Desarrollo de un esquema de enrutamiento dinámico basado en matrices de tráfico para garantizar calidad de servicio en redes definidas por software usando el protocolo OpenFlow

Emanuel Fernando Montoya Gómez

Universidad de Antioquia
Facultad de Ingeniería
Medellín, Colombia
2016

Desarrollo de un esquema de enrutamiento dinámico basado en matrices de tráfico para garantizar calidad de servicio en redes definidas por software usando el protocolo OpenFlow

Emanuel Fernando Montoya Gómez

Trabajo de investigación presentado como requisito para optar al título de:
Magister en Ingeniería de Telecomunicaciones

Director:
Ph.D. Juan Felipe Botero Vega

Grupo de Investigación en Telecomunicaciones Aplicadas, *GITA*
Línea de investigación en modelamiento de redes

Universidad de Antioquia
Facultad de Ingeniería
Medellín, Colombia
2016

Agradecimientos

Este trabajo fue desarrollado en la Universidad de Antioquia en el marco del proyecto *"Plataforma tecnológica para los servicios de Teleasistencia, Emergencias médicas, seguimiento y monitoreo permanente a los pacientes y apoyo a los programas de promoción y prevención"* de la Alianza Regional en Tecnologías de la Información y la Comunicación Aplicadas *(Artica)*, financiado por el Sistema General de Regalías.

Resumen

Las redes definidas por software permiten separar los planos de control y de datos, migrando el control a un ente central, capaz de indicar a los dispositivos de red sobre cómo manejar los flujos de datos. En este trabajo se presenta un esquema de enrutamiento dinámico que monitoriza periódicamente el estado de la red, utilizando el protocolo *OpenFlow*, para calcular su matriz de tráfico y tomar acciones que permitan transmitir las demandas extremo a extremo garantizando la calidad de servicio *(QoS, Quality of Service)* en los caminos establecidos y ahorrar la mayor cantidad posible de energía.

El esquema es probado en un escenario de telemedicina, específicamente de monitorización de pacientes con enfermedades crónicas en hospitalización domiciliaria, como parte del esfuerzo de la Alianza Regional en Tecnologías de la Información y la Comunicación Aplicadas *(Artica)* por ofrecer soluciones a los problemas que se presentan en el sistema de salud Colombiano. Para la evaluación en este escenario, se buscó minimizar el consumo energético total de la red, y los resultados obtenidos se comparan con los de otras dos propuestas que tienen objetivos similares, GreenMST y MdST.

Los resultados obtenidos muestran que el esquema planteado satisface holgadamente las restricciones para tráfico de telemedicina, y mejora tanto el consumo energético como el retraso de las rutas establecidas los resultados de las otras dos estrategias analizadas, logrando mejoras de hasta 66.67 % respecto de GreenMST y 30.77 % respecto de MdST. Tambien se logró establecer que a medida que el retraso mínimo permitido disminuye, aumenta el consumo energético, y disminuye el retraso observado en las rutas. A partir de los resultados obtenidos para el caso de uso seleccionado para la evaluación inicial se desprende que el esquema puede ser aplicado para otros escenarios, como juegos en línea o transmisión de video en tiempo real, y buscando otros objetivos en su modelo de optimización.

Palabras clave: ***SDNs, OpenFlow, QoS routing, Energy efficiency, Traffic Matrix***.

Contenido

1. Introducción

En las redes de datos tradicionales, el enrutamiento de los flujos de datos se hace de manera distribuida y no se tienen en cuenta los cambios que se presentan en las cargas de tráfico de los enlaces de la red en tiempo real, lo que lleva a que, en algunos casos, las redes se congestionen e impide garantizar los parámetros de QoS que las aplicaciones y servicios de red requieren. Normalmente se usan protocolos de enrutamiento basados en el camino más corto (*shortest-path-based*) tales como OSPF (*Open Shortest Path First*) [39] o IS-IS (*Intermediate System- Intermediate System*) [8], que intercambian información entre todos los dispositivos de la red y que ante algún fallo o cambio en las condiciones de esta proceden a actualizar la información en todos ellos, lo cual toma un tiempo considerable que aumenta con el número de dispositivos totales en la red, que a su vez crece a medida que aumenta la cantidad de usuarios conectados. Estos protocolos no tienen la capacidad de reacción suficiente para realizan cambios en las rutas a medida que varía la carga en la red, en tiempo real.

El incremento en el uso de teléfonos inteligentes y la introducción del Internet de las cosas (*IoT, Internet of Things*), hace que se generen nuevas aplicaciones y servicios, lo que obliga a que se amplíe la infraestructura de las redes y un aumento en su utilización para responder ante las nuevas demandas. Es por esto que las redes se hacen cada vez más grandes y complejas, lo que representa un gran reto para los administradores de red que deben gestionar cada vez un mayor número de dispositivos de diferentes fabricantes y que usan distintos entornos y herramientas de administración.

Entre los servicios y aplicaciones que más han aumentado en los últimos años están la voz sobre IP, el video sobre demanda, y los juegos en línea, que tienen requerimientos de retraso mínimos para garantizar la calidad de la experiencia del usuario. También se evidencia una tendencia al incremento en el uso de la telemedicina [1], que incluye servicios como la teleasistencia en procedimientos médicos y el monitoreo remoto de pacientes con enfermedades crónicas. Estos servicios tienen requerimientos estrictos de *QoS*, por lo que es importante garantizar que la red sea capaz de reaccionar ante los cambios en su infraestructura en tiempo real, dado que, de no cumplir con los requisitos, se puede comprometer incluso la vida de las personas que usan este tipo servicios.

Otro inconveniente que se presenta en las redes tradicionales es la dificultad para experimentar con nuevos protocolos de red en las redes en producción [37] ya que se pone en riesgo

la estabilidad de estas, puesto que es difícil que estos se puedan implementar en los equipos de todos los fabricantes y los tiempos de estandarización se hacen muy largos comparados con la velocidad con que cambian las condiciones de uso de las redes. Esto hace evidente la necesidad de encontrar alternativas que permitan probar nuevas estrategias de gestión de red, mientras las redes están operando, para evaluar su correcto funcionamiento, pero sin arriesgar la estabilidad de la red misma.

En los últimos años se ha presentado la arquitectura de redes definidas por software *(Software Defined Networking, SDN)* como una alternativa para facilitar el control de las redes. En estas redes se separa el control de la red (plano de control) de las funciones de redireccionamiento de paquetes (plano de datos), concentrando las labores de control en un ente centralizado. A su vez, se ha propuesto un canal de comunicación entre los dispositivos del plano de datos y el ente centralizado que ejerce el control de la red, siendo el protocolo OpenFlow el estándar más utilizado para esta comunicación [37].

En este trabajo se describe la propuesta, implementación y evaluación de un esquema de enrutamiento dinámico en redes definidas por software, que calcula la matriz de tráfico de la red y monitoriza el estado de sus enlaces en tiempo real, haciendo uso de las características del protocolo *OpenFlow*, para lograr establecer las mejores rutas disponibles en cada momento, de acuerdo a las condiciones reales de la red, como su nivel de carga y las restricciones de calidad de servicio de los flujos de tráfico que circulan por ella.

Tanto la matriz de tráfico como las estadísticas de los enlaces de la red se calculan periódicamente y se utilizan como parámetros de un problema de optimización que permite determinar las rutas que garantizan el flujo de tráfico entre cada par de dispositivos origen-destino de la red, cumpliendo los requerimientos de calidad de servicio y haciendo un uso eficiente de los recursos de la red. Cada que el problema de optimización se resuelve, se actualizan las reglas en los dispositivos del plano de datos para mantener las rutas que ofrezcan mejor uso de los recursos de la red, de acuerdo con su estado actual de utilización, minimizando su consumo energético total.

La evaluación de la propuesta se enfocó en el análisis del comportamiento del consumo energético total de la red y el retraso de las rutas establecidas a medida que cambian las demandas de tráfico en ella. Se realizaron varios experimentos, para determinar el impacto que tenía la variación de la restricción de retraso máximo permitido, lo que evidenció que el consumo energético aumenta y el retraso disminuye a medida que el valor de esta restricción disminuye, siendo más notorio este comportamiento en condiciones de carga media y alta en la red, mientras que ante condiciones de baja carga, tanto el consumo energético como el retraso observado permanecieron dentro del mismo rango de valores sin importar la restricción impuesta.

Luego de este análisis se realizó una comparación con los resultados obtenidos por GreenMST [47] que es una propuesta que busca reducir el consumo energético en SDNs, y también con MdST [38], que es una propuesta similar, que se desarrolló también en el marco de esta investigación, y que busca ofrecer caminos de retraso mínimo además de ahorro energético. La comparación se hizo con los valores obtenidos para la máxima restricción de retraso, que es la que mostraba mayores consumos energético para la propuesta desarrollada, y sometiendo la red a las mismas condiciones de tráfico para las tres propuestas.

Este análisis comparativo mostró que se obtienen mejores resultados tanto en consumo energético como en el retraso de las rutas establecidas para el esquema aquí presentado, obteniendo mejoras de hasta 20.61 % y 34.40 % en cuanto a consumo energético y hasta de 66.67 % y 37.70 % en cuanto a retraso, para GreenMST y MdST respectivamente.

El contenido de este trabajo está organizado de la siguiente manera:

En el capítulo 2 se presenta una descripción del problema que se pretende solucionar con el desarrollo de esta propuesta.

En el capítulo 3 se presentan formalmente los conceptos relevantes que se usan en el desarrollo del trabajo, así como una revisión del estado del arte de cada uno de los temas de interés para su desarrollo.

En el capítulo 4 se describe en detalle cómo se implementa cada etapa de la estrategia de solución.

El capítulo 5 presenta los resultados de la evaluación de la solución.

En el capítulo 6 se presentan las principales conclusiones del trabajo realizado y posibles líneas futuras de trabajo.

2. Definición del problema

En la arquitectura de red tradicional, donde el control se distribuye entre todos los dispositivos de red, las estrategias de enrutamiento dinámico que se usan no tienen en cuenta las demandas de tráfico que hay en la red en tiempo real y es difícil hacer pruebas con nuevos protocolos y herramientas de administración, debido a que se pueden presentar condiciones no deseadas que pongan en riesgo la estabilidad de la red.

Normalmente los administradores de red son los que hacen la optimización de las rutas, mediante el uso de las técnicas de ingeniería de tráfico, analizando estadísticas de la red en determinados intervalos de tiempo (semanas o meses) con el fin de establecer las rutas que consideren más convenientes para lograr el mejor funcionamiento de la red en un futuro, esperando que las demandas sigan un comportamiento similar al que se tuvo según las estadísticas utilizadas.

Una de las herramientas fundamentales para la ingeniería de tráfico son las matrices de tráfico, pero estas son difíciles de obtener en tiempo real para las grandes redes de datos actuales, debido a que el número de nodos de entrada-salida es muy elevado y las cargas de tráfico en cada uno de los enlaces son muy grandes, lo que supone el uso de una gran cantidad de dispositivos adicionales para medir los flujos necesarios para determinar la matriz de tráfico.

Ante la falta de la infraestructura necesaria para medir los flujos en todos los enlaces de la red [55], se recurre al uso de métodos estadísticos para su cálculo [58, 18, 54, 9, 60, 48, 41, 26], pero estos métodos incluyen errores de estimación que afectan el rendimiento de las estrategias planteadas para adquirir las matrices de tráfico de manera confiable, y de las acciones que se tomen basadas en ellas.

En los últimos años también se ha incrementado el uso de la virtualización, lo que implica otros desafíos específicos para los administradores de red para configurar las redes virtuales, ya que los flujos de tráfico cambian constantemente de ubicación e intensidad a través del tiempo [24] y los administradores se ven obligados a tratar todos estos desafíos dependiendo solamente de las herramientas de bajo nivel que les permita usar cada fabricante para sus dispositivos.

Algunas limitaciones de los enrutadores de las redes tradicionales son [24]:

- Solamente conocen el ancho de banda y costo de los enlaces directamente conectados a ellos, pero no tienen un conocimiento global de la red y todos sus enlaces.
- Debido a las características propias de los algoritmos que usan, la información que tienen de la red se basa en decisiones anteriores, lo que quiere decir que las información siempre está desactualizada respecto al momento propio de tomar la decisión.
- Cada enrutador tiene su propio algoritmo de enrutamiento, lo que hace que el control sea descentralizado, y no se pueda predecir el comportamiento exacto de la red completa.
- Cualquier cambio que se haga en la infraestructura o en las políticas de la red tomará tiempo para ser aplicado a la red completa.

Teniendo en cuenta los antecedentes enunciados, se hace evidente la necesidad de desarrollar herramientas que permitan administrar la red de una manera mucho más sencilla y flexible que las que hay disponibles actualmente.

Otro desafío importante para el desarrollo de las redes de datos a nivel mundial es el constante aumento en el consumo energético y su impacto en la emisiones de carbono globales. Para 2006 el consumo energético sólo de los elementos de red de los centros de datos en Estados Unidos fue de 3 TWh, mientras que en el año 2000 había sido tan solo de 1,4 TWh [11]. En el año 2011 se estimaba que las Tecnologías de la Información y Telecomunicaciones (*TIC*) eran responsables de entre 2 y 4 % del total de emisiones de carbono a nivel mundial, y se espera que para 2020 ese porcentaje se duplique, siendo las redes de telecomunicaciones partícipes de al menos una sexta parte de esa cifra [59]. Este incremento constante en el consumo energético, y en las emisiones de carbono, está relacionado con las cifras de incremento en el uso de Internet, que llega a tasas de 20 % a nivel mundial, y que alcanza a ser de hasta 40 % en países en desarrollo [59].

Debido a las restricciones propias de las redes tradicionales, como la descentralización del control, la dificultad para obtener la matriz de tráfico en tiempo real y la necesidad de intervención de los administradores usando las herramientas de bajo nivel que ofrecen los fabricantes de los equipos de red para modificar las rutas en los dispositivos, no es posible realizar un enrutamiento en tiempo real que permita hacer un uso eficiente de los recursos de la red a medida que varían los niveles de demanda, para minimizar su consumo energético.

A diferencia de las redes tradicionales, en la arquitectura de redes definidas por software, el control se concentra en un ente lógico centralizado, llamado controlador, que se comunica con los dispositivos del plano de datos utilizando un canal seguro y una API (*Application Programming Interface*) de comunicación, lo que permite al controlador obtener información de cada uno de los dispositivos para tener una imagen global del estado de la red en tiempo

real. Esta característica permite establecer las reglas de enrutamiento "en caliente"sin afectar la operación de la red, a diferencia de lo que ocurre en las redes tradicionales, donde las reglas de enrutamiento se instalan en los dispositivos antes de que la red entre en funcionamiento.

En este trabajo se describe el planteamiento, desarrollo y evaluación de un esquema de enrutamiento dinámico en redes definidas por software que permite hacer un uso eficiente de los recursos de la red, según varían las demandas de tráfico en la red. Este esquema utiliza características de la versión 1.3 del protocolo *OpenFlow* [44] para determinar la matriz de tráfico de la red y el retraso de sus enlaces, que se utilizan como parámetros de un problema de optimización que se resuelve periódicamente –*con intervalos de minutos o incluso segundos*– para determinar las rutas que garanticen el cumplimiento de las restricciones de calidad de servicio minimizando el consumo energético total de la red. Las rutas calculadas se instalan en los dispositivos del plano de datos sin intervención directa del administrador de la red, utilizando el protocolo *OpenFlow*.

3. Marco teórico y estado del arte

En esta sección se describen los principales conceptos utilizados en el desarrollo de la propuesta, y los trabajos más relevantes encontrados relacionados con cada uno de ellos.

3.1. Marco teórico

3.1.1. Protocolos de enrutamiento dinámico

Los protocolos de enrutamiento usados en las redes actuales se clasifican en 2 grupos, según se usen en entornos intra-dominio (dentro de un sistema autónomo) o inter-dominio (entre sistemas autónomos). Los que se usan en entornos intra-dominio, que es el escenario inicial de nuestra propuesta, son llamados protocolos de pasarela interior (*Interior Gateway Protocol*, IGP), mientras que los usados en entornos inter-dominio son llamados protocolos de pasarela exterior (*Exterior Gateway Protocol*, EGP).

Los protocolos IGP se pueden dividir en 2 grupos, según estén basados en algoritmos de vector de distancia, o de estado de enlace.

Protocolos de vector de distancia

En estos protocolos, cada nodo mantiene una vector o lista donde se almacenan los costos (número de saltos u otras métricas) asociados para llegar a cada uno de los nodos de la red, y comparte esa información periódicamente, o cada que se detecta un cambio en la topología, con sus vecinos, que usan esa información para crear sus propias tablas de enrutamiento; este mecanismo hace que tome un tiempo para que todos los nodos tengan una vista consistente de la red. En este proceso de convergencia se pueden presentar algunos inconvenientes, como bucles o el conocido como conteo al infinito, para el que se han presentado algunas soluciones parciales [43].

Los protocolos de vector distancia más utilizados son:

- *Routing Information Protocol, RIP*, en sus versiones RIPv2 [34] y RIPng [35], que son las implementaciones más simples de estos protocolos, tomando la unidad como métrica del costo de cada enlace, y asumiendo 16 como infinito (destino inalcanzable).

- *Interior Gateway Routing Protocol, IGRP* [50], y su versión mejorada *(Enhanced), EIGRP* [15], que son protocolos desarrollados por *CISCO*.

Protocolos de estado de enlace

Estos protocolos utilizan un modelo de base de datos distribuida y replicada. Los enrutadores intercambian paquetes en los que indican el estado de todas sus interfaces, o sea que se comparte información acerca de sus conexiones directas y no toda la tabla de enrutamiento como en los protocolos de vector de distancia. Cada nodo conoce cómo alcanzar sus vecinos directos y, si se garantiza que ese conocimiento llegue a todos los demás nodos, entonces cada nodo tiene suficiente información para crear un mapa de la red.

Cada nodo envía la información sobre el estado y costo de los enlaces a sus vecinos directos, que a su vez le comunica esa información a sus propios vecinos directos, hasta alcanzar la totalidad de los nodos de la red; ese envío de información se hace periódicamente, o cuando se detecta un cambio en la red, al igual que en los protocolos de vector distancia. Una vez que un nodo tiene la información de todos sus vecinos, puede crear el mapa de la topología y se aplica el algoritmo de Dijkstra para calcular la ruta de menor costo para llegar a los demás nodos de la red [43].

Los protocolos de estado de enlace más usados son:

- *Open Shortest Path First, OSPF*.
- *Intermediate System to Intermediate System, IS-IS*, que es un protocolo de capa 2.

En el entorno inter-dominio, el protocolo normalmente usado es *BGP (Border Gateway Protocol)* en su versión *BGP-4* [49].

3.1.2. Redes Definidas por Software

Las redes definidas por software *(Software Defined Networking, SDNs)* son una arquitectura emergente en la que se separa el control de la red del manejo de los datos propiamente dicho, haciendo el control programable [10]. El control lo ejerce un dispositivo central, llamado controlador, que es el responsable de indicar a los dispositivos de red cómo deben manejar los flujos de datos que llegan, mientras que estos últimos solo se encargan de la conmutación de los paquetes hacia sus destinos finales, siguiendo las instrucciones dadas por el controlador. Cuando llega algún paquete que el dispositivo de red no sepa cómo procesar, debe preguntarle al controlador para que lo instruya sobre qué hacer. Esta migración del control, que anteriormente estaba repartido entre todos los dispositivos de la red, permite que se haga una abstracción de la infraestructura para las aplicaciones y servicios de red, que pueden tratarla como una entidad lógica o virtual [10]. En SDNs ya no existen enrutadores, puesto

que los dispositivos de red no tienen inteligencia que les permita decidir por sí mismos qué deben hacer con los paquetes que llegan a ellos, por lo que se pueden denominar, de manera general, conmutadores a todos los dispositivos del plano de datos.

En las SDNs se define una interfaz denominada *Southbound* para la interacción entre la capa de control y la capa de datos o infraestructura, que le permite al controlador hacer las configuraciones necesarias en los dispositivos de red. De igual forma, la capa de control ofrece una interfaz, llamada *Northbound*, que permite la interacción con aplicaciones de negocio, que hacen uso de los servicios de red que ofrece el controlador. En la figura **3-1** se ilustra la arquitectura de una SDN, mientras que la figura **3-2** muestra la diferencia que existe entre una red tradicional y una SDN.

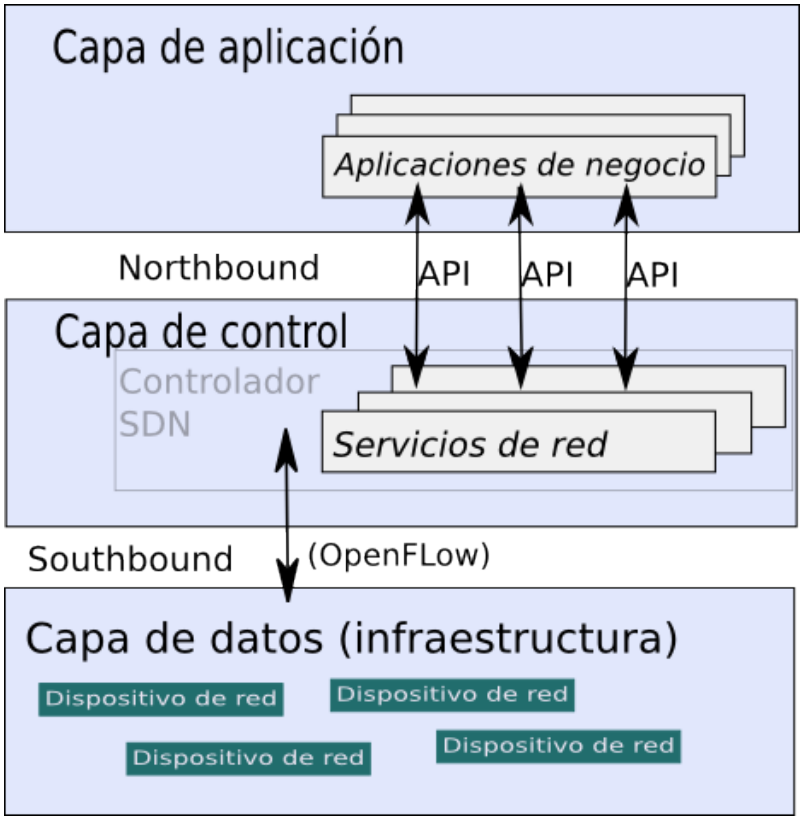


Figura 3-1.: Arquitectura de una SDN [10].

3.1.3. OpenFlow

OpenFlow [37] es un protocolo que permite la comunicación entre el controlador y los dispositivos de red dentro de la arquitectura SDN, es decir, es una implementación de la interfaz *Southbound*. Este protocolo hace uso de una API que le permite al controlador comunicarse

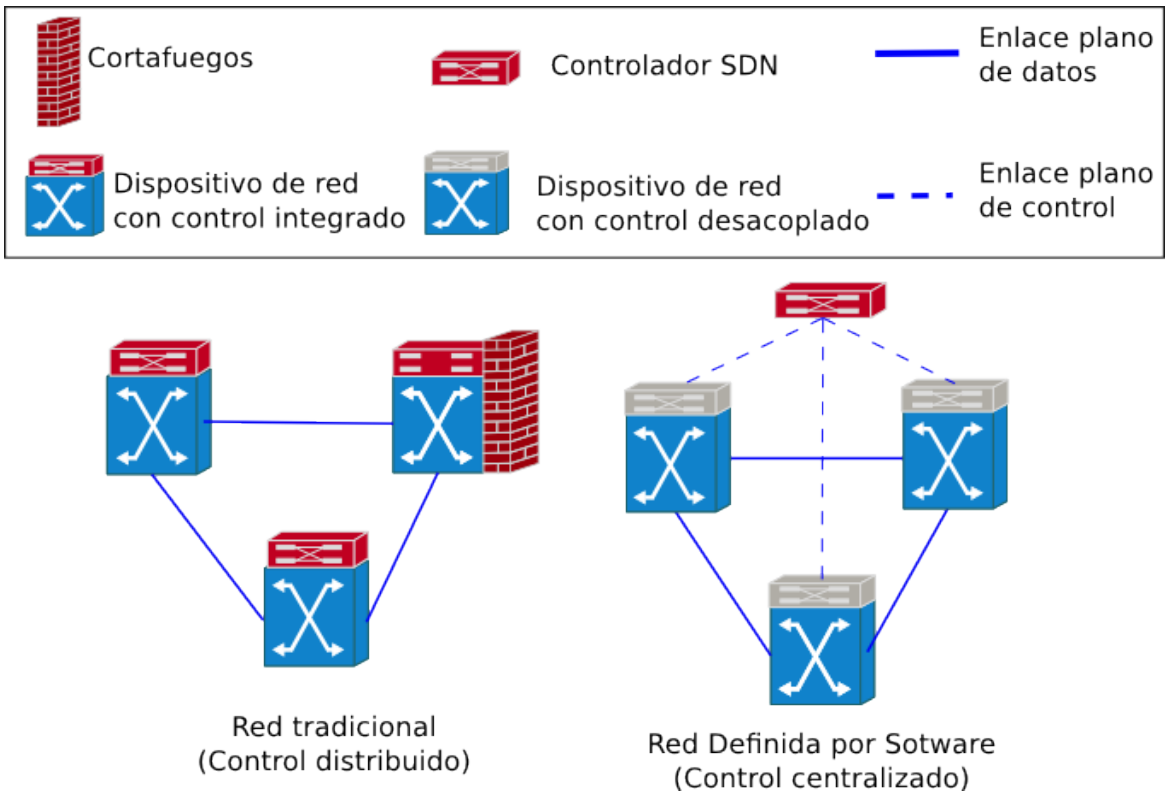


Figura 3-2.: Diferencia entre una red tradicional y una SDN [40].

con los demás dispositivos de la red, sin que los fabricantes tengan que exponer el funcionamiento interno de sus equipos, y se implementa sobre SSL (*Secure Sockets Layer*) o TLS (*Transport Layer Security*) para garantizar un canal seguro entre los conmutadores y el controlador. Las primeras redes que implementaron *OpenFlow* entraron en funcionamiento en campus universitarios, precisamente porque fue propuesto como una alternativa para los investigadores, para que pudieran probar sus desarrollos dentro de redes en producción, ante las restricciones que impone el software propietario de los dispositivos de red [37], ya que la mayoría de enrutadores y conmutadores actuales tienen tablas de flujo específicas de cada fabricante, pero *OpenFlow* brinda un mecanismo estándar para programar las tablas de flujo de conmutadores de diferentes fabricantes.

Es importante marcar la diferencia entre SDN y *OpenFlow*, ya que se tiende a confundir estos dos términos: para entender esta diferencia se hace una analogía con un sistema operativo de computador, diciendo que SDN hace la abstracción global de la red de la misma manera que el sistema operativo hace la abstracción global del sistema de cómputo, y que SDN se comunica con los dispositivos de la red usando *OpenFlow*, de la misma manera que el sistema operativo se comunica con el hardware a través de controladores de dispositivos [33].

En [44] se describen los requerimientos de un conmutador *OpenFlow*, sus componentes y funciones básicas, así como el protocolo *OpenFlow* para manejarlo desde un controlador remoto.

A continuación, se definen los términos principales de la especificación *OpenFlow*:

- Byte: Es un octeto de 8 bits.
- Paquete: Es una trama *Ethernet*, incluyendo su encabezado y payload.
- Pipeline: Es un conjunto de tablas de flujo enlazadas que permiten hacer comparación de campos, redireccionamiento, y modificaciones en el paquete dentro del conmutador *OpenFlow*.
- Puerto: Es por donde los paquetes entran y salen al pipeline. Puede ser un puerto físico, un puerto lógico que define el conmutador, o un puerto reservado que define el protocolo *OpenFlow*.
- Tabla de flujos: Es una etapa del pipeline y contiene entradas de flujo.
- Entrada de flujo: Es un elemento dentro de una tabla de flujos que se usa para comparar y procesar paquetes. Contiene un conjunto de campos de concordancia, para hacer la comparación con los paquetes, un campo de prioridad para determinar su precedencia respecto de otras entradas de flujo, un conjunto de contadores para mantener estadísticas de paquetes, y un conjunto de instrucciones para aplicar en caso de que el paquete concuerde.
- Campo de concordancia (*Match Field*): Es un campo contra el cual el paquete se compara, puede ser parte del encabezado del paquete, su puerto de entrada, o valores de metadatos. Un campo de concordancia puede ser un comodín (que concuerde con cualquier valor) y, en algunos casos, una máscara de bits.
- Metadatos: Es un valor de registro enmascarable que se usa para pasar información de una tabla a la siguiente.
- Instrucción: Las instrucciones están ligadas a una entrada de flujo, y describen cómo se procesa un paquete que concuerde con la entrada de flujo. Una instrucción contiene acciones para modificar el procesamiento en el pipeline, tal como enviar el paquete a otra tabla, un conjunto de acciones para aplicar inmediatamente al paquete, o para agregar al conjunto de acciones que se aplicarán posteriormente.
- Acción: Es una operación que modifica el paquete, decrementando su campo TTL por ejemplo, o lo encamina hacia un puerto específico. Las acciones se pueden especificar como parte del conjunto de instrucciones asociadas con una entrada de flujo, o en

Tabla 3-1.: Componentes de una entrada de medidor en la tabla de medidores.

Campo	Explicación
Identificador de medidor	Es un número de 32 bits que identifica unívocamente al medidor.
Bandas de medición	Es una lista bandas de medición, cada una de las cuales indica la forma en que se deben procesar los paquetes y la tasa a que se aplica.
Contadores	Se actualizan cuando un paquete es procesado por un medidor.

un conjunto de acciones asociadas con una entrada de grupo. Las acciones se pueden aplicar inmediatamente, o se acumulan en un conjunto de acciones asociadas al paquete.

- Conjunto de acciones (*Action Set*): Son acciones asociadas con el paquete, que se van acumulando a medida que este se procesa en cada una de las tablas, y que se ejecutan cuando el conjunto de instrucciones indican que el paquete debe salir del *pipeline*.
- Grupo: Es una lista de conjuntos de acciones acompañada de alguna forma de escoger uno o más de esos conjuntos para que se apliquen a cada paquete procesado.
- Contenedor de acciones (*Action Bucket*): Es un conjunto de acciones y parámetros asociados que se definen para grupos.
- Etiqueta (*Tag*): Es un encabezado que se puede insertar o remover de un paquete mediante acciones *push* o *pop*.
- Etiqueta externa (*Outermost Tag*): Es la etiqueta que aparece más al inicio del paquete.
- Controlador: Entidad que interactúa con el conmutador usando el protocolo *OpenFlow*.
- Medidor *(meter)*: Es un elemento del conmutador que puede medir y controlar la tasa de paquetes. El medidor activa una banda de medición (*meter band*) si la tasa de paquetes o de bytes que cuenta el medidor supera el nivel predefinido. Si la banda de medición descarta el paquete, se le llama limitador de velocidad (*Rate Limiter*).

Un elemento interesante para el desarrollo de nuestra propuesta es precisamente el último de los medidores. Cada entrada de flujo puede especificar un medidor que, a su vez, tiene un contador que se actualiza cada que se procesa un paquete. Para cada medidor se pueden definir una o varias bandas de medida, cada una de las cuales consiste de una tasa y una orden a ejecutar cuando el contador llegue a ese nivel. En la Tabla **3-1** se muestran los componentes de una entrada de medidor, en la que se tiene un identificador, una o más bandas y unos contadores que se incrementan cada que un paquete concuerda con alguno de los flujos ligados a el medidor.

En la Figura **3-3** se observan los componentes principales de un conmutador *OpenFlow*, que está constituido por un canal seguro para comunicarse con el controlador, una o más tablas de flujos que constituyen el pipeline, y una tabla de grupos, que son las que se usan para inspeccionar y redireccionar los paquetes que llegan al conmutador.

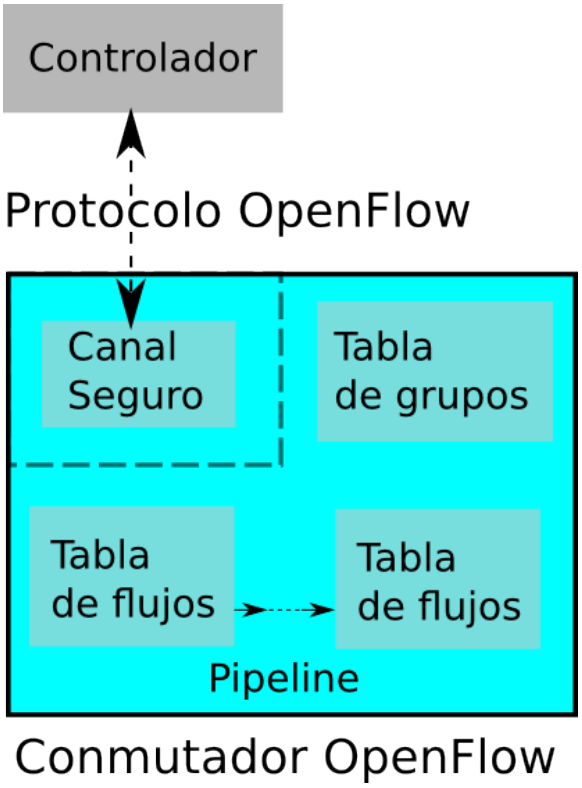


Figura 3-3.: Componentes básicos de un conmutador *OpenFlow* [44].

Cada tabla de flujos consiste de varias entradas flujos que, a su vez, están conformadas por contadores, campos de coincidencia, y un conjunto de instrucciones para aplicar a los paquetes que coincidan. En la Tabla **3-2** se especifica cada uno de los componentes de una entrada de flujo, y su función.

Existe un tipo de entrada de flujo especial, denominada entrada de flujo por defecto *(table-miss flow entry)*, que se define como aquella entrada que tiene prioridad cero y que tiene campo de coincidencia vacío, para que cualquier paquete concuerde con ella. Normalmente, la instrucción de esta entrada de flujo hace que el paquete sea enviado al controlador para que decida qué debe hacerse con él, aunque no es obligatorio que sea así.

Cuando un paquete llega al conmutador, se verifica si coincide con algunas de las entradas de la primera tabla de flujos, en caso de coincidir con alguna de ellas, se actualizan los con-

Tabla 3-2.: Campos de una entrada de flujo.

Campo	**Explicación**
Match fields	Campos de coincidencia para comparar con el paquete, tales como el puerto de entrada, encabezados del paquete o metadatos especificados en tablas anteriores.
Priority	Es un número que expresa la prioridad de la entrada de flujo.
Counters	Contadores que se actualizan cada que un paquete concuerda con la regla.
Instructions	Conjunto de instrucciones a ejecutar sobre un paquete que concuerda con la regla.
Timeouts	Cantidad máxima de tiempo que puede existir la entrada de flujo o tiempo de espera sin que haya concordancia de ningún paquete, antes de que la entrada expire y sea borrada de la tabla.
Cookie	Valores de datos elegidos por el controlador, y que se pueden usar para filtrar estadísticas, modificar o eliminar flujos, pero que no son usados al momento de procesar paquetes.

tadores correspondientes y se ejecutan las instrucciones especificadas; si el paquete coincide con más de una entrada de flujo, se ejecutan sólo las acciones de la entrada que tenga mayor prioridad. Dependiendo de las instrucciones se puede pasar el paquete para que sea analizado en otras tablas de flujos, o ejecutar finalmente las acciones a que haya lugar sobre él. En caso de que el paquete no coincida con ninguna de las entradas de la tabla de flujos, y no haya entrada por defecto, el paquete se descarta. Esta es la forma como se procesan los paquetes en un conmutador *OpenFlow*, y se ilustra en la Figura **3-4**.

Existen tres tipos de mensajes, que se enuncian a continuación, y que le permiten al controlador gestionar el funcionamiento de los conmutadores *OpenFlow*:

- Controlador-conmutador, son iniciados por el controlador y se usan para gestionar directamente o para indagar por el estado del conmutador. Estos mensajes pueden o no tener respuesta desde el conmutador, según sea su propósito.

- Asíncronos, son iniciados por el conmutador y se usan para informar al controlador sobre eventos en la red o cambios en el estado del conmutador.

- Simétricos, son iniciados por el controlador o el conmutador, y se envían sin necesidad de una solicitud previa.

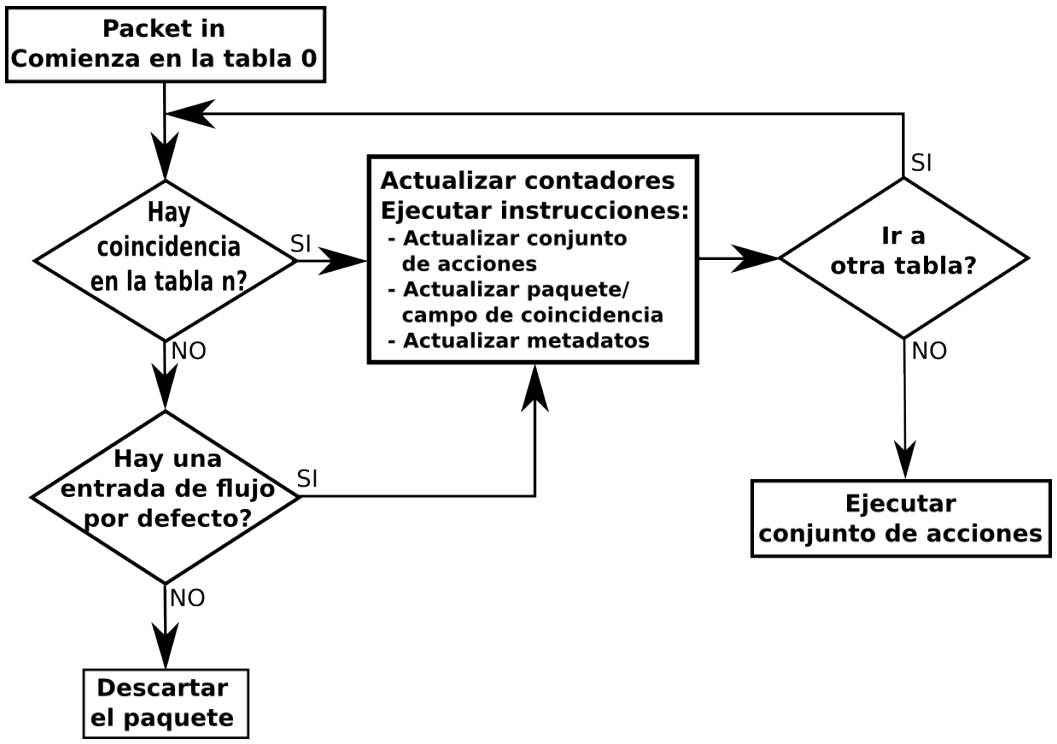


Figura 3-4.: Flujo de paquetes en un conmutador *OpenFlow* [44].

En la Tabla **3-3** se enuncian los mensajes pertenecientes a cada uno de los grupos mencionados, y su uso indicado.

Tabla 3-3.: Tipos de mensajes *OpenFlow*.

Tipo	**Nombre**	**Uso**
Controlador-Conmutador	Features	Preguntar al conmutador sobre sus capacidades, a lo que el conmutador debe responder con un mensaje en el que las especifique.
	Configuration	Consultar o fijar valores a los parámetros de configuración. El conmutador responde solo cuando es una consulta.
	Modify-state	Gestionar el estado del conmutador; se usan principalmente para agregar, modificar y eliminar entradas de flujo, o para fijar propiedades de los puertos del conmutador.
	Read-state	Recolectar información del estado del conmutador.
	Packet-out	Enviar paquetes por los puertos del conmutador, y redireccionar paquetes recibidos mediante mensajes Packet-in; deben contener un paquete o el identificador de un buffer donde haya un paquete almacenado en el conmutador, así como una lista de acciones a aplicar al paquete.
	Barrier	Asegurar de que las dependencias entre mensajes se cumplen, o recibir notificaciones de que las operaciones se completan.

	Role-Request	Fijar o consultar el rol de su canal *OpenFlow*. Se usa, sobre todo, cuando el conmutador se conecta a múltiples controladores.
	Asynchronous-configuration	Filtrar los mensajes asíncronos que el controlador quiere recibir por su canal *OpenFlow*, o para consultar cuáles son esos mensajes.
Asíncronos	Packet-in	Transferir el control de un paquete del conmutador al controlador, se envía el paquete para que el controlador decida qué debe hacerse con él.
	Flow-removed	Informar al controlador cuando se elimina una entrada de alguna de las tablas de flujos.
	Port-status	Informar al controlador cuando hay un cambio en algún puerto.
	Error	Informar al controlador cuando hay algún problema.
Simétricos	Hello	Se intercambian entre el conmutador y el controlador en el establecimiento de la conexión.
	Echo	Se pueden enviar desde el conmutador o el controlador y se debe retornar una respuesta. Se usa para verificar que la conexión entre ambos permanezca activa, principalmente.
	Experimenter	Brinda una forma estándar para que los conmutadores *OpenFlow* ofrezcan funcionalidades adicionales. Es un área de ensayos para futuras revisiones de *OpenFlow*.

Un conmutador *OpenFlow* debe soportar tres tipos de puertos: físicos, lógicos y reservados. Los puertos físicos son aquellos que corresponden a las interfaces de red físicas (interfaces hardware) del conmutador, mientras que los puertos lógicos son abstracciones de más alto nivel que se pueden mapear a varios puertos físicos y no corresponden exactamente a las interfaces hardware de red. La diferencia entre un puerto físico y uno lógico es que los paquetes procesados por los puertos lógicos pueden tener un campo de metadatos llamado *Tunnel-ID* asociado con él, y que cuando se envía un paquete al controlador que haya ingresado por uno de ellos, se indica tanto el puerto lógico como el puerto físico correspondiente.Por otro lado, los puertos reservados especifican acciones de direccionamiento, como enviar al controlador o inundar. Aunque se definen otros puertos reservados, es obligatorio que el conmutador implemente los que se declaran a continuación:

- *ALL*: Representa todos los puertos por los que el conmutador puede encaminar un paquete. Solo se puede usar como puerto de salida, en cuyo caso se envía una copia del paquete por todos los puertos estándar, exceptuando el puerto de entrada y los puertos que se hayan configurado explícitamente para no encaminar paquetes.

- *CONTROLLER*: Es el puerto que comunica con el controlador *OpenFlow*, y se puede usar como puerto de entrada o de salida. Cuando se usa como puerto de salida, se encapsula el paquete en un mensaje packet-in y se envía al controlador, usando el protocolo *OpenFlow*. Cuando se usa como puerto de entrada, identifica un paquete que se origina en el controlador.

- *TABLE*: Representa el inicio del pipeline, sólo es válido en una acción de salida en la lista de acciones de un mensaje packet-out, y envía el paquete a la primera tabla de flujos, para que sea procesado por el pipeline.

- *IN PORT*: Identifica el puerto por el que el paquete entra al conmutador. Solo puede usarse como puerto de salida, cuando se quiere enviar el paquete por el mismo puerto por el que ingresó al conmutador.

- *ANY*: Es un valor especial usado en algunos comandos *OpenFlow* cuando no se especifica un puerto (puerto comodín). No se puede usar como puerto de entrada ni de salida.

Adicionalmente, se define un grupo de puertos como puertos estándar, que está constituido por los puertos físicos, lógicos y el puerto reservado *LOCAL* (no obligatorio). Este grupo de puertos se pueden usar como entrada o salida de paquetes, también en grupos, y tienen contadores de paquetes por puerto.

3.1.4. Matrices de tráfico

Las matrices de tráfico son una representación del flujo de tráfico entre cada par de puntos de entrada-salida de una red en un intervalo de tiempo determinado [53]. Las matrices de tráfico se pueden representar en diferentes niveles de agregación, en los que un punto de entrada-salida puede ser un puerto de un enrutador, un enrutador en sí mismo, o incluso un sistema autónomo. En la figura **3-5** se ilustra la matriz de tráfico vista desde diferentes niveles de agregación, donde la diagonal de la matriz representa el tráfico interno en el punto de entrada-salida, que no atraviesa la red.

3.1.5. Modelo energético de elementos de red

Es comúnmente aceptado que el consumo energético de los elementos de red aumenta linealmente desde un valor E_0, que corresponde a un estado de reposo, hasta un valor M cuando se utilizan al máximo de sus capacidades [5]. Se considera que el consumo del dispositivo es cero sólo cuando no se está utilizando, momento en el cual el dispositivo entra en un estado de sueño [6].

En la Figura **3-6** se ilustra el comportamiento del consumo energético de los dispositivos de red, según su utilización. Se destacan dos casos especiales de este modelo: los dispositivos totalmente proporcionales, donde $E_0 = 0$; y los de consumo independiente de la utilización, que consumen lo mismo sin importar el nivel de utilización, donde $E_0 = M$.

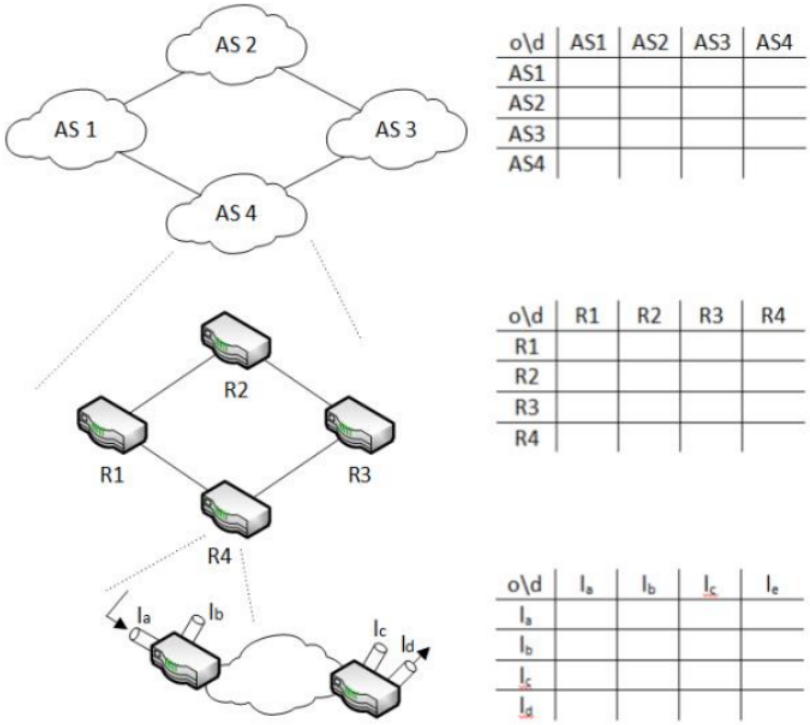


Figura 3-5.: Matriz de tráfico [52].

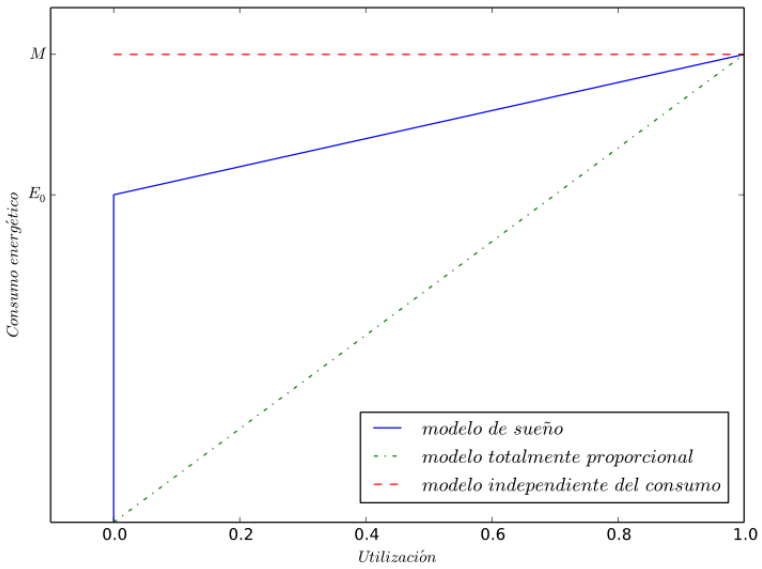


Figura 3-6.: Modelo energético de elementos de red [5].

3.2. Estado del arte

En esta sección se presentan los trabajos más importantes que se encontraron sobre los temas de mayor relevancia para el desarrollo de nuestra propuesta.

3.2.1. Matrices de tráfico

Como nuestro esquema utiliza matrices de tráfico como entrada fundamental para determinar la rutas que conectan cada par de dispositivos de entrada-salida de la red, en SDNs, a continuación se presentan algunos trabajos que han abordado el desafío de determinar las matrices de tráfico usando *OpenFlow* en SDNs.

OpenTM [55] hace uso de las características de *OpenFlow*, leyendo los contadores de paquetes y de bytes para los flujos activos desde los conmutadores, con lo que se hace el mínimo esfuerzo en los elementos de la red, y se obtiene una mayor precisión, pues no se hace ningún tipo de simplificación matemática o aproximación estadística.

En [52] se plantea la obtención de la matriz de tráfico de una forma distribuida, en la que cada nodo es responsable por calcular una parte de la matriz, usando las características de la versión 1.0 de *OpenFlow*. Los resultados obtenidos en esta investigación muestran que es posible obtener las matrices sin retraso, incluso en presencia de grandes flujos de tráfico. Comparado con OpenTM, que también hace la medida con datos directos de los conmutadores, esta nueva propuesta usa menos mensajes *OpenFlow* una vez que los conmutadores se sincronizan, pero usa más recursos de CPU, por lo que, al final del estudio, se plantean algunas mejoras que podrían servir para evitar el uso innecesario de CPU en el controlador.

Por su lado, en [29] se utiliza la misma técnica usada en [52], pero usando nuevas características de la versión 1.3 de *OpenFlow*, como los medidores, que se usan solo en los dispositivos de entrada-salida de la red, para contar la cantidad de tráfico de los flujos agregados que van hacia cada uno de los demás dispositivos de entrada-salida, lo que baja el número de dispositivos consultados, así como la cantidad de consultas que hay que hacer a cada uno de ellos, y el procesamiento adicional que se necesitaba en el controlador para agregar los flujos individuales.

3.2.2. Monitoreo de parámetros de QoS

Otra característica fundamental de nuestra propuesta es la garantía de QoS a las aplicaciones y servicios de la red, por lo que es fundamental poder medir estos parámetros en cada uno

de los enlaces, para poder determinar los caminos que cumplan con las restricciones establecidas. Son varias las herramientas desarrolladas con el fin de monitorizar el estado de los enlaces en las SDNs, pero muchas requieren de hardware adicional. A continuación se presentan algunas propuestas que buscan cumplir este objetivo sin utilizar dispositivos externos.

En [22] se presenta un análisis de cómo obtener estadísticas de red en un entorno de redes definidas por software, usando las características de *OpenFlow* para obtener estadísticas de los dispositivos de la red; también se hace un análisis del efecto que tiene la frecuencia con que se hacen las consultas a los conmutadores de la red en la utilización de los enlaces y en la precisión de las medidas. Como prueba de concepto se mide el ancho de banda disponible, en los enlaces de la red como:

$$UB = [(Bt_2 - Bt_1)/P] * [8bits],$$

donde UB es el ancho de banda utilizado en el intervalo de tiempo $P = t_1 - t_2$, y Bt_1 y Bt_2 son los números de bytes transmitidos en los instantes de tiempo t_1 y t_2, respectivamente. Los resultados obtenidos permitieron comprobar que es posible obtener gran precisión en las medidas, utilizando la estrategia planteada; los valores calculados eran un poco superiores a los reales, pero se explica la diferencia por el tráfico de paquetes de control generados por la misma aplicación.

Los autores de [57] presentan una estrategia para medir el retraso, el rendimiento *(throughput)*, y las pérdidas en los caminos de una red definida por software. Estos dos últimos se deducen al consultar los valores de los contadores de los conmutadores de la red, mientras que para el retraso se envían mensajes de prueba que siguen la ruta del camino. El retraso está dado por

$$t_{retraso} = t_{llegada} - t_{envío} - \frac{1}{2}(RTT_{s1} + RTT_{s2}),$$

donde RTT_{s1} y RTT_{s2} son los tiempos de ida y vuelta (*Round-Trip Time, RTT*) de los enlaces entre el controlador y los dispositivos inicial y final del camino al que se le está midiendo el retraso, mientras que $t_{llegada}$ y $t_{envío}$ son los tiempos en que se envía y se recibe el mensaje de prueba que va por el camino analizado, respectivamente. Para medir RTT_{s1} y RTT_{s2}, se envían paquetes de prueba desde el controlador hasta cada dispositivo de red, que son devueltos inmediatamente al controlador. Para mejorar la fiabilidad de la medida de estos tiempos, es necesario establecer una VLAN exclusivamente para el envío de los paquetes de prueba. Los resultados experimentales permitieron verificar la confiabilidad de las medidas de retraso y rendimiento, y además que las medidas de pérdidas de paquetes son un buen indicador de posibles degradaciones de servicio. Para disminuir el *overhead*, se categorizan los flujos, para minimizar la influencia de flujos insignificantes.

Tabla 3-4.: Valores de retraso en enlaces dirigidos según [3].

Condición	Retraso $A \longrightarrow B$	Retraso $B \longrightarrow A$
$\frac{X}{T} \leq C$ y $\frac{Y}{T} \leq C$	$\frac{RTT}{2}$	$\frac{RTT}{2}$
$\frac{X}{T} \geq C$ y $\frac{Y}{T} \leq C$	$(RTT + (\frac{X}{C} - T))/2$	$(RTT - (\frac{X}{C} - T))/2$
$\frac{X}{T} \geq C$ y $\frac{Y}{T} \geq C$	$(RTT + (\frac{X}{C} - T) - (\frac{Y}{C} - T))/2$	$(RTT - (\frac{X}{C} - T) + (\frac{Y}{C} - T))/2$

En [46] se presenta una propuesta para medir el retraso en los enlaces de la red, en la que el retraso se calcula como

$$Retraso(s1, s2) = T_{total} - \frac{1}{2}(RTT_{s1} - RTT_{s2}) - C,$$

y donde $T_{total} = t_{llegada} - t_{envío}$ y C es un factor de calibración experimental. RTT_{s1} y RTT_{s2} se miden como el tiempo entre el envío de un mensaje *OpenFlow STATISTICS_REQUEST*, que el controlador envía a los dispositivos para otros propósitos, y su correspondiente respuesta *STATISTICS_REPLY*. Como puede observarse, la medida del retraso se hace siguiendo la misma estrategia que en la propuesta anterior, con la única diferencia del factor de calibración experimental C. Los resultados reportados muestran más de 99 % de precisión, comparado con el comando ping, usando 81 % menos ancho de banda, dado que se usan paquetes de prueba de solo 24 bytes.

Por otro lado, los autores de [3] presentan una estrategia para calcular el retraso de los enlaces de una red, teniendo en cuenta su ocupación. La estrategia consiste en enviar paquetes de prueba que se intercambian entre los dispositivos origen y destino de cada enlace, recorriéndolo varias veces, con el objetivo de aumentar la longitud total del recorrido, para hacer despreciable el impacto del trayecto entre el controlador y los dispositivos de red. El paquete de prueba se envía desde el controlador al dispositivo inicial del enlace, indicándole que lo envíe por el puerto correspondiente para iniciar el recorrido, y que envíe una copia del mismo de vuelta al controlador, momento en el cual se inicia el conteo del tiempo del recorrido. Se establecen varias reglas en ambos dispositivos que van cambiando elementos del encabezado del paquete para que circule indefinidamente entre los dos extremos del enlace y cada cierto número de vueltas se envíe una copia al controlador para calcular el tiempo total del recorrido y determinar de esta forma el RTT *Round Trip Time*. Para determinar el retraso del enlace en cada sentido, se considera que el enlace entre dos dispositivos A y B tiene capacidad C, X e Y son las cantidades de bytes enviados desde A hacia B, y desde B hacia A, respectivamente en un intervalo de tiempo T, que se obtienen desde los dispositivos mediante mensajes *OpenFlow*. Según sean los valores de X e Y, el retraso de los enlaces se calcula como se indica en la Tabla **3-4**.

También en [3], se presenta otra estrategia que crea una representación de la red como un grafo dirigido, al que se le aplica el algoritmo descrito en [28] para obtener los circuitos

Tabla 3-5.: Parámetros de consumo de energía en Watts [6].

Elemento de red	E0 [Watt]	M [Watt]
Nodos	$0{,}85C^{2/3}$	$C^{2/3}$ [56]
Enlaces (0-100]Mbps	0,48	0,48
Enlaces (100-600]Mbps	0,90	1,00
Enlaces (600-1000]Mbps	1,70	2

independientes[1] que hay en ella. Como un circuito es la suma de varios enlaces, el recorrido total se hace mucho más largo y la medida se vuelve más confiable, disminuyendo el número total de paquetes que se envían al controlador. Cuando se tienen las medidas de los retrasos de todos los circuitos, se crea un sistema lineal de ecuaciones que se resuelve para obtener el retraso de cada uno de los enlaces.

Las técnicas presentadas en [3] son las que se usaron en la implementación de nuestro esquema de enrutamiento dinámico para determinar los retrasos en los enlaces de la red.

3.2.3. Modelos de consumo energético

En cuanto al consumo energético de los elementos de red, es difícil encontrar datos, pero en [6] se presenta la Tabla **3-5**, que resume los resultados presentados en otros trabajos sobre el tema. En esta tabla, los valores C son las capacidades de los nodos para procesar flujos, asumiendo que el nodo es capaz de procesar el doble del total de la suma de las capacidades de los enlaces conectados a él. Los resultados mostrados en esta tabla, permiten comprobar que, de los modelos presentados en la Figura **3-6**, el que mejor describe el consumo energético de los dispositivos reales es el modelo de sueño.

3.2.4. Calidad de servicio

A continuación se presenta un resumen de trabajos encontrados que buscan ofrecer calidad de servicio en SDNs.

PolicyCop se presenta en [4] como una plataforma para el manejo de políticas de QoS para SDNs que ofrece una interfaz para especificar los parámetros de QoS de los acuerdos de nivel de servicio (Service Level Agreement, SLA), y los garantiza usando la API de *OpenFlow*, monitoreando y ajustando los parámetros de la red, para cumplir con los SLAs. Se presentan

[1] Se denomina circuito a un camino que inicia y termina en el mismo nodo, sin visitar ningún nodo intermedio más de una vez.

algunos resultados que muestran la efectividad de PolicyCop para garantizar los niveles de rendimiento, retardo y disponibilidad.

Network Control Layer (NCL) es una plataforma basada en SDN, *OpenFlow* y el paradigma Network as a Service (NaaS) que plantea una innovación en el manejo y control al momento de ofrecer servicios de red punto a punto bajo demanda garantizando QoS, basado en especificaciones dadas por aplicaciones interactivas de alto nivel que se ejecutan en el controlador [7]. NCL se implementó usando OpenNaaS[2] e incluye mecanismos para el monitoreo del estado de la red y para la configuración de los dispositivos según las necesidades de QoS de las aplicaciones interactivas; las pruebas iniciales se realizaron con el simulador Mini-Net, pero se espera implementar en un banco de pruebas real, dentro del proyecto OFELIA[3].

QNOX es un sistema operativo de red, basado en NOX [20], que pretende dar solución a las falencias que tiene este en cuanto a garantías de QoS para grandes proveedores de Internet, como incrustamiento de redes virtuales con QoS, la valoración de QoS punto a punto en redes, y colaboración entre los elementos de control. En [25] se explica en detalle cada uno de los componentes de QNOX y se presenta el estado actual de la implementación del mismo, que deja ver, según resultados de experimentos, su usabilidad en proveedores de Internet de gran escala.

En [12] se presenta una arquitectura para QoS en la transmisión de multimedia en SDNs basada en OpenFlow, que hace optimización dinámica de las rutas de los flujos de tráfico multimedia. El modelo planteado se evaluó mediante simulación y se obtuvieron resultados que mejoran significativamente comparados con el enrutamiento tradicional de Internet. También se extiende a redes *OpenFlow* multi dominio y se propone una arquitectura de control distribuido y métodos para el enrutamiento dinámico interdominio con QoS. Se implementó un software controlador en una red real, corroborando los resultados obtenidos mediante simulación y mostrando que se logra garantizar muy poca o ninguna alteración en la experiencia del usuario final.

En [14] se presenta el controlador OpenQoS, que se enfoca en QoS para la transmisión de multimedia en SDNs, y que ofrece 3 interfaces: una para comunicarse con los dispositivos de la red, otra para la comunicación entre controladores y la tercera que le permite a los proveedores de servicios definir nuevos flujos e intercambiar información cuando se establece un flujo de datos para una nueva aplicación; aunque solo se implementó la primera de ellas, dejando las 2 últimas como trabajos futuros. La implementación se hizo en un controlador Floodlight[4] y se lograron resultados que garantizan totalmente la calidad del video con TCP,

[2]http://opennaas.org/.
[3]http://www.fp7-ofelia.eu/.
[4]http://www.projectfloodlight.org/floodlight/.

mientras que con UDP se tienen poca o ninguna alteración en la calidad del video. Por otro lado, en [13] se presenta una plataforma para la optimización del enrutamiento en la capa de control, para permitir QoS dinámico en redes *OpenFlow*, y se plantean aplicaciones de este en la transmisión de video codificado con dos niveles de QoS. En este trabajo se presenta la formulación del problema y el algoritmo usado para resolverlo, y se muestran resultados experimentales que dejan ver la mejora en la transmisión de video bajo diferentes codificaciones y escenarios de congestión de la red.

Es de notar que ninguno de estos trabajos tiene contemplado el tema de consumo energético dentro de sus objetivos, y que los últimos tres trabajos mencionados fueron desarrollados y presentados por el mismo grupo de investigadores, lo que deja ver claramente que que aún este es un tema emergente.

3.2.5. Ahorro energético

Los trabajos que se enuncian a continuación, tienen como objetivo la optimización del consumo energético de la red, pero sin tener en cuenta las restricciones de calidad de servicio.

ElasticTree [23] es un manejador de energía, cuyo comportamiento se analizó en un banco de pruebas con conmutadores *OpenFlow* de 3 fabricantes diferentes, lográndose hasta 50 % de ahorro de energía para cargas de centros de datos, sin afectar la capacidad de estos de manejar el tráfico que surja. ElasticTree no hace el cálculo de las matrices de tráfico; asume que están disponibles de antemano.

En [31] se muestra que se puede mejorar el rendimiento en las redes actuales mediante la incorporación gradual de SDN en ellas; se indica mediante medición y simulación en NS-2 que el rendimiento global de la red se aumenta, disminuyendo los retrasos y la pérdida de desempeño, pero se asume también que se tiene toda la información de la red de antemano.

En [27] se describen algunos ejemplos de cómo aplicar SDN y programación lineal para lograr algunas características como enrutamiento por camino de menor costo o por balanceo de cargas; también se describe detalladamente el caso de una SDN que realiza enrutamiento por consumo energético, y explican paso a paso el diseño de la aplicación en C/C++ para un controlador NOX [20]. Al final del trabajo se plantean algunas mejoras que se podrían realizar para optimizar el desempeño y otras líneas de trabajo que se pueden surgir a partir de lo que se describe en [27].

GreenMST [47] es una aplicación que utiliza el protocolo LLDP (*Link Layer Discovery Protocol*) para crear un grafo que representa la topología de la red y calcular su árbol de mínima

expansión (*Minimum Spanning Tree, MST*) para lograr una topología libre de ciclos, desactivando los enlaces que no pertenezcan a él con el fin de lograr así un ahorro de energía en la red. Otra ventaja de GreenMST es su corto tiempo de convergencia respecto de otras soluciones como por ejemplo STP *Spanning Tree Protocol*. Sin embargo, GreenMST no tiene en cuenta ninguna métrica de calidad de servicio, solo se busca ahorro energético.

De manera similar, MdST *(Minimum delay Spanning Tree)* [38] utiliza LLDP para obtener el grafo que representa la topología de la red, pero también hace un monitoreo constante de los retrasos en los enlaces de la red, usando la técnica descrita en [3], para utilizar esos valores como los pesos de los enlaces del grafo y calcular es MST respecto de los retrasos en la red. Una vez se tiene el MST, se calculan las rutas más cortas entre cada par de dispositivos de borde, y esas son las rutas que se instalan en los dispositivos del plano de datos. MdST permite desactivar tanto los enlaces como los nodos que no pertenecen a ninguna de las rutas establecidas, lo que representa un ahorro energético superior al obtenido con GreenMST, y con rutas que tienen mínimo retraso en todo momento, según el retraso medido en cada instante en los enlaces de la red. Los tiempos de convergencia de MdST mostraron ser muy similares a los de GreenMST.

Estas dos últimas propuestas son las que se utilizaron para contrastar los resultados obtenidos con nuestro esquema de enrutamiento.

4. Planteamiento de la solución

El funcionamiento general de nuestro esquema de enrutamiento se describe en el diagrama de flujo de la Figura **4-1**. A medida que los dispositivos de red se conectan al controlador, se van agregando al grafo que representa la red y se ejecuta el procedimiento que se describe en la Sección 4.1 para determinar los dispositivos de borde de la red. También se realiza el procedimiento descrito en la Sección 4.3 para iniciar el cálculo del retraso de los enlaces de la red.

Periódicamente se realiza el procedimiento que se describe en la Sección 4.3 para determinar los valores de los parámetros de *QoS* de cada uno de los enlaces de la red y el cálculo de matriz de tráfico de la red, según el procedimiento descrito en la Sección 4.2. Los valores obtenidos con estos procedimientos se utilizan como parámetro del problema de optimización descrito en la Sección 4.4, que se resuelve para determinar las mejores rutas según las condiciones actuales de la red y se establecen las reglas correspondientes en los dispositivos del plano de datos. Si no se logra encontrar una solución factible para el problema, no se modifican las reglas en el plano de datos, dejando las que se instalaron la última vez que se obtuvo una solución factible.

Para la implementación y evaluación de la propuesta y ante la falta de disponibilidad de dispositivos físicos que soportaran *OpenFlow*, se decidió utilizar *Mininet* [32] para emular las topologías elegidas para las pruebas. Por otro lado, dado que para el cálculo de la matriz de tráfico de la red se necesita que los dispositivos del plano de datos implementen los medidores, que son una característica opcional dentro de la especificación de la versión 1.3 del protocolo *OpenFlow*, se hizo una revisión de los conmutadores disponibles que implementaban esta característica y se encontraron los siguientes:

- Open Virtual Switch [45]. No está implementada para la versión software que es la que se utiliza para emular las redes en Mininet.
- Linc [1]. Implementa los medidores, pero el código es desarrollado en lenguaje Erlang, lo que es una debilidad puesto que ante cualquier dificultad que se pueda presentar, nos limita el desconocimiento del lenguaje de programación para realizar algún ajuste necesario.
- La implementación de *CPqD* [2]. El desarrollo de este conmutador se describe en detalle

[1] https://github.com/FlowForwarding/LINC-Switch.
[2] http://cpqd.github.io/ofsoftswitch13/.

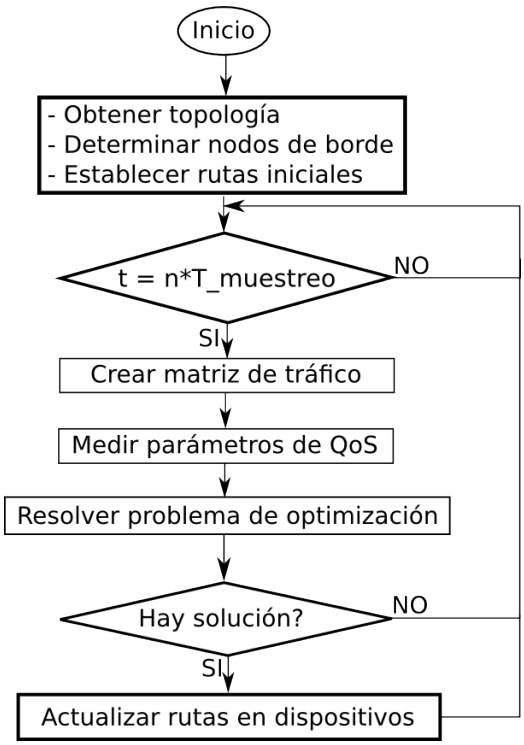


Figura 4-1.: Diagrama de flujo de la estrategia de solución.

en [16] y fue implementado en lenguaje C++, lo cual era una ventaja, ya que ante cualquier dificultad que se presentara se podría modificar el código fuente para ajustar a nuestra necesidad. Además este fué el conmutador utilizado en [29] para probar su técnica para calcular la matriz de tráfico. Este fué el conmutador elegido para el desarrollo.

En cuanto a la elección del controlador a utilizar, como en la aplicación era necesario utilizar características propias de la versión 1.3 del protocolo *OpenFlow*, se eligió el controlador *Ryu* [3] que era el único soportaba *OpenFlow 1.3* al momento de iniciar el desarrollo y, además, era el controlador con el que se desarrolló la prueba de concepto en [29].

Finalmente, para resolver el modelo de optimización se requería un software especializado, pero debido a que tanto para desarrollar la aplicación del controlador como para describir las topologías en *Mininet* era necesario usar el lenguaje *Python*, se buscó una herramienta que ofreciera una *API* para programar en este mismo lenguaje. La herramienta que se eligió fué *Gurobi* [42], en su versión 6.5.0, por su facilidad de uso y la buena documentación disponible, además de que ofrece licencias académicas gratuitas por espacio de un año, y fácilmente renovables.

[3] http://osrg.github.com/ryu.

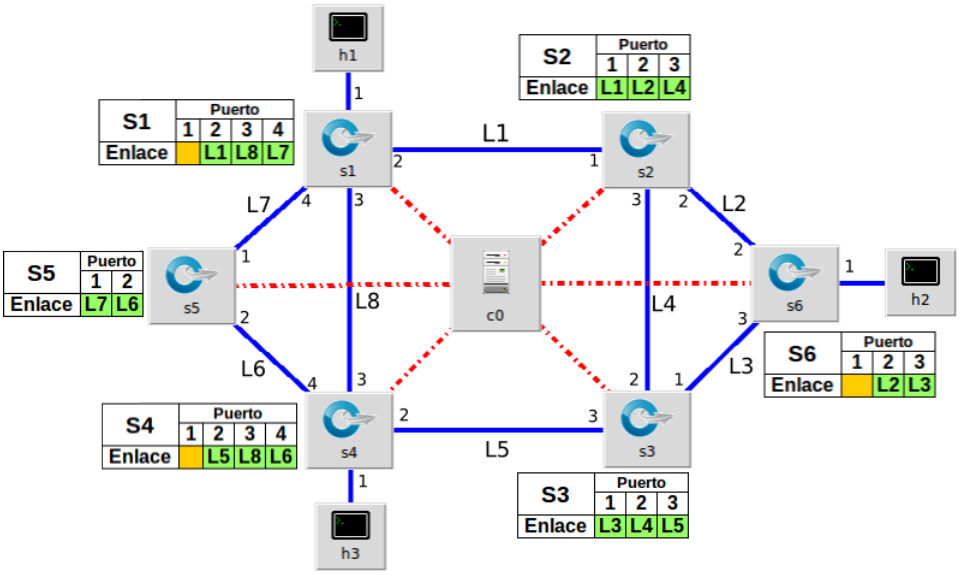


Figura 4-2.: Ejemplo determinación de dispositivos de borde.

En lo que resta de este capítulo se describe cómo se realizó cada una de las etapas de la solución propuesta.

4.1. Determinación de dispositivos de borde

El controlador *Ryu* tiene una implementación del protocolo LLDP, que permite determinar cómo están conformados los enlaces de la red, así como qué puertos activos tiene cada dispositivo, con lo que se hace un conteo de los puertos de cada dispositivo que están presentes en las descripciones de los enlaces para determinar qué clase de nodos son. Aquellos dispositivos que tengan todos sus puertos descritos en los enlaces, serán dispositivos de tránsito, mientras que los que tenga al menos uno de sus puertos sin ser descritos en los enlaces, serán los dispositivos de borde de la red, que conectarán a dispositivos finales, o bien con otras redes fuera del alcance del controlador, por medio de esos puertos no descritos. En la Figura **4-2** se presenta un ejemplo de este procedimiento, en donde se observan tablas que describen cómo se usan los puertos de cada dispositivo, y se destacan en verde los puertos que pertenecen a enlaces con otros dispositivos de la red, y en amarillos los que no. Puede observarse que s1 y s3 son los dispositivos de borde de esta topología, mientras que s2 y s4 son de tránsito.

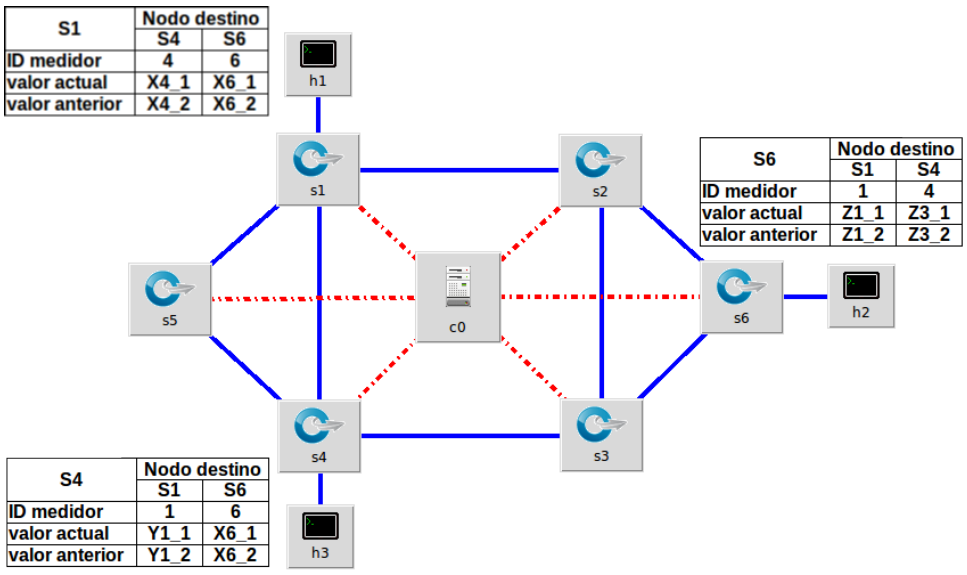


Figura 4-3.: Ejemplo determinación de matriz de tráfico.

4.2. Obtención de la matriz de tráfico de la red

Una vez determinados los dispositivos de borde, en cada uno de ellos se crea un *meter_id* para identificar los flujos que van a cada uno de los otros dispositivos de borde de la red; este *meter_id* corresponde al identificador único del conmutador de destino en cuestión. Cada que llega un paquete que no coincide con ninguna de las reglas de flujos ya establecidos en el conmutador, se verifica cuál es el identificador del conmutador de destino, y si el controlador ha calculado una ruta válida para llegar a él, en cuyo caso se fijan las reglas necesarias en los dispositivos correspondientes para establecer la ruta, especificando en la regla que se instala en el conmutador de origen, que los paquetes que coincidan con esa regla se asocien con el medidor correspondiente, según el *meter_id*. Con esto se garantiza que todos los flujos que vayan de un conmutador de origen dado hacia un mismo conmutador de destino, se contabilizan en el mismo medidor en el conmutador de origen, lo que corresponde a una celda de la matriz de tráfico. Este es el procedimiento que se utiliza en [52, 29] para determinar la matriz de tráfico, con la diferencia de que en esas propuestas se fijan los *meter_id* de manera manual.

El controlador consulta periódicamente el valor de los medidores de todos los dispositivos de borde, y almacena estos valores en unas tablas como las que se observan en la Figura **4-3**, donde se mantienen los valores actuales y anteriores de los medidores. La matriz de tráfico se calcula tal como se indica en la Figura **4-1**, donde T es el periodo de tiempo entre cada par de medidas.

Tabla 4-1.: Matriz de tráfico.

	s1	s4	s6
s1	0	(X4_1 - X4_2)/T	(X6_1 - X6_2)/T
s4	(Y1_1 - Y1_2)/T	0	(Y6_1 - Y6_2)/T
s6	(Z1_1 - Z1_2)/T	(Z4_1 - Z4_2)/T	0

4.3. Monitorizacion de parámetros de QoS

En esta etapa se miden los parámetros de calidad de servicio de la red, tales como retraso, porcentaje de pérdidas, o ancho de banda disponible en los enlaces, cuyos valores se usan como parámetros del problema de optimización que determina las mejores rutas disponibles para encaminar los flujos presentes en la red. Para probar el esquema diseñado solo se midió el retraso de los enlaces, para lo que, nuevamente, se utilizó la implementación del protocolo *LLDP* del controlador *Ryu* para crear una descripción de la red como un grafo dirigido y aplicar una implementación reducida de la estrategia presentada en [3] que se describe a continuación.

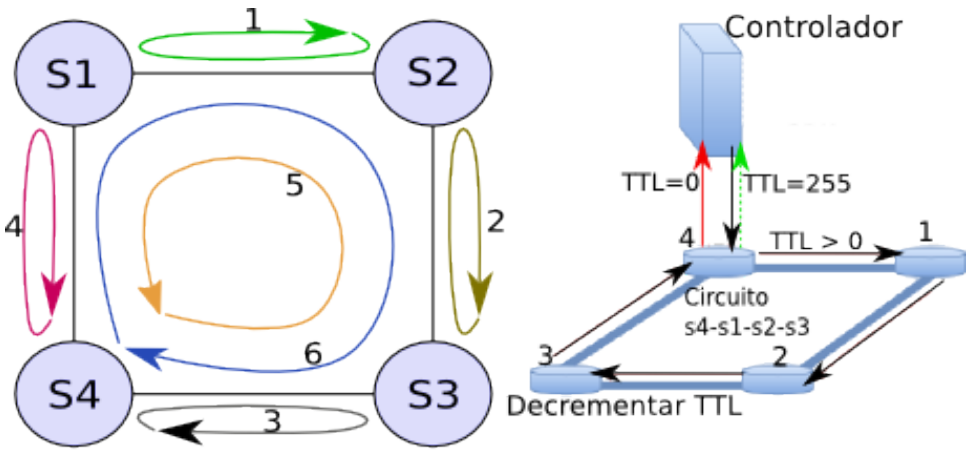
Cada que se detecta un cambio en la topología de la red, se actualiza el grafo que la representa, y se utiliza el algoritmo presentado en [28] para calcular los circuitos independientes de éste. Esto se hace usando el paquete python Networkx [21], que es un paquete especial para manejo de grafos.

Una vez se conocen los circuitos independientes del grafo, se lleva a cabo la siguiente secuencia de pasos para establecer esos circuitos en los dispositivos de la red, como paso inicial para determinar los retrasos de los enlaces:

1. Enumerar secuencialmente los circuitos.
2. Generar MAC que identifica cada circuito, a partir de su número de secuencia y los identificadores del primer y último dispositivo.
3. Instalar reglas en los dispositivos del circuito, para que los paquetes que tengan la MAC correspondiente como destino se redireccionen al siguiente dispositivo. En el último de ellos se agrega una acción más para que también se decremente el campo TTL (*Time-To-Live*) del paquete IP de prueba.

Luego de instalar las rutas de los circuitos en los dispositivos de red, el controlador inicia el procedimiento que se enuncia a continuación para lograr determinar los retrasos de lo enlaces de la red:

1. Envía al dispositivo inicial del circuito un mensaje *PACKET_OUT*, que contiene un paquete IP de prueba, con campo $TTL = 255$ y dirección MAC destino correspondiente



Circuito			Enlaces							
#	Inicio	Final	s1-s2	s2-s3	s3-s4	s4-s1	s2-s1	s3-s2	s4-s3	s1-s4
1	2	1	X				X			
2	3	2		X				X		
3	4	3			X				X	
4	4	1				X				X
5	4	1					X	X	X	X
6	4	3	X	X	X	X				

Figura 4-4.: Ejemplo de circuitos para medir retrasos.

para cada circuito. Las acciones de este mensaje le indican al dispositivo que envíe una copia de regreso al controlador, y que envíe el paquete hacia el puerto correspondiente para que llegue al segundo dispositivo en el circuito. En este momento el paquete comienza a circular por el circuito, decrementando su campo TTL cada que pasa por el último dispositivo, hasta que llega de nuevo al dispositivo inicial con $TTL = 0$, que es un valor inválido y se envía de nuevo al controlador, tal como se muestra en la parte superior derecha de la Figura **4-4** para el circuito S4-S1-S2-S3.

2. Inicia el conteo del tiempo de recorrido del paquete de prueba para el circuito correspondiente cuando recibe un mensaje *PACKET_IN* desde el dispositivo inicial del circuito con el paquete tal como lo acaba de enviar y puerto de entrada *CONTROLLER*.

3. Termina el conteo del tiempo de recorrido cuando recibe el paquete dentro de un mensaje *PACKET_IN* con campo $reason = 2$ que indica que el paquete tiene un valor inválido en el campo TTL ($TTL = 0$). En este momento el paquete habrá recorrido 255 veces el circuito.

Cuando se tienen las medidas de todos los circuitos del grafo se crea un sistema de ecuaciones lineales, tal como se describe en [3], donde las variables independientes corresponden a los retrasos de cada uno de los enlaces dirigidos de la red y cada ecuación representa un circuito.

La ecuación (4-1) es el sistema de ecuaciones lineales correspondiente para la topología que se ilustra en la Figura **4-4**, que tiene seis circuitos y ocho enlaces dirigidos, por lo que el sistema de ecuaciones no tiene solución.

$$\begin{pmatrix} 255 & 0 & 0 & 0 & 255 & 0 & 0 & 0 \\ 0 & 255 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 & 0 & 255 & 0 \\ 0 & 0 & 0 & 255 & 0 & 0 & 0 & 255 \\ 0 & 0 & 0 & 0 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} \tag{4-1}$$

En estos casos se expande el número de ecuaciones, agregando una ecuación por cada circuito conformado correspondiente a un enlace no dirigido (circuitos 1, 2, 3 y 4 en la Figura **4-4**). El retraso de los enlaces que conforman estos circuitos se calcula a partir del RTT, tal como se describe en la primera parte de [3] y en la Tabla **3-4**. De este modo se obtiene una matriz como la que se observa en la ecuación (4-2), donde se asume que el retraso de ambos enlaces del circuito es $RTT/2$. Este sistema tiene más ecuaciones que variables, por lo que no tiene solución única.

$$\begin{pmatrix} 255 & 0 & 0 & 0 & 255 & & 0 & 0 \\ 0 & 255 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 & 0 & 255 & 0 \\ 0 & 0 & 0 & 255 & 0 & 0 & 0 & 255 \\ 0 & 0 & 0 & 0 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 & & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 255 & 0 \\ 0 & 0 & 0 & 255 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_1/2 \\ x_2/2 \\ x_3/2 \\ x_4/2 \end{pmatrix} \tag{4-2}$$

Para lograr una matriz cuadrada se realiza el procedimiento que se enuncia a continuación.

1. Se calcula el rango de la matriz actual.

2. Se elimina al azar una de las ecuaciones y se calcula el rango de la matriz resultante. Si el rango es el mismo la ecuación se elimina definitivamente, de lo contrario se reintegra a la matriz, pues esa sería una de las ecuaciones independientes.

3. Se repite el paso anterior hasta lograr el resultado esperado, que es una matriz como la que se muestra en la ecuación (4-3).

$$\begin{pmatrix} 255 & 0 & 0 & 0 & 255 & 0 & 0 & 0 \\ 0 & 255 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 & 0 & 255 & 0 \\ 0 & 0 & 0 & 255 & 0 & 0 & 0 & 255 \\ 0 & 0 & 0 & 0 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 255 & 0 \end{pmatrix} * \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_1/2 \\ x_3/2 \end{pmatrix} \tag{4-3}$$

Cuando se tiene la matriz cuadrada se resuelve el sistema de ecuaciones resultante y se obtienen los retrasos de cada uno de los enlaces.

Esta implementación de la estrategia descrita en [3] permite obtener las medidas con buen nivel de precisión, pero instalando menor cantidad de reglas en los dispositivos de la red, pues solo se instala una regla por cada dispositivo del circuito correspondiente, lo que es deseable, puesto que la cantidad de entradas de flujo es un recurso limitado que se debe utilizar forma eficiente [17]. En [3] se indica que el número de reglas que debe instalarse para cada circuito es $(C * [M^{1/C}] + N + 1) + k$, donde C es el número de campos del encabezado del paquete que se modifican, M representa el número de veces que se desea que el paquete de prueba recorra el circuito antes de volver al controlador, N es número de nodos de la topología y k el número de nodos del circuito. Para 2048 vueltas se utilizan 96 +k reglas por cada circuito, modificando 2 campos del encabezado del paquete. En la Figura **4-4**, modificando también dos campos, para 255 vueltas se requerirían 37 + k reglas (modificando solo un campo serían 260 +k), mientras que con nuestra implementación son solo k+1, lo que son 216 reglas menos.

Otra diferencia con [3] es que nuestra implementación permite enviar los paquetes de prueba periódicamente, cada varios segundos o minutos, o incluso solo cuando se detecte un cambio en la red, dependiendo de las necesidades específicas de la aplicación desarrollada, lo que disminuye el tráfico total inyectado a la red y la carga del controlador para procesarlos.

4.4. Formulación del problema de optimización

Se modela la red como un grafo dirigido $G = (N, L)$, donde N es el conjunto de los nodos y L el de los enlaces de la red, y se formula un problema de optimización como un modelo entero mixto que determina un único camino para enviar cada elemento del conjunto de demandas origen-destino, D, de la red, con el fin de evitar que los paquetes de una misma demanda lleguen en desorden a su destino final, lo que podría ocurrir si el modelo planteado fuera

multicamino, ya que los diferentes caminos establecidos pueden tener retrasos distintos. El objetivo del modelo es minimizar el consumo energético de la red después de asignar las demandas. A continuación se explica paso a paso la formulación del problema de optimización.

Índices utilizados

$d = 1, 2, ...\|D\|$	demandas.
$l = 1, 2, ...\|L\|$	enlaces.
$n = 1, 2, ...\|N\|$	nodos.

Variables

X_n	= 1 si el nodo n está activo en la solución del problema, cero de lo contrario.
Y_l	= 1 si el enlace l está activo en la solución del problema, cero de lo contrario.
U_{dl}	= 1 si la demanda d se encamina por el enlace l, cero de lo contrario.
f_{dl}	cantidad de flujo de la demanda d que se envía por el enlace l.
f_l	cantidad total de flujo que se encamina por el enlace l.
q_n	cantidad total de flujo que procesa el nodo n.
r_l	retraso del enlace l.

Parámetros

a_{ln}	= 1 si n es el origen del enlace l, cero de lo contrario.
b_{ln}	= 1 si n es el destino del enlace l, cero de lo contrario.
s_d	= nodo origen de la demanda d.
t_d	= nodo destino de la demanda d.
h_d	= volúmen de la demanda d.
c_l	= capacidad del enlace l.
K_n	= capacidad de procesamiento del nodo n.
E_{0n}	= mínimo consumo energético del nodo n.
Q_n	= consumo energético del nodo n por cada unidad de tráfico procesado.
E_{0l}	= mínimo consumo energético del enlace l.
J_l	= consumo energético del enlace l por cada unidad de flujo que lleva.
R_{max_d}	= retraso máximo permitido para la demanda d.

La ecuación (4-4) define el valor de K_n como la suma de las capacidades de todos los enlaces conectados al nodo n, tanto los que entran como los que salen [6], teniendo en cuenta que al ser un grafo dirigido, la capacidad de un enlace entre dos nodos no es necesariamente igual en ambos sentidos.

$$K_n = \sum_{l \in L} (a_{ln} + b_{ln}) * c_l, \forall n \in N \tag{4-4}$$

Las siguientes ecuaciones definen los valores de los parámetros de consumo energético de los elementos de red, según la Tabla **3-5**:

$$E_{0n} = 0{,}85 * K_n^{(2/3)} \tag{4-5}$$

$$Q_n = 0{,}15 * \frac{K_n^{(2/3)}}{K_n}, \forall n \in N \tag{4-6}$$

$$J_l = \frac{M_l - E_{0l}}{c_l}, \forall l \in L \tag{4-7}$$

Restricciones

Las ecuaciones (4-8) a (4-15) representan las restricciones a tener en cuenta para resolver el modelo de optimización.

La ecuación (4-8) expresa que el flujo total que pasa por un enlace es igual a la suma de las cantidades de todas las demandas que se envían por él, que esa cantidad no puede superar a la capacidad del enlace, y debe ser cero si el enlace no se utiliza en la solución.

$$f_l = \sum_{d \in D} f_{dl} \leq Y_l * c_l, \forall l \in L \tag{4-8}$$

La ecuación (4-9) especifica que la cantidad de flujo de una demanda que se envía por un enlace dado, f_{dl}, debe ser el total de la demanda, esto es, que la demanda debe enviarse por un solo camino, sin dividirse en varios flujos.

$$f_{dl} = U_{dl} * h_d, \forall l \in L, \forall d \in D \tag{4-9}$$

La ecuación (4-10) define la cantidad de flujo que procesa el nodo, como la suma de todos los flujos que entran o salen de él, o sea, los flujos que circulan por los enlaces que inician o terminan en ese nodo.

$$q_n = \sum_{l \in L} (a_{ln} + b_{ln}) * f_l, \forall n \in N \tag{4-10}$$

La ecuación (4-11) indica que el nodo se debe utilizar si alguno de los enlaces que inician o terminan en él se usa en la solución.

$$X_n \geq (a_{ln} + b_{ln}) * Y_l, \forall n \in N, \forall l \in L \tag{4-11}$$

Por otro lado, con la ecuación (4-12) se obliga a que el nodo n se desactive cuando no se use ninguno de los enlaces que inician o terminan en él.

$$X_n \leq \sum_{l \in L} (a_{ln} + b_{ln}) * Y_l, \forall n \in N \tag{4-12}$$

De manera similar, la ecuación (4-13) indica que el enlace l se debe desactivar cuando no lleve ninguna demanda.

$$Y_l \leq \sum_{d \in D} U_{dl}, \forall l \in L \tag{4-13}$$

La ecuación (4-14) limita el retraso de la ruta calculada para llevar la demanda hasta el máximo permitido.

$$\sum_{l \in L} U_{dl} * r_l \leq R_{max_d}, \forall d \in D \tag{4-14}$$

Por último, la ecuación (4-15) expresa las restricciones de conservación de flujo en cada uno de los nodos del grafo. Esta ecuación indica que, para cada una de las demandas, la diferencia entre la cantidad de flujo que sale de un nodo y la cantidad de flujo que entra a él, es igual a la cantidad de demanda para el nodo origen, a cero para los nodos intermedios, e igual a la cantidad total de la demanda, pero negativa, para el nodo destino, puesto que en este nodo es mayor la cantidad de flujo que entra que la que sale.

$$\sum_{l \in L} a_{ln} * f_{dl} - \sum_{l \in L} b_{ln} * f_{dl} = \begin{cases} h_d, & \text{si } n = s_d \\ 0, & \text{en caso contrario} \\ -h_d, & \text{si } n = t_d \end{cases}, \forall n \in N, \forall d \in D \tag{4-15}$$

Función objetivo

Finalmente, la ecuación (4-16) expresa el objetivo final de nuestra formulación, que es el de encontrar la configuración de la red que minimice el consumo energético total de la red, expresado como la suma del consumo de los nodos y enlaces usados, usando los datos mostrados en la Tabla **3-5**. Es importante aclarar que este modelo y el esquema de enrutamiento completo son general y se puede buscar cualquier otro objetivo simplemente cambiando la ecuación (4-16) por la que convenga (e.g. minimización de retraso, balanceo de carga, etc.). Como se explicó anteriormente, según los estudios encontrados sobre el consumo energético

de los dispositivos de red, el modelo que mejor los describe es el modelo de sueño, razón por la cual es ese el modelo que se eligió para evaluar el esquema de enrutamiento presentado.

$$Minimizar \sum_{n \in N} (X_n * E_{0n} + q_n * Q_n) + \sum_{l \in L} (Y_l * E_{0l} + f_l * J_l) \tag{4-16}$$

4.5. Caso de estudio

Como parte del proyecto *"Plataforma tecnológica para los servicios de Teleasistencia, Emergencias médicas, seguimiento y monitoreo permanente a los pacientes y apoyo a los programas de promoción y prevención"*, el grupo de excelencia *Artica* desarrolló un sistema que consiste de un dispositivo que se conecta al paciente para monitorizar sus signos vitales y una unidad central de procesamiento que recibe analiza los valores obtenidos en busca de posibles eventos anormales en ellos, en cuyo caso se envía una notificación al círculo de acompañamiento del paciente y a un sistema remoto perteneciente al sistema de salud, con el fin de que se tomen las acciones correspondientes.

Además de enviar notificaciones de eventos anormales, el sistema envía reportes periódicos del estado del paciente hacia el sistema remoto, para que se pueda tener un registro histórico de la condición del paciente en el sistema de salud. De este modo se busca disminuir el impacto que tienen los pacientes con enfermedades crónicas en el sistema de salud, disminuyendo el número de citas de seguimiento, y ahorrando costos tanto al sistema como al propio paciente, que evita hacer desplazamientos continuos hacia la institución hospitalaria para sus revisiones. En la Figura **4-5** se muestra el escenario de funcionamiento del sistema desarrollado por el grupo de excelencia Artica.

Para la evaluación inicial del esquema de enrutamiento desarrollado se utilizó este escenario de transmisión de señales de telemedicina, debido a la importancia que han ganado este tipo de aplicaciones a nivel mundial, y su importancia en el contexto colombiano. En un sistema real, el esquema diseñado podría operar en la red que comunica al sistema remoto con la unidad central de procesamiento, que puede ser una red intra-hospitalaria, en la que solo circulen flujos de interés para el sistema de salud, o la internet, como se indica en la Figura **4-5**.

En la Tabla **4-2** se recoge un listado de los parámetros de calidad de servicio para las diferentes señales de telemedicina. Como puede observarse en esta tabla, las señales que mayor restricción en cuanto a retraso tienen son las estadísticas del paciente (100 ms), superando incluso a las de audio y video (150 ms), por lo que esta fué la restricción de retraso máximo ($R_{max_d} = 100ms$) elegida para la evaluación inicial del esquema desarrollado.

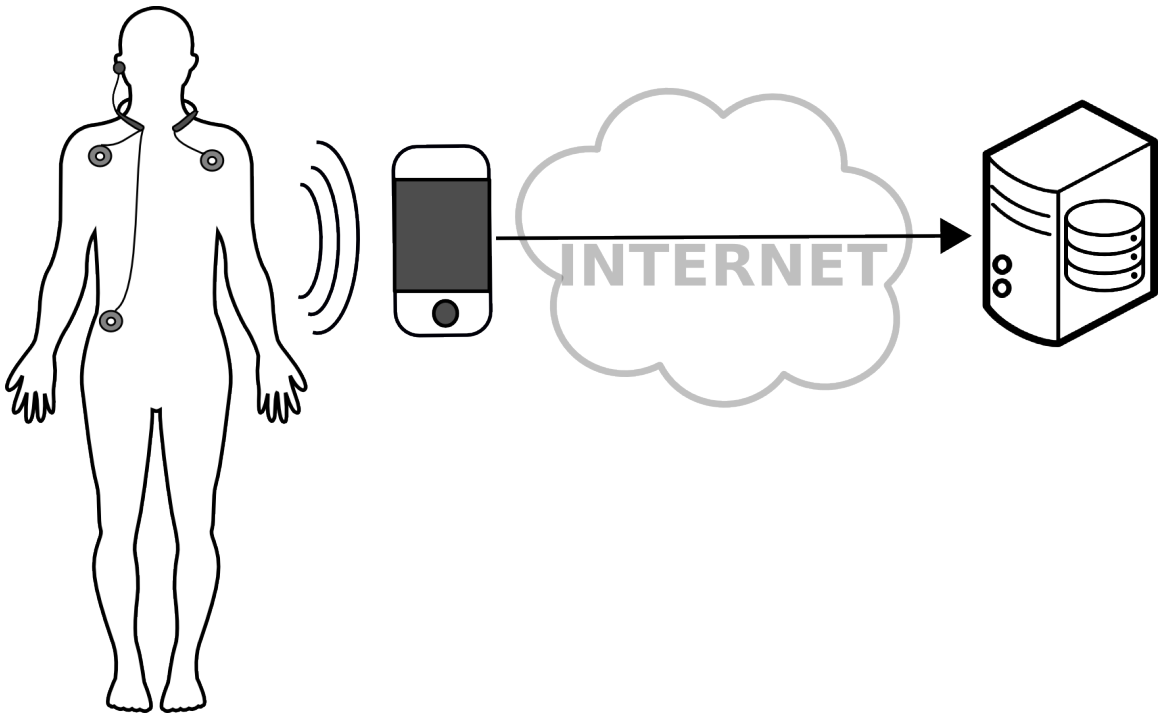


Figura 4-5.: Caso de uso.

Tabla 4-2.: Parámetros de QoS por tipo de dato de telemedicina.

Tipo de señal	**Tasa de transmisión**	**Retardo máximo**
Audio [51]	4-25 kbps	150-400 ms
Video [51]	32-384 kbps	150-400 ms
ECG [51]	1-20 kbps	≈1 s
Estadísticas del paciente	N/A	100 ms
Presión sanguínea [2]	10 kbps	10 s
Respiración [2]	10 kbps	5 s
Termómetro [2]	10 kbps	60 s
Saturación de oxígeno (SpO2) [2]	1 kbps	30 s
Detección de movimiento [2]	50 kbps	5 s
Voz [2]	100 kbps	5 s

5. Evaluación

5.1. Escenario de simulación

Para la evaluación de los resultados, se eligió la topología que se muestra en la Figura **5-1**, que consiste de 14 enlaces y 11 nodos, con solo 3 nodos de borde. Esta topología permite tener diversidad de caminos entre cada par de nodos de borde, que es una de las características esperadas del esquema de enrutamiento planteado, que debe ir cambiando la ruta establecida conforme van variando las condiciones de la red. Esta topología es una representación de la red Abilene, una red de alto rendimiento que sirvió como red troncal para la red Internet2[1], desarrollada por universidades, principalmente, en Estados Unidos. En la Figura **5-1** se observa que todos los enlaces tienen capacidades inferiores a 100Mbps, lo que implica, según los valores de la Tabla **3-5**, las siguientes simplificaciones en el problema de optimización, para este escenario específico:

$$E_{0l} = 0{,}48W$$

$$J_l = 0$$

Topology-zoo [30] es un proyecto que recolecta información de topologías de redes alrededor de todo el mundo, poniendo a disposición del público las representaciones de estas redes en formato GML (*Geography Markup Language*), en los que se describe, además de otras características de la red, las posiciones geográficas de sus conmutadores. Por otro lado, Auto-mininet[2] [19] es un proyecto que permite crear una topología de Mininet a partir de un archivo con formato *GML*, creando un dispositivo final por cada uno de los conmutadores descritos, y donde el retraso de los enlaces se calcula como el tiempo que tardaría en recorrerse la distancia entre los extremos del enlace a la velocidad de la luz. Para realizar nuestros experimentos, y crear la topología, con dispositivos finales conectados solo a algunos nodos de la red, se modificó un poco el código de Auto-Mininet y el archivo GML que representa la topología, agregando al final entradas tipo host, que representan dispositivos finales e indicando a qué conmutador se conecta. En el Anexo A se presenta el archivo GML modificado, y en el Anexo B se presenta el código de Auto-Mininet modificado.

[1] http://www.internet2.edu/.

[2] disponible en: http://141.13.92.69/index.php/projects/auto-mininet.

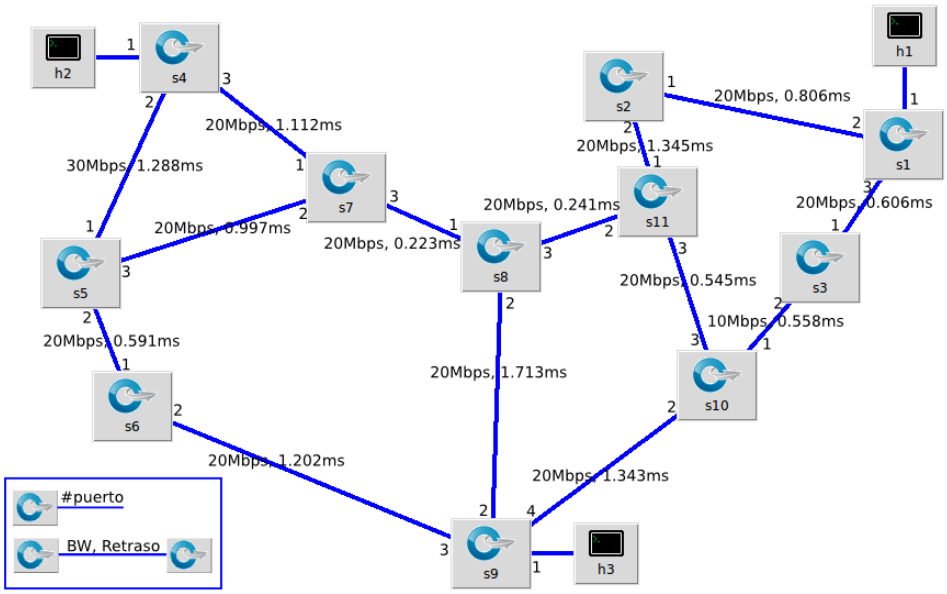


Figura 5-1.: Topología utilizada para el análisis de resultados.

Minievents[3] es un generador de eventos discretos para Mininet, que utiliza una descripción de los eventos en formato JSON (*JavaScript Object Notation*), en el que se especifica el tipo de evento, el tiempo en que se debe lanzar, y otros parámetros, según sea el tipo de evento. Para lograr simular largos periodos de tiempo, y diferentes niveles de carga en la red, es necesario describir muchos eventos, lo que se vuelve complicado de hacer manualmente, por lo que utilizamos *Simpy*[4] [36], que es un simulador de eventos discretos de propósito general, para generar esos archivos JSON. Se definió una función que genera una entrada en el archivo JSON que describe un evento iperf, eligiendo aleatoriamente los nodos origen y destino de la demanda, el tiempo de duración, el instante de tiempo en que se da el evento y la cantidad de demanda, dentro de un rango especificado. Con *Simpy* se lanzan varias instancias de esta función, que se ejecutan en paralelo, generando patrones de tráfico pseudo-aleatorios, según los parámetros que se le pasen a la función definida. El código completo utilizado para generar las demandas utilizadas en la evaluación del esquema se presenta en el Anexo C.

5.2. Resultados y análisis

Para la evaluación del esquema desarrollado, se utilizó un periodo de 30 segundos para calcular la matriz de tráfico y resolver el problema de optimización, midiendo los retrasos de los circuitos cada segundo, para tener una imágen clara de la variación del retraso en todo

[3]https://github.com/mininet/mininet/wiki/Minievents:-A-mininet-Framework-to-define-events-in-mininet-networks.
[4]https://pypi.python.org/pypi/simpy.

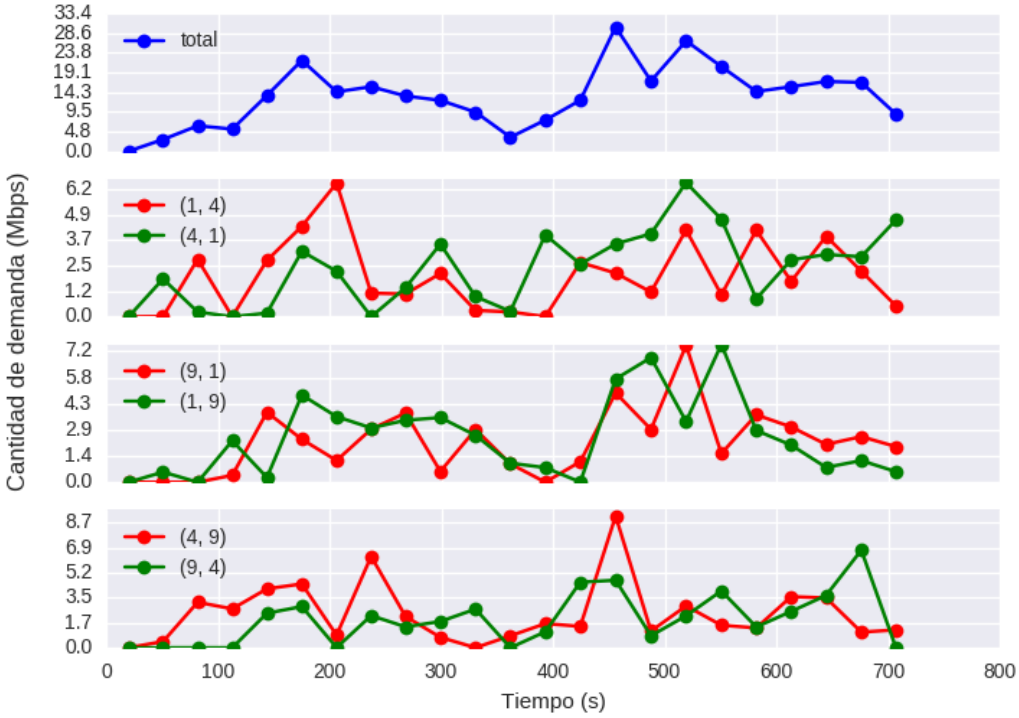


Figura 5-2.: Demandas utilizadas para el escenario de análisis.

momento.

En la figura **5-2** se muestran las demandas entre cada par de dispositivos de borde y la suma total de estas, para un periodo de simulación de 700 segundos, y se marcan los instantes de tiempo en que se realizó el cálculo de la matriz de tráfico (cada 30 segundos). Se puede ver que la red se somete a diferentes niveles de demanda, con el fin de poder analizar el comportamiento de nuestro esquema de enrutamiento ante diferentes niveles de carga en la red. En esta figura se observan varios puntos a tener en cuenta a lo largo del análisis, como los picos en la demanda total que se observan a los 180, 450 y 520 segundos, a los que nombramos punto 1, 2 y 3, respectivamente. En estos puntos se espera que el esquema utilice mayor cantidad de elementos de red y se aumente el consumo energético.

En este momento centramos nuestra atención en la Figura **5-3**, donde se puede ver cómo cambió el número de elementos de red (nodos y enlaces) que se utilizaron para encaminar las demandas, a medida que iba variando el nivel de demanda total en la red. En esta figura se observa que la cantidad de elementos de red permaneció constante en todo el periodo simulado, excepto en los puntos 2 y 3, que son los de mayor demanda total, y donde sí se incrementó el número de nodos y enlaces usados.

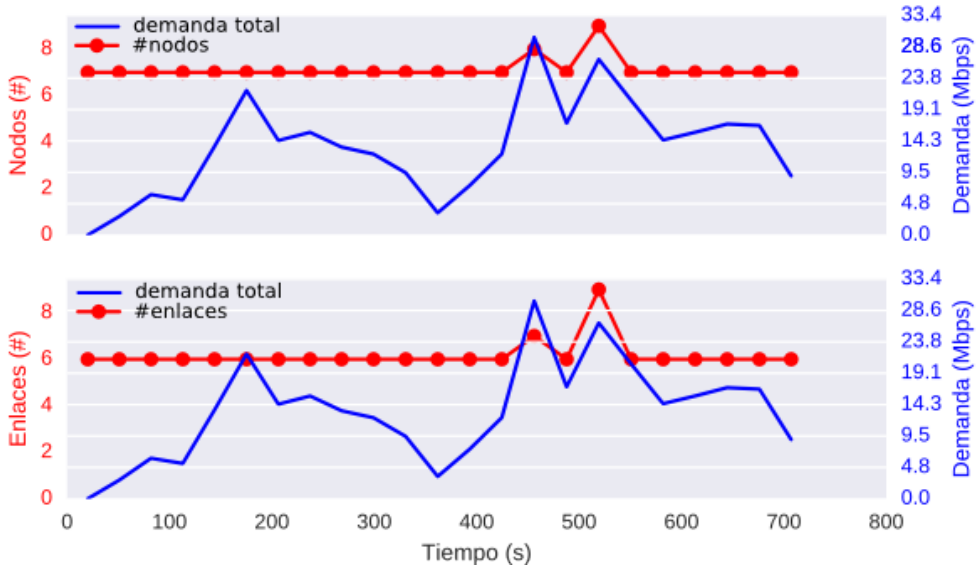


Figura 5-3.: Elementos usados para restricción de 100ms.

Para comprender de mejor manera la forma como cambió el número de elementos utilizados durante la simulación y por qué en el punto 1 no se evidenció el incremento que se presumía, nos apoyamos en las Figuras **5-4** a **5-7**, que muestran las diferentes soluciones obtenidas y la rutas establecidas por el esquema a través de todo el tiempo simulado, y en las que se ve que el número de elementos usados corresponde con los observados en la Figura **5-3**, confirmándose así que el esquema establece diferentes rutas según cambian las condiciones de la red. A estas soluciones las llamaremos solución 1, 2, 3 y 4, respectivamente.

En este punto nos interesa comprender en qué momento se utiliza cada una de estas soluciones y por qué razón. Como todas las soluciones deben conectar a los dispositivos de borde entre sí, estos siempre están presentes en cualquier solución obtenida, entonces la diferencia principal entre ellas va a ser el consumo energético de los nodos de paso que utilicen, y en una menor medida el número de enlaces usados.

La Tabla **5-1** muestra los valores de capacidad de procesamiento y consumo energético de los nodos de paso de la topología, donde se ve que el nodo de menor consumo energético es el nodo 3, seguido de los nodos 2 y 6, por lo que se espera que estos enlaces estén presentes en las soluciones de menor costo, como se comprueba en la Tabla **5-2**, que muestra los nodos de paso que utiliza cada una de las soluciones y su consumo energético mínimo total.

La información mostrada en las Tablas **5-1** y **5-2** indica que la solución 1 debe ser la solución

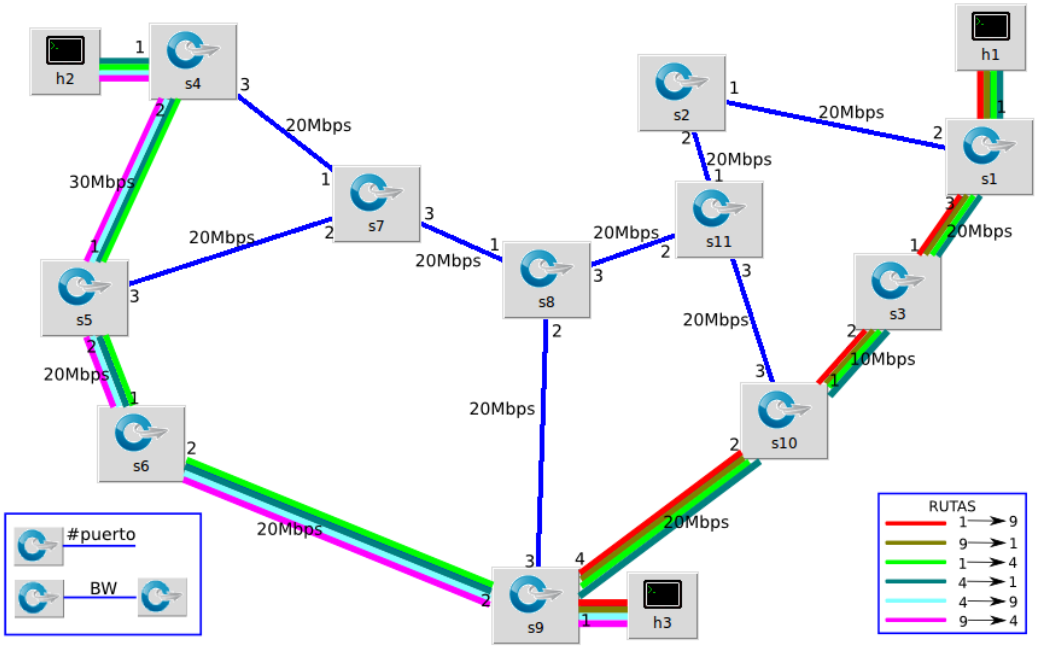


Figura 5-4.: Solución 1.

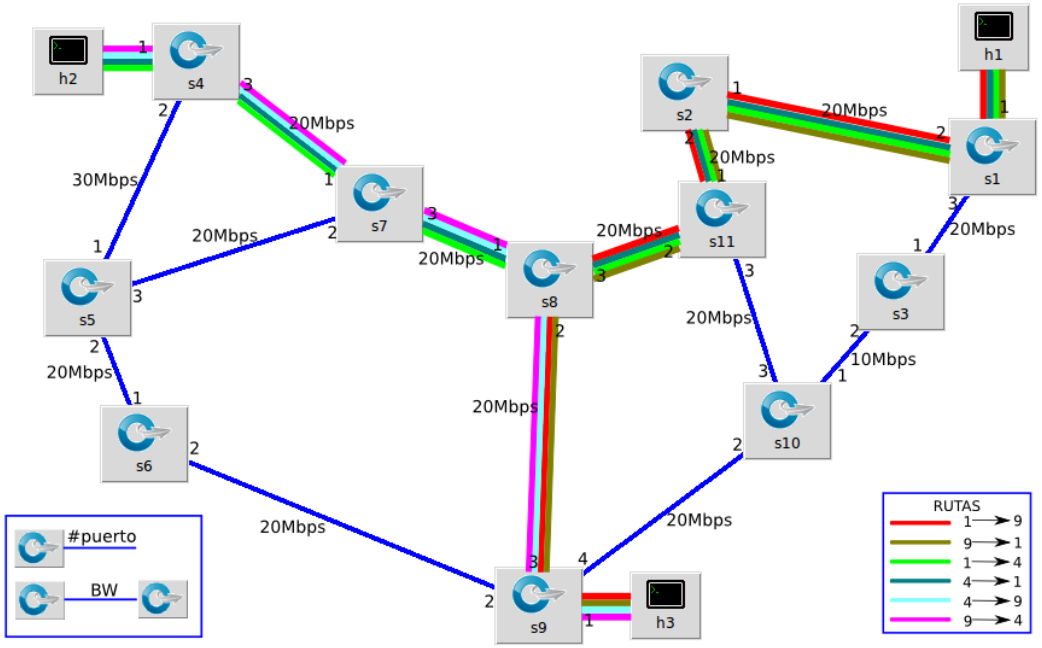


Figura 5-5.: Solución 2.

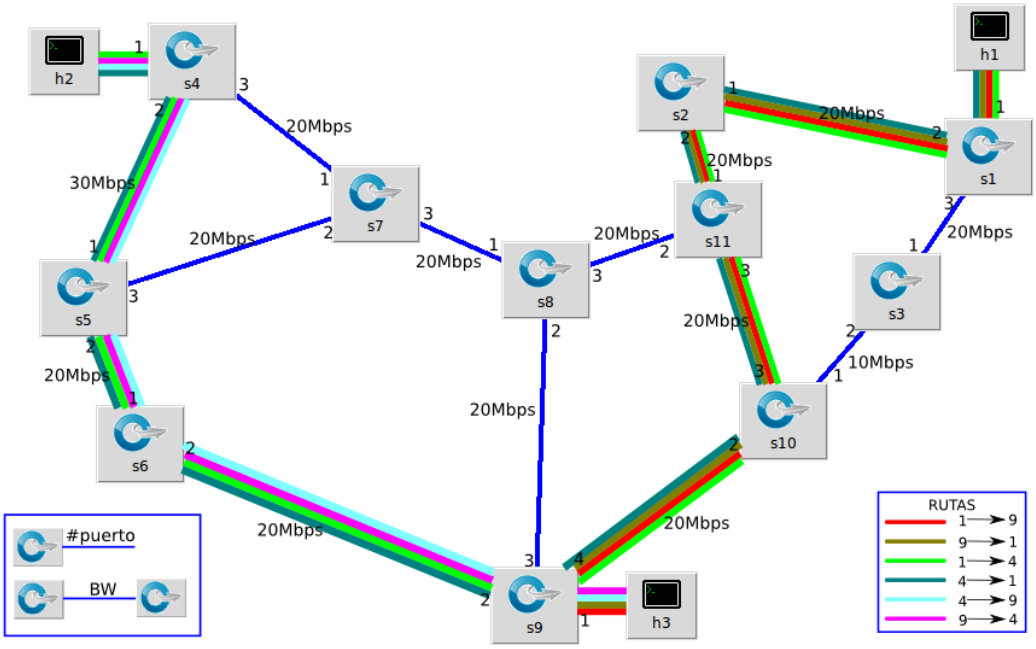


Figura 5-6.: Solución 3.

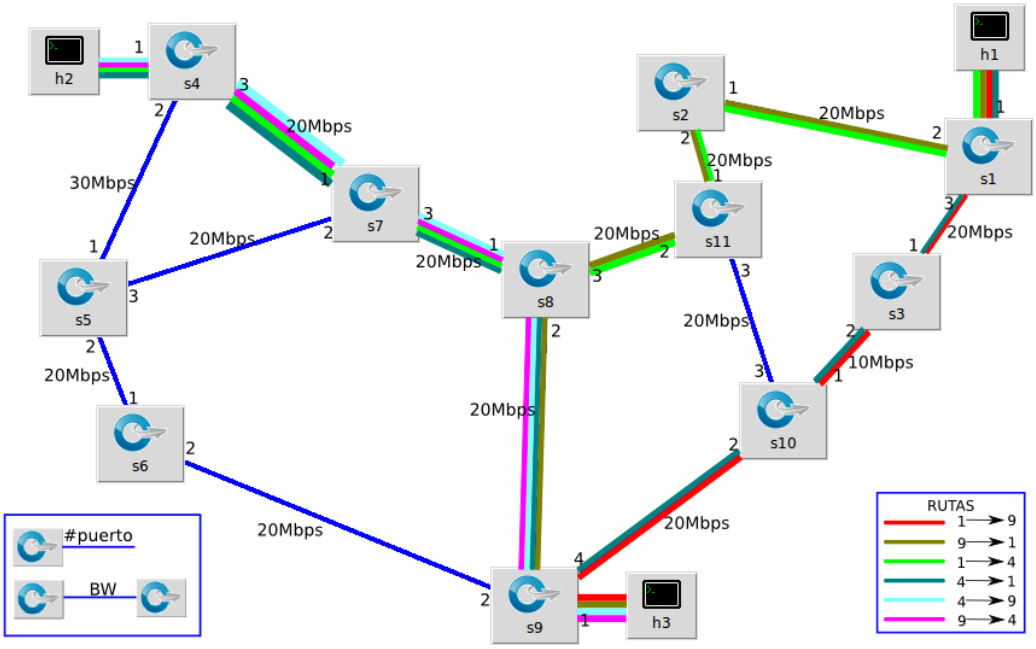


Figura 5-7.: Solución 4.

Tabla 5-1.: Capacidad y costo energético de los nodos de paso.

Nodo	K_n (Mbps)	E_0 (W)	Q_n (W/Mbps)	M (W)
2	40	9.942	0.044	11.696
3	30	8.207	0.048	9.655
5	70	14.437	0.036	16.985
6	40	9.942	0.044	11.696
7	60	13.027	0.038	15.326
8	60	13.027	0.038	15.326
10	50	11.536	0.041	13.572
11	60	13.027	0.038	15.326

Tabla 5-2.: Nodos de paso utilizados y costo energético mínimo de cada solución.

	X_n								
Nodo	2	3	5	6	7	8	10	11	Costo energético
Solución 1	0	1	1	1	0	0	1	0	44.122
Solución 2	1	0	0	0	1	1	0	1	49.023
Solución 3	1	0	1	1	0	0	1	1	54.063
Solución 4	1	1	0	0	1	1	1	1	63.946

preferida, siempre que las condiciones de la red lo permitan, y que la solución 4 debe ser la que se establezca como última medida. Al analizar detalladamente los resultados de las simulaciones, se verifica que, en efecto, la solución 1 es la solución que más veces se repite en todo el periodo de simulación.

La solución 2 se utiliza en los tiempos 180 (punto 1), 210, 490 y 550, donde las demandas que entran y salen del nodo 1 superan los 10 Mbps, que es la capacidad del enlace s3-s10, que hace parte de la solución 1, por lo que no se cumpliría la restricción de capacidad para ese enlace.

Por otro lado, la solución 3 es la que se utiliza en el punto 2, debido a que en este punto las demandas entre los nodos 1 y 9 superan los 10 Mbps, que es el límite del enlace s3-s10 en la solución 1, pero también se da que las demandas que entran y salen del nodo 9 superan los 20 Mbps, que es el límite del enlace s8-s9 en la solución 2.

Por último, en el punto 3, a pesar de que no es el de mayor demanda total, se utiliza la solución 4, que es la de mayor consumo energético. Esto se debe a que, en este punto, tanto la demanda entre los nodos 1 y 4 como entre los nodos 1 y 9 superan los 10 Mbps, por lo que sus rutas no pueden compartir enlaces, ya que superarían los 20 Mbps, que es la capacidad

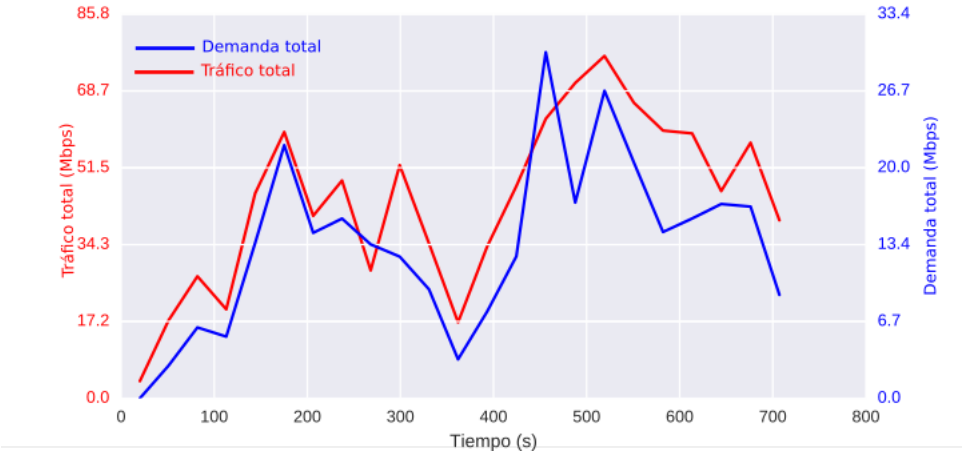


Figura 5-8.: Tráfico total en la red vs demanda total.

máxima de cualquiera de las rutas que se pueden establecer entre ellos. En la solución 4 se observa que la demanda que va del nodo 1 hacia el nodo 4 tiene una ruta diferente a la que tiene la demanda que va del nodo 4 al nodo 1, y lo mismo ocurre con la demanda que va del nodo 1 al nodo 9 y la que va del nodo 9 al nodo 1. La demanda entre los nodos 1 y 9 comparten enlaces con la demanda entre los nodos 4 y 9, pues estas dos, combinadas, no superan la capacidad de los enlaces s3-s7 y s7-s8, que es de 20 Mbps.

Ahora que vimos cómo cambió la utilización de elementos de red, pasamos a observar qué efecto tuvo esto en el consumo energético, que es fundamental para la evaluación de nuestro esquema de enrutamiento. En la Figura **5-9** se observa la variación en el consumo energético tanto de los enlaces como de los nodos a lo largo de la simulación. En esta figura se logra apreciar claramente la diferencia en consumo energético para los puntos en los que se usan diferentes soluciones, y que entre los puntos que usan la misma solución es muy pequeña la diferencia de consumo.

En la Figura **5-9** también se observa que el consumo total de los enlaces es muy pequeño, casi despreciable, comparado con el de los nodos, y que el consumo energético de los enlaces solo varía cuando cambia la cantidad que se utilice, lo que se explica recordando que el consumo energético de los enlaces es constante para nuestro escenario de simulación ($J_l = 0$).

Ahora nos fijamos en la forma en que varía el porcentaje de utilización de los nodos y enlaces respecto del tráfico total en la red, lo que se muestra en la Figura **5-10**, en la que se observa que en condiciones de demanda baja o media, donde se utiliza la misma cantidad de elementos de red, la utilización varía proporcionalmente a la cantidad total tráfico, pero cuando la carga de la red es alta, se utilizan más elementos y esto hace que baje un poco su

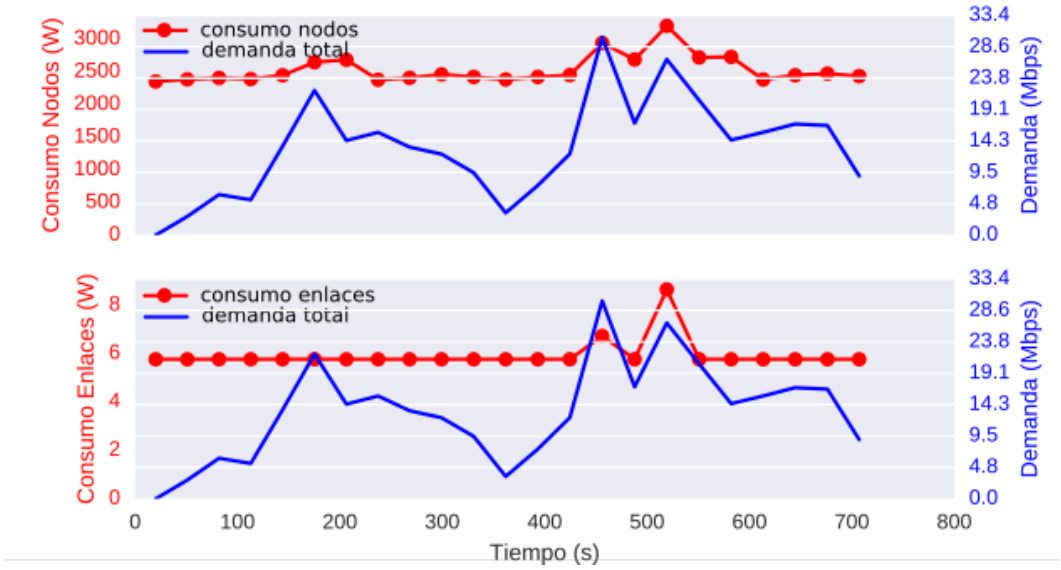


Figura 5-9.: Consumo energético vs demanda total.

porcentaje de uso, como puede verse para los puntos 2 y 3, donde se utiliza mayor cantidad de elementos de red y se observa una disminución.

Un dato importante es que nunca se observaron niveles de utilización que pudieran indicar saturación en ninguno de los elementos de la red, siendo el nivel máximo de utilización inferior a 70 % para los enlaces y a 40 % para los nodos. No sorprende que la utilización sea mayor para los enlaces que para los nodos, puesto que los primeros tienen menor capacidad, comparados con los segundos.

Como la utilización de los nodos y enlaces tiene que ver con la cantidad de flujo que procesan, es importante también observar cómo es la relación entre la demanda y el tráfico total en la red, que es la suma de los flujos que circulan por todos los enlaces de la red. Esta relación se muestra en la Figura **5-8** y se puede ver que el tráfico total es proporcional a la cantidad de demanda. Nuevamente en los puntos 2 y 3 se ve una excepción a este comportamiento, pues en el punto 2 hubo mayor demanda que en el 3, pero se nota mayor tráfico total en el punto 3, lo que se debe a que en este punto, como se mencionó antes, se establecen las rutas de la solución 4, que usan mayor cantidad de elementos de red.

Aparte de los puntos antes comentados, en la Figura **5-8** también resalta el punto a los 300 segundos de simulación, al cual llamamos punto 4, donde se nota un incremento en el tráfico total, a pesar de que la demanda va bajando. Para entender lo que sucede en este punto volvemos nuevamente hacia la Figura **5-2**, donde podemos observar que a pesar de que no

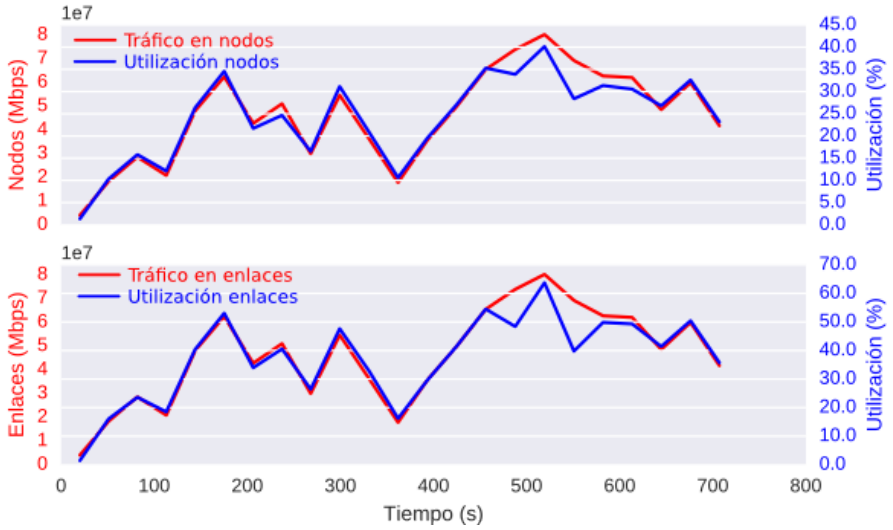


Figura 5-10.: Tráfico total vs utilización de elementos de red.

es un punto de gran demanda total, casi la mitad de esta se da entre los nodos 1 y 4, que es la ruta más larga en la red.

En la Figura **5-11** se presenta la relación entre el consumo energético en la red y el tráfico total que por ella circula, con la que se confirma que el consumo energético total en la red varía sustancialmente con el número de elementos usados, y muy poco con respecto a la cantidad de flujo que esos elementos procesan. Un ejemplo contundente de esto se observa a los 360 segundos, donde el tráfico cae abruptamente, pero no así el consumo energético, que baja muy levemente, debido a que se siguen utilizando los mismos 7 nodos y 6 enlaces para encaminar todo el tráfico de la red, por lo que la única diferencia es el volumen de tráfico que circula por los enlaces y es procesado en los nodos, que representa un valor pequeño, comparado con el consumo mínimo de los elementos de red, como se observa en la Tabla **5-1**.

Finalmente, la Figura **5-12** describe el comportamiento de los retrasos para cada una de las demandas en la red a lo largo del periodo simulado. Se observa una distribución simétrica de los valores para todas las demandas, lo que indica que siguen una distribución de probabilidad normal. Además, todas tienen su rango intercuartil por debajo de los 20 ms, con valores máximos que no llegan siquiera a los 30 ms, y solo algunos valores atípicos que superan los 50 ms para la ruta entre los nodos 1 y 4, pero que no llegan a los 40 ms para las demás rutas. Esto nos confirma que nuestro esquema garantiza el cumplimiento de la restricción de retraso para el caso de uso seleccionado, con un cierto nivel de holgura en todas las rutas.

Otro hecho que se observa es que el retraso de las rutas entre cada par de nodos es muy similar en ambos sentidos, lo que es un comportamiento esperado, ya que para lograr ahorro

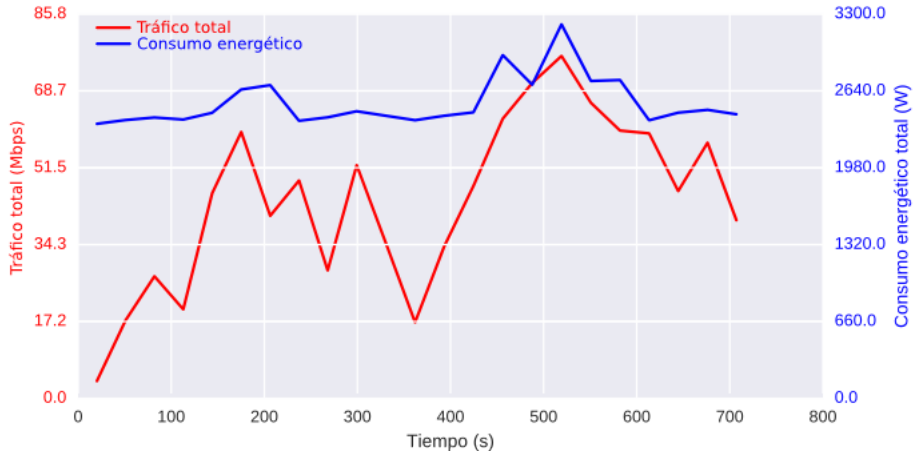


Figura 5-11.: Consumo energético vs cantidad de flujo total en la red.

energético, se utilizan los mismos enlaces para conectar los dispositivos de borde en ambos sentidos, salvo en ocasiones donde sea imposible hacerlo sin dejar de cumplir alguna de las restricciones fijadas, como fué el caso de la solución 4.

En este punto podemos concluir que el esquema de enrutamiento desarrollado cumple con los objetivos planteados en el modelo de optimización.

5.2.1. Evaluación ante diferentes restricciones de retraso máximo

Luego de verificar que el esquema planteado cumple con las restricciones establecidas para el caso de uso seleccionado, pasamos a analizar el comportamiento del retraso obtenido y su correspondiente consumo energético a medida que se va disminuyendo el valor máximo permitido de retraso para las demandas, utilizando la misma topología (Figura **5-1**) y demandas (Figura **5-2**) usadas en el análisis inicial . Los resultados se muestran en las Figuras **5-13** y **5-14**.

En la Figura **5-13** se comparan los consumos energéticos ante restricciones de 100, 50 y 20 ms, y se puede ver que en muchos tramos de la simulación se utilizan las mismas rutas para llevar las demandas, y que cuando se utilizan rutas diferentes, el consumo es mayor para la restricción de 20 ms. Se pudo verificar que hasta los 150 segundos, el esquema establece, para las 3 restricciones de retraso, la solución 1; en 180 y 210 segundos usa la solución 2 y en 240 vuelve a la solución 1.

Contrariamente, a los 270 segundos se usa la solución 2 para la restricción de 20 ms mientras que para las restricciones de 50 y 100 ms se utilizó la solución 1 , que tiene un menor consumo

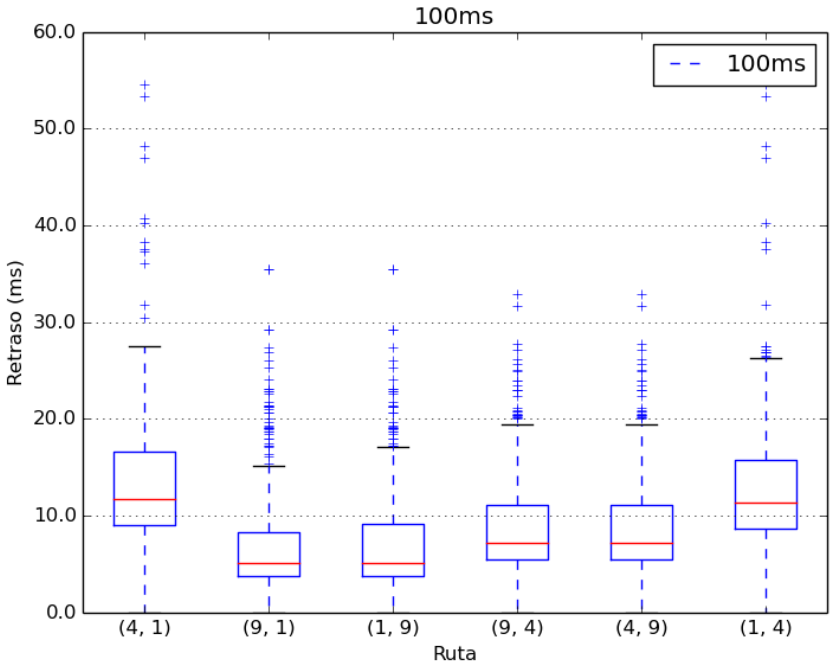


Figura 5-12.: Retrasos en cada una de las rutas.

energético, como se analizó anteriormente, pero que también utiliza una ruta más larga para encaminar la demanda entre los nodos 1 y 4, que son los más distantes en la topología, por lo que el retraso va a ser mayor. A lo largo de todo el periodo simulado se observa el mismo comportamiento, con soluciones iguales en casi todos los puntos, salvo contadas excepciones en las que, generalmente, se evidencia que el mayor consumo energético se da para la restricción de menor retraso, lo que permite inferir que el consumo aumenta a medida que disminuye el retraso máximo permitido.

Por otro lado, en la Figura **5-14** se muestran los resultados de los retrasos observados para las mismas restricciones. Se destaca que la ruta (4, 1) es la de mayor retraso, comparada con las otras rutas, independiente de la restricción que se imponga, lo que no es sorpresivo, si observamos la topología utilizada, y las soluciones encontradas, donde se ve que la ruta más corta posible entre estos dos nodos contiene cinco enlaces, mientras que para las otras, solo serían 3. Sin embargo, se observa que a medida que se disminuye el retraso permitido, se reduce un poco el rango intercuartil y la distancia entre los valores máximo y mínimo, aunque el valor máximo y promedio sigue siendo el mismo. La razón por la que no se observa una diferencia más amplia en estas medidas puede ser que la red se somete a altas demandas en un periodo muy corto de tiempo, que alcanza a marcar solo esta pequeña diferencia en los resultados totales.

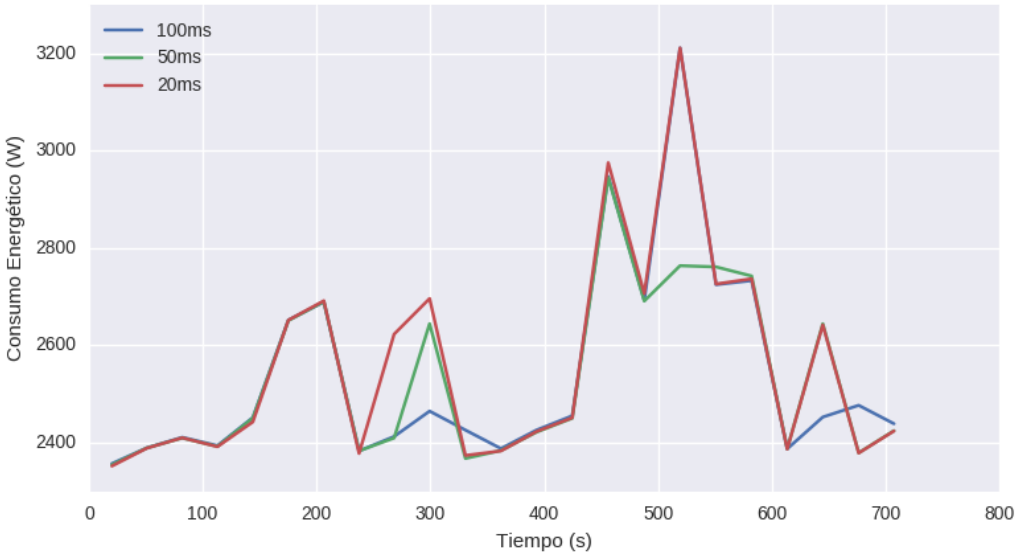


Figura 5-13.: Comparación de consumo energético para distintas restricciones del modelo.

Para ampliar un poco el análisis se llevaron a cabo simulaciones adicionales, con periodos de tiempo más amplios y donde se daban más periodos con altos niveles de demanda, con el fin de validar qué tanto se afectaba el comportamiento observado.

Estas nuevas simulaciones se llevan a cabo sólo para las restricciones de 50 y 20 ms, que son las que mayor consumo energético mostraron en las simulaciones anteriores, además en la Figura **5-13** se pudo ver que el comportamiento para 100 ms y 50 ms es muy similar, ya que ambas restricciones de retraso se cumplen con cierto nivel de holgura. En la Figura **5-15** se muestran las demandas que se utilizaron para este nuevo escenario de simulación.

Los resultados obtenidos en cuanto a consumo energético para el nuevo escenario de simulación se muestran en la Figura **5-16**, donde se ve que el consumo aumenta cuando el retraso máximo permitido disminuye, que es el comportamiento que se había observado anteriormente, pero en este caso, se ve mucho más claro debido a que la red se somente a escenarios de carga mucho más variables que en las simulaciones anteriores.

Es de notar que las mayores diferencias se marcan, no tanto ante escenarios de carga alta, sino cuando la carga de la red está en niveles medios, lo que indica que, ante escenarios de alta carga en la red, el esquema de enrutamiento utiliza las mismas rutas, las que mayor capacidad ofrecen ante esas condiciones de la red, y lo mismo ocurre cuando las cargas de tráfico con bajas, donde se usan las rutas que utilizan menos elementos de red. La diferencia se marca ante cargas de tráfico medios, donde existen varias opciones de rutas, y las restricciones impuestas son las que marcan la diferencia en cuanto a cúal de ellas se escoge.

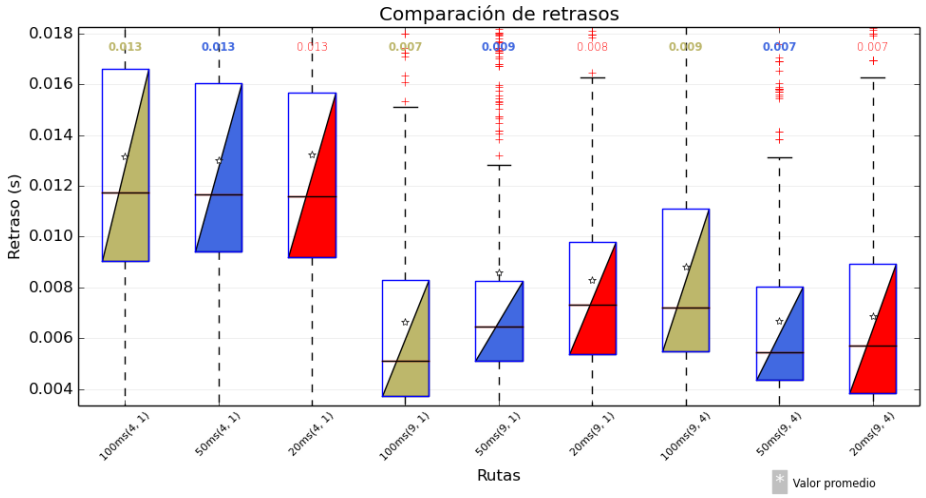


Figura 5-14.: Comparación de retrasos en las rutas para distintas restricciones del modelo.

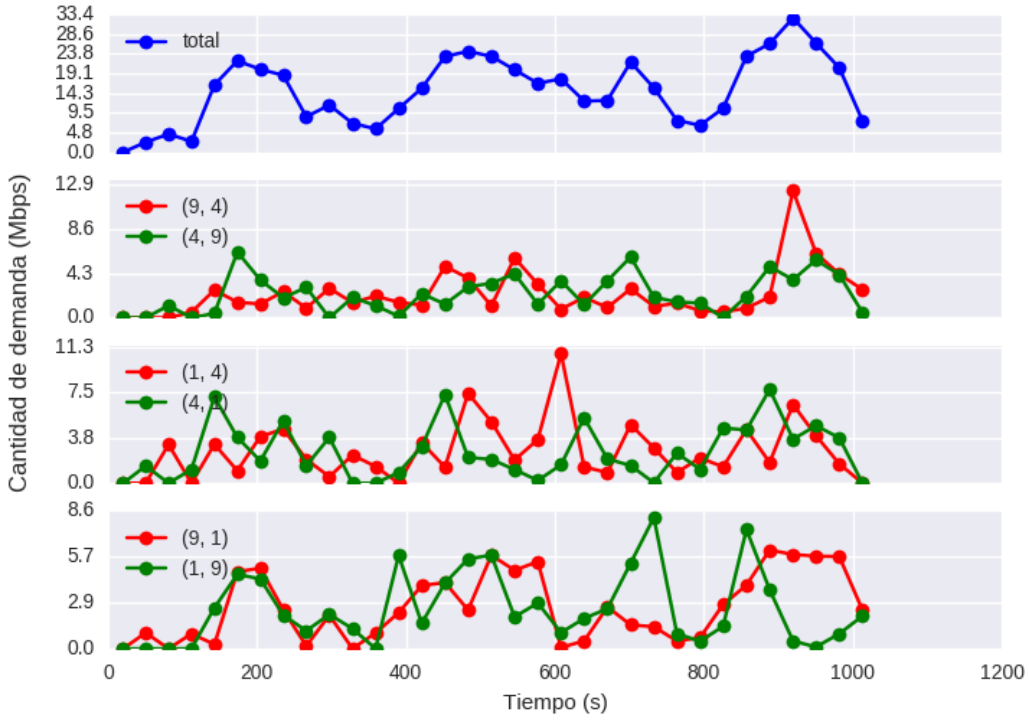


Figura 5-15.: Demandas usadas para 50 y 20 ms.

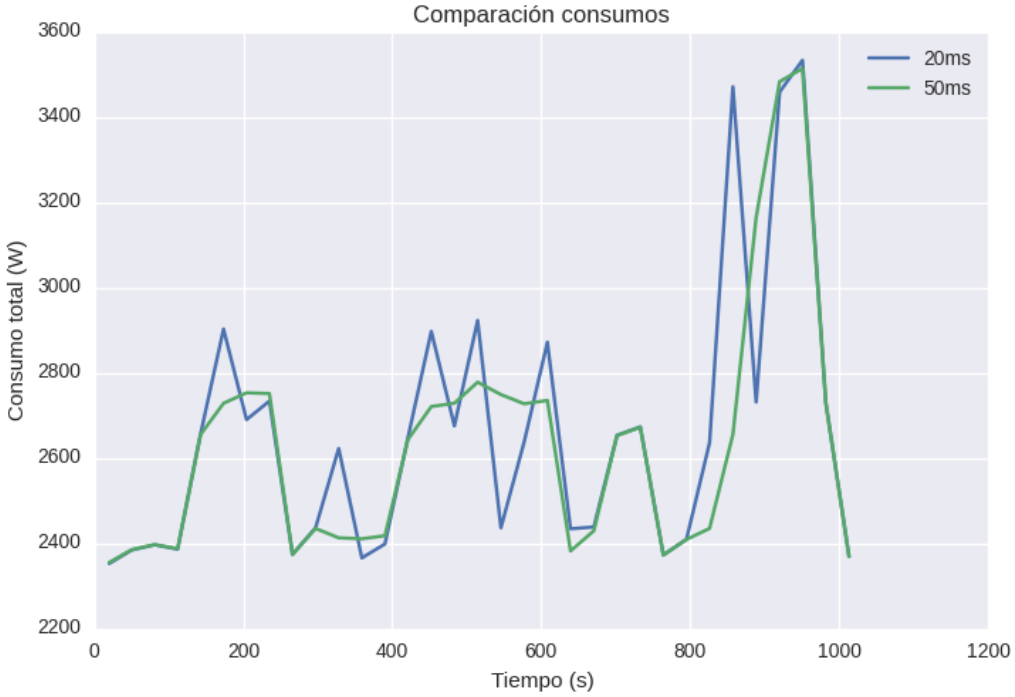


Figura 5-16.: Comparación de consumos energéticos para 50 y 20 ms.

Las Figuras **5-17** y **5-18** muestran otras soluciones que se dan para el nuevo escenario de simulación, que no se dieron en el escenario anterior, lo que nos refuerza la idea de que el esquema ofrece diferentes soluciones de acuerdo a las condiciones de la red.

En la Figura **5-19** se presenta la utilización promedio de los nodos y los enlaces para ambas restricciones de retraso, donde se ve que disminuye, en condiciones de demanda media o alta, a medida que disminuye la restricción, y que en condiciones de baja demanda se da la misma utilización para ambas restricciones. Esta observación confirma la hecha anteriormente de que ante condiciones de baja demanda se utilizan las mismas rutas sin importar la restricción. Si comparamos este comportamiento de la utilización con el del consumo energético, se ve que hay una relación inversa, esto es, que cuando el consumo es mayor para la restricción de 20 ms, la utilización es menor, y viceversa. Este comportamiento es debido a que circula la misma cantidad de flujo en la red pero varía el número de elementos utilizados según sea la restricción, como se observó anteriormente. Tanto los nodos como los enlaces muestran el mismo patrón de comportamiento, con variaciones un poco mayores para los enlaces, lo cual es normal, pues estos tienen menor capacidad que los nodos.

En este escenario nuevo tampoco se observaron niveles de utilización suficientemente altos

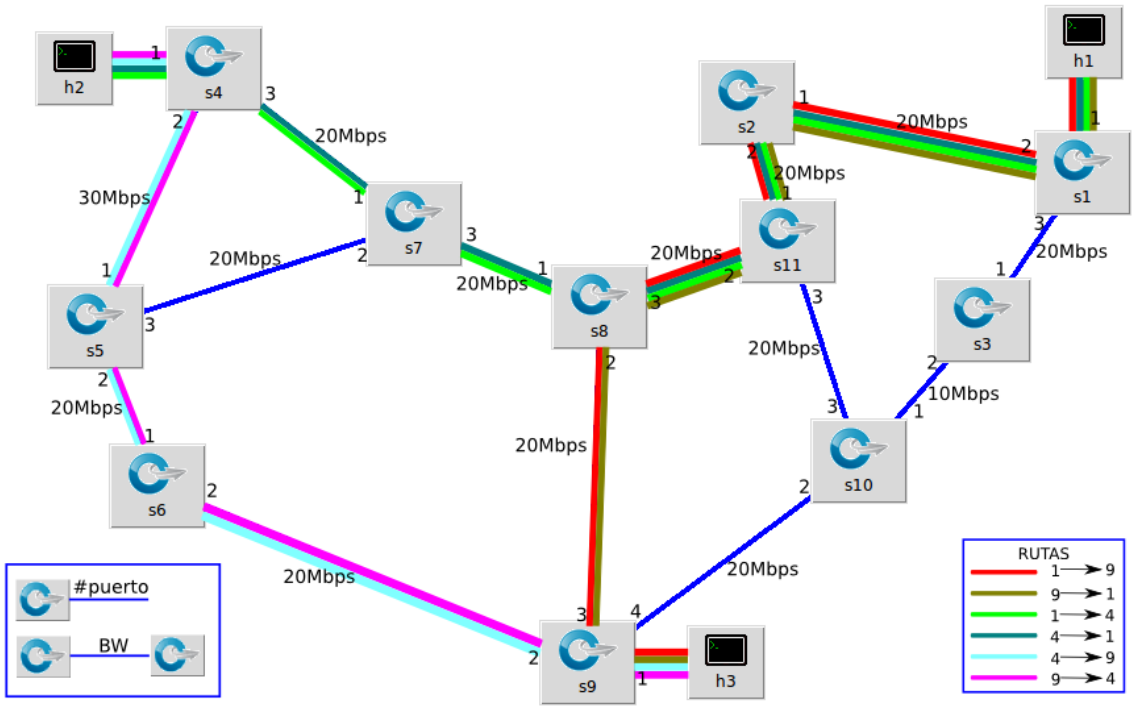


Figura 5-17.: Solucion 5.

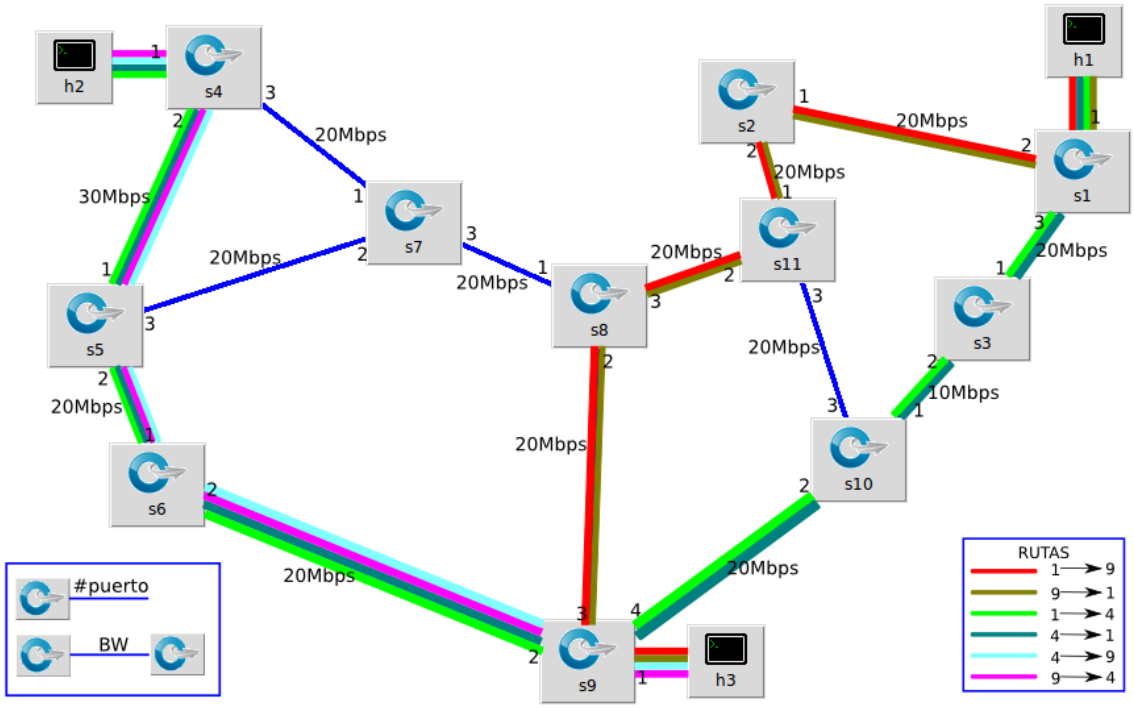


Figura 5-18.: Solucion 6.

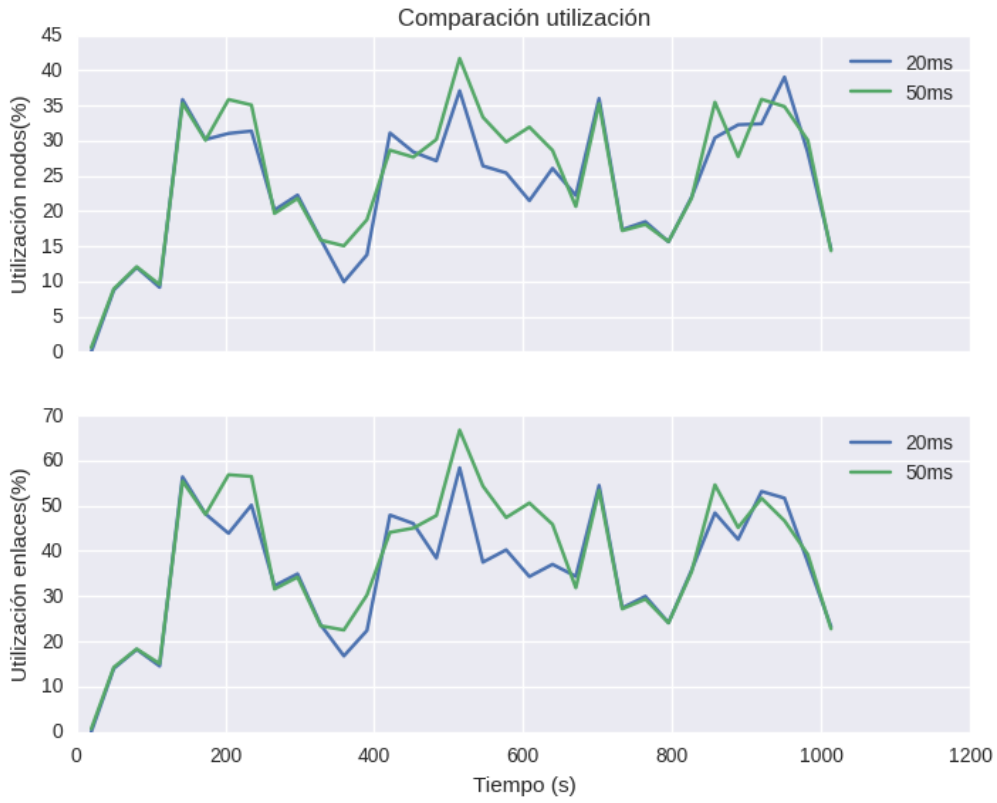


Figura 5-19.: Comparación utilización de nodos y enlaces para 50 y 20 ms.

para pensar en saturación de los elementos de la red. Los porcentajes máximos de uno se mantuvieron casi en los mismos niveles que en los experimentos anteriores (70 % y 40 % para enlaces y nodos, respectivamente).

Finalmente, en la Figura **5-20** se presenta un diagrama de cajas y bigotes en el que se comparan los retrasos observados para cada una de las rutas establecidas. En este diagrama se observa que el retraso para la restricción de 20 ms es claramente inferior al que se obtiene con la restricción de 50 ms, con mucha menor variabilidad, sobretodo para las rutas (4, 1) y (9, 1), en las que son menores tanto los rangos intercuartil como los máximos. En estas rutas se da una mejora de 7.14 % y 30.77 % en el valor promedio del retraso, respectivamente, comparado con los retrasos obtenidos para la restricción de 50ms. Para la ruta (4, 9) la diferencia es mucho más estrecha, incluso el valor promedio es el mismo, pero el rango intercuartil si es ligeramente menor, así como el valor máximo, lo que es indicativo de que el retraso en esa ruta se mantiene un poco más estable para la restricción de 20 ms respecto de la restricción de 50 ms.

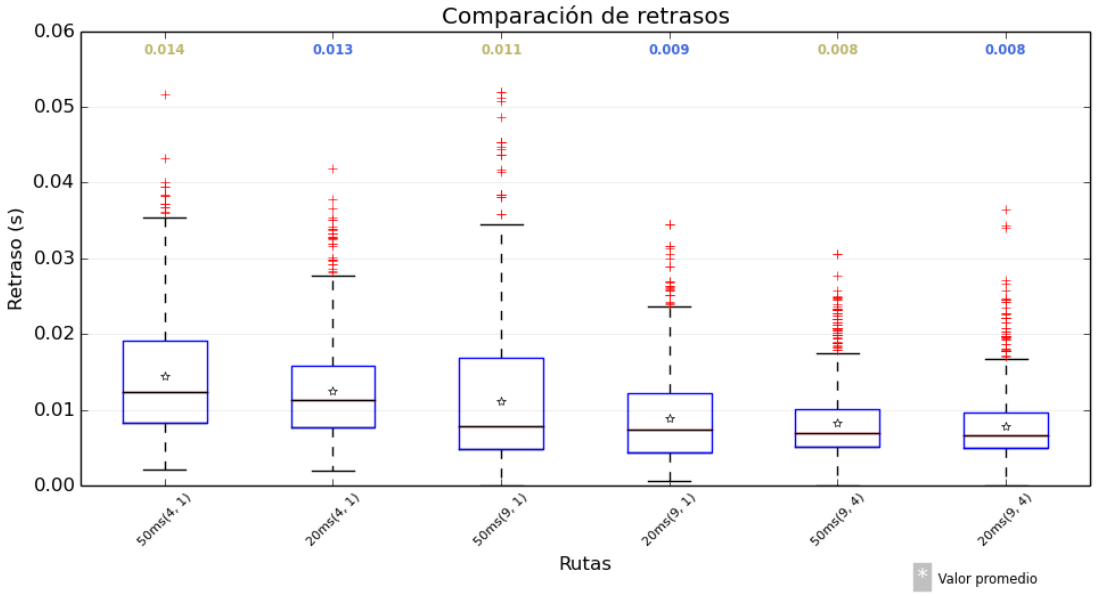


Figura 5-20.: Comparación de retrasos en las rutas para 50 y 20 ms.

5.2.2. Comparación de resultados con otras propuestas

Luego de verificar el correcto funcionamiento del esquema de enrutamiento planteado, se buscó comparar los resultados son los obtenidos con GreenMST y MdST, que son propuestas que buscan objetivos similares.

Para este análisis comparativo, y con el fin de no partir desde una posición de ventaja respecto de las otras dos estrategias analizadas, se utilizan los valores obtenidos para la restricción de retraso es de 20 ms que, como se vió en los análisis anteriores, es con la que se obtiene mayor consumo energético, respecto de las otras restricciones utilizadas en los análisis anteriores. Se hicieron simulaciones independientes para las otras dos propuestas, sometiendo la red (Figura **5-1**) a las mismas demanda mostradas en la Figura **5-15**.

En la Figura **5-21** se presenta la variación del consumo energético con las tres propuestas a lo largo del tiempo de simulación. En esta figura se ve que el consumo en muy inferior en todo momento respecto al de GreenMST, que muestra muy poca variación, lo cual es de esperarse, pues esta propuesta no cambia las rutas establecidas en ningún momento, salvo que haya un cambio en la topología de la red; la variación observada se debe al flujo que circula por los nodos utilizados. Respecto de MdST, se observa que el consumo es bastante inferior en casi todo momento, excepto en los últimos puntos de la simulación, donde la demanda total llega a su valor máximo. En este punto lo que sucede es que las rutas de menor

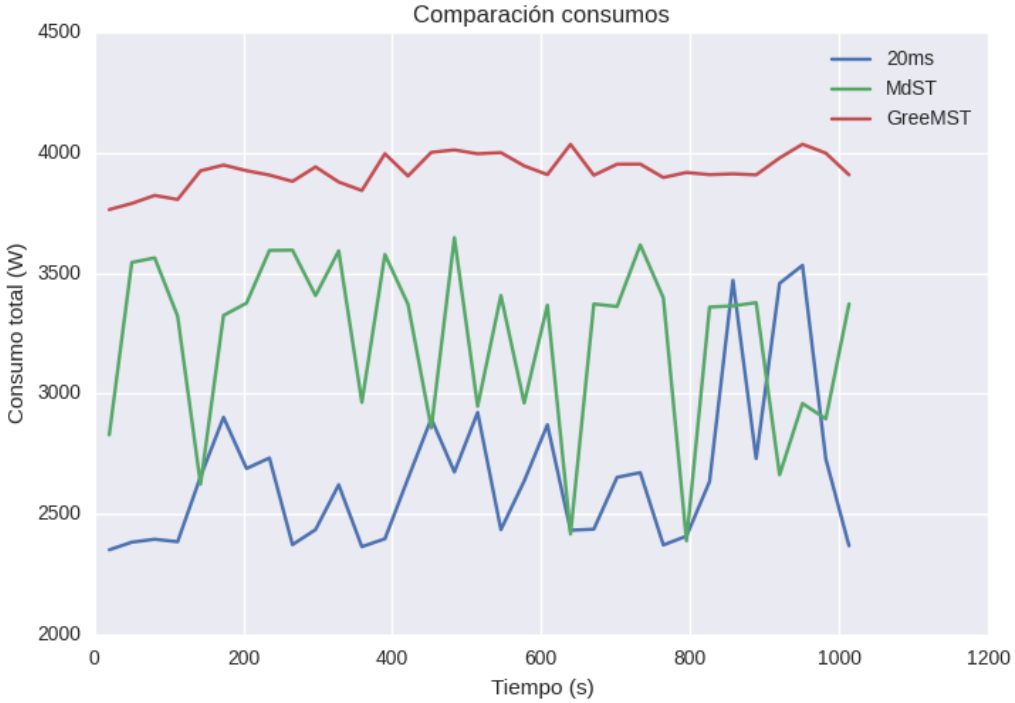


Figura 5-21.: Comparación de consumos energéticos entre las 3 propuestas analizadas.

retraso establecidas por MdST son unas que no garantizan el cumplimiento de la restricción de capacidad de los enlaces, mientra que el esquema dinámico logra encontrar unas rutas que cumplen todas las restricciones establecidas, aunque consumen un poco más de energía.

Para tener una mejor idea del comportamiento del consumo energético en todo el espacio de la simulación, se presenta el diagrama de cajas de la Figura **5-22**, donde aprecia que el consumo energético promedio para el esquema planteado es 20.60 % menor respecto de MdST y 34.40 % respecto de GreenMST. En esta figura se confirma la observación hecha para la figura anterior respecto del consumo energético de la red con GreenMST, pues se ve que hay muy poca dispersión en las medidas, pero siempre muy superior respecto de los valores para las otras dos estrategias analizadas. Por el contrario, para MdST se observa una variabilidad mucho mayor en las medidas, con valores máximo y promedio muy superiores respecto al esquema planteado en este trabajo, que tiene un valor máximo que alcanza a ser inferior a los valores promedio de las otras dos estrategias.

Los resultados en cuanto a retrasos se presentan en diagrama de cajas de la Figura **5-23**, en el que se logra visualizar nuevamente que el esquema planteado brinda menores retrasos para todas las rutas, y que GreenMST es la que peores resultados muestra en todas las rutas, teniendo la mayor variabilidad de todas, así como los valores promedio y máximos más

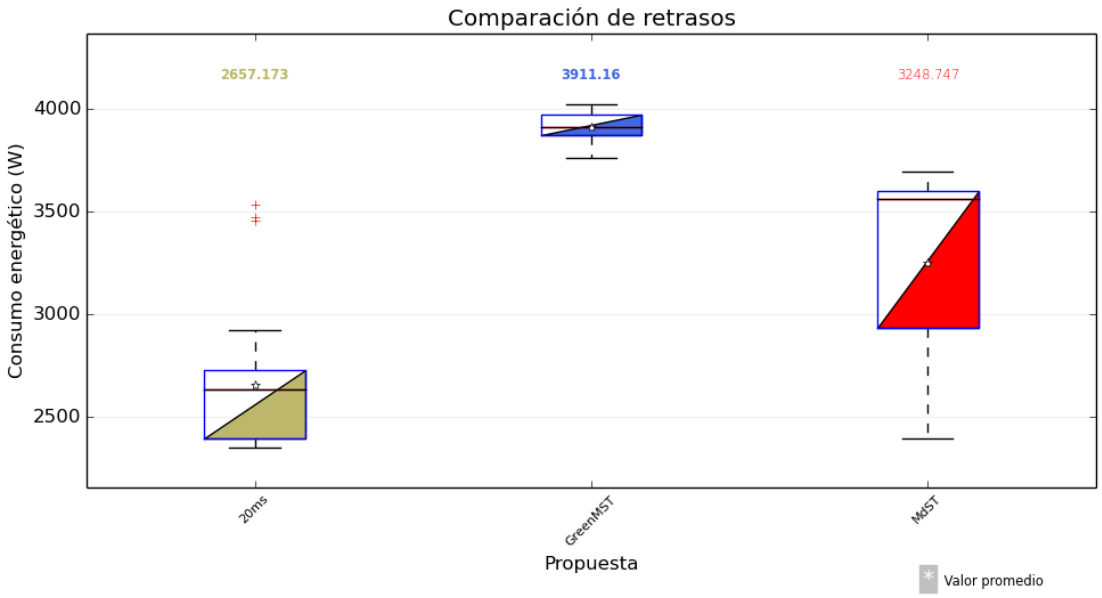


Figura 5-22.: Diagrama de cajas de consumos energéticos para las 3 propuestas analizadas.

altos. El esquema de enrutamiento dinámico planteado logra mejorar los valores de retraso promedio en un 66.67 % y 30.77 % para la ruta (9, 1), y 23.53 % y 13.33 % para la ruta (4, 1), respecto de GreenMST y MdST, respectivamente. Para la ruta (9, 4) se observa una mejora de 11.11 % respecto de GreenMST en valor promedio, pero también se observa una menor variabilidad y valor máximo respecto a MdST, aunque el valor promedio sea el mismo.

Los resultados obtenidos en la comparación de estas tres propuestas, tanto en retraso como en consumo energético, permiten concluir que el esquema de enrutamiento dinámico desarrollado cumple con las restricciones establecidas, permitiendo lograr caminos con menor retraso y consumo energético, respecto de las otras dos propuestas analizadas.

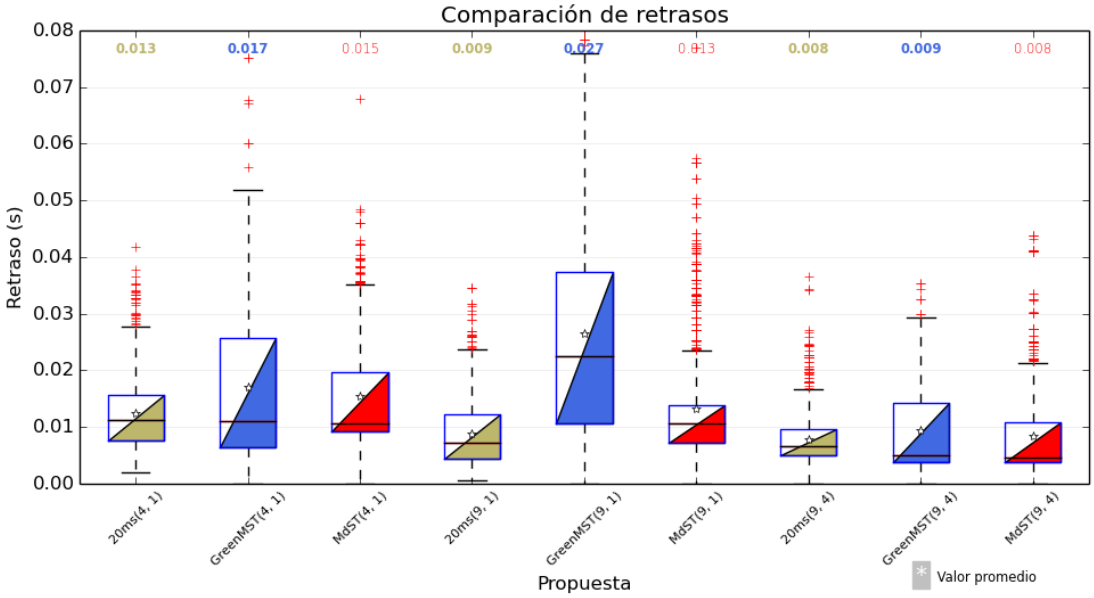


Figura 5-23.: Comparación de retrasos entre las 3 propuestas analizadas.

6. Conclusiones y recomendaciones

6.1. Conclusiones

En primera medida, se pudo verificar que la obtención de la matriz de tráfico en las redes definidas por software se puede lograr, de forma sencilla y confiable, haciendo uso de los medidores que se plantean en la versión 1.3 del protocolo *OpenFLow*, y que estas matrices sirven de base para la elaboración de esquemas de enrutamiento dinámicos como el que se planteó, desarrolló y evaluó en esta propuesta.

El análisis de los resultados obtenidos permite concluir que el esquema de enrutamiento dinámico presentado en este trabajo de investigación permite realizar el enrutamiento para tráfico de telemedicina, que fué el caso de uso seleccionado, asegurando calidad de servicio y haciendo un uso óptimo de los recursos de red, lo que permite disminuir su consumo energético total.

Se pudo comprobar que a medida que se disminuye el retraso máximo permitido, se incrementa el consumo energético de la red, debido a que se utilizan más elementos de red para ofrecer rutas disjuntas, que permitan evitar la saturación de los enlaces y la degradación en el retraso de la ruta.

Al comparar los resultados con los de GreenMST y MdST, que son dos propuestas que buscan objetivos similares en la red, se evidenció una mejora en el consumo energético de 20.60 % y 34.40 %, respectivamente.

En cuanto al retraso, se obtuvieron mejoras de hasta 66.67 % respecto de GreenMST y 30.77 % respecto de MdST, lo que permite decir que el esquema planteado puede utilizarse en otros escenarios que tengan restricciones en cuanto al retraso máximo permitido, como pueden ser los juegos en línea o el streaming de video.

Aunque el análisis de la propuesta se hizo buscando el mínimo consumo energético, se puede modificar la función objetivo del problema de optimización para lograr otros objetivos en la red, como puede ser la minimización del retraso total, o maximizar el ancho de banda disponible en las rutas establecidas, entre otras, sin hacer mayores modificaciones al esquema de enrutamiento completo.

6.2. Trabajo futuro

El primer trabajo futuro a realizar es la evaluación del funcionamiento del esquema desarrollado en una red con dispositivos físicos, utilizando diferentes topologías, y utilizando distintos tipos de tráfico en la red, para contrastar con los resultados obtenidos en el entorno de simulación.

Luego de esto, se pueden realizar pruebas cambiando la función objetivo en el modelo de optimización, buscando, por ejemplo, minimizar el retraso total o maximizar el ancho de banda disponible, entre otros objetivos, para validar el comportamiento del esquema planteado.

Otro trabajo importante es el de evaluar la escalabilidad del esquema, probándolo en diferentes topologías que tengan mayor cantidad de nodos y enlaces, lo que aumentará la complejidad del problema a resolver.

Una vez se evalúe la escalabilidad del esquema, se puede plantear e implementar algún tipo de heurística o metaheurística para resolver el problema de optimización y comparar sus resultados con los de la solución exacta, en busca de lograr llevar el esquema hacia una implementación real, en una red en producción.

Finalmente, se pueden mejorar los resultados obtenidos al modificar el modelo de optimización, para que se puedan calcular varias rutas para cada una de las demandas en la red, aunque teniendo siempre en cuenta que los caminos establecidos pueden tener diferentes retrasos y cómo puede afectar esto el rendimiento de los servicios en la red.

A. Anexo: Archivo gml de red Abilene modificado

```
graph [
  DateObtained "3/02/11"
  GeoLocation "US"
  GeoExtent "Country"
  Network "Abilene"
  Provenance "Primary"
  Access 0
  Source "http://www.internet2.edu/pubs/200502-IS-AN.pdf"
  Version "1.0"
  Type "REN"
  DateType "Historic"
  Backbone 1
  Commercial 0
  label "Abilene"
  ToolsetVersion "0.3.34dev-20120328"
  Customer 0
  IX 0
  SourceGitVersion "e278b1b"
  DateModifier "="
  DateMonth "02"
  LastAccess "3/02/11"
  Layer "IP"
  Creator "Topology Zoo Toolset"
  Developed 0
  Transit 0
  NetworkDate "2005_02"
  DateYear "2005"
  LastProcessed "2011_09_01"
  Testbed 0
  node [
    id 0
    label "New York"
    Country "United States"
    Longitude -74.00597
    Internal 1
    Latitude 40.71427
  ]
```

```
node [
  id 1
  label "Chicago"
  Country "United States"
  Longitude -87.65005
  Internal 1
  Latitude 41.85003
]
node [
  id 2
  label "Washington DC"
  Country "United States"
  Longitude -77.03637
  Internal 1
  Latitude 38.89511
]
node [
  id 3
  label "Seattle"
  Country "United States"
  Longitude -122.33207
  Internal 1
  Latitude 47.60621
]
node [
  id 4
  label "Sunnyvale"
  Country "United States"
  Longitude -122.03635
  Internal 1
  Latitude 37.36883
]
node [
  id 5
  label "Los Angeles"
  Country "United States"
  Longitude -118.24368
  Internal 1
  Latitude 34.05223
]
node [
  id 6
  label "Denver"
  Country "United States"
  Longitude -104.9847
  Internal 1
  Latitude 39.73915
]
```

```
node [
  id 7
  label "Kansas City"
  Country "United States"
  Longitude -94.62746
  Internal 1
  Latitude 39.11417
]
node [
  id 8
  label "Houston"
  Country "United States"
  Longitude -95.36327
  Internal 1
  Latitude 29.76328
]
node [
  id 9
  label "Atlanta"
  Country "United States"
  Longitude -84.38798
  Internal 1
  Latitude 33.749
]
node [
  id 10
  label "Indianapolis"
  Country "United States"
  Longitude -86.15804
  Internal 1
  Latitude 39.76838
]
edge [
  source 0
  target 1
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
]
edge [
  source 0
  target 2
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
]
edge [
  source 1
```

```
  target 10
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
]
edge [
  source 2
  target 9
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
  bw 5
]
edge [
  source 3
  target 4
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
  bw 15
]
edge [
  source 3
  target 6
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
]
edge [
  source 4
  target 5
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
]
edge [
  source 4
  target 6
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
]
edge [
  source 5
  target 8
  LinkType "OC-192"
  LinkLabel "OC-192c"
  LinkNote "c"
```

```
  ]
  edge [
    source 6
    target 7
    LinkType "OC-192"
    LinkLabel "OC-192c"
    LinkNote "c"
  ]
  edge [
    source 7
    target 8
    LinkType "OC-192"
    LinkLabel "OC-192c"
    LinkNote "c"
  ]
  edge [
    source 7
    target 10
    LinkType "OC-192"
    LinkLabel "OC-192c"
    LinkNote "c"
  ]
  edge [
    source 8
    target 9
    LinkType "OC-192"
    LinkLabel "OC-192c"
    LinkNote "c"
  ]
  edge [
    source 9
    target 10
    LinkType "OC-192"
    LinkLabel "OC-192c"
    LinkNote "c"
  ]
  host [
        switch 1
  ]
  host [
        switch 4
  ]
  host [
        switch 9
  ]
]
```

B. Anexo: Código python para crear topologías Mininet

```
#!/usr/bin/python

#GraphML-Topo-to-Mininet-Network-Generator
#This file parses Network Topologies in GraphML format from the Internet Topology Zoo.
#A python file for creating Mininet Topologies will be created as Output.
#Files have to be in the same directory.
#Arguments:
#  -f              [filename of GraphML input file]
#  --file          [filename of GraphML input file]
#  -o              [filename of GraphML output file]
#  --output        [filename of GraphML output file]
#  -b              [number as integer for bandwidth in mbit]
#  --bw            [number as integer for bandwidth in mbit]
#  --bandwidth     [number as integer for bandwidth in mbit]
#  -c              [controller ip as string]
#  --controller    [controller ip as string]
#  -h              [Lista de conmutadores donde van hosts]
#Without any input, program will terminate.
#Without specified output, outputfile will have the same name as the input file.
#This means, the argument for the outputfile can be omitted.
#Parameters for bandwith and controller ip have default values,if they are omitted,too.
#Wed Jul 17 02:59:06 PDT 2013
#TODO's:
#  -   fix double name error of some topologies
#  -   fix topoparsing (choose by name, not element <d..>)
#          =    topos with duplicate labels
#  -   use 'argparse' for script parameters, eases help creation
##############################################################################################
import sys
import math
from sys import argv
import networkx as nx
from conmutador import Conmutador

input_file_name = ''
output_file_name = ''
bandwidth_argument = ''
```

```
controller_ip = ''
hosts = []
# first check commandline arguments
for i in range(len(argv)):
    if argv[i] == '-f':
        input_file_name = argv[i+1]
    if argv[i] == '--file':
        input_file_name = argv[i+1]
    if argv[i] == '-o':
        output_file_name = argv[i+1]
    if argv[i] == '--output':
        output_file_name = argv[i+1]
    if argv[i] == '-b':
        bandwidth_argument = argv[i+1]
    if argv[i] == '--bw':
        bandwidth_argument = argv[i+1]
    if argv[i] == '--bandwidth':
        bandwidth_argument = argv[i+1]
    if argv[i] == '-c':
        controller_ip = argv[i+1]
    if argv[i] == '--controller':
        controller_ip = argv[i+1]
    if argv[i] == '-h':
        hosts1 = argv[i+1].split(',')
        print type(hosts1),hosts1
        hosts = argv[i+1].split(',')
# terminate when inputfile is missing
if input_file_name == '':
    sys.exit('\n\tNo input file was specified as argument....!')
# define string fragments for output later on
outputstring_1 = '''
class GeneratedTopo( Topo ):
    "Internet Topology Zoo Specimen."

    def __init__( self, **opts ):
        "Create a topology."

        # Initialize Topology
        Topo.__init__( self, **opts )
'''
outputstring_2a='''
        # add nodes, switches first...
'''
outputstring_2b='''
        # ... and now hosts
'''
outputstring_3a='''
        # add edges between switch and corresponding host
```

```
'''
outputstring_3b='''
        # add edges between switches
'''
outputstring_4a='''
topos = { 'generated': ( lambda: GeneratedTopo() ) }
# HERE THE CODE DEFINITION OF THE TOPOLOGY ENDS
'''
#WHERE TO PUT RESULTS
outputstring_to_be_exported = ''
outputstring_to_be_exported += outputstring_1
# id:value dictionaries
id_node_name_dict   = {}     # to hold all 'id: node_name_value' pairs
id_longitude_dict   = {}     # to hold all 'id: node_longitude_value' pairs
id_latitude_dict    = {}     # to hold all 'id: node_latitude_value' pairs
conmutadores = {}
capacidades = {}
#FIND OUT WHAT KEYS ARE TO BE USED, SINCE THIS DIFFERS IN DIFFERENT GRAPHML TOPOLOGIES
'''for i in index_values_set:
    if i.attrib['attr.name'] == 'label' and i.attrib['for'] == 'node':
        node_label_name_in_graphml = i.attrib['id']
    if i.attrib['attr.name'] == 'Longitude':
        node_longitude_name_in_graphml = i.attrib['id']
    if i.attrib['attr.name'] == 'Latitude':
        node_latitude_name_in_graphml = i.attrib['id']
'''
# NOW PARSE ELEMENT SETS TO GET THE DATA FOR THE TOPO
# GET NODE_NAME DATA
# GET LONGITUDE DATK
# GET LATITUDE DATA
G = nx.read_gml(input_file_name)
G = G.to_undirected()
for n in G.node:
    node_index_value = G.node[n]['id']
    id_node_name_dict[node_index_value] = G.node[n]['label'].replace(' ', '_')
    id_longitude_dict[node_index_value] = G.node[n]['Longitude']
    id_latitude_dict[node_index_value]  = G.node[n]['Latitude']
    if hosts1 == None :
        hosts.append(node_index_value)
# STRING CREATION
# FIRST CREATE THE SWITCHES AND HOSTS
tempstring1 = ''
tempstring2 = ''
tempstring3 = ''
#print id_node_name_dict
for i in range(0, len(id_node_name_dict)):
    #conmutadores[id_node_name_dict] =
    c = Conmutador(id_node_name_dict)
```

```
    conmutadores[i]=c
    #create switch
    temp1 =  '        '
    temp1 += id_node_name_dict[i]
    temp1 += " = self.addSwitch( 's"
    temp1 += str(i+1)
    temp1 += "' )\n"
    tempstring1 += temp1

for i in hosts:
    #create corresponding host
    i = int(i)-1
    if i not in id_node_name_dict.keys():
        continue
    print i
    print id_node_name_dict
    temp2 =  '        '
    temp2 += id_node_name_dict[i]
    temp2 += "_host = self.addHost( 'h"
    temp2 += str(i+1)
    temp2 += "' )\n"
    tempstring2 += temp2

    port1 = conmutadores[i].agregar_enlace()
    print port1
    # link each switch and its host...
    temp3 =  '        self.addLink( '
    temp3 += 'node1 = '+id_node_name_dict[i]
    temp3 += ' , port1 = '+ str(port1)
    temp3 += ' , '
    temp3 += 'node2 = '+id_node_name_dict[i]
    temp3 += "_host )"
    temp3 += '\n'
    tempstring3 += temp3

outputstring_to_be_exported += outputstring_2a
outputstring_to_be_exported += tempstring1
outputstring_to_be_exported += outputstring_2b
outputstring_to_be_exported += tempstring2
outputstring_to_be_exported += outputstring_3a
outputstring_to_be_exported += tempstring3
outputstring_to_be_exported += outputstring_3b


# SECOND CALCULATE DISTANCES BETWEEN SWITCHES,
#   set global bandwidth and create the edges between switches,
#   and link each single host to its corresponding switch
```

```
tempstring4 = ''
tempstring5 = ''
distance = 0.0
latency = 0.0
print G.node
print G.edge
enlaces = []
for e in G.edge:
    #print e, G.edge[e]
    src_id = e
    for d in G.edge[e]:
        dst_id = d
        #print dst_id
        print enlaces
        if (dst_id,src_id) not in enlaces and (src_id,dst_id) not in enlaces:
            enlaces.append((src_id,dst_id))
    #CALCULATE DELAYS
    #CALCULATION EXPLANATION
    #formula: (for distance)
    #dist(SP,EP)=arccos{sin(La[EP])*sin(La[SP])
    #                +cos(La[EP])*cos(La[SP])*cos(Lo[EP]-Lo[SP])}*r
    #r = 6378.137 km
    #formula: (speed of light, not within a vacuumed box)
    #v = 1.97 * 10**8 m/s
    #formula: (latency being calculated from distance and light speed)
    #t = distance / speed of light
    #t(in ms)=(distance in km*1000(for meters))/(speed of light/1000(for ms))
    #ACTUAL CALCULATION: implementing this was no fun.
            first_product = math.sin(float(id_latitude_dict[dst_id]))
                                    * math.sin(float(id_latitude_dict[src_id]))
            second_product_first_part = math.cos(float(id_latitude_dict[dst_id]))
                                    * math.cos(float(id_latitude_dict[src_id]))
            second_product_second_part= math.cos((float(id_longitude_dict[dst_id]))
                                        - (float(id_longitude_dict[src_id])))
            distance = math.radians(math.acos(first_product +
                        (second_product_first_part*second_product_second_part))
                                    )*6378.137
            # t(in ms)=(distance in km*1000(for meters))/(speed of light/1000(for ms))
            # t=(distance*1000)/( 1.97*10**8/1000)
            latency = ( distance * 1000 ) / ( 197000 )
            # BANDWIDTH LIMITING
            #set bw to 10mbit if nothing was specified otherwise on startup
            if bandwidth_argument == '':
                bw = '10';
            print type(G.edge[e][dst_id])
            G.edge[e][dst_id].setdefault('bw',10)
            if G.edge[e][dst_id]['bw'] != None:
                bw = G.edge[e][dst_id]['bw']
```

```
            # ... and link all corresponding switches with each other
            temp4 = ''
            port_src = conmutadores[src_id].agregar_enlace()
            port_dst = conmutadores[dst_id].agregar_enlace()
            capacidades[(conmutadores[src_id].id,port_src+1)]= bw
            capacidades[(conmutadores[dst_id].id,port_dst+1)] = bw
            temp4 =  '        self.addLink( '
            temp4 += ' node1 = '+id_node_name_dict[src_id]
            temp4 += ' , port1 = '+ str(port_src)
            temp4 += " , node2 = "
            temp4 += id_node_name_dict[dst_id]
            temp4 += ", port2 = "+ str(port_dst)
            temp4 += ", bw="
            temp4 += str(bw)
            temp4 += ", delay='"
            temp4 += str(latency)
            temp4 += "ms')"
            temp4 += '\n'
            # next line so i dont have to look up other possible settings
            #temp += "ms', loss=0, max_queue_size=1000, use_htb=True)"
            tempstring4 += temp4

outputstring_to_be_exported += tempstring4
outputstring_to_be_exported += '\ncapacidades_puertos = '+str(capacidades)
outputstring_to_be_exported += outputstring_4a


string_minievents = '''#!/usr/bin/python

#Estos son los import de minievents (y los que agrego yo)
import time
import json
import argparse
import os
from time import sleep

from minisched import scheduler

from mininet.log import setLogLevel, info, debug
from mininet.net import Mininet
from mininet.node import Host, UserSwitch, RemoteController,Node,CPULimitedHost
from mininet.link import TCLink, Intf, Link
from mininet.topo import Topo
from mininet.cli import CLI
from mininet.util import dumpNodeConnections

__author__ = 'Carlos Giraldo'
__copyright__ = "Copyright 2014, AtlantTIC - University of Vigo"
```

```
__credits__ = ["Carlos Giraldo"]
__license__ = "GPL"
__version__ = "2.2.0"
__maintainer__ = "Carlos Giraldo"
__email__ = "carlos.giraldo@gti.uvigo.es"
__status__ = "Prototype"

class Minievents(Mininet):
    def __init__(self, topo=None, switch=UserSwitch, host=Host,
                 controller=RemoteController, link=Link, intf=Intf,
                 build=True, xterms=False, cleanup=False, ipBase='10.0.0.0/8',
                 inNamespace=False,
                 autoSetMacs=False, autoStaticArp=False, autoPinCpus=False,
                 listenPort=None, waitConnected=False, events_file=None):
        super(Minievents, self).__init__(topo=topo, switch=switch, host=host,
                                         controller=controller,
                                         link=link, intf=intf, build=build,
                                         xterms=xterms, cleanup=cleanup,
                                         ipBase=ipBase, inNamespace=inNamespace,
                                         autoSetMacs=autoSetMacs,
                                         autoStaticArp=autoStaticArp,
                                         autoPinCpus=autoPinCpus,
                                         listenPort=listenPort,
                                         waitConnected=waitConnected)
        self.scheduler = scheduler(time.time, time.sleep)
        print events_file
        if events_file:
            json_events = json.load(open(events_file))
            self.load_events(json_events)

    def load_events(self, json_events):
        # event type to function correspondence
        event_type_to_f = {'editLink': self.editLink, 'iperf': self.iperf,
                           'ping': self.ping, 'stop': self.stop}
        for event in json_events:
            debug("processing event: time {time}, type {type},
                        params {params}".format(**event))
            event_type = event['type']
            self.scheduler.enter(event['time'], 1, event_type_to_f[event_type]
                                 , kwargs=event['params'])

    # EVENT COMMANDS
    def delLink(self, src, dst):
        # TODO This code should be tested
        info('{time}:deleting link from {src} to {dst}'.format(time=time.time(),
                                                               src=src, dst=dst))
        n1, n2 = self.get(src, dst)
        intf_pairs = n1.connectionsTo(n2)
```

```
        for intf_pair in intf_pairs:
            n1_intf, n2_intf = intf_pair
            info('{time}:deleting link from {intf1} and {intf2}'.format(time=time.time()
                                                                        intf1=n1_intf.na
                                                                        intf2=n2_intf.na
            n1_intf.link.delete()
            self.links.remove(n1_intf.link)
            del n1.intfs[n1.ports[n1_intf]]
            del n1.ports[n1_intf]
            del n1.nameToIntf[n1_intf.name]

            n2_intf.delete()
            del n2.intfs[n2.ports[n2_intf]]
            del n2.ports[n2_intf]
            del n2.nameToIntf[n2_intf.name]

    def editLink(self, **kwargs):
        """
        Command to edit the properties of a link between src and dst.
        :param kwargs: named arguments
            src: name of the source node.
            dst: name of the destination node.
            bw: bandwidth in Mbps.
            loss: packet loss ratio percentage.
            delay: delay in ms.
        """
        n1, n2 = self.get(kwargs['src'], kwargs['dst'])
        intf_pairs = n1.connectionsTo(n2)
        info('***editLink event at t={time}: {args}'.format(time=time.time(), args=kwarg
        for intf_pair in intf_pairs:
            n1_intf, n2_intf = intf_pair
            n1_intf.config(**kwargs)
            n2_intf.config(**kwargs)


    def iperf(self, **kwargs):
        """
        Command to start a transfer between src and dst.
        :param kwargs: named arguments
            src: name of the source node.
            dst: name of the destination node.
            protocol: tcp or udp (default tcp).
            duration: duration of the transfert in seconds (default 10s).
            bw: for udp, bandwidth to send at in bits/sec (default 1 Mbit/sec)

        """
        kwargs.setdefault('protocol', 'TCP')
        kwargs.setdefault('duration', 10)
```

```
kwargs.setdefault('bw', 100000)
info('***iperf event at t={time}: {args}'.format(time=time.time(), args=kwargs))

if not os.path.exists("output"):
    os.makedirs("output")
server_output="output/iperf-{protocol}-server-{src}-{dst}.txt".format(**kwargs)
client_output="output/iperf-{protocol}-client-{src}-{dst}.txt".format(**kwargs)
info('output filenames: {client} {server}'.format(client=client_output,
                                                  server=server_output))

client, server = self.get(kwargs['src'], kwargs['dst'])
iperf_server_cmd = ''
iperf_client_cmd = ''
if kwargs['protocol'].upper() == 'UDP':
    iperf_server_cmd = 'iperf -u -s -i 1'
    iperf_client_cmd = 'iperf -u -t {duration} -c {server_ip} -b {bw}'
                                    .format(server_ip=server.IP(), **kwargs)


elif kwargs['protocol'].upper() == 'TCP':
    iperf_server_cmd = 'iperf -s -i 1'
    iperf_client_cmd = 'iperf -t {duration} -c {server_ip}'
                                    .format(server_ip=server.IP(), **kwargs)
else:
    raise Exception( 'Unexpected protocol:{protocol}'.format(**kwargs))

#server.sendCmd('{cmd} &>{output} &'
#                    .format(cmd=iperf_server_cmd, output=server_output))
info('iperf server command: {cmd} -s -i 1 &>{output} &'
                                    .format(cmd=iperf_server_cmd,
                                            output=server_output))
# This is a patch to allow sendingCmd while iperf is running in
# background.CONS: we can not know when
# iperf finishes and get their output
#server.waiting = False

if kwargs['protocol'].lower() == 'tcp':
    while 'Connected' not in client.cmd('sh -c "echo A | telnet -e A %s 5001"'
                                        %server.IP()):
        info('Waiting for iperf to start up...')
        sleep(.5)

info('iperf client command: {cmd} &>{output} &'.format(
    cmd = iperf_client_cmd, output=client_output))
client.sendCmd('{cmd} &>{output} &'.format(
    cmd = iperf_client_cmd, output=client_output))
# This is a patch to allow sendingCmd while iperf is running in background.CONS:
#                                         we can not know when
```

```
        # iperf finishes and get their output
        client.waiting = False

    def ping(self, **kwargs):
        """
        Command to send pings between src and dst.
        :param kwargs: named arguments
            src: name of the source node.
            dst: name of the destination node.
            interval: time between ping packet transmissions.
            count: number of ping packets.
        """
        kwargs.setdefault('count', 3)
        kwargs.setdefault('interval', 1.0)
        info('***ping event at t={time}: {args}'.format(time=time.time(), args=kwargs))

        if not os.path.exists("output"):
            os.makedirs("output")
        output = "output/ping-{src}-{dst}.txt".format(**kwargs)
        info('output filename: {output}'.format(output=output))

        src,dst=self.get(kwargs['src'], kwargs['dst'])
        ping_cmd='ping -c {count} -i {interval} {dst_ip}'.format(dst_ip=dst.IP(),**kwarg

        info('ping command: {cmd} &>{output} &'.format(
            cmd = ping_cmd, output=output))
        src.sendCmd('{cmd} &>{output} &'.format(
            cmd = ping_cmd, output=output))
        #This is a patch to allow sendingCmd while ping is running in background.CONS:
        #                                                    we can not know when
        #ping finishes and get its output
        src.waiting = False

    def start(self):
        super(Minievents, self).start()
        iperf_server_cmd = 'iperf -u -s -i 1'
        for h in self.hosts:
            server_output = "output/iperf-server-{}.txt".format(h.name)
            h.sendCmd('{cmd} &>{output} &'.format(cmd=iperf_server_cmd,
                                                  output=server_output))
            h.waiting = False
        CLI(self) if self.scheduler.empty() else self.scheduler.run()
'''

main_string = '''
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--events",default="./eventos.json",
```

```
                                        help="json file with event descriptions")
    args = parser.parse_args()
    setLogLevel('info')
    topo=GeneratedTopo()
    net=Minievents(topo=topo,autoSetMacs=True,link=TCLink,controller=RemoteController,
                   switch = UserSwitch, events_file=args.events)
    net.start()
'''
outputstring_to_be_exported = string_minievents +
                              outputstring_to_be_exported + main_string
# GENERATION FINISHED, WRITE STRING TO FILE
outputfile = ''
if output_file_name == '':
    output_file_name = input_file_name + '-generated-Mininet-Topo.py'
output_file_name = './topologias_creadas/'+output_file_name
outputfile = open(output_file_name, 'w')
outputfile.write(outputstring_to_be_exported)
outputfile.close()

print "Topology generation SUCCESSFUL!"
```

C. Anexo: Código python para generar las demandas

```
'''
Created on Oct 26, 2015

@author: ubuntu
'''
import simpy
import random
import json
from datetime import datetime
from Crypto.Random.random import randint

SIM_TIME = 1000
KB = 1000
MB = 1000*KB
MIN_t = 2
MAX_t = 10
MAX_bw = 10*MB
MIN_bw = MAX_bw//4
nombre_archivo = './listas_eventos_generados/eventos{}.json'.format(datetime.now())
def run():
    print 'run'
    net = ['h1','h4','h9']
    f = open(nombre_archivo, 'a')
    f.write('[\n')
    f.close()
    env = simpy.Environment()
    env.process(evt_wait(env, 5, net))
    env.run(until=SIM_TIME)
    jsonData = {
            "time": SIM_TIME+MAX_t+5,
            "type": "stop",
            "params": {
            }
        }
    f = open(nombre_archivo, 'a')
    f.write(json.dumps(jsonData, ensure_ascii=True, indent=3, encoding='UTF-8'))
    f.write('\n]')
```

```
    f.close()
    print 'termina de crear el archivo'

def evt_wait(env, t, net):
    print 'evt_wait'
    yield env.timeout(t)

    env.process(evt_iperf(env,net,id_ev=1,max_delay=20,min_t=5,max_t=SIM_TIME))
    env.process(evt_iperf(env,net,id_ev=2,max_delay=35,min_t=75,max_t=SIM_TIME-100))
    env.process(evt_iperf(env,net,id_ev=3,max_delay=30,min_t=125,max_t=160))
    env.process(evt_iperf(env,net,id_ev=4,max_delay=25,min_t=135,max_t=145))
    env.process(evt_iperf(env,net,id_ev=5,max_delay=20,min_t=190,max_t=230))
    env.process(evt_iperf(env,net,id_ev=6,max_delay=25,min_t=95,max_t=145))
    env.process(evt_iperf(env,net,id_ev=7,max_delay=20,min_t=101,max_t=145))
    env.process(evt_iperf(env,net,id_ev=8,max_delay=15,min_t=105,max_t=165))
    env.process(evt_iperf(env,net,id_ev=9,max_delay=20,min_t=110,max_t=180))
    env.process(evt_iperf(env,net,id_ev=10,max_delay=15,min_t=160,max_t=230))
    env.process(evt_iperf(env,net,id_ev=11,max_delay=30,min_t=170,max_t=215))
    env.process(evt_iperf(env,net,id_ev=12,max_delay=10,min_t=250,max_t=285))
    env.process(evt_iperf(env,net,id_ev=13,max_delay=30,min_t=265,max_t=280))
    env.process(evt_iperf(env,net,id_ev=2,max_delay=15,min_t=350,max_t=510))
    env.process(evt_iperf(env,net,id_ev=3,max_delay=20,min_t=325,max_t=360))
    env.process(evt_iperf(env,net,id_ev=4,max_delay=35,min_t=335,max_t=345))
    env.process(evt_iperf(env,net,id_ev=5,max_delay=40,min_t=390,max_t=450))
    env.process(evt_iperf(env,net,id_ev=6,max_delay=35,min_t=400,max_t=445))
    env.process(evt_iperf(env,net,id_ev=7,max_delay=20,min_t=401,max_t=445))
    env.process(evt_iperf(env,net,id_ev=8,max_delay=15,min_t=415,max_t=455))
    env.process(evt_iperf(env,net,id_ev=9,max_delay=10,min_t=420,max_t=460))
    env.process(evt_iperf(env,net,id_ev=10,max_delay=25,min_t=480,max_t=530))
    env.process(evt_iperf(env,net,id_ev=12,max_delay=15,min_t=470,max_t=545))
    env.process(evt_iperf(env,net,id_ev=13,max_delay=10,min_t=460,max_t=630))
    env.process(evt_iperf(env,net,id_ev=14,max_delay=25,min_t=560,max_t=650))
    env.process(evt_iperf(env,net,id_ev=15,max_delay=20,min_t=580,max_t=685))
    env.process(evt_iperf(env,net,id_ev=16,max_delay=20,min_t=620,max_t=675))
    env.process(evt_iperf(env,net,id_ev=17,max_delay=30,min_t=670,max_t=690))
    env.process(evt_iperf(env,net,id_ev=2,max_delay=15,min_t=650,max_t=720))
    env.process(evt_iperf(env,net,id_ev=3,max_delay=20,min_t=725,max_t=760))
    env.process(evt_iperf(env,net,id_ev=4,max_delay=35,min_t=735,max_t=795))
    env.process(evt_iperf(env,net,id_ev=5,max_delay=40,min_t=790,max_t=850))
    env.process(evt_iperf(env,net,id_ev=6,max_delay=35,min_t=800,max_t=855))
    env.process(evt_iperf(env,net,id_ev=7,max_delay=20,min_t=815,max_t=885))
    env.process(evt_iperf(env,net,id_ev=8,max_delay=15,min_t=805,max_t=845))
    env.process(evt_iperf(env,net,id_ev=9,max_delay=10,min_t=850,max_t=920))
    env.process(evt_iperf(env,net,id_ev=10,max_delay=25,min_t=880,max_t=930))
    env.process(evt_iperf(env,net,id_ev=12,max_delay=15,min_t=870,max_t=945))
    env.process(evt_iperf(env,net,id_ev=13,max_delay=10,min_t=860,max_t=930))
    env.process(evt_iperf(env,net,id_ev=14,max_delay=25,min_t=860,max_t=950))
    env.process(evt_iperf(env,net,id_ev=15,max_delay=20,min_t=880,max_t=985))
```

```
    env.process(evt_iperf(env,net,id_ev=16,max_delay=20,min_t=880,max_t=975))
    env.process(evt_iperf(env,net,id_ev=17,max_delay=30,min_t=970,max_t=990))

# import copy
def evt_iperf(env, hosts, id_ev, max_delay, min_t, max_t):
    #import datetime
    print 'evt_iperf',id_ev
    yield env.timeout(min_t)
    while env.now < max_t:
    # Esperar un tiempo para lanzar el iperf
        #yield env.timeout(2)
        yield env.timeout(random.random()*max_delay)
        tiempo = env.now
        server = hosts[random.randint(0, len(hosts) - 1)]
        #server = hosts[0]
        # Seleccionar el destino (aleatorio)
        client = server
        #client = hosts[1]
        while client == server:
            client = hosts[random.randint(0, len(hosts) - 1)]
        #dur = 1
        #tam = 2*MB
        dur = randint(MIN_t, MAX_t)
        tam = randint(MIN_bw, MAX_bw)
        jsonData = {
                    #'hora': str(datetime.datetime.now()),
                    #'evento':{
                            "time": tiempo,
                            "type": "iperf",
                            "params": {
                                "src": client,
                                "dst": server,
                                "protocol": "UDP",
                                "bw": tam,
                                "duration": dur
                                }
                            #}
                    }
        f = open(nombre_archivo, 'a')
        f.write(json.dumps(jsonData, ensure_ascii=True, indent=3, encoding='UTF-8'))
        # f.write(str(jsonData))
        f.write(',\n')
        f.close()
if __name__ == '__main__':
    run()
```

Bibliografía

[1] *eHealth Market Size & Share — Global Industry Report, 2022*

[2] Adibi, Sasan: Biomedical sensing analyzer (BSA) for mobile-health (mHealth)-LTE. En: *IEEE Journal of Biomedical and Health Informatics* 18 (2014), Nr. 1, p. 345–351. – ISBN 2168–2194

[3] Altukhov, V. ; Chemeritskiy, E.: On real-time delay monitoring in software-defined networks. En: *SDN and NFV: Modern Networking Technologies - 2014 International Science and Technology Conference "Modern Networking Technologies", MoNeTec 2014 - Proceedings* (2014), p. 0–5. ISBN 9781479975952

[4] Bari, Md. F. ; Chowdhury, Shihabur R. ; Ahmed, Reaz ; Boutaba, Raouf: PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks. En: *2013 IEEE SDN for Future Networks and Services (SDN4FNS)* (2013), nov, p. 1–7. ISBN 978–1–4799–2781–4

[5] Barroso, L. A. ; Hölzle, U.: The Case for Energy-Proportional Computing. En: *Computer* 40 (2007), Dec, Nr. 12, p. 33–37. – ISSN 0018–9162

[6] Bianzino, Aruna P. ; Chaudet, Claude ; Larroca, Federico ; Rossi, Dario ; Rougier, Jean-Louis: Energy-aware routing: A reality check. En: *2010 IEEE Globecom Workshops*, IEEE, dec 2010. – ISBN 978–1–4244–8863–6, p. 1422–1427

[7] Bueno, Iris ; Aznar, Jose I. ; Escalona, Eduard ; Ferrer, Javier ; Antoni Garcia-Espin, Joan: An opennaas based sdn framework for dynamic qos control. En: *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for* IEEE, 2013, p. 1–7

[8] Callon, R: RFC 1195, entitled Use of OSI ISIS for routing in TCP. En: *IP and Dual Environments* (1990), p. 1–80

[9] Cao, Jin ; Davis, Drew ; Vander Wiel, Scott ; Yu, Bin: Time-varying network tomography: router link data. En: *Journal of the American Statistical Association* 95 (2000), Nr. 452, p. 1063–1075

[10] Committee, O N F Market E. [u. a.]: Software-Defined Networking: The New Norm for Networks. En: *ONF White Paper. Palo Alto, US: Open Networking Foundation* (2012)

[11] Dreibholz, T ; Becke, M ; Adhari, H: Report to Congress on Server and Data Center Energy Efficiency Public Law 109-431. En: *Tdr.Wiwi.Uni-Due.De* 109 (2007), p. 431

[12] Egilmez, HE: *Adaptive Video Streaming over OpenFlow Networks with Quality of Service*, Koç University Graduate, Tesis de Grado, 2012

[13] Egilmez, Hilmi E. ; Civanlar, Seyhan ; Tekalp, a. M.: An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks. En: *IEEE Transactions on Multimedia* 15 (2013), apr, Nr. 3, p. 710–715. – ISSN 1520–9210

[14] Egilmez, Hilmi E. ; Dane, S T. ; Bagci, K T. ; Tekalp, A M.: OpenQoS : An OpenFlow Controller Design for Multimedia Delivery with End-to-End Quality of Service over Software-Defined Networks. En: *Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, 2012, p. 1–8

[15] Farinachi, D: Introduction to enhanced IGRP (EIGRP). En: *Cisco Systems Inc* (1993)

[16] Fernandes, Eder L.: *Software Switch 1.3: An experimenter-friendly OpenFlow implementation*, UNIVERSIDADE ESTADUAL DE CAMPINAS, Tesis de Grado, 2015

[17] Giroire, Frederic ; Moulierac, Joanna ; Phan, Truong K.: Optimizing rule placement in software-defined networks for energy-aware routing. En: *2014 IEEE Global Communications Conference, GLOBECOM 2014* (2014), p. 2523–2529. ISBN 9781479935116

[18] Goldschmidt, O: ISP backbone traffic inference methods to support traffic engineering. En: *Internet Statistics and Metrics Analysis (ISMA) Workshop*, 2000, p. 1063–1075

[19] Grossmann, Marcel ; Schuberth, Stephan J A.: Auto-Mininet : Assessing the Internet Topology Zoo in a Software-Defined Network Emulator. En: *MMBnet 2013* (2013)

[20] Gude, Natasha ; Pettit, Justin ; Pfaff, Ben ; Mckeown, Nick ; Shenker, Scott: NOX : Towards an Operating System for Networks. En: *ACM SIGCOMM Computer Communication Review* 38 (2008), Nr. 3, p. 105–110

[21] Hagberg, Aric A. ; Schult, Daniel A. ; Swart, Pieter J.: Exploring network structure, dynamics, and function using {NetworkX}. En: *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA USA, 2008, p. 11–15

[22] Hamad, Diyar J. ; Yalda, Khirota G. ; Okumus, Ibrahim T.: Getting traffic statistics from network devices in an SDN environment using OpenFlow.

[23] Heller, Brandon ; Seetharaman, Srinivasan ; Mahadevan, Priya ; Yiakoumis, Yiannis ; Sharma, Puneet ; Banerjee, Sujata ; Mckeown, Nick: ElasticTree: Saving Energy in Data Center Networks. En: *NSDI* Vol. 3, 2010, p. 19–21

[24] Jamaliannasrabadi, Saba: *High Performance Computing as a Service in the Cloud Using Software-Defined Networking*, Tesis de Grado, 2015

[25] Jeong, Kwangtae ; Kim, Jinwook ; Kim, Young-tak: QoS-aware Network Operating System for software defined networking with Generalized OpenFlows. En: *2012 IEEE Network Operations and Management Symposium* (2012), apr, p. 1167–1174. ISBN 978–1–4673–0269–2

[26] Jiang, Dingde ; Hu, Guangmin: A Novel Approach to Large-Scale IP Traffic Matrix Estimation Based on RBF Neural Network. En: *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, 2008, p. 1–4

[27] Jimenez, Sergio F. ; Chaure, Frederic R. ; Rincón Rivera, David: *Encaminament amb optimització de consum energètic en una Software-Defined Network*, Tesis de Grado, 2012. – 144 p.

[28] Johnson, Donald B.: Finding All the Elementary Circuits of a Directed Graph. En: *SIAM Journal on Computing* 4 (1975), Nr. 1, p. 77–84. – ISBN 0097–5397

[29] Keyoumarsi, Arman ; Rincón Rivera, David: *Distributed Traffic Matrix Measurement in OpenFlow Enabled Networks*, Tesis de Grado, 2014. – 115 p.

[30] Knight, Simon ; Nguyen, Hung X. ; Falkner, Nickolas ; Bowden, Rhys ; Roughan, Matthew. *The internet topology zoo.* 2011

[31] Kokku, Ravi ; Mahindra, Rajesh ; Zhang, Honghai ; Rangarajan, Sampath: NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks. En: *IEEE/ACM Transactions on Networking* 20 (2012), 10, Nr. 5, p. 1333–1346

[32] Lantz, Bob ; Heller, Brandon ; McKeown, Nick: A network in a laptop: rapid prototyping for software-defined networks. En: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* ACM, 2010, p. 19

[33] Lara, Adrian ; Kolasani, Anisha ; Ramamurthy, Byrav: Network innovation using openflow: A survey. En: *Communications Surveys & Tutorials, IEEE* 16 (2014), Nr. 1, p. 493–512

[34] Malkin, G: RFC 2453: Rip version 2. En: *Request for Comments* 2453 (1998)

[35] Malkin, Gary ; Minnear, R: Ripng for ipv6 / RFC 2080, January. 1997. – Informe de Investigación

[36] Matloff, Norm S (University of California, Davis): Introduction to discrete-event simulation and the simpy language. En: *Davis, CA. Dept of Computer Science. University* (2008), p. 1–33

[37] Mckeown, Nick ; Anderson, Tom ; Balakrishnan, Hari ; Parulkar, Guru ; Peterson, Larry ; Rexford, Jennifer ; Shenker, Scott ; Turner, Jonathan: OpenFlow: Enabling Innovation in Campus Networks. En: *ACM SIGCOMM Computer Communication Review* 38 (2008), Nr. 2, p. 69–74

[38] Montoya, Emanuel F. ; Botero, Juan F.: On achieving energy efficient minimum delay paths in Software Defined Networks. En: *Communications and Computing (COLCOM), 2016 IEEE Colombian Conference on* IEEE, 2016. – to appear

[39] Moy, John: rfc 2328: Ospf version 2. En: *Internet Society (ISOC)* (1998), p. l1

[40] Nunes, Bruno Astuto a. ; Mendonca, Marc ; Nguyen, Xuan-Nam ; Obraczka, Katia ; Turletti, Thierry: A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. En: *Communications Surveys & Tutorials, IEEE* PP (2014), Nr. 99, p. 1–18

[41] Omidvar, A ; Shahhoseini, HS: Intelligent IP traffic matrix estimation by neural network and genetic algorithm. En: *Intelligent Signal Processing (WISP), 2011 IEEE 7th International Symposium on* IEEE, 2011, p. 1–6

[42] Optimization, Gurobi [u. a.]: Gurobi optimizer reference manual. En: *URL: http://www. gurobi. com* (2012)

[43] Peterson, Larry L. ; Davie, Bruce S.: *Computer networks: a systems approach.* Elsevier, 2007

[44] Pfaff, Ben ; Lantz, B ; Heller, B ; Others: Openflow switch specification, version 1.3. 0. En: *Open Networking Foundation* 0 (2012)

[45] Pfaff, Ben ; Pettit, Justin ; Koponen, Teemu ; Jackson, Ethan ; Zhou, Andy ; Rajahalme, Jarno ; Gross, Jesse ; Wang, Alex ; Stringer, Joe ; Shelar, Pravin [u. a.]: The design and implementation of open vswitch. En: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, p. 117–130

[46] Phemius, Kévin ; Bouet, Mathieu: Monitoring latency with OpenFlow. En: *2013 9th International Conference on Network and Service Management, CNSM 2013 and its three collocated Workshops - ICQT 2013, SVM 2013 and SETM 2013* (2013), p. 122–125. ISBN 9783901882531

[47] Prete, Luca ; Farina, Fabio ; Campanella, Mauro ; Biancini, Andrea: Energy efficient minimum spanning tree in OpenFlow networks. En: *Proceedings - European Workshop on Software Defined Networks, EWSDN 2012* (2012), p. 36–41. ISBN 9780769548708

[48] Qiao, Yan ; Hu, Zhiming ; Luo, Jun: Efficient traffic matrix estimation for data center networks. En: *IFIP Networking Conference, 2013*, 2013, p. 1–9

[49] Rekhter, Y ; Li, T ; Hares, S: RFC 4271. En: *Internet Engineering Task Force, http://www. rfc-editor. org/rfc/rfc4271. txt, access on* 6 (2014)

[50] Rutgers, Charles L H.: "An introduction to igrp. En: *The State University of New Jersey, Center for Computers and Information Services, Laboratory for Computer Science Research* (1991)

[51] Skorin-Kapov, Lea ; Matijasevic, Maja: Analysis of QoS Requirements for e-Health Services and Mapping to Evolved Packet System QoS Classes. En: *International journal of telemedicine and applications* 2010 (2010), jan, p. 628086. – ISSN 1687–6423

[52] Solé Orrit, Adriá ; Rincón Rivera, David ; Chaure, Frederic R.: *Distributed Traffic Matrix Measurement in Software-Defined Networks*, Tesis de Grado, 9 2013. – 102 p.

[53] Takeda, Tomonori ; Shiomoto, Kohei: Traffic matrix estimation in large-scale IP networks. En: *2010 17th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)* (2010), 5, Nr. 1, p. 1–6

[54] Tebaldi, Claudia ; West, Mike: Bayesian inference on network traffic using link count data. En: *Journal of the American Statistical Association* 93 (1998), Nr. 442, p. 557–573

[55] Tootoonchian, Amin ; Ghobadi, Monia ; Ganjali, Yashar: OpenTM: traffic matrix estimator for OpenFlow networks. En: *Passive and active measurement* Springer, 2010, p. 201–210

[56] Tucker, Rs: Modelling Energy Consumption in IP Networks. En: *Cisco Green Research Symposium* (2008)

[57] Van Adrichem, Niels L M. ; Doerr, Christian ; Kuipers, Fernando A.: OpenNetMon: Network monitoring in OpenFlow software-defined networks. En: *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014

[58] Vardi, Yehuda: Network tomography: Estimating source-destination traffic intensities from link data. En: *Journal of the American Statistical Association* 91 (1996), Nr. 433, p. 365–377

[59] Vereecken, W. ; Heddeghem, W. V. ; Deruyck, M. ; Puype, B. ; Lannoo, B. ; Joseph, W. ; Colle, D. ; Martens, L. ; Demeester, P.: Power consumption in telecommunication networks: overview and reduction strategies. En: *IEEE Communications Magazine* 49 (2011), June, Nr. 6, p. 62–69. – ISSN 0163–6804

[60] Zhang, Yin ; Roughan, Matthew ; Duffield, Nick ; Greenberg, Albert: Fast accurate computation of large-scale IP traffic matrices from link loads. En: *ACM SIGMETRICS Performance Evaluation Review* Vol. 31 ACM, 2003, p. 206–217