PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN BALANCEADOR DE CARGA PARA LA OPTIMIZACIÓN DE LOS RECURSOS DE PROTECCIÓN EN UNA RED ENTERPRISE MEDIANTE UN BANCO DE FIREWALLS N:1 CONTROLADO VÍA SDN

Tesis para optar el Título de Ingeniero de las Telecomunicaciones, que presenta el bachiller:

QUISPE ORDOÑEZ CHRISTIAN ISAAC

ASESOR: César Augusto Santiváñez Guarniz, PhD.

Lima, noviembre de 2019

DEDICATORIA

Para Yda e Isaac, se logró.

RESUMEN

El presente trabajo consiste en el diseño e implementación de un balanceador de carga que permite el funcionamiento del modelo de protección banco de *Firewalls* N:1 controlado vía *SDN* sin pérdida de paquetes o interrupción de las sesiones migradas. Para ello, se realiza el diseño del balanceador a través de un algoritmo que realice la migración de tráfico por *flow entries* al orden de una subred para no exceder el límite de entradas disponibles en la memoria *TCAM* del *switch SDN/OpenFlow* a utilizar (8K a lo máximo). La implementación del balanceador será en módulos escritos en el lenguaje *Python*. El objetivo principal del balanceador es garantizar que las pruebas realizadas en él tengan el mismo o mejor rendimiento que los balanceadores de carga comerciales (*legacy*) en presencia de tráfico que bordea el 1 Gbps.

En el primer capítulo, se describe el marco de la problemática, donde se compara los costos que requiere el modelo de protección activo - respaldo versus el banco de *Firewalls* N:1. Luego, se describe los retos que enfrenta el diseño del balanceador de este trabajo. Por último, con todo lo anterior se plantea la hipótesis y los objetivos de la presente tesis.

En el segundo capítulo se exponen las bases teóricas para entender el diseño del balanceador. Se explica el paradigma *SDN*; los modelos de protección y sus elementos más importantes entre ellos los *firewalls* de primera, segunda, tercera y cuarta generación; los balanceadores de carga comerciales, *ADC* como *BIG-IP* o *ServerIron ADX*, y el comportamiento del tráfico en las redes de banda ancha.

En el tercer capítulo, se describe la arquitectura del balanceador de carga. Se introduce el concepto de “Rama” y se explica el proceso *Hand Off* para entender la migración del tráfico de una subred desde un equipo congestionado a uno descongestionado. Asimismo, se mencionan los requerimientos a considerar en el diseño y las principales limitaciones y dificultades que se encuentran. Finalmente, se presenta el diseño a alto y bajo nivel incluyendo el algoritmo para el balanceo de carga, la funcionalidad de los módulos del diseño del balanceador y el diagrama a nivel de código para su posterior implementación en el lenguaje *Python*.

En el capítulo final, se presentan los resultados obtenidos al realizar la prueba de concepto con la finalidad de demostrar que el balanceador funciona en un entorno cuya intensidad de tráfico bordea el 1Gbps en base a los objetivos y requerimientos planteados.

Finalmente, se presentan las conclusiones obtenidas por la implementación del balanceador y la prueba de concepto, basadas en los objetivos de la tesis y los requerimientos del balanceador, así como los trabajos futuros a surgir debido a la presente tesis.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

Para contrarrestar los posibles ciberataques que puedan vulnerar la seguridad de una red empresarial, estas emplean *firewalls* de segunda o tercera generación, generalmente configuradas a través del esquema de protección activo - respaldo (un *firewall* está operativo mientras el otro le sirve de respaldo). Cada *firewall* se dimensiona para soportar el tráfico futuro de la red, con una proyección de al menos cinco años. Esto resulta la compra de dos equipos costosos con una capacidad hasta 5 veces mayor a la necesidad actual de la empresa.

El modelo de protección banco de *firewall* N:1 (N *firewalls* operativos y uno de respaldo) surge como una opción a reemplazar el modelo activo–respaldo permitiendo una reducción significativa del costo de capital (CaPex): sus elementos son adquiridos en función a la demanda real y se reduce el tamaño y costo del equipo de respaldo. Adicionalmente a la reducción del costo financiero, este esquema se beneficia de la natural reducción de costo de los equipos con el tiempo (para una misma capacidad de procesamiento).

Un banco de *firewall* N:1 requiere de un equipo balanceador de carga que distribuya el tráfico proveniente de la red empresarial a Internet y viceversa a los N *firewalls* que lo conforman. Balanceadores convencionales, con la granularidad del orden de una sesión, suelen ser equipos costosos para una red empresarial del tamaño de la PUCP con más de 80K sesiones activas en un instante dado. Por otro lado, un *switch SDN/OpenFlow* puede reemplazar estas funciones agrupando sesiones por subredes sin exceder el límite de entradas posibles en su memoria *TCAM* (no más de 8K). Esta solución reduce significativamente el costo de procesamiento en el *switch*, pero se enfrenta al reto de evitar interrumpir/migrar una sesión activa. Esto último es crítico pues a partir de los *firewall* de segunda generación estos dispositivos mantienen el estado de la conexión, y la migración de una sesión de un *firewall* a otro resultaría en el bloqueo de esta.

El objetivo de la presente tesis es el diseño e implementación de un sistema balanceador de carga vía un *switch SDN/OpenFlow*. El diseño garantiza que el sistema N:1 tenga el mismo o mejor rendimiento que el modelo de protección activo - respaldo en presencia de picos de tráfico y evita la pérdida de estado de una sesión debido a la migración de la carga de un *firewall* sobrecargado hacia otro.

1. ASPECTOS GENERALES

El presente capítulo describe la justificación del problema, lo objetivos, la hipótesis, las razones que motivaron a desarrollar el trabajo y la estructura de la presente tesis.

1.1 Definición del problema y justificación

1.1.1 Tráfico en la PUCP

El *CAGR* (Tasa de crecimiento Anual Compuesto) del tráfico en la PUCP es de 35% según el Área de Dirinfo [1]. El promedio del tráfico registrado el año 2015 era 900 Mbps. Para el año 2019 se estima que el tráfico promedio hacia Internet en la PUCP será de 3 Gbps [1].

Tabla 1-1 Crecimiento del tráfico en la PUCP

Año	Capacidad	
1er Año (2015)	900	Mbps
2do Año (2016)	1215	Mbps
3er Año (2017)	1640.25	Mbps
4to Año (2018)	2214.34	Mbps
5to Año (2019)	2989.36	Mbps

Fuente: [1]

El dimensionamiento de *firewalls* según el modelo de protección (activo – respaldo o banco de *firewalls* N:1) en la PUCP depende de la proyección a cinco años del tráfico.

Tabla 1-2 Dimensionamiento de firewalls

Año	Tráfico en 1 FW			Tráfico a 2 FW			Tráfico a 4 FW			Unidades
	Promedio	Hora Pico	Total	Promedio	Hora Pico	Total	Promedio	Hora Pico	Total	
1er Año	736.31	900	1800	368.16	495	990	184.08	280	560	Mbps
2do Año	1027.06	1215	2430	513.53	655	1310	256.76	365	730	Mbps
3er Año	1424.81	1640.25	3280.5	712.4	875	1750	356.2	480	960	Mbps
4to Año	1964.41	2214.34	4428.68	982.2	1170	2340	491.1	630	1260	Mbps
5to Año	2705.1	2989.36	5978.72	1352.55	1565	3130	676.28	835	1670	Mbps

Fuente: [1]

Según los cálculos en la Tabla 1-2 el tráfico total (asumiendo el peor caso donde el tráfico entrante y saliente a Internet son iguales) que procesara un *firewall* al quinto año será 6 Gbps. Si se utiliza dos *firewalls*, cada uno procesará 3.1 Gbps. Asimismo, en el caso que el banco de *firewalls* este conformado por 4 *firewalls* cada uno deberá procesar en promedio 1.7 Gbps.

En la Tabla 1-2 se muestra el procesamiento que deberá soportar un *firewall* en función a los años según el modelo de protección a utilizar (activo – respaldo, banco 2:1 y banco 4:1 respectivamente).

La proyección del tráfico al quinto año servirá para la elección de las características del *firewall* según el modelo de protección a utilizar. Esto influirá en los costos de capital (CaPex).

1.1.2 Comparación de costos entre los modelos de protección

A continuación se presente una comparación de precios entre el modelo de protección activo – respaldo y banco de *firewalls* 4:1 para el escenario PUCP presentado en la sección anterior. En [1] se eligió el *firewall* Juniper SRX3400 que soporta un tráfico de 8 Gbps (IMIX) para el modelo de protección activo – respaldo y el *firewall* Juniper SRX 550 que soporta un tráfico de 1.7 Gbps (IMIX) para el banco de *firewall* 4:1.

Tabla 1-3 Comparación de costos entre modelos de protección

Año	Tráfico Hora Pico	Modelo 1:1			Modelo N:1		
		Tráfico	Equipos	Precio($)	Tráfico	Equipos	Precio($)
1er Año	1800	8000	1 FW + spare	381,000	3400	2 FW + spare + 4 ADC	253,650
2do Año	2430	8000	-	0	3400	-	0
3er Año	3280.5	8000	-	0	5100	1 FW	39,094
4to Año	4428.68	8000	-	0	5100	-	0
5to Año	5978.72	8000	-	0	6800	1 FW	18,249
			PRECIO TOTAL	381,000		PRECIO TOTAL	310,993

Fuente: [1]

En la Tabla 1-3; por un lado, se muestra que para el modelo de protección 1:1 desde el primer año se adquiere un *firewall* activo y su respaldo con un costo de capital aproximado de 381 000 dólares. Por otro lado, en el modelo 4:1 se adquiere el primer año 3 *firewalls* (incluido el equipo de respaldo) más los equipos balanceadores de carga o ADC (*ServerIron* ADX 1016 - 4 que soportan hasta 9 Gbps de tráfico) con un precio aproximado de 20 370 dólares cada uno. En función a la demanda se adquieren dos *firewalls* más el tercer y el quinto año. El costo del capital según este modelo (incluido las licencias de los *firewalls*) aproximadamente es 310 993 dólares.

El modelo de protección banco de *firewall* N:1 en el ejemplo mostrado reduce el costo de capital (CaPex) en comparación al modelo de protección activo – respaldo.

1.1.3 Problemática en el balanceador de carga

El modelo de protección N:1 es un candidato para reemplazar al modelo activo – respaldo pues reduce el costo de capital CaPex y el tamaño del equipo de respaldo (de un *firewall* Juniper SRX 3400 a un SRX 550). Sin embargo, el modelo N:1 requiere un equipo balanceador de carga que distribuya el tráfico de internet a los elementos que la conforman.

Para los cálculos de costos en la Tabla 1-3 se utilizó 4 ADC (dos balanceadores de carga comerciales operativos y dos en modo respaldo) que distribuyen la carga a la granularidad de una sesión. El rendimiento de estos balanceadores de carga comerciales está limitado al quinto año, frente al futuro crecimiento del tráfico de internet se tendrá que renovar nuevos ADC (ADX 4000 que soporta hasta 35 Gbps con un precio de 34, 260 dólares por unidad) dejando obsoletas a los balanceadores de carga comerciales iniciales (ADX 1016 - 4).

A largo plazo el costo de inversión en el modelo N:1 tendrá el mismo precio que el modelo activo – respaldo. Sin embargo, la adaptación de un *switch SDN/OpenFlow* modelo P-3297 de la firma PICA8 con un precio de 3960 dólares por unidad puede sustituir las funciones de los balanceadores de carga comerciales. El costo de capital CaPex del modelo 4:1 reemplazando los 4 ADC por dos *switch SDN/OpenFlow* disminuye en 237, 433 dólares. Esto convierte al modelo N:1 en un candidato a reemplazar al modelo activo – respaldo ya que hay un ahorro significativo en CaPex (143, 567 dólares).

El diseño y la implementación de un balanceador de carga vía un *switch SDN/OpenFlow* se enfrenta a dos retos: (i) migrar la carga de un *firewall* a otro, sin afectar las sesiones activas ya establecidas, cuando se sobrecarga el primero. Esto porque a partir de los *firewalls* de segunda generación se manejan los estados de las conexiones activas donde la migración

de una sesión a otro *firewall* puede resultar en su denegación perjudicando la performance del sistema. (ii) El precio del *switch SDN/OpenFlow* en comparación a los ADC comerciales es más barato ya que tiene un *hardware* menos sofisticado con limitantes en el número de entradas en su memoria TCAM (8K a lo máximo).

1.2 Objetivos de la tesis

1.2.1 Objetivos General

- Diseño e implementación de un sistema de balanceo de carga vía un *switch SDN/OpenFlow* para un banco de firewall N:1
- Garantizar que el diseño tenga el mismo o mejor rendimiento que el modelo de protección activo – respaldo en presencia de picos de tráfico

1.2.2 Objetivos Específicos

- Análisis de trazas de una red empresariales o similar para la recolección de información sobre la dinámica de las sesiones
- Diseño del algoritmo de balanceo de carga y del procedimiento de migración de una subred en respuesta a picos de tráfico
- Implementación de módulos, en Python, como aplicación en el controlador *SDN/OpenFlow* para que los *switchs SDN/OpenFlow* operen como balanceadores de carga
- Simulación del tráfico IP para realizar pruebas de rendimiento del balanceador de carga

1.3 Hipótesis de la tesis

La adaptación de un *switch SDN/OpenFlow* para que realice las funcionalidades de un balanceador de carga sin interrumpir o migrar las sesiones activas ya establecidas en algún elemento del modelo N:1 tiene un rendimiento igual o superior a las soluciones comerciales lo cual permitirá reemplazar el modelo de protección activo – respaldo (solución *Legacy*) por un banco de *firewalls* N:1.

1.4 Razones que motivaron a desarrollar el trabajo

La presente tesis se desarrolla por el incentivo de diseñar e implementar soluciones basadas en software libre que puedan ser el reemplazo a los modelos de seguridad comerciales (*Legacy*) y ser mejorados por la comunidad de código abierto.

Este trabajo pretende ser un aporte al proyecto PICASSO (Protección Inteligente contra ciberataques Sobre redes *SDN/OpenFlow*), en desarrollo por el GIRA (Grupo de Investigación de Redes Avanzadas), a través de la implementación de uno de sus módulos (Firewall Load Balancing). PICASSO propone reducir aún más el costo de capital (CaPex) en la adquisición de equipos de seguridad con un rendimiento superior a las soluciones *Legacy*. Para ello implementará su módulo (*Firewall Bypass*) que consiste en la identificación de flujos elefantes (tráfico voluminoso no malicioso), que constituyen el veinte por ciento del tráfico total de una red empresarial, para que no sean procesados por los firewalls transportándolos por una interfaz especial (1GigE). Esto reducirá la cantidad de *firewall* a utilizar en el modelo de protección N:1.

Otras de las razones que motivaron este trabajo es que la solución propuesta pueda ser utilizada en empresas (Mypes) o escenarios similares a la PUCP donde la inversión en sistemas de seguridad excede el presupuesto de la empresa por los altos costos que ofrecen las soluciones comerciales. Por ende, se reducirá la brecha digital de seguridad en las Mypes.

1.5 Estructura de la tesis

La presente tesis está compuesta por 4 capítulos que a continuación se describen:

En el capítulo 1, se describe la problemática, hipótesis, objetivos y la motivación de la presente tesis.

En el capítulo 2, se documenta los conceptos que se utilizaran para el desarrollo del diseño del balanceador de carga para el modelo de protección N:1.

En el capítulo 3, el diseño del balanceador de carga.

En el capítulo 4, las pruebas de concepto del balanceador de carga y la conclusión de la tesis.

2. MARCO TEÓRICO

A continuación se presentan los conceptos más importantes para el diseño del balanceador de carga vía un *switch SDN/OpenFlow* descrito en el capítulo 3.

2.1 Software Defined Network (SDN)

Las redes IP empresariales tradicionales son complejas en su administración ya que están sujetas a políticas predefinidas y posteriores reconfiguraciones de estas ante fallas. Los elementos de una red IP tradicional, *routers* o *switches*, están verticalmente integrados (los planos de control y datos están empaquetados) lo que dificulta la administración de la red en escenarios complejos (red de campus PUCP) ya que la configuración de cada elemento es manual. SDN es un paradigma cuyo propósito principal es desacoplar los planos de control y datos para que la inteligencia de la red se centralice en un elemento llamado controlador [2]. Las redes basadas en SDN abstraen sus elementos y roles en el controlador para implementar aplicaciones (balanceador de carga) que la convierten en una red programable en función a los requerimientos de los usuarios [3].

2.1.1 RFC 7426 Arquitectura

En síntesis, la arquitectura SDN según el RFC 7426 está conformada por tres capas. En primer lugar, la capa de infraestructura o DAL (*Device and Resource Abstracion Layer*) que

está conformada por los distintos dispositivos de red que son simples elementos de reenvió de paquetes. En segundo lugar, la capa de control conformada por la CAL (*Control Abstraction Layer*) y MAL (*Management Abstraction Layer*) que se encargan de recibir y reenviar los paquetes provenientes de la capa de infraestructura. Por último, la capa de aplicación o NSAL (*Network Services Abstraction Layer*) que abstrae los elementos de la red para el desarrollo de aplicaciones que hacen posible la implementación de servicios de red como enrutamiento, seguridad, manejo del ancho de banda, ingeniería de tráfico, calidad de servicio, etc.

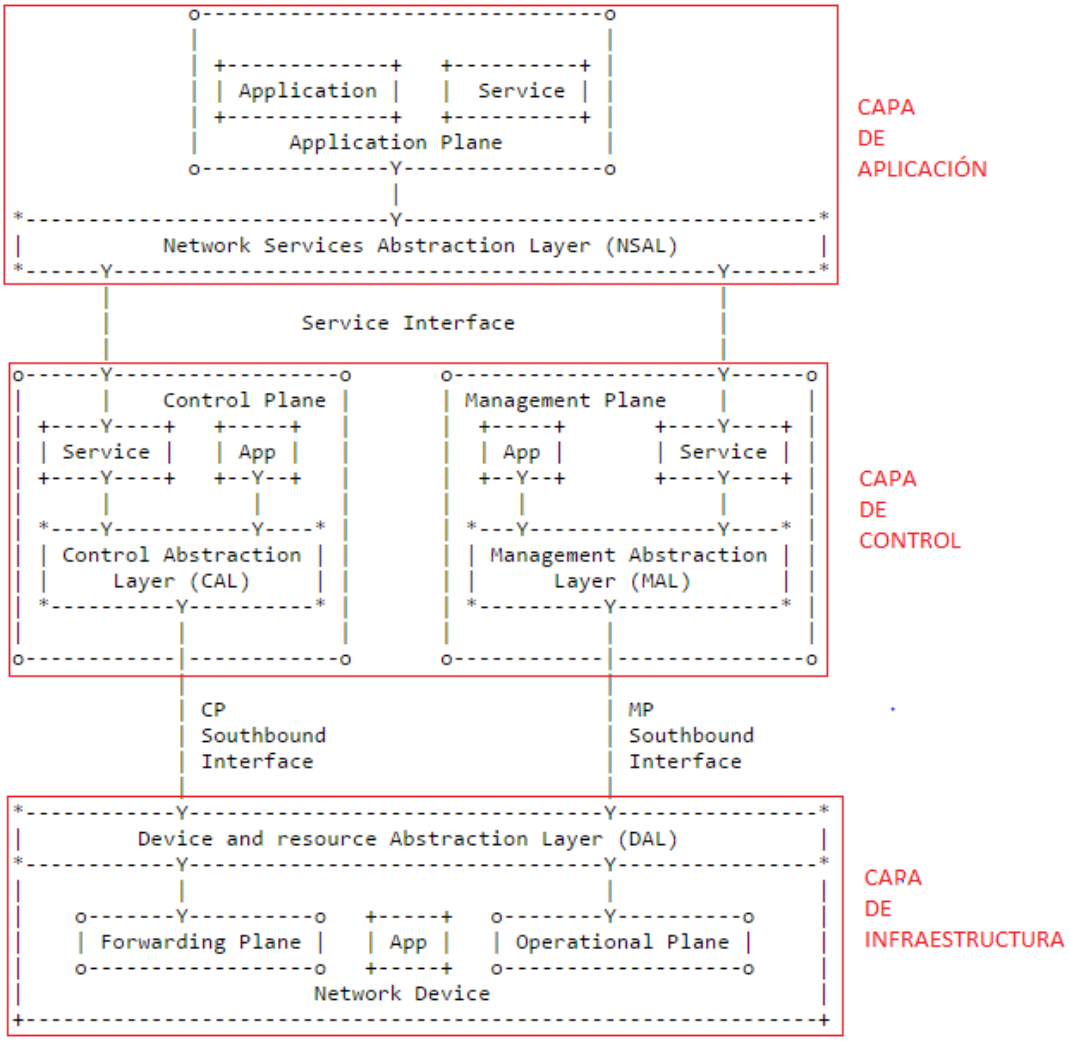

Figura 2-1 Arquitectura SDN

Fuente: [4]

La Figura 2-1 presenta las distintas capas mencionadas en la arquitectura SDN según el RFC 7426.

Una interfaz es un punto de interacción entre dos entidades. Las dos interfaces más importantes son la *SouthBound* Interface (que se encarga de comunicar los elementos de la

capa de infraestructura con el controlador SDN, capa de control, a través del protocolo *OpenFlow*) y la *NorthBound* Interface (que comunica la capa de control con la capa de aplicación a través de la interfaz de programación de aplicaciones API) [5].

2.1.2 Interface SouthBound – OpenFlow

OpenFlow es el primer protocolo SDN aceptado por la comunidad de investigadores y vendedores por ser de código abierto y es utilizado como interface *SouthBound*. Esta interfaz define la comunicación entre los dispositivos del plano de datos, *switch OpenFlow*, y los elementos del plano de control, controlador SDN.

Un *flow entry* es una abstracción de una regla de *Forwarding* que está compuesta por los siguientes elementos más importantes: *Match fields*, campo donde se definen las características (inspección de cabeceras L2/L3/L4) que deberá tener un paquete para pertenecer al *flow*; *Counter*, contador que se actualiza cuando un paquete es emparejado al *flow*; *Instruction*, ejecuta o modifica una acción; *Timeouts*, tiempo de espera para que un *flow entry* sea borrado automáticamente de la memoria TCAM; *Priority*, nivel de prioridad del *flow entry* (si dos *flow entries* comparten el mismo campo *Match Field* se ejecutará la acción del *flow entry* de mayor prioridad).

Un *flow table* es una abstracción de la tabla de *Forwarding* que está compuesto por un conjunto de *flow entries*.

OpenFlow-enabled Network Device

Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:..	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 2-2 Elementos de la *Flow Table*

Fuente: [3]

En este caso si un paquete tiene una dirección IP destino igual a 5.6.7.8 se procederá a reenviarlo por el puerto 2.

Un *group entry* es una regla que especifica un conjunto de acciones, un *flow entry* puede especificar un *group entry* para realizar métodos más sofisticados de *forwarding*.

Un *group table* es un conjunto de *group entries*.

Un *switch OpenFlow* consiste en uno o más *flow tables*, un *group table* y un canal *OpenFlow*.

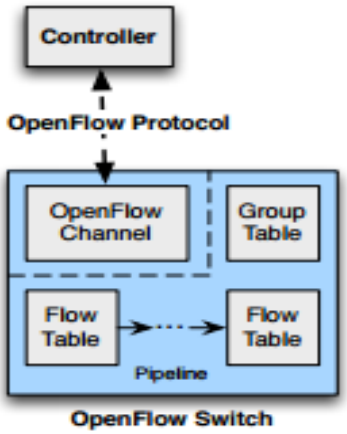

Figura 2-3 Principales componentes de un *switch*

Fuente: [5]

El *switch* se comunica con el controlador a través del *OpenFlow Channel*, a través de esta interfaz el controlador puede gestionar, enviar y recibir eventos provenientes del *switch* vía el protocolo *OpenFlow*. Además, el controlador puede agregar, actualizar o eliminar *flow entries* u *flow tables* del *switch* de forma reactiva o proactiva [5].

Los elementos de la capa de infraestructura, *firewalls*, IDS, servidores, *switchs* que están conectados a una de las interfaces físicas del *switch OpenFlow* serán dependiente de las acciones definidas en los *flow tables* del mismo cuando un paquete se empareje con un *flow entry* del *switch OpenFlow*.

2.1.3 Granularidad de los flow entries

Los *switch SDN/OpenFlow* utilizan en *hardware* memorias TCAM (*Ternary Content Addressable Memory*) para la inserción de *Flow Entries* en las tablas de bases de enrutamiento o FIB (*Forward Information Base*) del mismo. La memoria TCAM incrementa la velocidad de búsqueda de los flujos (en comparación a la memoria CAM) ya que maneja un tercer estado *("don´t care")* que permite una búsqueda en *line-rate* de las diferentes tuplas definidas en la *Flow Tables* del *switch*.

La granularidad de un *Flow Entry* al nivel L2/L3/L4 (versión *OpenFlow* 1.3) es útil para la identificación de sesiones TCP/UDP. Sin embargo, en escenarios como una red empresarial donde en un instante dado aparecen 80K sesiones activas (caso PUCP) aumentar la granularidad de los *Flow Entries* al nivel de una sesión resulta un problema debido a las limitación físicas de la memoria TCAM del *switch SDN/OpenFlow*.

Tabla 2-1 Limitantes físicas (TCAM) en los *switch* PICA8

PICA8 Model	Price	Chipset	TCAM Size	FIB Size	All tuples in TCAM	IPv4 in TCAM	IPv6 in TCAM	MAC in TCAM	ARP in TCAM	IPv4 in FIB	IPv6 in FIB
P-3290	-	Firebolt3	108KB	32K	2048	4096	1024	4096	2048	12000	12000
P-3297	$ 3,960.00	Triumph2	216kB	32K	4096	8192	2048	8192	4096	12000	12000
P-3922	$ 8,280.00	Trident+	54kB	128K	1024	2048	512	2048	1024	12000	12000
P-5401	$ 10,440.00	Trident2	108kB	Common FIB and L2 Memory	2048	4096	1024	4096	2048	105000	105000

Fuente: [6]

En la Tabla 2-1 se especifica las limitantes físicas de las memorias TCAM en una serie de *switchs* PICA-8. Un *flow entry* o entrada ocupa un espacio en la memoria TCAM. La cantidad de *Flow Entries* que se pueden insertar en la FIB de un *switch SDN/OpenFlow* está limitado por el espacio de su memoria TCAM. Para propósitos de esta tesis, la limitante será 8000 entradas como máximo pues se trabajaran con los *switchs* P-3297 (arquitectura de procesador *Broadcom Triumph2* que soporta 8192 direcciones IPv4 en su TCAM) disponibles en el laboratorio GIRA-PUCP.

2.1.4 Interface NorthBound

La interfaz *NorthBound* permite la comunicación entre el controlador SDN y las aplicaciones SDN a través de una API que abstrae los recursos y dispositivos de la red. La proliferación de controladores SDN con sus propias API no ha dejado una estandarización de esta interfaz; a diferencia de la *SouthBound*, donde el protocolo *OpenFlow* es el estándar por consenso entre los vendors y la comunidad *Open Source*. Sin embargo, existen esfuerzos para normalizar esta interfaz [7].

2.1.5 REST API para desarrollo de aplicaciones

La interfaz REST API (*Representational State Transfer*) es un tipo de arquitectura o *framework* para el desarrollo de aplicaciones basadas en el protocolo HTTP. Las características de esta interfaz son las siguientes:

- Utiliza el modelo cliente / servidor
- Utiliza el protocolo HTTP para la comunicación entre el cliente y el servidor
- Sus operaciones más importantes en relación al manejo de datos son POST(crear), GET(leer y consultar), PUT(editar) y DELETE(eliminar)

- Un objeto REST siempre se manipula a través de una URL para generar las consultas
- Transfiere información en formato XML o JSON

La REST API facilita el acceso, inserción y eliminación de información (consumo de tráfico en un puerto físico del *switch*, cantidad de paquetes y bytes emparejados a un *Flow Entry*, lista de DPID de los *switch*, creación de *Flow Entries*, eliminación de *Flow Entries* y más) que ofrecen las distintas funcionalidades del controlador SDN.

Tabla 2-2 REST API para la implementación de un balanceador de carga

Módulo	URL	Método	Descripción
Controller APIs	/wm/core/controller/switches/json	GET	Lista todos los switch DPIDs conectados al controlador
OpenFlow Stats / Multipart APIs	/wm/core/switch/<switchId>/flow/json	GET	Recuperar las caracteristicas de todos los flow exitentes en un switch
Statistics APIs	/wm/statistics/config/enable/json	POST / PUT	Habilita la colección de estadisticas
	/wm/statistics/bandwidth/<switchId>/<portId>/json	GET	Extraer el consumo de bps RX/TX por puerto en un switch
Static Entry Pusher APIs	/wm/staticflowpusher/json	POST / DELETE	Agregar y eliminar un flow estatico
Session Actives APIs (Desarrollado)	/start?<prefix_subred>	GET	Iniciar la recoleccion de sesiones activas de una subred
	/query?<prefix_subred>	GET	Dar la colección de sesiones activas de una subred
	/remove?<prefix_subred>	GET	Detener la recolección de sesiones activa de una sub red

Fuente: Elaboración propia

A continuación se presentan las funcionalidades más importantes de la REST API *Floodlight* para el desarrollo de esta tesis.

2.1.6 Controlador SDN

Un controlador ofrece una interfaz de comunicación con los elementos de red, *switch OpenFlow*, y una interfaz para la interacción con las aplicaciones de servicio de red. Un controlador se puede definir como un conjunto de módulos de software que ofrece lo siguiente:

- Gestión del estado de los elementos o la topología de la red
- Mecanismo de descubrimiento de dispositivos, topología y cálculo de rutas

- REST API que expone los servicios del controlador, recolección de estadísticas, enrutamiento y más a las aplicaciones *northbound*

	Beacon	Floodlight	NOX	POX	Trema	Ryu	ODL
Soporte OpenFlow	OF v1.0	OF v1.0	OF v1.0	OF v1.0	OF v1.3	OF v1.0, v1.2, v1.3 y extensiones Nicira	OF v1.0
Virtualización	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Construcción de una herramienta virtual de simulación	Mininet y Open vSwitch	Mininet y Open vSwitch
Lenguaje de desarrollo	Java	Java	C++	Python	Rudy/C	Python	Java
Provee REST API	No	Si	No	No	Si (Básica)	Si (Básica)	Si
Interfaz Gráfica	Web	Web	Python+, QT4	Python+, QT4, Web	No	Web	Web
Soporte de plataformas	Linux, Mac OS, Windows y Android para moviles	Linux, Mac OS, Windows	Linux	Linux, Mac OS, Windows	Linux	Linux	Linux, Mac OS, Windows
Soporte de OpenStack	No	Si	No	No	Si	Si	Si
Multiprocesos	Si	Si	Si	No	Si	No	Si
Código Abierto	Si	Si	Si	Si	Si	Si	Si
Tiempo en el mercado	4 años	2 años	6 años	1 años	2 años	1 años	5 meses
Documentación	Buena	Buena	Media	Pobre	Media	Media	Media

Figura 2-4 Características de algunos controladores SDN

Fuente: [8]

El controlador SDN administra las políticas de la red a partir de los requerimientos de los servicios a integrar. La variedad de controladores en el mercado los distingue por su arquitectura, el lenguaje que están escritos, la inclusión de una interfaz REST API, inclusión de una GUI, dependencia con *OpenStack*, código abierto, documentación, etc. A continuación se presentan una serie de controladores SDN de código abierto.

Cada controlador varía el diseño de su arquitectura. Sin embargo, todos comparten la siguiente arquitectura idealizada.

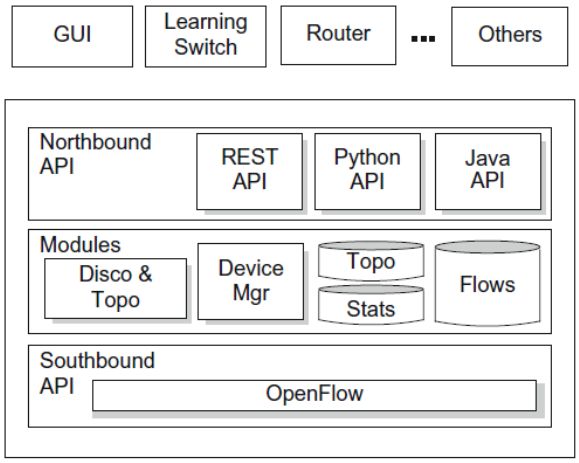

Figura 2-5 Arquitectura del controlador SDN idealizado

Fuente: [8]

2.2 Modelos de protección y sus elementos

2.2.1 Modelo activo – respaldo (Legacy)

El modelo activo - respaldo consiste en la adquisición de dos equipos de protección; por ejemplo, 2 ASA (*Adaptative Security Appliance*) Cisco *Firewall*, que operan en función al *Failover*. Es decir, el equipo activo maneja todo el procesamiento para filtrar el tráfico maligno mientras el equipo en modo respaldo, que es el equipo de contingencia del sistema, no realiza operación alguna hasta que el equipo activo falle y entre en operación.

Cuando se produce una conmutación por error debido a sobrecargas u otros factores en el elemento activo se envían sus sesiones activas, TCP/UDP existentes, al equipo de respaldo.

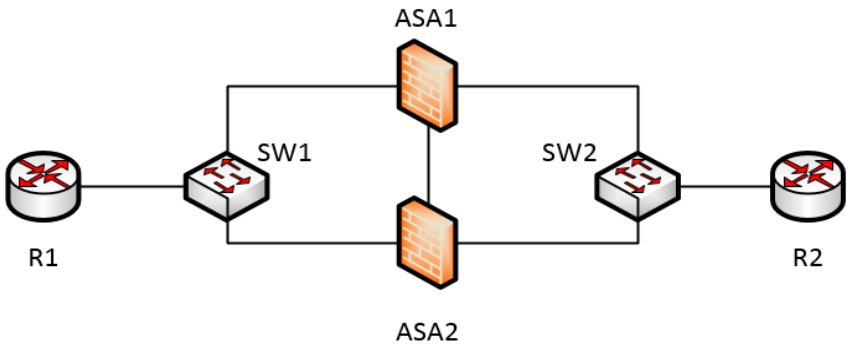

Figura 2-6 Modelo de protección activo - respaldo

Fuente: Elaboración propia

En la Figura 2-6 se presenta la configuración de los *firewalls* según el modelo activo - respaldo para proteger los recursos de la red empresarial.

2.2.2 Modelo N:1 o banco de firewalls

El modelo N:1 o banco de *firewalls* para propósitos de esta tesis es una extensión del modelo activo - respaldo. La diferencia consiste en que hay una cantidad N de equipos en modo activo y uno en respaldo, que opera en modo de contingencia. Cuando uno de los elementos activos falle por sobrecarga o *Failover* el equipo en modo de respaldo operará en su reemplazo.

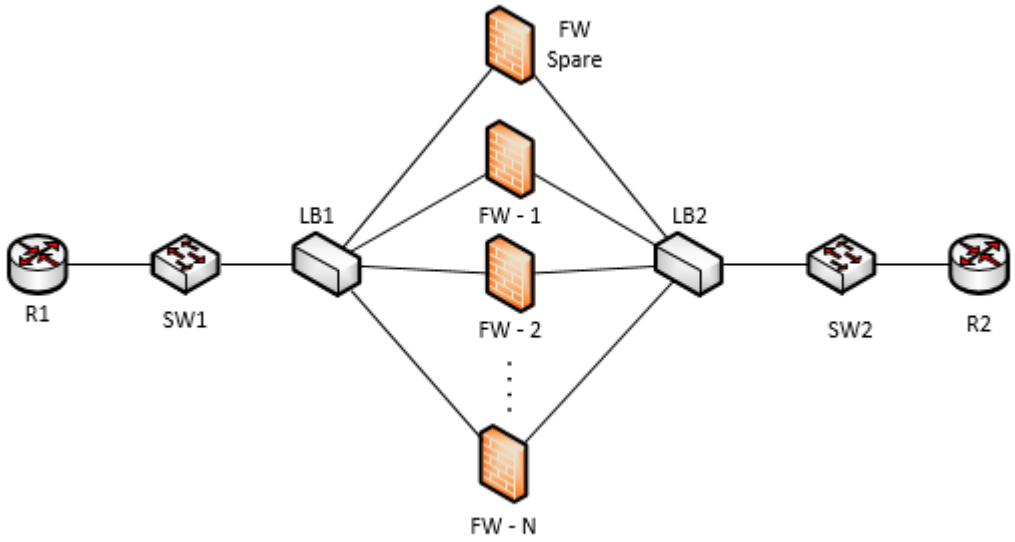

Figura 2-7 Protección de la red a través de un banco de Firewalls N:1

Fuente: Elaboración propia

A diferencia del modelo activo - respaldo, los nuevos elementos que aparecen son los balanceadores de carga (LB1, LB2) cuya función es redistribuir el tráfico IP que proviene de la intranet hacia internet y viceversa a través de los N *firewalls* activos del sistema.

2.2.3 Firewalls de primera generación o tradicionales

Los *firewalls* tradicionales o de primera generación se basan en el filtrado de paquetes. Se definen una serie de reglas, si un paquete coincide con una de las políticas definidas se procederá a permitirlo o denegarlo.

La herramienta *iptables*, componente del *framework Netfilter*, que está integrado al *kernel Linux* forma parte de la primera generación de *Firewalls* como servicio que incluye este sistema operativo.

Las dificultades que presentan la configuración y mantenimiento de reglas en los firewalls tradicionales es una limitante ya que es incapaz de prevenir vulnerabilidades como ataques SYN *floodings* (no maneja los estados de conexión de las sesiones TCP/UDP).

2.2.4 Firewalls de segunda generación o stateful

La segunda generación de *firewalls* toma en cuenta que la mayor parte del tráfico de internet está compuesto por conexiones TCP/UDP. Las sesiones TCP que manejan estados para su inicialización (*three-way handshake*), envío (*acknowledgement*) y finalización (*four-way handshake*) son propensas a ser distorsionadas para generar ataques SYN *floodings* o DoS. Los *firewalls* de inspección de estados resuelven estos problemas a través de la inserción de las sesiones TCP/UDP activas en una tabla de sesiones especificando sus características más importantes (puertos TCP/UDP origen y destino) para que se deniegue a los futuros paquetes que no pertenecen a estas sesiones. Un subsistema de *iptables* que se encarga de realizar los seguimientos de las conexiones es *"connection tracking system"*.

2.2.5 Firewalls de tercer y cuarta generación

La tercera generación de firewalls evalúa los paquetes hasta el nivel de aplicación manteniendo la inspección de estados. Además, brinda servicios de autenticación de usuarios y *Proxy* para controlar un protocolo específico (HTTP, FTP, DNS). La capacidad de inspeccionar por completo el paquete de datos evita ataques de suplantación (*spoofing*) asegurando un alto grado de protección a la red. La nueva generación de *firewalls* incluye una plataforma donde convergen las funcionalidades más importante de un sistema de protección como Inspección profunda de paquetes, IPS (*Intrusion Prevention System*), encriptación del tráfico por TLS/SSL, inspección de antivirus, RADIUS, filtrado de aplicaciones web o WAF(*Web Application Firewall*), administración de ancho de banda y más. La multiplicidad de funcionalidades es proporcional a los costos de adquisición de los NGFW.

En el siguiente gráfico se expone la evolución temporal de las generaciones de *firewalls*.

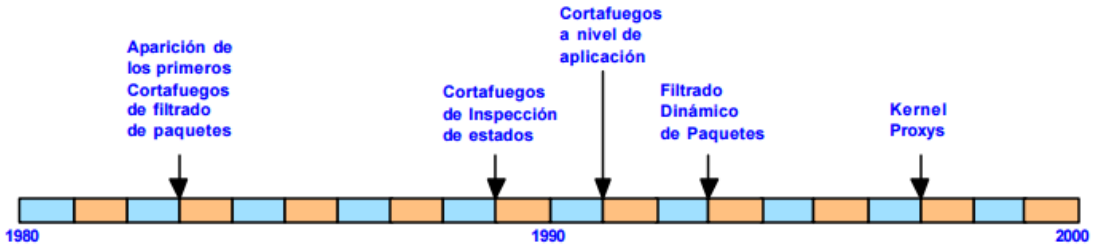

Figura 2-8 Evolución de los *firewalls*

Fuente: [9]

Una característica importante en los firewalls a partir de la segunda generación es su sensibilidad a las sesiones.

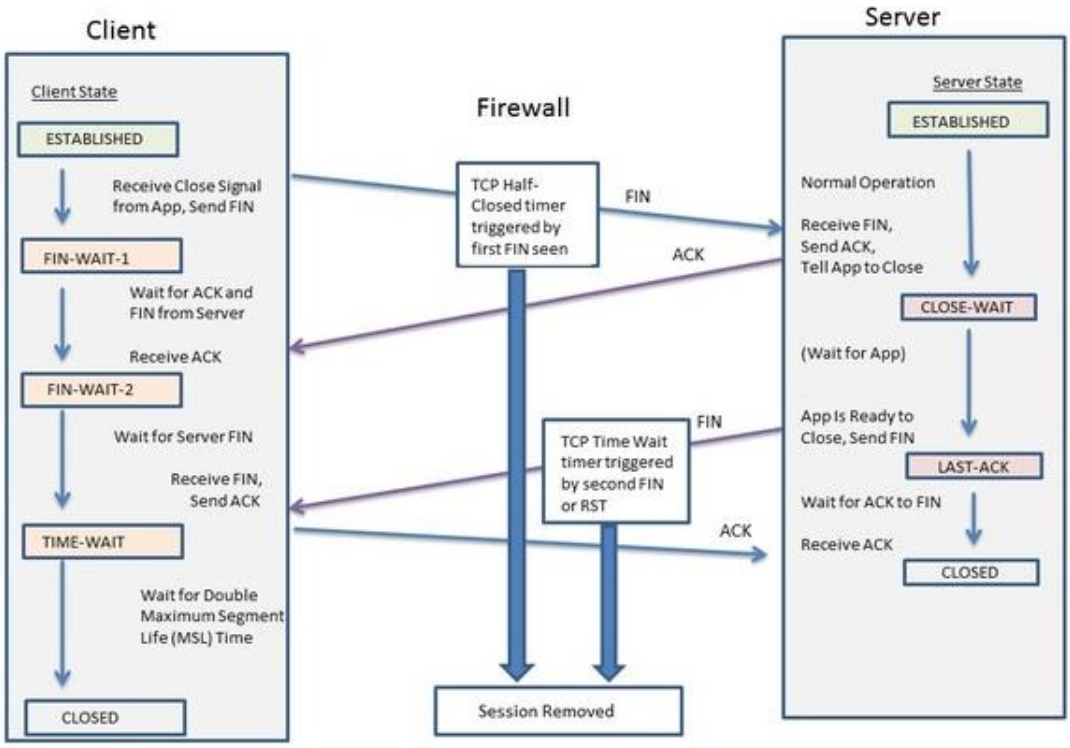

Figura 2-9 *Firewall* de la firma Palo-Alto Networks

Fuente: [10]

Por ejemplo, en *firewalls* comerciales como Palo Alto Networks *Firewalls*, con sistema operativo PAN-OS, se utiliza dos tiempos: TCP *Half Closed* y TCP *Wait Timers* para cerrar las conexiones TCP [10]. Cuando el *firewall* recibe el primer FIN y después el segundo FIN o RST se activan los temporizadores TCP *Half Closed* y TCP *Wait Timers* respectivamente. Cuando estos expiran se procede al cierre definitivo de la conexión TCP. El tiempo de duración del TCP *Wait Timer* por defecto es 15 segundos. Es decir, un *firewall* de la marca Palo-Alto olvida una conexión TCP después de dicho tiempo.

2.3 Balanceadores de carga

Los balanceadores de carga distribuyen la carga, virtualizando el servicio, a un grupo de servidores físicos para hacer que estos trabajen como un solo servidor. Estos surgieron por la incapacidad del manejo de tráfico de un solo servidor y la necesidad de utilizar clúster. La primera solución para el balanceo de carga a un conjunto de servidores fue a través de la repartición *Round-Robin* vía DNS (*Domain Name System*).

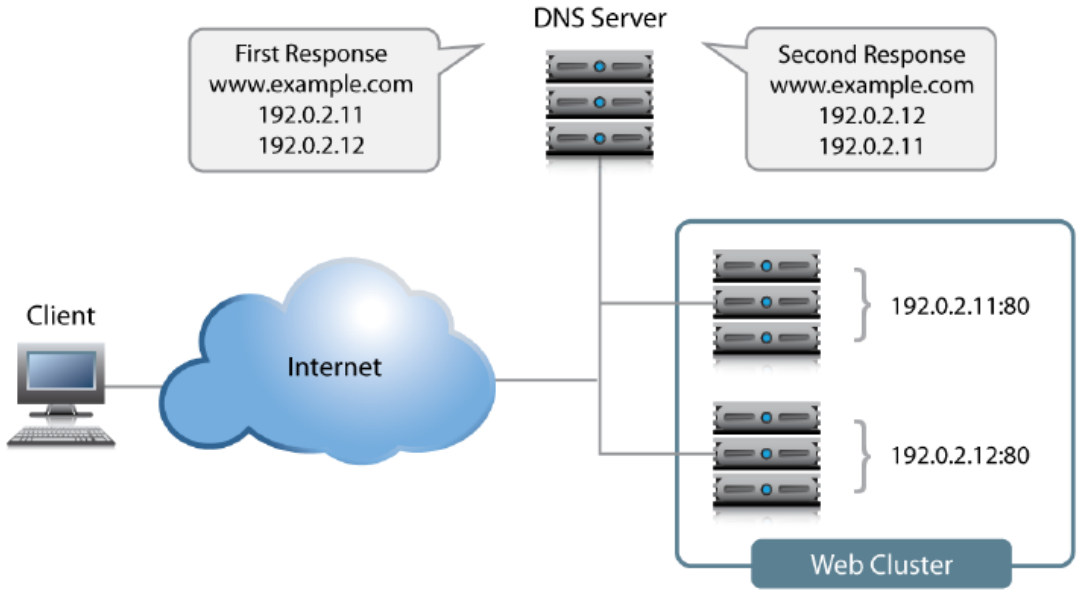

Figura 2-10 Balanceador de carga vía DNS

Fuente: [11]

El principio de funcionamiento se basa en la capacidad del servidor DNS para responder cada solicitud a un nombre de dominio (www.example.com) a múltiples IPs (192.0.2.11 / 192.0.2.12). Cuando un conjunto de usuarios soliciten a un nombre de dominio (servicio web), el servidor DNS brindara los recursos del primer servidor (192.0.2.11:80) al primer usuario, recursos del segundo (192.0.2.12:80) al segundo usuario, recursos del primero (192.0.2.11:80) al tercer usuario y así sucesivamente distribuyendo la carga en los distintos servidores *web* del *clúster*. Sin embargo, este mecanismo no tiene la capacidad de conocer la lista de servidores operativos y manejar el estado de las sesiones activas. Si el primer servidor (192.0.2.11) falla se interrumpe la sesión establecida por el primer usuario, se pierde el estado de la sesión y se brindará los recursos del segundo servidor (192.0.2.12) al primer usuario. Esta limitante es crítica en escenarios donde los servidores son reemplazados por *firewalls*, equipos sensibles a los estados de las sesiones.

2.3.1 Balanceadores comerciales

Los controladores de entrega de aplicaciones (ADC) o balanceadores comerciales se encargan de administrar a las aplicaciones basadas en los protocolos TCP o UDP para el balanceo, encriptación, aceleración y seguridad de estas. El mecanismo de balanceo de carga en el ADC se basa en la virtualización de los servicios. Se incluye un servidor virtual que brinda una “dirección IP virtual” para el mundo exterior, cuando un cliente accede a esta dirección, solicitando el servicio, se procede a realizar un NAT – bidireccional hacia el servidor físico más apropiado.

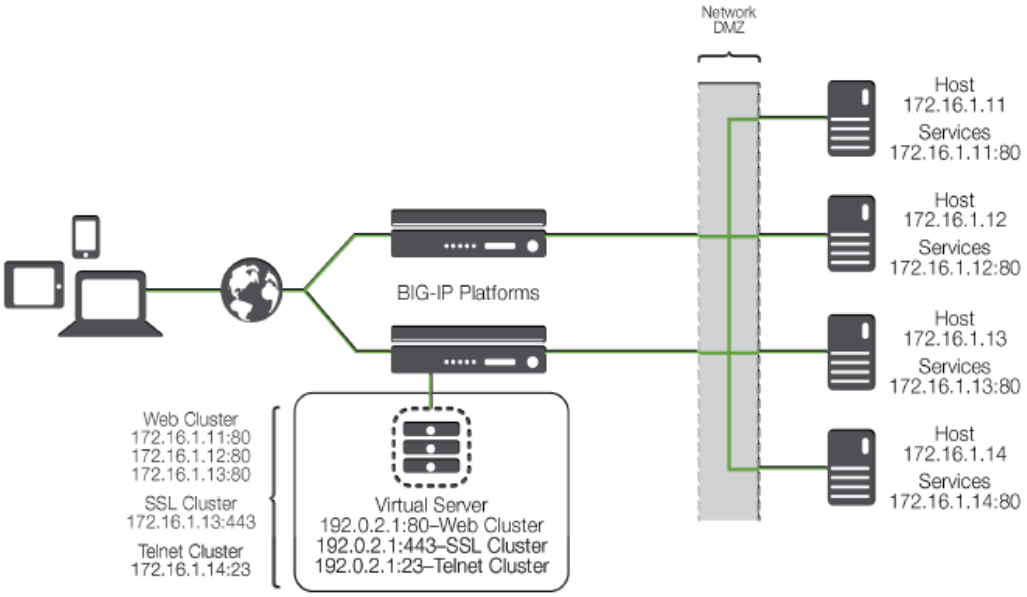

Figura 2-11 Balanceo de carga en las ADC

Fuente: [12]

Los ADC trabajan de la siguiente forma:

- Un cliente (192.168.0.11) en el exterior solicita un servicio (*web*), que provee uno de los servidores físicos en el clúster, al servidor virtual (192.0.2.1:80)
- El ADC acepta la conexión y decide que servidor físico debe recibir la conexión (172.16.1.11, 172.16.1.12, 172.16.1.13 o 172.16.1.14), se cambia la IP definida en el servidor virtual(192.0.2.1) por la IP del servidor físico elegido (172.16.1.11)
- El servidor físico elegido (172.16.1.11) responde la solicitud y establece la sesión con el usuario
- El ADC intercepta la respuesta del servidor físico y cambia la IP (NAT) de este por la IP del servidor virtual (192.0.2.1) enviándolo al cliente
- El cliente recibe la respuesta del ADC creyendo que el servicio proviene del servidor virtual
- El proceso continua indefinidamente hasta que la sesión establecida entre el cliente y servidor expire

La gran diferencia de los ADC frente a los balanceadores de carga *Round-Robin* vía DNS es que estos recuerdan las conexiones establecidas por los usuarios y las almacenan en una tabla de conexiones. Estos balanceadores de carga se adaptan a la problemática de distribuir la carga a un banco de *firewall* N:1.

2.3.2 Problemas en los balanceadores comerciales (costos)

En el mercado encontramos balanceadores comerciales como las soluciones BIG-IP o *ServerIron* ADX de las firmas F5 y Brocade respectivamente. La eficiencia de los ADC se basa en el nivel de granularidad para detectar una sesión y garantizar su permanencia hasta su expiración. Redes empresariales como la PUCP manejan 80 000 sesiones simultaneas en un momento dado [13]. Los ADC, ServerIron ADX, manejan entre 93 750 a 3 000 000 conexiones por segundo [14]. La alta granularidad de los ADC influye en los costos del *hardware* ya que utilizan memorias sofisticadas, TMM en caso de la solución BIG-IP [15], para el almacenamiento de las conexiones en su tabla de sesiones (*tabla hash*).

Tabla 2-3 Comparación entre las soluciones BIG-IP y ServerIron ADX

Caracteristicas/ADC	BIG-IP i2600	ServerIron ADX 1016-4
Conexiones por segundo (L7)	350 000	375 000
Máxima cantidad de sesiones concurrentes (L4)	14 000 000	16 000 000
Throughput (Gbps) (L4/L7)	10	9
Precio	$ 19 127.99	$ 20 369.99

Fuente: Elaboración propia

La Tabla 2-3 muestra una lista de precios de los balanceadores comerciales especificando sus características más importantes.

2.3.3 Switch SDN/OpenFlow como balanceador de carga

El controlador SDN permite la inserción, eliminación y actualización de *Flow entries* en el *switch SDN/OpenFlow*. Un *Flow Entry* puede manejar una acción de balanceo de carga.

Name	Switch DPID	MAC Src	MAC Dst	Eth Type	Protocolo	IP Src	IP Dst	TCP/UDP Sport	TCP/UDP Dport	Action
Flow 1	DPID switch	*	*	0x0800	TCP	192.168.1.18	10.20.101.21	80	65460	port 1
Flow 2	DPID switch	*	*	0x0800	UDP	IP Sub-Red 1 /mask	*	*	*	port 2
Flow 3	DPID switch	*	*	0x0800	UDP	192.168.1.20	103.45.67.3	8080	12345	port 3
...	...	...	...	...	...	...	...	...	...	...
Flow M	DPID switch	*	*	0x0800	TCP	IP Sub-Red 2 /mask	*	*	*	port X

Figura 2-12 *Flow table* de un balanceador de carga basado en un *switch SDN/OpenFLow*

Fuente: Elaboración propia

Por ejemplo, se puede asignar para que todos los paquetes que provienen de una sesión en particular (IP fuente = 192.168.1.18, IP destino = 10.21.101.21, protocolo = TCP, puerto fuente = 80 y puerto destino = 65460) se redirijan al *firewall* próximo al puerto 1 del *switch* a través de la inserción del *flow entry* (*Flow* 1) en la memoria TCAM del *switch*. También se

puede asignar que todo el tráfico de una subred en particular sean procesados por el *firewall* próximo al puerto 2 del *switch* a través del *flow entry* (*Flow* 2).

Un *switch SDN/OpenFlow* (P - 3297), costo 3000 dólares, y un controlador SDN se adaptan para realizar las funciones de un balanceador de carga.

2.3.4 Límite de granularidad debido al tamaño de las TCAM

Las entradas (*flow entries*) que soporta la memoria TCAM del *switch SDN/OpenFlow* están limitado por el procesador que utilizan. Por ejemplo, el modelo P-3297 soporta como máximo 8000 entradas.

Esta limitante afecta el nivel de granularidad de una regla de balanceo de carga. Ya que por ejemplo, si se definen reglas a la granularidad de una sesión en un balanceador de carga basado en un *switch SDN/OpenFlow* inspirado en el mecanismo de funcionamiento de los ADC se excederá en el uso de recursos disponibles. *El switch SDN/OpenFlow* no manejará 80K sesiones activas (cifra que aparece en una red empresarial) simultaneas para realizar el balanceo de las sesiones a determinados firewalls.

Esta limitante es una dificultad en la adaptación de un *switch SDN/OpenFlow* como balanceador de carga.

2.4 Tráfico en las redes de banda ancha

Estudios sugieren que el patrón del tráfico en las redes de banda ancha es auto similar [16]. Es decir, la visualización del tráfico en distintos grados de ampliación revela su naturaleza fractal [17].

2.4.1 Flujos de internet o sesiones

El tráfico de una red basada en conmutación de paquetes está conformado por un conjunto de flujos agrupados por las características de la cabecera IP y TCP/UDP. En la capa 3 del modelo OSI el formato de la cabecera Internet está definido por el RFC791 [18].

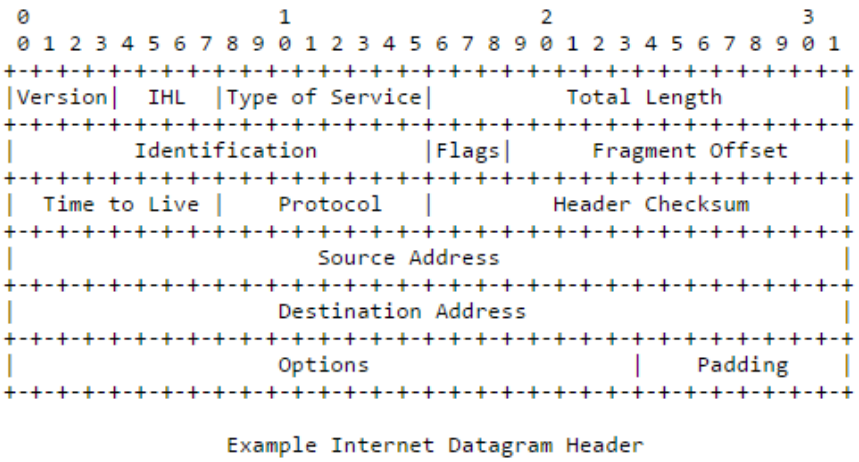

Figura 2-13 Cabecera del datagrama Internet

Fuente: [18]

Asimismo, las cabeceras de los protocolos TCP y UDP están definidos en los RFCs 793 y 768 respectivamente [19] [20].

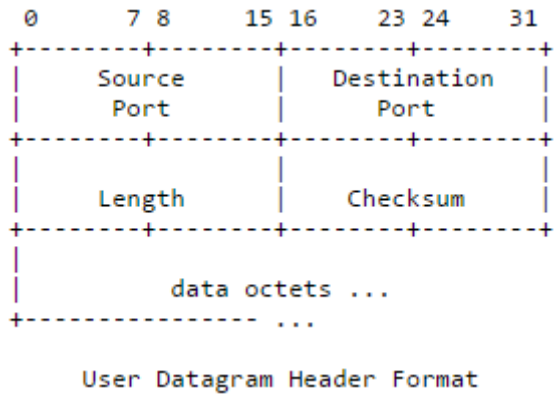

Figura 2-14 Formato de cabecera del protocolo UDP

Fuente: [20]

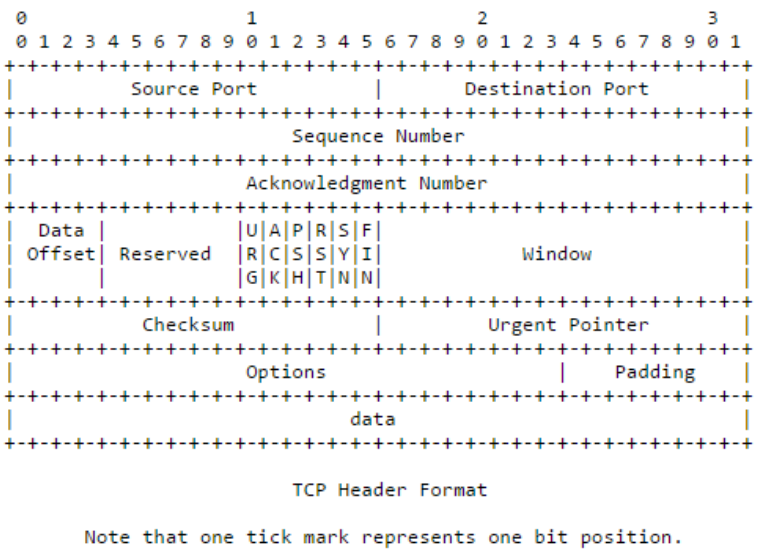

Figura 2-15 Formato de cabecera del protocolo TCP

Fuente: [19]

La definición de un flujo o sesión para este trabajo está conformada por un vector cuyos elementos son la dirección IP fuente, destino; el tipo de protocolo; y los puertos TCP o UDP fuente y destino. Las características más importantes de los flujos son: identificador del flujo, cantidad de paquetes, tamaño del flujo (*bytes*) y arreglo de los tiempos de llegada de los paquetes que conforman el flujo.

Flow = {identificador, Cantidad de paquetes, Tamaño, Tiempo de Llegada []}

Donde, identificador: "ip_fuente, ip_destino, número del protocolo, puerto_fuente, puerto_destino"

2.4.2 Trazas de tráfico ISPDSL-II

La captura de trazas de tráfico se realiza utilizando *splitter* ópticos unidos a tarjetas DAG6.2SE [21] o DAG 3.7G con un *software* de captura de trazas, WDCAp [22].

Para esta tesis, se utiliza la serie de trazas públicas en [22] (ISPDSL-II) que pertenecen al tráfico entre un Proveedor de servicios (ISP) y un conjunto de clientes residenciales a través de líneas de abonado digital que se asemeja al comportamiento de una red empresarial.

La recolección de paquetes en la traza ISPDSL-II se realizó a través de un *port mirror* en el *switch* que carga el tráfico en ambas direcciones hacia el *router* ISP. El tráfico capturado fue escrito en el formato ERF, compatible con pcap, en una serie de 562 archivos con la siguiente notación yyyymmdd-HHMMSS-[code].gz.

En efecto las trazas ISPDSL-II contienen la información de los paquetes capturados desde las 16:09:46 del miércoles 6 de junio del 2010 hasta las 09:30:01 del lunes 18 de junio del 2010 referidos al tiempo en UTC.

2.4.3 Taxonomía del tráfico en redes empresariales

El estudio de la naturaleza del tráfico de Internet en las redes empresariales es importante para la determinación de políticas que protejan la seguridad de la red. El tráfico de Internet está conformado en mayor porcentaje por el protocolo IP.

El estudio de los flujos de tráfico en [23] concluye que los flujos con más tiempo de duración, superior a los 10 minutos, representan el 20% del volumen de tráfico total y son aquellas sesiones peer-to-peer.

Para realizar la clasificación del tráfico en la traza ISPDSL-II se utiliza el script desarrollado por el autor de esta tesis y la librería *pypcapfile* de Python con algunas modificaciones, estas herramientas están almacenadas en [24] [25].

El primer análisis se realiza a la traza, 20100106-030946-0.dsl [22], de 20 minutos de duración. En esta se procesan 23 millones de paquetes, de los cuales se encuentran 853 678 sesiones o flujos según la definición especificada en 2.4.1.

Tabla 2-4 Estadísticas en una traza de veinte minutos

Estadisticas\Protocolo	ICMP	TCP	UDP	GRE	ESP
% paquetes	0.27	80.3	19.2	-	-
% volumen de tráfico (bytes)	0.04	90.5	9.3	0.13	-
% flujos	2.4	52.1	45.5	-	-

Fuente: Elaboración propia

El segundo análisis, mejorando la performance de la herramienta en [24] se logra procesar 14 trazas, desde 20100106-030946-0.dsl hasta 20100106-093000-0.dsl, que en conjunto representan 6 horas y media de duración. En total se procesan 484 700 000 paquetes, de los cuales se encuentran 14 509 555 sesiones o flujos según la definición en 2.4.1.

Tabla 2-5 Estadísticas en traza de seis horas

Estadisticas\Protocolo	ICMP	TCP	UDP	GRE	ESP
% paquetes	0.31	77.5	21.9	0.2	-
% volumen de tráfico (bytes)	0.05	89.4	10.3	0.17	-
% flujos	2.5	60.4	36.9	-	-

Fuente: Elaboración propia

Se puede apreciar que en las trazas analizadas el tráfico más frecuente es debido a las sesiones TCP y UDP. Por esta razón, el presente trabajo se concentra en el análisis de estos dos protocolos para la elaboración de políticas de balanceo de carga.

2.4.4 Dinámica temporal de las sesiones – Max Inter Arrival Gap Time

Los flujos pertenecientes al protocolo TCP (orientado a conexión) son los más abundantes en una red empresarial según los análisis en la sección 2.4.2 debido a que los usuarios de la red utilizan aplicaciones HTTP, HTTPS, FTP, SMTP, POP3, SSH, IMAP con mayor frecuencia. En efecto, un *firewall* procesará flujos TCP proporcional a la cantidad de flujos TCP existente en la red.

Llamaremos *"gaps"* a los máximos tiempos entre paquetes en una sesión. En el análisis de las trazas en 2.4.2 se encontraron dos tipos de sesiones TCP voluminosas con determinadas características.

En primer lugar, aquellas que tienen un solo *gap*, véase la siguiente figura.

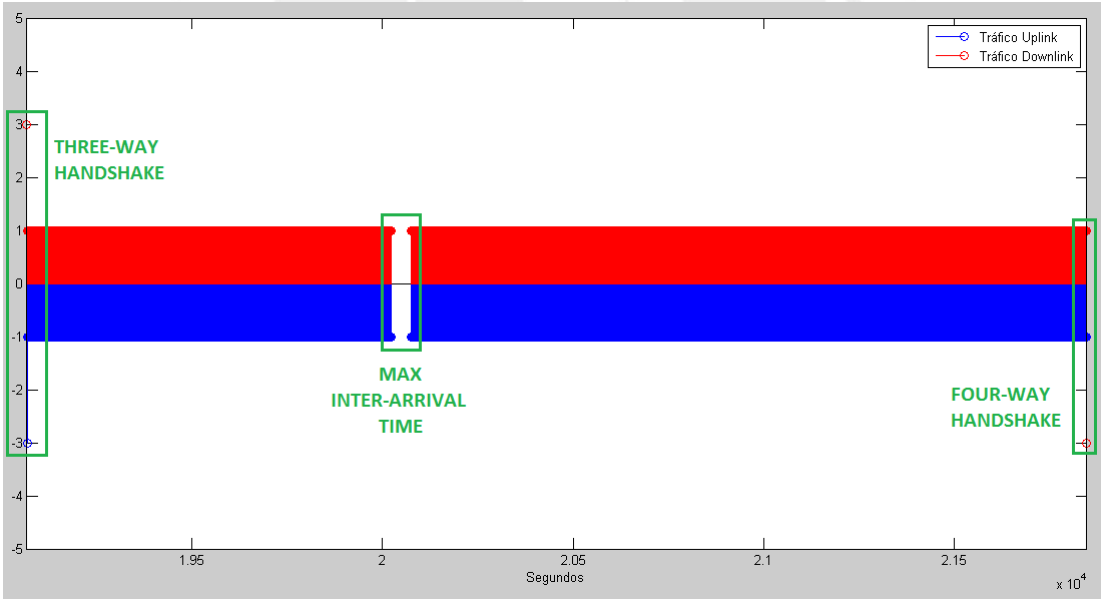

Figura 2-16 Sesión TCP voluminosa del primer tipo

Fuente: Elaboración propia

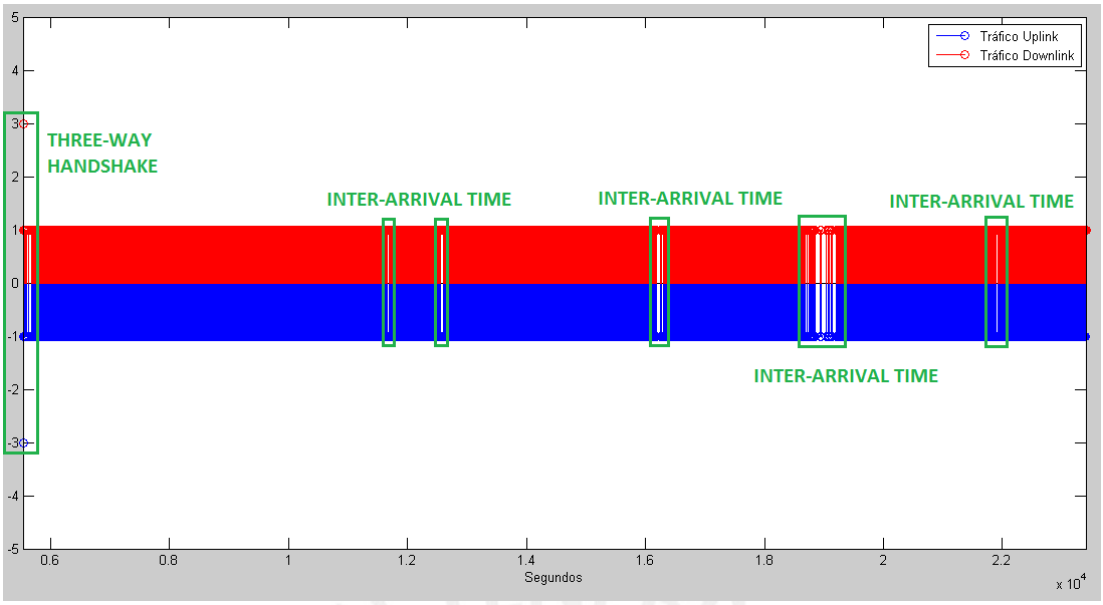

Figura 2-17 Sesión TCP voluminosa del segundo tipo

Fuente: Elaboración propia

El flujo que se representa en la Figura 2-16 pertenece a la sesión conformada por los siguientes atributos: IP_src = 70.191.28.83, IP_dst = 72.81.234.136, protocolo = TCP, Puerto_src = 54543, Puerto_dst = 80. Donde, el máximo tiempo entre paquetes (*Max Inter-Arrival Time*) es 52 segundos y tiene un tiempo de duración de 46 minutos.

En segundo lugar, aquellas que tienen un conjunto de *gaps* que varían de forma gradual véase la figura.

El flujo que se representa en la Figura 2-17 pertenece a la sesión conformada por los siguientes atributos: IP_src = 106.157.70.246, IP_dst = 172.141.81.133, protocolo = TCP, Puerto_src = 25953, Puerto_dst = 65426. Donde, el flujo tiene un tiempo de duración de 5 horas. Sin embargo, Sus máximos tiempos entre paquetes (*Max Inter-Arrival Time*) varían descendentemente gradual desde los 79,6 segundos hasta los 4 segundos.

Los *firewall* de segunda, tercera y cuarta generación mantienen la conexión de una sesión consumiendo parte de su memoria ya que guardan el estado del flujo hasta su expiración. Además, los flujos voluminosos son los que perduran más en el tiempo. Sin embargo, la presencia de gaps en una sesión, por ejemplo el flujo en la Figura 2-16, inutiliza el recurso de memoria en el *firewall* debido a que durante 52 segundos no se procesa algún paquete por el *firewall* pero se consume un recuso de memoria. Este efecto es más crítico cuando el tiempo en desuso de una sesión supera las horas. Para evitar estos problemas los *firewall* comerciales olvidan la sesión por defecto si el tiempo entre paquetes supera los 3600 segundos [10].

Una definición más detallada de una sesión en función a los máximos tiempo entre paquetes - *gaps* es necesaria en diseños donde la presencia de sesiones activas es crítica.

Tabla 2-6 Los flujos TCP voluminosos más críticos

Flujos TCP voluminosos (descendente) / Inter-ArrivalTime (segundos)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Gap - 1	131.77	85.44	35.76	33.97	33.61	25.70	24.44	24.15	14.43	12.56	9.62	9.41	8.94	7.64	7.26	5.64	4.47
Gap - 2	10.56	16.20	14.45	23.20	12.95	12.87	8.10	14.05	7.45	6.33	4.81	5.66	5.18		7.08	5.57	
Gap - 3	8.65	12.72	14.15	5.98	9.88	7.92		8.65	5.64						5.53	4.84	
Gap - 4	5.29	6.15	7.82		6.64	6.41		7.02							4.42	4.18	
Gap - 5			5.93					4.37									
Gap - 6			5.11														
Gap - 7			5.01														
Gap - 8			4.88														
Gap - 9			4.76														

Fuente: Elaboración propia

Por un lado, si definimos una sesión en función a flujos según la Figura 2-16 se inutiliza la capacidad de procesamiento del *firewall*. Por otro lado, si definimos una sesión en función a los *gaps* de un flujo como en la Figura 2-17 se toma el riesgo de crear aún más sesiones. Se analiza el caso más crítico, por lo cual se ordena los primeros mil flujos más voluminosos que no pertenecen a los tipos mencionados y cuyos *gaps* son mayores a 4 segundos. La lista se describe en la siguiente tabla.

La elección del máximo tiempo entre paquetes en una sesión TCP está limitado por un *tradeoff*. Si se elige un tiempo muy alto se corre el riesgo que estas sesiones sean olvidadas por el sistema operativo del *firewall*. Por otro lado, si se escoge un tiempo muy pequeño, se incrementará el procesamiento en el *firewall* debido a que manejará más conexiones. La Tabla 2-6 muestra los resultados para aquellas sesiones con *gaps* mayores a 4 segundos. Aunque algunas sesiones TCP presentan *gaps* de varios minutos, los paquetes después del *gap* son tráfico TCP de control para cerrar la sesión. Es decir, no hay tráfico de datos útil después del *gap*, por lo tanto, no habrá ningún daño al migrar esas sesiones por el Balanceador de Carga. Los casos mencionados están marcados en rojo en la Tabla 2-6. Por lo tanto, podemos ver que en la tabla que solo 6 de más de 14 millones de sesiones presentan un *gap* de más de 15 segundos (es decir, la posibilidad de que una sesión tenga un *gap* de más de 15 segundos es menor que 10^{-6}). Así, 15 segundos será el período para la recopilación de sesiones activas en este trabajo de tesis, ya que este valor induce una probabilidad en el orden de 10^{-9} de interrumpir una sesión, muy por debajo de la probabilidad de bloqueo debido a la congestión u otros problemas de una red empresarial.

No se realiza el análisis para el protocolo UDP ya que las sesiones que le pertenecen no manejan estados.

2.5 Estimadores y predictores

Debido a la presencia del *burst* de tráfico en las redes empresariales (tráfico dirigido a los *firewalls*) es necesario promediar el tráfico para evitar sus fluctuaciones al momento de decidir si la carga en un *firewall* excedió su umbral. Además, la predicción de un incremento del tráfico debido a las horas picos permitirá que el balanceador de carga sea proactivo para disminuir el tiempo de migración de carga desde un *firewall* a otro.

2.5.1 Forgetting Windows

El método *Forgetting Windows* promedia los valores de los elementos de una ventana de longitud L.

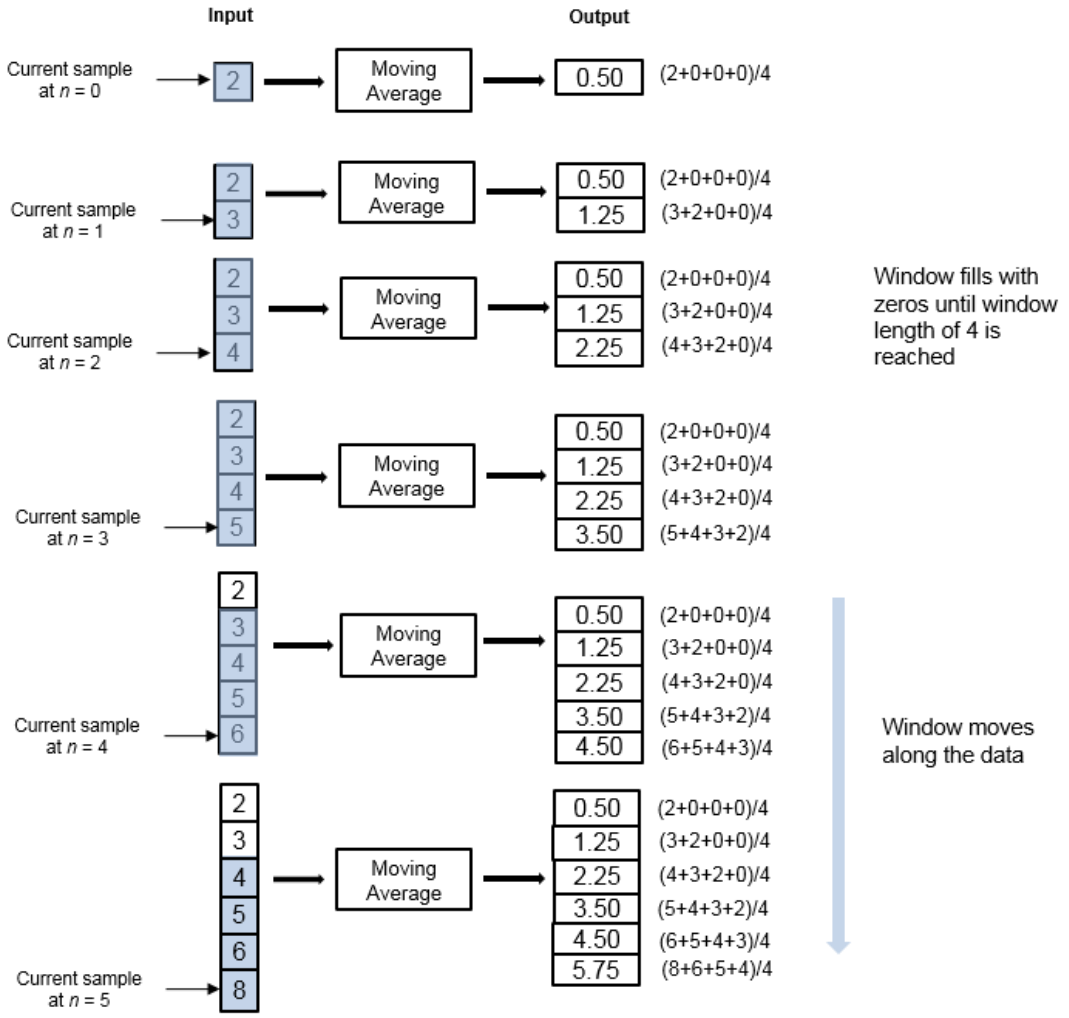

Figura 2-18 Método de ventana deslizante

Fuente: [26]

Sea una ventana de 4 elementos, ver Figura 2-18, cuya entrada recibe una serie de números enteros consecutivos. En el primer instante se promedia todos los valores de la ventana (en caso no haya un elemento se incluirá cero). En la tercera iteración se promedia todos los valores (3, 4, 5, 6) donde la salida es 3.5 (se conserva los valores promedios anteriores). Sucesivamente en cada iteración se va actualizando el valor promedio de la ventana (datos de entrada) en la salida. En la quinta iteración, se olvida que la ventana tuvo como datos de entrada (2 y 3) ya que la actualización del promedio se realiza a los últimos cuatro elementos de la ventana. Este es el método de ventana deslizante.

En el método anterior todos los datos en la ventana tenían el mismo peso al promediarse. A continuación, se presenta el método de pesos exponenciales.

$$y[n] = \rho * y[n-1] + (1-\rho) * x[n]$$

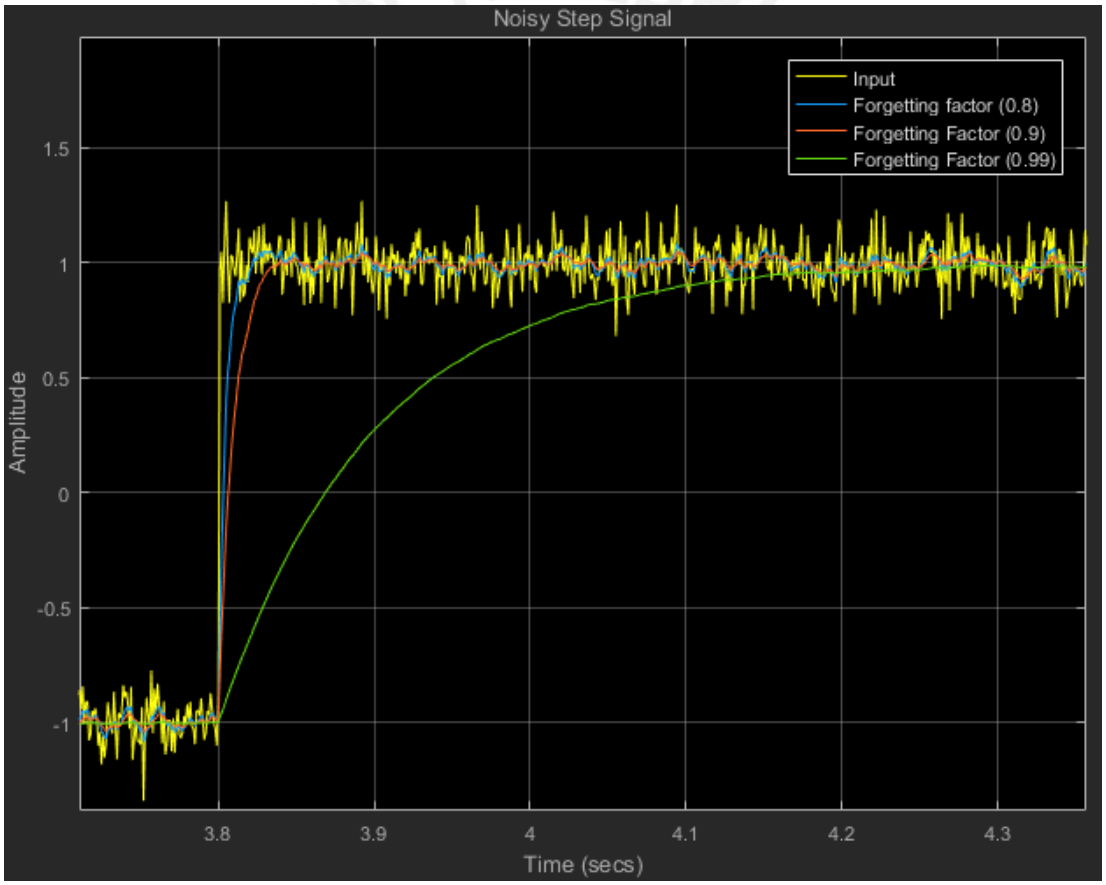

Figura 2-19 Factor de olvido

Fuente: [26]

En la ecuación anterior los valores en la entrada están representados por x[n] y los valores promediados por y[n]. El factor de olvido (ρ) varía entre cero y uno. Este valor indica la prioridad o peso con que se promediara los valores actuales en la ventana x[n]. En la Figura

2-19 se observa que en el tiempo 3.8 segundos hay una variación notoria. Se analizan tres valores del factor de olvido (0.8, 0.9 y 0.99).

El factor de olvido influye en la convergencia del promedio de la ventana al valor actual en x[n]. De la Figura 2-19, mientras el factor de olvido se acerca a cero (0.8) la respuesta (y[n]) se acerca más rápido al valor real de los datos de entrada. Si el factor de olvido se acerca a 1 (0.99) la respuesta es suavizada lo cual tarda más tiempo en alcanzar el valor promedio de la ventana.

2.5.2 Predictor de Kalmann

El filtro de *Kalmann* es un algoritmo recursivo, no requiere almacenar los datos previos, óptimo de procesamiento de datos. Procesa todas las mediciones en tiempo discreto sin importar la precisión para estimar el valor de las variables de interés.

En efecto, utilizando el filtro de *Kalmann* se podrá predecir (instante de tiempo) el incremento del tráfico debido a las horas pico.

3. DISEÑO DEL BALANCEADOR DE CARGA

En la sección anterior se presentaron los conceptos para el diseño del balanceador de carga. Se mencionó los balanceadores de carga comerciales (funcionamiento de los ADC) los cuales realizan el balanceo de carga a la granularidad del orden de una sesión. Estos resuelven la problemática del balanceo de carga a un banco de *firewall* N:1. Sin embargo, los precios de los ADC (20, 370 dólares cada uno) elevan el costo de capital en el modelo N:1. El uso de *switchs SDN/OpenFlow* (3, 000 dólares cada uno) como balanceadores de carga reduce significativamente (143, 567 dólares) el CaPex del banco de *firewalls* frente al modelo activo - respaldo.

A continuación se presentara la arquitectura del balanceador de carga basado en un *switch SDN/OpenFlow* especificando sus elementos, roles y definiendo algunos conceptos para la elaboración del algoritmo de balanceo de carga. Luego se describirán los requerimientos para su diseño según las limitantes encontradas en el capítulo anterior. Por último, se detalla el diseño del balanceador de carga a alto y bajo nivel (nivel de *software*).

3.1 Arquitectura del balanceador de carga

3.1.1 Elementos y roles

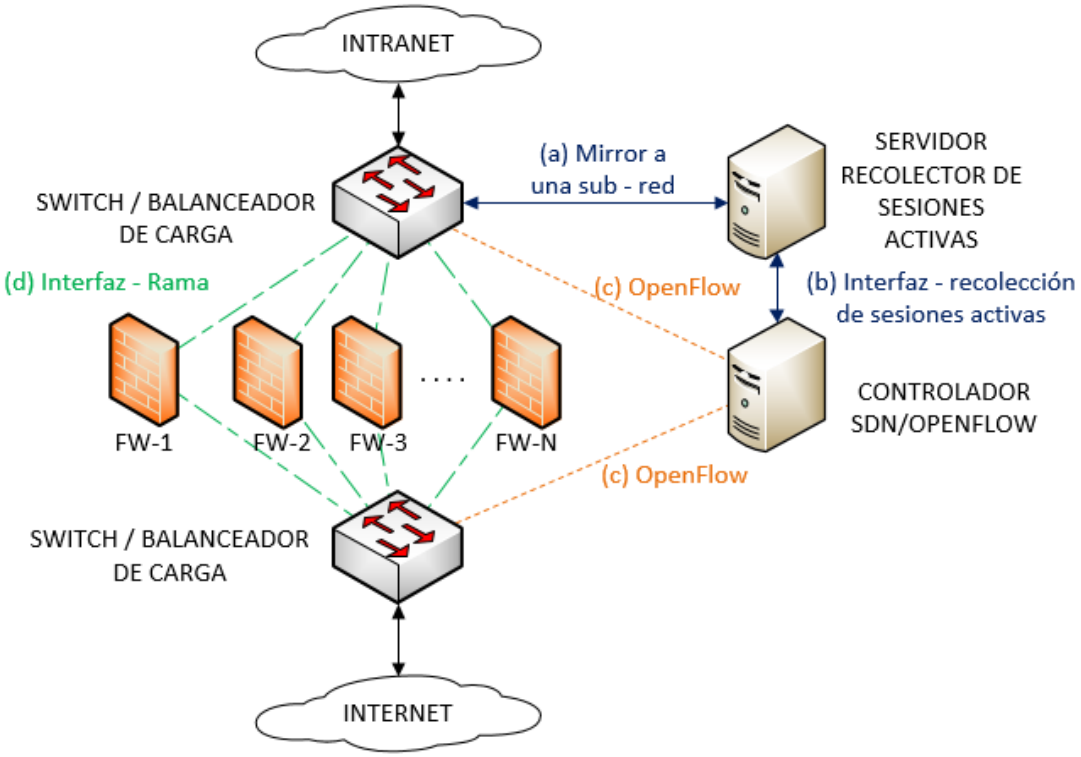

Figura 3-1 Diseño del balanceador de carga

Fuente: Elaboración propia

- *Switch SDN/OpenFlow*.- El *switch SDN/OpenFlow* cumple el rol de balanceador de carga. Este recibe reglas enviadas por el Controlador a través de la interfaz (c) para la creación, actualización o eliminación de *Flow Entries*. Estas reglas serán responsables del balanceo de carga en el banco de *Firewalls* N:1.
- Controlador *SDN/OpenFlow*.- El controlador *SDN/OpenFlow* cumple el rol de monitorear el trafico entrante y saliente por los *switch SDN/OpenFlow* que viajan por la interfaz (d). Está en constante comunicación con el servidor recolector de sesiones activas a través de la interfaz (b) para evaluar las sesiones activas que maneja un *firewall* en particular. Ejecuta una aplicación donde decide que reglas enviar al *switch SDN/OpenFlow* para realizar las reglas de balanceo de carga.
- Servidor recolector de sesiones activas.- El servidor recolector de sesiones activas cumple el rol de monitorear las sesiones activas presentes en un determinado *firewall* a través de la recolección de estas por la interfaz (a). Comunica al controlador *SDN/OpenFlow* las sesiones activas encontradas.

- *Firewalls*.- Estos equipos cumplen el rol de proteger a la red interna (intranet) frente a cyberataques inspeccionando los paquetes IP TCP/UDP al nivel de una sesión (*firewalls* de segunda generación).

3.1.2 Definición de Rama

Una Rama está conformado por los siguientes elementos: *firewall* y las interfaces de comunicación entre el *firewall* y los *switch SND/OpenFlow* que operan como balanceadores de carga.

El sistema tiene tantas Ramas como cantidad de elementos que conforman el banco de *Firewalls* N:1.

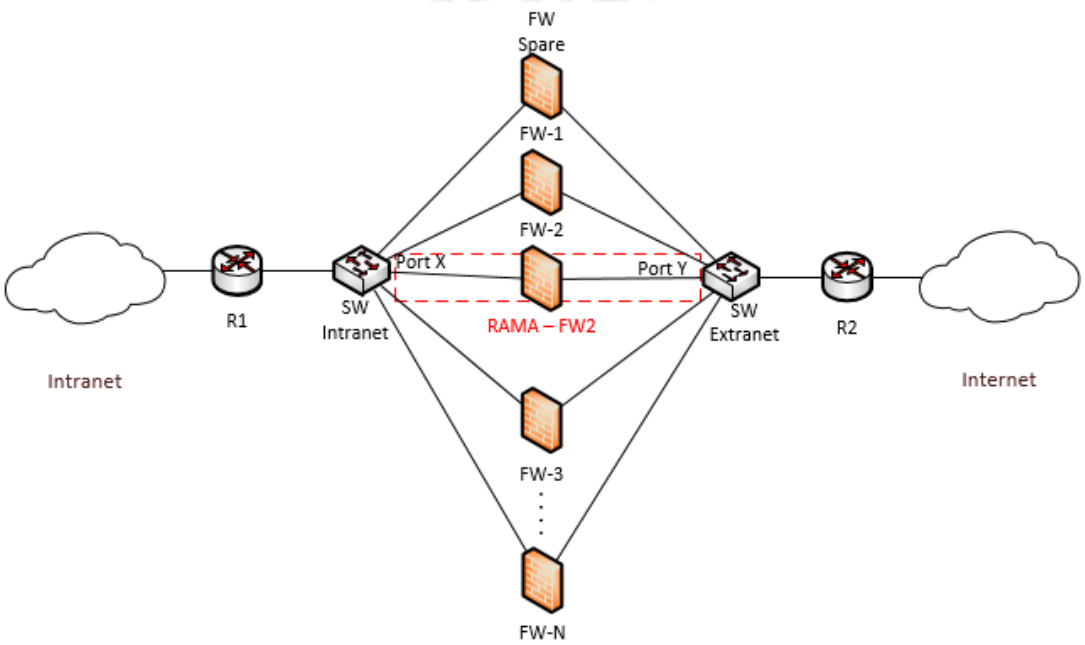

Figura 3-2 Ramas en el banco de *firewalls* N:1

Fuente: Elaboración propia

En la Figura 3-2 se representa gráficamente el concepto de Rama. El puerto (Port X) del *switch* (SW Intranet) próxima a la intranet recibe tráfico entrante - RX (paquetes que provienen desde internet hacia la intranet) y saliente - TX (paquetes que provienen desde la intranet hacia internet). Lo mismo sucede en el puerto (Port Y) del *switch* (SW Extranet) próxima a internet.

La máxima capacidad de tráfico (cantidad de paquetes por segundo o volumen en bps) que puede soportar una Rama depende del tráfico saliente-TX en el puerto *Port X* (paquetes que provienen de la intranet) y el saliente – TX del puerto *Port Y* (paquetes provenientes de

internet). El máximo valor de estas dos cantidades llamaremos ***valor representativo*** de esa Rama (Rama - FW2).

El valor representativo de la Rama está limitado por el máximo rendimiento del *firewall* que lo conforma. Por ejemplo, un *firewall* (modelo FG/FWF-50E de la firma Fortinet) tiene un rendimiento de 2.5 Gbps - 610 Kpps (512 *byte* UDP). Estas cantidades nos servirán para definir el valor umbral que determinará el estado de una Rama.

Otra característica de una Rama es que por esta se transporta el tráfico IP de un conjunto de subredes predeterminadas en la red interna.

3.1.3 Definición de estados de una Rama

Una Rama pertenece a un conjunto de estados que se definen a continuación:

- Una Rama está en el estado NORMAL cuando el tráfico que soporta (valor representativo) no supera el máximo rendimiento del *firewall* que la conforma (umbral).
- Una Rama está en el estado CANDIDATA si el tráfico que soporta excede el máximo rendimiento del *firewall* que la conforma. En este estado se selecciona una subred para migrar su carga hacia un *firewall* disponible (simultáneamente se recolecta las sesiones activas correspondiente a esa subred). Así se descongestiona el tráfico IP que procesa el *firewall* perteneciente a la Rama.

Una Rama está en estado BLOQUEADA cuando no puede ser usado para realizar migraciones de carga a otro *firewall*.

3.1.4 Proceso Hand Off por sobrecarga de tráfico

El tráfico de internet fluctúa por la presencia de tráfico *burst* desestabilizando el rendimiento de un *firewall*. Los *firewalls* del modelo de protección N:1 superan su capacidad de procesamiento en las horas pico de tráfico si la distribución de carga por parte del balanceador no es homogénea. Para evitar este fenómeno y utilizar todos los recursos disponibles del banco de *Firewall* N:1 se ejecuta el proceso *Hand Off*.

El proceso *Hand Off* involucra dos actores: un *firewall* sobrecargado y otro descongestionado.

Según el diseño del balanceador de carga de esta tesis, un *firewall* procesa el trafico IP correspondiente a un conjunto de Subredes (de la red empresarial) predeterminadas. Por ejemplo, de la Figura 3-3; por un lado, el *Firewall* 1 procesa solo el trafico proveniente de las subredes X, Y y Z. Sin embargo, el *Firewall* 1 comenzará a perder paquetes debido a que está sobrecargado (superó su máxima capacidad – umbral). Por otro lado, el *Firewall* 2

(equipo descongestionado) no opera a su máximo rendimiento pues solo procesa el tráfico de la subred W.

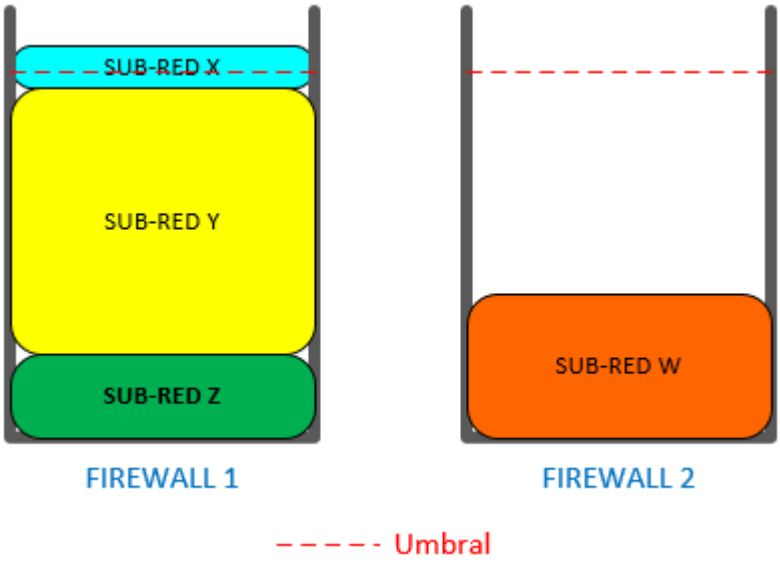

Figura 3-3 *Firewall* 1 sobrecargado

Fuente: Elaboración propia

El proceso *Hand Off* consiste en migrar la carga de la subred X (Subred de menor carga) perteneciente al *Firewall* 1 hacia el *Firewall* 2 (disponible).

Una vez realizada la migración, el *Firewall* 1 se descongestionará, ver Figura 3-4. Como los *firewall* a partir de la segunda generación son equipos sensibles a los estados de las conexiones abiertas (2.2.4) el proceso *Hand Off* incluye la permanencia de las sesiones activas ya establecidas por la subred X en el *Firewall* 1. En efecto, la migración de carga hacia el *Firewall* 2 solo se realiza a las nuevas sesiones provenientes de la subred X.

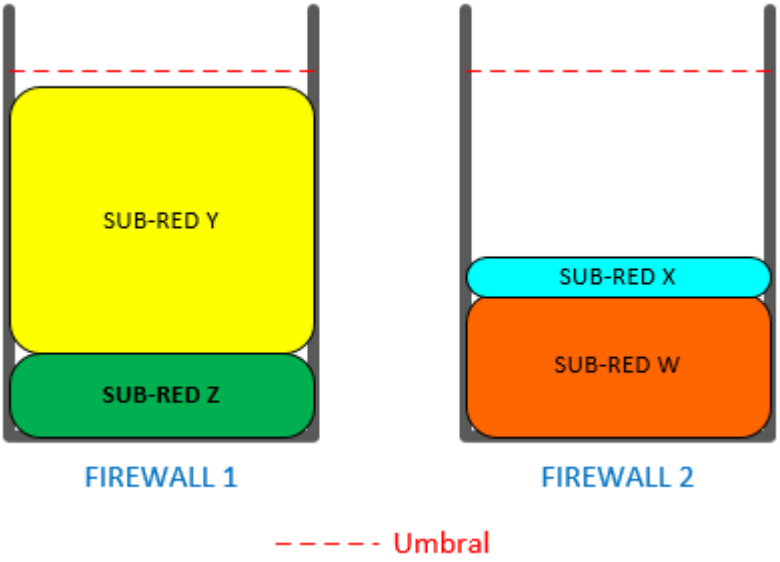

Figura 3-4 Estado de los *firewalls* después del proceso *Hand Off*

Fuente: Elaboración propia

3.2 Requerimientos para el balanceo de carga

El diseño del balanceador de carga depende de la naturaleza de los *firewalls* de segunda generación (equipos sensibles al estado de las sesiones), las limitantes físicas en la memoria del *switch SDN/OpenFlow* que se adaptará para realizar la función de balanceo de carga, evitar la pérdida de paquetes cuando se migra la carga de un *firewall* a otro, escalabilidad y alta disponibilidad ante la inserción de nuevos *firewall* al banco N:1 en el futuro. Por estos detalles, se describe los requerimientos que debe considerar un balanceador de carga basado en un *switch SDN/OpenFlow* para un banco de *Firewall* N:1.

3.2.1 Persistencia de las sesiones activas

El balanceador de carga debe garantizar la permanencia de las sesiones para no interrumpir la operatividad de los *firewalls* de segunda generación cuando se presenta migraciones de carga por sobrecarga en las horas pico de tráfico. Las sesiones que ya se iniciaron en un *firewall* no deben ser migradas a otro. La migración solo debe darse con aquellas sesiones nuevas. El balanceador de carga debe salvaguardar la persistencia de las sesiones (TCP/UDP) en los *firewalls* hasta la finalización de estas.

3.2.2 Entradas en la memoria TCAM de los switch SDN/OpenFlow

El diseño no debe exceder la cantidad máxima de entradas TCAM en los *switch SDN/OpenFlow* (8K a lo máximo). La red empresarial (PUCP) maneja un promedio de 80K sesiones activas en un momento dado distribuidas en 300 subredes (1 subred por pabellón

aproximadamente). Las reglas (1 *flow entry* por una subred) no deben exceder el máximo soportado por el *switch SDN/OpenFlow*. El uso de la memoria TCAM es crítico cuando un *firewall* es sobrecargado ya que se insertan nuevas entradas (*flow entries* a la granularidad del orden de una sesión) en los *switch SDN/OpenFlow* para garantizar la persistencia de las sesiones.

3.2.3 Tiempos de descongestión

Durante el tiempo que el *firewall* congestionado demora en descongestionarse se debe minimizar la pérdida de paquetes en el *firewall* congestionado.

3.2.4 Escalabilidad y alta disponibilidad

El balanceador de carga debe ser escalable a la adquisición de nuevos elementos al banco de *firewalls* N:1 ya que en el primer año el balanceador funcionará con dos *firewalls* y uno en modo de respaldo, el tercer y quinto año se agregará nuevos *firewalls* al banco N:1 (ver sección 1.1.2)

El diseño debe garantizar la operatividad de los *firewalls* debido a sobrecargas en los mismos en las horas pico de tráfico.

3.3 High level design del balanceador de carga

A continuación se presentará el diseño a alto nivel (algoritmo de balanceo de carga) para desarrollar la aplicación balanceo de carga en *software* (sección 3.4).

3.3.1 Máquina de estados de una Rama

Por defecto una Rama está en estado NORMAL, cuando se detecta o predice una sobrecarga en el *firewall* se procede a un cambio de estado, la Rama es CANDIDATA a una posible migración. Durante un tiempo, llamémosle periodo de recolección de sesiones (*Delta Y*), se analiza si la Rama en estado CANDIDATA debe ejecutar la migración de carga. Si durante ese periodo, la predicción de la carga en el *firewall* es un valor menor al umbral se procede al cambio de estado NORMAL. Si se supera el periodo (*Delta Y*), se procede a migrar la carga desde el *firewall* de la Rama hacia otro *firewall* disponible. Luego, la Rama entra en estado BLOQUEADA durante un tiempo llamémosle periodo de bloqueo donde se imposibilita a la Rama realizar migraciones de carga a otra Rama ya que la migración es un proceso costoso en el uso de *flow entries* (limitantes en la memoria TCAM). Cuando el periodo de bloqueo expira la Rama cambia al estado NORMAL y el proceso se repite.

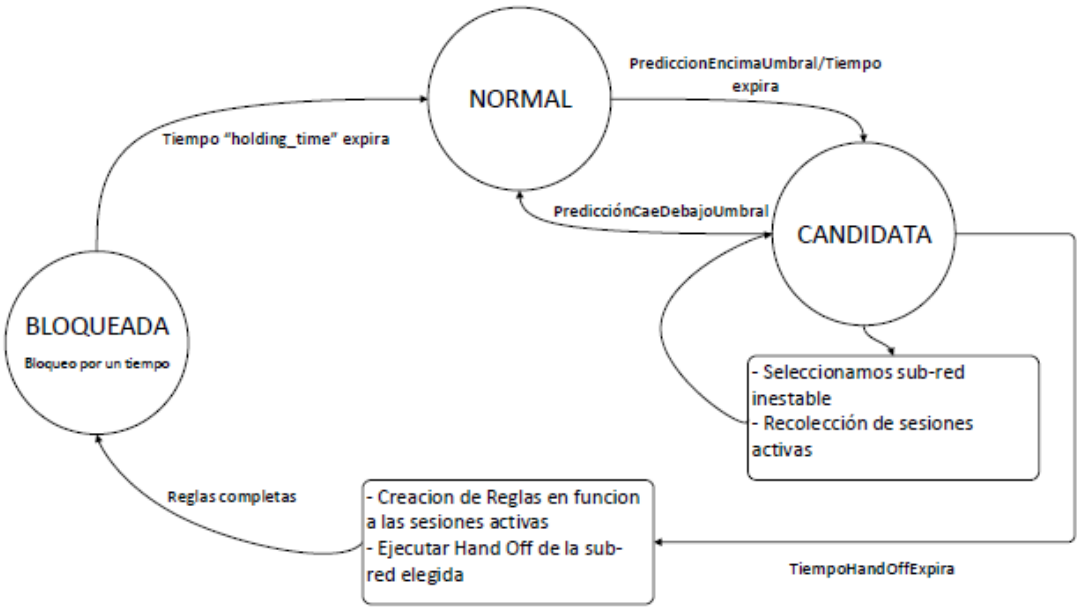

Figura 3-5 Máquina de estados de una Rama

Fuente: Elaboración propia

3.3.2 Definición de Triplets

Para realizar la migración de carga de un *firewall* sobrecargado a otro es necesario la determinación de tres elementos:

- La Rama origen, que incluye al *firewall* sobrecargado
- La subred perteneciente a la Rama origen cuya carga se migrará
- La Rama destino, que incluye el *firewall* que recibirá la carga de la subred migrada

Un *Triplet* está conformada por la Rama origen, subred *Hand Off*, Rama destino. Un *Triplet* nos servirá para ejecutar el proceso *Hand Off*.

3.3.3 Algoritmo y diagrama de bloques

El algoritmo está caracterizado por tres módulos principales: el módulo de detección de procesos *Hand Off* que se encarga de avisar periódicamente al sistema si hay una sobrecarga en algún elemento del banco de *Firewalls* N:1 a través de los *Triplets* encontrados, el módulo de ejecución de un proceso *Hand Off* que se encarga de procesar la información de los *Triplets* para ejecutar las migraciones de carga y el módulo de homogeneización de subredes cuyo objetivo es descomponer y unir subredes para que en el futuro se disminuya la cantidad de procesos *Hand Off*.

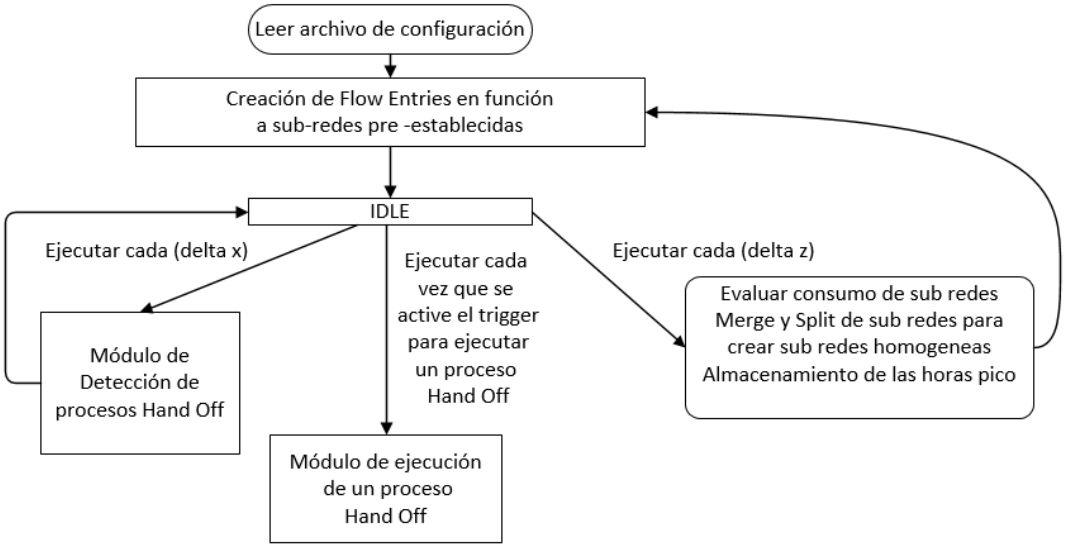

Figura 3-6 Algoritmo de balanceo de carga

Fuente: Elaboración propia

En primer lugar, se lee el archivo de configuración que contiene la siguiente información: prefijo (IP / mascara) de las 300 subredes pertenecientes a la red empresarial, DPID de los *switch SDN/OpenFlow*, Ramas disponibles en el sistema, IP del controlador SDN y del servidor de Recolección de Sesiones Activas y los umbrales definidos por la capacidad de los *firewalls* en bps. Con esta información se procede la creación de tantos *flow entries* como subredes en la red empresarial (cada *Flow Entry* corresponde a una subred) distribuidas en todas las Ramas del sistema vía *Round Robin*.

Periódicamente, cada (*Delta X*) segundos, se monitorea las posibles migraciones de carga en cada *firewalls* del sistema. En la Figura 3-7 se representa el diagrama de bloques del módulo detección de Procesos *Hand Off*. En primer lugar, se mide los nuevos valores de la carga en bps y pps de cada subred, asimismo se actualiza el valor de los predictores y estimadores en cada Rama. En segundo lugar, se ordena descendentemente las Ramas según su nivel de congestión. En tercer lugar, se elige o actualiza los *Triplets* (en función a las Ramas congestionadas) existentes evitando aquellas Ramas en estado BLOQUEADO. Si se encuentra alguna nueva Rama en estado CANDIDATA se activa el *trigger* para ejecutar el proceso *Hand Off* (donde se recolecta el registro de sesiones activas) en esa Rama. Si la predicción de la carga en una Rama ya CANDIDATA no supera el umbral se procede a detener el registro de sesiones activas.

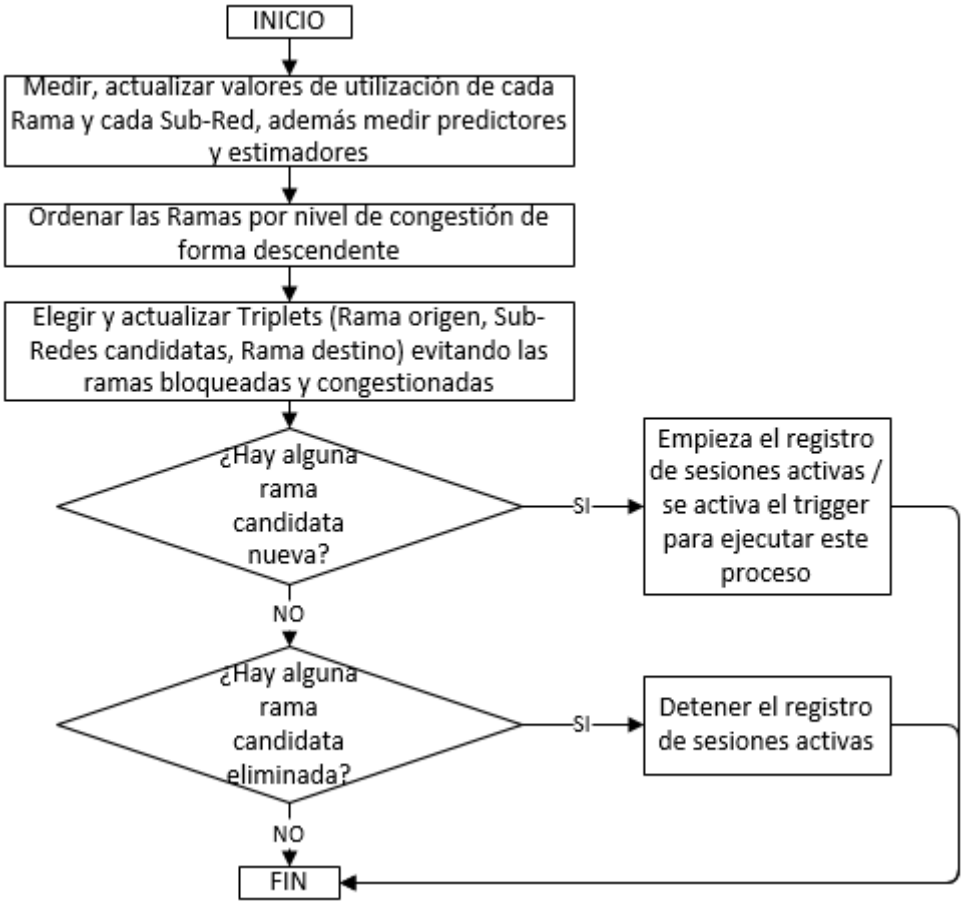

Figura 3-7 Módulo de detección de procesos *Hand Off*

Fuente: Elaboración propia

Una vez que se encontró una nueva Rama en estado CANDIDATA se activa el *trigger* para la inicialización del módulo de ejecución de un proceso *Hand Off*. En la Figura 3-8 se presenta en diagrama de bloques de este módulo. En primer lugar, por la Rama elegida se transportan paquetes pertenecientes a un conjunto de subredes, se selecciona aquella subred que transporta menor carga en bps. En segundo lugar, se comienza la recolección de sesiones activas pertenecientes a la subred escogida por un periodo de tiempo (*Delta Y*) segundos. En tercer lugar, si se supera el periodo de recolección de sesiones activas se ingresa tantos *Flow Entries* (cada *Flow Entry* con la información de una sesión) como sesiones encontradas, se migra la carga de la subred a una Rama disponible según su *Triplet*, luego se bloquea la Rama por un periodo de K segundos para que no se realice alguna migración debido a que ya se consumió recursos en la memoria TCAM. Posteriormente o si no se superó el periodo de recolección de sesiones activas, se detiene y elimina el registro de recolección.

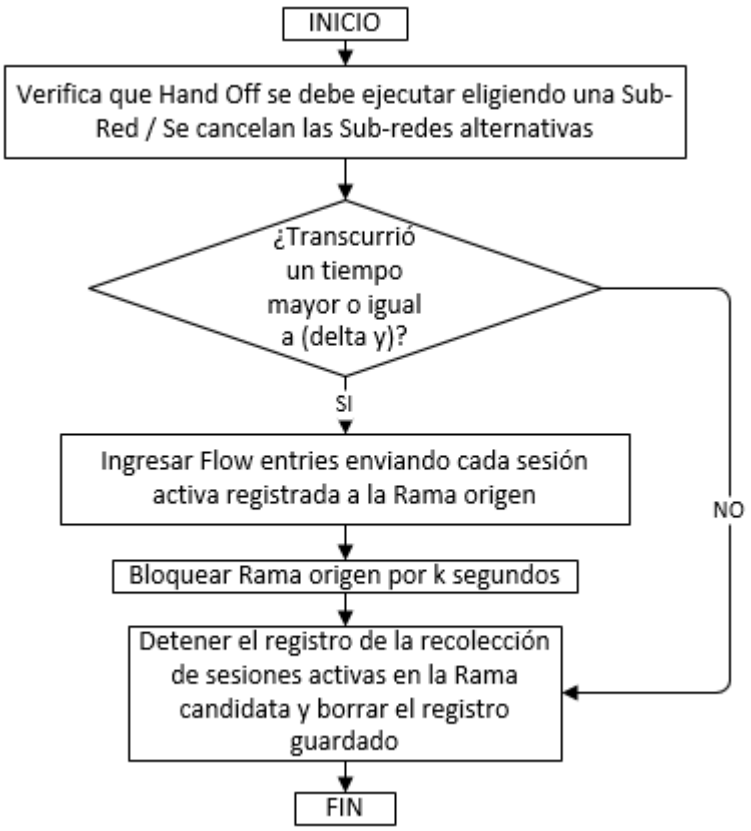

Figura 3-8 Módulo de ejecución de un proceso Hand Off

Fuente: Elaboración propia

Periódicamente un tiempo (*Delta Z*) se evalúa el consumo de las subredes en sus horas pico para redistribuir las subredes (*Merge o Split* de estas) en nuevas subredes homogéneas y disminuir en el futuro la cantidad de procesos *Hand Off*.

En la siguiente Figura 3-9 se expone en detalles el diagrama de bloques completo del algoritmo..

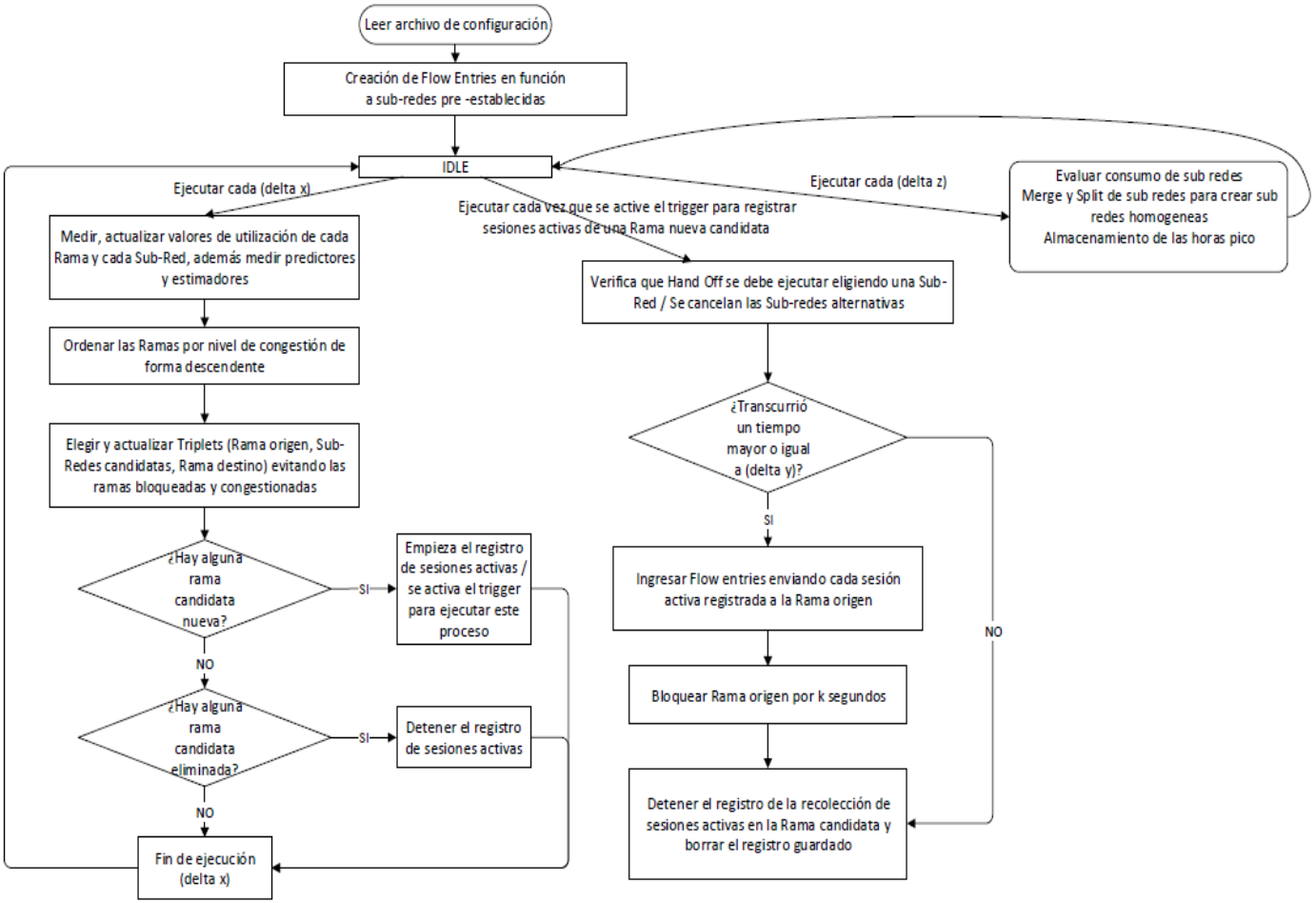

Figura 3-9 Diagrama de bloques del algoritmo balanceador de carga

Fuente: Elaboración propia

La implementación del algoritmo involucra la operación del módulo de balanceo de carga y el módulo de monitoreo de sesiones activas que se explican en las siguientes secciones.

3.3.4 Módulo balanceador de carga

El módulo de balanceo de carga está conformado por el *switch SDN/OpenFlow* cuya función es ejecutar las reglas definidas en su tabla de flujos (*Flow Table*) y el controlador SDN que elabora las reglas dinámicas para el balanceo de carga. Este monitorea el tráfico de las subredes en cada Rama y controla al *switch SDN/OpenFlow* a través de la interfaz (c), Figura 3-10.

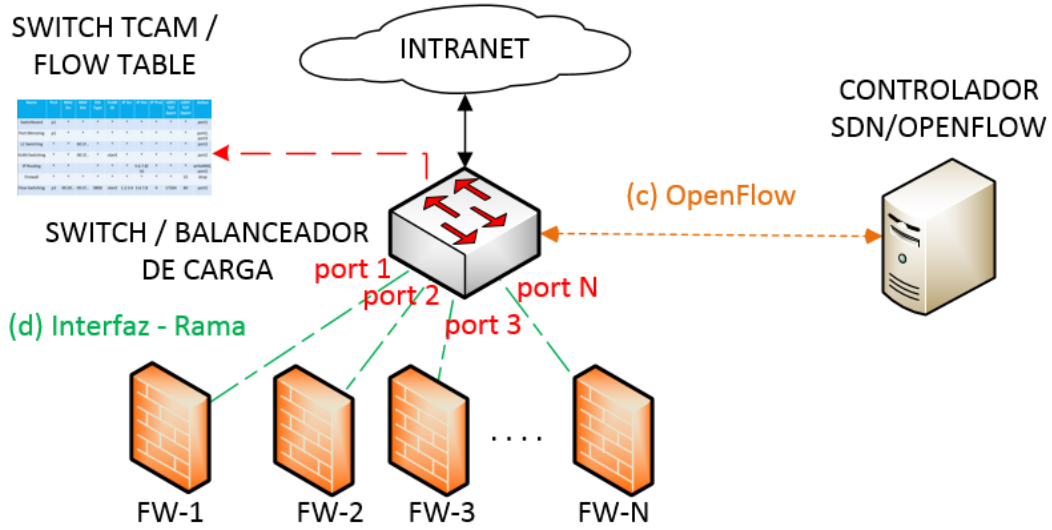

Figura 3-10 Arquitectura del módulo balanceador de carga

Fuente: Elaboración propia

Name	Switch DPID	Port	MAC Src	MAC Dst	Eth Type	IP Src	IP Dst	TCP/UDP Sport	TCP/UDP Dport	Action
Flow Sub - Red 1	DPID intranet		*	*	0x0800	IP Sub-Red 1/mask	*	*	*	port 1
Flow Sub - Red 2	DPID intranet		*	*	0x0800	IP Sub-Red 2/mask	*	*	*	port 2
Flow Sub - Red 3	DPID intranet		*	*	0x0800	IP Sub-Red 3/mask	*	*	*	port 3
...	...	...	...	...	...	...	...	...	...	...
Flow Sub - Red M	DPID intranet		*	*	0x0800	IP Sub-Red M/mask	*	*	*	port 1

Figura 3-11 *Flow table* del *switch SDN/OpenFlow*

Fuente: Elaboración propia

Según el algoritmo en la sección 3.3.3 previamente se crean *flow entries* en función a las subredes pre establecidas. Por ejemplo, basándonos en la Figura 3-10, supongamos que hay M subredes en la intranet. Por cada subred se ingresa un *flow entry* al *Switch* (Balanceador

de carga) especificando su prefijo, máscara y la acción (todos los paquetes de la subred 1 se redirigirán al *Firewall* 1, de la subred 2 al *Firewall* 2 y así sucesivamente).

En la Figura 3-11 se describe las reglas que se insertan en el *switch* cercano a la intranet (DPID intranet). Estas reglas definen un balanceo de carga al nivel de una subred.

3.3.4.1 Características del switch SDN/OpenFlow

- Procesador *Triumph2* que soporta hasta 8K entradas en su memoria TCAM
- Compatible con las versiones mayores o igual al protocolo *OpenFlow* 1.3 para que soporte mensajes OFPMP_PORT_STATS
- Versión *Open vSwitch* 2.5.0
- 48 puertos 1 *Gigabit Ethernet* y 4 puertos 10 *Gigabit Ethernet*

3.3.4.2 Sesiones por subred

El proceso *Hand Off* está limitado por el costo en el uso de entradas en la memoria TCAM de los *switch SDN/OpenFlow*. El módulo de homogenización de subredes (ver sección 3.3.2), que no se implementa en esta tesis, tiende agrupar las subredes para que todas tengan la misma cantidad de sesiones.

Asumiendo que en cada subred hay la misma cantidad de sesiones se definen las siguientes variables.

$$Instante\ de\ tiempo : \Delta t$$

$$Número\ de\ sesiones\ en\ un\ periodo\ \Delta t : M$$

$$Número\ de\ sesiones\ por\ subred : x$$

$$Número\ de\ subredes\ en\ M\ sesiones : \frac{M}{x}$$

$$Máxima\ cantidad\ de\ entradas\ en\ la\ memoria\ TCAM : \beta$$

$$Cantidad\ de\ entradas\ por\ un\ proceso\ Hand\ Off : y$$

Cuando se produce un proceso *Hand Off* la cantidad de entradas que ingresan a la memoria TCAM del *switch SDN/OpenFlow* está conformada por los *flow entries* de cada subred y el número de sesiones activas encontradas en la subred a migrar.

$$(i)\quad y = x + \frac{M}{x}$$

La cantidad de entradas que ingresan a la memoria TCAM del *switch* es mínima cuando se cumple lo siguiente.

$$(ii)\ x = \sqrt{M}$$

Sin embargo, el total de entradas en el *switch SDN/OpenFlow* está limitada por la capacidad de su memoria TCAM. Reemplazando (ii) en (i)

$$(iii)\ y_{min} = 2\sqrt{M} \leq \beta$$

$$(iv)\ M_{max} = (\frac{\beta}{2})^2$$

Para esta tesis, el número de sesiones encontradas en un instante dado en la red PUCP es aproximadamente 80 000 y la máxima cantidad de entradas en la memoria TCAM del *switch SDN/OpenFlow* a utilizar es 8 000.

$$(v)\ M = 80\,000$$

$$(vi)\ \beta = 8\,000$$

Reemplazando (vi) en (iv).

$$(vi)\ M_{max} = 16\,000\,000\ \frac{sesiones}{\Delta t}$$

En (vi) se verifica que la asignación de entradas según el algoritmo desarrollado en (3.3.3) para el caso de análisis de la red PUCP si soporta el mecanismo de *Hand Off* ya que la cantidad máxima de sesiones activas en un instante dado no supera al límite hallado en (vi).

Reemplazando (v) en (iii) se obtiene que la cantidad ideal de sesiones por subred debe ser 282 con 282 subredes predeterminadas.

Para criterios de esta tesis, la cantidad de subredes en la red PUCP es 300 (valor próximo al hallado idealmente).

3.3.4.3 Cantidad máxima de procesos Hand Off

Como en el proceso *Hand Off* se consumen recursos de entradas es necesario calcular la cantidad máxima de procesos *Hand Off* que el *switch SDN/OpenFlow* puede soportar en un instante dado.

Siguiendo la notación del análisis en la sección (3.3.4.2) se agrega la siguiente notación.

$$Cantidad\ de\ procesos\ Hand\ Off : \alpha$$

En el peor caso, α *firewalls* del banco N:1 necesitaran realizar el proceso *Hand Off* para garantizar la operatividad del sistema. Entonces la cantidad de entradas en la memoria TCAM del *switch SDN/OpenFlow* está representada por lo siguiente.

$$(vii) \quad y = \alpha * x + \frac{M}{x} \leq \beta$$

Reemplazando (ii) en (vii) para minimizar la expresión en (vii)

$$(viii) \quad y = \alpha * \sqrt{M} + \sqrt{M} \leq \beta$$

$$(viii) \quad y = (1 + \alpha) * \sqrt{M} \leq \beta$$

Donde la cantidad máxima de procesos *Hand Off* en un instante Δt es

$$(ix) \quad \alpha_{max} = \frac{\beta}{\sqrt{M}} - 1$$

Reemplazando los valores (v) y (vi) de (3.3.4.2) en (ix) se obtiene que la cantidad máxima de procesos *Hand Off* que soportará la memoria TCAM del *switch SDN/OpenFlow* es 27.

Como la cantidad de *firewalls* que se usan en el banco de *firewalls* N:1 para esta tesis (N=5) ver sección (1.1.2) es menor a la cantidad hallada en (ix) se garantiza el uso del proceso *Hand Off* en el diseño del balanceador de carga.

3.3.5 Módulo de monitoreo de sesiones activas

El fin de este módulo que está compuesto por el *switch SDN/OpenFlow*, un servidor dedicado a la recolección de sesiones activas y el controlador SDN es la identificación de las sesiones activas de una determinada subred para completar el proceso de migración o *Hand Off*.

Este módulo surge en necesidad de que el controlador SDN pueda manejar las sesiones de una subred y completar lo descrito en el algoritmo. Se agrega un servidor cuyo único fin es recolectar las sesiones activas dejando al controlador como un cliente que solicita el recurso de sesiones activas.

El servidor recolector de sesiones activas procesa en cada instante los paquetes proveniente de una subred a través de su interface (a). Depura la información y devuelve al controlador (si este le solicita) la información en un formato de texto (*Json, XML*, etc).

Para completar la búsqueda de sesiones activas surgen dos problemas: (i) duplicar los paquetes de las subredes para que uno viaje por el *firewall* y otro por la interfaz (a) para el respectivo análisis del servidor, (ii) escoger una estructura de datos eficiente para el almacenamiento de las sesiones activas.

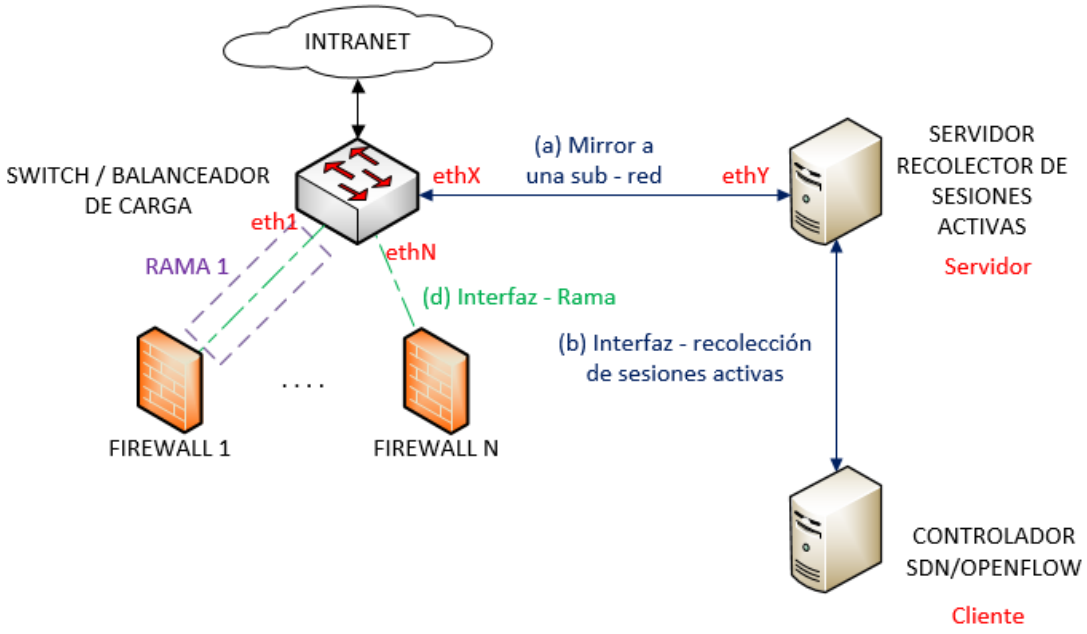

Figura 3-12 Arquitectura del módulo de monitoreo de sesiones activas

Fuente: Elaboración propia

3.3.5.1 Group entries para el mirror a subredes

Para resolver el primer problema, se crea N *group entries* en el *switch SDN/OpenFlow* en función a la cantidad de Ramas en el sistema para realizar un *multicast* de paquetes con destino a la Rama y al puerto de la interfaz (a) de la Figura 3-12.

Group ID	Type	Action Buckets	Counters
1	All	output=X,output=1	
2	All	output=X,output=2	
3	All	output=X,output=3	
. . .	. . .	. . .	. . .
N	All	output=X,output=N	

Figura 3-13 Tabla de grupos del *switch SDN/OpenFlow*

Fuente: Elaboración propia

Name	Switch DPID	Port	MAC Src	MAC Dst	Eth Type	IP Src	IP Dst	TCP/UDP Sport	TCP/UDP Dport	Priority	Action
Flow Sub - Red 1	DPID intranet		*	*	0x0800	IP Sub-Red 1/mask	*	*	*	10	port 1
Mirror Sub - Red 1	DPID intranet		*	*	0x0800	IP Sub-Red 1/mask	*	*	*	20	groupID = 1

Figura 3-14 Reglas para el *mirror* de una subred

Fuente: Elaboración propia

Por ejemplo, de la Figura 3-13, se crea un grupo con ID 1 (de la Rama 1) tipo *multicast* (*All*) cuya acción es enviar paquetes por los puertos 1 (de la Rama 1) y X (con destino al servidor recolector de sesiones activas).
Asimismo, una vez que se necesite recolectar las sesiones activas de una subred el controlador *SDN/OpenFlow* deberá enviar una regla (*Mirror* subred 1) con prioridad más alta (20) que la regla ya definida previamente (*Flow* subred 1) para realizar el *multicast* de paquetes.

En la Figura 3-14 se describe las reglas que el *switch SDN/OpenFlow* manejará para que el servidor (recolector de sesiones activas) comience a operar.

3.3.5.2 Estructura de datos para la recolección de sesiones activas

En una red empresarial (caso PUCP – 300 subredes) donde en horas pico el tráfico bordea los (5Gbps), el tráfico que procesa la subred mínima será 16.5 Mbps (asumiendo el peor caso donde todas las subredes transportan la misma carga). Con un IMIX (64 bytes UDP) se estima que el servidor procesará en promedio 32 220 paquetes por segundo.

La recolección de sesiones activas involucra el almacenamiento de estas en una estructura de datos. Sin embargo, la cantidad de búsquedas en la estructura será superior a la cantidad de inserciones en la misma. En efecto, se debe seleccionar una estructura de datos que sea eficiente en la búsqueda de sus elementos.

Tabla 3-1 Comparación de la operación buscar en las estructuras de datos

Operación : buscar x (elemento) en la s (estructura)				
Estructura	Lista	Diccionario		Arbol binario
Caso	Promedio	Promedio	Peor	Promedio
Tiempo de complejidad	O(n)	O(1)	O(n)	O(log(n))

Fuente: [27]

En la Tabla 3-1 se muestra una comparación de tres estructuras de datos conocidas en función al tiempo de complejidad de su operación búsqueda de elementos. Por un lado, la estructura lista, con un tiempo de complejidad lineal queda descartada por ineficiente. La decisión de utilizar una estructura de datos para nuestros propósitos se debate entre el diccionario y el árbol binario. Se escoge el diccionario por dos motivos: (i) porque en lenguajes como (*Python*) ya están implementados a diferencia de los arboles binarios y (ii) tiene un tiempo de complejidad constante (muy eficiente) a comparación del tiempo logarítmico del árbol binario. Para las limitantes de este trabajo, 32 200 búsquedas por segundo, el diccionario en *Python* se adapta a los requisitos. Sin embargo, si se pretende escalar este

trabajo donde involucre más búsquedas que la estimada se recomienda utilizar la estructura árbol binario pues los diccionarios pueden llegar a ser ineficiente (afectar el rendimiento del servidor recolector de sesiones activas) por las colisiones en su tabla hash.

3.3.6 Heurística de los temporizadores

En la sección 3.3.3 se especifica tres periodos cuyos valores, se determinan a continuación.

3.3.6.1 Periodo de actualizar valores de utilización – Delta X

El monitoreo del consumo del ancho de banda por parte de las subredes y en cada Rama debe realizarse periódicamente. Mientras el periodo (*Delta X*) de actualización de valores de utilización es más pequeño se logra obtener estos valores con más precisión. Sin embargo, la latencia que produce las consultas desde la aplicación en el controlador SDN al servidor de sesiones activas y el uso de la REST API determinan el valor mínimo del periodo *Delta X*.

Para esta tesis, se predetermina que el valor del periodo *Delta X* es dos segundos.

3.3.6.2 Periodo de recolección de sesiones – Delta Y

Según el análisis en la sección 2.4.4 se determinó que el máximo valor del tiempo entre paquetes de una sesión en una red empresarial es quince segundos. Se utiliza este resultado para definir el periodo de recolección de sesiones ya que este valor garantiza la persistencia de una sesión.

Para esta tesis, el valor del periodo *Delta Y* es quince segundos.

3.3.6.3 Periodo de aprendizaje de subredes homogéneas – Delta Z

Para objetivos de esta tesis no se implementa el módulo de homogenización de subredes que se actualiza un periodo *Delta Z*. Sin embargo, se propone que dicho periodo se realice cada veinticuatro horas.

3.4 Low level software design del balanceador de carga

En la sección anterior se describió el diseño a alto nivel, el algoritmo, módulo de balanceo de carga, módulo de monitoreo de sesiones activas y las heurísticas de los temporizadores para desarrollar el diseño del balanceador de carga en *software*. A continuación se elige el controlador *SDN/OpenFlow* a utilizar, luego se describe el diagrama de bloques de la solución al nivel de código, posteriormente los detalles técnicos del servidor recolector de sesiones activas y por último las dependencia y herramientas utilizadas para la implementación del balanceador.

3.4.1 Requerimientos del controlador SDN

La elección del controlador SDN para la implementación del balanceador de carga depende de los criterios analizados en [13]. Para los objetivos de esta tesis, los criterios más importantes son: lenguaje de programación, documentación de la REST API, curva de aprendizaje y performance del controlador.

3.4.1.1 Lenguaje de programación

La elección de un lenguaje de programación depende de la fácil adaptación en código del diseño a alto nivel. Se eligió *Python* porque disfrutan de una amplia comunidad de desarrollo y una diversidad de librerías (*Scapy*, para la generación y recepción de paquetes TCP/UDP, *pycapfile* para el análisis de paquetes y más) documentadas que permiten el *Rapid Prototyping* de la aplicación a implementar.

A esto se suma que es multiparadigma (orientado a objetos, funcional e imperativa) lo que proporciona una performance en la abstracción del algoritmo balanceador de carga (Rama, Subred, *Triplet*). Además, aplicaciones REST de controladores como *Floodlight* están basadas en este lenguaje.

3.4.1.2 Documentación de la REST API

Controladores como *NOX, POX, Trema, Ryu y ODL* tiene una documentación media o baja. Los controladores con una documentación más detallada son *Beacon* y *Floodlight*. Sin embargo, *Floodlight* posee una REST API más extensa que *Beacon* [13] (pag41)

3.4.1.3 Curva de aprendizaje

Controladores como *OpenDaylight* con una librería más completa dificultan el proceso de aprendizaje para la implementación de nuevos módulos (balanceador de carga) ya que consume horas de dedicación. Trabajos en la plataforma *Floodlight* (*PUCPLight*) [13] disminuyen la curva de aprendizaje por la documentación de su arquitectura.

3.4.1.4 Performance

La performance del controlador *SDN PUCPlight* [13] basada en *Floodlight* soporta hasta 56 420 nodos o host en tráfico uniforme y 100 004 en tráfico Pareto. Esto ubica a *PUCPLight* como un controlador con alta performance para una red de campus universitario que en promedio alberga 40 000 nodos.

3.4.1.5 Selección de la plataforma del controlador SDN

Por estos criterios se selecciona *Floodlight* como el controlador a utilizar en la implementación de esta tesis. Otra de las razones que llevaron a elegir *Floodlight* como plataforma base es que fue utilizada por el controlador *PUCPLight*, desarrollado por el grupo GIRA de la PUCP. La implementación del balanceador de carga como aplicación en el controlador *Floodlight* permitirá el fácil acoplo del módulo (*Firewall Load Balancing*) al proyecto PICASSO, en desarrollo por el GIRA.

3.4.2 Diagrama de bloques de la solución al nivel de código

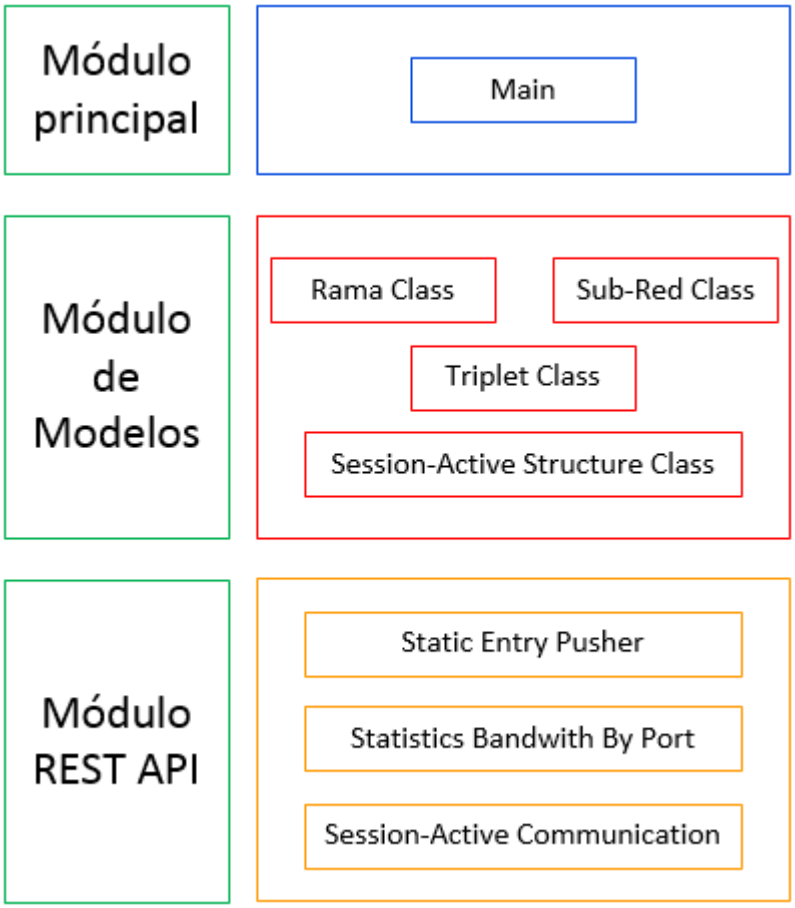

Figura 3-15 Arquitectura a nivel de código del módulo de balanceo de carga

Fuente: Elaboración propia

Con una visión más completa del algoritmo de balanceo de carga, los requerimientos y el diseño a alto nivel se elabora la solución a nivel de código (*software*). Esta comprende el desarrollo de dos aplicaciones: una para el balanceo de carga implementada en el controlador SDN y la otra para el monitoreo de las sesiones activas en el servidor recolector. Se opta por

la arquitectura cliente (controlador SDN) – servidor (servidor recolector de sesiones activas) en el desarrollo de la segunda aplicación respectiva.

Por un lado, el diseño de la aplicación para realizar el balanceo de carga comprende tres módulos: principal, modelos y REST API.

En primer lugar, el módulo REST API es la interfaz de comunicación entre la aplicación en el controlador, los *switch SDN/OpenFlow* y el servidor recolector de sesiones activas.

- *Static Entry Pusher*.- Esta clase está basado en el módulo de *Floodlight Static Entry Pusher* cuya función es crear, actualizar y eliminar *flow entries* de manera estática a través de la REST API *Floodlight*. Para propósitos de esta tesis, la inserción de *flow entries* será proactiva.
  En la Figura 3-16 se presenta la consulta al controlador SDN por el puerto 8080 por la REST API del módulo *Static Entry Pusher* para la inserción de un *flow entry* a un *switch SDN/OpenFlow*. La consulta incluye la información detallada del *flow entry*.

```
> curl -X POST -d 'flow_details' http://<controller_ip>:8080/wm/staticentrypusher/json

flow_details = {
            "switch": "00:00:00:00:00:00:00:01",
            "name": "flow-mod-1",
            "cookie": "0",
            "priority": "32768",
            "in_port": "1",
            "active": "true",
            "actions": "output=2"
            }
```

Figura 3-16 Consulta vía la REST API (*Static Entry Pusher*)

Fuente: Elaboración propia

- *Statistics Bandwith By Port*.- Esta clase está basada en el módulo *Statistics* de *Floodlight*. La recolección de la tasa de transmisión (TX) o recepción (RX) en bps en algún puerto del *switch* se determina por el pedido del controlador a través del mensaje OFMP_PORT_STATS (disponible desde la versión OF 1.3) [28] [pág. 85] *OpenFlow* de los *bytes* recibidos y transmitidos. Simultáneamente, por cada pedido de esta métrica se ejecuta un hilo que calcula el tiempo transcurrido desde el envío del mensaje hasta la respuesta del *switch SDN/OpenFlow* al controlador *Floodlight*. La diferencia entre los *bytes* consumidos y el tiempo transcurrido nos da una aproximación del ancho de banda en un puerto del *switch*.

En la Figura 3-17 se muestra la consulta al controlador SDN por el puerto 8080 para obtener la tasa de transmisión o recepción del puerto (portId) de un *switch* específico (switchId).

- *Session-Active Communication*.- Esta clase ejecuta las consultas para determinar los estados en el servidor recolector de sesiones activas. Su propósito es obtener la información de las sesiones activas de una subred en una estructura de diccionario según (3.3.5.2).

```
> curl http://<controller_ip>:8080/wm/statistics/bandwidth/<switchId>/<portId>/json
```

Figura 3-17 Consulta para obtener la tasa de transmisión en un puerto del *switch SDN/OpenFlow*

Fuente: Elaboración propia

En segundo lugar, el módulo de modelos se encarga de almacenar la data recolectada por el módulo REST API en objetos. Se implementan las clases Rama, Subred (Figura 3-18), *Triplet* y la estructura de sesiones activas.

```
class SubRed(object):
        def __init__(self, nombre, ip_mask, packetCount_old, bytesConsumidos_old, duration_old, pps, bps):
                self.nombre = nombre
                self.ip_mask = ip_mask
                self.packetCount_old = packetCount_old
                self.bytesConsumidos_old = bytesConsumidos_old
                self.duration_old = duration_old
                self.pps = pps
                self.bps = bps
        def get_packetCount_old(self):
                return self.packetCount_old
        def set_packetCount_old(self, packetCount_old):
                self.packetCount_old = packetCount_old
```

Figura 3-18 Clase Subred

Fuente: Elaboración propia

Por último, el módulo principal ejecuta cada *Delta X* segundos el monitoreo del consumo de las Ramas y las Subredes. Si encuentra un nuevo *Triplet* comienza a realizar en este el proceso *HandOff*. En este módulo se definen las siguientes funciones para realizar las reglas de balanceo de carga.

- crearGroupEntriesPorRama.- esta función inserta los *group entries* por cada Rama según (3.3.5.1) en el *switch SDN/OpenFlow* que será balanceador de carga
- crearFlowEntriesPorSubNet.- esta función inserta *flow entries* con la información de cada subred en el *switch SDN/OpenFlow* que será balanceador de carga

- medirBps_Ovs.- con la información de los datos del puerto, dirección (TX o RX) y el identificador del *switch* (PID) devuelve el consumo en bps
- medirFlows_Ovs.- con la información del identificador de un *switch* (PID) depura la data de la REST API del módulo *OpenFlow Stats* (recuperar las características de todos los *flow* existentes en un *switch*) para retornar los valores duración, cantidad de paquetes y *bytes* de todos los *flow entries* del *switch* en una estructura diccionario.
- ejecutarFlowEntriesSessionesActivas.- esta función recibe la información de las sesiones activas para insertarlas en un *switch SDN/OpenFlow*.

Por otro lado, el código del servidor recolector de sesiones activas está agrupada en tres módulos: comunicación, modelos y eventos.

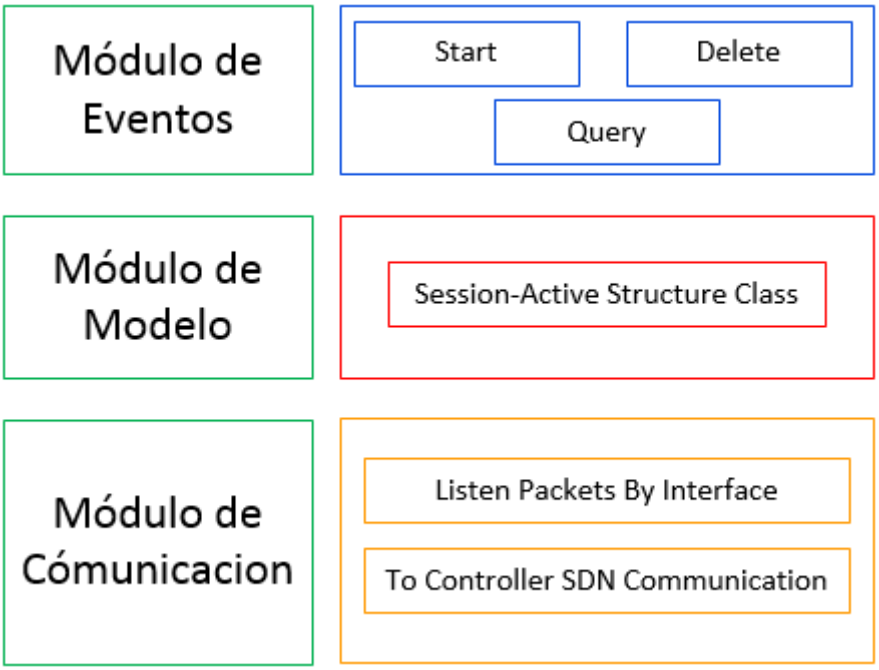

Figura 3-19 Arquitectura a nivel de código del servidor recolector de sesiones activas

Fuente: Elaboración propia

En primer lugar, el módulo de comunicación comprende la interfaz de comunicación entre el servidor y el controlador SDN (cliente) a través de solicitudes HTTP. Asimismo, este módulo captura en todo instante paquetes (proveniente de las subredes a migrar) por una interfaz especifica.

En segundo lugar, el módulo de modelos almacena las sesiones activas en la estructura de datos elegida en (3.3.5.2) para su manipulación en el módulo de eventos.

Por último, el módulo de eventos está conformado por los siguientes eventos que el cliente (controlador SDN) solicita. Cada solicitud del cliente contiene información sobre la subred a migrar.

- *Start*.- crea una instancia del modelo (*Session-Active Structure*) para almacenar las sesiones activas de una subred específica.
- *Query*.- envía el modelo (*Session-Active Structure*) de una subred específica al solicitante.
- *Delete*.- elimina el modelo (*Session-Active Structure*) de una subred específica.

3.4.3 Servidor de monitoreo de sesiones activas

La implementación del servidor monitoreo de sesiones activas se basa en un servidor HTTP escrito en el lenguaje *Python*. Para ello utilizamos la clase BaseHTTPRequestHandler que maneja solicitudes (GET/POST) HTTP que llegan desde el cliente al servidor. Para esta tesis, las solicitudes del controlador SDN (cliente) hacia el servidor recolector que maneja los eventos descritos en (3.4.2) será por medio de mensajes GET.

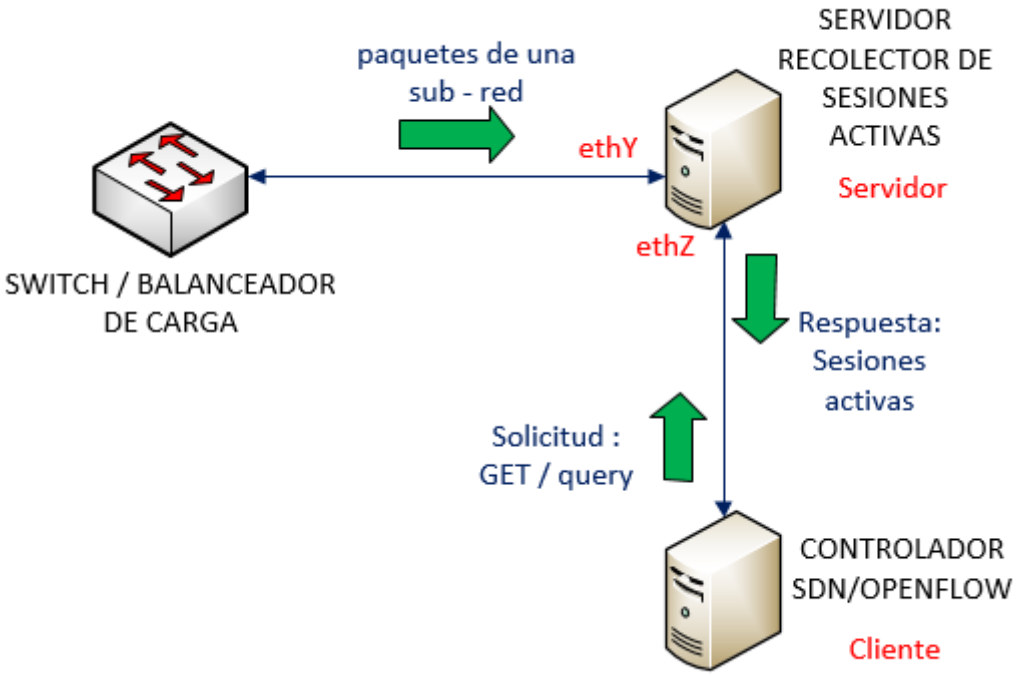

Figura 3-20 Servidor de monitoreo de sesiones activas

Fuente: Elaboración propia

La captura de paquetes por la interfaz (ethY), según la Figura 3-20, se realiza con la herramienta *sniff* de la librería *Scapy* de *Python*. Constantemente se monitorea los paquetes provenientes de las subredes a migrar.

4. PRUEBAS Y RESULTADOS

En el siguiente capítulo se detalla la prueba realizada para determinar la funcionabilidad del balanceador de carga según los requerimientos planteados. La prueba que se muestra en la siguiente sección evidencia que el diseño propuesto tiene el mismo rendimiento que el modelo de protección activo - respaldo durante operación normal.

4.1 Prueba de concepto

En esta sección se mostrará cómo el sistema es capaz de mover tráfico de una Rama congestionada a otra sin congestión previniendo la pérdida de paquetes con una probabilidad de error que está en el rango de (10^{-5} a 10^{-6}). Para esto se genera tráfico IP en las dos Ramas donde se incrementa el volumen de tráfico en la primera Rama. De no mediar la intervención de nuestro sistema (balanceador de carga) resultaría en la perdida de paquetes. Los elementos para la prueba son los siguientes:

- Generador de tráfico IP.- 1 PC, procesador arquitectura x86-64 - 8 núcleos, memoria RAM 16GB, OS *Linux Kernel* versión 4.15.0-42-generic, versión OS 18.04.02 LTS, instalado *Python* > versión 2.7.12 con la librería *Scapy*.

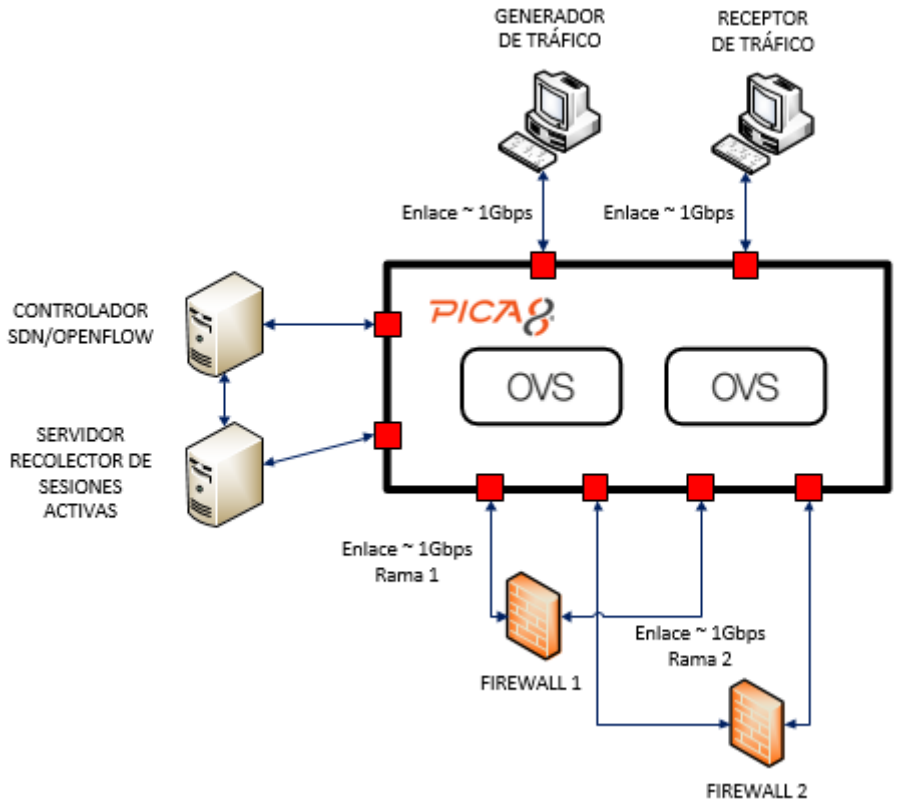

Figura 4-1 *Testbed* para la prueba de concepto en equipos físicos

Fuente: Elaboración propia

- Receptor de tráfico IP.- 1 PC, procesador arquitectura x86-64 - 8 núcleos, memoria RAM 16GB, OS *Linux Kernel* versión 4.15.0-42-generic, versión OS 18.04.02 LTS, instalado la aplicación *Gulp* [29].
- *Switch SDN/OpenFlow*.- 1 *switch* PICA8 modelo P-3297.
- *Firewalls*.- 2 PC, procesador arquitectura x86-64 - 8 núcleos, memoria RAM 16GB, OS *Linux Kernel* versión 4.15.0-42-generic, versión OS 18.04.02 LTS, dos tarjetas de red disponibles (capacidad ~ 1 Gbps), instalado la librería *"connection tracking system"*
- Servidor para la recolección de sesiones activas.- 1 PC, procesador arquitectura x86-64 - 8 núcleos, memoria RAM 16GB, OS *Linux Kernel* versión 4.15.0-42-generic, versión OS 18.04.02 LTS, dos tarjetas de red disponibles (capacidad ~ 1 Gbps).
- Controlador *SDN/OpenFlow*.- 1 PC, procesador arquitectura x86-64 - 8 núcleos, memoria RAM 16GB, OS *Linux Kernel* versión 4.15.0-42-generic, versión OS 18.04.02 LTS, dos tarjetas de red disponibles (capacidad ~ 1 Gbps), instalado *Floodlight* versión 1.2.
- Enlace.- todos los enlaces de ~ 1 Gbps.

Los elementos descritos anteriormente se configuran como se muestra en el Figura 4-1. Estos están disponibles en el laboratorio V305 de la sección de Telecomunicaciones – PUCP.

4.2 Resultados de la prueba de concepto

El generador de tráfico utiliza la información de la traza ISPDSL-II (20100106-083000-0.dsl) [22] para la generación de los paquetes que están clasificados en dos subredes: Subred 1 - 0.0.0.0/1 y Subred 2 - 128.0.0.0/1. El *script* que genera el tráfico fue desarrollado por el autor de esta tesis (utilizando la librería *Scapy* de *Python*) donde se envían 4 585 995 720 *Bytes* (aproximadamente 4.5 GB) que corresponden a 22 000 000 paquetes en un tiempo de duración de 40 segundos.

Se clasificó el tráfico en dos subredes: Subred 1 - 0.0.0.0/1 y Subred 2 - 128.0.0.0/1. El tráfico generado tiene una intensidad promedio de aproximadamente 901.3 Mbps, con picos temporales que alcanzan hasta 1000 Mbps (máxima capacidad de la tarjeta de red). La subred 1 tiene la menor intensidad de tráfico. Se modeló un escenario donde la Rama 1 soportó toda la capacidad total del tráfico. Se predeterminó el nivel umbral para el proceso de migración de carga en 500 Mbps. A través de esta prueba se verifica que el proceso de migración de tráfico (de la Subred 1) a la Rama 2 es efectivo. Se evita la sobrecarga sin pérdidas de paquetes o interrupciones de sesión.

La (Figura 4-2, arriba) muestra en azul el tráfico que se transporta por la Rama 1 mientras usa el balanceador de carga. La curva en rojo muestra el tráfico original que se transportaría por la Rama 1 si el sistema no estuviera presente. Se puede ver que el sistema original resultaría en una sobrecarga a partir del segundo 24. Por otro lado, el sistema evita la congestión al migrar el tráfico a otra Rama (que se muestra en la Figura 4-2, abajo). Para visualizar el proceso, la curva negra discontinua en la parte superior de la Figura 4-2 muestra la salida del filtro que realiza el promedio y el momento en que esta salida alcanza el umbral (500 Mbps) alrededor del segundo 8. En este momento, el Servidor Recolector de Sesiones Activas comienza a recopilar información sobre las sesiones activas (2996 en total) e inserta las entradas *OpenFlow* específicas de la sesión en el *switch SDN/OpenFlow*. Se observa que 15 segundos después, el sistema comienza a redireccionar el nuevo tráfico de sesión hacia la segunda Rama (Figura 4-2, abajo).

Se utiliza la aplicación *Gulp* [29] que propone asignar explícitamente los procesos de lectura y escritura de paquetes a diferentes CPU / núcleos y aumenta la prioridad en el proceso de lectura de paquetes. Cuando la velocidad de escritura es menor a la velocidad de lectura *Gulp* utiliza el *ring buffer* de la NIC. La escritura de paquetes en el disco en la PC que se comporta como Receptor de Tráfico tiene un impacto significativo en 234 paquetes (de 22 millones) *overrun* identificados. Esto es debido a que la memoria caché del *buffer* de la PC se llena al

capturar esa magnitud de paquetes. Para optimizar el proceso de escritura se debe utilizar la memoria RAM de la PC y dividir en trozos el archivo donde se guardó los paquetes recibidos. Esto debido a que el proceso de escritura lee todo el archivo *pcap* para escribir un nuevo paquete, mientras más grande es el archivo más lento es el proceso de lectura. Por esta razón, se utilizó la *ramdisk* de la PC Receptor de Tráfico (con 6GB de reserva) para que el proceso de lectura se guarde en la memoria RAM de la PC.

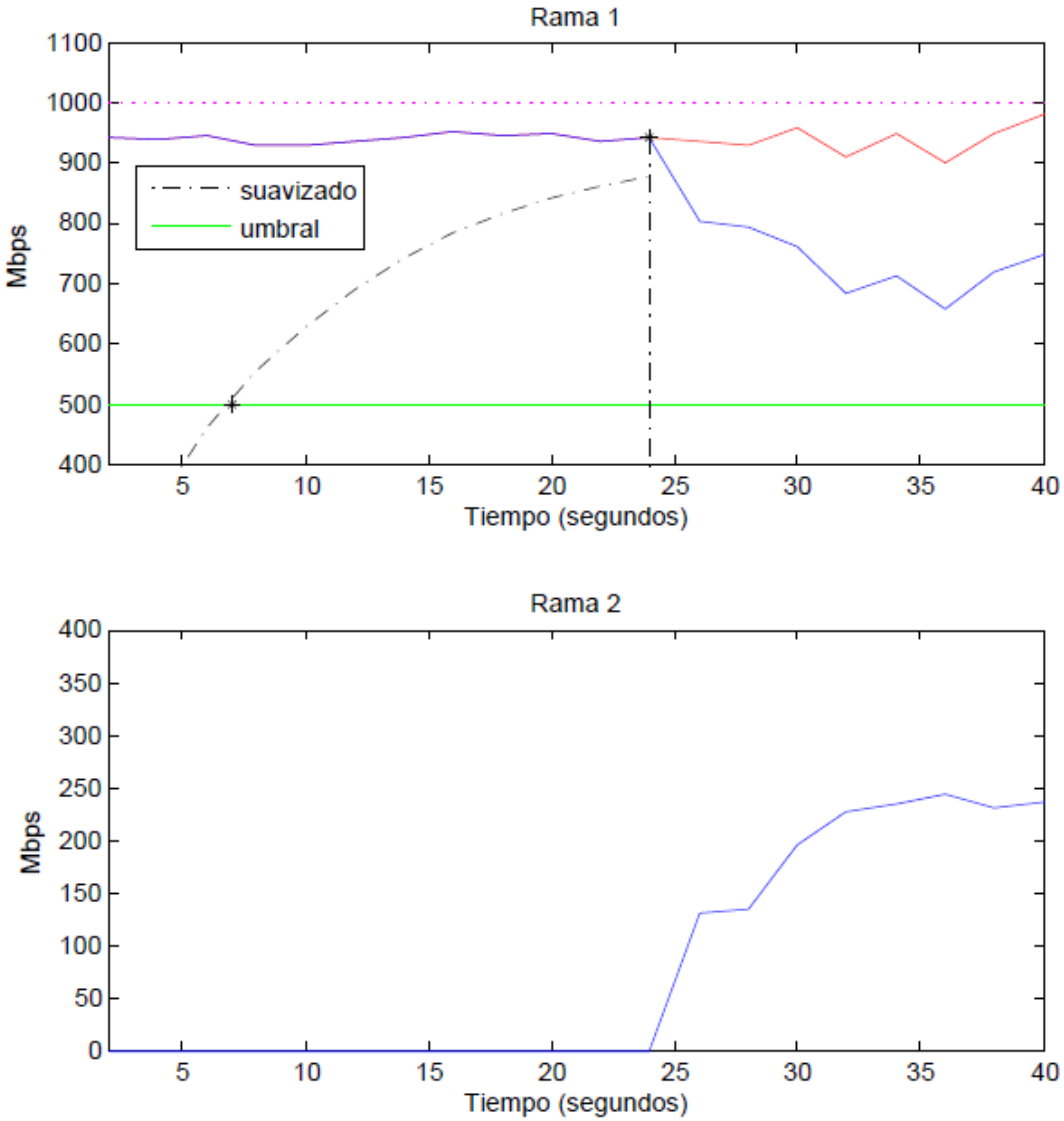

Figura 4-2 Proceso de *Hand Off* en las Ramas

Fuente: Elaboración propia

Se observa que 42 paquetes de 22 millones (con una probabilidad que está en el rango de 10^{-5} a 10^{-6}, error aceptable para este trabajo) se pierden en el segundo ocho donde el controlador *SDN/OpenFlow* comienza a enviar la instrucción de *mirroring* al *switch SDN/OpenFlow* y comienza la recolección de sesiones activas. Esto es debido a que la instrucción *mirror* (implementada con *buckets)* ejecutada desde el controlador al *switch* PICA8

consume más del 75% de la utilización del CPU por el proceso *ovs-vswitchd* lo cual genera la pérdida de estos paquetes. Cabe mencionar que este incidente ocurre en el instante antes de la migración de tráfico desde el *firewall* congestionado al descongestionado por lo que el proceso de *HandOff* no es sensible a pérdida de paquetes.

Para mejorar la prueba de concepto se sugiere usar una instrucción de *mirroring* ejecutada en el hardware del PICA8 para disminuir la utilización del CPU y asegurarse de que el controlador no envíe mensajes *OpenFlow* excesivos al *switch* PICA8. Asimismo, utilizar otras herramientas más sofisticadas y de mejor rendimiento para la recepción del tráfico IP como las siguientes: (a) *n2disk* (gran performance para capturas de tráfico del orden de los 10Gbps), (b) usar una tarjeta capturadora *Endace* DAG 3.7G y el software de captura de trazas WDCap (version 3.1.1) con Libtrace (versión 3.0.6) o (c) usar una tarjeta NIC complementada con el *kernel patch* de Linux *nCap* (este parche mejora el rendimiento de la captura de paquetes usando una tarjeta de red simple). Con estas sugerencias se conseguirá mejoras en la efectividad de recibir todos los paquetes enviados a tasas más altas que 1Gbps. Con esta prueba se demuestra la funcionalidad del balanceador de carga propuesto en este trabajo de tesis según los requerimientos planteados.

CONCLUSIONES

- Se logró cumplir con el objetivo principal de la tesis, pues se realizó el diseño e implementación de un balanceador de carga vía un *switch SDN/OpenFlow* para un banco de *Firewalls* N:1, el cual permite la permanencia de las sesiones activas establecidas en los *firewalls*. Se evidencia que el balanceador de carga presenta un funcionamiento en un entorno cuya intensidad de tráfico bordea el 1Gbps.

- Se observa que el diseño del balanceador soporta la migración de carga de una subred predeterminada de una Rama a otra sin pérdida de paquetes o interrupción de las sesiones migradas.

- El diseño del balanceador funciona en *switches* físicos o virtuales que soporten el estándar *OpenFlow*. En las pruebas se demostró su funcionamiento a través del controlador *Floodlight*. Se demostró que usando el algoritmo para el balanceador se logró utilizar 2996 *flow entries* para no perjudicar las sesiones activas. Esta cifra es menor a 8K el cual es el límite de entradas posibles en la memoria *TCAM* del *switch*.

- Se concluye que el balanceador de carga es compatible con el modelo de protección banco de *Firewalls* N:1 que sustituye al modelo activo - respaldo. Por lo tanto, se reduce significativamente el costo de capital en equipos de protección debido a la adquisición de *firewalls* en función a la demanda por el uso del balanceador propuesto en este trabajo.

TRABAJOS FUTUROS

- A través del modelo *fractional Brownian Motion* (*fBm*) derivar un método para el dimensionamiento de la cantidad de *firewalls* a utilizar en el banco N:1. Para esto se deberá hallar los indicadores estadísticos que representen el tráfico de Internet de una red empresarial en particular.
- Modelar el proceso de balanceo de carga y *"firewall hand-off"* como un sistema de control de lazo cerrado para que el porcentaje de paquetes perdidos en el proceso de migración de una subred sea mínimo.
- Implementación del módulo *"firewall bypass"* que consiste en la migración de sesiones con alto volumen de tráfico que ya fueron validadas y no son un riesgo para el sistema (por ejemplo flujos de video-*streaming*) para evitar que estos flujos sean procesados por los *firewalls* y así reducir aún más el número de *firewalls* en el banco N:1 ; en efecto, reducir el costo monetario del sistema.
- Comparación del monitoreo de tráfico entre el recolector de sesiones activas y *sFlow*.
- Probar el rendimiento del balanceador de carga en un *TestBed* de ciberseguridad o en un escenario *backbone* donde el tráfico tenga una intensidad superior a 10 Gbps.
- Utilizar una tarjeta capturadora *Endace* DAG 3.7G, la aplicación *n2disk* o una tarjeta NIC complementada con el *kernel patch* de Linux *nCap* para implementar el receptor de tráfico para velocidades superiores a 1Gbps.
- Como las pruebas realizadas en este trabajo están sujetas a una tasa de transmisión de 1 Gbps se observa que el tráfico generado por el sistema presenta fluctuaciones en el tiempo que dura la prueba; es decir, 40 segundos (debido a las limitantes en memoria, procesador y tarjetas de red de la PC que actúa como generador de tráfico). Probar que en entornos superiores a (3 o 10 Gbps) estas fluctuaciones disminuirán mejorando la performance del balanceador de carga.

BIBLIOGRAFÍA

[1] H. León, «Diseño de un sistema de proteccion compartida N:1 para firewalls de la PUCP,» Lima, 2015.

[2] F. M. V. R. P. D. Kreutz, Software-Defined Networking: A Comprehensive Survey, 2014.

[3] O.N.F, «Software-defined networking: The new norm for networks,» 2012.

[4] K. P. E. Haleplidis, «Internet Research Task Force (IRTF),» Enero 2015. [En línea]. Available: https://tools.ietf.org/html/rfc7426.

[5] B. Heller, «OpenFlow Switch Specification 1.0.0,» 2009.

[6] R. Communications, «Store NetGate,» 2017. [En línea]. Available: https://store.netgate.com/Pica8--C188.aspx.

[7] S. R. a. D.Lenrow, «Open Networking Foundation North Bound Interface Working Group,» 2013.

[8] C. M. R. A.G. Centeno, «Controladores SDN, elementos para su selección y evaluación,» *Revista Digital de las tecnologías de la informacion y las comunicaciones,* vol. 13, nº 3, pp. 10-20, 2014.

[9] J. M. M. Vázquez, «Cortafuegos, Comparativa entre las Distintas Generaciones y Funcionalidades Adicionales,» 2002.

[10] A. Guide, «Palo Alto Networks,» 2017. [En línea]. Available: https://www.paloaltonetworks.com/documentation/70/pan-os/pan-os/networking/tcp#_43095.

[11] K. S. Jr, «Load Balancing 101: The Evolution to Appication Delivery Controllers,» 2007.

[12] K. Salshow, «Load Balancing 101: Nuts and bolts,» 2007.

[13] J. B. G. Cuba, *Diseño e Implementación de un controlador SDN/OpenFlow para una red de campus académica,* Lima: PUCP, 2015.

[14] Brocade, *Brocade ServerIron ADX Application Delivery Switches.*

[15] F. Networks, «K16419: Overview of BIG-IP memory usage,» 29 abril 2016. [En línea]. Available: https://support.f5.com/csp/article/K16419.

[16] M. T. W. Leland, «On the Self-Similar Nature of Ethernet Traffic,» 1993.

[17] J. Azor, «Análisis del comportamiento auto similar con distribución Pareto del tráfico Ethernet,» 2010.

[18] D. A. R. P. Agency, «IETF Tools,» setiembre 1981. [En línea]. Available: https://tools.ietf.org/html/rfc791.

[19] D. A. R. P. Agency, «IETF Tools,» setiembre 1981. [En línea]. Available: https://tools.ietf.org/html/rfc793.

[20] D. A. R. P. Agency, «IETF Tools,» agosto 1980. [En línea]. Available: https://tools.ietf.org/html/rfc768.

[21] W. J. a. S. Tafvelin, «Analysis of internet backbone traffic and header anomalies observed,» de *7th ACM SIGCOMM*, 2007.

[22] WAND, «WAND Network Research Group,» enero 2010. [En línea]. Available: https://wand.net.nz/wits/ispdsl/2/.

[23] L. Q. a. J. Heidemann, «On the characteristics and reasons of long-lived internet flows,» de *10th Annu. Internet Meas*, 2010.

[24] C. Quispe, «GitHub,» febrero 2017. [En línea]. Available: https://github.com/wirkentod/Trace-Traffic-Analizer.git.

[25] C. Quispe, «GitHub,» febrero 2017. [En línea]. Available: https://github.com/wirkentod/pypcapfile.git.

[26] MatLab, «MathWorks - Sliding Window Method and Exponential Weighting Method,» 2017. [En línea]. Available: https://www.mathworks.com/help/dsp/ug/sliding-window-method-and-exponential-weighting-method.html.

[27] J. Hartley, «Python - TimeComplexity,» junio 2017. [En línea]. Available: https://wiki.python.org/moin/TimeComplexity.

[28] B. Heller, «OpenFlow Switch Specification 1.0.0,» 2009.

[29] C. Satten, «Lossless Gigabit Remote Packet Capture With Linux,» University of Washington, 9 Agosto 2007. [En línea]. Available: https://staff.washington.edu/corey/gulp/.