UNIVERSIDAD DE ANTIOQUIA

MAESTRÍA EN INGENIERÍA DE TELECOMUNICACIONES

Trabajo de Investigación

MODELADO Y VALIDACIÓN FORMAL DE UNA TOPOLOGÍA SDN/OPENFLOW.

Autor

SEBASTIAN CASTRILLÓN OSPINA

Directora

NATALIA GAVIRIA GÓMEZ

Medellín, Colombia
2016

Agradecimientos

A todos los que me apoyaron durante los últimos años y la paciencia que me tuvieron. Agradezco a mi padre Henry Castrillon y mi madre Beatriz Ospina por darme la oportunidad de emprender el camino profesional con mi pregrado, primer paso para lograr continuar con mis estudios. A la profesora Natalia Gaviria persona que aprendí a admirar bastante, por la paciencia y la enseñanza que me deja. A mi socio John Sosa por apoyarme en terminar este proyecto aun cuando tenemos los suficientes en nuestra empresa. Al profesor Juan Felipe Botero quien con su conocimiento puso las primeras bases de este proyecto. A Héctor Flórez de la Facultad de Minas de la Universidad Nacional de Colombia, por su asesoría constante.

Contenido

1. INTRODUCCIÓN

Las redes de datos tradicionales han sido adoptadas alrededor del mundo permitiendo el intercambio de altas cantidades de información, esto ha hecho que las redes sean complejas y difíciles de administrar. El plano de control (que decide cómo para manejar el tráfico de red) y el plano de datos (que remite tráfico de acuerdo con las decisiones tomadas por el plano de control) están agrupados dentro de los dispositivos de red, reduciendo la flexibilidad y obstaculizando la innovación y la evolución en las actuales infraestructuras de red.

La llegada de las redes definidas por software (SDN) es un cambio sustancial en la forma en cómo se administran y trabajan las redes actuales. SDN es un paradigma que separa el plano de datos y el plano de control del dispositivo de red, lo cual se logra agregando nuevas funcionalidades de red simples a los equipos actuales. En esta nueva arquitectura, un servidor actúa como controlador administrando las tareas del plano de control y tomando las decisiones para conmutar, enrutar y filtrar los paquetes.

Los sistemas centralizados son fáciles de coordinar, pero existe la probabilidad de errores en la implementación y el funcionamiento de las topologías y los protocolos de red. Aunque existen especificaciones detalladas del protocolo OpenFlow el cual permite el despliegue de SDN, se presentan comportamientos inesperados en las diferentes topologías y arquitecturas de red, esto principalmente se debe a la falta de especificaciones formales de la secuencia de eventos que ocurren en los componentes SDN como el controlador y el switch.

Durante las dos últimas décadas se han implementado técnicas de modelado formal para verificar estándares y protocolos comunes de red como: OSPF, STP, SIP, PPPoE entre otros. El modelado formal permite un entendimiento adecuado del sistema, establecer las especificaciones relacionadas con sus funcionalidades y realizar análisis detallados del comportamiento en condiciones específicas.

El análisis por eventos discretos se ha convertido en una herramienta bastante útil para el modelado de protocolos de red y el consecuente análisis de comportamiento y desempeño. Las técnicas de eventos discretos, como autómatas finitos y redes de Petri han sido ampliamente utilizadas como método formal apropiado para este fin [1][2]. Las características principales de los sistemas de eventos discretos son: concurrencia, asincronía, manejo de recursos compartidos y exclusión mutua. Adicionalmente se cuenta con herramientas de simulación de alto desempeño, que contribuyen a la validación y el análisis de los modelos propuestos.

Una de las ventajas de tener modelos formales es permitir la realización de pruebas analíticas sobre el modelo, para obtener resultados en poco tiempo. Conseguir modelos eficientes que permitan hacer una aproximación real del funcionamiento de una topología SDN/OpenFlow, requiere de una técnica de modelado que permita

incluir procesos concurrentes, estocásticos, dinámicas discretas, y además adaptabilidad.

Las redes de Petri coloreadas (CPN) han sido ampliamente utilizadas como método formal apropiado para modelar redes de comunicaciones. Trabajos realizados previamente muestran las CPN como una herramienta adecuada para modelar y analizar protocolos de red comunes como SIP[3], PPPoE[4], OSPFv3[5] y LDP[6]. Se destacan algunos aprovechamientos de CPN como lo describe [3] donde se realiza un modelo del comportamiento del protocolo SIP resaltando la utilidad de CPN para el diseño y análisis de los comportamientos deseados de los protocolos antes de su implementación. En [4] se utiliza el modelo para entender en detalle la ocurrencia de eventos en el establecimiento de una conexión PPPoE. En [5] se realiza un análisis del comportamiento de un modelo del protocolo OSPFv3 y sus propiedades mediante el análisis del espacio de estados generado de la simulación. Estos modelos aprovechan la disponibilidad de herramientas de simulación de alto desempeño de las CPN y las características de los sistemas de eventos discretos mencionadas anteriormente.

1.1. Planteamiento del problema

Las redes actuales están creciendo a una velocidad donde la administración se vuelve engorrosa y complicada al momento de operarlas, mantenerlas y asegurarlas. Es por esto que el concepto de administración de la red está cambiando a nivel mundial.

Las redes definidas por software (SDN) se definen como el futuro de Internet al permitir separar el plano de control del plano de envío de los datos de la red, donde el plano de control, mediante un controlador basado en software, administra múltiples dispositivos de red asignándoles políticas definidas para el tratamiento de los flujos de datos. SDN es una arquitectura emergente que es dinámica y de bajo costo, siendo ideal para altos anchos de banda, dinámica natural de las aplicaciones actuales. El protocolo OpenFlow es el elemento fundamental y primer estándar para la implementación de soluciones SDN al hacer realidad la comunicación entre el equipo de red (Plano de datos) y el controlador SDN (Plano de control).

El cambio de paradigma con SDN ha generado nuevos retos, siendo objeto de estudio de diferentes grupos de investigación alrededor del mundo en los últimos años. En particular, dada la relevancia que adquiere el controlador y el protocolo Openflow, la predicción de su desempeño ha generado alto interés investigativo.

Las herramientas de modelamiento de sistemas dinámicos a eventos discretos (DEDS) son ampliamente utilizadas para realizar modelos de protocolos y dispositivos de telecomunicaciones, permitiendo analizar y comprender el detalle del comportamiento mediante simulación y validación formal y funcional.

¿Es posible utilizar estrategias de eventos discretos como las Redes de Petri (PN) o Redes de Petri coloreadas (CPN) para modelar una topología básica SDN/OpenFlow que permita identificar y probar el funcionamiento de la arquitectura formalmente, y adicionalmente analizar el comportamiento de cada uno de los componentes principales de una red SDN?

1.2. Objetivos

1.2.1. Objetivo General

Proponer y desarrollar un modelo basado en Redes de Petri Coloreadas para la especificación formal y la validación de una topología básica SDN/OpenFlow.

1.2.2. Objetivos específicos

- Identificar los principales atributos y entidades, tanto dinámicas, como estáticas que intervienen en el flujo de información de los bloques que componen el modelo.
- Validar las propiedades de vivacidad, alcanzabilidad y libre de lazos del modelo mediante el árbol de estados.
- Diseñar e implementar casos de prueba para simular, analizar, evaluar y realizar ajustes a los modelos desarrollados.
- Definir los parámetros adecuados para validar el modelo con una red SDN/OpenFlow experimental.
- Implementar un experimento que permita validar el modelo propuesto mediante pruebas de desempeño cualitativo y cuantitativo mediante los parámetros de rendimiento definidos en el punto anterior.

1.3. Estructura del documento

En el capítulo 2 se presenta un análisis del estado del arte con los artículos más relevantes asociados al proyecto.

En el capítulo 3 se realiza una descripción clara de los conceptos claves de la arquitectura de Redes Definidas por Software SDN y el protocolo OpenFlow.

En el capítulo 4 se resume los conceptos de Redes de Petri y Redes de Petri Coloreadas, describiendo los componentes principales y mostrando ejemplos para dar claridad a las definiciones.

En el capítulo 5 se muestra el primer modelo adecuado para el análisis formal con tiempos determinísticos.

En el capítulo 6 se modifica el modelo para el análisis experimental con tiempos determinísticos y validado con un experimento de laboratorio.

En el capítulo 7 se realizan las conclusiones del trabajo y se plantean los trabajos futuros.

2. ESTADO DEL ARTE

Para realizar un análisis del estado actual de los diferentes verticales que soportan este proyecto, en la Tabla 1 se muestran de forma tabulada los principales artículos y autores separados en columnas de acuerdo al alcance de dicho trabajo, para lograr un rápido y óptimo. El análisis realizado busca definir qué tipo de red, protocolo o topología se modeló, que estrategia de modelamiento se implementó y el alcance formal y experimental que se usó para validar lo realizado en cada proyecto.

Articulo	Estrategia de modelado	Tipo de red	Validación	Anotación
H. Abdul Rauf[1] Colored Petri Net Modeling And Throughput Analysis For Wireless Infrastructure Networks	CPN	Wireless LAN	Experimental, no se identifica validación de las propiedades del modelo o del espacio de estados	Se realiza el modelo de un Access Point, un switch y una estación de trabajo, adicionalmente se realiza la conversión de unidades de tiempo a unidades de CPN para realizar la validación.
Barzegar, 2011[2] Modeling and Simulation Firewall Using Colored Petri Net	CPN	Listas de control de acceso	No se realiza	Se realiza un modelo en CPN que describe formalmente el conjunto de eventos del proceso de verificación de Listas de Control de Acceso (ACL) de un firewall. Adicionalmente realiza el modelo de un switch, un servidor y una estación de trabajo para completar la topología propuesta. El autor no realiza ninguna validación experimental.
V. Gehlot and C. Nigro, 2010 [3] Colored Petri Net model of the Session Initiation Protocol (SIP)	CPN	SIP	Experimental, no se identifica validación de las propiedades del modelo o del espacio de estados	Se realiza un modelo en CPN del comportamiento del protocolo SIP resaltando la utilidad de CPN para el diseño y análisis de los comportamientos deseados de los protocolos antes de su implementación.
Xiao-jing, 2010 [4] Analysis and verification of colored petri net in pppoe protocol	CPN	PPPoE	Propiedades de CPN, No se realiza validación experimental.	El autor diseña el modelo en CPN para el proceso de descubrimiento de PPPoE entre routers. El modelo es analizado mediante el espacio de estados y el árbol de alcanzabilidad para verificar sus propiedades principales. Adicionalmente realiza la descripción detallada de cada uno de los lugares del modelo y las variables utilizadas.
Wang, 2003 [5] OSPFv3 protocol simulation with colored Petri nets	CPN	OSPF	Propiedades de CPN con análisis del espacio de estados, no se realiza validación experimental.	El autor diseña el modelo en CPN para el proceso de intercambio de mensajes Link state de OSFPv3 entre routers. El modelo es analizado mediante la metodología de espacio de estado para verificar el cumplimiento de las propiedades del modelo. Se presenta el modelo de una forma bastante detallada.

R. Sa, Y. Zhao, and C. Hai, 2010[6] Research on the LDP Protocol Verification Based on Coloured Petri Nets.	CPN	Label Distribution Protocol (LDP)	Análisis de las propiedades de CPN mediante el espacio de estados, No se realiza validación experimental.	Se realiza el modelo en CPN del intercambio de mensajes de LDP basado en la RFC 3036, se muestra en detalle el modelo general, las variables en CPN tool y el resultado del análisis de las propiedades del espacio de estados.
Aliannezhadi, Zobeideh Azgomi, Mohammad Abdollahi[7] Modeling and Analysis of a Web Service Firewall Using Coloured Petri Nets	CPN	Firewall	Análisis de las propiedades de CPN mediante el espacio de estados, No se realiza validación experimental.	El articulo muestra en detalle cada uno de los lugares y transiciones del modelo del firewall, lo cual es de gran utilidad para el desarrollo de nuevos modelos.
Jarschel, 2011[8] Modeling and Performance Evaluation of an OpenFlow Architecture	Teoria de colas	SDN/OPENFLOW	Validación experimental.	El artículo propone un modelo matemático para el análisis del rendimiento de una topología básica SDN con un switch y un controlador mediante teoría de colas. Este modelo permite el análisis del rendimiento incluyendo la probabilidad de perdida de paquetes en el buffer del controlador. Se realiza, además, un montaje físico experimental para definir los tiempos de procesamiento del switch y del controlar SDN.
Kang, 2013 [9] Formal Modeling and Verification of SDN-OpenFlow	Teoría de grafos	SDN/OPENFLOW	Propiedades de los grafos y rendimiento del modelo traducido a python	Se propone un modelo basado en teoría de grafos para describir y simular el intercambio de los mensajes Openflow en una topología básica SDN. La validación formal se realiza mediante el análisis de las propiedades de los grafos y la estrategia propuesta por el autor llamada VERSA, que realiza la traducción del modelo a Python y la validación de su funcionamiento.
Sethi, Divjyot Narayana, Srinivas Malik, Sharad, 2013[10] Abstractions for model checking SDN controllers	Murphi Model	SDN/OPENFLOW	Validación experimental	Se realiza el modelo y validación experimental con un Switch y un firewall mediante la estrategia Murphi, esta estrategia de modelamiento es interesante para la validación de protocolos de seguridad. http://seclab.stanford.edu/pcl/mc/mc.html.

Antonio, 2014[11] Modelling and Verification of Security Rules in an OpenFlow Environment with Coloured Petri Nets	CPN	SDN/OPENFLOW	No se realiza	El autor presenta el modelo de un firewall SDN para el procesamiento de paquetes con coincidencia en listad de control de acceso. El autor no realiza validación formal o experimental. Aunque el autor propone el funcionamiento de un firewall los estados modelados son muy generales y se debe profundizar más en el modelo para representar adecuadamente el proceso.
Dong, 2013[12] Testing OpenFlow interaction property based on hierarchy CPN	CPN	SDN/OPENFLOW	Análisis de espacio de estados y experimental	El autor propone el modelo para el Switch y el controlador SDN en CPN, además de modelar el Canal de comunicación entre estos. Aunque el artículo no describe mucho sobre la validación formal se identifica que se realizó un análisis del espacio de estado y un montaje experimental. El modelo del switch es poco entendible y falta mucho alcance en su diseño para que asemeje el funcionamiento real de los diferentes dispositivos SDN
Koizumi, Seiichi Fujiwaka, Masaya, 2015[13] SDN + cloud integrated control with statistical analysis and discrete event simulation	CPN	SDN/OPENFLOW	Validación experimental	Aunque no se realiza una descripción detallada del modelo en CPN se identifica que esta herramienta de modelación está siendo utilizada por grandes empresas como NEC en Japón para apoyar sus proyectos de investigación

Tabla 1: Estado del arte

Después de verificar los artículos más relevantes asociados a la temática de este proyecto, se concluye que el proyecto tiene las bases para clasificar la temática SDN como actual en el medio investigativo y comercial. Adicionalmente, se concluye que basado en todos los trabajos asociados a telecomunicaciones que fundamentan su base formal y matemática en las estrategias de eventos discretos como Redes de Petri Coloreadas, le da la viabilidad para tener un modelo optimo y flexible en CPN para realizar simulaciones que permitan ser acogidas por futuros trabajos de la comunicad investigativa y comercial.

3. FUNDAMENTOS DE LAS REDES DEFINIDAS POR SOFTWARE.

El concepto de SDN fue introducido por primera vez por Nick McKeown [14], profesor de la Universidad de Stanford. El objetivo de este nuevo paradigma es trasladar las decisiones de los dispositivos de red a un elemento central, llamado controlador, buscando reducir la complejidad de la administración de las redes y mejorando de manera simultánea el rendimiento de las mismas.

Las redes de datos tradicionales están diseñadas para que los equipos tengan propósitos específicos e implementen protocolos, control de acceso y monitoreo de forma independiente. Estos equipos tienen integrado el plano de control y el plano de datos, el administrador de la red debe configurar políticas específicas en cada uno de los equipos de la red[15], haciendo la administración compleja y el rendimiento limitado a cada equipo sin permitir una visibilidad completa de la red.

SDN ha ganado terreno en las aplicaciones de ámbito empresarial y comercial gracias a la adopción de las nuevas tecnologías en la nube y el concepto de redes virtuales [8], permitiendo que grandes fabricantes se interesen en crear compatibilidad de sus equipos con esta nueva tecnología y desarrollar controladores con características innovadoras.

3.1. Arquitectura SDN

En la Figura 1 se muestra un modelo básico de la arquitectura SDN basado en capas. Se observa que la inteligencia de la red está centralizada (lógicamente) en el controlador SDN, el cual mantiene una vista global de la red tomando todas las decisiones de procesamiento de los flujos de datos[16] desde las aplicaciones en la capa superior. OpenFlow se ha convertido en el protocolo estándar para permitir la comunicación entre el plano de control y el plano de datos, por lo que será un tema de estudio que se tratará durante el desarrollo de este trabajo.

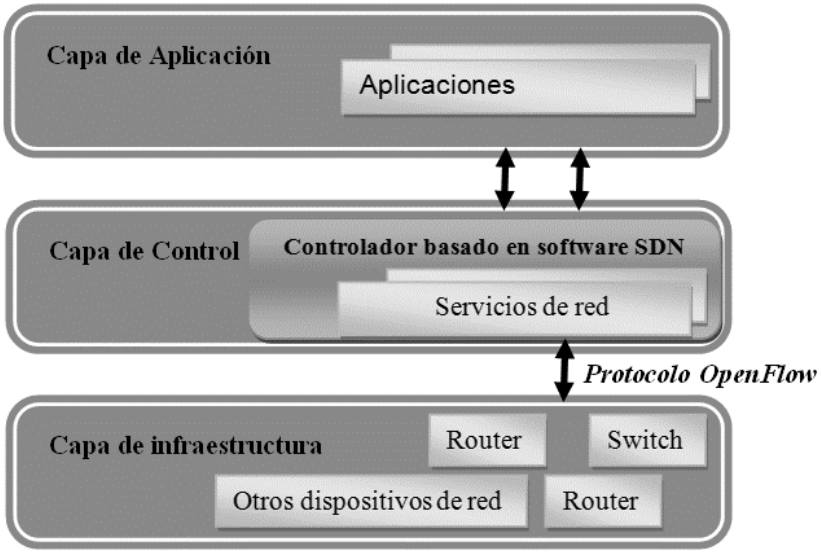


Figura 1: Arquitectura SDN

3.1.1. Capa de infraestructura

La capa de infraestructura o plano de datos se divide en la infraestructura física de red y la denominada subcapa Southbound. En la infraestructura física de red se encuentran todos los dispositivos compatibles con SDN [17]. La compatibilidad ha crecido exponencialmente en los últimos dos años, incluyendo los principales fabricantes de red de todo el mundo. La capa Southbound funciona con interfaz entre el plano de datos y el plano de control, alojando los protocolos que permiten el intercambio de información entre éstos: Openflow, SNMP, NETCONF y otros protocolos que operan en esta subcapa.

3.1.1.1. Openflow

El protocolo OpenFlow se ha convertido en el protocolo estándar de SDN, utilizado como interfaz entre los dispositivos de la infraestructura de red y el software que funciona como controlador SDN. Actúa en la subcapa Southbound, funcionando como canal de comunicaciones entre el plano de datos y el plano de control. OpenFlow usa el concepto de flujos para identificar el tráfico de la red con base en un conjunto de reglas de coincidencia predefinidas, que pueden ser estática o dinámicamente programadas por el controlador SDN. En la Figura 2 se observa un ejemplo de la tabla de flujos de los dispositivos OpenFlow: el controlador es el encargado de agregar o modificar cada una de las reglas o políticas de la tabla, cada regla identifica el flujo y ejecuta la acción requerida con los paquetes que coincidan. Esto ayuda al administrador a definir cómo el tráfico debería fluir a través de los dispositivos de red con base en parámetros como patrones de uso, aplicaciones y recursos de la red. OpenFlow busca mejorar los tiempos de respuesta de la red y del administrador ante los cambios de comportamiento en aplicaciones, usuarios y sesiones. Actualmente, el enrutamiento basado en IP no

permite control de este nivel, debido a que todos los flujos entre dos puntos finales normalmente fluyen por el mismo camino a través de la red, independientes de los diferentes requerimientos del flujo.[16]

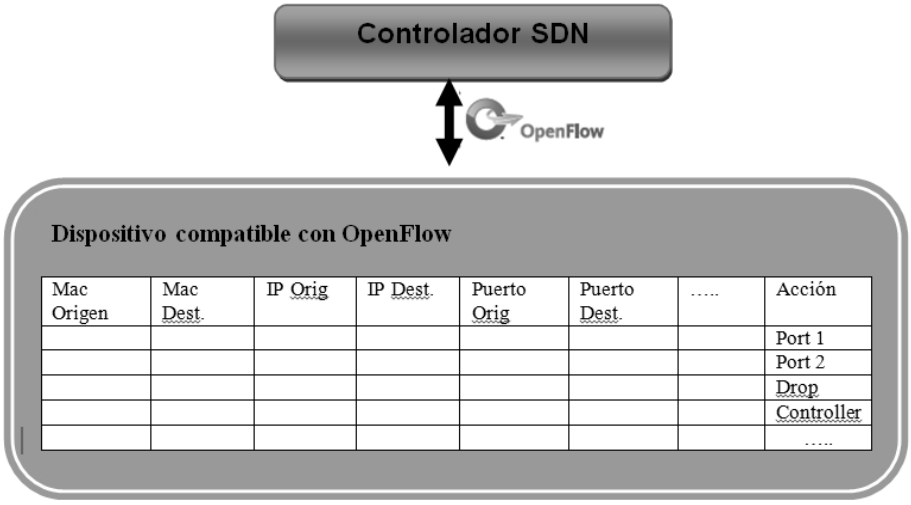
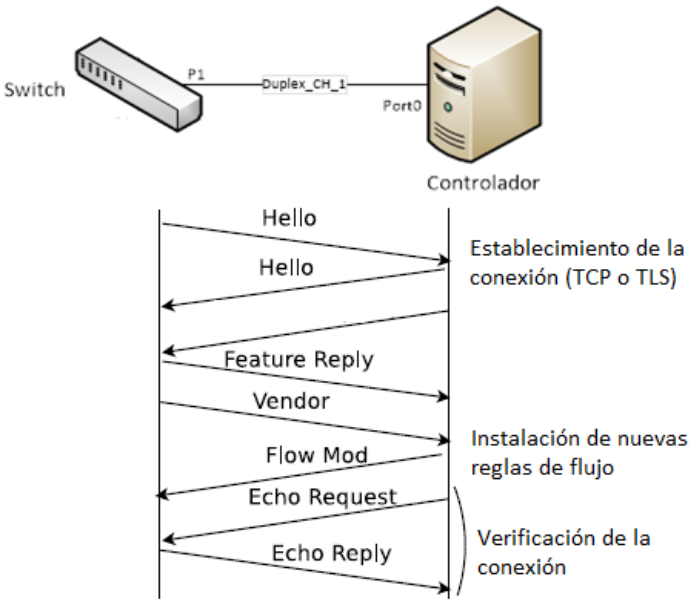


Figura 2: Protocolo OpenFlow



En la Figura 3 se presenta el intercambio de mensajes básico entre un controlador y un switch compatible con OpenFlow. El controlador OpenFlow está configurado para escuchar el switch mediante conexiones TLS o TCP, TLS es preferido para establecer un canal seguro y cifrado. El primer paso es intercambiar mensajes *Hello* simétricos para iniciar y establecer la conexión. El controlador entonces envía un mensaje *Feature request* al cual el switch responde con un *Feature replay,* donde incluye información de las características físicas y lógicas soportadas por el switch e información adicional sobre el fabricante [18]. Los switch basan su funcionamiento en la tabla de flujos, la cual contiene entradas que determinan que campos deben coincidir con un paquete que ingresa para tomar la acción definida por el

controlador. Cuando un paquete ingresa y no hay coincidencia con alguna de las entradas en la tabla de flujos el switch envía al controlador una copia del primer mensaje encapsulada en el protocolo OpenFlow, a lo cual el controlador responde con un mensaje *Flow-mod* que adiciona una nueva entrada en la tabla de flujos, un ejemplo de una entrada en la tabla de flujos se muestra en la Figura 4. Adicionalmente OpenFlow define un mecanismo de mantenimiento del enlace mediante mensajes Echo, similar a ICMP, para identificar que el canal establecido permanezca activo y funcional para la comunicación entre controlador y switch.

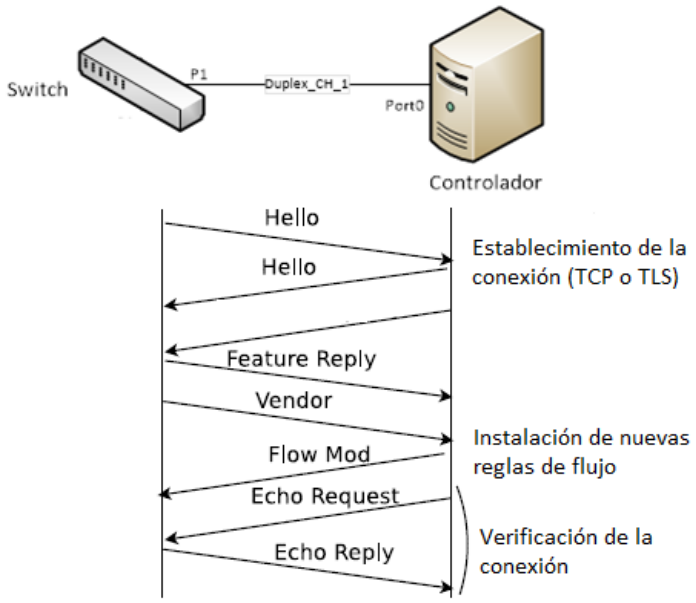


Figura 3: Intercambio de mensajes OpenFlow [18]

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:2E:..	00:1F:.	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	Fwd to port6

Figura 4: Ejemplo de tabla de flujos

El protocolo OpenFlow está en desarrollo constante para dar mayor compatibilidad y mejorar las características propuestas en la primera versión declarada como estable Openflow v1.0; cada versión agrega nuevos campos y características para hacer más flexible la identificación del flujo. La Tabla 2 muestra de forma tabulada un resumen de las diferentes versiones actuales.

Versión	Descripción
Openflow 1.0	Es la versión más usada en trabajos de investigación y en la compatibilidad con dispositivos. El protocolo define los campos principales para la coincidencia de los flujos con las reglas definidas en la tabla Openflow (Puerto de ingreso, campos Ethernet, campos IPv4, puertos TCP/UDP) con se muestra en la Figura 4.
Openflow 1.1	La principal característica adicional es la compatibilidad con tecnologías WAN MPLS usadas en los proveedores de

	servicio, esto permite agregar a las reglas de la tabla de flujos los campos: label y traffic class.
Openflow 1.2	Agrega los campos para la compatibilidad básica con flujos IPv6 adicionando los campos como origen, destino, flow label e ICMPv6 en la tabla de flujos
Openflow 1.3	Mejora la compatibilidad con IPv6 y adiciona estadísticas detalladas a los Flujos transmitidos. Los fabricantes han trabajado bastante para lograr la compatibilidad con esta versión haciendo upgrade de sus controladores y dispositivos de la versión 1.0 a la versión 1.3. HP ha sido uno de los grandes colaboradores desde 2007 trabajando de la mano con la Universidad de Stanford, logrando actualmente la compatibilidad de más de 40 modelos de switch con la versión 1.3 [19].
Openflow 1.4	En esta versión se agregan campos adicionales para lograr la compatibilidad con redes ópticas. Estos campos adicionales permiten la configuración y el monitoreo de la transmisión y recepción de frecuencias laser [20].
Openflow 1.5	Entre las características adicionales que incluye esta versión se resalta las tablas de egreso, agregando un procesamiento adicional que permite definir reglas y tablas de flujos que analizan el puerto de salida y no solo el puerto de ingreso como se trabaja en las versiones anteriores [21].

Tabla 2: Versiones de Openflow

3.2. Capa de control

En esta capa se encuentra el controlador SDN, el cual facilita la administración y la operación de la red. Este permite configurar de manera ágil múltiples equipos evitando la creación de políticas individuales en la capa de infraestructura.

La capa de control se divide en 3 partes que definen: La virtualización de redes como objetivo principal de SDN, el controlador como elemento central de una topología SDN y la comunicación con la capa de administración.

3.2.1. Virtualización de redes

La virtualización de redes permite la combinación de los recursos disponibles y la administración eficiente de los mismos. Esta combinación de recursos permite la división de los canales con ancho de banda disponible, en canales independiente para cada red virtual, estos pueden ser fácilmente reasignados en tiempo real a dispositivos como routers, switchs y máquinas virtuales, entre otros equipos de red. El uso de la tecnología de virtualización reduce los costos totales de inversión en equipos y enlaces al compartir la infraestructura de red [22][23]. En la
Figura **5** se muestra la arquitectura de una red virtual:

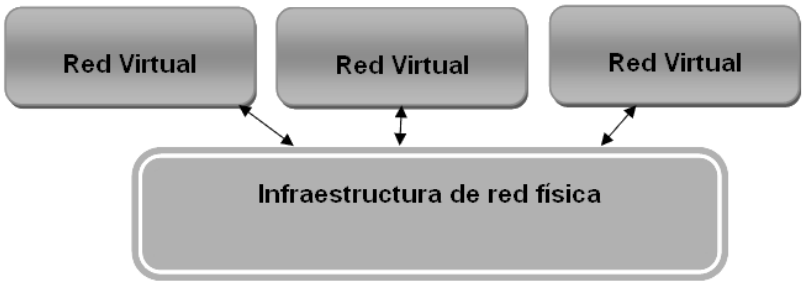


Figura 5: Virtualización de redes

La base de la virtualización de redes son los Network Hypervisor. Estos son software que se ubican de manera lógica entre el controlador y los dispositivos de red como un proxy, su tarea es lograr un modelo de independencia y aislamiento de los servicios, las topologías y la configuración de cada uno de los inquilinos virtuales que utilicen la infraestructura física [24].

3.2.2. Controlador

El controlador es el elemento más crítico e importante de la arquitectura SDN. Sus tareas básicas incluyen el descubrimiento de los dispositivos, la información de la topología de red y la distribución de la configuración y políticas definidas por el administrador de la red y las aplicaciones. Los controladores actúan como la inteligencia de los dispositivos al asumir toda la responsabilidad del plano de control, tomando las decisiones de conmutación en las capas 2, 3 y 4 del modelo OSI, dependiendo del dispositivo y las políticas establecidas en las tablas de flujos.

A medida que una red aumenta su tamaño, puede requerir características adicionales en el controlador. Estas características buscan administrar altos flujos de solicitudes y además permitir la tolerancia a fallas mediante la utilización de múltiples controladores que evitan un único punto de falla en la red. Actualmente, los desarrolladores de los controladores están incluyendo características como Clustering y la capacidad de tener múltiples controladores distribuidos para lograr arquitecturas con plano de control resiliente [25].

En Figura 6 se muestra la comparación de los principales controladores actuales, analizando las siguientes características tomadas de la página web de cada uno de los proyectos:

Lenguaje de programación: Los lenguajes utilizados por los desarrolladores son principalmente C, C++, Python y JAVA. Algunos proyectos han usado Ruby pero no es ampliamente usado.

Arquitectura: Como se indica anteriormente en este documento, se busca que los controladores sean resilientes y permitan escalabilidad. Por esto se definen si son distribuidos o centralizados y si admiten procesamiento multihilo para aumentar su rendimiento.

Versión de OpenFlow: Define la compatibilidad de cada controlador con las diferentes características asociadas a cada versión.

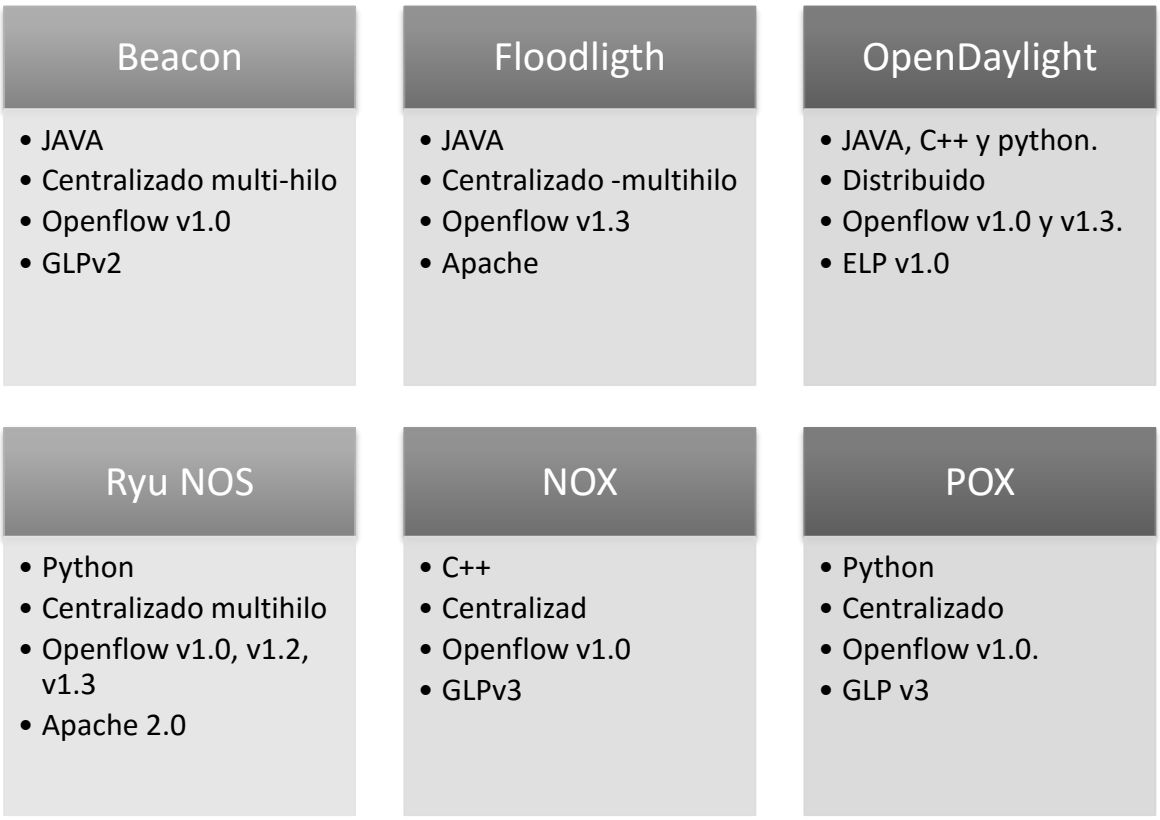


Figura 6: Controladores SDN

Actualmente, los principales fabricantes han adoptado controladores basados en Java, como lo han hecho HP (con su controlador HP VAN SDN) y Cisco (con CiscoONE). Adicionalmente, Juniper desarrolló el controlador OpenContrail [26] que permite programación multilenguaje en C++, Python y Java.

3.2.3. Northbound

Esta subcapa de la capa de control tiene la tarea de funcionar como interfaz de comunicación con la capa de administración, donde se ejecutan los servicios y las aplicaciones de la red. A diferencia de la subcapa Southbound, donde se ha consolidado Openflow como estándar de comunicación, en northbound los diferentes proyectos investigativos y fabricantes definen su compatibilidad basada en el controlador con el cual es compatible y el lenguaje de programación de las aplicaciones soportadas[27]. Se espera que pronto se estandarice alguna Application Programming Interface (API) propuesta por los desarrolladores para evitar que las aplicaciones de red dependan de la compatibilidad de esta interfaz y

lograr toda la flexibilidad que la arquitectura SDN tiene como objetivo en las redes futuras.

Algunos lenguajes de programación usados en SDN para el desarrollo de aplicaciones y una descripción de su propósito en la red puede observarse en la Tabla 3**.**

Lenguaje de programación	Propósito
Pyretic	Permite a los programadores y operadores de red escribir aplicaciones de red modulares proporcionando abstracciones potentes de los planos de control y de datos. [28]
Procera	Incorpora un alto nivel de abstracción que facilita el análisis de los comportamientos temporales y reactivos.
Merlin	Permite mecanismos para delegar la administración de sub-políticas a las redes virtuales huéspedes sin violar las restricciones globales[29]
Maple	Permite la implementación de soluciones multicore con alta eficiencia y escalables, con controladores de más de 40 core.

Tabla 3: Lenguajes de programación SDN

3.3. Capa de administración

Las aplicaciones de red pueden ser vistas con el “cerebro de la red”. Ellas implementan el control lógico que se traducirá en los comandos a instalar en el plano de datos, definiendo el comportamiento y las políticas para el reenvío de los paquetes en los dispositivos. Un ejemplo de aplicación es el enrutamiento, cuyo objetivo es definir el camino a través del cual los paquetes de un flujo serán enviados desde el transmisor (punto A) hasta el receptor (punto B). Para lograr este objetivo, la aplicación de enrutamiento debe tener información detallada de la topología extraída por las capas inferiores que se definieron anteriormente. Con base en esta información, se decide el camino que debe tomar el flujo y se envían al controlador las instrucciones para instalar las reglas de reenvío en todos los dispositivos implicados en el camino del flujo desde A hasta B [30].

Múltiples lenguajes de programación se han adoptado para el desarrollo de aplicaciones. La variedad de lenguajes permite que gran cantidad de investigadores y empresas fabricantes desarrollen protocolos y mecanismos para optimizar la seguridad, la confiabilidad y la calidad de los servicios de red soportados.

En la Figura 7 se ilustra la arquitectura SDN. En la parte superior se muestra el API descrito anteriormente en el northbound y la capa superior donde se agrupan las Apps según su rol en la red.

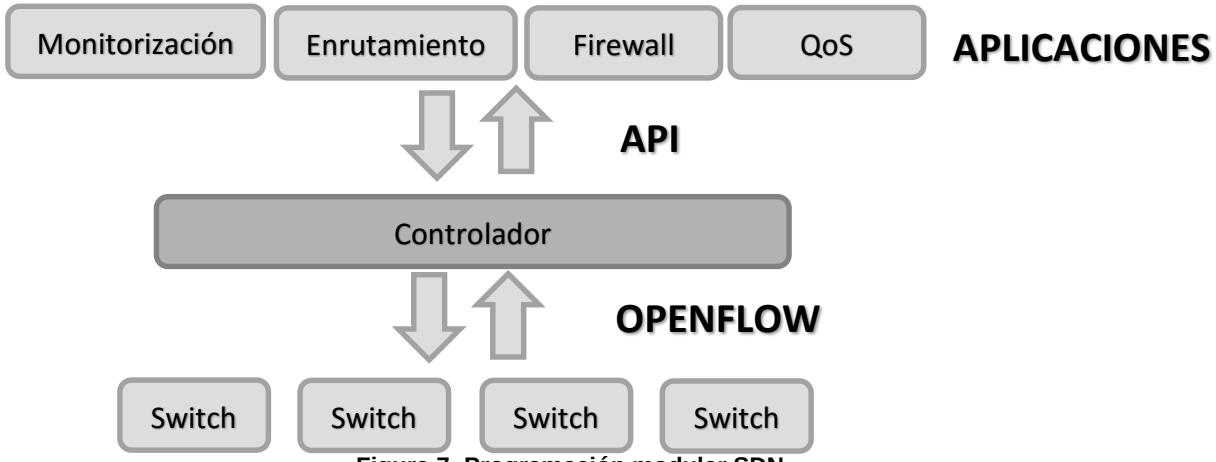


Figura 7: Programación modular SDN

3.3.1.1. Aplicaciones

SDN permite el desarrollo de aplicaciones de red propias. Las aplicaciones actuales pueden agruparse en categorías que se enfocan en administrar y optimizar diferentes características de las redes. La gran cantidad de aplicaciones y la implementación de tiendas comerciales para la distribución de éstas, muestran la evolución de la arquitectura SDN y la flexibilidad que se está dando al permitir múltiples fabricantes interactuar con diferentes dispositivos de red [31]. A continuación, se realiza una descripción las principales categorías de aplicaciones.

Ingeniería de tráfico

Esta ha sido una de las primeras características de red que se ha trabajado desde el comienzo del concepto SDN. Los objetivos principales de estas aplicaciones permiten disminuir el consumo de energía, maximizar la utilización de los enlaces de red, optimizar el balanceo de carga, y aplicar técnicas de optimización. La aplicación de estas técnicas es viable debido a la capacidad de tener una visión general de la topología y tomar decisiones basadas en información real entregada por las capas inferiores de la arquitectura SDN.

Movilidad y Wireless

Open Network Fundation es uno de los principales grupos de trabajo donde se reúnen los esfuerzos que combinan Empresa-Academia. La estrategia de trabajo define grupos que se enfocan en temas y desarrollos específicos, uno de los cuales está enfocado en Wireless and Mobile [32]. El desarrollo de aplicaciones en esta área busca optimizar la administración del espectro y la radio frecuencia (RF) y el balanceo de cargas en redes de alta densidad. SDN tienen un alto impacto en la administración de las redes inalámbricas en topología malla. Las redes malla requieren de optimización de enrutamiento, y la abstracción de la topología de red permite una visibilidad completa para optimizar los flujos mediante ingeniería de tráfico y calidad de servicio [33].

Monitorización y análisis

Las capas inferiores de la arquitectura SDN definen mediante el protocolo OpenFlow desde su versión 1.0 características básicas que permiten obtener información de contadores asociados a flujos. Esta información es consultada periódicamente por el controlador mediante mensajes de solicitud de estadísticas. Este tipo de aplicaciones permite implementar reglas proactivas que ajusten las políticas de asignación de ancho de banda basado en Traffic Shaping, la seguridad de la red y la optimización de caminos.
Actualmente se busca reducir la cantidad de solicitudes que las aplicaciones de la capa de administración hacen a la capa de control, debido al alto flujo de mensajes de control que genera en la red. El análisis de la información mediante estrategias estocásticas permite realizar la toma de decisiones con menos datos, reduciendo el trafico necesario para la administración de la red[34].

Seguridad y confiabilidad

La seguridad es una de las áreas más beneficiadas con la arquitectura SDN. Lograr una visibilidad completa de la red y un correlacionamiento de eventos de seguridad más amplio, permite implementar estrategias distribuidas para la mitigación de los diferentes ataques. Una red programable SDN logra un despliegue de mecanismos nuevos con desarrollo propio para controlar los ataques de día cero. Un ejemplo es OrchSec, definiendo una arquitectura para detectar ataques dentro de redes complejas SDN. Tan pronto detecta la anomalía, OrchSec notifica al controlador y a las aplicaciones de monitoreo, detectando ataques de envenenamiento de cache ARP, inundación ARP, ataques de denegación de servicios distribuidos y ataques DNS [35].

4. MODELADO FORMAL

La aplicación de métodos formales para describir la semántica de los sistemas de comunicación, permite identificar patrones como la coordinación o la cooperación de los protocolos de comunicación. Las redes de Petri [36] ofrecen un formalismo para describir de manera simplificada, sistemas que se pueden descomponer en subsistemas con estructuras y comportamientos repetitivos, permitiendo un modelado robusto y la posterior simulación y el análisis de sistemas complejos.

El uso de las redes de Petri (PN) o sus variaciones han sido comparadas con estrategias de modelado comunes como las cadenas de Markov. Las PN evitan dos de los inconvenientes básicos del análisis de las cadenas de Markov. En primer lugar, el modelo no crece en tamaño cuando el número de componentes del modelo aumenta, debido a que los estados en las PN son locales en lugar de ser globales, logrando que el modelo no crezca fuera de control a media que aumenta su complejidad. En segundo lugar, el análisis de las cadenas de Markov está limitado normalmente a modelar la probabilidad de cambios del sistema basados en una distribución exponencial, sin embargo los procesos relacionados con la confiabilidad, la disponibilidad y la seguridad, no necesariamente se ajustan a una distribución exponencial[37][38].

En esta sesión se describen los conceptos de diseño de Redes de Petri y su posterior análisis mediante la estrategia de espacio de estados.

4.1. Redes de Petri

Una red de Petri (PN) es un término genérico usado para representar sistemas dinámicos a eventos discretos (DEDS), cuya mayor ventaja reside en el soporte matemático subyacente y en la representación gráfica de estas. Mediante una PN, los DEDS se pueden describir mediante una topología distribuida, paralela o concurrente Las redes de Petri más primitivas fueron definidas en la década de los años 1960 por Carl Adam Petri [39], como una generalización de la teoría de autómatas que permite expresar un sistema a eventos concurrentes.

Una red de Petri se representa mediante un grafo dirigido y está formada por lugares, transiciones, arcos dirigidos y marcas o fichas; que ocupan posiciones dentro de los lugares como se observa en la Figura 8 . Las reglas definidas para el grafo son: Los arcos conectan desde un lugar a una transición, así como una transición a un lugar. No puede haber arcos entre lugares ni entre transiciones. Los lugares contienen un número finito o infinito contable de marcas. Las transiciones se disparan, es decir consumen marcas de un lugar de inicio y producen marcas en un lugar de llegada. Una transición está habilitada si tiene marcas en todas sus posiciones de entrada.

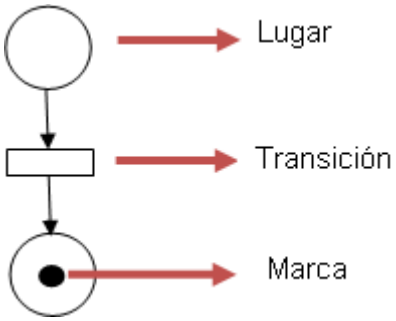


Figura 8: Redes de Petri

Matemáticamente, las PN básicas se definen como una tripleta $\langle P, T, F \rangle$, donde $P = \{p_1, p_2, \dots, p_n\}$ contiene los n lugares de la PN. $T = \{t_1, t_2, \dots, t_m\}$ Contiene las m transiciones de la PN. $F = P \times T \cup T \times P$ Representa los arcos que conectan las transiciones y los lugares. Debido a que estas conexiones representan un flujo y por tanto son direccionadas, "×" representa el producto cartesiano.

Usualmente, dicha tripleta se representa matemáticamente mediante una matriz de incidencia $C = p \times t$ donde:

$$c(i,j) = \begin{cases} W(t_i, p_j) & , iff\ (t_i, p_j) \in F \\ -W(p_i, t_j) & , iff (p_i, t_j) \in F \\ 0 & en\ otros\ casos \end{cases}$$

W Representa la magnitud del peso del arco que conecta los elementos indicados en el conjunto [40].

Esta representación matricial de las PN trae consigo la representación del marcaje de la PN mediante dos vectores: M que representa el marcaje actual de los lugares de la PN y M_0 que representa el marcaje inicial. Formalmente $M = [|m(p_1)|, |m(p_2)|, \dots, |m(p_n)|]^T$ donde $m(p_i) \geq 0$ es el marcaje del i-ésimo lugar.

En su forma más básica, las marcas que circulan en una red de Petri son todas idénticas. Se puede definir una variante de las redes de Petri, las redes de Petri Coloreadas, en las cuales las marcas pueden tener un color (una información que las distingue), un tiempo de activación y una jerarquía en la red.

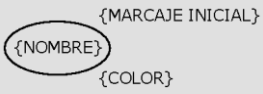
4.2. Redes de Petri coloreadas (CPN)

Las CPN se caracterizan principalmente porque las marcas que se ubican en los lugares pueden tener un tipo o color determinado, lo que permite distinguir las marcas unas de otras. A cada tipo de datos se le denomina conjunto de colores. Estos tipos o colores pueden combinarse para dar lugar a tipos compuestos o producto de colores. Las CPN poseen tres diferencias fundamentales con respecto a las PN clásicas: (a) los lugares de una CPN pueden contener elementos de diferentes tipow, (b) los arcos de una CPN tienen expresiones relacionadas con

éstos, y (c) las transiciones tienen una característica adicional que se denomina “guarda”. La existencia de tipos de datos en las CPN, permite que cada lugar pueda contener uno o más valores correspondientes a un tipo básico o a un producto de tipos. Cada dato o color presente en un lugar lleva un indicador de su multiplicidad. Es decir, se puede determinar cuántas copias del dato se encuentran en un lugar [36].

Las características particulares de las CPN confieren un mayor potencial para el modelado de sistemas, ya que las expresiones de arco y las guardas de las transiciones permiten restringir mejor las condiciones de disparo de las transiciones en una CPN con respecto a las PN clásicas. El empleo de colores y de expresiones de arco hace que el modelo que se obtenga con CPN sea más compacto que el modelo equivalente con PN clásicas [41]

Un modelo CPN es creado por el usuario mediante un modelo gráfico. Las partes que constituyen un modelo gráfico de CPN se describen en la Tabla 4.

NOMBRE	GRAFICO	FORMAL	DESCRIPCIÓN
LUGARES		$P = \{p_0, p_1, p_2, \dots, p_m\}$	Representan estados del modelo. Cada uno puede contener una o más marcas, cada marca posee información. Esta información es el color de la marca. El conjunto de marcajes y su valor en cada lugar, son los que representan el estado del sistema modelado. El nombre del lugar no posee significado formal pero tiene un gran impacto en la legibilidad del modelo.
MARCAJE		$\{cantidad1\}`(\{valor1\})$ $++\{cantidad2\}`(\{valor2$	Cada lugar posee un marcaje inicial y un marcaje actual. El marcaje inicial es el conjunto de valores y marcas con los que el sistema inicia, el marcaje actual corresponde al estado de simulación que se esté analizando. Los símbolos ` y **++** son operadores para construir el multiconjunto correspondiente a la marca: El operador ` toma un entero positivo como cantidad de marcas con el mismo valor y el operador ++ toma dos multiconjuntos y retorna su unión (suma).

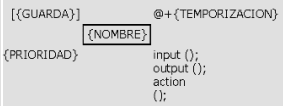
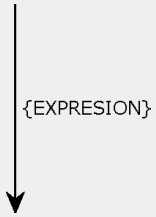
TRANSICIÓN		$T = \{t_0, t_1, t_2, \dots, t_n\}$	Representan los eventos que pueden generarse en el sistema. El nombre de las transiciones no posee significado más allá de la legibilidad del modelo gráfico. Cuando una transición se "*dispara*" remueve las marcas de los lugares de entrada y adiciona marcas a los lugares de salida. Los colores (información) de las marcas son removidos o adicionados según sea el caso, en función de la expresión presente cada arco que conecta los lugares con la transición.
ARCOS		$f: M(p_{in})$	Las expresiones de los arcos están compuestas por variables, constantes, operadores y funciones. Cuando las variables de la expresión pueden ligarse a un color de las marcas en los lugares de entrada a la transición, entonces la expresión puede ser evaluada y la transición disparada.

Tabla 4: Descripción de las partes de una CPN

En la Figura 9 se muestra ejemplo de CPN aplicado a protocolos de comunicación simples mediante el software de simulación CPN Tools, el modelo define la trasmisión de paquetes con probabilidad de pérdida y el control mediante mensajes ACK. El modelo es tomado de la librería de ejemplos de CPN Tools.

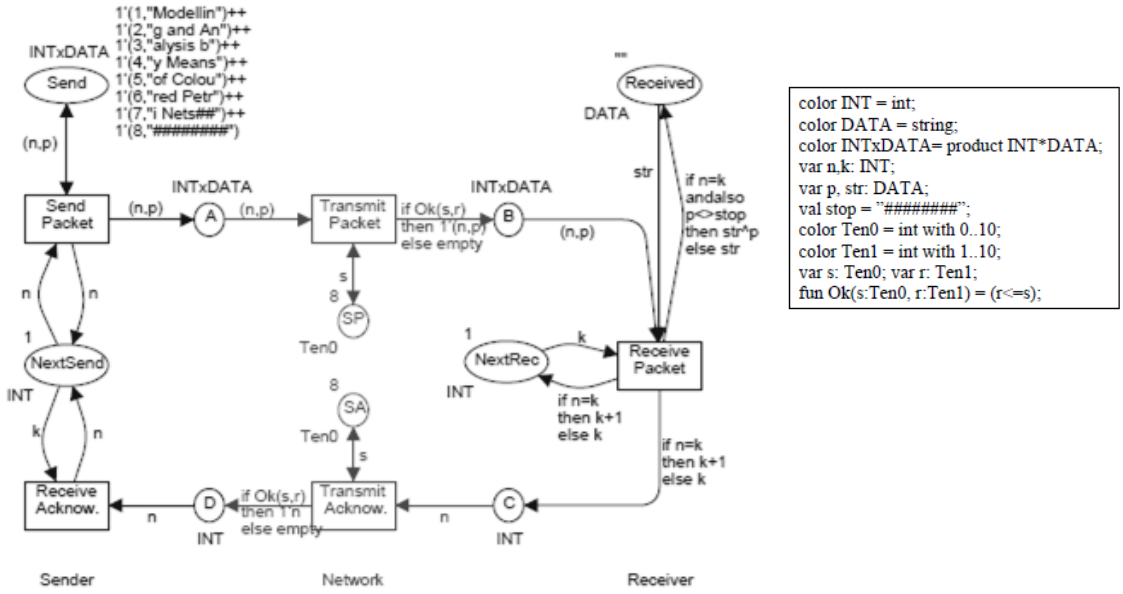


Figura 9: Modelo de protocolo simple

4.2.1.1. Propiedades dinámicas de las CPN

Antes de implementar un análisis funcional de un modelo formalizado por medio de redes de Petri Coloreadas, es necesario realizar un análisis de sus propiedades que permita descartar comportamientos no deseados en la simulación. El cumplimiento o no de cualquiera de estas propiedades permite verificar si el modelo puede representar o no funcionalmente cierto comportamiento deseado.

La verificación de las propiedades es soportada por el método de espacio de estados. La idea básica de este método es procesar todos los estados alcanzables y los cambios de estado del modelo CPN. El espacio de estados se ilustra como un grafo dirigido, donde los nodos representan los estados y los arcos la ocurrencia de eventos. Desde un espacio de estados construido, es posible responder un conjunto extenso de preguntas de verificación relacionadas al comportamiento de los sistemas [42]. La Figura 10 muestra un ejemplo del espacio de estados parcial para un paquete enviado por el modelo de la Figura 9.

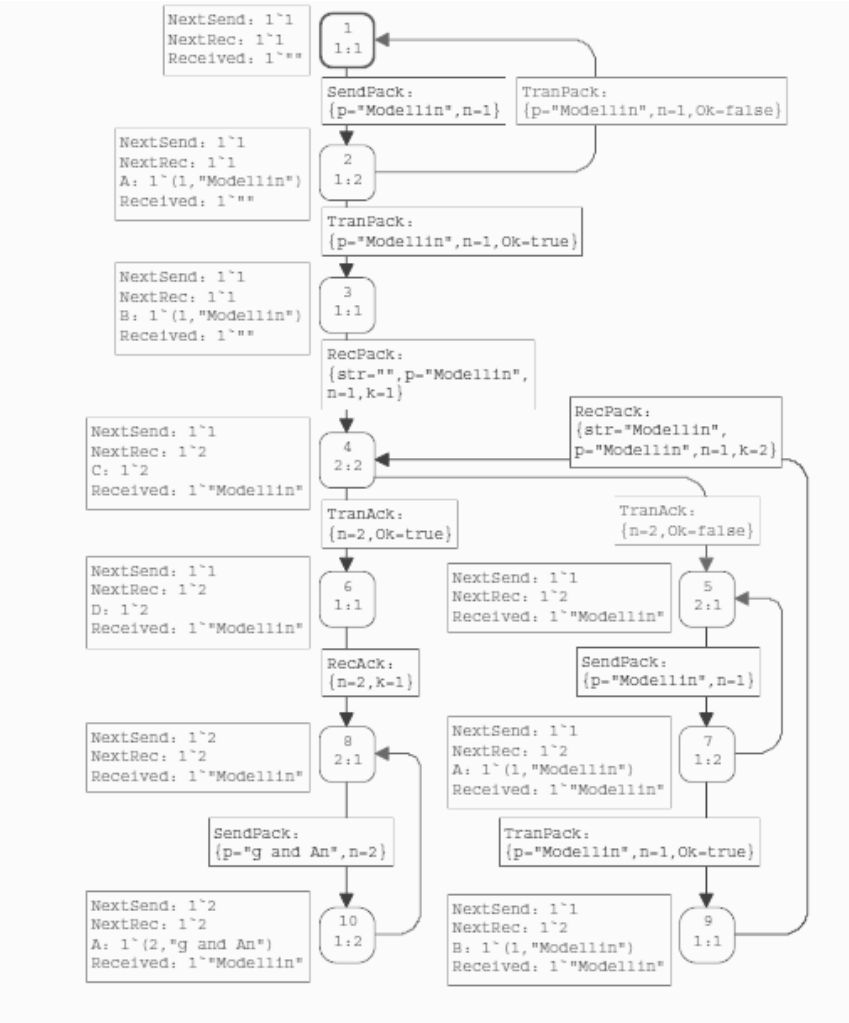


Figura 10: Espacio de estados

Las siguientes propiedades de una CPN son verificadas mediante el análisis del espacio de estados para validar formalmente un modelo. **(1) Limitación** Una red es limitada cuando todos sus lugares acumulan una cantidad limitada de marcas para cada marcaje alcanzable del árbol de alcanzabilidad. **(2) Vivacidad** Una red es Viva cuando para todo marcaje inicial, existe siempre un conjunto no vacío de transiciones sensibilizadas que permiten el marcaje continuar avanzando en los lugares de la red. **(3) Ciclicidad** describe la capacidad de una red para alcanzar un marcaje previo al marcaje actual mediante determinada secuencia de disparo.

Redes de Petri temporizadas y Estocásticas

Los modelos de rendimiento probabilísticos buscan representar el comportamiento de sistemas determinísticos complejos por medio de procesos estocásticos. De esta forma es posible evitar una detallada descripción determinística de las operaciones del sistema, sustituyéndola por asunciones probabilísticas, que capturen la esencia del sistema.

Las Redes de Petri Estocásticas o temporizadas se obtienen asociando la función de disparo de una transición en una variable aleatoria o un tiempo determinístico, esta expresa el retardo desde la habilitación hasta el disparo de la transición. La evaluación probabilística de los modelos requieres evolución del tiempo en el sistema. En una CPN temporizadas o estocástica, el tiempo es dado por un reloj global de simulación. Adicional a los colores, la marca contiene un valor de tiempo, llamado "*time stamp*". Cuando una transición se habilita, esta se dispara y cambia el time stamp sumando a la marca el retardo asociado a la transición. La marca queda congelada y no puede habilitar otra transición hasta que el tiempo del modelo sea menor que su time stamp. En otras palabras el time stamp define el momento en el cual la marca puede ser usada para disparar una transición en una red temporizada [43]

Los retardos asociados a las transiciones pueden ser determinísticos o tener una distribución exponencial que permita definir el modelo como un proceso markoviano. Un sistema de colas es un ejemplo adecuado para identificar los beneficios y posibilidades de análisis que las CPN estocásticas y las posibilidades que brindan para el trabajo con protocolos de comunicaciones. En la Figura 11 se muestra un sistema de colas M/M/1 con llegada de paquetes y tiempo de servicios definidos con una distribución exponencial. La transición "start" define la función porctime, que permite agregar a la marca un tiempo exponencial con media 90 para representar el tiempo de procesamiento del servidor,

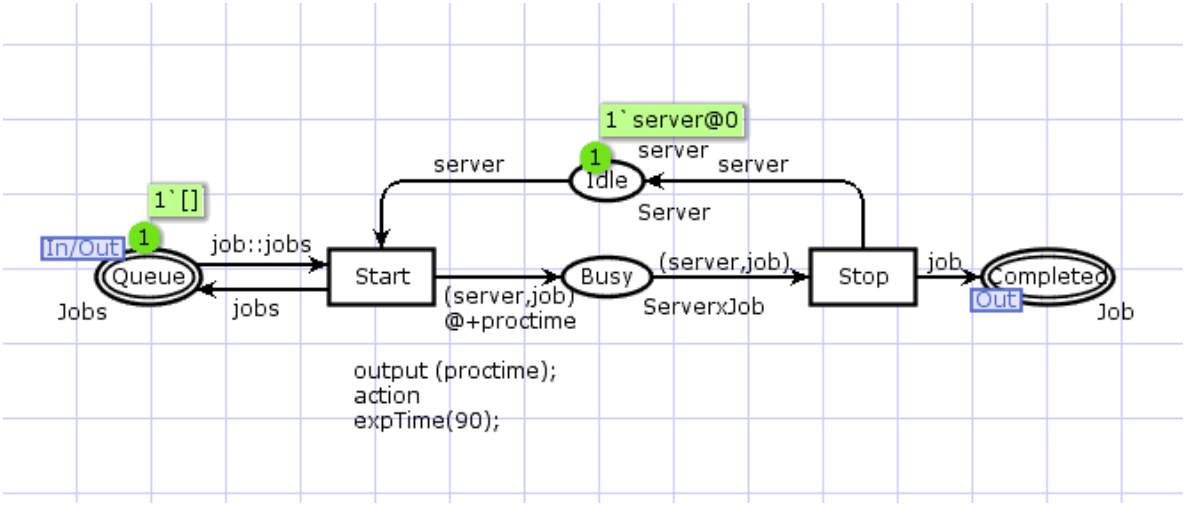


Figura 11: Queue System

5. MODELO CPN DE ARQUITECTURA BÁSICA SDN

Para definir los diferentes componentes que permitan diseñar el modelo de una topología básica SDN, se requiere entonces definir los diferentes participantes de la comunicación. Es necesario dos equipos finales para originar y finalizar el mensaje que para nuestro caso se modelan las tarjetas de red de dos computadores, un dispositivo intermedio compatible con OpenFlow, se decide por un switch dado que su análisis es ideal para un primer modelo. Adicionalmente requerimos de un cuarto dispositivo que es el controlador SDN, quien solo se comunica con el switch y en ningún momento con los dispositivos finales. Por último, se requiere modelar los diferentes canales de comunicación que representan un cable físico full-duplex, logrando una representación más cercana a la realidad y dando mayor flexibilidad al modelo, al permitir simular los tiempos de retardo asociados al canal de comunicación y los tiempos de lectura asociados a las tarjetas de red.

Para el modelado y la simulación de Redes de Petri Coloreadas se selecciona la herramienta CPN Tools, desarrollada por la Universidad de Aarhus, Dinamarca, y actualmente sostenida por el AIS group, Eindhoven University of Technology, The Netherlands[44]. Esta herramienta facilita la simulación y el análisis con una interfaz gráfica intuitiva y estable. Además, incluye herramientas formales para la validación de las propiedades básicas de un modelo de eventos discretos.

El modelo diseñado busca no solo modelar la entra y salida de información o flujos de cada uno de los dispositivos. Este se desarrolla buscando acercarse al comportamiento detallado de la secuencia de eventos que ocurren dentro de cada uno. El protocolo Openflow es modelado mediante el intercambio de los mensajes básicos que permiten solicitar información al controlador por parte del switch y modificar la tabla de flujos con la respuesta obtenida.

La Figura 12 muestra mediante un diagrama de flujo la secuencia de eventos que representan el comportamiento de la topología SDN y un acercamiento a los equipos reales con que se realiza la validación funcional. Este diagrama identifica los principales eventos sin mostrar el detalle de los procedimientos internos de cada uno, esto para permitir un adecuado entendimiento del comportamiento general del trabajo realizado.

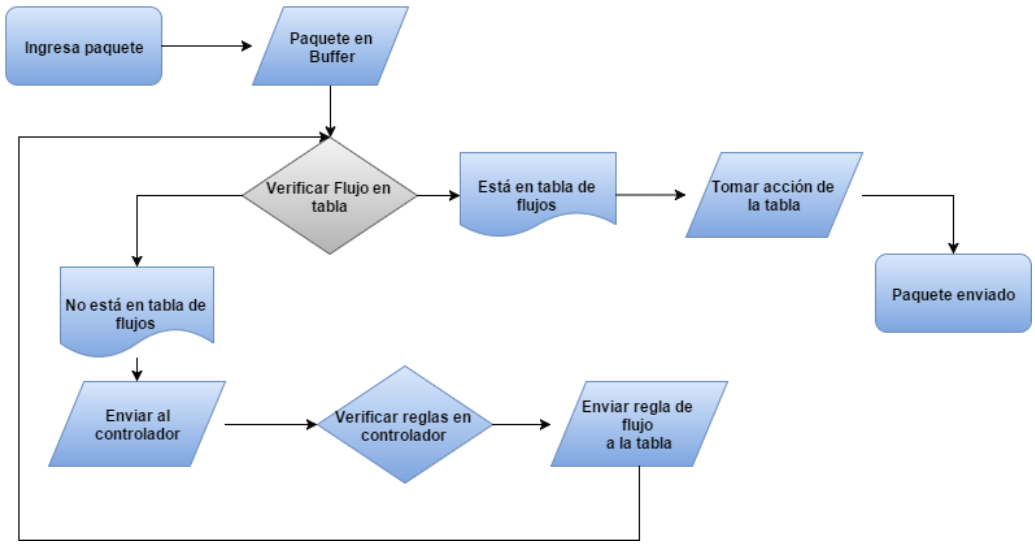


Figura 12 Diagrama de Flujo de eventos a modelar

Para llegar al modelo final se realizaron múltiples simulaciones que se pueden agrupar de acuerdo a los resultados obtenidos, en tres casos de prueba. El primer caso de prueba buscó verificar experimentalmente los tiempos de procesamiento del switch y del controlador, mostró que adicional a los dispositivos SDN como el controlador y el switch, se requería del modelado de los canales de comunicación entre los diferentes dispositivos, esto debido a que adicional a los tiempos medidos en los dispositivos hay tiempos adicionales generados por el cable y el procesamiento de los paquetes al ingresar y salir de los equipos. Posterior a las adecuaciones necesarias se procede al segundo caso de prueba que busca verificar el comportamiento adecuado de los cambios realizados, Se logra un modelo con un comportamiento muy similar al esperado, adicionalmente se identifican problemas de procesamiento aleatorio de los paquetes sin respetar la secuencialidad esperada. Es por esto que es necesario garantizar un procesamiento FIFO en el buffer del Switch mediante la utilización de un campo de secuencia en la información del paquete que ingresa al modelo, se realiza la adecuación del modelo y se verifica nuevamente con resultados satisfactorios. Como tercer caso de prueba se realizan las adecuaciones a los monitores que realizan los cálculos de los tiempos en el modelo hasta llegar a los valores presentados más adelante.

La Figura 13 muestra los dispositivos de la topología de red modelada: La topología descrita se ha modelado mediante CPN Jerárquicas que permite implementar submodelos para resumir el modelo de una manera ordenada y entendible.

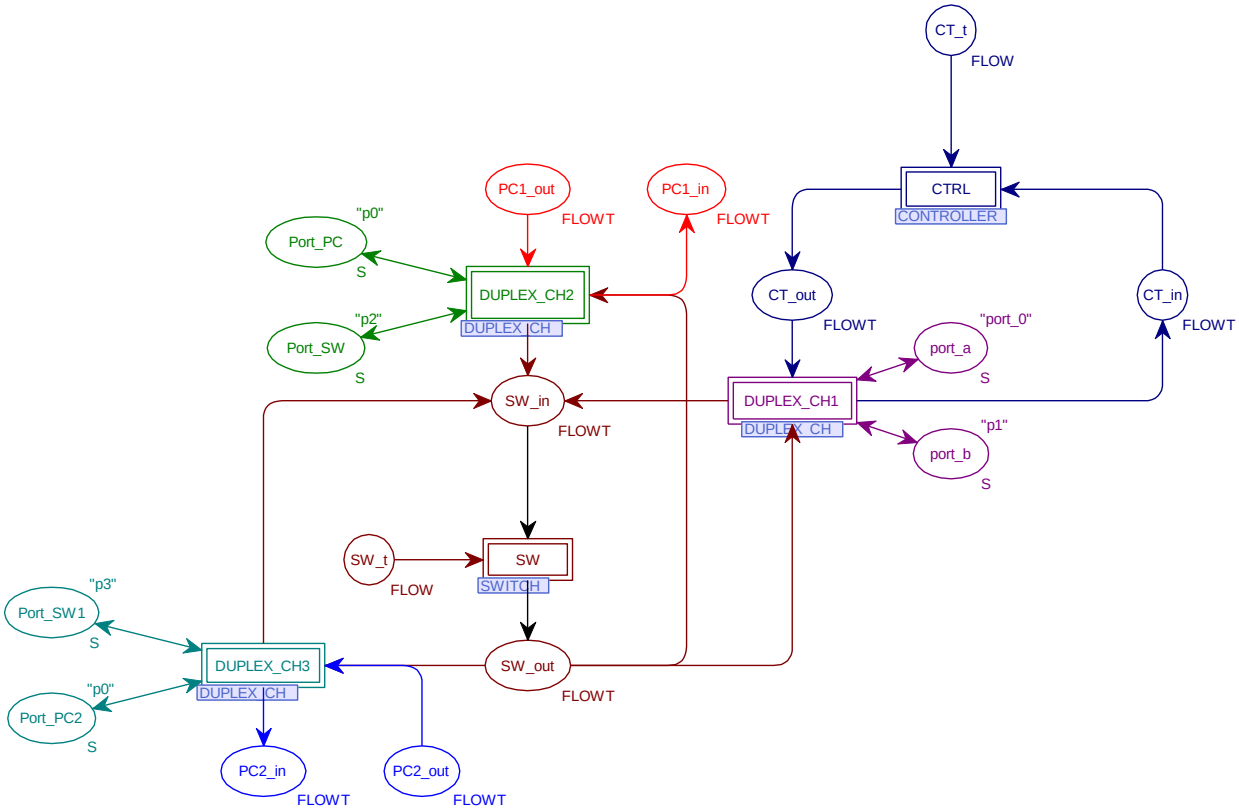


Figura 13: Topología básica SDN

5.1. Paquetes

Para el modelo CPN desarrollado en este trabajo, cada marca representa un paquete que puede ser de dos tipos diferentes: Para la comunicación entre switch y PC, y los paquetes para la comunicación con el controlador, simulando el comportamiento de los mensajes básico Openflow. Cada uno se explica a continuación:

- FLOWT: Son paquetes pertenecientes a un flujo que el computador genera para la comunicación entre los PC y el switch, también es usado para enviar una copia al controlador cuando el switch no encuentra coincidencia en la tabla de flujos. Está compuesto por:
    - Número de secuencia (num_seq: INT)
    - Puerto de salida (port_out: STR)
    - Puerto de entrada (port_in: STR)
    - Dirección MAC de origen (mac_src: STR)
    - Dirección MAC de destino (mac_dst: STR)
    - Dirección IP de origen (ip_src: STR)
    - Dirección IP de destino (ip_dst: STR)

        - Payload (payload: INT)

- MODFLOWT: Paquete que el controlador envía al switch como una instrucción para modificar su tabla de flujos con nuevas reglas. Está compuesto por:

    - Puerto de entrada al controlador (port_inc: STR)
    - Número de secuencia del paquete (num_seq: INT)
    - Puerto de salida del paquete (port_out: STR)
    - Puerto de entrada al switch (port_in: STR)
    - Dirección MAC de origen del paquete (mac_src: STR)
    - Dirección MAC de destino del paquete (mac_dst: STR)
    - Dirección IP de origen del paquete (ip_src: STR)
    - Dirección IP de destino del paquete (ip_dst: STR)

5.2. Switch

El modelo ilustrado en la Figura 14, tiene por lugar de entrada el *buffer_in* donde llegan los paquetes para ser comparados el lugar *checker*. Si es un paquete de tipo FLOWT se compara con la tabla del switch, de lo contrario si es un paquete MODFLOWT, se actualiza la tabla del switch con nuevas reglas enviadas por el controlador.
El disparo de la transición *to_check*, añade al paquete FLOWT un número de secuencia que corresponde al orden de llegada de estos paquetes al switch, de forma que el modelo tenga un procesamiento de paquetes de forma FIFO.
Según su orden de llegada, cada paquete es comparado con la tabla de switch y direccionado según la acción asociada al flujo en la tabla o enviado al controlador si no se encuentra una coincidencia con la tabla de flujos.
Los lugares y transiciones FIFO sirven para darle un comportamiento FIFO al modelo y procesar las tramas secuencialmente. En la transición *FIFO TRANSITION* ocurre el retardo correspondiente al procesamiento del switch, este retardo tiene unidades de microsegundos y se desarrolló con una función de proporcionalidad al tamaño del paquete que permite darle dinámica al retardo, el tamaño del paquete se incluye en la información de cada marca que representa el paquete y puede ser modificado para verificar el comportamiento del switch con diferentes tamaños de paquetes.

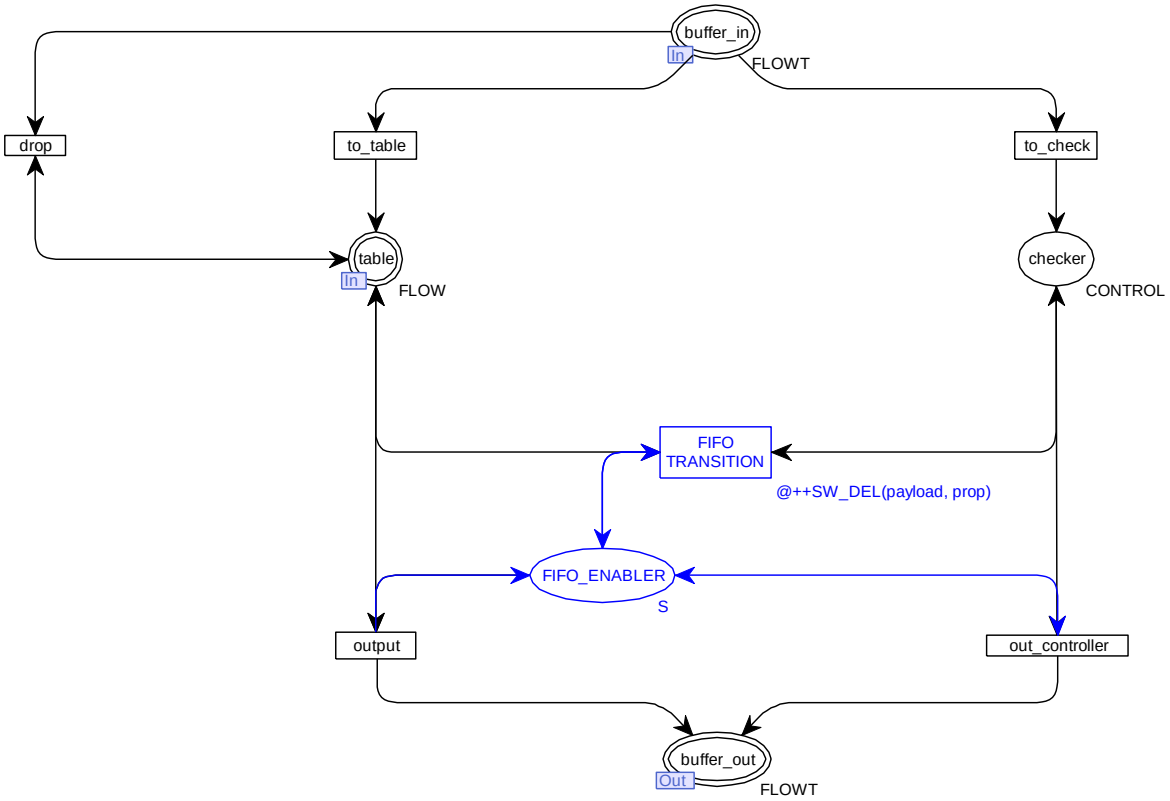


Figura 14: Modelo de switch SDN

5.3. Controlador

El controlador es modelado como se muestra en la Figura 15. Su funcionamiento indica que cuando se recibe una marca desde el switch, compara con la información contenida en el lugar *controller_table* donde se definen las reglas de los flujos. Cada regla tiene asociada una acción (Puerto de envío para este trabajo), si la marca que ingresa coincide con una de las reglas, el controlador envía un mensaje de modificación del flujo MODFLOW al switch para actualizar la tabla de flujos con la acción necesaria para procesar la marca (paquete). En este caso, el modelo del controlador, determina qué puerto de envío debe asignársele a un flujo. Un flujo es definido en este modelo como un conjunto de paquetes con que coinciden en MAC origen, MAC destino, IP de Origen e IP de destino.

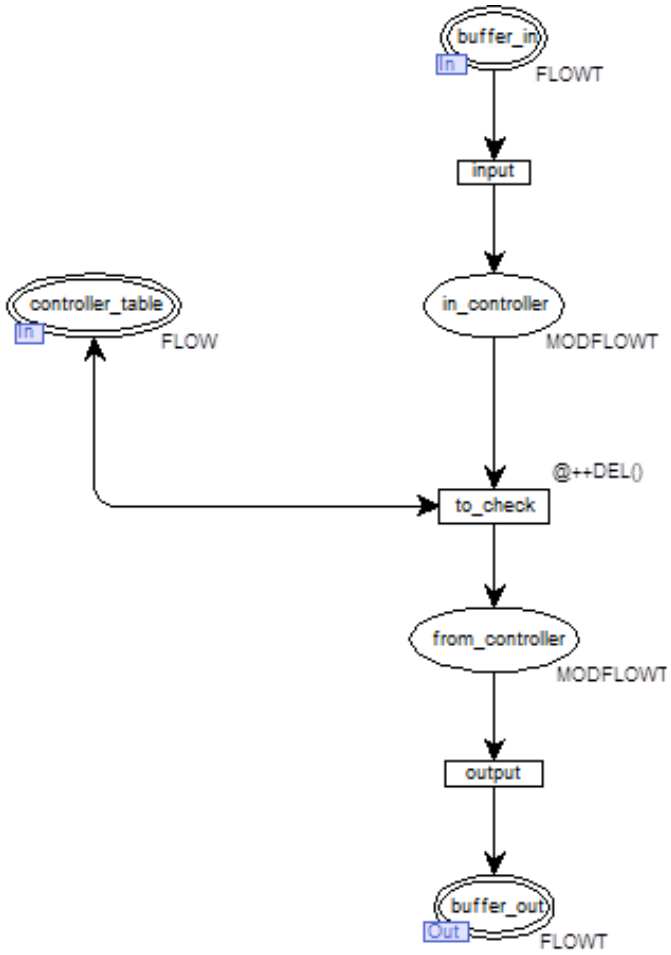


Figura 15: Modelo CPN del controlador

5.4. Canal

El modelo de la Figura 16, representa la conexión que existe entre los elementos que componen la topología de red modelada. Este modelo posee la información de los puertos de conexión de cada componente de tal manera que pueda enviarse cada paquete a su destino correctamente.

El modelo de canal está compuesto por la agregación de submodelos que representan el canal de envío o recepción de datos como se ilustra en la Figura 17. El modelo de canal ha sido diseñado para ser genérico y pueda ser replicado la cantidad de veces necesaria según la cantidad de enlaces de la topología a simular.

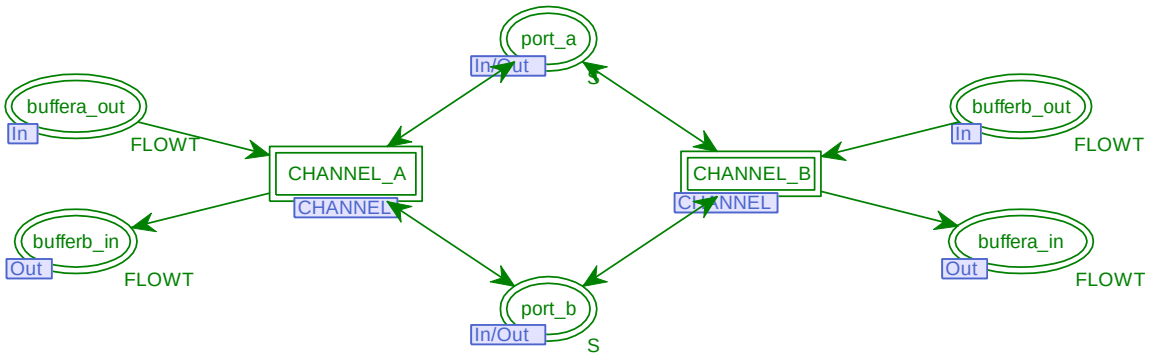


Figura 16: Modelo de Canal

:

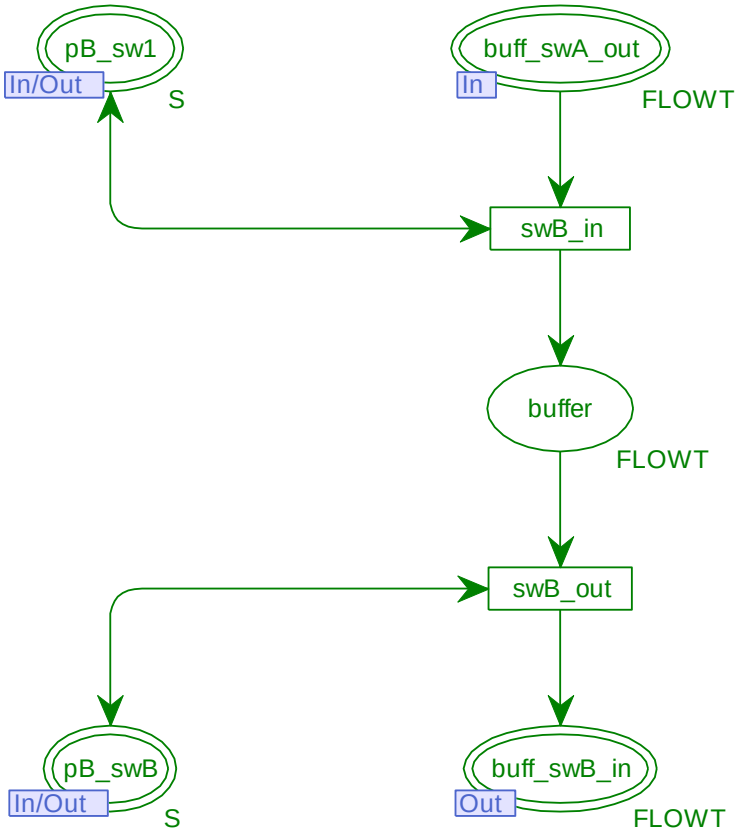


Figura 17: Submodelo de envío o recepción

5.5. Validación de las propiedades formales del modelo

CPN Tools cuenta con herramientas de generación de informes para el análisis de las propiedades del modelo propuesto y el árbol de alcanzabilidad explicado anteriormente.

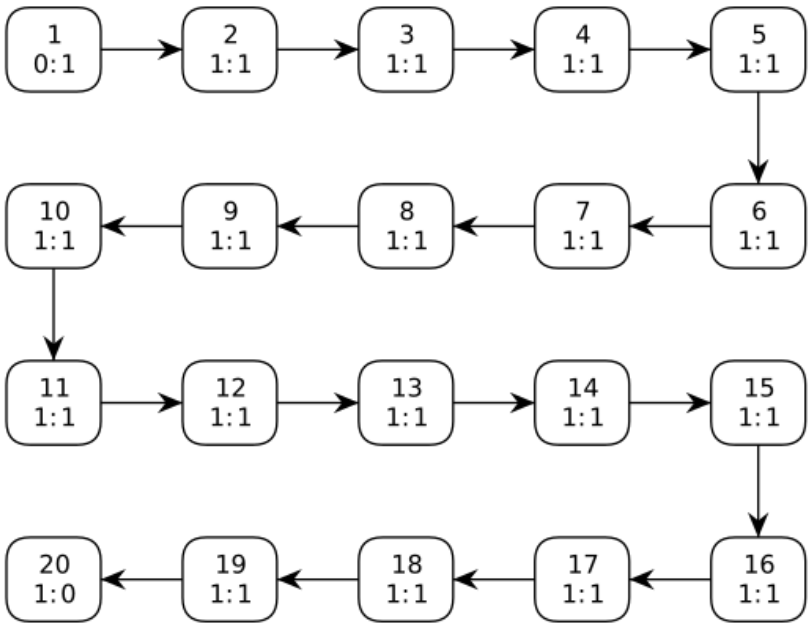


Figura 18: Árbol de alcanzabilidad generado en CPN Tools

El resultado del análisis de las propiedades basado en el árbol de estados presentado en la Figura 17 indica que el modelo cumple con siguientes propiedades:

1. **Limitación**: La cantidad de marcas acumuladas en cada lugar del modelo deben ser finitas y nunca mayor a la cantidad de flujos de entrada al modelo. En este caso, el modelo cumple: Para n flujos de entrada, hay n marcas acumuladas en el lugar de salida del PC2 y para m flujos diferentes (con n < m) hay m marcas acumuladas en el lugar de la tabla del controlador y del switch. Ver Tabla 5.

2. **Ciclicidad (o home marking**): Debe existir un único estado home marking (Nodo alcanzable desde cualquier otro nodo del árbol de alcanzabilidad). En este caso, el modelo cumple, debido a que existe un nodo home marking que es el último estado del a red, con todos los flujos de entrada al modelo finalizando en el lugar de salida del PC2, garantizando la no existencia de bucles en el modelo, evitando que los paquetes no sean procesados o se queden infinitamente en el modelo. Ver Tabla 6.

3. **Vivacidad:** Debe existir un nodo final en el árbol de alcanzabilidad, es decir, el modelo no debe ser vivo. En este caso, el modelo cumple: Para un flujo de entrada finito, existe una cantidad finita de estados del árbol de alcanzabilidad. Ver Tabla 7

Statistics	
State Space	
Nodes:	20
Arcs:	19
Secs:	0
Status:	Full

Boundedness Properties		
Lugar	**Upper**	**Lower**
CHANNEL'buffer 1	1	0
CHANNEL'buffer 2	1	0
CHANNEL'buffer 3	1	0
CHANNEL'buffer 4	1	0
CHANNEL'buffer 5	0	0
CHANNEL'buffer 6	0	0
CONTROLLER'AUX 1	1	0
CONTROLLER'from_controller 1	1	0
CONTROLLER'in_controller 1	1	0
CONTROLLER'intelligence 1	0	0
NETWORK2'CH_IN 1	1	0
NETWORK2'CT_in 1	1	0
NETWORK2'CT_out 1	1	0
NETWORK2'CT_t 1	1	1
NETWORK2'Input_flow 1	1	0
NETWORK2'PC1_buffer_in 1	1	0
NETWORK2'PC1_in 1	0	0
NETWORK2'PC1_out 1	1	0
NETWORK2'PC2_in 1	1	0
NETWORK2'PC2_out 1	0	0
NETWORK2'PC_timer 1	1	1
NETWORK2'Port_PC 1	1	1
NETWORK2'Port_PC2 1	1	1
NETWORK2'Port_SW 1	1	1
NETWORK2'Port_SW1 1	1	1
NETWORK2'SW_in 1	1	0
NETWORK2'SW_out 1	1	0
NETWORK2'SW_t 1	1	0
NETWORK2'port_a 1	1	1
NETWORK2'port_b 1	1	1
SWITCH'FIFO_ENABLER 1	1	1
SWITCH'FIFO_IN 1	1	1
SWITCH'FIFO_OUT 1	1	1
SWITCH'FIFO_PLACE 1	1	0
SWITCH'SYS_T 1	1	1
SWITCH'TIMER 1	1	1
SWITCH'checker 1	1	0
SWITCH'pause 1	1	0

Tabla 5: Limitación

Home Properties	
Home Markings	[20]

Tabla 6: Ciclicidad

Liveness Properties	
Dead Markings	[20]

Tabla 7: Vivacidad

6. VALIDACIÓN EXPERIMENTAL

Para realizar una validación de la flexibilidad y funcionamiento del modelo explicado en el capítulo 5, se propone realizar las mediciones de los tiempos de procesamiento de los principales dispositivos de una topología básica SDN experimental. Con la información de los tiempos experimentales se procederá a adecuar el modelo y verificar su comportamiento. Las variables de medición asociadas al experimento son:

Tiempo de procesamiento del Switch: Es el tiempo medido desde que el paquete entra hasta que sale del Switch, teniendo en cuenta que el flujo tiene coincidencia con una regla asociada al paquete en la tabla de flujos y no se consulte al controlador.

Tiempo de procesamiento del controlador: Este es medido con un sniffer dentro del controlador, mide el tiempo que toma desde que ingresa un paquete hasta el envío de la respuesta con la modificación de flujos al switch. Este tiempo no tiene en cuenta el retardo del enlace o el tiempo de lectura y escritura de la tarjeta de red

Retardos adicionales: Los retardos adicionales son asociados a las lecturas y escrituras de las tarjetas de red del enlace entre el switch y el controlador. Los tiempos de los enlaces entre los PC y el switch se consideran 0 para el experimento. Estos retardos son calculados al restar al tiempo del sistema máximo los tiempos calculados de procesamiento del switch y el controlador.

Tiempo del sistema: Es el total de los tiempos de procesamiento y los retardos adicionales. El tiempo del sistema es variable dependiendo si el paquete debe esperar la consulta al controlador, el cual es llamado tiempo del sistema máximo, o si el switch ya cuenta con la regla en la tabla de flujos, donde el tiempo del sistema es el mismo tiempo de procesamiento del switch.

Descripción del experimento

Las mediciones experimentales son realizadas en dos partes: en la primera para obtener 100 muestras de los tiempos de procesamiento del controlador se generan 100 flujos ICMP diferentes cada uno con un solo paquete; y la segunda para tomar 90 muestras del tiempo de procesamiento del Switch, se realizó el envío de 10 flujos ICMP diferentes, cada flujo con 10 paquetes iguales y diferenciados de los otros flujos por la MAC de origen. En la segunda medición el primer paquete de cada flujo es enviado al controlador para que este actualice la tabla de flujos del switch, los siguientes 9 paquetes de cada flujo son procesados sin necesidad de consultar el controlador y esto permite tomar los 90 tiempos de procesamiento del switch. El tiempo del sistema es variable dependiendo si el paquete debe esperar una respuesta del controlador o si el switch ya cuenta con la regla en la tabla de flujos.

6.1. Topología experimental

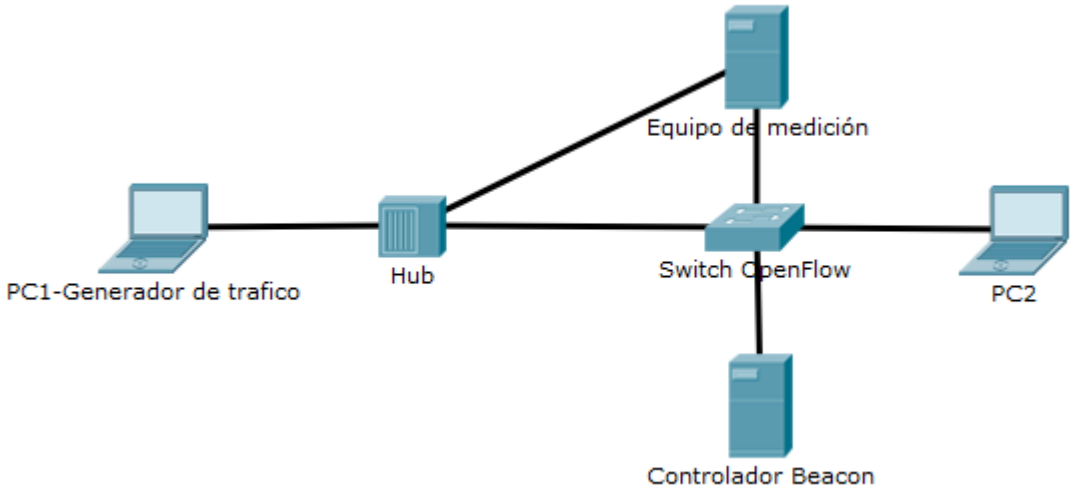


Figura 19: Topología experimental

Los componentes de la topología son:

Generador de tráfico: Este dispositivo hace las funciones de equipo cliente para generar los paquetes, el software generador de Trafico Ostinato es elegido por su flexibilidad en la generación de múltiples flujos con variación automática de la dirección MAC de origen y destino.

Hub: Este dispositivo replica simultáneamente los paquetes a todos los puertos a los que está conectado, logrando obtener una réplica del paquete que ingresa a switch y tomar el tiempo de ingreso.

Equipo de media: Se concluye después de pruebas previas donde los tiempos de procesamiento del Switch y controlador superan los milisegundos lo cual es un tiempo muy alto para el experimento, que se requiere un equipo servidor con dos tarjetas de red exactamente iguales y embebidas en la misma tarjeta para lograr disminuir los tiempos de diferencia entre las medidas de cada interfaz. Un microserver HP con Windows server 2012 y el software wireshark instalado fue usado para este fin.

Switch Openflow: Se cuenta con un dispositivo Mikrotik compatible con OpenFlow 1.0, a este dispositivo se le realiza la configuración básica de IP y OpenFlow para ser adoptado por el controlador.

Controlador Openflow: Un equipo portátil hace la función de controlador, el Beacon es el controlador escogido para realizar las mediciones de laboratorio.

6.2. Resultados y análisis experimentales

Para realizar un análisis estadístico de los resultados obtenidos del tiempo del switch, el tiempo del controlador y el tiempo del sistema; se procede a extraer los siguientes resultados:

Tiempo promedio del sistema (Anexo 1) 310,5 microsegundos
Tiempo promedio de procesamiento del switch (Anexo 2): 28,33 microsegundos.
Tiempo promedio de procesamiento del controlador (Anexo 3):440,38 microsegundos.
Tiempo promedio máximo del sistema 2850.1 microsegundos.

Los tiempos de retardo adicionales asociados a las lecturas de tarjetas y retardos en los canales como se describió anteriormente, son calculados al restarle al tiempo del sistema máximo, el tiempo de procesamiento del switch y el controlador. Este tiempo asociado al procesamiento de las solicitudes openflow entre switch y controlador.

Retardos adicionales = 2850,1 – 28,33 – 440,38 = 2381,39 microsegundos.

Para validar la confiabilidad de los datos se calcula la desviación estándar de las de la variable principal del experimento:

Desviación estándar para tiempo del sistema: 0,85702626 milisegundos

6.3. Validación funcional del modelo

El modelo diseñado durante este trabajo, es flexible para ser adecuado a cualquier switch o controlador. Los tiempos de procesamiento y retardo se pueden cambiar de una manera flexible con las variables declaradas y asociadas a una transición específica dentro del switch, el controlador y el enlace al controlador. En la Figura 20 se identifican las variables que definen los tiempos necesarios para validar el modelo.

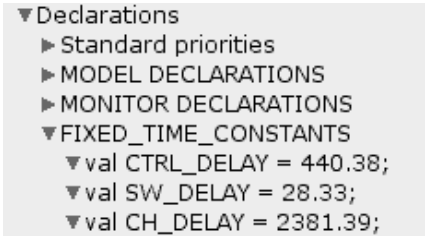


Figura 20: Variables de tiempo CPNtools

Al igual que la segunda muestra experimental descrita anteriormente para el montaje físico, Se procede en CPN Tools a realizar el envío de 10 flujos de marcas

(paquete), cada flujo con 10 marcas iguales y diferenciados de los otros flujos por la MAC de origen para un total de 100 paquetes o marcas. La lectura automática de las marcas se realiza desde un archivo de texto separado por comas como se muestra en el anexo 4. Estas marcas son ingresadas con una distribución de Poisson que permite el ingreso con tiempos aleatorios desde el PC1. Al modelo se adiciona un conjunto de lugares que permite controlar los tiempos para que los paquetes sean consecutivos y se procesen línea por línea del archivo de texto, como se muestra en la Figura 21.

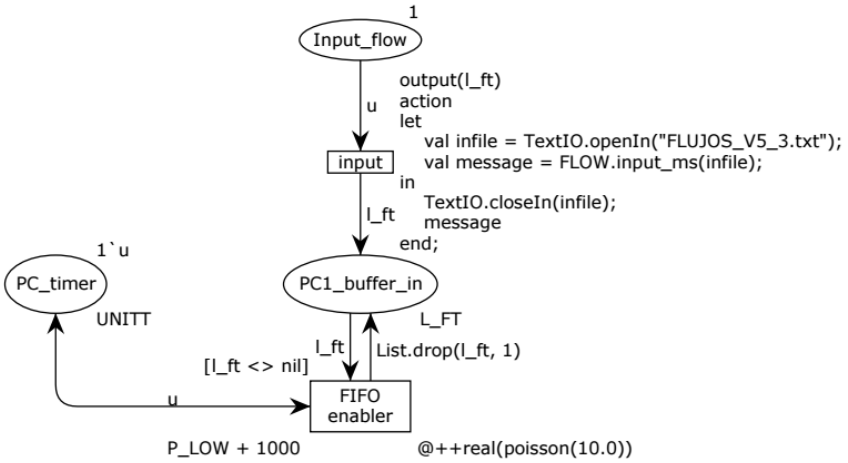


Figura 21: Lectura e ingreso de paquetes aleatorios

CPN tools permite la utilización de funciones con tareas de monitores, estos monitores permiten tomar los tiempos asociados a múltiples transiciones y realizar cálculos estadísticos definidos como una función. En la Anexo 4 y Anexo 5 muestra el detalle de la configuración realizada para los monitores.

Finalmente, al realizar la simulación completa con la lectura de los datos desde el archivo de texto y el procesamiento de todos los paquetes, el resultado de la simulación es entregado por la herramienta CPN tools de manera tabulada y en formato html como se muestra en la Figura 22.

CPN Tools Simulation Performance Report
Net: /cygdrive/C/Users/Usuario/Dropbox/Personales/Documentos/OPF/OPF-MODHF-007-modelo_switch_controlador_chanr

---

Note that these statistics have been calculated for data that is not necessarily independent or identically distributed.

Untimed statistics					
Name	**Avrg**	**Variance**	**StD**	**Min**	**Max**
CONTROLLER_DELAY	440.380000	0.000000	0.000005	440.380000	440.380000
SWITCH_DELAY	28.330000	0.000000	0.000002	28.330000	28.330000
SYSTEM_DELAY	310.507000	723853.266627	850.795667	28.330000	2850.100000

Simulation steps executed: 901
Model time: 31055.7

Generated: Wed Jan 20 18:02:38 2016

Figura 22: Resultados simulación CPN Tools

Al realizar el análisis de los resultados de la simulación en la Figura 22 se identifica que el tiempo del sistema promedio es el esperado comparado con el tiempo del sistema promedio experimental de 310.5 segundos. La varianza cuenta con un valor alto debido a la aleatoriedad del tiempo de procesamiento de los primeros paquetes de cada flujo nuevo, que deben esperar la respuesta del controlador al switch. También se valida la desviación estándar que es entregada por CPNtools con la calculada para el montaje experimental. El modelo logra entonces representar formalmente el comportamiento esperado de una topología básica Openflow generando gran expectativa para trabajos futuros, como herramienta para implementar simulaciones con miles de paquetes en tiempos cortos, resultado esperado al momento de planear el presente proyecto.

7. CONCLUSIONES Y TRABAJOS FUTUROS

Los resultados para el alcance del proyecto son los planeados inicialmente, permitiendo concluir que las CPN fueron una selección adecuada para el modelado de una topología básica SDN. El modelo permitió desde su interfaz gráfica modificar y administrar los tiempos determinísticos de una manera flexible y escalable. A diferencia de NS3, CPN Tools permite de una manera gráfica hacerle seguimiento a cada uno de los eventos del dispositivo o el protocolo modelado, permitiendo que sea de fácil trasmisión el conocimiento obtenido durante este trabajo.

El tiempo del controlador es viable para futuros trabajos mejorando el análisis con tiempos estocásticos, este modelo propuesto permite fácilmente cambiar los tiempos del controlador o del switch para que sean compatibles con distribuciones estocásticas como la exponencial. Es adecuado entonces hacer una validación más detallada del comportamiento de la arquitectura SDN mediante CPN estocásticas.

La amplia utilización del lenguaje Java en los controladores SDN con códigos de fácil utilización para toda la comunidad, y adicionalmente la capacidad de CPN para integrarse así mismo con java permite visualizar proyectos futuros con esta comunicación. Una interesante implementación podría logra que un modelo en CPN tools no solo permita analizar el comportamiento de una topología, sino también la posibilidad de crear un controlador con la capacidad de comunicarse en tiempo real con dispositivos SDN/Openflow

Finalmente, aunque en este trabajo se modeló el intercambio básico de los mensajes Openflow, es la base para un futuro desarrollo más detallado de los mensajes adicionales Openflow que permite realizar el control de las comunicaciones entre el controlador y el switch. Buscando una representación y un análisis más detallado del comportamiento de la comunicación entre dispositivos de red Openflow y el controlador SDN.

8. REFERENCIAS

[1] H. A. Rauf, D. C. S. E. It, V. L. B. J. College, and A. E. Jeyakumar, “COLORED PETRI NET MODELING AND THROUGHPUT ANALYSIS FOR WIRELESS INFRASTRUCTURE NETWORKS,” vol. 3, no. 3, pp. 155–160, 2005.

[2] B. Barzegar and H. Motameni, “Modeling and Simulation Firewall Using Colored Petri Net,” vol. 10, no. 1, pp. 39–44, 2011.

[3] V. Gehlot and C. Nigro, “Colored Petri Net model of the Session Initiation Protocol (SIP),” *IECON 2010 - 36th Annu. Conf. IEEE Ind. Electron. Soc.*, pp. 2150–2155, Nov. 2010.

[4] M. Li-li, “analysis and verification of colored petri net in pppoe protocol,” *3rd Int. Conf. Adv. Comput. Theory Eng.*, pp. 78–82, 2010.

[5] J. Wang, J. Yang, G. Xie, and M. Zhou, “OSPFv3 protocol simulation with colored Petri nets,” *… Proceedings, 2003. ICCT …*, no. 60273070, 2003.

[6] R. Sa, Y. Zhao, and C. Hai, “Research on the LDP Protocol Verification Based on Coloured Petri Nets,” *2010 Int. Conf. Internet Technol. Appl.*, pp. 1–4, Aug. 2010.

[7] Z. Aliannezhadi and M. A. Azgomi, “Modeling and Analysis of a Web Service Firewall Using Coloured Petri Nets,” pp. 548–553, 2008.

[8] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-gia, “Modeling and Performance Evaluation of an OpenFlow Architecture,” *2011 23rd Int. Teletraffic Congr.*, pp. 1–7, 2011.

[9] M. Kang, E.-Y. Kang, D.-Y. Hwang, B.-J. Kim, K.-H. Nam, M.-K. Shin, and J.-Y. Choi, “Formal Modeling and Verification of SDN-OpenFlow,” *2013 IEEE Sixth Int. Conf. Softw. Testing, Verif. Valid.*, pp. 481–482, Mar. 2013.

[10] D. Sethi, S. Narayana, and S. Malik, “Abstractions for model checking SDN controllers,” *Form. Methods Comput. …*, pp. 145–148, 2013.

[11] M. Antonio, T. Rojas, E. T. Ueda, T. Cristina, and M. De Brito, “Modelling and Verification of Security Rules in an OpenFlow Environment with Coloured Petri Nets,” *Inf. Syst. Technol.*, vol. 9th Iberia, pp. 1 – 7, 2014.

[12] L. Dong, H. Li, N. He, and Y. Xing, “Testing OpenFlow interaction property based on hierarchy CPN,” *Proc. - Int. Conf. Netw. Protoc. ICNP*, pp. 4–5, 2013.

[13] S. Koizumi and M. Fujiwaka, “SDN + cloud integrated control with statistical analysis and discrete event simulation,” pp. 289–294, 2015.

[14] N. McKeown, “Software Defined Network.” [Online]. Available: http://yuba.stanford.edu/.

[15] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, D. Walker, M. R. Harrison, and U. S. M. Academy, “Languages for Software-Defined Networks,” *IEEE Comun. Mag.*,

no. February, pp. 128–134, 2013.

[16] O. Fundation, “Software-defined networking: The new norm for networks,” *ONF White Pap.*, 2012.

[17] Open Networking Foundation, “SDN Product Directory,” 2013. [Online]. Available: http://sdndirectory.opennetworking.org/products.

[18] K. Nguyen, Q. T. Minh, and S. Yamada, “Increasing resilience of OpenFlow WANs using multipath communication,” *2013 Int. Conf. IT Converg. Secur. ICITCS 2013*, pp. 3–4, 2013.

[19] HP, “Production-ready SDN with OpenFlow 1.3,” 2013.

[20] Open Networking Foundation, *OpenFlow Switch Specification v1.4.0*, vol. 0. 2013, pp. 1–36.

[21] V. W. Protocol, “OpenFlow Switch Specification v1.5.0,” vol. 0, pp. 1–205, 2013.

[22] D. Drutskoy, E. Keller, and J. Rexford, “Scalable network virtualization in software-defined networks,” *Internet Comput. IEEE*, 2013.

[23] E. Salvadori and R. Corin, “Generalizing virtual network topologies in OpenFlow-based networks,” *… 2011), 2011 IEEE*, 2011.

[24] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, “FlowVisor: A Network Virtualization Layer,” 2009.

[25] V. Yazici, M. O. Sunay, and A. O. Ercan, “Controlling a Software-Defined Network via Distributed Controllers,” *arXiv Prepr. arXiv1401.7651*, vol. 90, no. 11580, p. 6, 2014.

[26] Juniper, “Contrail,” *http://www.juniper.net/us/en/products-services/sdn/contrail/*. .

[27] J. Dix, “Clarifying the role of software-defined networking northbound APIs,” 2013. .

[28] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, “Modular SDN Programming with Pyretic,” *USENIX ;login:*, vol. 38, pp. 40–47, 2013.

[29] R. Soule, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster, “Managing the Network with Merlin,” *Twelfth ACM Work. Hot Top. Networks*, 2013.

[30] D. Kreutz and F. Ramos, “Software-Defined Networking: A Comprehensive Survey,” *arXiv Prepr. arXiv …*, pp. 1–61, 2014.

[31] HP, “SDN App Store,” 2015. [Online]. Available: http://www8.hp.com/co/es/networking/sdn/devcenter-index.html.

[32] Open Networking Foundation, “Wireless & Mobile,” 2013.

[33] P. Dely, A. Kassler, and N. Bayer, “OpenFlow for Wireless Mesh Networks,” *2011 Proc. 20th Int. Conf. Comput. Commun. Networks*, pp. 1–6, 2011.

[34] S. A. Mehdi, J. Khalid, and S. A. Khayam, “Revisiting Traffic Anomaly Detection using Software Defined Networking,” *Entropy*, vol. 6961, pp. 1–20, 2011.

[35] Fraunhofer Institute for Secure Information Technology SIT, “OrchSec,” 2015.

[36] K. Jensen and L. M. Kristensen, “Coloured Petri Nets: Modelling and Validation of Concurrent Systems,” *Springer*, vol. 9, p. 384, 2009.

[37] N. B. Fuqua, “Markov Analysis,” *J. RAC*, vol. Third Quar, 2003.

[38] Captain Richard V. Melnyk, “Petri Nets: An Alternative to Markov Chains.” [Online]. Available: http://theriac.org/DeskReference/viewDocument.php?id=80&Scope=reg.

[39] M. Silva, “Half a century after Carl Adam Petri’s Ph.D. thesis: A perspective on the field,” *Annual Reviews in Control*, vol. 37. pp. 191–219, 2013.

[40] X. L. X. Liao, X. Z. X. Zhang, and J. J. J. Jiang, “Petri net-based modeling of switching arrangements and simulation,” *IEEE Int. Conf. Mechatronics Autom. 2005*, vol. 3, 2005.

[41] O. López, M. A. Laguna, and F. J. García, “Representación de Requisitos mediante Redes de Petri Coloreadas,” 2002.

[42] B. Pinna, G. Babykina, N. Brînzei, and J. Pétin, “Deterministic and stochastic dependability analysis of industrial systems using Coloured Petri Nets approach,” pp. 2969–2977, 2014.

[43] K. Jensen and L. M. Kristensen, “Formal Definition of Hierarchical Coloured Petri Nets,” in *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*, Springer Berlin Heidelberg, 2009, pp. 127–149 LA – English.

[44] T. N. AIS group, Eindhoven University of Technology, “CPN Tools,” 2013. [Online]. Available: http://cpntools.org/.

Anexo 1: Tabla de tiempos del sistema

Tiempo del Sistema	
Paquete	**Tiempo de procesamiento (seg.)**
1	0,003797
2	0,002757
3	0,002908
4	0,002722
5	0,002701
6	0,002676
7	0,002763
8	0,002778
9	0,00267
10	0,002729
11	0,000028
12	0,00004
13	0,000042
14	0,000025
15	0,000027
16	0,000041
17	0,000013
18	0,000041
19	0,000042
20	0,000018
21	0,000041
22	0,000027
23	0,00004
24	0,000041
25	0,000042
26	0,000028
27	0,000041
28	0
29	0,000026
30	0,000037
31	0,000041
32	0,000027
33	0,000015
34	0,000042
35	0,00004
36	0,000041
37	0,000028
38	0,000039
39	0,000043
40	0,000036
41	0,000042
42	0,000042
43	0,000041
44	0,000028
45	0,000028
46	0,000022
47	0,000014
48	0,00004
49	0,00004
50	0,000026
51	0,000028
52	0,000045
53	0,000038
54	0,000035
55	0,000042
56	0,000005
57	0,000037
58	0,000027
59	0,000027
60	0,000028
61	0,000026
62	0,000036
63	0,000002
64	0
65	0,000037
66	0,000007

67	0,000041
68	0,000013
69	0,000009
70	0,000037
71	0,000035
72	0,000011
73	0,000032
74	0,000036
75	0,00001
76	0,000047
77	0,000038
78	0
79	0,000036
80	0,000009
81	0,000037
82	0,000037
83	0,000002

84	0,000002
85	0,000013
86	0,000012
87	0,000037
88	0,000001
89	0,000017
90	0,000001
91	0,000037
92	0,000035
93	0,000036
94	0,000039
95	0,00004
96	0,000028
97	0,000019
98	0,000028
99	0,00002
100	0,00002

Anexo 2: Tabla de tiempos de procesamiento del switch

Tiempo de procesamiento de switch	
Paquete	**Tiempo de procesamiento (seg)**
1	0.000028000
2	0.000040000
3	0.000042000
4	0.000025000
5	0.000027000
6	0.000041000
7	0.000013000
8	0.000041000
9	0.000042000
10	0.000018000
11	0.000041000
12	0.000027000
13	0.000040000
14	0.000041000
15	0.000042000
16	0.000028000
17	0.000041000
18	0.000000000
19	0.000026000
20	0.000037000
21	0.000041000
22	0.000027000
23	0.000015000
24	0.000042000
25	0.000040000
26	0.000041000
27	0.000028000
28	0.000039000
29	0.000043000
30	0.000036000
31	0.000042000
32	0.000042000
33	0.000041000
34	0.000028000
35	0.000028000
36	0.000022000
37	0.000014000
38	0.000040000
39	0.000040000
40	0.000026000
41	0.000028000
42	0.000045000
43	0.000038000
44	0.000035000
45	0.000042000
46	0.000005000
47	0.000037000
48	0.000027000
49	0.000027000
50	0.000028000
51	0.000026000
52	0.000036000
53	0.000002000
54	0.000000000
55	0.000037000
56	0.000007000
57	0.000041000
58	0.000013000
59	0.000009000
60	0.000037000
61	0.000035000
62	0.000011000
63	0.000032000
64	0.000036000
65	0.000010000
66	0.000047000
67	0.000038000
68	0.000000000
69	0.000036000
70	0.000009000
71	0.000037000
72	0.000037000
73	0.000002000
74	0.000002000
75	0.000013000

76	0.000012000	84	0.000039000
77	0.000037000	85	0.000040000
78	0.000001000	86	0.000028000
79	0.000017000	87	0.000019000
80	0.000001000	88	0.000028000
81	0.000037000	89	0.000020000
82	0.000035000	90	0.000020000
83	0.000036000		

Anexo 3: Tabla de tiempos de procesamiento del controlador

Paquetes Controlador	
Paquetes (unid)	**Tiempo de procesamiento (seg)**
1	0.000425000
2	0.000432000
3	0.000451000
4	0.000419000
5	0.000417000
6	0.000442000
7	0.000496000
8	0.000407000
9	0.000466000
10	0.000412000
11	0.000451000
12	0.000412000
13	0.000440000
14	0.000403000
15	0.000408000
16	0.000440000
17	0.000362000
18	0.000439000
19	0.000422000
20	0.000448000
21	0.000411000
22	0.000460000
23	0.000419000
24	0.000422000
25	0.000431000
26	0.000416000
27	0.000457000
28	0.000410000
29	0.000470000
30	0.000397000
31	0.000449000
32	0.000401000
33	0.000454000
34	0.000407000
35	0.000445000
36	0.000417000
37	0.000409000
38	0.000423000
39	0.000408000
40	0.000467000
41	0.000438000
42	0.000474000
43	0.000459000
44	0.000515000
45	0.000417000
46	0.000429000
47	0.000835000
48	0.000434000
49	0.000429000
50	0.000413000
51	0.000399000
52	0.000397000
53	0.000406000
54	0.000458000
55	0.000420000
56	0.000450000
57	0.000436000
58	0.000526000
59	0.000432000
60	0.000515000
61	0.000417000
62	0.000501000
63	0.000415000
64	0.000473000
65	0.000408000
66	0.000408000
67	0.000411000
68	0.000439000
69	0.000509000
70	0.000457000
71	0.000441000

72	0.000443000	82	0.000398000
73	0.000450000	83	0.000400000
74	0.000445000	84	0.000396000
75	0.000441000	85	0.000496000
76	0.000435000	86	0.000478000
77	0.000402000	87	0.000454000
78	0.000455000	88	0.000404000
79	0.000440000	89	0.000427000
80	0.000434000	90	0.000408000
81	0.000502000		

Anexo 4: Lista de paquetes de entrada

1`(1,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc2","ip_pc1","ip_pc2",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc3","ip_pc1","ip_pc3",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc4","ip_pc1","ip_pc4",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc5","ip_pc1","ip_pc5",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)

```
+++1`(5,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc6","ip_pc1","ip_pc6",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc7","ip_pc1","ip_pc7",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc8","ip_pc1","ip_pc8",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc9","ip_pc1","ip_pc9",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(2,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc10","ip_pc1","ip_pc10",100)
+++1`(1,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
```

```
+++1`(2,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(3,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(4,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(5,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(6,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(7,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(8,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(9,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
+++1`(10,"p0","p2","mac_pc1","mac_pc11","ip_pc1","ip_pc11",100)
```

Anexo 5: Monitores para medir los tiempos de procesamiento el switch y el controlador

Monitors
- CONTROLLER_DELAY
  - Type: Data collection
  - Nodes ordered by pages
  - Predicate
  - Observer

```
fun obs (bindelem) =
let
  fun obsBindElem (CONTROLLER'output (1,
                        {aux0,ip_dst_t,ip_src_t,mac_dst_t,
                        mac_src_t,num_seq,payload,port_in,
                        port_in_t,port_out_t})) = time() - aux0
      | obsBindElem _ = ~1.0
in
  obsBindElem bindelem
end
```

  - Init function
  - Stop
- SWITCH_DELAY
  - Type: Data collection
  - Nodes ordered by pages
  - Predicate
  - Observer

```
fun obs (bindelem) =
let
  fun obsBindElem (SWITCH'output (1,
                        {s, ip_dst,ip_dst_t,ip_src,ip_src_t,
                        mac_dst,mac_dst_t,mac_src,
                        mac_src_t,num_seq,num_seq_t,
                        payload,payload_t,port_in,
                        port_in_t,port_out,port_out_t, aux0, aux1})) = SW_DELAY
      | obsBindElem _ = ~1.0
in
  obsBindElem bindelem
end
```

  - Init function
  - Stop

Anexo 6: Monitor para medir el tiempo del sistema

SYSTEM_DELAY

- Type: Data collection
- Nodes ordered by pages
- Predicate
- Observer

```
fun obs (bindelem) =
let
  fun obsBindElem (SWITCH'output (1,
                        {s, aux0,aux1,ip_dst,ip_dst_t,ip_src,
                        ip_src_t,mac_dst,mac_dst_t,mac_src,
                        mac_src_t,num_seq,num_seq_t,
                        payload,payload_t,port_in,
                        port_in_t,port_out,port_out_t})) = time() - aux1
    | obsBindElem _ = ~1.0
in
  obsBindElem bindelem
end
```

- Init function
- Stop