado

En la captura se observa el intento de inicio de conexión HTTP por parte del cliente al servidor, pero este termina la conexión con un mensaje RST. Luego se instalan los flujos en el switch con los comandos mencionados mas arriba, entonces, la tabla de flujos del switch (s1) queda de la siguiente manera:

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.544s, table=0, n_packets=0, n_bytes=0, idle_age=4, p
 riority=65535,tcp,dl_src=00:00:00:00:00:02,nw_src=10.0.0.2,nw_tos=0,tp_src=8
 0 actions=mod_nw_src:10.0.0.1,mod_dl_src:00:00:00:00:00:01,output:3
 cookie=0x0, duration=18.059s, table=0, n_packets=0, n_bytes=0, idle_age=18,
 priority=65535,tcp,dl_dst=00:00:00:00:00:01,nw_dst=10.0.0.1,nw_tos=0,tp_dst
 =80 actions=mod_nw_dst:10.0.0.2,mod_dl_dst:00:00:00:00:00:02,output:2
mininet@mininet-vm:~$
```

Figura 36: Tabla de flujos s1

Una vez configurados los flujos, se vuelve a realizar el HTTP GET Request desde el cliente (h3) al servidor de origen (h1), y esta vez se obtiene un resultado satisfactorio.

```

mininet>
mininet> h3 wget -O - h1
--2017-02-16 10:07:39-- http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 129 [text/html]
Saving to: 'STDOUT'

0% [          ] 0          --.-K/s
      <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<title>SURROGATE</title>
<body>
<h1>SURROGATE</h1>
</body>
</html>
100%[=====] 129          --.-K/s
    in 0.02s

2017-02-16 10:07:39 (7.72 KB/s) - written to stdout [129/129]
mininet>

```

Figura 37: HTTP Request OK

487	6.802697000	00:00:00_00:00:03		ARP	44 Who has 10.0.0.1?
488	6.802702000	00:00:00_00:00:03		ARP	44 Who has 10.0.0.1?
489	6.802704000	00:00:00_00:00:03		ARP	44 Who has 10.0.0.1?
490	6.802721000	00:00:00_00:00:01		ARP	44 10.0.0.1 is at 00:
491	6.802978000	00:00:00_00:00:01	00:00:00_00:00:03	OF 1.0	128 of packet in
492	6.803877000	127.0.0.1	127.0.0.1	OF 1.0	148 of flow add
493	6.804096000	00:00:00_00:00:01		ARP	44 10.0.0.1 is at 00:
494	6.804108000	10.0.0.3	10.0.0.1	TCP	76 39100 > http [SYN]
495	6.804463000	10.0.0.3	10.0.0.2	TCP	76 39100 > http [SYN]
496	6.804492000	00:00:00_00:00:02		ARP	44 Who has 10.0.0.3?
497	6.804883000	00:00:00_00:00:02	Broadcast	OF 1.0	128 of packet in
498	6.805522000	127.0.0.1	127.0.0.1	OF 1.0	92 of packet out

Figura 38: Captura HTTP Request OK

503	6.805881000	00:00:00_00:00:03	00:00:00_00:00:02	OF 1.0	128 of packet in
504	6.806360000	127.0.0.1	127.0.0.1	OF 1.0	148 of flow add
505	6.806523000	00:00:00_00:00:03		ARP	44 10.0.0.3 is at 00:00:00:
506	6.806534000	10.0.0.2	10.0.0.3	TCP	76 http > 39100 [SYN, ACK]
507	6.806853000	10.0.0.1	10.0.0.3	TCP	76 http > 39100 [SYN, ACK]
508	6.806878000	10.0.0.3	10.0.0.1	TCP	68 39100 > http [ACK] Seq=1
509	6.806883000	10.0.0.3	10.0.0.2	TCP	68 39100 > http [ACK] Seq=1
510	6.807532000	10.0.0.3	10.0.0.1	HTTP	174 GET / HTTP/1.1
511	6.807543000	10.0.0.3	10.0.0.2	HTTP	174 GET / HTTP/1.1
512	6.807571000	10.0.0.2	10.0.0.3	TCP	68 http > 39100 [ACK] Seq=1
513	6.807577000	10.0.0.1	10.0.0.3	TCP	68 http > 39100 [ACK] Seq=1

Figura 39: Captura HTTP Request OK 2

542	6.888636000	10.0.0.2	10.0.0.3	HTTP	197 HTTP/1.0 200 OK (
543	6.888644000	10.0.0.1	10.0.0.3	HTTP	197 HTTP/1.0 200 OK (

Figura 40: Captura HTTP Request OK 3

En la primer imagen de capturas se observa el paquete SYN HTTP para iniciar la conexión, que tiene como origen h3 y destino h1, con puerto de origen 39100. Luego se observa el mismo paquete pero con destino h2, esto es por los cambios que realiza el switch openflow. En la segunda imagen de capturas se observa el establecimiento de la conexión (SYN ACK desde h2), de la misma forma los paquetes tienen origen h2 y destino h3, y el switch cambia el origen de los mismos a h1. Finalmente en la tercer imagen, se observa el response al GET, sucediendo lo mismo que en los otros casos, los paquetes tienen origen h1 y destino h3 y el switch cambia el origen a h2.

A.2. Prueba 2

La segunda prueba de concepto consta de una topología de red similar a la primera, salvo que en esta se tendrá un controlador SDN externo (ryu, se anexa la aplicación que ejecutará el controlador, prueba2.py), el cual, agregará únicamente dos flujos en el switch. Estos flujos son los correspondientes a los cambios de direcciones ip y mac. Para el resto de paquetes pertenecientes a otros flujos, el controlador indicará al switch por qué puerto enviar el paquete, pero no en este caso, no agregará ningún flujo. En esta prueba se utilizara la version 1.3 del protocolo OpenFlow, por otro lado, el controlador se ejecuta en la máquina donde corre la maquina virtual que incluye MiniNet. La topología en esta prueba representa un entorno LAN, la prueba se realiza para verificar el cambio de campos en los cabzales ip y ethernet utilizando el controlador sdn ryu. Tanto en esta prueba como en la anterior es necesario cambiar las direcciones mac ya que estamos en un entorno LAN. La topología por lo tanto es la siguiente:

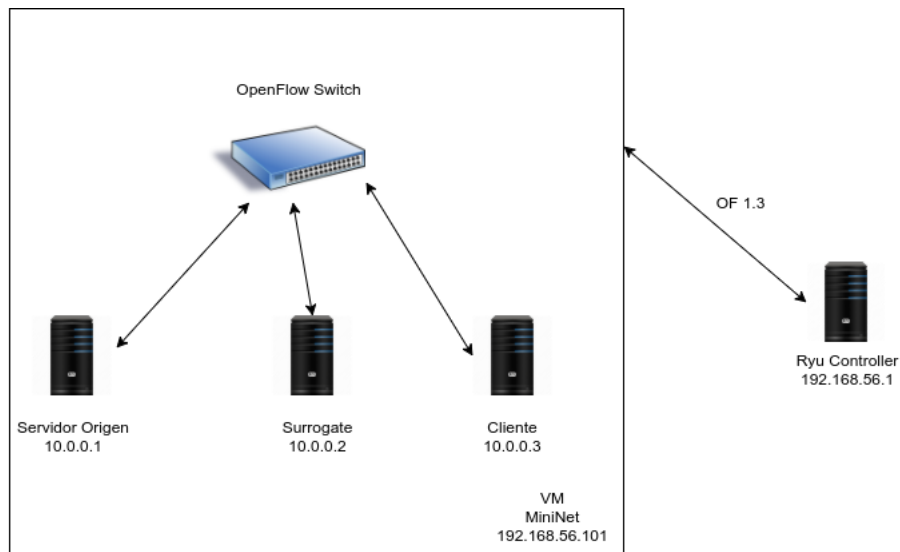


Figura 41: Prueba Concepto 2

Pruebas:

Se levanta el controlador ryu, el cual ejecuta la aplicacion prueba2.py.

```
$ sudo ryu-manager --ofp-tcp-listen-port 6633 prueba2.py
```

Luego se levanta la topologia en MiniNet, indicando donde se encuentra el controlador SDN, a traves del comando:

```
$ sudo mn --topo=single,3 --switch ovs,protocols=
    OpenFlow13 --controller=remote,ip=192.168.56.1,port
    =6633 --mac
```

Se observa que inicialmente las tablas de flujos del switch s1 están vacías. Esto se obtiene a través del comando:

```
$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13
```

Una vez iniciada la emulación, se pone a correr el servidor web en el host h2 (surrogate). Luego se realiza el HTTP request desde h3 (cliente) hacia h1 (origen).

```

mininet@mininet-vm: ~
mininet@mininet-vm:~$ sudo mn --topo=single,3 --switch ovs,protocols
=OpenFlow13 --controller=remote,ip=192.168.56.1,port=6633 --mac
*** Creating network
*** Adding controller
Unable to contact the remote controller at 192.168.56.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>

```

Figura 42: Inicio topología 2

Source	Destination	Protocol	Length	Info
10.0.0.3	10.0.0.1	TCP	76	41059 > http [SYN] Seq=0 Win=29200
10.0.0.3	10.0.0.1	OF 1.3	184	[TCP Out-Of-Order] of_packet_in
192.168.56.1	192.168.56.101	OF 1.3	212	of_flow_add
10.0.0.3	10.0.0.1	OF 1.3	214	[TCP Retransmission] of_packet_out
			68	<Ignored>
10.0.0.3	10.0.0.2	TCP	76	41059 > http [SYN] Seq=0 Win=29200

Figura 43: Capturas prueba 2

En esta primer imagen de capturas del HTTP request por parte del cliente al servidor de origen, se puede observar el inicio de conexión HTTP desde el cliente al servidor de origen, el segundo paquete muestra el mensaje que envía el switch hacia el controlador una vez recibido el paquete de inicio de conexión. El tercer paquete muestra el mensaje que envía el controlador al switch para que este agregue a su tabla de flujos, el flujo que tiene como direccion ip destino "10.0.0.1", mac destino "00:00:00:00:00:01z puerto tcp destino 80 y le aplique los cambios de ip y mac por los del surrogate y reenvíe el paquete por el puerto 2 (puerto al que está conectado el surrogate). El cuarto paquete muestra el mensaje que envía el controlador al switch para que reenvíe el paquete que acaba de entrar, aplicandole las acciones antes mencionadas. Por último vemos el HTTP SYN luego de que el switch aplica las acciones y este paquete tiene como destino el surrogate. En las siguientes imagenes se observa con mas detalle como el controlador agrega el flujo en el switch.

192.168.56.1	192.168.56.101	OF 1.3	212 of_flow_add
▼ of_match type: OFPMT_OXM (1) length: 39 ▼ of_oxm list ▼ of_oxm_eth_dst type_len: 2147485190 value: 00:00:00_00:00:01 (00:00:00:00:00:01) ▼ of_oxm_eth_type type_len: 2147486210 value: 2048 ▼ of_oxm_ip_proto type_len: 2147488769 value: 6 ▼ of_oxm_ipv4_dst type_len: 2147489796 value: 10.0.0.1 (10.0.0.1) ▼ of_oxm_tcp_dst type_len: 2147490818 value: 80			

Figura 44: Match Flow Add

192.168.56.1	192.168.56.101	OF 1.3	212 of_flow_add
▼ of_action_list ▼ of_action_set_field type: OFPAT_SET_FIELD (25) len: 16 ▼ of_oxm_ipv4_dst type_len: 2147489796 value: 10.0.0.2 (10.0.0.2) ▼ of_action_set_field type: OFPAT_SET_FIELD (25) len: 16 ▼ of_oxm_eth_dst type_len: 2147485190 value: 00:00:00_00:00:02 (00:00:00:00:00:02) ▼ of_action_output type: OFPAT_OUTPUT (0) len: 16 port: 2 max_len: 65509			

Figura 45: Actions Flow Add

En la siguiente imagen se observa la respuesta del inicio de la conexión del surrogate al cliente:

En esta imagen de capturas, se puede observar que el primer paquete corresponde a la respuesta del surrogate al cliente para el inicio de conexión HTTP, el segundo paquete muestra el mensaje que envía el switch hacia el controlador una vez recibido el paquete de respuesta de conexión. El tercer paquete muestra el mensaje que envía el controlador al switch para que este agregue a su tabla de flujos, el flujo que tiene como dirección ip origen "10.0.0.2", mac origen

Source	Destination	Protocol	Length	Info
10.0.0.2	10.0.0.3	TCP	76	http > 41059 [SYN, ACK] Seq=0 Ack=1
10.0.0.2	10.0.0.3	OF 1.3	184	[TCP Out-Of-Order] of_packet_in
192.168.56.1	192.168.56.101	OF 1.3	212	of_flow_add
10.0.0.2	10.0.0.3	OF 1.3	214	[TCP Retransmission] of_packet_out
			68	<ignored>
10.0.0.1	10.0.0.3	TCP	76	http > 41059 [SYN, ACK] Seq=0 Ack=1

Figura 46: Capturas prueba 2

”00:00:00:00:00:02” puerto tcp origen 80 y le aplique los cambios de ip y mac por los del surrogate y reenvíe el paquete por el puerto 3 (puerto al que está conectado el cliente). El cuarto paquete muestra el mensaje que envía el controlador al switch para que reenvíe el paquete que acaba de entrar, aplicandole las acciones antes mencionadas. Por último vemos el HTTP SYN ACK luego de que el switch aplica las acciones y este paquete tiene como origen el servidor de origen. En las siguientes imagenes se observa con mas detalle como el controlador agrega el flujo en el switch.

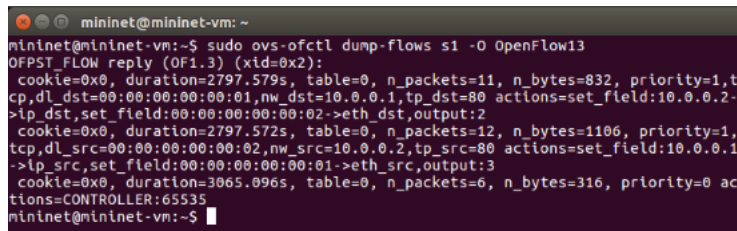
192.168.56.1	192.168.56.101	OF 1.3	212 of_flow_add
of_match type: OFPMT_OXM (1) length: 39 of_oxm list of_oxm_eth_src type_len: 2147485702 value: 00:00:00_00:00:02 (00:00:00:00:00:02) of_oxm_eth_type of_oxm_ip_proto of_oxm_ipv4_src type_len: 2147489284 value: 10.0.0.2 (10.0.0.2) of_oxm_tcp_src type_len: 2147490306 value: 80			

Figura 47: Match Flow Add

192.168.56.1	192.168.56.101	OF 1.3	212 of_flow_add
of_action list of_action_set_field type: OFPAT_SET_FIELD (25) len: 16 of_oxm_ipv4_src type_len: 2147489284 value: 10.0.0.1 (10.0.0.1) of_action_set_field type: OFPAT_SET_FIELD (25) len: 16 of_oxm_eth_src type_len: 2147485702 value: 00:00:00_00:00:01 (00:00:00:00:00:01) of_action_output type: OFPAT_OUTPUT (0) len: 16 port: 3 max_len: 65509			

Figura 48: Actions Flow Add

Por ultimo se muestra como quedan las tablas de flujo en el switch s1:

A terminal window with a dark background and light text. The prompt is 'mininet@mininet-vm: ~'. The command entered is 'sudo ovs-ofctl dump-flows s1 -O OpenFlow13'. The output shows three flow entries. The first entry has cookie=0x0, duration=2797.579s, table=0, n_packets=11, n_bytes=832, priority=1, and actions=set_field:10.0.0.2->ip_dst, set_field:00:00:00:00:00:02->eth_dst, output:2. The second entry has cookie=0x0, duration=2797.572s, table=0, n_packets=12, n_bytes=1106, priority=1, and actions=set_field:10.0.0.1->ip_src, set_field:00:00:00:00:00:01->eth_src, output:3. The third entry has cookie=0x0, duration=3065.096s, table=0, n_packets=6, n_bytes=316, priority=0, and actions=CONTROLLER:65535.

```
mininet@mininet-vm: ~  
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13  
OFPST_FLOW reply (OF1.3) (xid=0x2):  
  cookie=0x0, duration=2797.579s, table=0, n_packets=11, n_bytes=832, priority=1, t  
cp,dl_dst=00:00:00:00:00:01,nw_dst=10.0.0.1,tp_dst=80 actions=set_field:10.0.0.2-  
>ip_dst,set_field:00:00:00:00:00:02->eth_dst,output:2  
  cookie=0x0, duration=2797.572s, table=0, n_packets=12, n_bytes=1106, priority=1,  
tcp,dl_src=00:00:00:00:00:02,nw_src=10.0.0.2,tp_src=80 actions=set_field:10.0.0.1  
->ip_src,set_field:00:00:00:00:00:01->eth_src,output:3  
  cookie=0x0, duration=3065.096s, table=0, n_packets=6, n_bytes=316, priority=0 ac  
tions=CONTROLLER:65535  
mininet@mininet-vm:~$
```

Figura 49: Tabla de flujos s1

A.3. Prueba 3

En esta prueba se emulará un entorno WAN, para esto es necesario poder emular routers, en este caso y a diferencia de los casos anteriores, no se utilizara MiniNet puro, se utilizará la extensión de MiniNet mencionada anteriormente. La idea de estas pruebas es probar la arquitectura en un entorno WAN, ver sus limitantes y su potencial. En la figura 51 se observa la topología a probar, se anexa el script para la creación y configuración de la misma (topo3.wan.py). Para esta prueba es necesario modificar la aplicación RAUFlow (aplicación para control de red basada en el controlador SDN Ryu), para que el controlador sea capaz de indicar a los RAUSwitch como debe redirigir el tráfico hacia los surrogates. Se anexan los cambios realizados a la aplicación (gui.topology_3_cambios.py). En esta prueba la idea fue poner el cliente detrás de un router y ver si solo con cambiar la cabecera ip de los paquetes era suficiente para encaminar los pedidos a algún surrogate. Para esto se realizaron una serie de subpruebas sobre la topología emulada. La prueba solo acepta tráfico al puerto 80 de tcp (HTTP).

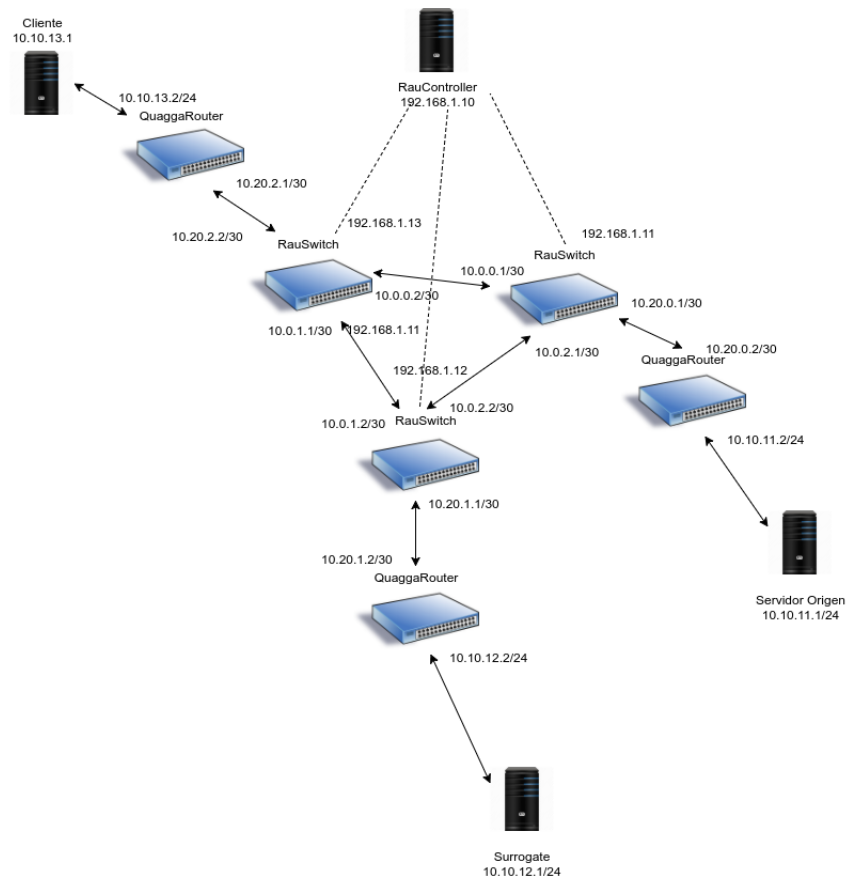


Figura 50: Prueba Concepto 3

- Prueba 1: Una vez levantada la topología se agregaron en el RAUSwitch por el que ingresan los pedidos del cliente los siguientes flujos (Se asume que el unico que relizara pedidos al origen sera el cliente):

```
$ sudo ovs-ofctl add-flow r3 "table=0,priority
=65535,ip_dst=10.10.11.1,tcp,tp_dst=80,
actions=set_field:10.10.12.1->ip_dst,output:2
" -O OpenFlow13
```

```
$ sudo ovs-ofctl add-flow r3 "table=0,priority
=65535,ip_src=10.10.12.1,tcp,tp_src=80,
actions=set_field:10.10.11.1->ip_src,output:3
" -O OpenFlow13
```

El primero realiza el cambio de ip destino en el pedido del cliente al origen, asignando la ip del surrogate. Mientras que el segundo realiza el cambio

de ip origen en la respuesta del surrogate, asignando la ip del servidor de origen.

Una vez configurados estos flujos, se procede a levantar el servidor web en el surrogate y a realizar el pedido HTTP desde el cliente hacia el origen en la consola de mininet.

```
mininet> surrogate python -m SimpleHTTPServer 80
&
mininet> cliente wget -O - origen
```

El resultado al realizar el pedido es el siguiente:

```
mininet> cliente wget -O - origen
--2017-02-28 18:36:52-- http://10.10.11.1/
Connecting to 10.10.11.1:80... ^C
mininet>
```

Figura 51: Prueba 3

- Prueba 2: En la prueba anterior el surrogate no recibe los pedidos del cliente, esto se debe a que los RAUSwitches de borde no saben rutear esos pedidos, por este motivo es necesario agregar en el RAUSwitch de borde que conecta con el surrogate los flujos correspondientes a HTTP. Esto se logra agregando en dicho RAUSwitch los siguientes flujos:

```
$ sudo ovs-ofctl add-flow r2 "table=0,priority
=65535,ip_dst=10.10.12.0/24,tcp_actions=
output:3" -O OpenFlow13
```

```
$ sudo ovs-ofctl add-flow r2 "table=0,priority
=65535,ip_dst=10.10.13.0/24,tcp_actions=
output:2" -O OpenFlow13
```

Con estos dos flujos agregados en el RAUSwitch que conecta al surrogate, se puede observar que los pedidos llegan al QuaggaRouter conectado al mencionado RAUSwitch y tienen destino el surrogate, pero el QuaggaRouter no los rutea hacia el destino.

No.	Time	Source	Destination	Protocol	Length	Info
13	8.758137000	10.10.13.1	10.10.12.1	TCP	76	47577 > http [SYN] Seq=0 Win=29200 Len=
15	9.757031000	10.10.13.1	10.10.12.1	TCP	76	[TCP Retransmission] 47577 > http [SYN]
18	11.762594000	10.10.13.1	10.10.12.1	TCP	76	[TCP Retransmission] 47577 > http [SYN]
26	15.768637000	10.10.13.1	10.10.12.1	TCP	76	[TCP Retransmission] 47577 > http [SYN]

Figura 52: Prueba 3

- Prueba 3: Estudiando el tráfico de la prueba anterior, se observa que el QuaggaRouter de borde no es capaz de rutear los pedidos hacia el surrogate porque la dirección mac de esos paquetes no tienen como destino la

del QuaggaRouter, esto se debe a que como los RAUSwitches de borde no tienen en su tabla de ruteo las rutas hacia el surrogate (ya que no se cuenta con un algoritmo de enrutamiento EGP, solo se corre un algoritmo de enrutamiento IGP), aplican los matcheos de la tabla de flujo OpenFlow y únicamente reenvía el paquete hacia el QuaggaRouter, pero deja como dirección mac destino la que estaba originalmente en el paquete. Por este motivo se tiene como limitante que el RAUSwitch de borde debe conocer la mac del QuaggaRouter al que está conectado. De esta forma cambiando los flujos en el RAUSwitch que conecta al surrogate y realizando los cambios de mac pertinentes la prueba queda funcional.

```
$ sudo ovs-ofctl add-flow r2 "table=0,priority
=65535,ip_dst=10.10.12.0/24,tcp_actions=
set_field:00:00:00:00:00:02->dl_dst,output:3"
-O OpenFlow13

$ sudo ovs-ofctl add-flow r2 "table=0,priority
=65535,ip_dst=10.10.13.0/24,tcp_actions=
set_field:00:00:00:00:00:03->dl_dst,output:2"
-O OpenFlow13
```

A.4. Prueba 4

En esta prueba se utilizará la misma topología de la prueba anterior, pero se diferencia de la prueba 3 en que se probará la implementación para una red ip, en la prueba anterior fue necesario realizar cambios en los cabezales ethernet y se deben tomar decisiones de enrutamiento (se debe setear el puerto de salida en las acciones de cada flujo). En esta prueba, al igual que en la anterior se utilizará la extensión de mininet, en la cual se utiliza el protocolo de enrutamiento interno de capa 3 OSPF. Para realizar el enrutamiento externo, se setearán a mano las rutas en cada uno de los RAUSwitches de la topología, esto se realiza de la siguiente manera:

- RAUSwitch 1

```
$ route add -net 10.10.11.0 netmask 255.255.255.0
gw 10.20.0.2 dev vr1-eth3

$ route add -net 10.10.12.0 netmask 255.255.255.0
gw 10.0.2.2 dev vr1-eth1

$ route add -net 10.10.13.0 netmask 255.255.255.0
gw 10.0.0.2 dev vr1-eth2
```

- RAUSwitch 2

```
$ route add -net 10.10.12.0 netmask 255.255.255.0
    gw 10.20.1.2 dev vr2-eth3
```

```
$ route add -net 10.10.11.0 netmask 255.255.255.0
    gw 10.0.2.1 dev vr2-eth1
```

```
$ route add -net 10.10.13.0 netmask 255.255.255.0
    gw 10.0.1.1 dev vr2-eth2
```

■ RAUSwitch 3

```
$ route add -net 10.10.13.0 netmask 255.255.255.0
    gw 10.20.2.2 dev vr3-eth3
```

```
$ route add -net 10.10.11.0 netmask 255.255.255.0
    gw 10.0.0.1 dev vr3-eth1
```

```
$ route add -net 10.10.12.0 netmask 255.255.255.0
    gw 10.0.1.2 dev vr3-eth2
```

Luego se instala en el RAUSwitch al que está conectado el cliente (r3), un flujo que indica que todo tráfico que sea dirigido al servidor de origen (10.10.11.1) y que ingrese por la interfaz física de borde, se le aplique el cambio de dirección ip destino, a la dirección ip del surrogate (10.10.12.1) y sea enviado a la interfaz virtual de borde para que el tráfico sea enrutado según las tablas de ruteo y se agrega un segundo flujo que indica que todo tráfico que ingrese a la interfaz virtual de borde (este flujo debe ser instalado sólo en los RAUSwitch de borde, en el caso de esta topología son todos de borde, por lo tanto este flujo ira en todos los RAUSwitches), sea enviado a la interfaz física de borde. Esto se logra con los siguientes comandos:

```
$ ovs-ofctl add-flow r3 "table=0,priority=1,ip_dst
    =10.10.11.1,tcp,in_port=3_actions=set_field
    :10.10.12.1->ip_dst,output:6" -O OpenFlow13
```

```
$ add-flow r3 "table=0,priority=1,in_port=6_actions=
    output:3" -O OpenFlow13
```

Una vez instalado el primero de los flujos anteriores en r3, este ya es capaz de rutear el pedido del cliente al surrogate. Ahora se deben instalar flujos en el RAUSwitch que conecta al QuaggaRouter al que se conecta el surrogate (r2) para que realice el cambio de ip de origen en el pedido, seteando la ip del servidor de origen, para que la elección de surrogate sea transparente para el cliente, además de ese flujo se instala (como en el caso de r3) un segundo flujo que indica que todo tráfico que ingrese a la interfaz virtual de borde, sea enviado a la interfaz física de borde. Estos flujos se instalan con los siguientes comandos:

```
$ ovs-ofctl add-flow r2 "table=0,priority=1,ip_src
=10.10.12.1,tcp,in_port=3,actions=set_field
:10.10.12.1->ip_src,output:6" -O OpenFlow13
```

```
$ add-flow r2 "table=0,priority=1,in_port=6,actions=
output:3" -O OpenFlow13
```

Por ultimo se instalan en r1 los flujos que logran que r1 routee paquetes que ingresan por la interfaz fisica de borde.

```
$ add-flow r1 "table=0,priority=1,in_port=3,actions=
output:6" -O OpenFlow13
```

```
$ add-flow r1 "table=0,priority=1,in_port=6,actions=
output:3" -O OpenFlow13
```

Luego de realizadas las configuraciones anteriores, se pasa a realizar una simple prueba. En el surrogate se levanta un servidor web en el puerto 80 y desde el cliente (10.10.13.1) se realiza un HTTP GET al servidor de origen (10.10.11.1). Gracias a los flujos instalados en r3, este realiza el cambio de ip destino y rutea el request del cliente hacia el surrogate (10.10.12.1).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.10.13.1	10.10.11.1	TCP	74	41556 > http [SYN]
2	0.000000000	10.10.13.1	10.10.12.1	TCP	74	41556 > http [SYN]

Figura 53: r3 modifica ip destino

Una vez que el surrogate responde al cliente, r2 realiza el cambio de ip de origen, seteando como ip de origen la del servidor de origen. Luego de realizar este cambio, el router no es capaz de rutear la respuesta hacia el cliente. Los logs del router indican los siguiente:

```
martian source 10.10.13.1 from 10.10.11.1, on dev vr3-eth3
```

Investigando las posibles causas, se llegó a la conclusión que los routers por defecto aplica el filtro de reverse path para realizar el ruteo [48]. Como solución provisoria se decidió deshabilitar ese filtrado en todos los RAUSWitches, para esto se debe modificar una variable en el kernel.

```
$ sysctl -w net.ipv4.conf.all.rp_filter = 0
```

```
$ sysctl -p
```

Una vez deshabilitado el filtro las pruebas se logran realizar de forma exitosa, cabe destacar que el filtro rp de deshabilita para poder realizar las pruebas, en un ambiente en producción esto no es aceptable, ya que el filtro de reverse path ayuda a prevenir, entre otras cosas, ataques de denegación de servicio. este problema se debería resolver al utilizar full-routing con BGP.

A.5. Prueba 5

En esta prueba se mostrará como instalar el servidor web, el cual se encarga de exponer los servicios RESTful del controlador CDN y se encarga de hostear el sitio web de administración de la arquitectura. También se mostrará como instalar la base de datos mongoDB. Como se mencionó en la descripción de cada uno de estos componentes, en este trabajo, estos componentes corren fuera del entorno virtual. Su instalación se realizó en un equipo que tiene 4 GB de memoria RAM, un procesador Intel® Core™ i3-3110M CPU @ 2.40GHz 4, y tiene como sistema operativo un Ubuntu 16.04 LTS de 64 bits. También se mostrará cómo funciona el sitio web para administrar servidores de origen y surrogates de la CDN.

En [49] se muestra como realizar la instalación de mongoDB en un Ubuntu 16.04 LTS.

Node.JS será el encargado de levantar el servidor web, que hostea el sitio y expone los métodos del controlador CDN, su instalación se logra a través del siguiente comando:

```
$ apt-get install npm
```

Luego es necesario instalar algunas dependencias que necesitará el controlador CDN, las cuales se instalan con los siguientes comandos:

```
$ npm install express
```

```
$ npm install --save body-parser
```

```
$ npm install request
```

```
$ npm install mongodb
```

Luego es necesario bajar los codigos fuentes de los componentes mencionados desde el repositorio de este trabajo [50]. Una vez descargados, se ingresara por consola al directorio donde se encuentra el sitio web ("web server_cdnController/public") y se ejecutarán los siguientes comandos:

```
$ npm update
```

```
$ npm run build
```

Por último se accede a la carpeta del web server ("web server_cdnController") y se levanta la aplicación, a través del comando:

```
$ node server.js
```

De esta forma el web server queda levantado y corriendo en el puerto 3000. Para acceder al sitio web de administración, se debe abrir en un navegador la dirección "http://ip_web_server:3000" (en esta prueba se utiliza "localhost", ya que se realizará la prueba en la misma máquina donde corre el servidor web). Una vez que se accede al sitio web, la primer pantalla que se visualiza contiene una introducción al trabajo, la cual se puede observar en la figura 54.



Figura 54: Administración CDN

Desde el menú de la izquierda es posible navegar por el sitio web de administración. Para agregar un nuevo servidor de origen se debe seleccionar en el menú de la izquierda la opción "Agregar origen". Esta pantalla pide que se ingrese la ip del nuevo servidor de origen, el puerto donde el servidor estará escuchando y el protocolo de transporte del nuevo servidor de origen. En la figura 55 se observa la pantalla para agregar un nuevo servidor de origen.

Luego para agregar un nuevo surrogate se debe seleccionar en el menú de la izquierda la opción "Agregar surrogate". Esta pantalla pide que se ingrese la ip del nuevo surrogate, el puerto donde el surrogate estará escuchando, la ip y el puerto del servidor de origen que se va a replicar y el protocolo de transporte del servidor de origen. En la figura 56 se observa la pantalla para agregar un nuevo surrogate.

The screenshot shows the 'Agregar Origin' form within the 'Administración CDN' application. The interface has a dark blue header with the title 'Administración CDN' and a user profile 'Admin'. A sidebar on the left contains navigation links: 'Inicio', '+ Agregar Origin' (highlighted), '+ Agregar Surrogate', '+ Eliminar Origin', '+ Eliminar Surrogate', and '+ CDN'. The main content area has a yellow header 'Agregar Origin'. It contains two text input fields: 'IP Origin' with the placeholder 'ip origin' and 'Port Origin' with the placeholder 'port origin'. Below these are two radio buttons for 'TCP' and 'UDP', with 'TCP' selected. A blue 'Agregar' button is at the bottom. The footer is dark blue with the text 'Tesis de Ingeniería 2017 Mathias Duarte'.

Figura 55: Agregar Origin

The screenshot shows the 'Agregar Surrogate' form within the 'Administración CDN' application. The interface is similar to the previous one, with a dark blue header and a sidebar. The sidebar highlights '+ Agregar Surrogate'. The main content area has a yellow header 'Agregar Surrogate'. It contains four text input fields: 'IP Origin' (placeholder 'ip origin'), 'Port Origin' (placeholder 'port origin'), 'IP Surrogate' (placeholder 'ip surrogate'), and 'Port Surrogate' (placeholder 'port surrogate'). Below these are two radio buttons for 'TCP' and 'UDP', with 'TCP' selected. A blue 'Agregar' button is at the bottom. The footer is dark blue with the text 'Tesis de Ingeniería 2017 Mathias Duarte'.

Figura 56: Agregar Surrogate

Para eliminar un servidor de origen se debe seleccionar en el menú de la izquierda la opción .eliminar origen”. Esta pantalla pide que se ingrese la ip del servidor de origen a eliminar, el puerto donde el servidor a eliminar esta escuchando y el protocolo de transporte del servidor de origen a eliminar. En la figura 57 se observa la pantalla para eliminar un servidor de origen.

Para eliminar un surrogate se debe seleccionar en el menú de la izquierda la opción .eliminar surrogate”. Esta pantalla pide que se ingrese la ip del surrogate a eliminar, el puerto donde el surrogate a eliminar esta escuchando, la ip y el puerto del servidor de origen replicado por el surrogate a eliminar y el protocolo

de transporte del servidor de origen. En la figura 58 se observa la pantalla para eliminar un surrogate.

The screenshot shows the 'Administración CDN' interface with a sidebar on the left containing navigation links: 'Inicio', '+ Agregar Origin', '+ Agregar Surrogate', '+ Eliminar Origin', '+ Eliminar Surrogate', and '+ CDN'. The main content area is titled 'Eliminar Origin' and contains the following fields: 'IP Origin' (text input with placeholder 'ip origin'), 'Port Origin' (text input with placeholder 'port origin'), and radio buttons for 'TCP' and 'UDP'. A blue 'Eliminar' button is positioned below the radio buttons. The footer of the interface reads 'Tesis de Ingeniería 2017 Mathias Duarte'.

Figura 57: Eliminar Origin

The screenshot shows the 'Administración CDN' interface with the same sidebar as Figure 57. The main content area is titled 'Eliminar Surrogate' and contains the following fields: 'IP Origin' (text input with placeholder 'ip origin'), 'Port Origin' (text input with placeholder 'port origin'), 'IP Surrogate' (text input with placeholder 'ip surrogate'), and 'Port Surrogate' (text input with placeholder 'port surrogate'). Radio buttons for 'TCP' and 'UDP' are located below these fields, along with a blue 'Eliminar' button. The footer of the interface reads 'Tesis de Ingeniería 2017 Mathias Duarte'.

Figura 58: Eliminar Surrogate

Por último, desde el sitio web es posible obtener información sobre la CDN, se puede consultar los servidores de origen y sus surrogates asociados, para esto se debe seleccionar la opción **CDN**.^{en} el menú de la izquierda, en la figura 59 se observa como se presenta la información sobre orígenes y surrogates.

The screenshot shows a web application titled 'Administración CDN'. On the left is a sidebar menu with options: 'Inicio', '+ Agregar Origin', '+ Agregar Surrogate', '+ Eliminar Origin', '+ Eliminar Surrogate', and '+ CDN'. The main area displays a table titled 'Origins' with the following data:

#	Ip Origin	Puerto Origin	Transport	Surrogates
0	10.2.1.2	80	tcp	10.9.1.2:8089; 10.1.1.2:80;
1	10.5.1.2	8080	tcp	10.8.1.2:88;

At the bottom of the page, there is a footer that reads: 'Tesis de Ingeniería 2017 Mathias Duarte'.

Figura 59: CDN sitio web

En lo que sigue de la prueba se emulara la topología de la figura 60 y se agregarán servidores de origen y surrogates, y se observará cómo se selecciona el mejor surrogate según el algoritmo de selección desarrollado.

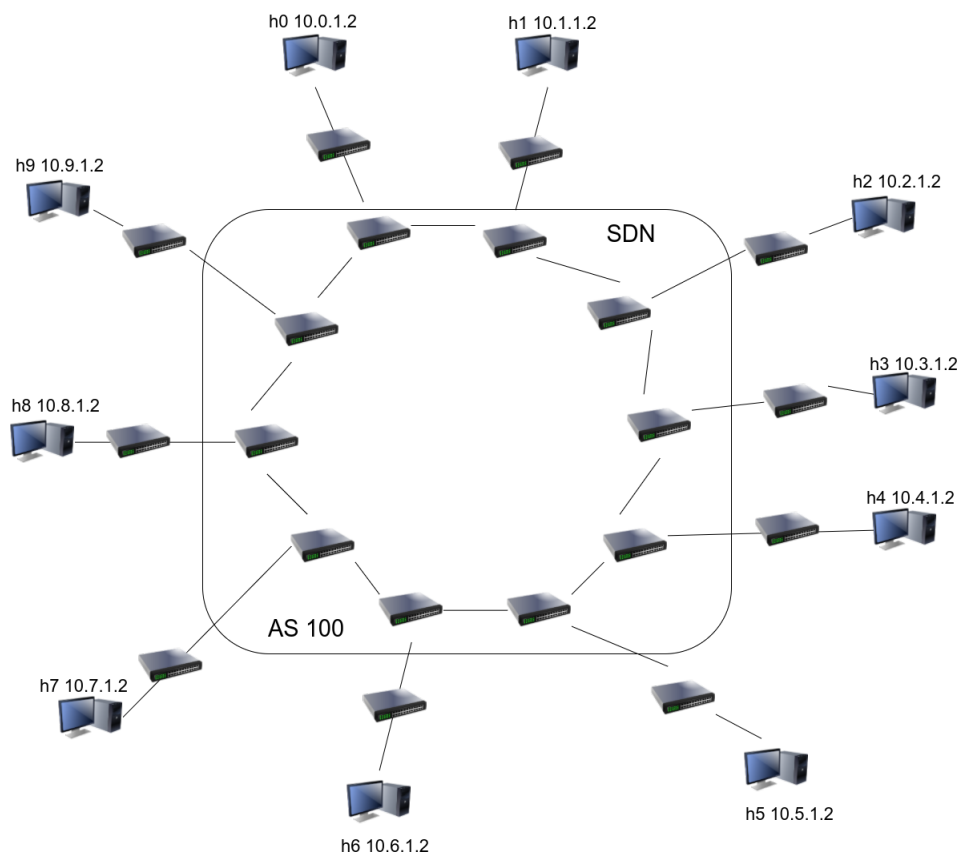


Figura 60: Topología prueba 5

Dada la topología de la figura 60, se realizará el pedido de una página web desde un cliente hacia un servidor de origen. Se utilizará h2 como cliente y desde la web se agregara un servidor de origen (h3) y tres surrogates (h4, h5, h6). Los puertos en donde atenderán los surrogates son los siguientes: h4 -¿8084, h5 -¿8085 y h6 -¿8086, mientras que el servidor web de origen escucha en el puerto 80.

El primer paso luego de agregar el servidor de origen, será agregar dos surrogates h6 y h5. Según la topología indicada arriba, el mejor surrogate tiene que ser h5, h2 se encuentra a una distancia de tres saltos a h5 y a una distancia de cuatro saltos a h6.

En las siguientes capturas se puede observar como el controlador le indica al RAUSwitch de borde que conecta a h3 a la SDN, que la mejor opción de surro-

gate es h5 y las modificaciones en los cabezales ip que debe realizar. También se muestran las tablas de flujo del RAUSwitch que conecta al cliente (h2) a la SDN y se puede observar el flujo que indica el cambio de dirección ip y puerto tcp con los datos del mejor surrogate para el cliente.

No.	Time	Source	Destination	Protocol	Length	Info
1509	196.17215606	10.2.1.2	10.3.1.2	OF 1.3	212	[TCP Retransmission] of_packet_out
OpenFlow						
version: 4						
type: OFPT_PACKET_OUT (13)						
length: 146						
xid: 2612369826						
buffer_id: 4294967295						
in_port: 3						
actions_len: 48						
of_action list						
▼ of_action_set_field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm_tcp_dst						
type_len: 2147490818						
value: 8085						
▼ of_action_set_field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm_ipv4_dst						
type_len: 2147489796						
value: 10.5.1.2 (10.5.1.2)						

Figura 61: Capturas prueba 5

No.	Time	Source	Destination	Protocol	Length	Info
1515	196.18484106	10.5.1.2	10.2.1.2	OF 1.3	212	[TCP Out-Of-Order] of_packet_out
OpenFlow						
version: 4						
type: OFPT_PACKET_OUT (13)						
length: 146						
xid: 2612369828						
buffer_id: 4294967295						
in_port: 6						
actions_len: 48						
of_action list						
▼ of_action_set_field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm_tcp_src						
type_len: 2147490306						
value: 80						
▼ of_action_set_field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm_ipv4_src						
type_len: 2147489284						
value: 10.3.1.2 (10.3.1.2)						

Figura 62: Capturas prueba 5

```

root@antiscr-VirtualBox: /P001_44# ovs-vsctl dump-flows switch3 -O openflow15
OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x0, duration=1667.828s, table=0, n_packets=1102, n_bytes=56202, priority=65535,d1_dst=01:80:c2:00:00:0e,d1_type=0x00cc actions=CONTROLLER:85535
  cookie=0x0, duration=0.56s, table=0, n_packets=5, n_bytes=150, idle_timeout=30, priority=100,tcp_in_port=5,nw_dst=10.5.1.2,ip_dst=0 actions=set_field:8085->tcp_dst,set_field:10.5.1.2->ip_dst,output:5
  cookie=0x0, duration=0.482s, table=0, n_packets=4, n_bytes=1095, idle_timeout=30, priority=100,tcp_in_port=5,nw_src=10.5.1.2,ip_src=8085 actions=set_field:80->tcp_src,set_field:10.3.1.2->ip_src,output:3

```

Figura 63: Tablas flujo prueba 5

Luego se agrega el surrogate h4, el cual está a una distancia de dos saltos del cliente (h2), por lo que el mejor surrogate para el cliente h2 ahora será h4. En las siguientes capturas se observa como ahora el controlador indica h4 como mejor surrogate.

No.	Time	Source	Destination	Protocol	Length	Info
239	24.571148806	10.2.1.2	10.3.1.2	OF 1.3	212	[TCP Retransmission] of_packet_out
▼ OpenFlow						
version: 4						
type: OFPT_PACKET_OUT (13)						
length: 146						
xid: 2612371825						
buffer_id: 4294967295						
in_port: 3						
actions_len: 48						
▼ of_action list						
▼ of_action set field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm tcp_dst						
type_len: 2147490818						
value: 8084						
▼ of_action set field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm ipv4_dst						
type_len: 2147489796						
value: 10.4.1.2 (10.4.1.2)						

Figura 64: Capturas prueba 5

No.	Time	Source	Destination	Protocol	Length	Info
245	24.577270006	10.4.1.2	10.2.1.2	OF 1.3	212	[TCP Retransmission] of packet out
▼ OpenFlow						
version: 4						
type: OFPT_PACKET_OUT (13)						
length: 146						
xid: 2612371827						
buffer_id: 4294967295						
in_port: 6						
actions len: 48						
▼ of_action list						
▼ of_action_set_field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm_tcp_src						
type_len: 2147490306						
value: 80						
▼ of_action_set_field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of_oxm_ip4_src						
type_len: 2147489284						
value: 10.3.1.2 (10.3.1.2)						

Figura 65: Capturas prueba 5

```
root@antigo-VirtualBox: /P2015_44# ovs-ofctl dump-flows switch3 -O OpenFlow13
OFS1 FLOW reply (OF1.3) (xid=62):
cookie=00, duration=230.366, table=0, n_packets=1336, n_bytes=80736, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x80cc actions=CONTROLLER:65535
cookie=00, duration=3.756, table=0, n_packets=5, n_bytes=514, idle_timeout=30, priority=100,tcp_in_port=3,nw_dst=10.5.1.2,tcp_dst=80 actions=set_field:80&tcp_dst.set_field:10.4.1.2->ip_dst,output:5
cookie=00, duration=3.765, table=0, n_packets=5, n_bytes=1336, idle_timeout=30, priority=100,tcp_in_port=3,nw_src=10.4.1.2,tcp_src=80&tcp_src.set_field:10.5.1.2->ip_src,output:1
```

Figura 66: Tablas flujo prueba 5

Por último se tomará h7 como cliente, en este caso el controlador deberá indicar al RAUSwitch de borde que conecta a h7 a la SDN, que la mejor opción de surrogate será h6, ya que existe un salto de distancia entre ambos. A continuación se observan las capturas y la tabla de flujos del RAUSwitch de borde que conecta a h7 a la SDN.

No.	Time	Source	Destination	Protocol	Length	Info
344	42.560919006	10.7.1.2	10.3.1.2	OF 1.3	212	[TCP Retransmission] of packet out
▼ OpenFlow						
version: 4						
type: OFPT_PACKET_OUT (13)						
length: 146						
xid: 348227739						
buffer_id: 4294967295						
in_port: 3						
actions len: 48						
▼ of action list						
▼ of action set field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of oxm tcp_dst						
type_len: 2147490818						
value: 8086						
▼ of action set field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of oxm ipv4_dst						
type_len: 2147489796						
value: 10.6.1.2 (10.6.1.2)						

Figura 67: Capturas prueba 5

No.	Time	Source	Destination	Protocol	Length	Info
350	42.566497006	10.6.1.2	10.7.1.2	OF 1.3	212	[TCP Out-Of-Order] of packet out
▼ OpenFlow						
version: 4						
type: OFPT_PACKET_OUT (13)						
length: 146						
xid: 348227741						
buffer_id: 4294967295						
in_port: 6						
actions len: 48						
▼ of action list						
▼ of action set field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of oxm tcp_src						
type_len: 2147490306						
value: 80						
▼ of action set field						
type: OFPAT_SET_FIELD (25)						
len: 16						
▼ of oxm ipv4_src						
type_len: 2147489284						
value: 10.3.1.2 (10.3.1.2)						

Figura 68: Capturas prueba 5

```
root@swtting:~/VirtualBox/2015.444_ova-ofctl_dump-flows_switch8 ~# OpenFlow3
OFPT_FLOW_REPLY (OF1.3) (xid=0x2):
cookie=0x0, duration=3508.128s, table=0, n_packets=2318, n_bytes=18218, priority=65535, dl_dst=01:80:c2:00:00:0e, dl_type=0x80cc actions=CONTROLLER:65535
cookie=0x0, duration=5.421s, table=0, n_packets=5, n_bytes=45, idle_timeout=30, priority=100, tcp_in_port=5, nw_dst=10.6.1.2, ip_dst=0 actions=set_field:0x00->tcp_dst.set_field:10.6.1.2->ip_dst.output:5
cookie=0x0, duration=5.484s, table=0, n_packets=4, n_bytes=106, idle_timeout=30, priority=100, tcp_in_port=5, nw_src=10.6.1.2, ip_src=0 actions=set_field:80->tcp_src.set_field:10.3.1.2->ip_src.output:3
```

Figura 69: Tablas flujo prueba 5

B. Códigos Fuente

Todos los códigos fuente de pruebas de concepto e implementaciones de la arquitectura se encuentran en [50].