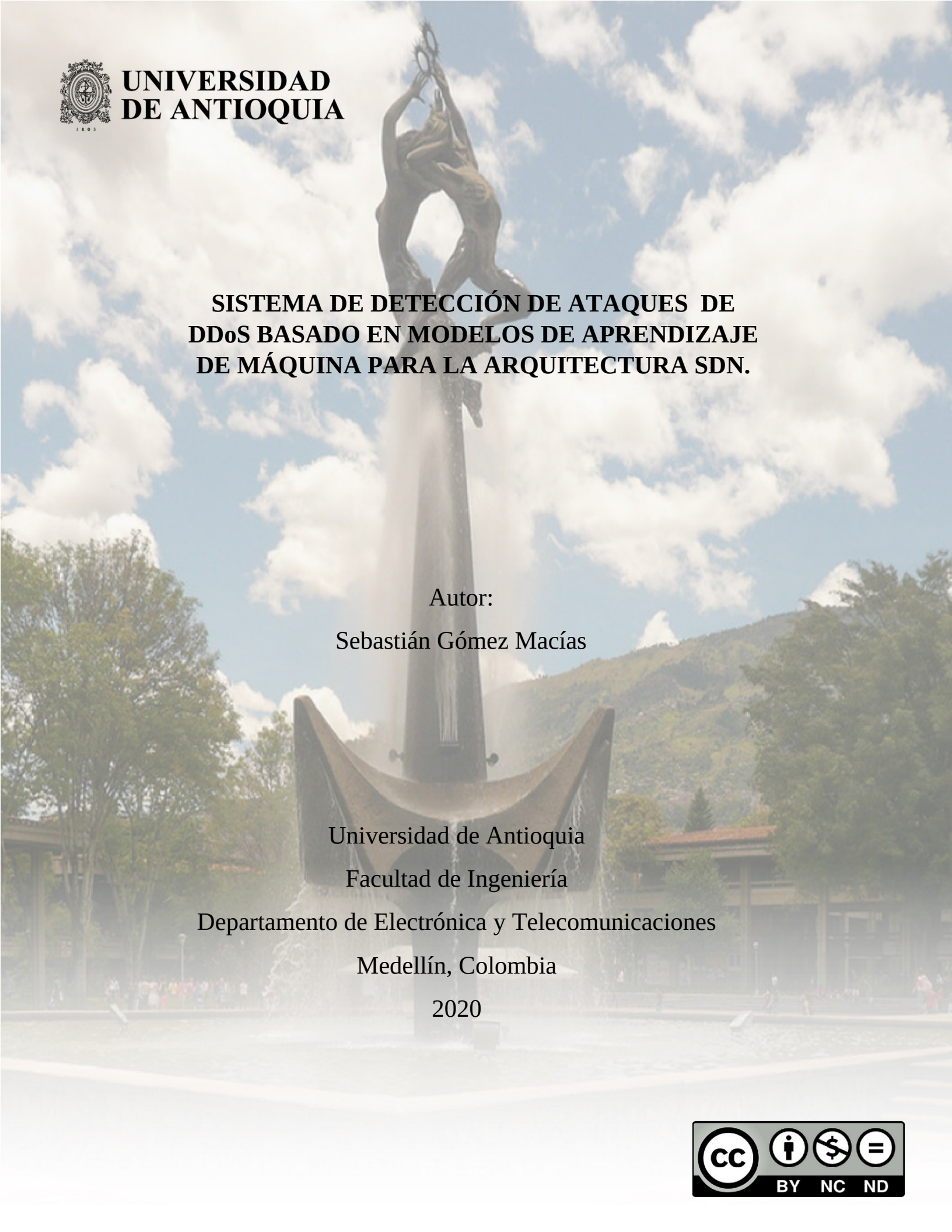


UNIVERSIDAD DE ANTIOQUIA

SISTEMA DE DETECCIÓN DE ATAQUES DE DDoS BASADO EN MODELOS DE APRENDIZAJE DE MÁQUINA PARA LA ARQUITECTURA SDN.

Autor:

Sebastián Gómez Macías

Universidad de Antioquia

Facultad de Ingeniería

Departamento de Electrónica y Telecomunicaciones

Medellín, Colombia

2020

Sistema de Detección de Ataques de DDoS Basado en Modelos de Aprendizaje de Máquina para la Arquitectura SDN.

Sebastián Gómez Macías

Trabajo de investigación presentado como requisito parcial para optar al título de:

Magíster en Ingeniería de Telecomunicaciones

Director:
Ph.D. Juan Felipe Botero Vega

Línea de Investigación:
Modelamiento de sistemas de comunicaciones
Grupo de Investigacion en Telecomunicaciones Aplicadas (GITA)

Universidad de Antioquia
Facultad de Ingeniería
Medellín, Colombia
2020

Agradecimientos

Primero que todo agradezco a mi querida Universidad de Antioquia por la beca de “Estudiante Instructor” que me concedió para poder hacer realidad mi sueño de ser Magíster. Agradezco al profesor PhD. Juan Felipe Botero Vega por la orientación y la dedicación en la realización de este trabajo, además de la oportunidad brindada para hacer la pasantía en la UFRGS de Brasil.

Agradezco también a mi familia y a mi novia por estar incondicionalmente presentes a lo largo de este proceso, especialmente en aquellos momentos de duda. Por sus palabras fortalecedoras muchas gracias.

Resumen

Los ataques de denegación de servicio distribuidos (DDoS) son uno de los ataques mas utilizados por los ciberdelincuentes a nivel mundial por el alto grado de letalidad al que pueden llegar. Estos ataques pueden causar la caída de hasta los mas grandes sitios web en Internet, generando consigo consecuencias financieras y/o políticas.

Por otro lado, las redes definidas por software (SDN) son el nuevo paradigma de gestión de las redes de datos pues promueven que tanto el control como la visualización de la red sea llevado a cabo desde un controlador centralizado por medio de software. Gracias a las ventajas de SDN y las estadísticas de tráfico que proporciona, se han podido implementar diversos mecanismos de detección de DDoS en infraestructuras de red soportadas por SDN. Sin embargo, se han identificado dos limitantes a la hora de implementar este tipo de soluciones: i) los procesos desplegados por el sistema de detección pueden sobrecargar el controlador en presencia de gran cantidad de tráfico a analizar, y ii) la información del tráfico que se puede extraer desde el plano de datos es muy limitada, lo que restringe el uso de ciertas características en el sistema.

Para disminuir el impacto de estas limitantes en el momento de diseñar e implementar el sistema de detección, esta monografía propone un entorno colaborativo entre un plano de datos programable y un plano de control. En este entorno, se utiliza P4, un reciente lenguaje de programación para el plano de datos, para programar los dispositivos de este plano con el objetivo de extraer y procesar información de los flujos a un mayor nivel de granularidad. Para llevar a cabo esta propuesta, se propone un mecanismo de almacenamiento de información por flujo usando estructuras hash. Adicionalmente, se propone un mecanismo de reporte de dicha información al plano de control, plano que se encarga posteriormente del cálculo y la clasificación del conjunto de características.

Con este entorno y usando dichos mecanismos, fue posible la implementación de características de tipo Intra-Flujo en la detección de DDoS, características que no son extraíbles en un entorno tradicional SDN-OpenFlow. Con este conjunto de características se entrenaron modelos de K-vecinos más cercanos (KNN) y bosque aleatorio (RF), los cuales arrojaron buenos resultados en desempeño de detección, siendo el mejor de ellos KNN con 96 % de exactitud. Adicionalmente, al reportar información ya procesada al plano de control, se optimizaron los procesos de cálculo y clasificación del conjunto de características, obteniendo consumos promedio de CPU inferiores al 50 % incluso en presencia de ataque de DDoS.

Glosario

Bwd Backward Direction.

CAPEX Gastos de Capital.

CTE Carga de trabajo de entrenamiento.

CTP Carga de trabajo de prueba.

DDoS Distributed denial of service.

DSL Domain Specific Lenguaje.

DT Decision Tree.

Flow-IAT Tiempo de llegada entre paquetes del mismo flujo.

Fwd Forward Direction.

KNN K-Nearest Neighbors.

ML Machine Learning.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

OPEX Gastos de Operación.

RF Random Forest.

SDN Software Defined Networking.

std Standard Deviation.

TW Time-Window.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El ataque de denegación de servicio distribuido (DDoS) es un tipo de ataque que busca que los usuarios de un servicio de red tengan dificultades parciales o totales en el momento de usar dichos servicios [1]. Esta denegación es ocasionada por una gran cantidad de peticiones a una alta velocidad, con el fin de agotar los recursos del dispositivo e imposibilitarlo para atender nuevas peticiones legítimas. El alto volumen de tráfico DDoS es conseguido mediante un trabajo coordinado entre múltiples computadoras infectadas conocidas como robots o zombies, de manera que sus propietarios normalmente desconocen aquella inyección de tráfico de ataque que sale de sus máquinas hacia la víctima. Las altas tasas a las que puede llegar el ataque, pueden ser lo suficientemente letales para ocasionar daños que van desde el apagado del sistema y la corrupción de archivos, hasta la pérdida total o parcial de los servicios [1].

Los ataques de DDoS son considerados una de las armas mas poderosas en contra de la seguridad de las redes debido a la gran capacidad de poder denegar hasta los mas grandes servicios existentes en internet [2], trayendo consigo posibles consecuencias políticas y económicas [3]. De acuerdo al reporte Q1 del 2019 de *Kaspersky Lab*, el numero total de ataques a nivel mundial se ha incrementado en un 89 %, la duración promedio de los ataques se ha incrementado 4.21 veces y se han detectado picos máximos de transmisión de tráfico DDoS de hasta 468 Gbps [4]. Estas estadísticas nos demuestran la tendencia creciente del nivel devastador que está alcanzando este tipo de ataques.

Por otro lado, el nuevo paradigma de la gestión de las redes de datos: las Redes Definidas por Software (SDN)[5], son catalogadas como un nuevo paradigma de gestión gracias a la

centralización lógica del plano de control. Las principales contribuciones que trae esta nueva arquitectura radican en la vista global de la red desde un lugar centralizado y la posibilidad de desarrollar aplicaciones en el plano de control que permiten definir el comportamiento de la red gracias a las nuevas entidades programables que se pueden modificar en tiempo real. Este nuevo enfoque conlleva beneficios para las áreas de TI en ahorro de gastos de capital y de operación(CAPEX y OPEX), mayor flexibilidad en la automatización de la red y una disminución en el tiempo de la gestión.

Sin embargo, mas allá de los beneficios que trae SDN, los ataques de DDoS desplegados en esta arquitectura son igual de devastadores que en las redes tradicionales. Estos ataques son capaces de denegar también cualquier servicio de red incluyendo el servicio proporcionado por el mismo controlador [6][7]. Esta problemática de seguridad que aún predomina en las redes de nueva generación, dificulta el proceso de adopción y estandarización de las redes SDN en entornos de producción, ya que es necesario que se garanticen ciertos niveles de seguridad. Con base en lo anterior, se evidencia la necesidad de desarrollar mecanismos eficientes que detecten y mitiguen en una fase temprana los posibles ataques de DDoS que se generen en una red SDN.

Para la comunicación entre los dispositivos del plano de datos con el plano de control, SDN cuenta con una interfaz cuyo protocolo estándar de comunicación es OpenFlow [8]. Gracias a que OpenFlow proporciona algunas estadísticas del tráfico y SDN permite programar aplicaciones de red en el plano de control, fue posible que se implementaran diversos sistemas de detección de ataques de DDoS basados en modelos de aprendizaje de máquina sobre dicha arquitectura [9][10].

Sin embargo, analizando como operan los sistemas de detección existentes, se observa que la información suministrada por OpenFlow y la extraíble de los paquetes que ingresan al controlador (*packet_in*), limitan la consecución de ciertos tipos de características que podrían ser usadas también por el sistema de detección, al ser estas excelentes descriptoras del tráfico DDoS. Esta limitante surge de la poca información del tráfico que se puede recopilar en el controlador de un entorno SDN-OpenFlow. A partir de esta información recopilada, el sistema extrae el conjunto de características que clasifica el modelo de aprendizaje de máquina en busca de posibles ataques de DDoS. Por ejemplo, dentro del grupo de características no extraíbles se encuentran aquellas de tipo Intra-FLujo como lo son las basadas en los tiempos de llegada entre paquetes del mismo flujo, características que son recomendadas por los autores de [11].

En respuesta a esta limitante, se propone en esta monografía usar un entorno SDN que emplee un plano de datos programable en reemplazo de uno basado en OpenFlow. Este nuevo paradigma de los planos de datos permite programar la manera en que los diferentes dispositivos procesan los paquetes entrantes, lo que permite a la vez, extraer la información requerida directamente de la cadena de bits que forma cada paquete, es decir, a un nivel de granularidad mayor. Con base en esto, se propone entonces un sistema eficaz de detección de ataques de DDoS basado en un trabajo colaborativo entre el plano de datos y el plano de control como principal contribución de este trabajo.

Para llevar a cabo el trabajo colaborativo entre planos, se usó P4 [12], un lenguaje de programación de código abierto que permite programar los dispositivos del plano de datos. Con base en el uso de P4, se propone dentro del sistema un mecanismo en el plano de datos capaz de extraer y almacenar de forma eficiente la información extraída y procesada por cada flujo. Adicionalmente, se propone un mecanismo capaz de reportar periódicamente dicha información al plano de control, donde posteriormente se calcula y se clasifica el conjunto de características de cada flujo reportado.

Las contribuciones de este trabajo son:

- Se delega al plano de datos el despliegue de tareas como almacenamiento, organización y procesado de la información requerida por flujo. Esto permite facilitar la tarea de cálculo y clasificación del conjunto de características en el plano de control.
- La arquitectura implementada en este sistema propone lo necesario para poder extraer y usar diferentes tipos de características sin restricción alguna programando ambos planos SDN. Como prueba de esto, este trabajo explica la estrategia abordada para extraer en tiempo real características de tipo Intra-Flujo y así usarlas en la detección de DDoS, características que no son extraíbles en un entorno SDN-OpenFlow.

1.1. Estructura de la monografía

El contenido de este trabajo se divide en 6 capítulos organizados de la siguiente manera:

En el **capítulo 2** se introducen los conceptos relevantes que son usados constantemente a lo largo del resto de capítulos. Dentro de estos conceptos, se introduce el paradigma de las SDN, la programación del plano de datos por medio de P4, qué es y como se despliega un ataque de DDoS, como también el uso de algoritmos de aprendizaje de máquina para

la clasificación del tráfico. En el **capítulo 3** se presentan los trabajos previos relacionados, donde se exponen algunas limitantes encontradas que soportan el trabajo propuesto en esta monografía. En el **capítulo 4** se realiza una revisión conceptual de la solución propuesta al problema abordado y se explica a fondo la arquitectura implementada coordinando el plano de datos y de control para la detección de ataques DDoS. En el **capítulo 5** se explica el procedimiento experimental llevado a cabo, como también los resultados obtenidos en detección y consumo de recursos de procesamiento. Por último, en el **capítulo 6** se concluye la monografía y se presentan posibles trabajos futuros. Por otro lado, en los anexos se muestra información complementaria que puede ser de interés para el lector.

Capítulo 2

Marco Teórico

2.1. Redes Definidas por Software (SDN)

A principios de la década del 2000, el aumento de los volúmenes de tráfico y la necesidad de aumentar la confiabilidad y el rendimiento de la red, llevó a los operadores de redes a desplegar diferentes tipos de tareas tales como ingeniería de tráfico, depuración de problemas de configuración o la predicción y control del comportamiento del sistema de enrutamiento. Al poseer los dispositivos tradicionales una estrecha integración entre el plano de datos y el de control, ambos implementados en hardware, el despliegue de este tipo de tareas se tornó extremadamente difícil y frustrante especialmente en redes de gran tamaño [13]. En respuesta a la rigidez que posee ese tipo de arquitecturas, Nick Mckeown y Martín Casado, en la universidad de Stanford, introdujeron por primera vez el concepto de SDN [14] donde proponen la centralización lógica de todo el control de la red y la implementación de una interfaz abierta entre el plano de datos y control. Este nuevo paradigma proporciona una visión lógicamente centralizada de toda la red y ayuda a realizar cambios a nivel global sin necesidad de realizar configuración alguna directamente en cada dispositivo. Esta ventaja proporciona una mayor facilidad en la administración de las funciones de la red. Por otro lado, al extraer el plano de control de los dispositivos, se simplifican enormemente las tareas que estos realizan, permitiendo así la fabricación de dispositivos mas eficientes en la tarea especifica de reenvío de paquetes [15].

La arquitectura SDN es definida en 3 planos, el plano de aplicación, el plano de control y el plano de datos como se muestra en la figura 2.1. En el plano de datos se encuentran los dispositivos físicos de la red (conmutadores) encargados de la conmutación de los paquetes

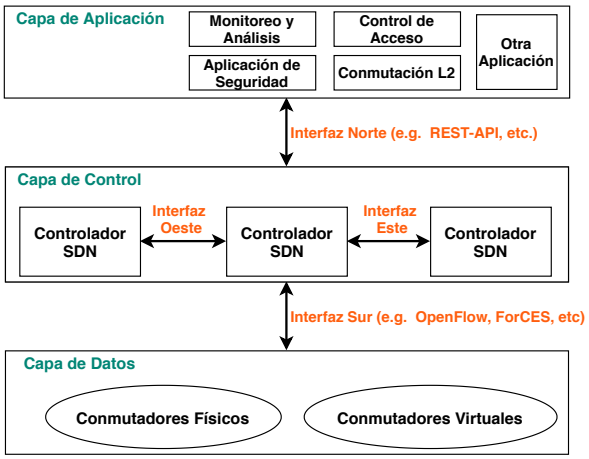


Figura 2.1: Arquitectura SDN.

con base en unas reglas de flujo ya preestablecidas y almacenadas en unas tablas que cada dispositivo tiene internamente.

El plano de control es una instancia basada en software que se encuentra lógicamente centralizada en alguna parte de la red. En este plano se encuentra el controlador, encargado de proporcionar a las aplicaciones la abstracción de detalle de bajo nivel, proporcionar la información del estado de la red, además de instanciar las reglas en las tablas de flujo que poseen los dispositivos del plano de datos. Para que se lleve acabo dichas funciones el controlador hace uso de la interfaz sur donde actualmente existen varias propuestas como *ForCES* [16], *OVSDB* [17], *OpenFlow* [18], entre otras, siendo esta última el estándar mas usado en las redes SDN.

Finalmente, el plano de aplicación es donde se encuentran todas las aplicaciones que definen el comportamiento de la red mediante aplicaciones que determinan el comportamiento de los flujos de tráfico. Dependiendo del controlador, las aplicaciones pueden ser desarrolladas en diferentes lenguajes de programación, incluso algunos controladores permiten tener interfaces estándar con las aplicaciones, por lo que es posible utilizar cualquier lenguaje.

2.1.1. OpenFlow

OpenFlow se ha convertido en el mecanismo estándar de la interfaz sur de la arquitectura SDN, apoyado por la Open Networking Foundation (ONF) en temas de desarrollo y manutención [19][20]. Este protocolo controla la interacción entre el controlador y los dispositivos de red para llevar a cabo tareas como la instancia de reglas de flujos para el reenvío de paquetes o la recopilación de estadísticas de flujo. En la figura 2.2 se pueden observar los principales

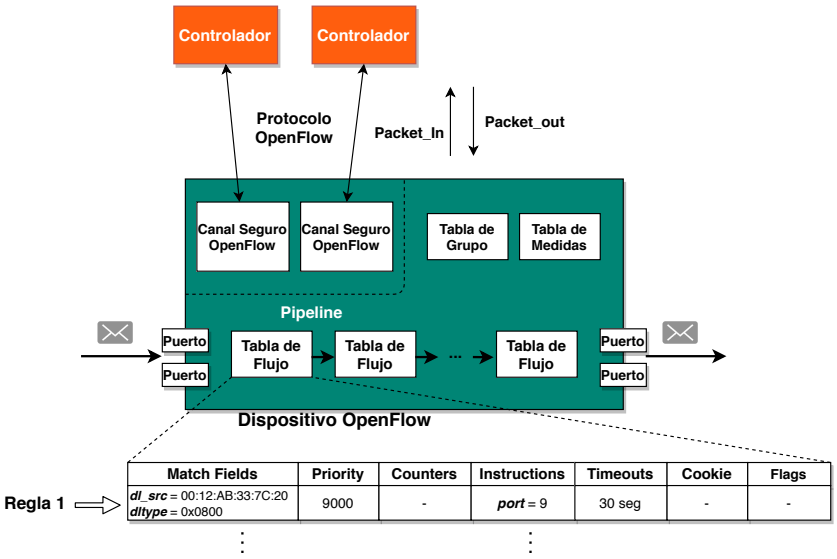


Figura 2.2: Principales componentes de OpenFlow versión 1.5.0 [8]

componentes que OpenFlow define: uno de ellos son las tablas de flujo en los dispositivos, donde se albergan por un tiempo determinado las reglas instanciadas por el controlador. Cada regla de flujo contiene 7 campos, entre ellos, los valores de cabecera que definen el flujo (Match-Fields), la acción a aplicarle al paquete que ingrese al dispositivo cuando pertenezca a ese flujo y el tiempo de vida de la regla.

Cuando ingresa un paquete al dispositivo, este comienza a buscar en todas las tablas alguna regla almacenada con la que el paquete tenga coincidencia, en caso que no exista, se envía un paquete llamado *packet_in* al controlador, para que éste lo procese e instale una nueva regla en el dispositivo. Este intercambio de paquetes entre controlador y dispositivos es realizado por medio de una interfaz proporcionada por OpenFlow, que usa el protocolo TLS (Transport layer security) para conexiones cifradas o en su defecto TCP. OpenFlow se encarga también de definir los estándares de serialización y negociación al establecerse la conexión, como también los diferentes mensajes que puede desplegar una aplicación para insertar, eliminar, modificar o consultar las entradas de flujo contenidas en las tablas de los dispositivos [18][8].

Una funcionalidad importante que proporciona OpenFlow para el desarrollo de sistemas de seguridad basados en detección de anomalías, son los contadores (estadísticas). El estado de estos contadores puede ser solicitado por el controlador en cualquier momento enviando un mensaje llamado OFPT al dispositivo. OpenFlow define contadores para las siguientes entidades: tabla y entrada de flujo, puerto, cola, grupo y medidores. Estos contadores contienen el número de paquetes que interactuaron con cada entidad mencionada, al igual que

el agregado de bytes. En la tabla 2.1 podemos apreciar los contadores proporcionados por OpenFlow por cada flujo detectado en el plano de datos; la tabla muestra los mas usados por los sistemas de detección para definir descriptores de tráfico.

Tabla 2.1: Contadores OpenFlow por entrada de Flujo [8]

Contador	Peso
Paquetes Recibidos	8 Bytes
Bytes Recibidos	8 Bytes
Duración (segundos)	4 Bytes
Duración (nanosegundos)	4 Bytes

2.2. Programabilidad en el Plano de Datos

Una noción que se hizo popular en SDN es la habilidad de "programar"las tablas de los conmutadores al poder poblarlas en tiempo real con reglas de flujo definidas por aplicaciones programadas en la plano de control y/o de aplicación. Este enfoque simplista, flexible e innovador permitió controlar de manera eficiente el comportamiento de la red al definir acciones de reenvío asociadas a cada flujo. Sin embargo, este nuevo método de gestión de las redes terminó evidenciando nuevas limitantes que contradicen el concepto de facilidad y flexibilidad en la gestión de las redes. Las principales limitantes identificadas fueron:

- Los operadores no pueden diseñar sus encabezados de paquete personalizados, ni tampoco la forma de particionar la cadena de bits del paquete a analizar y procesar dentro del dispositivo [21].
- Para admitir nuevos protocolos en el plano de datos se requiere de un proceso arduo y lento. El proceso tiene que pasar por una fase de estandarización del protocolo, una fase de implementación en hardware, además de implementar una nueva versión de OpenFlow que admita los campos del encabezado de dicho protocolo. Este proceso puede llevar años, lo que dificulta considerablemente la propuesta y la adopción de protocolos innovadores [21].

Para superar las anteriores limitaciones, emerge el concepto de plano de datos programable que reestructura el panorama SDN al permitir a los operadores de red re-programar los conmutadores. Esta nueva reestructuración busca que sea posible programar completamente la forma como se procesan los paquetes, desde la modificación y/o adición de encabezados, hasta la implementación de diferentes tablas de flujo. Con un plano de datos completamente

maleable, la implementación y adopción de nuevos protocolos de red se hará de una forma rápida, permitiría personalizar el comportamiento de la red y, en consecuencia, desarrollar servicios innovadores.

Con base en este nuevo concepto, para programar los conmutadores del plano de datos es necesario contar con un lenguaje especifico de dominio (DSL) y un compilador estándar suplido por el proveedor de dichos dispositivos. Con estas dos herramientas se puede desarrollar una aplicación de plano de datos, compilarla e instalarla en caliente en los conmutadores por medio del plano de control. El DSL mas usado y estandarizado por la ONF es el lenguaje de programación P4 [12].

2.2.1. P4

P4 es un lenguaje de alto nivel diseñado para programar el procesador de paquetes de los dispositivos del plano de datos. Este lenguaje permite especificar las operaciones que un procesador ejecuta sobre un paquete desde que se recibe hasta que se reenvía o se descarta (análisis, modificación de cabeceras, búsqueda en tablas de encaminamiento, etc). P4 fue desarrollado para cumplir tres objetivos principales:

- **Reconfigurable.** La forma de particionar el paquete y de procesar los campos de las cabeceras deben ser re-programables.
- **Independiente del protocolo.** El conmutador no debe estar vinculado a formatos de paquetes específicos. En cambio, el operador de la red debe poder programar una forma de "traducir" el paquete para extraer los campos de las cabeceras con un nombre y tipo particular, además de poder definir una colección de tablas Match+Action que procesen estos encabezados.
- **Independiente del dispositivo.** P4 debe ser agnóstico a la plataforma donde se implemente la aplicación P4 programada, es decir, la misma aplicación debe poder mapearse a dispositivos distintos (por ejemplo, un Barefoot Tofino, BMV2 o una NetFPGA). Es el respectivo compilador quien genera una salida adecuada para la plataforma especifica.

A la fecha existen dos versiones de este lenguaje: $P4_{14}$ la cual está obsoleta y $P4_{16}$ que posee una sintaxis mas precisa y una menor cantidad de funcionalidades para cumplir los estándares exigidos para una generalización en las diferentes plataformas. En este documento toda la implementación fue realizada con la versión $P4_{16}$, por lo tanto nos referiremos a las especificaciones de esta versión.

Para entender como se aplica el lenguaje P4 a las plataformas programables, hay que tener presentes tres conceptos esenciales:

- **Target:** Es toda aquella plataforma sea hardware o software en la que se ejecuta una aplicación P4.
- **Arquitectura:** Hace referencia al conjunto de componentes P4 programables, objetos externos, interfaces y demás recursos con los que fue dotado el *target* en el momento de su diseño. En un principio P4 fue pensado para programar solamente conmutadores, pero hoy en dia son varios los *targets* en los que se puede desplegar, por lo que existe más de una arquitectura diferente.
- **Pipeline:** hace referencia al conjunto de elementos (tablas y objetos externos) conectados normalmente en serie, que procesan las cabeceras del paquete.

Conocer bien como esta formada la arquitectura del *target* a programar, es un paso crucial antes de comenzar a desarrollar la aplicación P4 ya que de ella depende la forma como estará estructurado el programa. En la Figura 2.3, se muestra la arquitectura *V1model*, una de las arquitecturas implementadas en los targets tipo conmutador, pensada para acoplarse a la forma de trabajar de estos dispositivos.

Un conmutador, con varios puertos de red, genera conexiones internas entre los puertos de entrada y salida por los que pasa cada paquete. De acuerdo a este funcionamiento, la arquitectura V1model es basada en 2 *pipelines* independientes (Ingreso y Egreso) interconectados por un buffer (ver Fig. 2.3). Cuando ingresa un paquete al conmutador, este pasa directamente a ser tratado por el *pipeline* de ingreso que cuenta con un bloque programable encargado de particionar (parsear) el paquete, seguido de otro bloque programable donde se define el conjunto de tablas *match-action* y la verificación del *checksum*. Antes de transmitir el paquete, la información traducida y tratada en el *pipeline* de ingreso pasa por el *pipeline* de egreso. Este segundo *pipeline* cuenta con otro bloque programable donde se pueden definir diferentes tablas *match-action*, además de actualizar el *checksum* con base en las modificaciones hechas a los campos de las cabeceras al pasar por estas tablas. Adicionalmente, el *pipeline* de egreso se encarga de ensamblar de nuevo el paquete y de generar la cadena de bits a ser transmitida a su destino.

Con base en el funcionamiento lineal de la arquitectura V1model, una aplicación P4 estará constituida principalmente de 5 partes: i) definición de cabeceras, ii) *parser*, iii) bloque control de ingreso, iv) bloque control de egreso y v) *Deparser*. Estas 5 partes se explicarán a

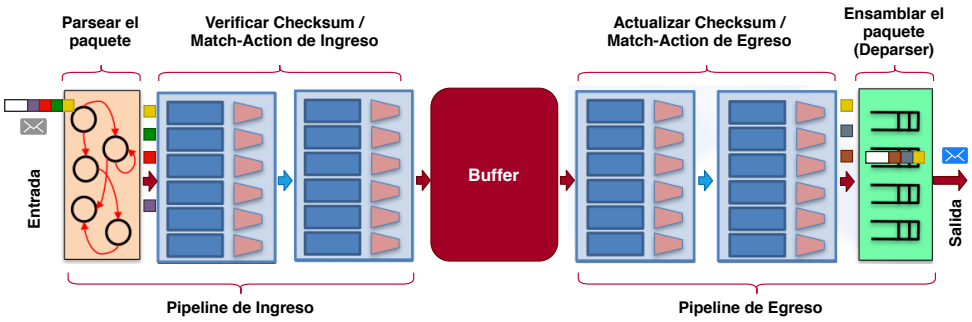


Figura 2.3: Arquitectura V1model [22]

continuación junto con los objetos externos que puede poseer un *target*.

2.2.1.1. Cabeceras y Metadatos

P4 es un lenguaje agnóstico a los diferentes protocolos de red, es decir P4 no puede reconocer y extraer por si solo los campos de la cabecera de cierto protocolo ya que este solo ve el paquete como una cadena de bits sin ningún formato (plano físico). De acuerdo a esto, es el programador quien debe definir el formato a esa cadena de bits entrante. Para llevar a cabo este proceso, el primer paso es declarar al inicio de la aplicación p4, las cabeceras de los posibles protocolos con los que interactuará la aplicación P4. El formato de dichas cabeceras a definir es totalmente libre, siendo ésta una de las principales características de P4, ya que permite definir tanto protocolos estándar (Ethernet, IPv4, ARP, UDP, etc...) tal cual como lo indica la norma, como también protocolos personalizados.

Una cabecera en P4 se define como una colección de variables ordenadas que contienen el valor de cada uno de los campos de dicha cabecera. Para especificar esta cabecera se debe anteponer la palabra clave *header* al nombre que se le quiera dar, y a continuación definir cada campo con el tipo de dato, tamaño y nombre, en orden de aparición en el paquete. A continuación, como ejemplo, se muestra la forma de definir la cabecera Ethernet:

```
header ethernet_t {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}
```

Por otra parte, los metadatos son variables que almacenan información relacionada con el procesamiento del paquete durante su paso por el conmutador. En ellos se puede almacenar información alternativa a la disponible en las cabeceras. Existen dos tipos de metadatos:

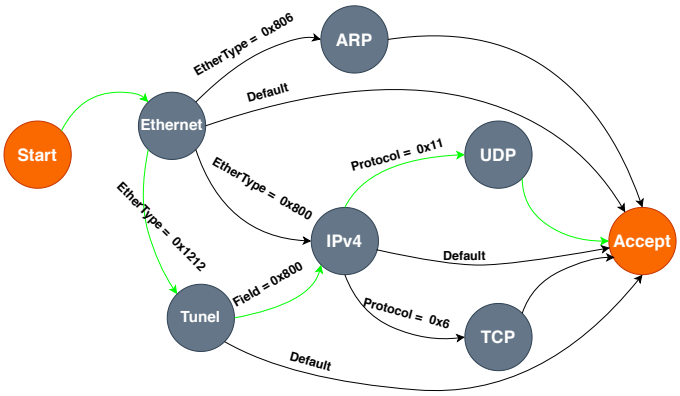


Figura 2.4: Diagrama de estado de un proceso de *Parser* programado.

los proporcionados por el fabricante, y los definidos por el programador. Los metadatos del fabricante contienen información intrínseca del *target* como el puerto de entrada y la estampa de tiempo de llegada del paquete, o información para que sea consumida por el *target* como el puerto de salida del paquete.

2.2.1.2. Parser

Un *Parser* es un bloque programable con el que se le da un formato a la cadena de bits del paquete. Esta cadena es dividida y almacenada en las variables (campos) de las cabeceras definidas previamente. Dicha división se define por medio de una máquina de estados con estado inicial "start" y estado final "accept"; esta máquina representa todas las posibles combinaciones de cabeceras que posee el paquete en el respectivo orden de aparición.

En la Figura 2.4 se puede observar un proceso de *Parser* con las cabeceras ya definidas Ethernet, ARP, IPv4, UDP, TCP, mas una cabecera personalizada (Túnel), que encapsula información del paquete. La figura representa las posibles transiciones entre cabeceras (estados) con base en una condición a cumplir. La condición usada para pasar de la cabecera Ethernet a otra es que se tenga el valor esperado en el campo EtherType de un protocolo de planos superiores. Igualmente para pasar de la cabecera IPv4 se debe cumplir la condición del campo Protocol (para identificar el protocolo de capa de transporte del modelo TCP/IP). Si ninguna de las transiciones definidas se cumple, ya sea porque el valor de la condición hace referencia a otro protocolo no definido o es un paquete corrupto, el paquete pasa inmediatamente al estado final. De acuerdo a este funcionamiento, a medida que el paquete va pasando por cada estado, el valor de los campos de las respectivas cabeceras son extraídos y asignados a las variables cabecera. La cantidad límite de estados y transiciones es definida por la arquitectura del *target*.

Como ya se dijo en el apartado anterior, la capacidad de P4 de definir cabeceras arbitrarias permite diseñar un Parser que procese paquetes con cabeceras no estándar y en diferente orden. Además, se puede observar que efectivamente se cumple el segundo objetivo de P4 de ser independiente de cualquier protocolo al definir la máquina de estados con base en los protocolos de interés anteriormente identificados.

2.2.1.3. Deparser

El *Deparser* es aquel que realiza lo opuesto al *Parser*, es decir, se encarga de ensamblar de nuevo las cabeceras, como también adicionar nuevas o eliminar existentes. Estas cabeceras son incluidas al comienzo de la carga útil que es considerada por P4 como la porción de bits del paquete que no fueron procesados en el *Parser*.

2.2.1.4. Bloques de Control

Los bloques de control son aquellos donde se lleva a cabo el procesamiento de las cabeceras del paquete, usando principalmente las tablas *match-action*. Estas tablas son estructuras de datos implementadas en memorias TCAM [23], las cuales mapean valores de campos de cabecera y metadatos *(match-key)* en la ejecución de una acción. Cada entrada de la tabla posee además de la *match-key*, el nombre de la acción (método P4) a ejecutar y los parámetros con los que procesa los campos de cabeceras y metadatos. Una entrada puede ser añadida, modificada o eliminada en tiempo real desde un plano de control.

Por otro lado, es en los bloques de control donde se pueden implementar sentencias condicionales if... else, usar operadores lógicos, realizar operaciones binarias elementales (suma, resta y multiplicación) y manejar cadenas de bits. A pesar de ser el bloque donde el programador tiene la libertad para programar e implementar su forma de procesar la información de los paquetes, existen varias restricciones que acotan el alcance de la aplicación. Entre ellas las mas relevantes son la ausencia de ciclos For y While, como también la imposibilidad de realizar divisiones.

2.2.2. Objetos Externos

Los objetos externos son funcionalidades propias del *target*, es decir, son implementadas fuera del lenguaje P4 como se observa en la Figura 2.5. Para hacer uso de esas funcionalidades, P4 proporciona los objetos externos como medio de interacción entre el lenguaje y el objeto. Muchas de las entidades P4 definidas en la versión $P4_{14}$ se han eliminado del lenguaje

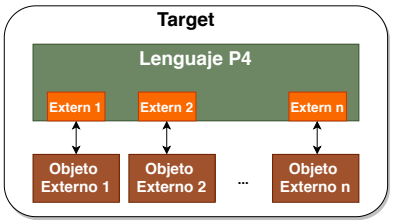


Figura 2.5: Objetos externos de un Target.

y se han modelado como objeto externo en $P4_{16}$, a la espera de que en un futuro sean estandarizadas en todos los *targets* y no haya una dependencia intrínseca con el hardware del proovedor. Estos objetos externos solo se pueden invocar en los bloques de control de ingreso y de egreso. A continuación introduciremos los registros y las funciones hash ya que son los objetos externos en los que se basó la implementación P4 que se presentará en el Capitulo IV de esta monografía.

2.2.2.1. Registros

Los registros son memorias que almacenan estados cuyos valores se pueden leer y escribir durante el procesamiento de cada paquete que pasa por el *target*. El uso de estos registros es muy útil ya que permite almacenar información relevante que se necesita que perdure a lo largo de la ejecución de la aplicación, a diferencia de los metadatos que son reiniciados cada vez que se procesa un nuevo paquete.

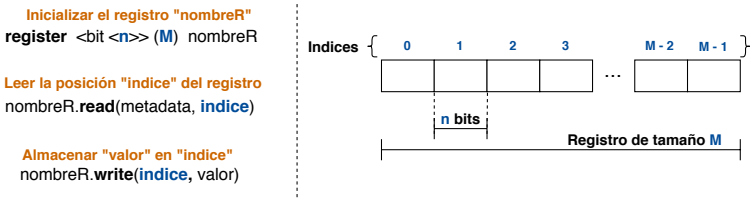


Figura 2.6: Manejo de Registros

Un registro lo podemos ver como un arreglo de tamaño fijo de M elementos, como se aprecia en la Figura 2.6. Cada elemento (celda) del registro es referenciado con un numero entero como índice, que es usado para leer o escribir un valor dentro de la respectiva celda. Estas celdas a la vez poseen un tamaño limite de n bits, valor que es definido en el momento de la inicialización del registro.

2.2.2.2. Hash

Es un objeto externo que permite calcular diferentes funciones *Hash* a cualquier tupla formada por valores de cabeceras y metadatos, que llamaremos llaves. Entre las diferentes funciones están aquellas basadas en algoritmos CRCs de 16 y 32 bits, como también las basadas en operaciones XOR de 16 bits. Estas funciones hash nos permiten mapear un conjunto N de llaves a un rango de salida finito (M) de valores enteros ($H : N \rightarrow M$), como se puede observar en la Figura 2.7.

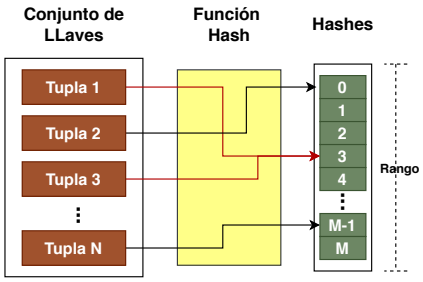


Figura 2.7: Funcionamiento de la función Hash

El principal uso que se le ha dado a estas funciones hash en las aplicaciones P4 es en la implementación de estructuras de datos eficientes en el proceso de búsqueda de elementos almacenados. En estos casos asocian el rango de salida de la función hash con los índices de un registro de tamaño M. De esta forma cada tupla correspondería al identificador de un elemento almacenado dentro de la estructura de datos. Un ejemplo de una tupla sería el conjunto de campos de cabeceras que identifican a un flujo de la red.

Por otro lado, las funciones hash no son perfectas, ya que pueden generar colisiones, es decir, se asigna el mismo hash a más de una llave diferente. Para garantizar una probabilidad de colisión baja, el rango de salida de la función hash debe ser mucho mayor al conjunto de de llaves. Dicha probabilidad de colisión (C) se puede estimar con base al conjunto de llaves (N) y el rango de salida (M) como lo expresa la ecuación 2.1 [24].

$$C = 1 - \left(1 - \frac{1}{M}\right)^N \tag{2.1}$$

2.2.3. P4Runtime

Teniendo en cuenta la necesidad de estandarizar una API que permita al plano de control interactuar en tiempo de ejecución con las entidades y configuraciones del plano de datos

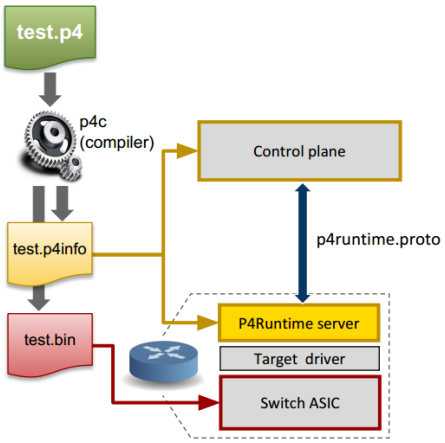


Figura 2.8: consolidación de P4Runtime [22]

basado en P4, surge como respuesta, la implementación de la API P4Runtime. Esta API deja a un lado la rigidez que posee OpenFlow para proporcionar un mecanismo flexible capaz de adaptarse a las implementaciones de cada aplicación P4 con indiferencia de las tablas, objetos externos o cabeceras que se definan. Con base en esto, P4Runtime brinda la capacidad de desplegar procesos de configuración en tiempo de ejecución como la adición o eliminación de entradas en las tablas P4, y la consulta de los valores de los contadores y las celdas de los registros. Por otro lado, una de las grandes cualidades de esta interfaz, es la de permitir instalar dinámicamente desde el plano de control las aplicaciones a ejecutar en el plano de datos, es decir, poder cargar dichas aplicaciones en los targets sin necesidad de reiniciar el sistema [25].

P4Runtime fue desarrollado y estandarizado en el 2017 por la API-Working-Group en colaboración con Google y Barefoot Networks [26]. Dentro del estándar, P4Runtime utiliza *protobuf* [27] como protocolo de serialización de datos, reconocido por generar mensajes de 3 a 10 veces mas livianos a velocidades 50 veces mayor con respecto a XML.

Por otro lado, dentro del mismo estándar, P4Runtime define gRPC [28] como el framework encargado de la comunicación entre ambos planos. gRPC es un sistema de llamada a procedimiento remoto (RPC) el cual esta diseñado para implementar microservicios (cliente - servidor) de una forma muy eficiente gracias al uso de HTTP/2 [29] y *protobuf* como lenguaje de interfaz.

En la conexión entre cada *target* y el plano de control, es el target el que se encarga de implementar el servidor P4Runtime, mientras que el plano de control sería el cliente P4Runtime de todos los dispositivos. Como P4Runtime se implementa como un microservicio gRPC, las

especificaciones de la aplicación p4 se definen en un fichero .proto, similar a como se implementa cualquier otro servicio gRPC. El formato exacto de este fichero se determina a partir del fichero P4Info el cual se obtiene al compilar la aplicación y que contiene la información que debe conocer P4Runtime sobre la aplicación P4 para luego interactuar con ella.

En la Figura 2.8 podemos apreciar el proceso de consolidación de P4Runtime entre un servidor (target) y un cliente (plano de control). Primero, la aplicación P4 es compilada obteniendo dos ficheros, la aplicación P4 compilada y el fichero P4Info. La representación compilada la consume el target para reconfigurarse internamente, mientras que el fichero P4Info lo consume tanto el servidor P4Runtime para implementar la API como el cliente para extraer la información necesaria para realizar configuraciones en las entidades de la aplicación en tiempo de ejecución.

2.2.4. Soporte de P4Runtime en el controlador ONOS

ONOS (Open Network Operating System) [30] es un controlador SDN de código abierto estructurado de forma modular e implementado en el lenguaje Java. Las principales características de ONOS son su escalabilidad, su alta disponibilidad, su rendimiento y sus modelos de abstracción para el desarrollo de aplicaciones [31].

ONOS fue el primer controlador SDN en soportar P4Runtime para instalar y gestionar aplicaciones P4 sobre un plano de datos programable. Sin embargo, cabe resaltar que a pesar que el soporte de P4Runtime es totalmente funcional, aun carece de algunas funcionalidades referente a los objetos externos como es el caso con los registros, que no se pueden gestionar aún desde el plano de control.

En la Figura 2.9 podemos observar como es la arquitectura del controlador ONOS referente al soporte de P4Runtime. En esta figura observamos que ONOS posee un *driver* P4Runtime que le permite conectarse como cliente a los diferentes dispositivos, además de intercambiar mensajes serializados. También, se aprecia que ONOS posee soporte para tres diferentes targets P4: BMv2, Barefoot Tofino y Mellanox Spectrum. Debido a que el formato en el que se expresa la aplicación P4 compilada varía de acuerdo al target, ONOS debe proporcionar un driver diferente para cada uno, el cual combine el proveedor genérico P4Runtime con la extensión específica precisada para interactuar con dicha aplicación. Por ejemplo, la extensión para el target BMv2 es en formato JSON y para Barefoot Tofino es en formato binario.

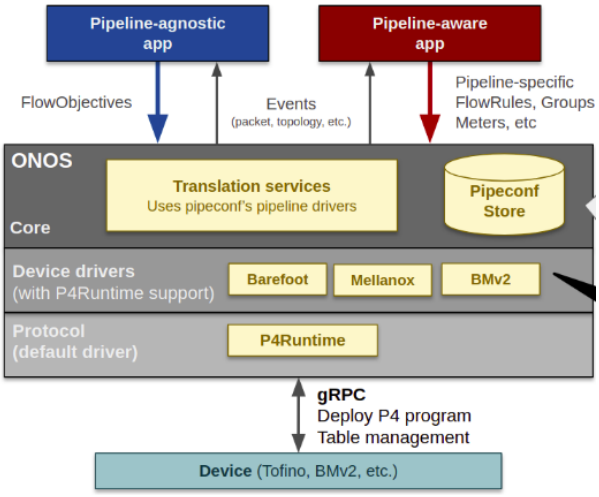


Figura 2.9: Arquitectura de ONOS con soporte P4Runtime [32]

Por otro lado, una de las principales contribuciones de ONOS es la disposición de una API mucho mas amigable para el programador en comparación a la proporcionada directamente por P4Runtime heredada de gRPC y Protobuf. Esta API se conoce como PI-Framework la cual es un conjunto de clases y servicios, que proporcionan el control de las entidades del plano de datos, actuando como una API envolvente sobre P4Runtime. PI entonces hace de puente entre P4Runtime y las aplicaciónes ONOS, que realiza un mapeo de la información de las entidades P4 (tablas, campos, acciones, cabeceras, etc...) contenidas en el fichero P4Info, a una representación que se acopla a la sintaxis de ONOS desplegando así tareas como la instanciación de entradas de flujo en las tablas [26].

2.3. Ataque de Denegación de Servicio Distribuido (DDoS)

El ataque DDoS es la forma mas avanzada del ataque DoS [33]. Este ataque consiste en el despliegue de ataques DoS desde muchas computadoras al mismo tiempo, de forma coordinada y distribuida a través de internet. Este ataque colaborativo permite multiplicar significativamente la efectividad del ataque, al aprovechar los recursos de dichas computadoras.

Hoy en día, los ataques DDoS son lanzados casi en su mayoría por una red de computadoras *Zombies* o *Botnet* controladas de forma remota, bien organizadas y ampliamente dispersas en Internet, todas ellas atacando al mismo objetivo. Estas computadoras zombies normalmente son reclutadas al momento de ser víctimas de un gusano, troyano o *backdoors* [34] sin que el propietario de dicha máquina se entere. De esta forma, el atacante puede hacer uso de los recursos de dicha máquina para potenciar el ataque de denegación de servicios.

Por otro lado, una gran ventaja que poseen estos ataques DDoS es la complejidad que conlleva el rastreo del origen de dicho ataque. Esta complejidad se debe a que el tráfico enviado hacia la víctima puede poseer direcciones IP falsificadas, además de ser enrutado pasando por diferentes proxys antes de llegar a su destino [35].

Los objetivos principales de este tipo de ataques son:

- Interrumpir la conectividad de un usuario legítimo agotando el ancho de banda, la capacidad de procesamiento del enrutador o los recursos de red [35]. Estos son esencialmente ataques de inundación DDoS a nivel de red/tranporte.
- Interrumpir los servicios prestados a un usuario legítimo agotando los recursos del servidor (por ejemplo, sockets, CPU y memoria) [35]. Estos son esencialmente ataques de inundación DDoS a nivel de aplicación.

2.3.1. Estrategia para desplegar ataques DDoS

La mayoría de los ataques de inundación DDoS más recientes y problemáticos han empleado redes de computadoras *zombies* o *botnets*. En la Figura 2.10 se observa la arquitectura empleada para desplegar un ataque basado en dichas redes. Los maestros, son el medio de comunicación que tiene el atacante con los diferentes zombies (agentes), los cuales eventualmente llevarán a cabo el ataque contra el sistema de la víctima. Estos maestros pueden ser programas instalados en un conjunto de dispositivos comprometidos (por ejemplo, servidores de red) con los que el atacante se comunica para enviar comandos que se ejecuten en los diferentes agentes. Estos comandos indican el inicio del ataque, la víctima, la duración del ataque y las características especiales como el tipo, longitud, el TTL, los números de puertos, entre otros.

Las botnets pueden tener cientos de formas diferentes de ser implementados. Según la forma en que los maestros controlan los agentes, las botnets se clasifican en dos categorías principales: basadas en IRC y basadas en web:

- **Basada en IRC [36]:** IRC es un protocolo de mensajería instantánea en línea basado en texto. Este protocolo usa una arquitectura cliente-servidor, con canales predeterminados para comunicarse entre servidores permitiendo conectar cientos de clientes a cualquiera de ellos. Al usar los canales IRC como maestros, el atacante puede usar puertos IRC para enviar comandos a los diferentes agentes. Las ventajas que trae este método es que un atacante puede ocultar fácilmente su presencia debido al gran

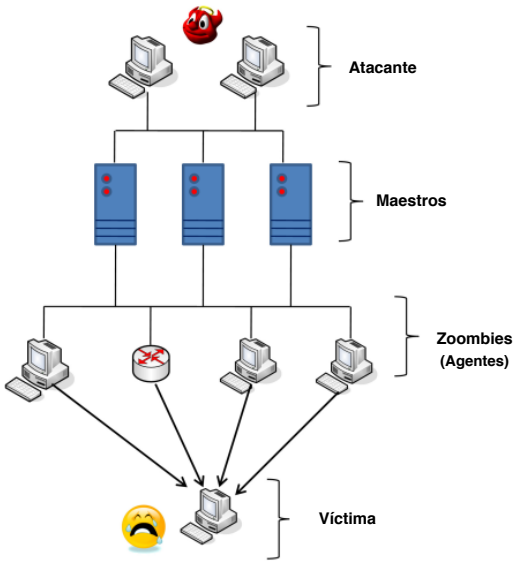


Figura 2.10: Arquitectura de una Botnet [35]

volumen de tráfico que suelen tener los servidores IRC, además de poder compartir fácilmente archivos para distribuir el código malicioso. Sin embargo, el ataque puede ser frustrado fácilmente si ocurre una falla en aquellos servidores IRC que se encuentran en puntos centralizados sensibles a fallas. Entre las herramientas mas conocidas para desplegar ataques DDoS basadas en maestros IRC está Trinity-v3 [37] y Kaiten [38].

- **Basada en web [39]:** Los ataques DDoS mas recientes han optado por usar el protocolo HTTP para enviar los comandos a los agentes, debido a que son mucho mas difíciles de rastrear. En estos casos, los agentes no necesitan mantener una conexión con un servidor centralizado como en el caso de los botnets basados en IRC. En cambio, cada agente descarga periódicamente las instrucciones mediante peticiones HTTP. Cabe resaltar que, los botnets basados en web son más sigilosos ya que se esconden dentro del tráfico HTTP legítimo. Adicionalmente, los agentes son configurados y controlados a través de programas en PHP utilizando canales cifrados a través de HTTPS (puerto 443). Una herramienta basada en web que es ampliamente utilizada es Low-Orbit Ion Cannon (LOIC) [40],

2.4. Modelos de Aprendizaje de Máquina para la detección de ataques de DDoS

Una de las áreas enfocadas a la detección de intrusiones de red, entre ellas los ataques DDoS, es la basada en la detección de anomalías en el tráfico de la red. Dentro de esta área se encuentran los métodos de aprendizaje de máquina (ML), considerados los mejores detectores en términos de métricas clave, como la velocidad, la precisión y la confiabilidad en el proceso de detección [41].

Los métodos de detección de DDoS basados en ML, son un conjunto de modelos entrenados con base en una base de datos formada con los descriptores o conjunto de características que son calculadas periódicamente con las estadísticas extraídas del tráfico de la red. Con dichas características el modelo es capaz de aprender a clasificar el comportamiento del tráfico en dos clases diferentes: tráfico *legítimo o de DDoS*. Ya con dicho modelo entrenado, en la fase de prueba o en producción, se puede clasificar nuevo tráfico con base en las nuevas características extraídas.

Existe un gran numero de características definidas en la literatura, que pueden reducirse de forma que se escoja un subconjunto que permita entrenar el modelo de clasificación mas exacto y generalizado. Según [42], las características esenciales para detectar ataques DDoS se pueden clasificar como sigue:

- **Características a nivel de Paquete:** En este grupo se encuentran las características que se pueden calcular con la información obtenida a nivel de paquete como la media, varianza o raiz cuadrada media (RMS) de los paquetes. Estas características son independientes de la noción de flujos, conexiones u otras agregaciones de nivel superior.
- **Características a nivel de Flujo:** Son características calculadas con base en las estadísticas agregadas a nivel de flujo: duración del flujo, volumen de datos y paquetes totales del flujo. Entre estas características están la duración media del flujo, el volumen medio de datos por flujo, el número medio de paquetes por flujo y la varianza de estas métricas, entre otras.
- **Características a nivel de Conexión:** Estas características son calculadas con base en las estadísticas de nivel de transporte orientadas a la conexión, como las conexiones TCP. Esto obliga a la necesidad de rastrear el estado de la conexión para obtener información como simetría de la conexión, los tamaños de ventana anunciados y la

distribución de *troughput*. Este tipo de información de nivel de conexión generalmente proporcionan datos con mayor granularidad y calidad que la información a nivel de flujo, pero requieren una sobrecarga adicional.

- **Características a nivel de Intra-Flujo:** Son características basadas en la noción de flujo o conexión TCP, pero que requieren estadísticas sobre los paquetes dentro de cada flujo para su cálculo. Un ejemplo claro son las estadísticas de los tiempos de de llegada entre los paquetes del mismo flujo (*Flow-IAT*). Esto requiere datos recogidos a nivel de paquete, pero luego agrupados en flujos para su cálculo.
- **Características a nivel de flujos múltiples:** Son características que se calculan con base a las estadísticas de múltiples flujos/conexiones. Por ejemplo, muchas aplicaciones *peer-to-peer* logran la descarga de un archivo grande por masivas descargas de trozos más pequeños ubicados en múltiples máquinas.

con base en lo anterior, es importante definir los diferentes tipos de flujos que pueden ser considerados según [43], a la hora de extraer estadísticas para dicho propósito:

- **Flujo unidireccional:** Es aquel comprendido por una serie de paquetes que comparten los mismos campos: direcciones IP de origen y destino, puertos UDP/TCP de origen y destino y número de protocolo de transporte.
- **Flujo Bidireccional:** Es aquel conformado por un par de flujos unidireccionales que van en direcciones opuestas entre las mismas direcciones IP y puertos de origen y destino.
- **Flujo-Completo:** Es un flujo Bidireccional capturado durante toda su vida útil, desde el establecimiento hasta el final de la conexión.

Capítulo 3

Trabajos Previos

La posibilidad de desarrollar aplicaciones de red en el plano de control de una arquitectura SDN, y poder tener acceso a la información de estado de la red por medio de OpenFlow, permitieron que los modelos de ML [43] para la clasificación de tráfico en busca de ataques DDoS sean implementados en dicha arquitectura para poder realizar detecciones de comportamientos anómalos en tiempo real.

Un proceso crucial que permitió dichas implementaciones fue la posibilidad de adquirir periódicamente estadísticas y demás información vital para el cálculo de diferentes características que describan el comportamiento del tráfico. Con base en las diferentes implementaciones evidenciadas en el estado del arte, el proceso de recolección de información usando las funcionalidades de SDN se ha llevado a cabo de 3 formas generalmente:

- El controlador solicita periódicamente los contadores en bytes y paquetes disponibles por OpenFlow ya sea por flujo (Tabla 2.1), tabla o puerto físico. Esta información permite el cálculo de algunas características a nivel de paquete y de flujo.
- Usando el método de muestreo de paquetes ofrecido por el estándar sFlow [44], en el cual cada dispositivo del plano de datos actúa como un agente que recolecta y reporta estadísticas al colector-sFlow ubicado en el plano de control. Luego, el colector-sFlow y el controlador SDN interactúan usando la interfaz norte para el calculo y uso de características de tráfico [45][46].
- El controlador extrae de los *packet_in* los valores de los campos de cabecera del paquete con el que se construyó dicho *packet_in*. Entre estos campos están las direcciones IP y

puertos de origen y destino. Por otro lado, con un procesamiento adicional es posible contar el numero de conexiones TCP que se establecen.

A continuación se presentarán las principales propuestas de implementación de sistemas de detección de ataques DDoS basados en modelos de ML y OpenFlow. En estas propuestas se detallará la arquitectura implementada, las características y los modelos de ML usados en la detección de tráfico.

En [9], los autores implementaron un sistema que detecta varias intrusiones, entre ellas los ataques de DoS/DDoS, por medio del monitoreo y la clasificación del tráfico de la red. Este sistema fue diseñado para que el plano de control solicite con una periodicidad de 1 segundo, los contadores OpenFlow asociados a cada flujo unidireccional (Tabla 2.1) que esté activo en cada dispositivo. Utilizando el identificador de dichos flujos (IP y puerto de origen y destino, y protocolo), se asocian los contadores de los flujos opuestos para formar flujos bidireccionales y así almacenarlos temporalmente en el plano de control. Este proceso de petición, asociación y almacenado se realiza en varias repeticiones antes de extraer las características. El motivo es almacenar varias medidas del mismo flujo y poder calcular el conjunto de características que se forma por el valor de desviación estándar (std) de cada contador. Después de poseer dicho conjunto de características por cada flujo, se realiza una clasificación del flujo usando alguno de los modelos de la Tabla 3.1 en busca de posibles ataques. En esta misma tabla se muestra el porcentaje de eficiencia en detección obtenido por los autores en cada modelo.

Tabla 3.1: Resultados de detección en [9]

Modelo	**Eficiencia en Detección**
Máquinas de Soporte Vectorial (SVM)	85 %
Arboles de decisión (DT)	89 %
Bagged	96 %
Bosques aleatorios (RF)	96 %
K vecinos mas cercanos (KNN)	94 %

En [47], los autores implementaron un sistema detección de DDoS usando redes neuronales y mapas auto-organizados (SOM) como modelo de clasificación. Al igual que la anterior implementación, el mecanismo desarrollado en el plano de control esta constituido por un colector de flujos, un extractor de características y un clasificador, como se aprecia en la Figura 3.1. El sistema realiza ciclos de detección cada 3 segundos. En cada ciclo, el

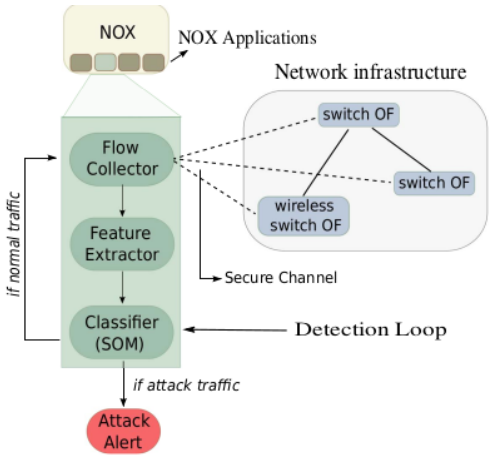


Figura 3.1: Operación de detección implementado por [47]

colector adquiere los contadores por flujo de cada dispositivo, para que luego el extractor de características genere una tupla con 6 de ellas. El conjunto de 6 características está formado por: la media de paquetes y bytes por flujo, la duración media por flujo, el porcentaje de flujos bidireccionales, y la cantidad de puertos y flujos individuales en el intervalo examinado. Para llevar a cabo el calculo de estas características, los autores especificaron procesos de ordenamiento ascendente por valor, y la búsqueda y conteo de flujos bidireccionales. Ya en el ultimo paso del ciclo, la clasificación por flujos, se obtienen eficiencias de detección del orden del 98 %.

De igual manera, en [48] implementaron otro sistema de detección de DDoS que utiliza un modelo de aprendizaje profundo (Deep-Learning) basado en el enfoque de Auto-encoders (SAE). Este modelo clasifica el tráfico ya sea en la clase *tráfico normal* o en alguna de las 7 clases de ataque de DDoS consideradas por los autores. Para alimentar el modelo de clasificación, el sistema almacena los campos de cabecera de todos los *packet_in* en el plano de control, con el fin de extraer luego un conjunto de características diferente por cada tipo de protocolo (UDP: 14, TCP: 20 o ICMP: 14 características). Los diferentes conjuntos de características son conformados principalmente por valores de entropía, contadores y fracciones sobre dichos campos de cabecera, por cada flujo. A pesar que el sistema de detección registró una eficiencia del 95 %, presenta una gran carga para el plano de control extraer periódicamente conjuntos de características tan grandes.

Por otro lado, en [10] los autores implementaron un sistema de detección y mitigación de ataques DDoS, pensado en la reducción del consumo de recursos del plano de control al desplegar dichas funcionalidades. El mecanismo de detección se conformó por dos fases: una

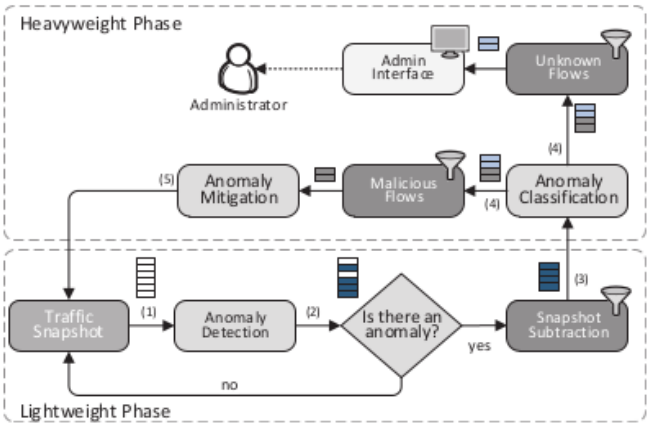


Figura 3.2: Systema de detección de dos fases, implementado por [10]

fase liviana y una fase pesada con respecto a la exigencia de los procesos a ejecutar (Ver Figura 3.2). En la fase liviana, usando los contadores por flujo y demás información recopilada, los autores implementaron un análisis de entropía (detector liviano de anomalías) para detectar variaciones en la distribución de ciertas características de flujo. Dicho resultado de entropía, calculado por cada flujo, se compara con el obtenido en la medición previa. Luego, con base en unos umbrales, se catalogan dichos flujos como sospechosos (gran diferencia en entropías) o normales (poca diferencia). Esta fase liviana funciona como un filtro en la que solo los flujos sospechosos pasaban a ser analizados en la fase pesada, reduciendo así, la cantidad de procesos a ejecutar.

La fase pesada consistió en la clasificación de flujos sospechosos usando un modelo entrenado con SVM. En esta fase se extrae una tupla de 3 características por cada flujo: número total de paquetes, de bytes y la duración del flujo. Con este simple pero óptimo conjunto de características, los autores obtuvieron una eficiencia en detección del 88 %. Sin embargo, aunque el sistema evita sobrecargar el plano de control, la eficiencia en detección es castigada.

Al igual que en la anterior propuesta, en [49] los autores implementaron otro sistema de detección y mitigación DDoS que usa tanto la entropía como modelos de clasificación de ML (Ver Figura 3.3). El sistema constantemente calcula la entropía de las direcciones IP en búsqueda de una anomalía. En el momento de ocurrir, el sistema empieza a extraer características y realizar la clasificación usando un modelo de SVM. En este trabajo los autores usaron 9 características a nivel de conexión TCP que demandan, por parte del plano de control, un conteo constante de conexiones TCP por servicio, puertos y dispositivos de origen y destino.

En [50] implementaron un sistema de detección de anomalías usando el clasificador: red neuronal con *Backpropagation*. Este clasificador es alimentado periódicamente con 7 características basadas en contadores de paquetes, bytes y conexiones que son recolectados de las estadísticas de OpenFlow y de los *packet_in*. En esta implementación obtuvieron eficiencia de detección del 97 %.

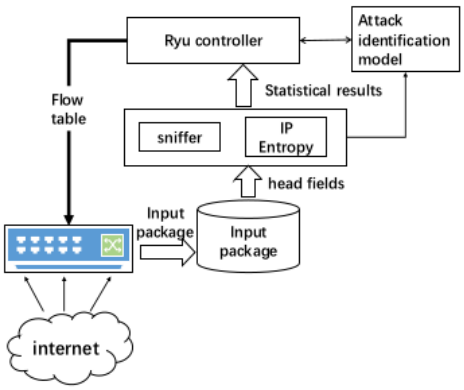


Figura 3.3: Systema de detección de dos fases, implementado por [49]

De igual manera, en [51] implementaron otro sistema de detección que extrae 14 características basadas en conteo de banderas TCP, duración de la conexión TCP y conteo de paquetes y bytes entre todo par origen-destino en ambas direcciones. Para este propósito, y al igual que en los anteriores trabajos, los autores implementaron una gran cantidad de contadores en el plano de control que fueron almacenados en estructuras de datos poco óptimas que pueden sobrecargar dicho plano. El clasificador usado fue C4.5.

Por otro lado, el sistema de detección implementado en [52] posee en el plano de control una estructura de datos basada en mapas *hash* para almacenar y acceder de forma óptima a la información colectada de cada flujo. La información colectada, que a la vez son las características usadas, fueron la dirección de origen y destino, el puerto origen y destino, el total de paquetes y bytes, y la duración del flujo. Los autores usaron el clasificador K-vecinos mas cercanos para la detección de intrusiones. Aunque el mecanismo propuesto en el plano de control reduce la sobrecarga en el controlador, el conjunto de características usado no fueron tan exitosos pues los resultados en eficiencia de detección son del 77 %.

En la Tabla 3.2 se resume los detalles de interés de los diferentes sistemas de detección introducidos previamente:

Tabla 3.2: Resumen de trabajos previos.

Ref	**# de características extraídas**	**Tipo de características**	**Procesos (Tareas) desplegados por el plano de control**	**Eficiencia en detección**
[9]	9	A nivel de flujo	Almacenar información de flujos. Encontrar flujos bidireccionales. Calcular características. Clasificar.	96%
[47]	6	A nivel de flujo A nivel de paquete	Formar flujos bidireccionales. Ordenar flujos por valor. Calcular características. Clasificar.	98%
[48]	TCP ->34 UDP ->20	A nivel de flujo A nivel de paquete	Almacenar campos de cabecera por *packet_in*. Encontrar flujos bidireccionales. Calcular características. Clasificar.	95%
[10]	3	A nivel de flujo	Calculo de entropía por flujo. Clasificar.	88%
[49]	9	A nivel de conexión (solo conteo)	Contadores por servicio y host. Calcular características. Clasificar.	99%
[50]	7	A nivel de conexión (solo conteo)	Contadores por servicio y host. Calcular características. Clasificar.	97%
[51]	14	A nivel de conexión A nivel de flujo	Contadores por servicio y origen-destino. Clasificar.	96%
[52]	10	A nivel de flujo	Almacenar información de flujos. Calcular características. Clasificar.	77%

Analizando las diferentes formas de implementar el mecanismo detección en la arquitectura SDN, se identificó dos limitantes que dificultan la implementación del mecanismo ideal. La primera consiste en la limitada información del tráfico que se puede extraer en tiempo de ejecución [53]. Esta limitante se refleja en la imposibilidad de poder calcular periódicamente

ciertas características que podrían ser los mejores descriptores del tráfico al ser usadas en ciertos modelos de ML. Por otro lado, la segunda limitante se genera en parte por la dificultad de obtener dicha información con el suficiente nivel de granularidad. Para obtener esta información se ha evidenciado múltiples procesos que debe desplegar el plano de control para la recolección, procesamiento, cálculo y clasificación de dichas características. Esta gran cantidad de procesos que se despliegan repetitivamente pueden sobrecargar el controlador en presencia de gran cantidad de flujos de tráfico.

Esta última limitante ya ha sido identificada previamente en otros trabajos. Un claro ejemplo de esto es el trabajo realizado en [54] donde los autores deciden realizar la detección de los ataques de DDoS directamente en el plano de datos para evitar así sobrecargar el plano de control. La solución propuesta consistió en usar P4 para calcular la entropía de las direcciones IP de origen y destino de los diferentes paquetes de cada flujo. Luego, con base a un umbral ajustado previamente en la fase de aprendizaje, el mecanismo toma la decisión si el flujo posee un comportamiento semejante al de DDoS o no. Aunque este método muestra excelentes resultados, solo analiza una única anomalía al realizar la detección. En un escenario de producción podrían surgir altos falsos positivos cuando el comportamiento legítimo de los usuarios del servicio tenga una alta aleatoriedad. Sin embargo, aunque esta solución no emplea modelos de clasificación de ML por las limitadas primitivas de P4, la forma de abordar la implementación influyó bastante en la solución propuesta en el capitulo 4 de esta monografía.

Capítulo 4

Planteamiento de la Solución

4.1. Motivación

De acuerdo a los diferentes mecanismos de detección de DDoS presentados en la sección de trabajos previos (Capitulo 3), podemos decir que SDN ha aportado funcionalidades muy útiles que han permitido el rápido despliegue de novedosos sistemas de seguridad para las redes de datos. Gracias a la posibilidad de obtener en tiempo real información relevante del tráfico de la red, fue posible que se implementaran en entornos de producción, modelos de clasificación de ML que sean capaces de generar una alerta en presencia de un ataque de DDoS.

Sin embargo, aunque estos sistemas de detección son efectivos llevando a cabo dicha tarea (última columna Tabla 2.1), existen dos limitantes que dificultan la construcción y el despliegue de los sistemas:

- **Limitante 1:** La información que proporcionan los contadores de OpenFlow y la que se puede extraer de los *packet_in* solo permiten calcular algunas características del conjunto grande de posibilidades propuestas en la literatura [55][56]. En la Tabla 3.2 podemos observar que en los trabajos previos optan por usar características a nivel de flujo, paquete y unas pocas a nivel de conexión, ya que son las únicas posibles de obtener usando SDN tradicional (Openflow).

  Si se quisiera implementar un sistema que utilice características a nivel de Intra-Flujo sería imposible debido a que los contadores de OpenFlow no proporcionan información con la suficiente granularidad requerida. Como ejemplo de estas características se

encuentra la media o la desviación estándar del tiempo de llegada entre paquetes del mismo flujo (Flow-IAT). Para calcular dichas características se necesitaría conocer cada una de los tiempo de diferencia de llegada entre paquetes consecutivos, información que no es obtenible (ver Figura 4.1).

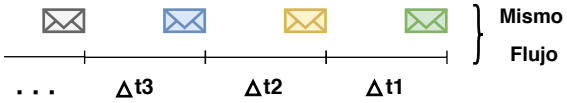


Figura 4.1: Tiempo de diferencia de llegada entre paquetes consecutivos del mismo flujo.

De igual manera, si se quisiera rastrear los diferentes valores de ventana TCP que se comparten dentro de un flujo y así calcular algunas características a nivel de conexión, tampoco sería posible debido a que solo el primer paquete del flujo es enviado al controlador y los demás son reenviados a su destino gracias a la regla de flujo preestablecida (ver Figura 4.2). De esta manera, solo se conocería el valor de ventana TCP inicial de la conexión.

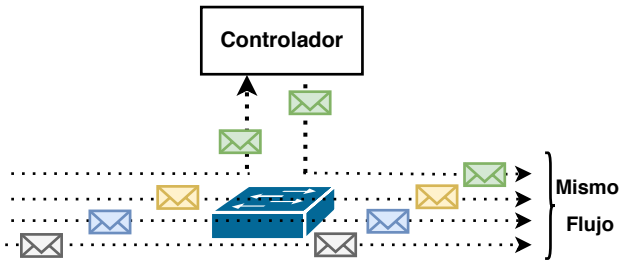


Figura 4.2: Paquetes del mismo flujo pasando por un conmutador.

- **Limitante 2:** Las diferentes tareas que se llevan a cabo periódicamente en el proceso de detección, pueden sobrecargar el plano de control en presencia de grandes cantidades de tráfico a considerar en la extracción y clasificación de características. Esto puede incrementar considerablemente la latencia en el plano de control y afectar el procesamiento de las solicitudes de los dispositivos de la red. Estos procesos se pueden apreciar en la Tabla 3.2; y van desde el almacenamiento y procesamiento de información, hasta el cálculo y clasificación de características.

De acuerdo a estas dos limitantes, nació el interés por buscar mecanismos nuevos de extracción de información de tráfico con el suficiente nivel de granularidad requerido, reduciendo a la vez la sobrecarga en el plano de control. De acuerdo a esto, contar con una plataforma

que brinde estas ventajas facilitaría la búsqueda de un mejor modelo de clasificación en eficiencia y generalización, al poseer mas opciones factibles de implementar en un entorno SDN.

El uso de P4 como soporte de programabilidad en el plano de datos, es un potente mecanismo para lograr dichos objetivos por las siguientes razones:

- Es posible extraer información de flujos, paquetes y conexiones con un buen nivel de detalle al poder acceder a toda la información que lleva cada paquete que pase por los dispositivos de la red.
- Al tener la libertad de programar la forma como se procesan los paquetes, además de poder disponer de varias primitivas que proporciona P4, es posible enviar la información extraída ya pre-procesada al plano de control con el fin de disminuir el número de procesos a realizar antes de la clasificación.

En vista a las prometedoras funcionalidades ofrecidas por P4, junto con las ya proporcionadas por la arquitectura SDN, en esta monografía se propone, hasta donde alcanza nuestro conocimiento, el primer sistema de detección de ataques de DDoS que emplea un entorno colaborativo entre el plano de datos y el plano de control. Este sistema es capaz de calcular en tiempo real, un conjunto de características formado tanto por descriptores a nivel de flujo como a nivel Intra-flujo, con una buena eficiencia y bajo procesamiento en el plano de control.

4.2. Revisión Conceptual del Mecanismo Propuesto

El mecanismo que se presentará a continuación está orientado a explicar principalmente el proceso que se lleva a cabo para calcular el conjunto de características especificado en la Tabla 4.1. Este conjunto de 4 características es recomendado por los autores de la base de datos CICIDS-2017 [11], y fueron seleccionadas entre 80 posibles usando el método *RandomForestRegressor*. Este método les permitió encontrar el menor conjunto con los mejores descriptores del ataque de DDoS. Como se observa en la Tabla 4.1 este conjunto está conformado por dos características a nivel de flujo (F1 y F3) y dos a nivel Intra-Flujo (F2 y F4). Por otro lado, se tienen características que se calculan considerando las estadísticas en ambas direcciones del flujo (bidireccionales), como una sola de ellas (F4) que se calcula considerando solamente las estadísticas recolectadas en la dirección de retorno (Bwd) del mismo flujo.

Tabla 4.1: Conjunto de Características Usado

#	Nombre de la Característica	Descripción	Tipo	Nivel
F1	FlowDuration	Duración del flujo en microsegundos	Bidireccional	Flujo
F2	Flow-IAT Std	Desviación estándar del tiempo de llegada entre paquetes consecutivos del mismo flujo.	Bidireccional	Intra-Flujo
F3	avgPacketSize	Tamaño promedio de la carga útil de los paquetes del mismo flujo.	Bidireccional	Flujo
F4	Bwd Packet Length Std	Desviación estándar del tamaño de la carga útil de los paquetes del mismo flujo, en dirección de retorno.	Unidireccional	Intra-Flujo

De acuerdo a este conjunto características, F2, F3 y F4 se obtienen calculando la media y la desviación estándar con las ecuaciones especificadas en 4.1 y 4.2 respectivamente. En estas ecuaciones, x_i corresponde a una unidad de información extraída del paquete i el cual hace parte del conjunto de n paquetes pertenecientes al mismo flujo. El significado de x_i cambia según la característica a calcular. Para calcular F2, x_i corresponde a la diferencia de tiempo de llegada entre el paquete actual y el anterior del mismo flujo. Igualmente, para F3 y F4 x_i corresponde al tamaño en bytes de la carga útil del paquete i.

Las características a nivel de flujo (F1 y F3) pueden ser obtenidas en un entorno SDN-OpenFlow con las estadísticas por flujo que proporciona esta arquitectura. En un entorno SDN-P4 tampoco sería problema obtenerlas si se programan algunos contadores y se almacena dicha información en registros.

El verdadero desafío (una de las contribuciones de este trabajo) es calcular las características a nivel Intra-Flujo (F2 y F3) que necesitan que el plano de control conozca todos los valores x_i para poder calcular la desviación estándar dentro del flujo. Estos valores individuales son necesarios ya que a cada x_i se le debe restar la media para luego elevarlo al cuadrado (ver Ecuación 4.2, lado izquierdo), información que no es proporcionada por OpenFlow. Si se programa el plano de datos si sería posible enviar al plano de control cada x_i recolectado pero esto no es viable ya que existiría un flujo de envío constante muy grande, lo que resultaría en altas latencias y, aún más crítico, un alto consumo de recursos no deseado.

Por otro lado, una solución más ambiciosa podría ser calcular dichas características Intra-Flujo directamente en el plano de datos. Sin embargo, esto aún no es posible con las limitadas primitivas que ofrece actualmente P4, siendo mas específico, no es posible realizar divisiones o calcular raíces cuadradas que son necesarias para la media y la desviación estándar.

$$\overline{x} = \frac{\sum_{i:1}^{n} x_i}{n} \tag{4.1}$$

$$std = \sqrt{\frac{\sum_{i:1}^{n} \left(x_i - \overline{x}\right)^2}{n-1}} = \sqrt{\frac{\sum_{i:1}^{n} x_i^2 - 2\overline{x}\sum_{i:1}^{n} x_i + n\overline{x}^2}{n-1}} \tag{4.2}$$

En vista a lo expuesto anteriormente, se propone descomponer la tradicional Ecuación 4.2 usando propiedades aritméticas, para llegar a la representación del lado derecho en dicha ecuación. Ahora, esta nueva representación depende de dos sumatorios: la agregación de los valores individuales ($\sum x_i$), y la agregación de los valores individuales al cuadrado $\left(\sum x_i^2\right)$. Ya con esta nueva representación, podemos programar un mecanismo en el plano de datos capaz de calcular el resultado de ambos sumatorios por cada flujo, a medida que los paquetes en curso pasan por el dispositivo. De acuerdo a esto, cuando los resultados de agregación de cada flujo se reportan al plano de control junto con los contadores de paquetes y bytes, el controlador solo tendría que reemplazar estos valores en ambas ecuaciones y calcular la media y la desviación estándar de manera fácil y rápida.

El objetivo de este mecanismo radica en evitar enviar todos los valores individuales para el cálculo de las características Intra-Flujo; lo que se propone es enviar agregados (información procesada) para dichos cálculos. Por otro lado, esta propuesta busca que el plano de control solo se dedique a calcular las características y a realizar la clasificación de las mismas, esto con el fin de ganar eficiencia en el proceso de detección y así evitar su sobrecarga. Los diferentes procesos previos al cálculo de las características, presentados en la Tabla 3.2, son delegados al plano de datos. De acuerdo a esto, el plano de datos debe encargarse de almacenar las diferentes estadísticas por flujo en ambas direcciones, actualizarlas en tiempo real, y en el momento de reportarlas al plano de control, enviar la información recolectada de ambas direcciones como un solo flujo.

4.2.1. Cálculo de las Características

Con base en la estrategia planteada, las estadísticas precisadas por el plano de control para calcular las características de un flujo, se muestran en la tabla 4.2. En esta tabla se observan los contadores de paquetes y bytes en ambas direcciones del flujo (S2 y S3). Adicionalmente, encontramos los agregados de IAT (S7 y S8) para calcular F2, y los agregados de tamaño de carga útil (S4, S5 y S6) para calcular F3 y F4. Por otro lado, F1 es calculado directamente en el plano de datos y enviada junto con las estadísticas. El cálculo de F1 es relativamente sencillo

Tabla 4.2: Información reportada al plano de control por cada flujo.

#	Característica / Estadistica	Descripción	Peso
F1	FlowDuration	Duración del flujo en microsegundos	6 Bytes
S2	Tot Fwd Pkt	Numero total de paquetes en la dirección Fwd	4 Bytes
S3	Tot Bwd pkt	Numero total de paquetes en la dirección Bwd	4 Bytes
S4	Tot Len Fwd Pkt	Agregado de tamaños de carga útil de los paquetes en dirección Fwd ($\sum x_i$)	4 Bytes
S5	Tot Len Bwd Pkt	Agregado de tamaños de carga útil de los paquetes en dirección Bwd ($\sum x_i$)	4 Bytes
S6	Tot Len Bwd Pkt Sqrt	Agregado de tamaños al cuadrado de carga útil de los paquetes en dirección Bwd $\left(\sum x_i^2\right)$	5 Bytes
S7	IAT Total	Agregado de tiempos de llegada entre paquetes ($\sum x_i$)	6 Bytes
S8	IAT Total Sqrt	Agregado de tiempos al cuadrado de llegada entre paquetes $\left(\sum x_i^2\right)$	7 Bytes

en P4 gracias a que es posible almacenar la marca de tiempo del primer y último paquete del flujo, como también hacer la resta de ambos.

Cuando llegan las estadísticas de un flujo al controlador, este inmediatamente procede a realizar el cálculo de las características sin hacer ningún proceso adicional, simplemente reemplazando los valores en las ecuaciones 4.1 y 4.2, como se especifica a continuación:

- **F2:**

$$n = S2 + S3 - 1 \quad (Bidireccional) \tag{4.3}$$

$$\overline{x} = \frac{S7}{n} \tag{4.4}$$

$$std = \sqrt{\frac{S8 - 2 * \overline{x} * S7 + n\overline{x}^2}{n-1}} \tag{4.5}$$

- **F3:**

$$n = S2 + S3 \quad (Bidireccional) \tag{4.6}$$

$$\overline{x} = \frac{S4 + S5}{n} \tag{4.7}$$

- **F4:**

$$n = S3 \quad (Unidireccional) \tag{4.8}$$

$$\overline{x} = \frac{S5}{n} \tag{4.9}$$

$$std = \sqrt{\frac{S6 - 2 * \overline{x} * S5 + n\overline{x}^2}{n-1}} \tag{4.10}$$

4.3. Arquitectura del Sistema de Detección

Para implementar el sistema de detección basado en el trabajo colaborativo ya mencionado entre el plano de control y el plano de datos, se propone la arquitectura mostrada en la Figura 4.3. Esta arquitectura está conformada por un desarrollo en el plano de datos usando P4, y un desarrollo en el plano de control, usando ONOS como controlador.

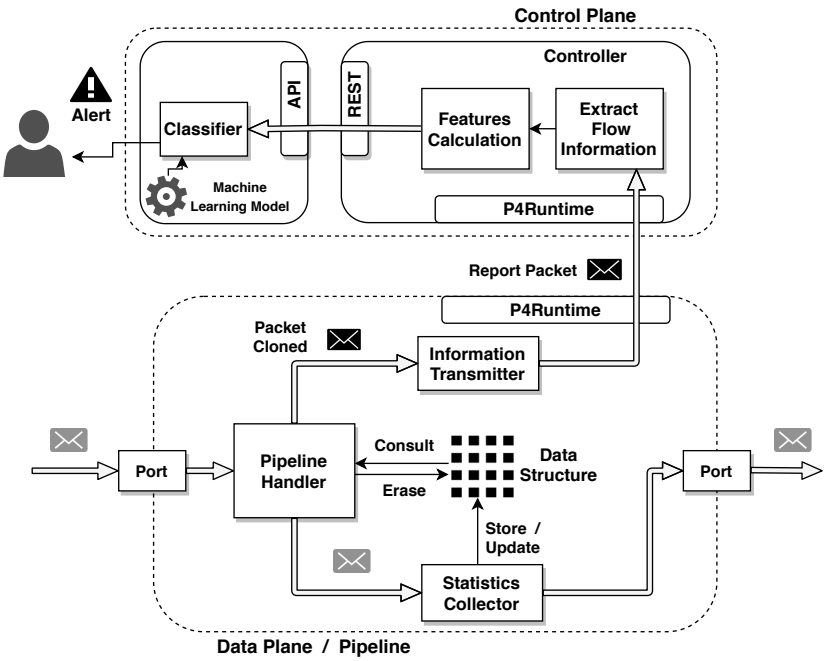


Figura 4.3: Arquitectura propuesta para el Sistema de Detección de Ataques DDoS

- **Plano de datos:** La implementación en el plano de datos se enfoca en extraer las estadísticas por flujo y reportarlas al plano de control usando la interfaz P4Runtime. El mecanismo para llevar a cabo esta tarea se basa en el uso de ventanas de tiempo. En cada ventana el dispositivo debe ser capaz de continuar recopilando estadísticas de flujo mientras que envía al mismo tiempo, las estadísticas de los flujos recopilados en la anterior ventana de tiempo. Este proceso de recolección y reporte se logró por medio del desarrollo de una aplicación P4 que contiene un *pipeline* dividido en 3 componentes principales: 1) Colector, 2) Control, y 3) Reporte, que interactúan con una estructura de datos implementada con registros y funciones hash.

  Cuando un paquete ingresa al dispositivo, el componente *colector* se encarga de extraer y procesar la información extraída de él. Luego, este mismo componente se encarga de actualizar las estadísticas del flujo al que pertenece dicho paquete, dentro de la

estructura de datos.

El componente de *Control*, antes que el paquete pase por el componente *Colector*, valida en la estructura de datos si aún hay flujos de la anterior ventana de tiempo por reportar. En caso que haya, el componente utiliza el *objeto externo* clonar, para crear una réplica del paquete que se envía inmediatamente al componente *Reporte* junto con la información de los flujos a enviar. Mientras tanto, el paquete original continúa su curso por el pipeline directo a su destino.

El componente *Reporte* se encarga de convertir el paquete clonado en un paquete de reporte. Lo primero que hace es eliminar cabeceras no necesarias en la comunicación con el controlador, como también la carga útil. Luego, la información de los flujos es adicionada al paquete en forma de una cabecera personalizada, antes de ser enviado dicho paquete por la interfaz P4Runtime.

- **Plano de control:** Este plano contiene dos implementaciones diferentes, una aplicación ONOS que se encarga de calcular las características (F2, F3 y F4) con las estadísticas del flujo recibido, y una segunda implementación que corresponde al componente de clasificación de tráfico, el cual es externo al controlador pero con una interfaz REST-API por donde se comunican.

  Cuando un paquete de reporte llega al controlador, el componente de *extracción de información* se encarga de fragmentar el paquete con el fin de obtener las estadísticas y el valor de la característica F1, por cada flujo reportado. Los valores de estas estadísticas (Tabla 4.2) son usados por el componente encargado de calcular las características restantes. Luego, una vez se forme la tupla con las 4 características (Tabla 4.1), estas son enviadas al componente *Clasificador* en una petición REST.

  El *Clasificador* es un componente a la espera de peticiones de clasificación. Este componente carga un modelo de ML previamente entrenado, el cual despliega su funcionalidad en el momento que una tupla de características arriba al componente. Este modelo clasifica cada tupla en dos posibles clases: el flujo es "Legítimo.º es de "DDoS". Cuando se clasifica un flujo como DDoS, se notifica al administrador de la red.

4.4. Implementación en el Plano de Datos

A continuación se explicará en detalle el funcionamiento de las dos tareas desempeñadas por el plano de datos: la recolección de estadísticas por flujo y el reporte de ellas al plano de control.

4.4.1. Mecanismo de Recolección de Estadísticas por Flujo

Como se detalló en la arquitectura, el mecanismo de recolección de estadísticas por flujo se lleva a cabo por medio de la interacción del componente *colector* con la estructura de datos, en los momentos que los paquetes pasan por el *pipeline*. Este mecanismo busca cumplir los siguientes requerimientos:

- Ser capaz de identificar el flujo al que pertenece cada paquete y con base en esto actualizar las estadísticas de su respectivo flujo.
- Permitir leer y escribir información en la estructura de datos a velocidad de línea para no generar latencia en el reenvío de los paquetes.
- Ser capaz de almacenar de forma independiente las estadísticas en ambas direcciones de un flujo bidireccional.
- Permitir que el componente de *Control* pueda identificar, dentro de la estructura de datos, las estadísticas de los flujos almacenados en la anterior ventana de tiempo.

Para satisfacer estos requerimientos, se implementa una estructura de datos construida con registros y funciones hash para el acceso rápido a cada elemento. Esta estructura posee N registros de tamaño M, que forman una especie de matriz (ver Figura 4.4). Cada registro (una fila de la matriz) almacena valores de una estadística en específico y cada elemento de la fila hace referencia al valor de un flujo unidireccional diferente. De acuerdo a esto, todas las estadísticas del mismo flujo unidireccional se encuentran almacenadas en una columna de la matriz.

Con base en lo anterior, el índice de la columna es entonces el identificador del flujo unidireccional dentro de la estructura de datos. Con el fin de identificar la columna de un flujo en particular, se usa la función hash para realizar un mapeo entre el identificador real de dicho flujo (tupla con campos de cabecera) con su identificador dentro de dicha estructura de datos (índice).

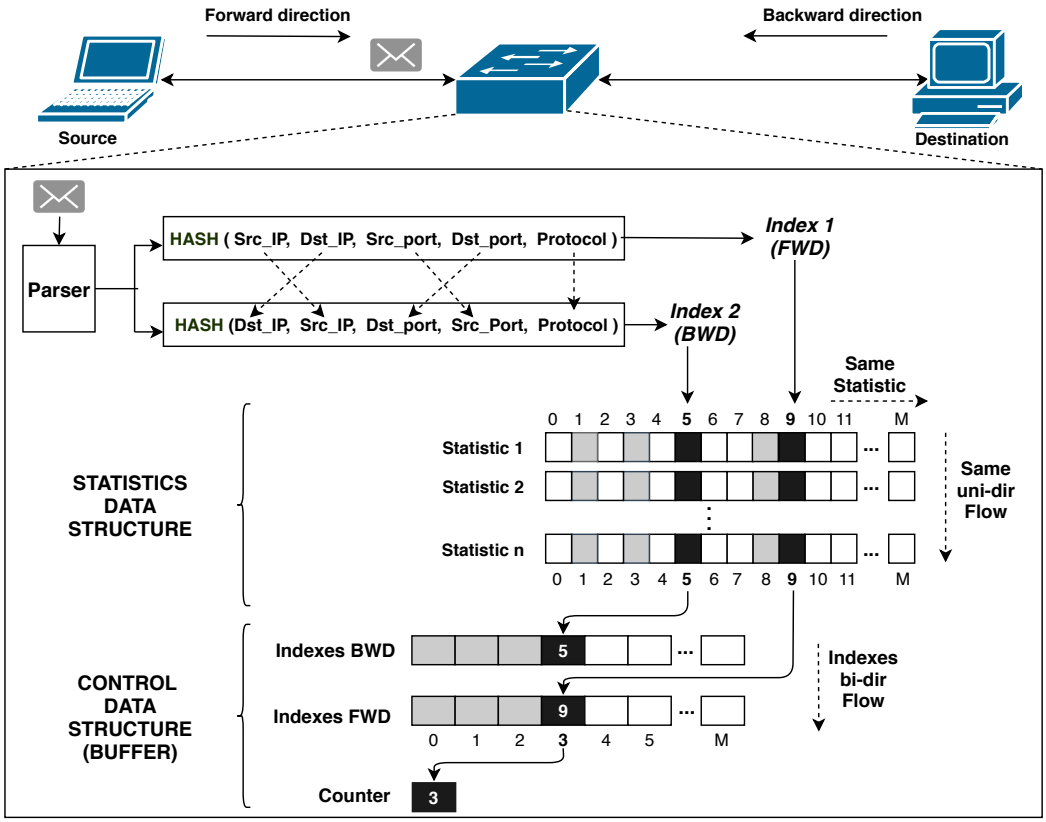


Figura 4.4: Mecanismo para Almacenar Estadísticas por Flujo.

El identificador de flujo adoptado en esta implementación se define con la siguiente tupla de 5 campos de cabeceras en el siguiente orden: *[IP_Origen, IP_destino, Puerto_Origen, Puerto_Destino, Protocolo]*.

En la Figura 4.4 se puede apreciar el mecanismo para actualizar las estadísticas de un flujo unidireccional. Cuando un paquete ingresa al dispositivo, este se fragmenta y se asignan los valores de sus campos de cabecera a variables, como se explicó en la sección 2.2.1.2 (Parser). Con los valores de las cabeceras, el componente *Colector* forma la tupla que identifica el flujo al que pertenece el paquete. A esta tupla se le calcula la función hash para obtener el índice donde se almacenan las estadísticas y así poder actualizarlas accediendo de forma óptima a cada elemento de este flujo unidireccional.

Sin embargo, como el plano de datos reporta flujos bidireccionales y el mecanismo almacena flujos unidireccionales, se debe identificar el índice donde están almacenados el par de flujos unidireccionales que forman el flujo bidireccional. Para conocer el índice del flujo en la dirección opuesta a la del paquete entrante, se calcula nuevamente la función hash a una

segunda tupla con los mismos campos de cabecera pero en diferente orden (ver Fig. 4.4). En este caso la IP y puerto de origen se convierten en IP y puerto destino. De acuerdo a esto se define como dirección Fwd al sentido del primer paquete registrado en el flujo, y la dirección Bwd al sentido opuesta.

Por otro lado, en el preciso momento en que se definen las direcciones de un nuevo flujo bidireccional (en el primer paquete del flujo), ambos índices son almacenados en una segunda estructura de datos. Esto se hace con el fin de que el componente de *Control* pueda gestionar este flujo, sin precisar de una tupla para calcular los índices. En la próxima sección (4.4.2) se entrará en mayor detalle sobre esta segunda estructura de datos.

4.4.2. Mecanismo de Reporte de Información de Flujo

El mecanismo de reporte en el plano de datos tiene como objetivo enviar al plano de control, al comienzo de cada ventana de tiempo, las estadísticas de todos los flujos bidireccionales recolectados en la anterior ventana de tiempo. Esta tarea se realiza gracias al trabajo en equipo entre el componente de *Control*, que interactúa con la estructura de datos para obtener la información a enviar, y el componente de *Reporte* que construye el paquete de reporte. Este mecanismo busca cumplir los siguientes requerimientos:

- Como la única manera de acceder a las estadísticas de un flujo es por medio de un mapeo hash, y sabiendo que en el momento de reportarlo no se cuenta con la tupla identificadora, el mecanismo debe poder acceder a todos los flujos bidireccionales a enviar sin realizar el hash para encontrar los índices en ambas direcciones.
- A pesar del funcionamiento secuencial de los *pipelines*, no puede haber una interrupción en el proceso de recolección mientras se están reportando flujos.
- El mecanismo debe ser capaz de reportar varios flujos bidireccionales en un mismo paquete de reporte, con el fin de reducir el tiempo de envío de todos los flujos a reportar.

De acuerdo con lo anterior, para solucionar el primer requerimiento, se opta por almacenar los índices de ambas direcciones del flujo en una segunda estructura de datos. Esta nueva estructura está formada por dos registros del mismo tamaño mas un contador. Los dos registros forman una matriz de $2xM$ en la que se va almacenando cada par de índices de forma ordenada y ascendente ocupando columnas (ver Fig.4.4). Por otro lado, el contador siempre apuntará a la posición donde fue almacenado el último par de índices con el fin de que el componente de *Control* sepa cuantos flujos bidireccionales debe reportar al plano de

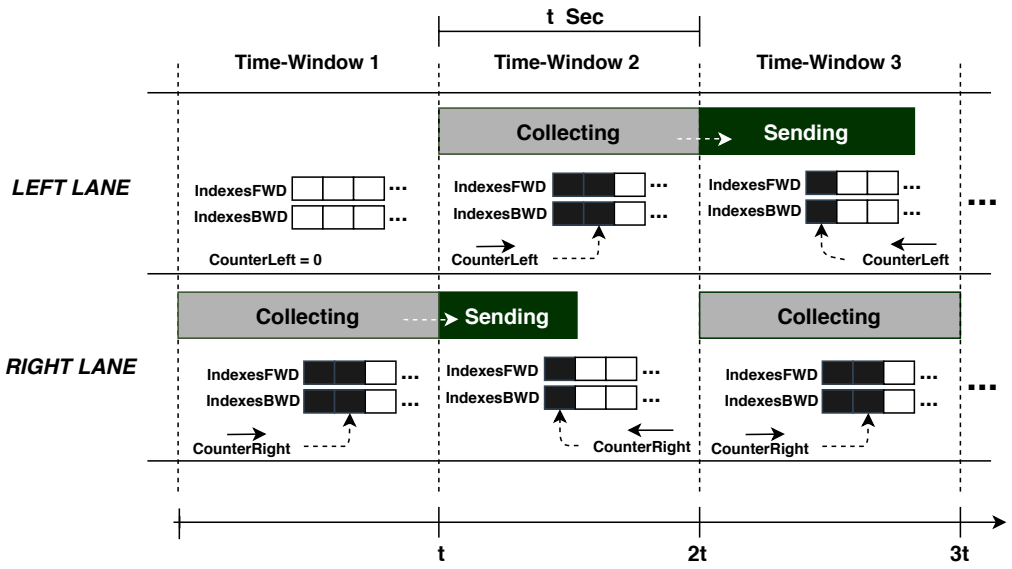


Figura 4.5: Mecanismo basado en carriles para el control de Ventanas de Tiempo y bufferes.

control en esa ventana de tiempo.

La forma de trabajar de esta estructura de datos es igual a un *buffer LIFO*: los índices se almacenan secuencialmente a medida que se instancian nuevos flujos, luego, cuando es el momento de reportarlos, el par de índices de cada flujo son consumidos y eliminados del *buffer* en el orden opuesto de llegada. Este proceso se realiza hasta que el *buffer* se encuentre vacío totalmente (contador en cero).

Por otro lado, con base en el segundo requerimiento, si se utiliza un solo *buffer* para almacenar y reportar, el mecanismo está obligado a interrumpir el proceso de recolección mientras vacía el *buffer*. Esta interrupción es necesaria ya que, de lo contrario, sería complejo diferenciar los índices de los flujos de la anterior ventana con los que están siendo colectados en la ventana actual. Como este proceso no puede ser interrumpido, se propone abstraer el concepto de una carretera con dos carriles, cada uno con su propio *buffer* independiente (Ver Fig. 4.5). De esta forma es posible, en la misma ventana de tiempo, delegar a un carril la tarea de recolección de índices nuevos mientras que el otro carril se encarga de reportar los flujos almacenados en la anterior ventana de tiempo. En esta Figura 4.5 se aprecia que el *buffer* de cada carril siempre esta alternando entre almacenar y reportar a medida que transcurren las ventanas de tiempo de t segundos.

De acuerdo a lo anterior, usando dos *buffer* independientes el componente de *Control* puede validar constantemente si hay flujos para reportar. Si los hay, este componente puede

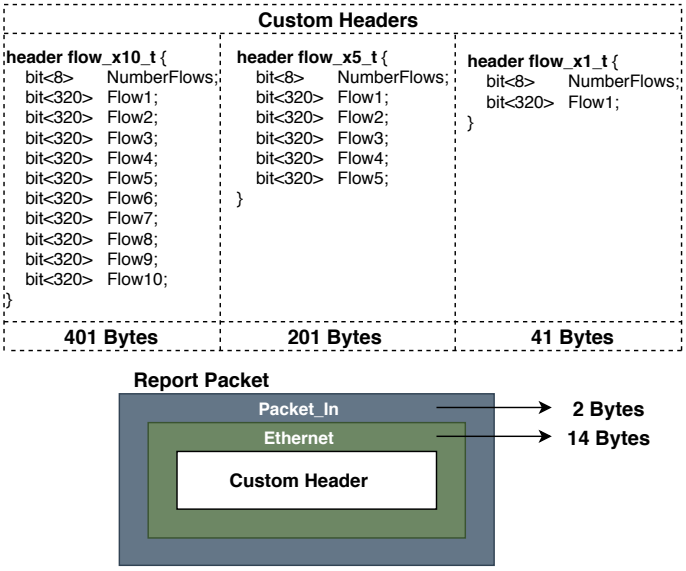


Figura 4.6: Estructura del Paquete de Reporte.

consumir los índices almacenados en el respectivo *buffer*. Con estos índices, el componente extrae las estadísticas de los flujos a reportar, calcula por cada uno la característica F1 con la marca de tiempo del primer y último paquete del flujo, clona el número de paquetes necesarios y delega el proceso de envío al componente de *Reporte*.

El componente de *Reporte* transforma los paquetes clonados en paquetes de reporte eliminando información no necesaria y adicionando la información por flujo dentro de una cabecera personalizada. De acuerdo al tercer requerimiento, entre mas flujos se envíen por paquete, el proceso de reporte es más eficiente, teniendo en cuenta que el número máximo de flujos es limitado por la MTU de la red. Sin embargo, enviar demasiados flujos en un solo paquete podría generar latencia en el dispositivo. De acuerdo con esto se decide reportar, por cada paquete, grupos de 10, 5 o 1 flujo(s) en su defecto, de acuerdo a la demanda de flujos a enviar, siendo 10 la prioridad.

La estructura del paquete de reporte construído se aprecia en la Figura 4.6. Como en esta implementación se usó un canal fuera de banda entre el plano de datos y el plano de control, el paquete contiene solamente las cabeceras *Packet_in*, Ethernet y la cabecera personalizada con la información de los flujos, en ese orden de encapsulamiento. Las demás cabeceras junto con la carga útil son eliminadas del paquete clonado para hacerlo mas liviano.

Por otro lado, las cabeceras personalizadas para cada caso de envío (10, 5 o 1 flujo(s)) se especifican también en la Figura 4.6. Cada campo es la concatenación en bits de la información a

reportar de un flujo, los cuales tienen un tamaño de 40 bytes (ver Tabla 4.2). Adicionalmente, estas cabeceras poseen al comienzo un campo de 1 byte que le indica al plano de control cuántos flujos fueron reportados en el paquete, y así saber como proceder a fragmentar el paquete.

4.5. Implementación en el Plano de Control

El objetivo principal del plano de datos es realizar en tiempo real la clasificación de los flujos bidireccionales de la red en dos clases: *"Legítimo"* o *"DDoS"*. Para realizar la clasificación, se entrenaron modelos supervisados de ML usando dos algoritmos de clasificación diferentes: K-Vecinos mas Cercanos (KNN) y Bosque Aleatorio (RF). Estos modelos se entrenaron con una base de datos construída con el mismo mecanismo de extracción de características propuesto en esta monografía, que se explicará en detalle en el próximo Capítulo.

4.5.1. Clasificador KNN

Este es uno de los algoritmos clásicos y sencillos para realizar clasificación de dos o más clases. Este algoritmo es basado en instancias, es decir, memoriza todas las instancias de una base datos en la fase de entrenamiento, para que en la fase de clasificación, clasifique una nueva instancia con base en las memorizadas. En nuestro caso una instancia sería la tupla de 4 características que describen un flujo, como se define en la Tabla 4.1. De acuerdo a esto, KNN considera todos los flujos de esta implementación como un conjunto de datos que viven en 4 dimensiones.

Para que un modelo de KNN clasifique una nueva instancia de flujo, este realiza los siguientes pasos [57]:

- Calcula la distancia entre la nueva instancia y el resto de instancias memorizadas en la fase de entrenamiento. Existen varios criterios de distancia pero el mas usado es la distancia euclidiana.
- Forma un vecindario con las K instancias más cercanas, con las que se conoce la clase a la que pertenecen.
- Realiza una votación de mayoría entre los K vecinos más cercanos, es decir, se asigna a la nueva instancia la clase predominante dentro del vecindario. Por esta razón, K debe ser preferiblemente un numero impar para no tener empates en la votación.

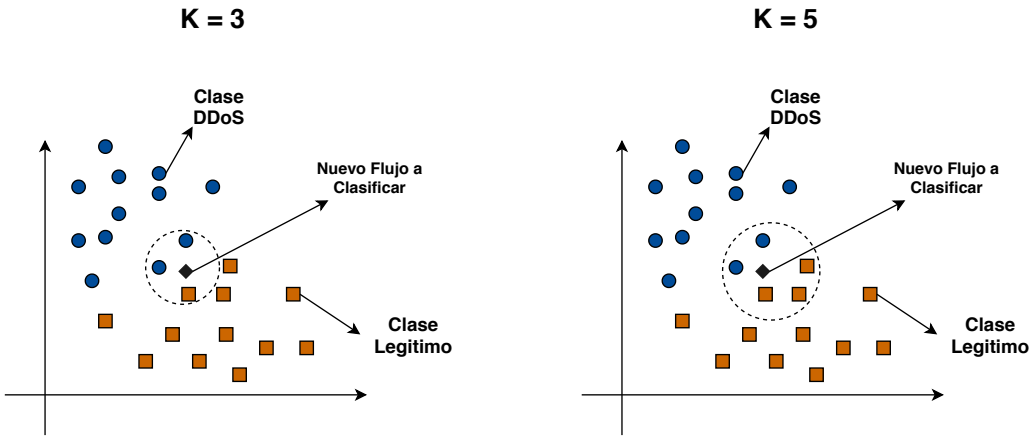


Figura 4.7: Lógica de K-Vecinos mas cercanos.

En la Figura 4.7 podemos ver gráficamente la lógica con la que clasifica KNN. En la figura del lado izquierdo, tenemos un vecindario de K=3, el cual es conformado por 2 flujos DDoS y un flujo Legítimo. Acorde a esto, el nuevo flujo es clasificado como DDoS. Por otro lado, en la figura del lado derecho, se aumenta el tamaño del vecindario a K=5, el cual ya está conformado por los mismos 2 flujos DDoS pero con 3 flujos legítimos. Por lo tanto, el nuevo flujo es clasificado como Legítimo.

Con base en lo anterior ***K*** es el hiper-parámetro a optimizar en KNN, es decir, encontrar el *K* que proporciona una mayor exactitud en la clasificación.

4.5.2. Clasificador RF

RF es un modelo de aprendizaje conformado a la vez por un conjunto de modelos de árboles de decisión (DT), los cuales permiten una clasificación mas robusta y generalizada de los flujos de la red, cuando se compara con un solo modelo de DT.

En la Figura 4.8 se puede apreciar el mecanismo de clasificación empleado por RF. Este mecanismo divide la base de datos que contiene los flujos de entrenamiento en porciones iguales con instancias de flujos escogidas aleatoriamente. El número de porciones es igual al número de árboles (modelos) que contendrá el bosque. Cada árbol es entrenado con una porción de flujos diferente los cuales tienen como propósito común clasificar nuevas instancias de flujo en las clases “Legítima” ó “DDoS”. Cuando se quiere clasificar una nueva instancia de flujo, cada uno de los árboles del bosque realizan la clasificación arrojando la clase resultante con la que se hace una votación. RF da el veredicto final asignando el flujo a

la clase con mayor votación dentro del bosque.

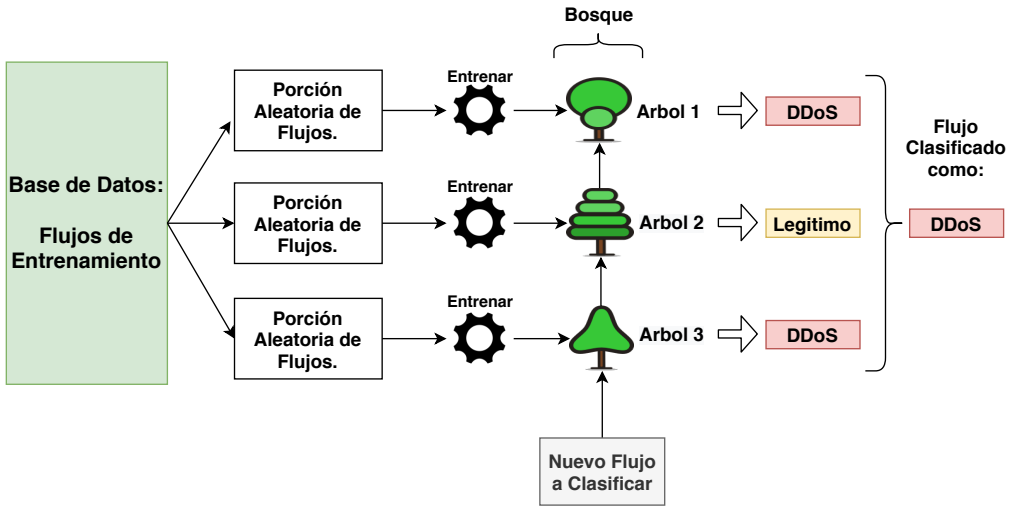


Figura 4.8: Bosque Aleatorio.

Cuando se entrena un modelo de RF, hay que limitar el valor del conjunto de hiper-parámetros de entrada para evitar modelos sobre-ajustados o modelos con bajo rendimiento en detección. Entre ellos, los hiper-parámetros que se iteraron en la fase de entrenamiento de los modelos de este trabajo son:

- **n_estimators:** número de árboles que va a tener el bosque aleatorio. Normalmente cuantos más mejor, pero a partir de cierto punto deja de mejorar y sólo hace mas pesado el proceso de clasificación.
- **max_depth:** profundidad máxima que pueden tener los árboles. Si se escoge un valor muy grande, puede ser muy propenso a que el modelo se aprenda los datos de entrenamiento y no sea capaz de clasificar nuevos datos con una buena exactitud.

Capítulo 5

Evaluación de Desempeño

En este capítulo se presenta la metodología experimental que se lleva a cabo para evaluar el mecanismo de detección de ataques de DDoS propuesto en el Capítulo 4. Mas específicamente se explica el escenario experimental utilizado, los modelos de clasificación que se entrenaron, además de los resultados obtenidos en eficiencia de detección y de consumo de recursos de procesamiento en el plano de control. Con los resultados presentados en este capitulo se busca responder las siguientes preguntas:

- **P1:** ¿La duración de la ventana de tiempo (TW) definida en el mecanismo del plano de datos, influye en la eficiencia de detección y de consumo de recursos?
- **P2:** ¿El sistema de detección propuesto es efectivo en la detección de ataques de DDoS con un consumo razonable de recursos del plano de control?
- **P3:** ¿Cuál es el mejor clasificador de los propuestos?

5.1. Configuraciones Experimentales

5.1.1. Ataque de DDoS

Para poner a prueba el sistema de detección, se optó por replicar la traza de paquetes resultante del experimento hecho por [11] en un escenario real. Este experimento consistió en el despliegue por 20 minutos de un ataque de DDoS proveniente desde Internet, con destino un servidor web ubicado en la DMZ de una red local. Esta traza, la cual posee tráfico capturado por 90 minutos, está conformada por 762.973 paquetes de DDoS y 1'382.900

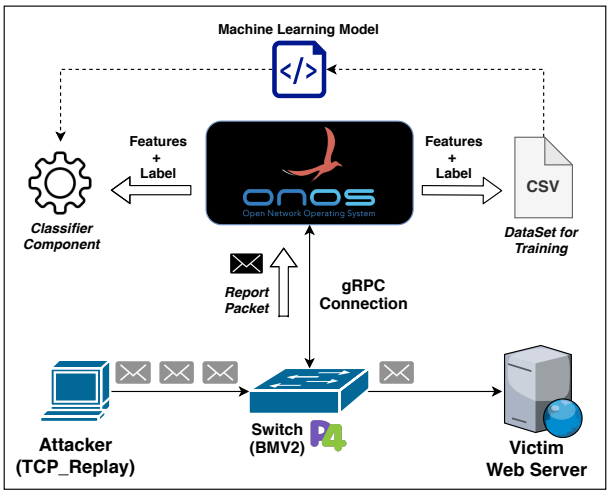


Figura 5.1: Escenario de pruebas.

paquetes legítimos, almacenados en un archivo .PCAP que está disponible para el uso libre de la comunidad científica [58].

Este ataque fue desplegado desde varias máquinas coordinadas (zombies) que ejecutaban la herramienta generadora de tráfico de DoS: Low Orbit Ion Canon (LOIC) [40], todas apuntando al mismo objetivo. Esta herramienta generó grandes cantidades de tráfico basura TCP, UDP y peticiones HTTP tipo GET con el fin de sobrecargar el servidor.

5.1.2. Escenario Experimental

Las diferentes pruebas hechas al sistema de detección fueron realizadas en el escenario emulado de la Figura 5.1. En este escenario, el plano de datos se implementó mediante *mininet* [59] de la siguiente manera:

- Un único dispositivo de borde que separa la red local de la externa (Internet). Para implementar un dispositivo con funcionalidades P4 y a la vez con soporte de interfaz P4Runtime, se carga con *mininet* el conmutador virtual *BMV2* (behavioral-model version 2) [60]. Este conmutador funciona bajo la arquitectura V1model especificada en la Figura 2.3 y soporta los diferentes objetos externos requeridos por la aplicación P4 desarrollada para este sistema: registros, funciones hash y clonación de paquetes. De acuerdo con esto, este es el conmutador que lleva a cabo la extracción y reporte de información por cada flujo hacia el plano de control, a medida que el tráfico replicado lo atraviesa.

- Un *host* que representa el ataque distribuído proveniente de internet. Es en este *host* donde se reproduce el archivo .PCAP con el trafico legítimo y de DDoS, por medio de la herramienta *TCPReplay* [61].
- Un *host* víctima, que representa el servidor web a donde se dirigió todo el tráfico replicado desde el *host* atacante.

Por otro lado, el plano de control es conformado por el controlador ONOS quien establece una conexión gRPC con el conmutador *BMV2* usando la interfaz P4Runtime. Adicionalmente, este plano cuenta con el componente clasificador. Este es un servicio desarrollado en python el cual carga el modelo de ML previamente entrenado usando las librerías de *scikit-learn* [62]. Este servicio expone una API que permite atender las peticiones de clasificación del controlador.

Este escenario se implementó en una máquina con las características descritas en la Tabla 5.1. Por otro lado, el plano de datos se aisló en una máquina virtual instanciada en la misma máquina física. Las características de la máquina virtual se muestran en la Tabla 5.2.

Tabla 5.1: Máquina Física de pruebas.

Parámetro	**Valor/Version**
CPU	Core i5 (4 hilos)
RAM	12 GB
Sistema operativo	Ubuntu 18.04.2 LTS
Mininet	2.3.0d6
P4	16
BMV2	1.13.0-10c2d343
p4c-bm2-ss (compilador)	1.1.0-rc1

Tabla 5.2: Máquina Virtual.

Parámetros	**Valor/Version**
CPU delegada	2 hilos
RAM delegada	4 GB
ONOS	2.2.0
Python	2.7

5.1.3. Cargas de Trabajo

Los diferentes experimentos se realizaron usando dos cargas de trabajo, una para la construcción de las bases de datos con que se entrenaron los modelos, y la otra para probar dichos modelos. Para crear ambas cargas, se fragmenta el archivo .PCAP en 18 partes iguales de 5 minutos, donde se numeraron del 1 al 18 respetando su orden en la traza. Los fragmentos con numeración “impar” se unieron para construir el .PCAP de entrenamiento (CTE), y de la misma manera, se unen los fragmentos con numeración “par” para construir el .PCAP de prueba (CTP).

En la tabla 5.3 se muestra el número de paquetes legítimos y de DDoS por cada carga de trabajo. Adicionalmente, se especifica la proporción de paquetes de DDoS con respecto a la cantidad de paquetes legítimos.

Tabla 5.3: Caracterización de las cargas de trabajo de Entrenamiento y de Prueba.

Carga de Trabajo	**# de Paquetes Legítimos**	**# de Paquetes DDoS**	**Proporción**	**Duración**
Entrenamiento (CTE)	611.310	371.230	60,7 %	45 min
Prueba (CTP)	771.590	391.743	50,8 %	45 min

5.1.4. Métricas de Desempeño

Cuando se prueba la eficiencia de un modelo de clasificación usado para detectar ataques de DDoS, al final de la prueba se obtienen 4 estadísticas importantes:

- *TP:* número de flujos DDoS clasificados como DDoS.
- *FP:* número de flujos legítimos clasificados como DDoS.
- *TN:* número de flujos legítimos clasificados como legítimos.
- *FN:* número de flujos DDoS clasificados como legítimos.

Con base en estas estadísticas, se definen las siguientes métricas basadas en proporciones para evaluar dicha eficiencia:

- **Exactitud:** Es el número de casos detectados correctamente dentro del total de instancias de flujo, considerando ambas clases ("legítimo", y "DDoS").

$$Exactitud = \frac{TN + TP}{TN + TP + FP + FN} \tag{5.1}$$

- **Sensibilidad:** Representa la proporción de instancias de flujo DDoS detectadas correctamente con relación al total de ellas. Esta métrica es mejor conocida como la rata de verdaderos positivos (TPR).

$$Sensibilidad = TPR = \frac{TP}{TP + FN} \tag{5.2}$$

- **Especificidad:** Representa la proporción de instancias de flujo legítimo detectadas correctamente con relación al total de ellas. Esta métrica es mejor conocida como la rata de verdaderos negativos (TNR).

$$Especificidad = TNR = \frac{TN}{TN + FP} \tag{5.3}$$

- **F1_Score:** Es otra métrica que mide la exactitud del sistema considerando ambas clases. La diferencia radica en que se basa en el promedio ponderado entre las métricas de precisión y de Especificidad.

$$F1\_score = \frac{2 * TP}{2 * TP + FP + FN} \tag{5.4}$$

5.1.5. Factores

En la evaluación de desempeño del sistema de detección, se definieron dos factores en el momento de realizar los experimentos:

- La duración de la ***ventana de tiempo*** en que el plano de datos recolecta y reporta periódicamente flujos al plano de control. Este factor varía en los siguientes valores: 5, 20, 40 y 60 segundos de ventana.
- El tipo de ***clasificador***: RF o KNN.

De acuerdo a estos factores se entrenaron y probaron 8 modelos de clasificación. Cada modelo es producto de una combinación posible entre los valores de ambos factores.

5.2. Entrenamiento de Modelos

El primer paso antes de entrenar un modelo de clasificación, es adquirir la base de datos con que se realiza dicho proceso. Para construir esta base de datos, se desarrolló una aplicación ONOS que permitió almacenar, dentro de un archivo .csv, el conjunto de características y la respectiva etiqueta (clase: “legítimo”, o “DDoS”) por cada flujo reportado por el plano de datos (ver Fig 5.1). Estas etiquetas, necesarias para entrenar los modelos supervisados, se obtuvieron directamente de los paquetes pertenecientes a la carga de trabajo CTE una vez era replicada en el plano de datos. Para este propósito, todos los paquetes de esta carga de trabajo fueron previamente marcados (00:Legítimo, 11:DDoS) modificando los últimos dos

bits del campo de cabecera IPv4: "Tipo de Servicio". Con base en estas marcaciones, fue posible en la fase de construcción de la base de datos, reportar al plano de control la etiqueta de cada flujo junto con sus respectivas estadísticas y así almacenarlas con las características calculadas.

Este proceso se realizó por cada valor de duración de ventana de tiempo considerado (5, 20, 40 y 60 segundos), obteniendo así 4 bases de datos diferentes. En la Tabla 5.4 se puede ver la cantidad de flujos legítimos y de DDoS almacenados en cada una de ellas.

El mecanismo de reporte de flujos no considera Flujos-Completos (definidos en la sección 2.4), es decir, todos los flujos son enviados al plano de control en la siguiente ventana de tiempo sin considerar si finalizaron o no. Por esta razón, aunque la carga de trabajo replicada es la misma, observamos en la Tabla 5.4 una disminución en el total de flujos almacenados a medida que incrementa la duración de la ventana de tiempo. Esto es debido a que entre mas dure la ventana de tiempo, en menos partes se fragmentarían aquellos flujos de duración mayor a dicha ventana. Con base en esto, podemos notar que hay menos cantidad de flujos DDoS de duración larga en comparación a los flujos legítimos ya que el decremento es menos significativo entre duraciones de ventana. La razón para no considerar Flujos-Completos en esta implementación es porque podría convertirse en una vulnerabilidad si el atacante lo descubre. De esta forma el atacante podría atacar con flujos DDoS de duración indefinida y así evitar que esos flujos sean reportados y clasificados en el plano de control.

Tabla 5.4: Caracterización de las Bases de Datos de Flujos

Base de Datos por cada Ventana de Tiempo				
Clase / Ventana de Tiempo	**DB1: 5 Sec**	**DB2: 20 Sec**	**DB3: 40 Sec**	**DB4: 60 Sec**
Flujos Legítimos	61.736	40.462	31.894	29.670
FLujos DDoS	49.012	45.688	44.511	43.255
Total de Flujos por Base de Datos:	110.748	86.150	76.405	72.925

Por otro lado, por cada base de datos construida se entrenaron dos modelos, uno con RF y otro con KNN para la respectiva duración de ventana de tiempo (8 modelos en total). Al entrenar cada modelo se implementó una validación cruzada [63] de 10 iteraciones para encontrar el modelo mas exacto y generalizado dentro de muchos posibles. En cada iteración, un conjunto diferente de entrenamiento y validación es adquirido de la misma base de datos. Con dichos conjuntos se llevó a cabo una malla de búsqueda que encontró el mejor

conjunto de valores de hiper-parámetros que entrenó el mejor modelo de clasificación en cada iteración de la validación cruzada. Al final del proceso de entrenamiento se contó con los 10 mejores modelos, de los cuales se eligió el mejor de los mejores en términos de exactitud en clasificación.

En la Tabla 5.5 se encuentran los valores de los hiper-parámetros que entrenaron el mejor modelo en cada caso. Adicionalmente, se puede apreciar el tiempo promedio en segundos que fue requerido para encontrar los hiper-parámetros y entrenar el modelo. Por otro lado, podemos notar que el tiempo empleado en RF fue mayor que el tiempo empleado en KNN. Esto se de a que la malla de búsqueda de RF tenía que hacer una búsqueda mas exhaustiva al contar con más hiper-parámetros.

Tabla 5.5: Mejores valores de Hiper-parámetros / Tiempo de entrenamiento

Bosque Aleatorio (RF)				
Hiper-parámetro / Ventana de T.	**M1: 5 Sec**	**M2: 20 Sec**	**M3: 40 Sec**	**M4: 60 Sec**
Max_depth	6	6	6	6
N_estimators	300	500	500	300
Tiempo de entrenamiento	332.8 ± 6	304.4 ± 8.1	**264.4 ± 20**	272.6 ± 20
K-Vecinos mas Cercanos (KNN)				
Hiper-parámetro / Ventana de T.	**M5: 5 Sec**	**M6: 20 Sec**	**M7: 40 Sec**	**M8: 60 Sec**
K	3	3	3	3
Tiempo de entrenamiento	59.8 ± 2	37.7 ± 1	36.3 ± 3	**26.7 ± 1**

5.3. Resultados

En esta sección se mostrarán los resultados obtenidos en la fase de prueba del sistema de detección. En dichas pruebas se realizaron repetitivos experimentos con los 8 modelos de clasificación previamente entrenados y cargados en el componente clasificador.

Para llevar a cabo cada experimento, se replicó en el escenario de prueba la carga de trabajo CTP, que al igual que la carga de trabajo CTE (ver sección 5.1.3), posee también todos los paquetes marcados. Por otro lado, en esta fase de prueba tanto el conjunto de características y la etiqueta del flujo no son almacenados, por el contrario, son enviados al componente clasificador que se encarga de clasificar el flujo y comparar el resultado con la etiqueta recibida. El objetivo de enviar estas etiquetas al clasificador radica en saber en que ocasiones

acertó o erró el sistema de detección y poder obtener las estadísticas TP, FP, TN y FN con que se calculan las diferentes métricas por cada experimento realizado. Cabe resaltar que en un escenario de producción estas etiquetas no son extraídas por el sistema, este solo se limitaría a la extracción de características por flujo y a su clasificación.

5.3.1. Resultados en Detección

En la Figura 5.2 se muestran los resultados obtenidos referente a la métrica de exactitud de clasificación (Ecuación 5.1) en función de las diferentes duraciones de ventana de tiempo. Adicionalmente, se comparan ambos clasificadores (KNN y RF) para cada valor de ventana.

De la Figura 5.2 podemos observar que los 8 modelos arrojaron resultados en exactitud por encima al 93 %, valores que se encuentran dentro del rango superior visto en la literatura, lo que los hace aptos para ser usados en el sistema de detección. Por otro lado, podemos apreciar que el resultado de exactitud de KNN es superior al de RF en todos los valores de ventanas de tiempo, sin embargo dicha diferencia entre modelos no supera el 1.2 % en todos los casos.

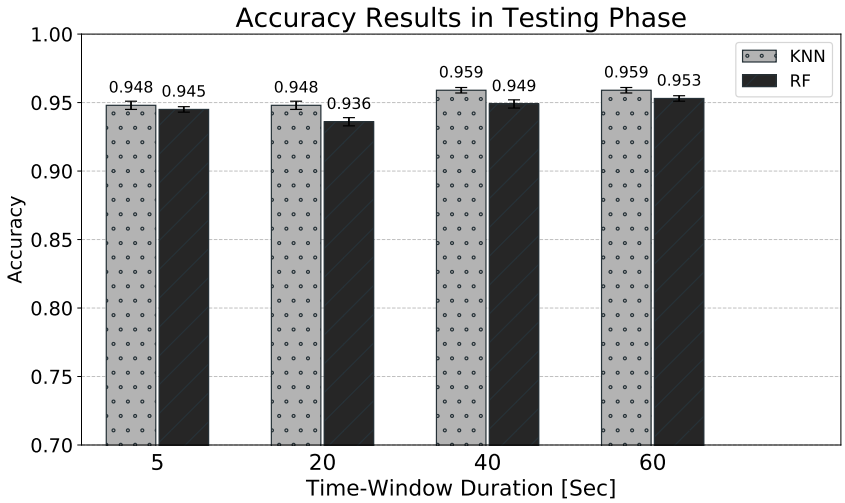


Figura 5.2: Resultados en exactitud de clasificación por modelo y ventana de tiempo.

Por otro lado, en la misma Figura 5.2 se nota en ambos clasificadores una tendencia creciente en el valor de exactitud a medida que la duración de la ventana de tiempo incrementa. De acuerdo con esto, el sistema de detección sería mas exacto al usar ventanas de 60 segundos. Sin embargo, 60 segundos podría ser un intervalo de tiempo muy amplio que conduciría a retrasos no tolerables en la detección de ataques de DDoS. Analizando la compensación

que hay entre rapidez y exactitud en detección, se considera que no es necesario penalizar la rapidez ya que la diferencia en exactitud entre la ventana de 60 segundos y 5 segundos no es muy significativa (1.1 % para KNN y 1.7 % para RF).

En la Tabla 5.6 se presentan los resultados obtenidos de las métricas restantes para los 8 modelos. Con estas métricas se analiza que tan bueno es cada modelo clasificando por separado cada clase (Sensibilidad: "DDoS", Especificidad: "Legítimo"), además del segundo indicador de exactitud considerando ambas clases (F1 Score).

Tabla 5.6: Comparación de métricas en la fase de prueba.

Bosque Aleatorio (RF)			
T.W. / Metrica	**Sensibilidad**	**Especificidad**	**F1 Score**
M1: 5 seg.	0.982 ± 0.001	0.92 ± 0.003	0.934 ± 0.002
M2: 20 seg.	0.933 ± 0.005	0.939 ± 0.003	0.933 ± 0.003
M3: 40 seg.	**0,957 ± 0.002**	**0,957 ± 0.003**	**0,952 ± 0.003**
M4: 60 seg.	0.963 ± 0.003	0.941 ± 0.002	0.959 ± 0.002
K-Vecinos mas Cercanos (KNN)			
T.W. / Metrica	**Sensibilidad**	**Especificidad**	**F1 Score**
M5: 5 seg.	0.967 ± 0.002	0.935 ± 0.003	0.937 ± 0.003
M6: 20 seg.	0.951 ± 0.003	0.943 ± 0.004	0.945 ± 0.003
M7: 40 seg.	**0,97 ± 0.003**	**0,967 ± 0.003**	**0,962 ± 0.002**
M8: 60 seg.	0.97 ± 0.003	0.946 ± 0.003	0.964 ± 0.002

El sistema de detección tiene como interés principal poseer un modelo de clasificación que sea muy preciso clasificando la clase DDoS (FN pequeños) y a la vez que presente la menor cantidad posible de falsas alarmas (FP pequeños). De acuerdo a las métricas, entre mayor sea el valor de sensibilidad mas preciso sería el modelo detectando ataques de DDoS y entre mayor sea el valor de especificidad menor sería la cantidad de falsas alarmas de ataques de DDoS (ver ecuaciones 5.3 y 5.2).

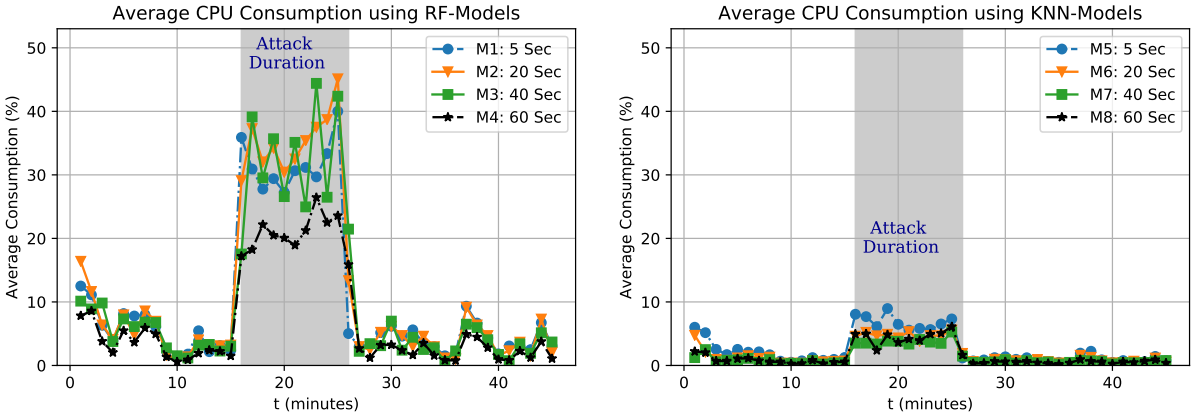
De acuerdo con los resultados de la Tabla 5.6, el modelo M1 es el mas preciso de todos detectando ataques de DDoS con un valor de sensibilidad del 98 %. sin embargo M1 posee el valor de especificidad mas bajo de todos, lo que convierte a este modelo en uno de los menos confiables ya que la tasa de falsas alarmas sería mayor a los otros. Este análisis se ve reflejado en el valor de F1_Score que es uno de los mas bajos en comparación a los otros. Por otro lado, pensando en una mejor compensación entre precisión, falsas alarmas y duración de la ventana de tiempo, se consideran M3 y M7 las mejores opciones a usar en el sistema de

detección. Estos modelos poseen los valores de sensibilidad y especificidad más cercanos entre ellos, lo que refleja un equilibrio en ambos propósitos. Adicionalmente, estos resultados sobrepasan el 95 % de precisión clasificando cada clase y ambos modelos coinciden en la ventana de 40 segundos para ambos clasificadores. Sin embargo, de ambos modelos el mejor de ellos es el basado en KNN (M7) ya que posee un F1_Score de 96 % el cual supera en 1 % al basado en RF (M3).

5.3.2. Resultados en Consumo de Recursos

Para analizar como es el consumo de recursos en el plano de control cuando está en funcionamiento el sistema de detección, se decide medir la métrica de consumo promedio de CPU. Para llevar a cabo esta medición en los 8 experimentos, el valor de consumo instantáneo de CPU es medido cada 2 segundos durante los 45 minutos de tráfico que contiene la carga de trabajo CTP.

En las Figuras 5.3a y 5.3b se muestran los resultados obtenidos para el consumo promedio de CPU, agrupados por el tipo de clasificador (RF o KNN) usado en el experimento. Cada curva en estas gráficas hace referencia a la interpolación de 45 puntos que representan el consumo promedio de CPU en cada minuto transcurrido.



(a) Consumo Promedio usando modelos **RF**.

(b) Consumo Promedio usando los modelos **KNN**.

Figura 5.3: Consumo Promedio de CPU en el Plano de Control.

De la Figura 5.3 se observa que el consumo promedio en todos los casos fue inferior al 50 %, incluso en presencia del ataque de DDoS. Estos resultados demuestran el buen nivel de resiliencia que brinda el sistema de detección propuesto. Por otro lado, comparando ambos

clasificadores se evidencia una gran disminución en el consumo de CPU usando modelos KNN, valor que no supera el 10 %. Esto nos da a entender que el mecanismo empleado por KNN para clasificar tráfico es mucho más simple y eficiente que el empleado por RF.

Por otro lado, en la Figura 5.3a se aprecia claramente el incremento anormal en el consumo promedio de CPU durante el intervalo de ataque. En este intervalo, se observa que los modelos M2 y M3 tienden a consumir más en comparación a los modelos M1 y M4. Esta diferencia de consumo es debido a que M2 y M3 poseen bosques de 500 árboles y, M1 y M4 de 300 árboles (ver Tabla 5.5). De acuerdo a esto se evidencia que el tamaño del bosque en los clasificadores RF es un factor importante en el uso de modelos de bajo consumo de CPU. Por otro lado, comparando M1 y M4, ambos con el mismo tamaño de bosque, se observa que la duración en la ventana de tiempo también influye en el consumo promedio. Con base en lo anterior, se puede concluir que, entre mayor sea el valor de la ventana, menor es el consumo de CPU, siendo así M4 el modelo RF que menos consume CPU.

Tabla 5.7: Tiempo Promedio requerido para el despliegue de cada Proceso en el Plano de Control.

Bosque Aleatorio (RF)			
T.W.	**Calcular Características [ms]**	**Clasificar [ms] Flujos**	**Tiempo Total [ms]**
M1: 5 seg.	0,04 ± 0,3	43,16 ± 11	44,82 ± 12
M2: 20 seg.	0,05 ± 0,3	61,29 ± 16	62,97 ± 17
M3: 40 seg.	**0,05 ± 0,3**	**64,44 ± 27**	**66,27 ± 28**
M4: 60 seg.	0,06 ± 0,4	40,42 ± 17	42,4 ± 18
K-Vecinos mas Cercanos (KNN)			
T.W.	**Calcular Características [ms]**	**Clasificar [ms] Flujos**	**Tiempo Total [ms]**
M5: 5 seg.	0.08 ± 0.3	4.74 ± 4	7.54 ± 5
M6: 20 seg.	0.05 ± 0.3	3.68 ± 3	5.79 ± 3
M7: 40 seg.	**0.05 ± 0.3**	**3.34 ± 2**	**5.29 ± 3**
M8: 60 seg.	0.06 ± 0.3	3.77 ± 2	5.97 ± 3

En la Tabla 5.7, se mide el tiempo promedio total en milisegundos requerido por el sistema para clasificar 10 flujos reportados en un solo paquete. Este tiempo fue medido desde el instante en que llega el paquete al controlador, hasta que el clasificador termina de clasificar el último flujo. Adicionalmente se presenta el tiempo promedio requerido para llevar a cabo individualmente el cálculo de los 10 conjuntos de características y su clasificación.

Como se puede observar en la Tabla 5.7, la mayor parte del tiempo total es empleado en el proceso de clasificación de los 10 flujos. El proceso de fragmentar el paquete y calcular los conjuntos de características, es poco significativo (menos de 1 ms) en comparación con el proceso de clasificación. Por otro lado, al igual que lo sucedido con el consumo promedio de CPU, el tiempo total empleado por los modelos de KNN son mucho menores al empleado por los modelos de RF. Adicionalmente, se observa para RF que los modelos M2 y M3, que poseen un bosque de 500 árboles, requieren aproximadamente 20 milisegundos en promedio de más para clasificar los flujos, en comparación con M1 y M4 con 300 árboles. Con respecto al valor de la ventana de tiempo usada en el plano de datos, este no influye en los tiempos resultantes en ambos procesos desplegados en el plano de control.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En este trabajo, hasta donde alcanza nuestro conocimiento, se propuso el primer entorno colaborativo entre un plano de datos programable y el plano de control para la implementación de un sistema de detección de ataques de DDoS, el cual usa modelos de aprendizaje de máquina para la clasificación de los flujos. En este entorno, la implementación en el plano de datos está conformada por un mecanismo de recolección de información por flujo y un mecanismo que reporta periódicamente dicha información ya procesada al plano de control.

El uso de un plano de datos programable brindó la posibilidad de extraer y procesar información de flujo con un mayor nivel de granularidad. Esto permitió que varios tipos de características, entre ellas las catalogadas como Intra-Flujo, fueran posibles de calcular y usar en un sistema de detección implementado sobre la arquitectura SDN. Por otro lado, aprovechando las diferentes primitivas que brinda P4, se logró reportar al plano de control en cada ventana de tiempo, información de flujo ya procesada que facilitó y optimizó el cálculo del conjunto de características, como también el proceso de clasificación.

Para la clasificación de los flujos se entrenaron modelos de RF y KNN bajo los diferentes valores de ventana de tiempo usados en el plano de datos. En los 8 modelos entrenados se obtuvo buenos resultados en exactitud de detección, resultados que estuvieron por encima del 93 %. Esto nos permite concluir que tanto los clasificadores como el conjunto de características escogido fueron excelentes descriptores del tráfico de la red, logrando así diferenciar el comportamiento legítimo del de DDoS de los flujos. Sin embargo, cabe resaltar que los

modelos KNN demostraron superioridad en eficiencia de detección y consumo de recursos de CPU en comparación a los modelos de RF.

De acuerdo al conjunto de experimentos hechos con cada modelo y a los resultados analizados, se concluye que el mejor sistema de detección de DDoS se obtuvo usando una ventana de tiempo de 40 segundos y un clasificador KNN. Bajo estos parámetros se obtuvo la mejor compensación entre precisión, reducción de falsas alarmas y tamaño de ventana de tiempo.

Por otro lado, más que una solución funcional para la detección de DDoS, este trabajo presentó una arquitectura alterna para la implementación de sistemas de detección de DDoS. Esta arquitectura podría ser adoptada para experimentar con diferentes modelos de clasificación entrenados con diferentes tipos de características, sin restricción alguna en el cálculo de ellas en tiempo de ejecución. Con una arquitectura más flexible en la extracción de información de los flujos de la red, es más fácil construir sistemas mas eficientes y generalizados de detección.

6.2. Trabajo Futuro

Como trabajo futuro, prevemos: (i) llevar a cabo experimentos de la solución propuesta en dispositivos físicos que soporten P4 como los Barefoot Tofino [64] o NetFPGA [65]; (ii) idear en el plano de datos una estructura de datos más eficiente (potencialmente basada en BloomFilter [66]) que pueda almacenar la información de cada flujo de manera más compacta (sin incurrir en colisiones y, por lo tanto, comprometer la precisión); (iii) explorar la solución propuesta con diferentes conjuntos de características y modelos de ML; e (iv) implementar un sistema de detección que permita detectar diversos tipos de ataques, no solo DDoS, utilizando descriptores que se calculen en el plano de datos.

Bibliografía

[1] C. Douligeris y A. Mitrokotsa, «DDoS attacks and defense mechanisms: classification and state-of-the-art», *Computer Networks*, vol. 44, n.º 5, págs. 643-666, 2004.

[2] S. Weisman, *What is a distributed denial of service attack (DDoS) and what can you do about them?*, `https://us.norton.com/internetsecurity-emerging-threats-what-is-a-ddos-attack-30sectech-by-norton.html`, Accedido 25-04-2020, Norton, 2019.

[3] R. Ottis, «Analysis of the 2007 cyber attacks against estonia from the information warfare perspective», en *Proceedings of the 7th European Conference on Information Warfare*, 2008, pág. 163.

[4] A. G. Oleg Kupreev Ekaterina Badovskaya, *DDoS attacks in Q1 2019*, `https://securelist.com/ddos-report-q1-2019/90792/`, Accedido 25-04-2020, kaspersky, 2019.

[5] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky y S. Uhlig, «Software-defined networking: A comprehensive survey», *Proceedings of the IEEE*, vol. 103, n.º 1, págs. 14-76, 2014.

[6] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras y G. Gu, «Flow wars: Systemizing the attack surface and defenses in software-defined networks», *IEEE/ACM Transactions on Networking*, vol. 25, n.º 6, págs. 3514-3530, 2017.

[7] J. M. Dover, «A denial of service attack against the Open Floodlight SDN controller», *Dover Networks, Tech. Rep.*, 2013.

[8] A. Nygren, B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn, D. Malek, D. Talayco, D. Erickson y col., «Openflow switch specification version 1.5. 1», *Open Networking Foundation, Tech. Rep.*, 2015.

[9] G. A. Ajaeiya, N. Adalian, I. H. Elhajj, A. Kayssi y A. Chehab, «Flow-based intrusion detection system for SDN», en *2017 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2017, págs. 787-793.

[10] A. S. da Silva, J. A. Wickboldt, L. Z. Granville y A. Schaeffer-Filho, «ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN», en *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2016, págs. 27-35.

[11] I. Sharafaldin, A. H. Lashkari y A. A. Ghorbani, «Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.», en *ICISSP*, 2018, págs. 108-116.

[12] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese y D. Walker, «P4: Programming Protocol-independent Packet Processors», *SIGCOMM Comput. Commun. Rev.*, vol. 44, n.º 3, págs. 87-95, jul. de 2014, ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. dirección: http://doi.acm.org/10.1145/2656877.2656890.

[13] N. Feamster, J. Rexford y E. Zegura, «The road to SDN: an intellectual history of programmable networks», *ACM SIGCOMM Computer Communication Review*, vol. 44, n.º 2, págs. 87-98, 2014.

[14] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown y S. Shenker, «NOX: towards an operating system for networks», *ACM SIGCOMM Computer Communication Review*, vol. 38, n.º 3, págs. 105-110, 2008.

[15] F. Hu, Q. Hao y K. Bao, «A survey on software-defined network and openflow: From concept to implementation», *IEEE Communications Surveys & Tutorials*, vol. 16, n.º 4, págs. 2181-2206, 2014.

[16] L. Yang, R. Dantu, T. Anderson y R. Gopal, «Forwarding and control element separation (ForCES) framework», RFC 3746, April, inf. téc., 2004.

[17] *Open vSwitch Database Management Protocol (OVSDB)*, https://tools.ietf.org/html/rfc7047.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker y J. Turner, «OpenFlow: enabling innovation in campus networks», *ACM SIGCOMM Computer Communication Review*, vol. 38, n.º 2, págs. 69-74, 2008.

[19] A. Lara, A. Kolasani y B. Ramamurthy, «Network innovation using openflow: A survey», *IEEE communications surveys & tutorials*, vol. 16, n.º 1, págs. 493-512, 2013.

[20] F. Hu, Q. Hao y K. Bao, «A survey on software-defined network and openflow: From concept to implementation», *IEEE Communications Surveys & Tutorials*, vol. 16, n.º 4, págs. 2181-2206, 2014.

[21] W. L. da Costa Cordeiro, J. A. Marques y L. P. Gaspary, «Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management», *Journal of Network and Systems Management*, vol. 25, n.º 4, págs. 784-818, 2017.

[22] *P4-16 Language Specification version 1.2.0*, `https://p4.org/p4-spec/docs/P4-16-v1.2.0.html`, Accedido 17-04-2020, 2019.

[23] M. Moshref, M. Yu y R. Govindan, «Resource/accuracy tradeoffs in software-defined measurement», en *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, págs. 73-78.

[24] S. Geravand y M. Ahmadi, «Bloom filter applications in network security: A state-of-the-art survey», *Computer Networks*, vol. 57, n.º 18, págs. 4047-4064, 2013, ISSN: 1389-1286. DOI: `https://doi.org/10.1016/j.comnet.2013.09.003`. dirección: `http://www.sciencedirect.com/science/article/pii/S1389128613003083`.

[25] *P4Runtime Specification version 1.1.0*, `https://p4.org/p4runtime/spec/v1.1.0/P4Runtime-Spec.pdf`, Accedido 22-04-2020, 2020.

[26] I. Martinez-Yelmo, J. Alvarez-Horcajo, M. Briso-Montiano, D. Lopez-Pajares y E. Rojas, «ARP-P4: deep analysis of a hybrid SDN ARP-Path/P4Runtime switch», *Telecommunication Systems*, vol. 72, n.º 4, págs. 555-565, 2019.

[27] *Protocol Buffers*, `https://developers.google.com/protocol-buffers/`, Accedido 22-04-2020, 2020.

[28] *Framework: gRPC*, `https://grpc.io/docs/guides/concepts/`, Accedido 22-04-2020, 2018.

[29] *HTTP/2*, `https://http2.github.io/`, Accedido 22-04-2020, 2018.

[30] *ONOS*, `https://wiki.onosproject.org/display/ONOS/ONOS`, Accedido 23-04-2020, 2020.

[31] M. Briso-Montiano Marco y col., «Implementación de un switch ARP-Path basado en el lenguaje P4 con capacidades para seguridad perimetral», 2018.

[32] *ONOS+P4 Tutorial Controlling P4 data planes with ONOS*, `http://bit.ly/onos-p4-tutorial-slides`, Accedido 23-04-2020, 2018.

[33] C. Douligeris y A. Mitrokotsa, «DDoS attacks and defense mechanisms: classification and state-of-the-art», *Computer Networks*, vol. 44, n.º 5, págs. 643-666, 2004.

[34] S. T. Zargar, J. Joshi y D. Tipper, «A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks», *IEEE communications surveys & tutorials*, vol. 15, n.º 4, págs. 2046-2069, 2013.

[35] J. Liu, Y. Xiao, K. Ghaboosi, H. Deng y J. Zhang, «Botnet: classification, attacks, detection, tracing, and preventive measures», *EURASIP journal on wireless communications and networking*, vol. 2009, n.º 1, pág. 692 654, 2009.

[36] R. van Loon, *An IRC tutorial*, `http://www.irchelp.org/faq/irctutorial.html`, Accedido 25-04-2020, 1997.

[37] B. Hancock, «Trinity v3, a DDoS tool, hits the streets», *Computers & Security*, vol. 19, n.º 7, págs. 574-574, 2000.

[38] *Kaiten.E. Troyano capaz de realizar ataques DDoS*, `http://www.vsantivirus.com/kaiten-e.html`, Accedido 25-04-2020, 2006.

[39] team-cymru Inc, *A Taste of HTTP Botnets*, `http://www.team-cymru.com/ReadingRoom/Whitepapers/2008/http-botnets.pdf`, Accedido 25-04-2020, 2008.

[40] *Low Orbit Ion Cannon: DDoS Attack Tool*, `https://github.com/NewEraCracker/LOIC/`, 2019.

[41] J. C. C. Chica, J. C. Imbachi y J. F. Botero, «Security in SDN: A comprehensive survey», *Journal of Network and Computer Applications*, pág. 102 595, 2020.

[42] M. Roughan, S. Sen, O. Spatscheck y N. Duffield, «Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification», en *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 2004, págs. 135-148.

[43] T. T. Nguyen y G. Armitage, «A survey of techniques for internet traffic classification using machine learning», *IEEE communications surveys & tutorials*, vol. 10, n.º 4, págs. 56-76, 2008.

[44] *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*, `https://tools.ietf.org/html/rfc3176`.

[45] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz y J. González, «Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN», *Future Generation Computer Systems*, vol. 111, págs. 763-779, 2020.

[46] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras y V. Maglaris, «Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments», *Computer Networks*, vol. 62, págs. 122-136, 2014.

[47] R. Braga, E. Mota y A. Passito, «Lightweight DDoS flooding attack detection using NOX/OpenFlow», en *IEEE Local Computer Network Conference*, IEEE, 2010, págs. 408-415.

[48] Q. Niyaz, W. Sun y A. Y. Javaid, «A deep learning based DDoS detection system in software-defined networking (SDN)», *arXiv preprint arXiv:1611.07400*, 2016.

[49] L. Yang y H. Zhao, «DDoS attack identification and defense using SDN based on machine learning method», en *2018 15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)*, IEEE, 2018, págs. 174-178.

[50] A. Abubakar y B. Pranggono, «Machine learning based intrusion detection system for software defined networks», en *2017 Seventh International Conference on Emerging Security Technologies (EST)*, IEEE, 2017, págs. 138-143.

[51] A. Le, P. Dinh, H. Le y N. C. Tran, «Flexible network-based intrusion detection and prevention system on software-defined networks», en *2015 International Conference on Advanced Computing and Applications (ACOMP)*, IEEE, 2015, págs. 106-111.

[52] A. AlEroud e I. Alsmadi, «Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach», *Journal of Network and Computer Applications*, vol. 80, págs. 152-164, 2017.

[53] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin y Z. Zhang, «Enabling security functions with SDN: A feasibility study», *Computer Networks*, vol. 85, págs. 19-35, 2015.

[54] Â. C. Lapolli, J. A. Marques y L. P. Gaspary, «Offloading real-time DDoS attack detection to programmable data planes», en *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE, 2019, págs. 19-27.

[55] A. S. Da Silva, C. C. Machado, R. V. Bisol, L. Z. Granville y A. Schaeffer-Filho, «Identification and selection of flow features for accurate traffic classification in SDN», en *2015 IEEE 14th International Symposium on Network Computing and Applications*, IEEE, 2015, págs. 134-141.

[56] A. Binbusayyis y T. Vaiyapuri, «Identifying and Benchmarking Key Features for Cyber Intrusion Detection: An Ensemble Approach», *IEEE Access*, vol. 7, págs. 106 495-106 513, 2019.

[57] Y. Liao y V. R. Vemuri, «Use of k-nearest neighbor classifier for intrusion detection», *Computers & security*, vol. 21, n.º 5, págs. 439-448, 2002.

[58] *Intrusion Detection Evaluation Dataset (CICIDS2017)*, `https://www.unb.ca/cic/datasets/ids-2017.html`.

[59] *Mininet*, `http://mininet.org/`.

[60] *BMV2 $simple_switch_grpc$*, `https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc`.

[61] *TCPReplay*, `https://github.com/appneta/tcpreplay`.

[62] *Scikit-Learn*, `https://scikit-learn.org/stable/`.

[63] R. R. Picard y R. D. Cook, «Cross-validation of regression models», *Journal of the American Statistical Association*, vol. 79, n.º 387, págs. 575-583, 1984.

[64] *Conmutadores P4 Barefoot-Tofino*, `https://www.barefootnetworks.com/`.

[65] *NetFPGA*, `https://netfpga.org/site/#/`.

[66] S. Geravand y M. Ahmadi, «Bloom filter applications in network security: A state-of-the-art survey», *Computer Networks*, vol. 57, n.º 18, págs. 4047-4064, 2013.

Apéndice A

Proceso de instalación de una aplicación P4 a través de ONOS

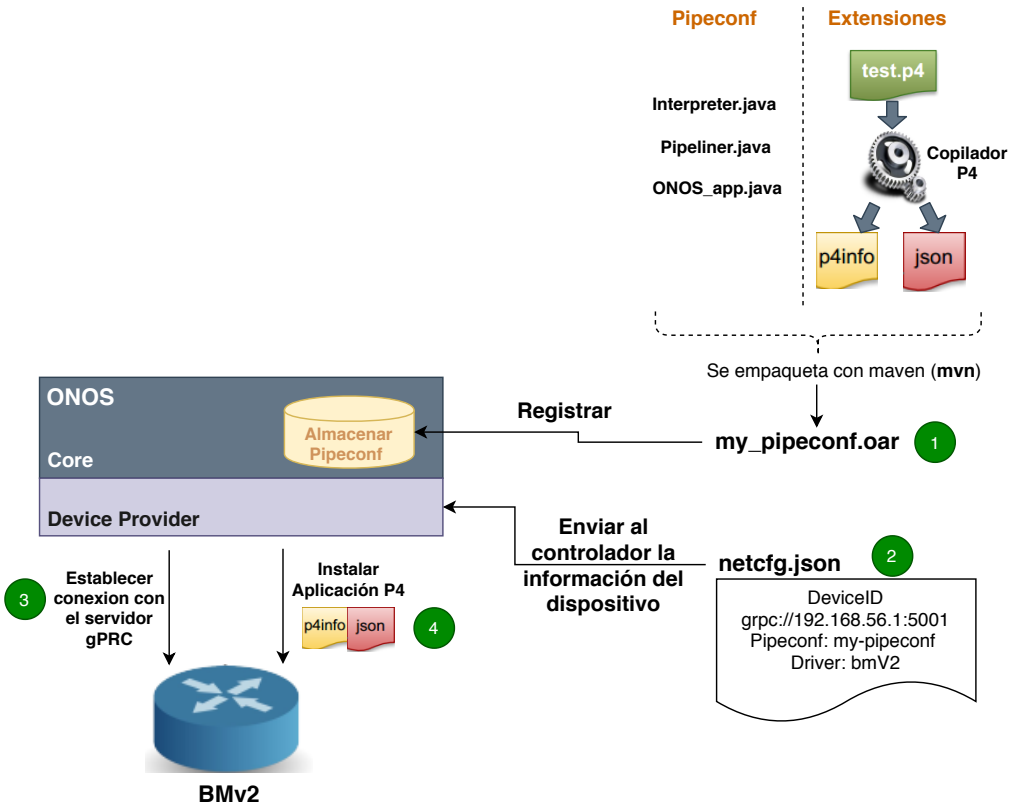
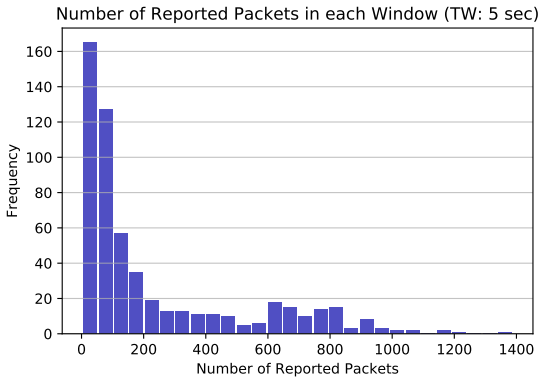


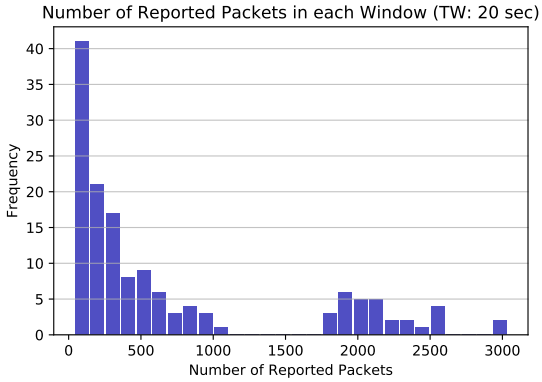
Figura A.1: Despliegue de la aplicación test.p4 a través de ONOS

Apéndice B

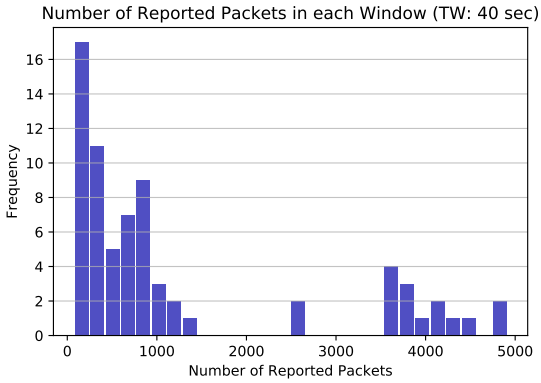
Histograma de paquetes de reporte enviados al Plano de Control



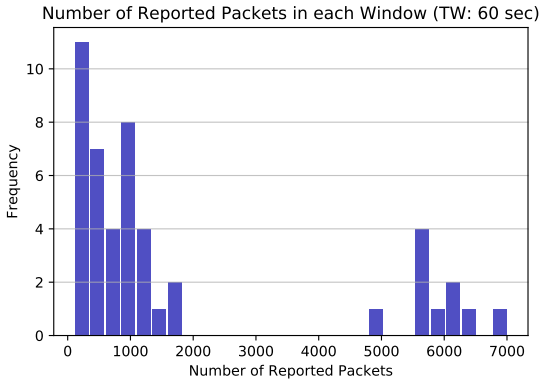
(a) Histograma del numero de paquetes reportados en cada ventana de tiempo de **5 segundos** dentro del experimento.



(b) Histograma del numero de paquetes reportados en cada ventana de tiempo de **20 segundos** dentro del experimento.



(c) Histograma del numero de paquetes reportados en cada ventana de tiempo de **40 segundos** dentro del experimento.



(d) Histograma del numero de paquetes reportados en cada ventana de tiempo de **60 segundos** dentro del experimento.

Figura B.1: Histograma resultante al experimentar con cada valor de ventana de tiempo

Tabla B.1: Cantidades Minima, Maximas y Media de paquetes reportados en ventanas de tiempo de cierta duración, durante el experimento.

Ventana de Tiempo	**Mínima cantidad de paquetes reportados**	**Máxima cantidad de paquetes reportados**	**Media de paquetes reportados**
5 seg.	3	1388	234
20 seg.	38	3036	695
40 seg.	73	4912	1260
60 seg.	103	7010	1814

Apéndice C

Repositorio de la implementación

https://github.com/sebitas0623/ORACLE_ddos