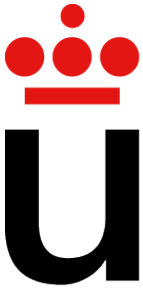


Universidad
Rey Juan Carlos

GRADO EN INGENIERIA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Curso Académico 2020/2021

Trabajo Fin de Grado

VISUALIZACIÓN EN 3D DE LA TOPOLOGÍA DE REDES SDN UTILIZANDO A-FRAME

Autor : Alberto Abades Hoyas

Tutor : Dr. Pedro de las Heras Quirós

Trabajo Fin de Grado

Visualización en 3D de la Topología de Redes SDN Utilizando A-Frame

Autor : Alberto Abades Hoyas

Tutor : Dr. Pedro de las Heras Quirós

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2021, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2021

Dedicado a

mis padres, mi hermano y Laia

II

Agradecimientos

En primer lugar quiero dar las gracias a mis padres, Juan Luis y Blanca, que siempre han hecho todos los sacrificios y esfuerzos posibles para que mi única preocupación fuera la de formarme lo mejor posible y tratar de conseguir todas mis metas. Sin sus esfuerzos no habría llegado a terminar el grado ni hubiera podido centrarme en encontrar un buen trabajo para poder desarrollar mi proyecto de vida.

También tengo que darle las gracias a mi hermano Jorge `aka Chapu`, que siempre ha estado apoyándome fuera cual fuera el ámbito, durante mi formación, mi vida laboral, mi vida deportiva..., siempre ha estado ahí empujándome cuando parecía que iba a desfallecer.

Y a Laia, mi perrita, que aunque ella no lo sepa ha sido fundamental para conseguir ciertos objetivos a lo largo del grado, sobre todo en lo referente a desarrollo de aplicaciones o resolución de problemas. Cada vez que me atascaba le ponía la correa cogía la pelota y me iba a pasear con ella para despejarme y pensar qué estaba haciendo que me atascara. Hemos hecho muchos kilómetros juntos.

Por supuesto tengo mucho que agradecer al Dr. Pedro de las Heras Quirós que ha entendido a la perfección mis circunstancias y que siempre ha estado disponible para cualquier duda. Además siempre me ha dado todas las facilidades posibles para poder llevar a cabo este proyecto en los pocos meses que he tenido para dedicarle puesto que el tiempo apremiaba y la cuenta atrás estaba llegando a su fin.

Quería hacer una pequeña mención al Dr. Jesús M. González Barahona, que a pesar de no estar relacionado con mi proyecto siempre ha estado disponible para ayudarme con la tecnología A-Frame y me ha dado buenos consejos para seguir adelante cuando me topaba con un muro.

Resumen

Con este proyecto se trata de crear una herramienta para poder representar en el navegador redes en 3D y Realidad Virtual. La idea es poder representar cualquier tipo de red, no sólo redes predeterminadas. El objetivo principal es hacer más amigable la representación que podemos hacer actualmente con el visualizador del controlador de redes definidas por software RYU.

Para llevar a cabo la consecución de los objetivos se ha utilizado el framework A-Frame, que es una herramienta muy poderosa que combina desarrollo en HTML y JavaScript para poder modificar el DOM. Además es una herramienta sencilla para poder representar en el navegador escenarios 3D o de Realidad Virtual.

Para poder generar la representación de una red compleja se ha partido de la creación de elementos sencillos por separado para entender cómo se manejaban y qué se podía hacer con ellos para terminar juntándolos todos en una misma escena y llegar a la representación que se tenía como objetivo.

Para poder generar las redes que se van a representar se ha utilizado Mininet para arrancar y generar las redes y el controlador RYU con el que se ha integrado el desarrollo para poder obtener la información de las redes que se van arrancando en Mininet.

Una vez que se ha integrado el código HTML con el controlador RYU se ha podido comprobar con cinco redes diferentes (dos redes lineales, dos redes con una jerarquía de árbol y una red con todos los elementos conectados entre sí) que se representaban todas correctamente en 3D en el navegador y por tanto se ha conseguido alcanzar el objetivo principal que tenía este proyecto.

Summary

This project is about creating a tool to represent 3D networks and Virtual Reality in the browser. The idea is to be able to represent any type of network, not only predetermined networks. The main objective is to make more user friendly the representation that we can currently do with the RYU software defined network controller viewer.

To achieve the objectives, the A-Frame framework has been used, which is a very powerful tool that combines HTML and JavaScript development to be able to modify the DOM. It is also a simple tool for representing 3D or Virtual Reality scenarios in the browser.

In order to generate the representation of a complex network, we started with the creation of simple elements separately to understand how they were handled and what could be done with them to end up putting them all together in the same scene and arrive at the representation that was the objective.

In order to generate the networks to be represented, Mininet has been used to start and generate the networks and the RYU controller has been used to integrate the development to obtain the information of the networks that are being started in Mininet.

Once the HTML code has been integrated with the RYU controller, it has been possible to verify with five different networks (two linear networks, two networks with a tree hierarchy and one network with all elements connected to each other) that they were all correctly represented in 3D in the browser and therefore the main objective of this project has been achieved.

Índice general

1. Introducción **1**

1.1. Contexto histórico 1

1.2. Visualización de redes 2

1.3. Estructura de la memoria 6

2. Objetivos **9**

2.1. Objetivo general 9

2.2. Objetivos específicos 9

2.3. Planificación temporal 10

3. Estado del Arte **13**

3.1. HTML5 13

3.2. JavaScript 14

3.3. A-Frame 14

3.4. Three.js 16

3.5. WebGL 17

3.6. WebXR 17

3.7. SDN 18

3.8. Mininet 18

3.9. RYU 20

4. Diseño e implementación **21**

4.1. Visualización de una escena 21

4.2. Generación de entidades desde JavaScript 24

4.3. Incluir enlaces entre las cajas 29

4.4. Lectura de un archivo JSON 32
4.5. Arrancar una red SDN en la máquina virtual 34
4.6. Análisis de los archivos JSON 38
4.7. Generación de la escena a partir de los archivos JSON descargados 40
4.7.1. Guardar la información de los switches y enlaces 42
4.7.2. Generar las coordenadas de cada switch 42
4.7.3. Generar los switches e incluirlos en la escena 43
4.7.4. Generar los enlaces e incluirlos en la escena 45
4.8. Un archivo HTML para cada SDN 46
4.9. Archivo HTML para integrar con Mininet 48

5. Experimentos y validación **51**

6. Conclusiones **55**
6.1. Consecución de objetivos 55
6.2. Aplicación de lo aprendido 55
6.3. Lecciones aprendidas 56
6.4. Trabajos futuros 57

A. Manual de usuario **59**

Bibliografía **61**

Índice de figuras

2.1. Diagrama de Gantt con la planificación temporal 11

3.1. Diagrama de una aplicación con Three.js 17
3.2. Topología vista desde el visualizador de RYU 20

4.1. Diseño de la red que vamos a representar 22
4.2. Cabecera del archivo HTML 22
4.3. Código para generar la malla del switch S1 23
4.4. Visualización de la red desde el inspector 25
4.5. Visualización de la red desde el navegador 25
4.6. Visualización de la red desde el navegador 26
4.7. De esta manera se define el esquema que va a tener el componente 27
4.8. Inicialización de las variables al iniciar el componente 27
4.9. Lógica para generar las diferentes cajas de la escena 28
4.10. Visualización de las cajas en el navegador 29
4.11. Esquema del componente cilindro 29
4.12. Datos de cada uno de los cilindros que unirán las cajas 30
4.13. Lógica para generar las diferentes cilindros de la escena 31
4.14. Visualización de la red completa 32
4.15. Información que contiene el archivo JSON que vamos a recuperar 32
4.16. Función para recuperar un archivo JSON y convertirlo a un objeto de JavaScript 33
4.17. Archivo HTML que usamos para cargar un archivo JSON 34
4.18. Salida de la consola - Representación del archivo JSON 35
4.19. Red tree,depth=3 vista desde el visualizador de RYU 36

4.20. Red tree,depth=4 vista desde el visualizador de RYU 36
4.21. Red linear,4 vista desde el visualizador de RYU 37
4.22. Red linear,8 vista desde el visualizador de RYU 37
4.23. Red torus,3,3 vista desde el visualizador de RYU 37
4.24. Función para recuperar un archivo JSON del controlador 38
4.25. Archivo JSON de la topología linear,4 39
4.26. Esquema del componente para crear la red 40
4.27. Esquema del componente para crear la red 41
4.28. Completar los arrays con la información del JSON 43
4.29. Generar las coordenadas de cada switch 43
4.30. Configuración de los atributos de cada switch 44
4.31. Configuración de los atributos de cada enlace 46
4.32. HTML para representar la SDN tree,depth=3 47
4.33. HTML para representar la SDN tree,depth=4 47
4.34. HTML para representar la SDN linear,4 47
4.35. HTML para representar la SDN linear,8 48
4.36. HTML para representar la SDN torus,3,3 48
4.37. HTML genérico para representar cualquier SDN 49

5.1. Representación de una red lineal con 4 switches 52
5.2. Representación de una red lineal con 8 switches 52
5.3. Representación de una red torus 3x3 52
5.4. Representación de una red con forma de árbol con 3 niveles 53
5.5. Representación de una red con forma de árbol con 4 niveles 53
5.6. Red tree,depth=4 vista desde el visualizador de RYU 53
5.7. Red tree,depth=4 vista en 3D 53
5.8. Red tree,depth=4 vista en 3D desde otra perspectiva 54
5.9. Aproximación a un elemento de la Red tree,depth=4 54

Capítulo 1

Introducción

En este proyecto vamos representar una red en 3D sobre el navegador. Para ello se hará un tratamiento de un archivo JSON con la información de una red arrancada en Mininet.

Se integrará la tecnología A-Frame[9], que nos permitirá visualizar cualquier red en el navegador, con la tecnología Mininet a través del controlador RYU que nos permitirá generar diferentes redes virtuales para ser representadas.

1.1. Contexto histórico

El primer intento de generar un entorno de realidad virtual de manera analógica fue en 1836[2] con el invento del estereoscopio, el cual permitía visualizar dos imágenes prácticamente iguales de manera independiente para cada ojo, de esta manera era el cerebro el que generaba una sensación de profundidad al combinarlas.

Esto se replicaría en 1939 con la creación del View-Master basado en los mismos principios que el estereoscopio.

En 1957 el cineasta Morton Heilig ideó el sensorama, que no era más que una gran cabina en la que combinar imágenes estereoscópicas con sonido estéreo real, nunca llegó a ver la luz pero sus principios asentaron las bases de lo que a día de hoy conocemos como realidad virtual.

En 1968, el dispositivo conocido como La espada de Damocles adaptaba la perspectiva de las imágenes en función de la posición y el movimiento de la cabeza, además como las imágenes se superponían sobre un fondo real, podemos decir que fue el primer intento de realidad aumentada. Debido al gran tamaño del dispositivo no pudo ser comercializado.

La realidad virtual ha sido muy importante en campos como el de la aviación, en 1929 se creó para entrenamiento militar el Blue box que simulaba condiciones meteorológicas y reaccionaba a las órdenes del piloto y en 1980 se creó el simulador de vuelo Super Cockpit, que permitía al piloto controlar el avión con gestos, palabras e incluso movimientos oculares. Además proyectaba datos aeronáuticos en tiempo real sobre un espacio tridimensional

En 1986 la NASA mostró al mundo las primera gafas de realidad virtual, estas además tenían control por voz y llevaban un traje y unos guantes con sensores para poder capturar y procesar todos los gestos que se pudieran hacer.

A principios de los años 90 la realidad virtual llegó al mundo de los videojuegos con diferente éxito. Sega creó el prototipo de la consola Géneis, que se trataba de unas gafas de realidad virtual con auriculares estéreo y sensores que detectaban la posición de la cabeza, tras varios estudios Sega decidió que no se comercializaría. Mientras tanto Nintendo creó Virtual Boy, unas gafas que mostraban imágenes en 3D pero únicamente con tonos rojos y negros. Fue un fracaso comercial.

20 años después de los intentos de Sega y Nintendo y hasta la actualidad, la realidad virtual ha sido foco de las principales empresas tecnológicas de entretenimiento, llevando al tecnología hasta límites insospechados. En estos momentos en el mercado podemos encontrarnos gafas de realidad virtual de todo tipo y a diferentes precios, aunque por lo general son accesibles para cualquier público.

Gafas como Oculus Rift, PlayStation VR, Microsoft HoloLens, Google Cardboard son algunos ejemplos de lo que podemos encontrar en el mercado orientadas principalmente a la inmersión en los videojuegos.

Cada vez más dispositivos móviles están empezando a soportar la realidad aumentada, que no es más que un tipo de realidad virtual en el que se generan objetos en 3D sobre un fondo real (una mesa, el suelo) en lugar de sobre un fondo renderizado y definido previamente.

1.2. Visualización de redes

La visualización del tráfico de redes facilitan la comprensión de los patrones y gracias a ello se puede agilizar la detección del tráfico malicioso. Por lo general las herramientas suelen utilizar visualizaciones basadas en trazados para detectar esos patrones pero no deja de ser una

información incompleta. NetVis[16] permite diferentes perspectivas a la vez de datos en tiempo real, lo que proporciona una visión más completa de la red. Como está diseñado para que la información sea modular se puede analizar de manera general toda la actividad de la red o puede centrarse en un subconjunto de ella. NetVis se puede utilizar para detectar de manera sencilla actividades inusuales en la red.

NetVis es una aplicación Java que proporciona una interfaz de usuario. Se ha tratado de diseñar de manera modular para que sea más fácil ampliar su funcionalidad y mantenimiento. Para obtener información de la red y de su tráfico se integra con el analizador de paquetes Wireshark.

La característica principal es el control del tiempo, pudiendo el usuario ralentizar o acelerar la velocidad de los paquetes, esto hace que resulte más sencillo analizar con más detalle los intervalos de tiempo críticos. NetVis incluye un sistema de filtrado que maneja dos tipos de filtros, los que hace el usuario y los que se aplican en las propias visualizaciones. Los datos de entrada filtrados se envían al controlador que proporciona las visualizaciones y la interfaz de usuario.

Para analizar los atributos de los paquetes de la red éstos se procesan de manera normalizada, se mapean los valores de los diferentes atributos y se llevan al intervalo [0,1] para facilitar su comprensión. Cada clase normalizadora tiene su propio filtro.

The Spinning Cube of Potential Doom[4] es la visualización del tráfico de red analizado por el sistema de detección de intrusiones Bro. Este sistema de detección analiza todo el tráfico de una red supervisando los enlaces en busca de tráfico que viole las políticas de acceso o de uso. Este tipo de visualización hace comprensible el análisis del nivel de tráfico malicioso incluso para los que no tienen experiencia con este tipo de análisis.

Esta visualización registra todas los intentos de conexión TCP, conseguidos y fallidos, y los distribuye a lo largo de los tres ejes. En el eje X se representan las direcciones IP locales, el eje Y representa los diferentes números de puerto y el eje Z representa las direcciones IP globales.

Los registros de intentos fallidos se pueden interpretar como intentos de acceso maliciosos, aunque algunos intentos fallidos pueden deberse a caídas en las conexiones o en las máquinas, se ha podido demostrar que ese tipo de intentos fallidos se registran muy pocas veces como intentos realmente maliciosos, por lo que el algoritmo descarta correctamente la mayoría de estos errores.

Los patrones que se visualizan son muy reconocibles para detectar el tipo de ataque que se está produciendo. Los patrones lineales indican un barrido a lo largo de todos los puertos y las direcciones IP e incluso sobre un mismo Host en busca de algún puerto que esté escuchando. Otro patrón que muy identificable es el helicoidal que va variando los números de puerto y las IP para tratar de esquivar los detectores, pero al visualizarlo de esta manera queda resaltado como un patrón. Hay otro patrón que se conoce como cortadora de césped, éste surge cuando se atacan puertos contiguos mientras va revisando todas las direcciones IP locales.

Otra forma de visualización de las redes y su tráfico es la combinación de representación 3D y 2D[5], la representación 3D ofrece una visión de la comunicación entre varias mallas de la red y la representación 2D da una vista detallada de algunas partes específicas de la representación 3D. El mayor problema de este tipo de análisis es la cantidad de datos que hay que recoger, esto se soluciona con sistemas automatizados tipo IA o análisis estadísticos. Aunque también se pueden utilizar técnicas de visualización, principalmente 2D, para estos análisis.

La visualización 3D se puede representar como cubos texturizados, las texturas varían en función de la actividad, el tráfico se muestra como enlaces entre los cubos y el color representar el atributo seleccionado. Se combina con una visualización 2D que representan una zona de la red seleccionada y los hosts y el tráfico de red asociado.

Otro de los objetivos de esta combinación de visualización 3D y visualización 2D es tratar de hacerlo más escalable y con mayor usabilidad.

Muchas organizaciones dependen de una buena monitorización para tomar decisiones en diferentes ámbitos, desde la medicina hasta la ingeniería o el ejército[19]. Nuestros ojos están preparados por la evolución para detectar movimiento y cambios en el entorno la visualización de las redes es facilita mucho la labor de analizar información altamente dinámica.

Hay muchas herramientas de visualización pero no todas proporcionan la información suficiente para el administrador de red. Para analizar las redes y encontrar posibles problemas o ataques se hacen dos tipos de monitorización o recogida de datos. Uno es la recogida de datos en tiempo real y la otra es la recogida de datos históricos. Para la recogida de datos en tiempo real lo ideal es que la herramienta pueda indicar cuándo se produce un problema, ya sea provocado o no. Los datos históricos suelen ser estadísticas que dan una imagen de cómo ha sido el rendimiento y el funcionamiento a lo largo del tiempo.

Las herramientas de visualización puede acelerar el proceso de solución de incidencias,

pero nos encontramos un problema de escalabilidad cuando la topología de las redes es muy grande ya que tenemos que poner más dispositivos de monitorización La recogida de datos es sencilla pero su análisis se complica y necesitamos un hardaware lo suficientemente potente para recoger datos e interpretarlos para su posterior visualización. La mayoría de aplicaciones de visualización se centra en su mayoría en la seguridad de las redes.

El diagrama de dispersión 3D, el gráfico de líneas o el gráfico de barras son algunas de las técnicas que dan una imagen más general y detallada de las direcciones IP y de los servicios. La visualización de análisis de redes muestra los diferentes con colores textuales que se basan en diferentes servicios, puertos y direcciones IP. Pero este método tiene el problema de la transferencia de grandes datos a un entorno virtual que genere análisis fiables.

VISUAL[19]. es una herramienta que permite examinar el tráfico entre hosts internos de una red y hosts externos. Esto lo hace dividiendo el espacio en direcciones IP locales y direcciones IP globales. Esto genera una visión rápida de los patrones de comunicación

Gigascope[19].ha incluido una interfaz SQL para facilitar la interpretación del flujo de datos. Esto hace que sea mucho más sencillo optimizar los datos obtenidos.

SeeNet y SeeNet3D[19]. se utilizan para interpretar representaciones en 2D y 3D respectivamente. Estas herramientas nos permiten utilizar zoom, vista panorámica y rotaciones.

Cichlid[19]. es una herramienta de visualización de cliente-servidor que genera una representación 3D de los datos de un red en tiempo real.

CyberNet[19]. es una herramienta implementada en Java, VRML y CORBA que presenta la gestión de la red en un entorno de Realidad Virtual. Tiene un paso de recopilación de datos que se encuentran en toda la red, estos datos se recogen a partir de unos parámetros preestablecidos. Una vez recogidos los datos se envían para iniciar el siguiente paso, la estructuración de datos. Este paso analiza los datos y los reorganiza de manera que sean más útiles y se vuelcan a unos archivos para presentarlos. Después se envían al último paso, la representación en la Realidad Virtual. Una vez representados se facilita su uso pudiendo agrupar los datos según diferentes criterios o niveles de detalle.

Time-tunnel[17] es una herramienta de visualización de datos multidimensional y combinado con Coordenadas Paralelas proporciona una visualización de datos de red debido a que los datos de los paquetes IP poseen muchos atributos que pueden ser analizados, esos datos son representados gracias a la integración de Coordenadas Paralelas.

La herramienta PCTT se puede combinar con la visualización 2Dto2D que muestra múltiples líneas que representan dos atributos dibujados desde un plano a otro con otros dos atributos en un espacio 3D. La visualización 2Dto2D se suele utilizar para detectar patrones de acceso, para ello los atributos que se extraen de los paquetes IP son las direcciones origen y destino y los puertos origen y destino.

Esta combinación de PCTT con 2Dto2D hace que en un espacio de Realidad Virtual se puedan superponer los diferentes planos en los que se representan los atributos y así poder identificar de manera sencilla las similitudes y diferencias. Time-Tunnel sólo muestra un plano por cada dato, por lo que para llegar a esta representación de superposición se deberían generar tantos planos como registros se quieran analizar. Para evitar esto se combina con Coordenadas Paralelas para que se puedan representar múltiples datos en el mismo plano.

La Agrupación de Equivalencia Estructural (SEG)[18] condensa el gráfico más de 20 veces conservando la información crítica de la red, para ello se basa en la naturaleza intrínseca del tráfico de red. SEG admite tráfico de hasta un millón de nodos. Los gráficos que admite pueden ser de tráfico no dirigido, dirigido y ponderado.

Escalar la visualización gráfica de un gran número de hosts y patrones de conexión es un reto. Los métodos de dibujo dirigido por fuerza no suelen representar de manera correcta los datos en tiempo real para redes con más de cien nodos. También es un problema calcular diseños para gráficos grandes porque se genera un desorden visual con el cruce de enlaces y por último los algoritmos de detección de comunidades no funcionan correctamente en los gráficos de tráfico de la red. Cuando se agrupan los los hosts se pierden detalles del contexto y de la topología.

1.3. Estructura de la memoria

La estructura de la memoria es la siguiente:

- En el primer capítulo se hace una introducción al proyecto y un acercamiento histórico a la Realidad Virtual y a la visualización de redes.

- En el capítulo 2 planteo los objetivos generales y los específicos que se quieren alcanzar con este proyecto. Así como la planificación temporal que ha llevado su desarrollo.

- En el capítulo 3 hago un acercamiento a las diferentes tecnologías que van a estar involucradas en el proyecto.
- En el capítulo 4 hago un recorrido sobre la evolución y los pasos que he seguido desde el comienzo del proyecto hasta conseguir el objetivo final.
- En el capítulo 5 muestro cómo se visualizan en 3D sobre el navegador las diferentes redes que hemos generado en Mininet.
- En el capítulo 6 indico cuáles han sido los objetivos alcanzados, así como posibles evoluciones que se podrían hacer sobre el proyecto para llevar la experiencia un paso más allá.

Capítulo 2

Objetivos

2.1. Objetivo general

Mi trabajo de fin de grado tiene como objetivo crear una manera amigable de visualizar redes en 3D, ya sea desde el navegador o con algún dispositivo de realidad virtual.

Para ello hay que analizar la información de una red generada con Mininet y representarla utilizando la tecnología A-Frame

2.2. Objetivos específicos

Los objetivos específicos que se han alcanzado con este trabajo son:

- La red se tiene que poder visualizar en 3D sobre el navegador, dispositivos móviles o dispositivos de realidad virtual.
- Para generar la red en 3D se utilizará la tecnología A-Frame
- Visualizar cualquier tipo de red generada en Mininet de manera dinámica
- Para visualizar cualquier red se obtendrá su información a través de un fichero JSON, que será tratado para extraer la información principal.
- Aprender tanto A-Frame como Three.js. A-Frame es una tecnología de alto nivel que está construida sobre Three.js, por lo que habrá comprender el funcionamiento de ambas para su correcto uso.

2.3. Planificación temporal

La primera toma de contacto con el Dr. Pedro de las Heras Quirós para que me indicara y explicara sus ideas sobre lo que se podía hacer para abordar este trabajo fueron a principios del mes de Marzo de 2021, desde ese momento y debido a mi jornada laboral empecé a dedicar entre 2 y 3 horas por las tardes de lunes a viernes pudiendo subir ese tiempo hasta las 4 o 5 horas al día los fines de semana, todo ello para documentarme sobre lo que era la tecnología A-Frame y cómo me podía ayudar a conseguir mis objetivos.

Tras documentarme aproximadamente durante un mes, ya que las tecnologías que iba a usar eran desconocidas para mí, comencé a escribir pequeños ejemplos de código para familiarizarme con el manejo del DOM, por lo que generaba pequeñas escenas en las que sólo había un elemento y trabajaba sobre él, una vez familiarizado con el manejo de DOM generé una escena estática sobre código HTML para comprobar que me estaba encaminando hacia el objetivo del trabajo. Esto me llevó aproximadamente unas tres semanas bajo las mismas condiciones horarias que el período de documentación.

Una vez confirmado que el objetivo a alcanzar era el deseado, comencé a llevar el código más allá del HTML, es decir, esos mismos objetos que he generado y los he manejado en DOM los he llevado al lenguaje Javascript para poder trabajar con ellos de una manera más dinámica, a pesar de encontrarme con ciertos problemas en este punto pude superarlos gracias a los consejos de los profesores que me han ayudado. El tiempo que tardé en superarlos y generar una escena directamente desde Javascript fue de cuatro semanas aproximadamente.

Una vez conseguido este hito tuve que pasar al siguiente punto, lectura y tratamiento de un archivo JSON, tanto de manera estática con un archivo definido previamente como de manera dinámica en la integración con la tecnología Mininet. Tras conseguir leer de manera satisfactoria el JSON y extraer únicamente los datos que necesitaba, pasé a integrarlo con la generación y manejo de objetos en el DOM. Dejando de este modo la foto final del objetivo que buscaba con este trabajo. Esta parte me llevó aproximadamente dos semanas superarla

Tras tener desarrollado todo el código me dediqué en exclusiva a escribir la memoria, dedicándole una media de 4 horas al día a lo largo de, aproximadamente, un mes.

En la figura 2.1 se puede ver un Diagrama de Gantt con la planificación temporal aproximada.

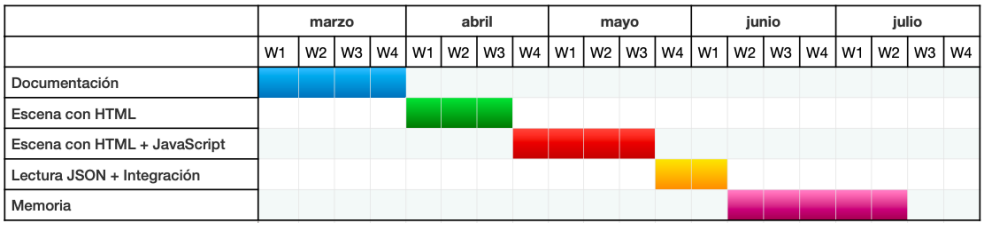


Figura 2.1: Diagrama de Gantt con la planificación temporal

Capítulo 3

Estado del Arte

3.1. HTML5

HTML5[20] es la versión 5 del lenguaje que se utiliza para crear páginas web. Tiene dos elementos principales que son la cabecera y el cuerpo. HTML utiliza etiquetas para definir la estructura como una jerarquía de árbol. Para acceder a esas etiquetas y poder modificar la estructura se suele utilizar el lenguaje JavaScript que nos permite modificar el DOM de manera sencilla. Para modificar el aspecto y darle formato se utilizan elementos CSS.

Algunas novedades de HTML5 con respecto a HTML4 son las siguientes:

- Se incluyen nuevas etiquetas para poder manejar contenidos multimedia como audio o video.
- Nuevas etiquetas para manejar grandes conjuntos de datos, estas etiquetas nos permite generar tablas dinámicas en se puede ordenar y filtrar el contenido de las diferentes columnas que pudiera haber.
- Se mejoran los formularios para poder validar los datos introducidos sin necesidad de acceder al código JavaScript
- Visores para funciones matemáticas y gráficos vectoriales, además se pueden interpretar otros lenguajes XML.
- Nueva funcionalidad para arrastrar objetos sobre la página web.

También se han incluido diferentes APIs para facilitar ciertas funcionalidades como la geolocalización, trabajo off-line, WebSockets y WebWorkers

3.2. JavaScript

JavaScript[6] se conoce como un lenguaje de scripting para páginas web. Es un lenguaje basado en prototipos y admite estilos de programación orientado a objetos, imperativos y funcionales.

Se ejecuta del lado del cliente de la web y por lo general se usa para generar cambios sobre el DOM de las páginas web en función de los diferentes eventos que puedan ser disparados como rellenar un formulario, hacer click sobre un botón o sobre una imagen....

Cuando desarrollamos código de JavaScript disponemos de un gran número de APIs predefinidas que nos facilitará mucho el acceso a los diferentes eventos o a las diferentes modificaciones que queramos hacer sobre el DOM.

Este lenguaje se carga directamente sobre el navegador, lo que hace que no necesite ser compilado y que sea el propio navegador quien lea e interprete el código que tiene que ser ejecutado al dispararse el evento.

3.3. A-Frame

A-Frame es un framework web usado para crear experiencias en Realidad Virtual. A-Frame se encuadra a alto nivel dentro de HTML, esto hace que sea relativamente sencillo comenzar a trabajar con ello. A bajo nivel se estructura de una manera más compleja, la parte principal son las entidades que proporcionan una estructura declarativa y extensible de three.js

Fue creado originalmente por Mozilla a finales del 2015 como un proyecto de código abierto para tratar de desarrollar de forma sencilla pero potente entornos de realidad virtual sobre la web. El hecho de que todo fuera código abierto ha sido crucial para generar una comunidad de usuarios que se encarga del mantenimiento y de la ampliación del código base. A-Frame ha reunido a una de las mayores comunidades de realidad virtual.

A-Frame se puede utilizar con la mayoría de gafas de realidad virtual que hay en el mercado y se puede generar también escenarios de realidad aumentada. El objetivo principal es definir

experiencias de realidad virtual totalmente inmersivas, haciendo un uso amplio del seguimiento posicional y los controladores, dejando atrás los escenarios básicos de 360º

Las principales características de A-Frame son las siguientes:

- Hace que sea muy sencillo comenzar con la realidad virtual, sólo hay que colocar las etiquetas básicas en el HTML como son `<script>` para implementar código que modifique las diferentes entidades que podemos crear y la etiqueta `<a-scene>` que es la que se utilizará para que el navegador interprete que estamos generando una escena de realidad virtual.

- Gracias a la sencillez del lenguaje HTML, en el que se basa A-Frame al más alto nivel, a la hora de ser leído y entendido es accesible para todo tipo de usuarios, desde desarrolladores web hasta diseñadores.

- A-Frame a bajo nivel se sustenta sobre Three.js que tiene una estructura entidad-componente declarativa, reutilizable y fácil de componer. Esto hace que se usen tecnologías como JavaScript, APIs para el DOM, WebXR y WebGL.

- Una de las ventajas de A-Frame es que se pueden desarrollar aplicaciones que funcionan tanto en gafas de realidad virtual, como en ordenador o smartphones convirtiéndola en una tecnología muy flexible y fácil de adaptar.

- A-Frame se diseñó optimizado para WebXR. Aunque A-Frame se usa en el DOM, sus elementos no se cargan en el motor de diseño del navegador. Los objetos y sus modificaciones se generan en memoria, esto hace que incluso las aplicaciones más exigentes se vean fluidas a 90fps.

- A-Frame tiene un inspector que se puede abrir desde el navegador y permite ver las propiedades de todas las entidades creadas, así como poder modificarlas y adaptarlas a nuestras necesidades, aunque estas modificaciones no se verán reflejados en el código fuente pero nos puede servir para saber qué cambios necesitamos hacer en ese código fuente.

- Las entidades creadas con A-Frame tiene una parte de componentes como puede ser la geometría, el material, las luces, las animaciones. Todas estas características y muchas más van siendo corregidas y ampliadas gracias a la comunidad que hay involucrada en el crecimiento de A-Frame

- Gracias a esa comunidad es una tecnología que está más que probada y es muy escalable. Grandes tecnológicas como Google, Samsung, Microsoft hacen grandes contribuciones al crecimiento de A-Frame.

3.4. Three.js

Three.js[1] es una librería que se usa para simplificar el código para acceder a la API WebGL. Three.js maneja de manera sencillas cosas como las escenas, las luces y sombras, materiales y texturas...

La mayoría de los navegadores se actualizan automáticamente para soportar Three.js, por lo que no hay que hacer nada especial para poder usarlo.

En la figura 3.1 se puede observar el diagrama de uso de Three.js que consiste en crear diferentes elementos y relacionarlos todos entre sí.

La parte principal de Three.js es el renderizador, a éste le pasas información de la escena y de la cámara y el se encarga de dibujar la escena dentro del radio de acción de la cámara.

La escena consiste en una estructura jerárquica de tipo árbol en el que se definen objetos como `Mesh`, `Object3D`, `Group`, `Light`.... Esta estructura jerárquica hace que los objetos se posicionen en relaciones padre/hijo, es decir, cada hijo se posicionará en función de cómo esté posicionado su padre y no cualquier otro objeto de la escena.

`Mesh` necesita la geometría y el material específicos de un objeto para poder pintarlos. Tanto geometría como material pueden ser utilizados múltiples veces por objetos `Mesh`.

La geometría de los objetos representan los datos de los vértices de los objetos. Pueden ser geometrías primitivas como cubos o esferas proporcionadas directamente por la librería Three.js o pueden ser geometrías personalizadas.

El material de los objetos representa propiedades como el color o el brillo de los objetos que se utiliza para dibujarlos.

La textura de los materiales suelen representar imágenes que se cargan de fichero externo para dibujar los objetos, por ejemplo, en lugar de ponerle algún color a un objeto le pondremos una imagen para dibujarlo.

La luz representa el foco o focos de luz que tiene la escena, esto provoca que los objetos que se crean puedan generar sombras en función de los puntos de luz que tenga la escena.

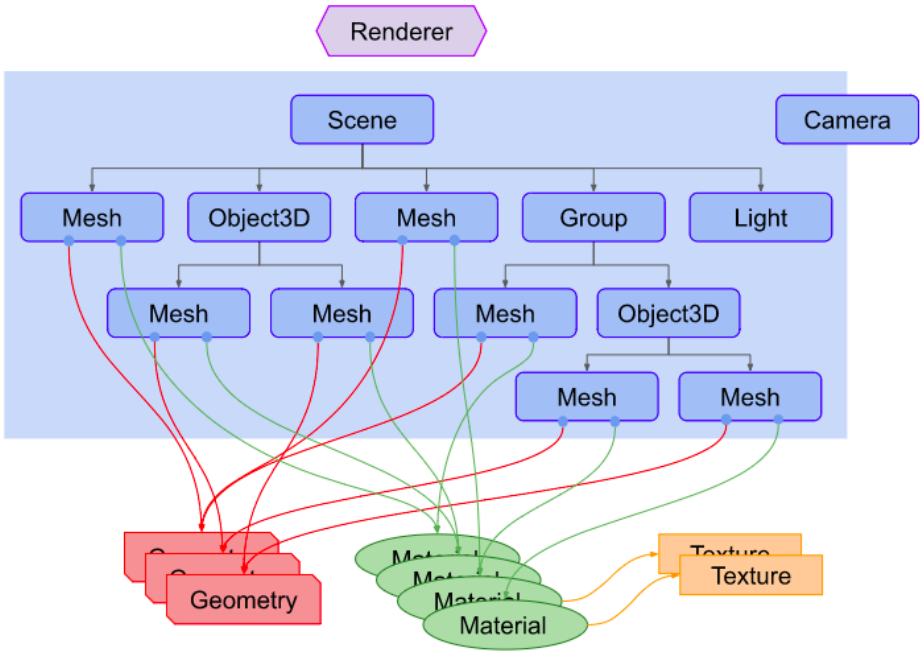


Figura 3.1: Diagrama de una aplicación con Three.js

3.5. WebGL

WebGL[10] es una API que se utiliza para pintar en el navegador escenas 3D. La particularidad que tiene WebGL es que accede al hardware, la tarjeta gráfica, para tener más potencia de renderizado y así poder generar gráficos 3D más complejos en el navegador. Una librería que se usa para acceder a la API de WebGL es Three.js, las librerías facilitan mucho la programación ya que es muy complicado desarrollar directamente sobre WebGL.

3.6. WebXR

WebXR[11] se utiliza para el renderizado de escenas 3D para hardware de Realidad Virtual o Realidad Aumentada. La API de los dispositivos implementa el conjunto de características de WebXR, esto hace que se renderice la escena en 3D a la velocidad de fotograma adecuada y en función de los vectores de movimiento relacionados con los controladores de entrada.

Los dispositivos compatibles con WebXR son las gafas de Realidad Virtual, los auriculares 3D y los teléfonos móviles compatibles con la Realidad Aumentada.

Un escena se pinta en 3D teniendo en cuenta la posición de los ojos del usuario, se renderiza

desde esa posición y en la dirección en la que mira el usuario. Las imágenes y el punto de vista se renderizan de manera cruzada, es decir, se renderiza la imagen del ojo izquierdo con la perspectiva del ojo derecho y viceversa, una vez renderizadas las dos se envía al dispositivo para hacerlo visible al usuario.

3.7. SDN

SDN (Software Defined Network) nace de la idea de separar las funciones de reenvío y las funciones de control en los dispositivos de red, moviendo esas funciones a un servidor común centralizado. El sistema operativo de la red analiza y controla su estado, ejecuta protocolos de enrutamiento, control de acceso o virtualización de la red y el reenvío a través de una interfaz independiente. Este reenvío está definido a bajo nivel en el SO de la red, si uno de los paquetes que se tienen que reenviar cumple las condiciones de las reglas definidas se ejecutan diferentes acciones como reenviar el paquete, descartarlo, modificarlo o encolarlo para enviarlo más tarde.

Uno de los primeros retos que surgieron fue la necesidad de un crear un protocolo común entre el controlador del SDN y el dispositivo de red. Para solventar este problema se creó OpenFlow que se usa sobre el canal seguro (TLS sobre el puerto 6633 de TCP). Se utiliza para modificar las tablas de flujos y grupos en un dispositivo de red.

La mayor ventaja de usar SDN es que se puede definir la funcionalidad de la red tras el despliegue de la misma, no es necesario hacerlo previamente. Esto hace que sobre una misma red se pueden ir adaptando funciones para optimizar su velocidad. El uso de SDN permite obtener nuevos puntos de vista para gestionar los estados de la red y nuevos usos para los paquetes y para las cabeceras de esos paquetes.

El flujo de trabajo de prototipado rápido podría ser la clave para alcanzar el máximo potencial de las redes definidas por software.

3.8. Mininet

Mininet proporciona una manera rápida para crear, interactuar, personalizar y compartir una SDN.

Para crear una red virtual, Mininet simula los links, hosts, switches y controladores, además

utiliza los mecanismos de virtualización propios de Linux.

Los diferentes componentes que simula Mininet se describen de la siguiente manera:

- Links: es un par virtual de Ethernet que actúa como unión de dos interfaces virtuales. Los paquetes son creados en una interfaz y enviado a otra a través del link. Cada interfaz se corresponde con un puerto Ethernet plenamente funcional para todo el sistema. Se pueden conectar a switches virtuales.

- Hosts: los Namespaces de red son el aspecto principal de los contenedores donde está todo el estado de la red. Proporciona a los procesos la propiedad exclusiva de interfaces, puertos y tablas de enrutamiento. Un host en Mininet no es más que un proceso de la shell movido a su propio Namespace de la red. Cada host tiene su propia interfaz virtual de Ethernet enlazado con el proceso padre de Mininet que envía comandos y monitoriza su salida.

- Switches: replica de manera exacta la semántica de entrega de paquetes que pudiera tener un switch físico.

- Controladores: pueden estar en cualquier punto de la red, ya sea ésta simulada o real. La única condición indispensable es que la máquina donde se ejecutan los switches tenga conectividad a nivel de IP con el controlador. Un controlador puede ejecutarse en una máquina virtual, en una máquina física o en la nube.

Tras lanzar una red tenemos la posibilidad de interactuar con ella a través de una interfaz de línea de comandos, podemos ejecutar comandos sobre los distintos hosts, verificar que los switches están enrutando en función de la tabla y apagar o encender los hosts para comprobar cómo se adapta la conectividad. Como la interfaz de línea de comandos tiene los nombres de los diferentes hosts podemos ejecutar comandos directamente con el nombre, sin la necesidad de conocer las direcciones IP de cada uno.

Mininet utiliza una API de Python para crear diferentes topologías para analizar. Se puede definir una red, ejecutar diferentes comandos sobre los nodos y mostrar los resultados.

Por lo general Mininet debe usarse sobre una máquina virtual, ya que así es más sencillo compartir los diferentes prototipos que se hayan creado. Aunque puede instalarse de forma nativa en ciertas distribuciones de Linux.

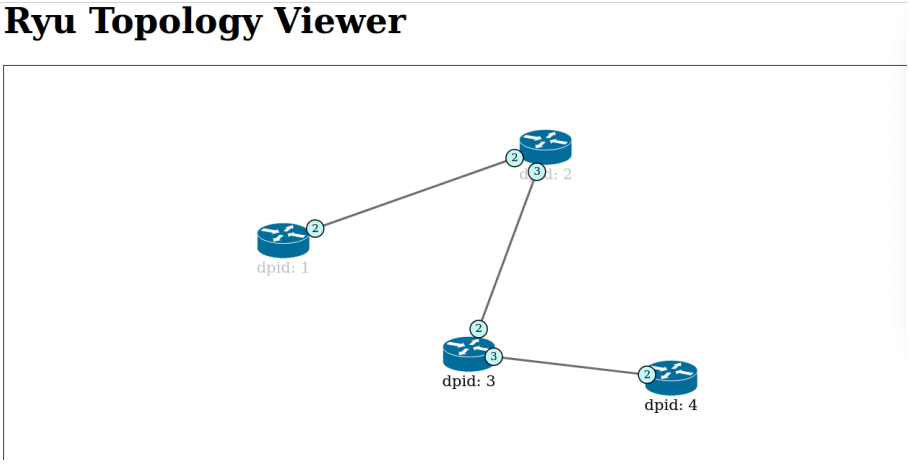


Figura 3.2: Topología vista desde el visualizador de RYU

Una vez que tenemos la red virtual podemos trasladarla al mundo físico, pero para que funcione de manera correcta debemos asegurarnos que cada uno de los componentes físicos, ya sean los hosts o los switches sean iguales a los que hemos configurado de manera virtual. Lo único que no es necesario modificar es el controlador. Esto es porque el controlador ya interpreta como componentes físicos cada uno de los componentes virtuales que hemos creado. La interfaz de la línea de comandos permite poder interactuar con los componentes físicos de la misma manera que hacíamos con los componentes virtuales.

3.9. RYU

Ryu es un software de código abierto implementado en Python que proporciona herramientas y librerías para crear nuevas aplicaciones de control y gestión de redes SDN de manera sencilla. Soporta múltiples protocolos de gestión de dispositivos.

También se puede utilizar para visualizar la topología de la red gracias a la información recibida en el controlador una vez que se ha arrancado tal y como se puede observar en la figura 3.2

Capítulo 4

Diseño e implementación

En este capítulo haré una descripción sobre el proceso que he seguido para llegar al final del desarrollo del proyecto. Comenzaré desde el escenario más simple con el que compruebo que realmente se puede ver una red de dispositivos conectados entre sí y terminaré con el escenario definitivo integrando la parte de visualización con la parte de generación de la red.

4.1. Visualización de una escena

Basándome en el ejemplo de iniciación que propone A-Frame traté de crear una escena que se asemejara lo más posible al resultado final que quiero conseguir con este proyecto.

Para ello creé un archivo HTML[1]. En él incluí las diferentes entidades que necesitaba para visualizar una red previamente diseñada, sin ningún tipo de funcionalidad y con todos los valores predeterminados. El único objetivo que tenía era comprobar cómo quedaba sobre el navegador una red en 3D.

En la figura 4.1 se pueden encontrar tres elementos que son los que componen nuestra red:

- El controlador, que estaría etiquetado como S0, es el que tendría toda la información de la red para poder analizarla y trabajar con ella, además de poder controlarla.

- Tenemos dos switches (S1 y S2) que serían los encargados de suministrar al controlador con toda la información de la red.

[1] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_estatica.html`

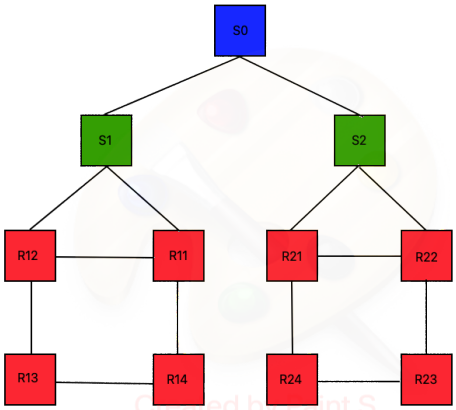


Figura 4.1: Diseño de la red que vamos a representar

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
    <script src="https://raw.githack.com/andreasplesch/aframe-meshline-component/master/dist/aframe-meshline-component.min.js"></script>
  </head>
```

Figura 4.2: Cabecera del archivo HTML

- Los routers o hosts que componen la red, etiquetados con RXY, donde X indica el switch al que está conectada la malla de routers a la que pertenece y la Y indica el número asignado al router.

En el HTML podemos observar los diferentes elementos que nos van a ayudar a crear la escena de la figura 4.1.

En este caso como se trata de un ejemplo en el que no hemos utilizado archivos de JavaScript para generar las entidades la cabecera será muy sencilla, en la podemos encontrar, bajo la etiqueta `<script>`, dos scripts que en este caso están referenciados como una URL, que son los que utilizará el navegador para interpretar y pintar las entidades que vamos a crear. La referencia a `aframe.min.js` será fundamental para poder crear entidades básicas de A-Frame, este archivo tiene, en formato JavaScript todas las funciones básicas que se utilizarán.

La referencia a `aframe-meshline-component.min.js` será utilizada, en este caso, para generar las líneas que se utilizan para unir los diferentes elementos de nuestra escena.

En la figura 4.3 se muestra el código utilizado para generar algunas de las entidades que va a tener nuestra escena. En este caso, la malla que corresponde al S1 y los routers R11, R12, R13

```
<!-- Switch 1 -->
<a-box position="2.3 0.5 -4" rotation="0 0 0" color="#3FD920" scale="0.5 0.5 0.5"></a-box>
<!-- Router 11 -->
<a-box position="1 0.5 -3" rotation="0 0 0" color="#D91012" scale="0.5 0.5 0.5"></a-box>
<!-- Router 12 -->
<a-box position="1 0.5 -5" rotation="0 0 0" color="#D91012" scale="0.5 0.5 0.5"></a-box>
<!-- Router 13 -->
<a-box position="-1 0.5 -5" rotation="0 0 0" color="#D91012" scale="0.5 0.5 0.5"></a-box>
<!-- Router 14 -->
<a-box position="-1 0.5 -3" rotation="0 0 0" color="#D91012" scale="0.5 0.5 0.5"></a-box>
<!-- Union 11-14 -->
<a-entity meshline="lineWidth: 10; path: 0 0 0, 2 0 0; color: #2212D9" position="-1 0.5 -3"></a-entity>
<!-- Union 12-13 -->
<a-entity meshline="lineWidth: 10; path: 0 0 0, 2 0 0; color: #2212D9" position="-1 0.5 -5"></a-entity>
<!-- Union 13-14  -->
<a-entity meshline="lineWidth: 10; path: 0 0 0, 0 0 2; color: #2212D9" position="-1 0.5 -5"></a-entity>
<!-- Union 12-11 -->
<a-entity meshline="lineWidth: 10; path: 0 0 0, 0 0 2; color: #2212D9" position="1 0.5 -5"></a-entity>
<!-- Switch 2 -->
```

Figura 4.3: Código para generar la malla del switch S1

y R14.

Como se puede observar en la figura 4.3 para el switch y los diferentes routers se utiliza la misma entidad de A-Frame y para los enlaces hemos usado una diferente. Cada entidad tiene diferentes atributos que se utilizan para visualizarla.

En este caso y como se aprecia en la figura 4.3 hemos usado los siguientes para generar los switches y los routers:

- Geometria: `box` al tratarse de una geometría primitiva de A-Frame no se refleja en los atributos sino que se hace en la etiqueta del HTML `a-box`
- Posición: son las coordenadas en las que se va a situar el centro del objeto que vamos a crear. El orden de las coordenadas es X, Y, Z.
- Rotación: es el ángulo (en grados) que se rota la figura con respecto a cada eje de coordenadas, al igual que ocurre con la posición el orden de los ejes es X, Y, Z.
- Color: es el color con el que se va a mostrar la entidad, el valor que aparece es hexadecimal pero también puede ponerse el nombre de colores básicos (red, black, cyan...).
- Escala: por defecto las entidades con una geometría primitiva tienen un ancho, largo y alto con un valor de 1, esto se puede cambiar de dos maneras, modificando los atributos correspondientes a cada medida poniéndole el que nos interese, o modificando la escala

de la entidad. En este caso he decido utilizar la escala porque implica menos código y quiero que todos tengan el mismo tamaño.

Como se puede ver en la figura 4.3 para generar los enlaces hemos utilizado una entidad con una geometría no primitiva. Por lo tanto los atributos que tenemos que modificar son diferentes:

- Meshline: se invoca a la función para pintar una línea con los valores para algunas de sus atributos. Los atributos que modificamos son LineWidth, Path y Color
  - LineWidth: se indica el grosor de la línea en píxels
  - Path: son dos conjuntos de coordenadas que indican el inicio, primer conjunto, y el final, segundo conjunto, de la línea, estas coordenadas son relativas a la posición del objeto en la escena. Por ejemplo, si el primer conjunto es `0 0 0` la línea comenzará exactamente en la posición en la que está colocada la entidad dentro de la escena. Si el segundo conjunto tiene los valores `2 0 0` la línea terminará en las coordenadas de la posición de la identidad a las que se le sumará dos unidades a la coordenada X y ninguna unidad a las coordenadas Y y Z
  - Color: es el color con el que se va a mostrar la entidad, el valor que aparece es hexadecimal pero también puede ponerse el nombre de colores básicos (red, black, cyan...).
- Posición: son las coordenadas en las que se va a situar el objeto que vamos a crear. El orden de las coordenadas es X, Y, Z.

En la figura 4.4 se puede observar cómo se han generado las diferentes cajas y líneas que hemos creado para formar una red de routers y switches conectados entre sí. En la figura 4.5 se puede observar directamente sobre el navegador cómo se han generado las diferentes cajas y líneas que hemos creado para formar una red de routers y switches conectados entre sí.

4.2. Generación de entidades desde JavaScript

Una vez que he conseguido crear una escena que se asemeja al resultado final que quiero conseguir, el siguiente paso es generar las entidades desde un archivo de JavaScript.

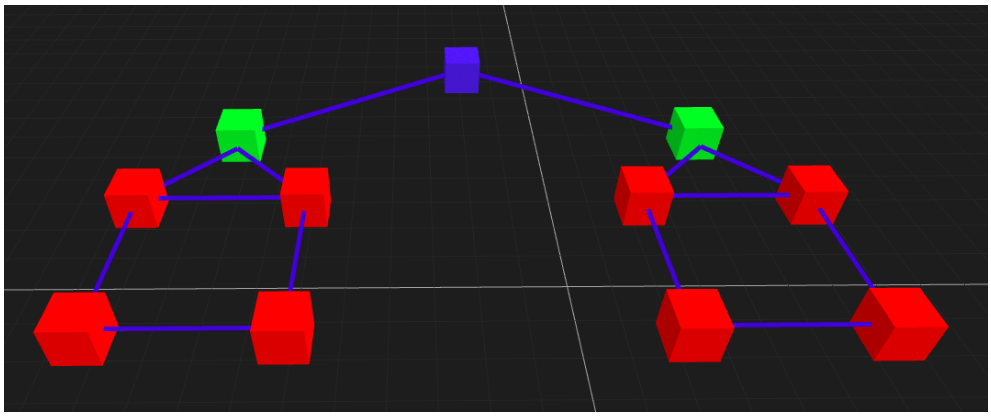


Figura 4.4: Visualización de la red desde el inspector

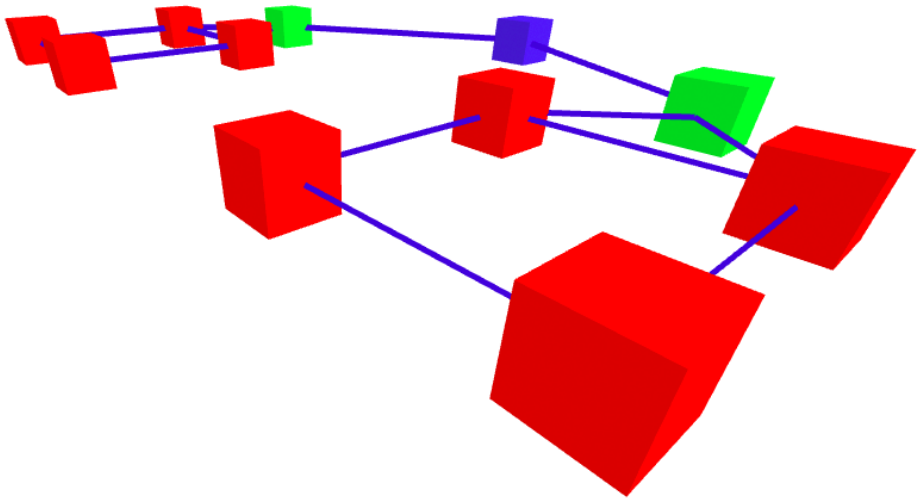


Figura 4.5: Visualización de la red desde el navegador

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
    <script src="https://raw.githack.com/andreasplesch/aframe-meshline-component/master/dist/aframe-meshline-component.min.js"></script>
    <script src="crear_cajas.js"></script>
  </head>
  <body>
    <a-scene inspector="https://cdn.jsdelivr.net/gh/aframevr/aframe-inspector@master/dist/aframe-inspector.min.js" box>
    </a-scene>
  </body>
</html>
```

Figura 4.6: Visualización de la red desde el navegador

Para ello en este caso he necesitado crear un archivo HTML[2] y un archivo JavaScript[3].

Como se hace prácticamente todo desde el archivo JavaScript el archivo HTML es muy sencillo, como se puede apreciar en la figura 4.6. En este caso, como sólo nos interesa generar las cajas desde el archivo JavaScript y no vamos a dibujar líneas sólo hemos cargado la referencia a `aframe.min.js`.

También se puede ver que se carga en la cabecera el script con nombre `crear_cajas.js`, esto hará que se cargue el archivo de JavaScript en el que hemos desarrollado la lógica para incluir un nuevo componente que generará las diferentes entidades.

La última parte fundamental de este código es cómo cargamos el componente que hemos creado, llamado `box`. Éste lo cargamos dentro de la etiqueta `<a-scene>`, lo cual hace que el navegador interprete que el componente llamado `box` que hemos creado en el archivo JavaScript se ejecute en este punto y se aplique la lógica que tenga.

En el archivo JavaScript que estamos cargando en la cabecera del HTML podemos ver algunas partes claramente diferenciadas.

En la figura 4.7 se observa como se crea el schema de un componente, estos son los atributos que va a llevar el componente, los que se pueden pasar a través de la función si se quieren modificar de alguna manera, si no se pasa ningún valor en la función se mantendrán los que definamos por defecto. En este caso hemos definido como atributos principales el alto, el ancho y la profundidad para tener el tamaño del objeto así como el color, he decido definir el valor como hexadecimal aunque se puede definir con el nombre de los colores básicos (red, black, cyan...).

[2] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_semidinamica_cajas.html`

[3] `https://github.com/albertoabades/TFG-AFrame/blob/main/crear_cajas.js`

```
AFRAME.registerComponent('box', {
  schema: {
    width: {type: 'number', default: 0.5},
    height: {type: 'number', default: 0.5},
    depth: {type: 'number', default: 0.5},
    color: {type: 'color', default: '#74BEC1'}
  },
```

Figura 4.7: De esta manera se define el esquema que va a tener el componente

```
init: function () {
  var data = this.data;
  var el = this.el;

  //Matriz para asignar coordenadas a cada caja/router que se va a crear
  let coordenadas = [
    ["s0", 4, 0.5, -1],
    ["s1", 2.3, 0.5, -4],
    ["s2", 2.3, 0.5, 2],
    ["s11", 1, 0.5, -3],
    ["s12", 1, 0.5, -5],
    ["s13", -1, 0.5, -5],
    ["s14", -1, 0.5, -3,],
    ["s21", 1, 0.5, 1],
    ["s22", 1, 0.5, 3],
    ["s23", -1, 0.5, 3],
    ["s24", -1, 0.5, 1]
   ]
```

Figura 4.8: Inicialización de las variables al iniciar el componente

Una vez definido el esquema del componente pasamos a desarrollar la lógica que va a tener ese componente al iniciarse. Como se puede apreciar en la figura 4.8 se inicializan diferentes variables que se van a utilizar en la lógica. La variable `coordenadas` es un array de arrays en el que definimos las coordenada que va a tener cada entidad que vamos a crear, en este caso también le asignamos un ID único.

En este punto desarrollamos la lógica con la que vamos a crear las diferentes entidades que va a tener nuestra escena, como se ve en la figura 4.9 consta de tres partes claramente diferenciadas. Todo se construye sobre un bucle en el que vamos a recorrer el array de arrays `coordenadas` para ir recuperando los datos de `ID` y `Position` de cada una de las entidades que vamos a crear.

Los tres pasos que tenemos que seguir para crear la entidad son:

- Lo primero que hacemos es generar la entidad, en este caso no lo hacemos como entidad genérica sino que lo hacemos con la geometría primitiva de una caja, esto nos lo facilita A-Frame con las geometrías primitivas.

```
for (let i = 0; i < coordenadas.length; i++) {
  //Se crea un elemento genérico de tipo cada
  var boxEl = document.createElement('a-box');
  //Se asignan las coordenadas recuperadas de la matriz de coordenadas
  boxEl.setAttribute('position', {
    x: coordenadas[i][1],
    y: coordenadas[i][2],
    z: coordenadas[i][3]
  })
  //Se asigna un color diferente a cada caja/router que se acaba de crear en función del tipo que es
  if (coordenadas[i][0] == "s0"){
    boxEl.setAttribute('color', '#000000');
  }else if (coordenadas[i][0].length == 2 && coordenadas[i][0] != "s0"){
    boxEl.setAttribute('color', '#264BFF');
  }else{
    boxEl.setAttribute('color', '#C13C39');
  }
  //Se asigna la escala de a la caja/router que se acaba de crear, en este caso se reduce a la mitad
  boxEl.setAttribute('scale', {x: 0.5, y:0.5, z:0.5})
  //Se asigna un ID único a la caja/router que se acaba de crear para poder identificarlo de manera única
  boxEl.setAttribute('ID', coordenadas[i][0]);
  //Se incluye la caja/router que se acaba de crear en la escena general
  document.querySelector('a-scene').appendChild(boxEl);
```

Figura 4.9: Lógica para generar las diferentes cajas de la escena

- Una vez que tenemos creada la entidad, le asignamos los valores a sus diferentes atributos como `ID` y `Position` cuyos valores recuperamos de la variable `coordenadas` que hemos definido previamente. También la asignamos valores a los atributos `Scale` o `Color`. La entidad tiene muchos más atributos pero no nos importa que se genere con sus valores por defecto.

- Por último hay que incluir en la escena la entidad que acabamos de crear y configurar. Esto se hace seleccionando la escena en el DOM, por eso se utiliza la función `QuerySelector` sobre el documento, una vez seleccionada la escena le incluimos como hijo la entidad que acabamos de crear.

El resultado que obtenemos al ejecutar sobre el nevagador el HTML[4] es el que se puede observar en la figura 4.10. Se puede observar que en este caso se ha generado una red similar a la que se generó con el archivo HTML[5] en el que definimos la red sin utilizar archivos JavaScript. En este caso, no aparecen las conexiones entre cajas pero el dibujo de la red es el mismo.

[4] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_semidinamica_cajas.html`

[5] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_estatica.html`

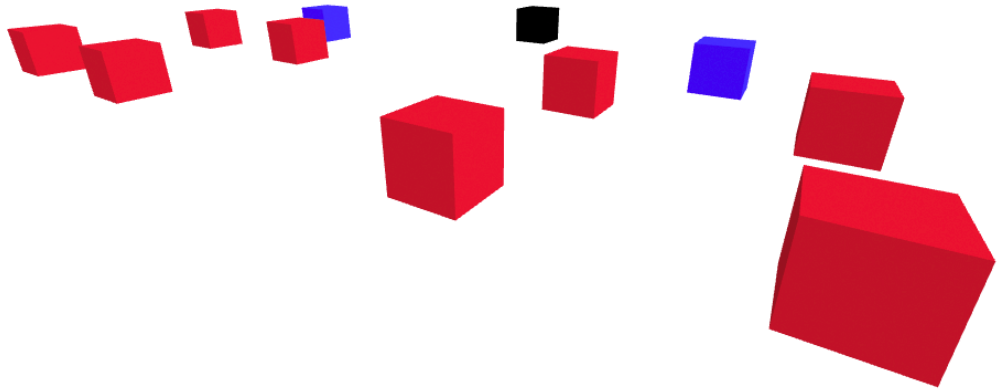


Figura 4.10: Visualización de las cajas en el navegador

```
AFRAME.registerComponent('cylinder', {
  schema: {
    radius: {type: 'number', default: 0.5},
    height: {type: 'number', default: 2},
    color: {type: 'color', default: '#74BEC1'}
  },
```

Figura 4.11: Esquema del componente cilindro

4.3. Incluir enlaces entre las cajas

Una vez que hemos conseguido generar una malla de cajas desde un archivo JavaScript, el siguiente paso es generar los enlaces entre esas cajas. Para ello generamos un nuevo archivo HTML[6] y un nuevo archivo JavaScript[7].

En este punto he decidido generar los enlaces como cilindros, ya que da mucha más versatilidad que las líneas, para ello hemos creado el archivo JavaScript `crear_cilindros.js`. Este archivo es muy similar al que hemos utilizado para generar las cajas.

Como se puede apreciar en la figura 4.11 el esquema del componente es diferente al de las cajas que ya hemos creado, en este caso definimos el radio del cilindro, la longitud o altura del cilindro y el color.

En la figura 4.12 se puede ver como en la variable en la que definimos los datos del cilindro

[6] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_semidinamica.html`

[7] `https://github.com/albertoabades/TFG-AFrame/blob/main/crear_cilindros.js`

```
let datos_cilindros =[
  ["s11s12", 1, 0.5, -4, 0, 90, 90],
  ["s12s13", 0, 0.5, -5, 0, 0, 90],
  ["s13s14", -1, 0.5, -4, 0, 90, 90],
  ["s14s11", 0, 0.5, -3, 0, 0, 90],
  ["s21s22", 1, 0.5, 2, 0, 90, 90],
  ["s22s23", 0, 0.5, 3, 0, 0, 90],
  ["s23s24", -1, 0.5, 2, 0, 90, 90],
  ["s24s21", 0, 0.5, 1, 0, 0, 90],
  ["s11s1", 1.65, 0.5, -3.5, 0, 45, 90],
  ["s12s1", 1.65, 0.5, -4.5, 0, -45, 90],
  ["s21s2", 1.65, 0.5, 1.5, 0, -45, 90],
  ["s22s2", 1.65, 0.5, 2.5, 0, 45, 90],
  ["s1s0", 3.15, 0.5, -2.5, 0, -55, 90],
  ["s2s0", 3.15, 0.5, 0.5, 0, 55, 90]
]
```

Figura 4.12: Datos de cada uno de los cilindros que unirán las cajas

tenemos más valores dentro de cada array. El primer valor se corresponde con el ID del cilindro y nos indica las cajas que estamos uniendo, los tres siguientes valores corresponden a la posición y los tres últimos corresponden con la rotación del cilindro con respecto a los distintos ejes, el orden de los valores de rotación son X, Y, Z.

Los cilindros por defecto se crean perpendiculares al eje X, lo que necesitamos es hacer dos rotaciones, la primera sobre el eje Z para poner el clilindro paralelo al eje X y por lo tanto, se sitúa a la misma altura que las cajas. Y la segunda rotación se hace sobre el eje Y para rotarlo de tal manera que haga contacto con dos cajas para simular que están unidas.

La posición en este caso la calculamos de tal manera que se sitúe en el centro geométrico entre las dos cajas que queremos unir con el cilindro, por tanto el centro de la longitud del cilindro al situarlo paralelo al eje X quedará en ese punto con lo que podremos calcular de manera sencilla la longitud que debe tener cada cilindro.

Para generar las diferentes entidades creamos un bucle para recorrer el array con los datos de cada cilindro como se puede ver en la figura 4.13.

Al igual que antes seguimos tres pasos para llegar a incluir los cilindros en la escena:

- Se crea una entidad con el tipo primitivo `a-cylinder` que ya nos proporciona la base de A-Frame.

```
//Bucle para recorrer la matriz de datos_cilindros y generar cada cilindro que servirá de enlace
for (let i = 0; i < datos_cilindros.length; i++) {
  // Se crea un cilindro para que haga de uniones entre cajas
  var cylEl = document.createElement('a-cylinder');
  //Se asignan las coordenadas recuperadas de la matriz de datos_cilindros al cilindro que servirá
  cylEl.setAttribute('position',
    {
     x: datos_cilindros[i][1],
     y: datos_cilindros[i][2],
     z: datos_cilindros[i][3]
    })
  //Se asigna la rotación al cilindro para que se unan los router/cajas, se recuperan de la matriz
  cylEl.setAttribute('rotation',
    {
      x: datos_cilindros[i][4],
      y: datos_cilindros[i][5],
      z: datos_cilindros[i][6]
    })
  // Se asigna un ID único al cilindro que servirá de enlace
  cylEl.setAttribute('ID', datos_cilindros[i][0]);
  // Se asigna un color al cilindro que servirá de enlace y se acaba de crear
  cylEl.setAttribute('color', '#74BEC1')
  // Se asigna un radio al cilindro que servirá de enlace y se acaba de crear
  cylEl.setAttribute('radius', 0.03)
  // Se asigna una longitud a los cilindro que servirán de enlace y se acabn de crear.
  if(datos_cilindros[i][0] == "s1s0" || datos_cilindros[i][0] == "s2s0"){
    cylEl.setAttribute('height', 3)
  }else{
    cylEl.setAttribute('height', 2)
  }
  // Se incluye el cilindro que servirá de enlace y se acaba de crear en la escena
  document.querySelector('a-scene').appendChild(cylEl);
```

Figura 4.13: Lógica para generar las diferentes cilindros de la escena

- Una vez tenemos la entidad configuramos los diferentes atributos que nos interesan, para nuestra escena necesitamos configurar, además de los atributos `Position`, `ID` y `Color` que los configuramos de la misma manera que antes, tenemos que configurar el atributo `Rotation` recuperando la información del array de arrays que hemos inicializado anteriormente, también configuramos el atributo `Radius` que lo haremos con un valor pequeño para que los enlaces no queden muy gruesos. Por último configuramos el atributo `Height`, como conocemos previamente la distancia que hay entre las diferentes cajas podemos darle un valor predefinido.

- Una vez que tenemos la entidad creada y los atributos correctamente configurados hay que incluirla en la escena. Para ello, al igual que hemos hecho antes, seleccionamos la escena en el DOM y le incluimos como hijo la entidad que acabamos de crear y configurar.

En la figura 4.14 se puede ver cómo queda la escena que hemos generado a través de un archivo HTML[8], un archivo JavaScript[9] para las cajas y un archivo JavaScript[10] para los enlaces

[8] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_semidinamica.html`

[9] `https://github.com/albertoabades/TFG-AFrame/blob/main/crear_cajas.js`

[10] `https://github.com/albertoabades/TFG-AFrame/blob/main/crear_cilindros.js`

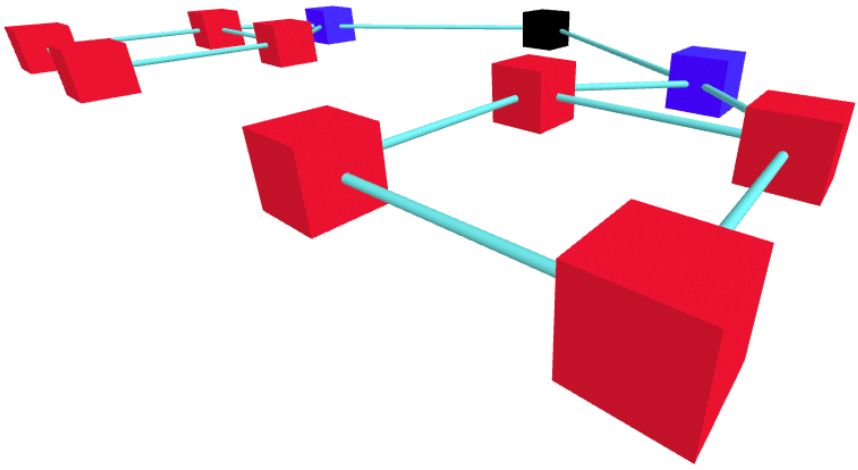


Figura 4.14: Visualización de la red completa


```
[
    {
        "src": {"dpid": "0000000000000002", "port_no": "00000002", "hw_addr": "ea:54:6d:2b:64:da", "name": "s2-eth2"},
        "dst": {"dpid": "0000000000000001", "port_no": "00000002", "hw_addr": "06:31:ad:c6:15:87", "name": "s1-eth2"}
    },
    {
        "src": {"dpid": "0000000000000002", "port_no": "00000003", "hw_addr": "5e:18:5f:86:c3:7c", "name": "s2-eth3"},
        "dst": {"dpid": "0000000000000003", "port_no": "00000002", "hw_addr": "36:86:f6:3f:c8:a8", "name": "s3-eth2"}
    }
]
```


Figura 4.15: Información que contiene el archivo JSON que vamos a recuperar

4.4. Lectura de un archivo JSON

Una vez que hemos conseguido crear una escena modificando el DOM desde archivos JavaScript vamos a pasar a la lectura de archivos JSON, esto hará que podamos integrar nuestro código con alguna aplicación externa.

En la figura 4.15 se ve la información que contiene el JSON[11] sencillo que hemos creado con la única finalidad de recuperarlo del servidor y convertirlo en un objeto JavaScript

Una vez que tenemos creado el archivo JSON desarrollamos el código JavaScript[12] como se ve en la figura 4.16. Con él lo recuperamos del servidor con una llamada GET. Para ello creamos una función llamada `fetchJSONFile`, a esa función le vamos a pasar como parámetros la ruta del fichero JSON que esperamos recuperar y una función que nos lo convertirá en un objeto JavaScript para poder tratarlo en nuestra lógica.

Lo primero que hacemos es inicializar una variable como un objeto llamado `XMLHttpRequest`[12],

[11] `https://github.com/albertoabades/TFG-AFrame/blob/main/Prueba_Lectura.json`

[12] `https://github.com/albertoabades/TFG-AFrame/blob/main/lectura_json.js`

```
fetchJSONFile('Prueba_Lectura.json', function(data){
    // do something with your data
    console.log(data);
});

function fetchJSONFile(path, callback) {
    var httpRequest = new XMLHttpRequest();
    httpRequest.onreadystatechange = function() {
        if (httpRequest.readyState === 4) {
            if (httpRequest.status === 200) {
                var data = JSON.parse(httpRequest.responseText);
                if (callback) callback(data);
            }
        }
    };
    httpRequest.open('GET', path);
    httpRequest.send();
}
```

Figura 4.16: Función para recuperar un archivo JSON y convertirlo a un objeto de JavaScript

este objeto nos facilita obtener información de un path sin necesidad de tener que recargar una página.

A continuación llamamos a la función `onreadystatechange` del objeto que acabamos de inicializar, esta función se llama cuando el atributo `readyState` cambia. Dentro de la función comprobamos que el valor de `readyState` es 4, lo que significa la operación que estamos haciendo ha terminado y además comprobamos el atributo `status`, este atributo nos indica el estado de la respuesta, en este caso 200 indica que todo ha ido correctamente.

Una vez que hemos hecho las comprobaciones de que todo ha ido correctamente utilizamos el método `JSON.parse`[7] que convierte una cadena de texto en un objeto JSON. La cadena de texto se obtiene de la respuesta cuando todo ha ido bien a través del atributo `responseText`. El objeto JSON que hemos parseado se lo pasamos a la función `callback` para luego poder trabajar con él. En este caso, como sólo estamos creando código para recuperar el archivo JSON lo único que vamos a hacer es imprimir por consola el objeto obtenido para ver si todo ha ido correctamente.

Para iniciar la petición debemos usar el método `open()` al que le vamos a pasar como parámetros el método HTTP que vamos a usar `GET` y la ruta sobre la que se hace la petición. Una vez que ya tenemos la información con la que vamos a hacer la petición la realizamos con el método `send()`.

Ya tenemos definida la lógica con la que vamos a recuperar el archivo JSON y el propio

```
<html>
  <head>
    <script src="lectura_json.js"></script>
  </head>
  <body>
  </body>
</html>
```

Figura 4.17: Archivo HTML que usamos para cargar un archivo JSON

archivo archivo JSON. Ahora tenemos que tener en cuenta varios puntos importantes:

- Debido a protocolos de seguridad sólo se pueden recuperar archivos a través de un servidor, he optado por usar un servidor sencillo de Python que se lanza en el terminal en la carpeta en la que tenemos el HTML y el JSON con el comando `python3 -m http.server`. Debido a la política *same-origin*[8], que es un mecanismo de seguridad para evitar cargar documentos maliciosos, tenemos que tener en la misma carpeta donde hemos lanzado el servidor el archivo HTML y el archivo JSON que carga el HTML.

- Al igual que antes, debemos cargar en la cabecera del archivo HTML el archivo de JavaScript con el que hacemos la petición al servidor para recuperar el archivo JSON.

En la figura 4.17 vemos que el archivo HTML es muy sencillo. En este caso no incluimos nada en el cuerpo y en la cabecera únicamente cargamos el archivo `lectura_json.js` en el que hemos definido la lógica de la llamada para recuperar el JSON y su tratamiento.

Para comprobar que todo ha ido correctamente cargamos en el navegador el fichero HTML y comprobamos la salida de la consola para ver si se ha recuperado la información que hemos solicitado. En la figura 4.18 se ve la salida de la consola y comprobamos que se ha creado un objeto tipo diccionario con claves y valores. Una vez que tenemos ese objeto podremos trabajar con él.

4.5. Arrancar una red SDN en la máquina virtual

Ahora vamos a arrancar las diferentes SDN que hemos configurado para obtener los diferentes archivos JSON correspondientes a cada una y así poder trabajar con ellos sin necesidad

Figura 4.18: Salida de la consola - Representación del archivo JSON

de tener activa la máquina virtual.

Lo primero que tenemos que hacer es arrancar el controlador Ryu desde el terminal, así lo pondremos a escuchar en el puerto 8080 del localhost y podremos hacer la llamada `GET` para obtener el JSON. Para ello tendremos que ejecutar los siguientes comandos en el terminal:

- `source /Escritorio/venv/bin/activate`
- `cd /Escritorio/venv/lib/python3.8/site-packages`
- `ryu run --observe-links ryu/app/gui_topology/gui_topology.py`

En la figura se puede ver que al ejecutar el último comando se queda escuchando en localhost:8080.

Una vez que hemos arrancado el controlador tenemos que arrancar las diferentes SDN de prueba, hemos configurado cinco diferentes que ejecutaremos de manera independiente sobre un terminal distinto al que hemos utilizado para arrancar el controlador. Para visualizarlas únicamente hay que abrir el navegador y escribir `http://localhost:8080/` y se mostrará la red que esté activada en ese momento.

Los comandos que tenemos que ejecutar son los siguientes:

- `sudo mn --controller remote --topo tree,depth=3`. La red SDN generada se puede ver en la figura 4.19
- `sudo mn --controller remote --topo tree,depth=4`. La red SDN generada se puede ver en la figura 4.20
- `sudo mn --controller remote --topo linear,4`. La red SDN generada se puede ver en la figura 4.21

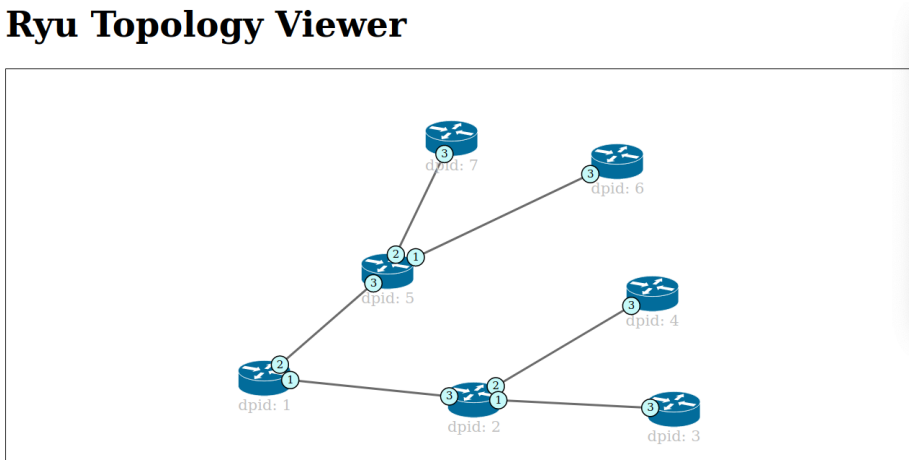


Figura 4.19: Red tree,depth=3 vista desde el visualizador de RYU

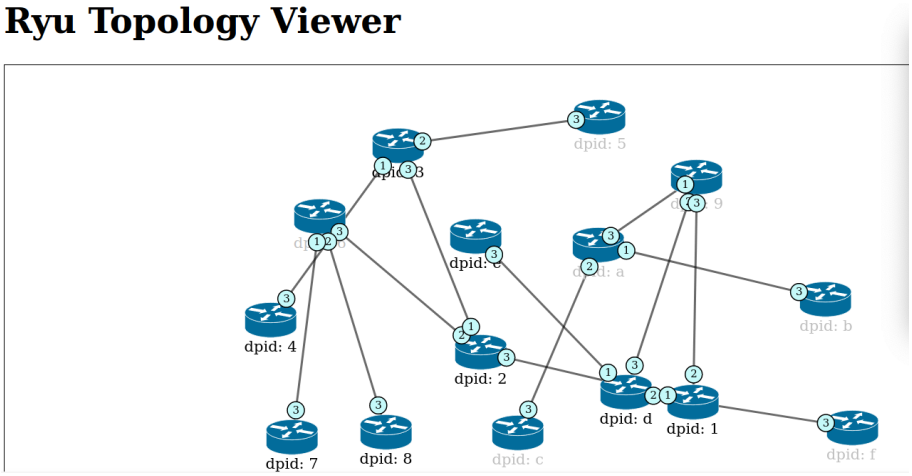


Figura 4.20: Red tree,depth=4 vista desde el visualizador de RYU

- `sudo mn --controller remote --topo linear,8`. La red SDN generada se puede ver en la figura 4.22
- `sudo mn --controller remote --topo torus,3,3`. La red SDN generada se puede ver en la figura 4.23

Como nos interesa coger los diferentes JSON de cada una de las redes hemos creado un archivo HTML en el que hemos definido con la etiqueta `<script>` el código de JavaScript específico para mostrar en la consola el JSON y así poder descargarlo para poder trabajar con él. Es igual que el que utilizamos para recuperar un archivo JSON ya descargado pero, como se puede ver en la figura 4.24, la ruta que le pasamos es la del controlador y lo que buscamos es la información de los links, ya que nos proporciona en un único archivo la información de los

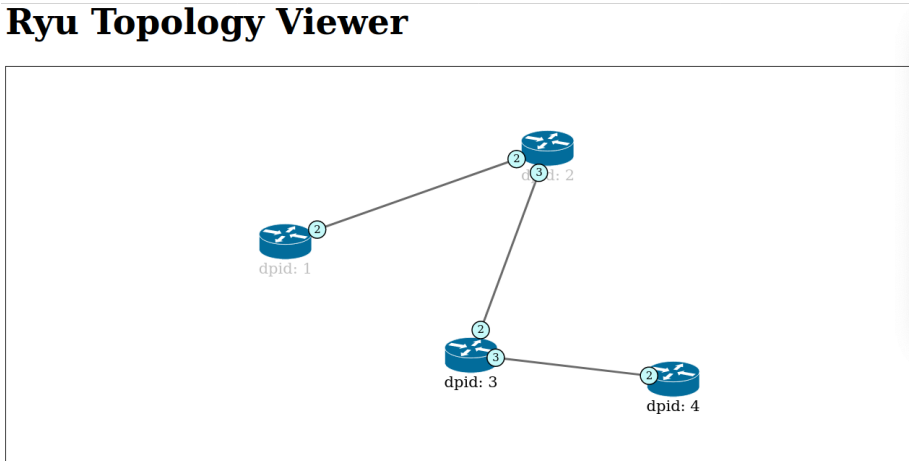


Figura 4.21: Red linear,4 vista desde el visualizador de RYU

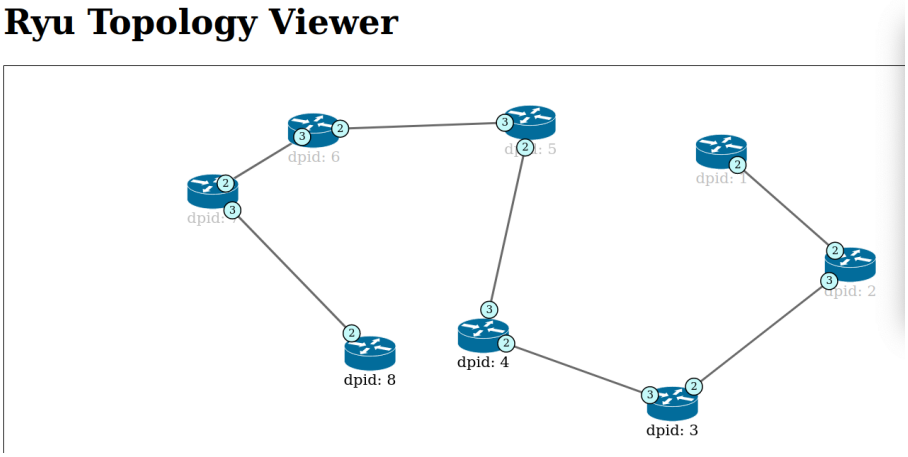


Figura 4.22: Red linear,8 vista desde el visualizador de RYU

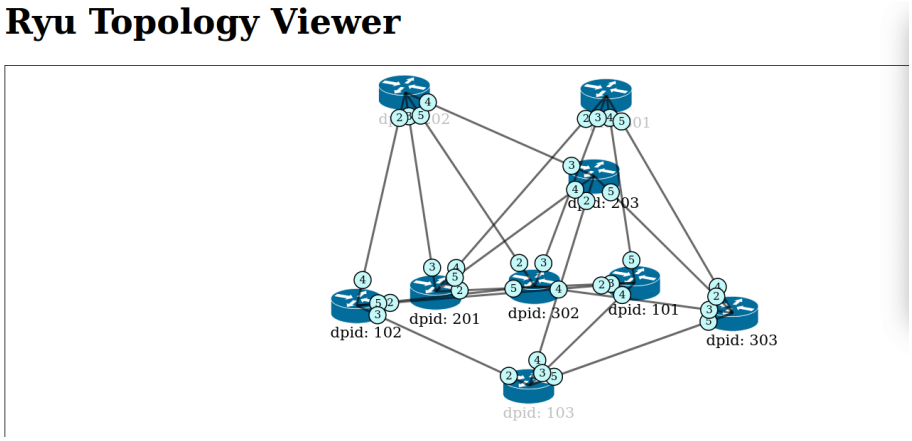


Figura 4.23: Red torus,3,3 vista desde el visualizador de RYU

```
fetchJSONFile('http://localhost:8080/v1.0/topology/links', function(data){
    // do something with your data
    console.log(data);
});
```

Figura 4.24: Función para recuperar un archivo JSON del controlador

diferentes switches que tenemos y cómo están conectados entre sí.

Para comprobar en la consola que se está recuperando el archivo JSON de manera correcta debemos abrir el navegador y escribir la dirección del archivo HTML en el que hemos desarrollado el código para recuperar el archivo JSON, en este caso
`http://localhost:8080/3dGUI.html`

En la figura se puede ver cómo se muestra en la consola el archivo JSON como si fuera un diccionario con clave y valor. Lo haremos para cada una de las redes SDN para poder tener la información fuera de la máquina virtual y que sea más sencillo poder tratar la información para usarla en la generación de nuestra red en A-Frame.

4.6. Análisis de los archivos JSON

Una vez que tenemos descargados fuera de la máquina virtual todos los archivos JSON vamos a analizarlos para ver cuáles son los datos que nos interesa extraer para generar nuestra red en el navegador.

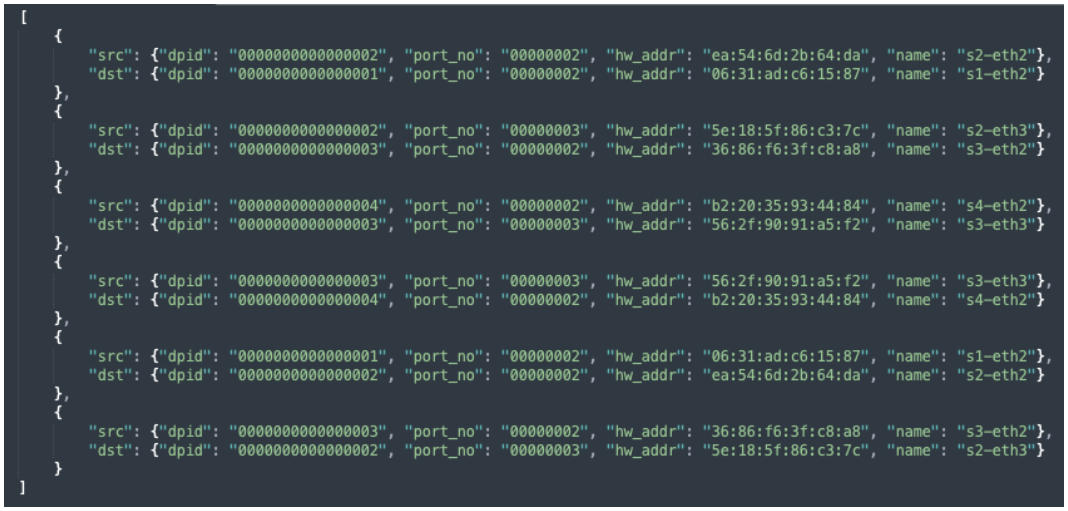
En la figura 4.25 se puede ver el archivo JSON correspondiente a la topología `linear,4`, hemos escogido este archivo en concreto porque esta topología es la que menos elementos tiene, esto hace que sea más sencilla de analizar.

Se pueden apreciar diferentes elementos dentro del JSON, necesitamos saber a qué corresponde cada elemento para ver cuales tenemos que extraer y tratar en nuestra lógica.

Lo primero que apreciamos es que tenemos pares de `"src"-"dst"` que podemos interpretar como pares de origen-destino, es decir, dos elementos que van a estar conectados entre sí.

Cada uno de los elementos de estos pares tienen la misma información asociada a ellos:

- `"dpid"`: este valor se interpreta como un ID asociado al elemento, en este caso los elementos serán switches, por tanto, cada switch tendrá asociado un ID único.



```
[
    {
        "src": {"dpid": "0000000000000002", "port_no": "00000002", "hw_addr": "ea:54:6d:2b:64:da", "name": "s2-eth2"},
        "dst": {"dpid": "0000000000000001", "port_no": "00000002", "hw_addr": "06:31:ad:c6:15:87", "name": "s1-eth2"}
    },
    {
        "src": {"dpid": "0000000000000002", "port_no": "00000003", "hw_addr": "5e:18:5f:86:c3:7c", "name": "s2-eth3"},
        "dst": {"dpid": "0000000000000003", "port_no": "00000002", "hw_addr": "36:86:f6:3f:c8:a8", "name": "s3-eth2"}
    },
    {
        "src": {"dpid": "0000000000000004", "port_no": "00000002", "hw_addr": "b2:20:35:93:44:84", "name": "s4-eth2"},
        "dst": {"dpid": "0000000000000003", "port_no": "00000003", "hw_addr": "56:2f:90:91:a5:f2", "name": "s3-eth3"}
    },
    {
        "src": {"dpid": "0000000000000003", "port_no": "00000003", "hw_addr": "56:2f:90:91:a5:f2", "name": "s3-eth3"},
        "dst": {"dpid": "0000000000000004", "port_no": "00000002", "hw_addr": "b2:20:35:93:44:84", "name": "s4-eth2"}
    },
    {
        "src": {"dpid": "0000000000000001", "port_no": "00000002", "hw_addr": "06:31:ad:c6:15:87", "name": "s1-eth2"},
        "dst": {"dpid": "0000000000000002", "port_no": "00000002", "hw_addr": "ea:54:6d:2b:64:da", "name": "s2-eth2"}
    },
    {
        "src": {"dpid": "0000000000000003", "port_no": "00000002", "hw_addr": "36:86:f6:3f:c8:a8", "name": "s3-eth2"},
        "dst": {"dpid": "0000000000000002", "port_no": "00000003", "hw_addr": "5e:18:5f:86:c3:7c", "name": "s2-eth3"}
    }
]
```


Figura 4.25: Archivo JSON de la topología linear,4

- `"port_no"`: este valor se interpreta como el número de puerto por el que va estar conectado ese elemento a su par.

- `"hw_addr"`: este valor se interpreta como la dirección MAC o dirección física que tiene el puerto. Hay que tener en cuenta que la configuración dentro de la SDN debe ser igual a la que tengamos en el mundo físico para no tener problemas al llevarlo del mundo digital al físico.

- `"name"`: esta valor se interpreta como el nombre o pseudónimo que tiene el puerto, este valor se utiliza para poder acceder de manera sencilla a la información del puerto sin necesidad de tener que utilizar la dirección MAC.

Una vez que hemos analizado los elementos que nos encontramos en el archivo JSON hay que decidir cuáles son interesantes o cuáles necesitamos para poder generar nuestra red en el navegador.

Fundamentalmente necesitaremos dos elementos:

- Necesitamos conocer cuántos elementos vamos a tener que crear en nuestra escena. Para ello nos vamos a quedar con `"dpid"`, como es un valor único por elemento tendremos el número total de elementos que necesitamos en nuestra escena para replicar la red SDN.

- Necesitamos conocer cómo están conectados esos elementos entre sí. Para ello nos vamos a quedar con `"dpid"` de los pares de `"src"-"dst"`. Esto nos indicará qué elemento

```
AFRAME.registerComponent('red', {
  schema: {
    file_links: {type: 'string'}
  },
```

Figura 4.26: Esquema del componente para crear la red

hay que unir con otro, lo que utilizaremos para generar los enlaces.

Es decir, en este caso nos quedaremos con la información que va desde `"dpid"` = 0000000000000001 hasta `"dpid"` = 0000000000000004 y con información de todos los pares `"src"-"dst"` tipo 0000000000000002-0000000000000001. Esto quiere decir que el elemento con `"dpid"` = 0000000000000001 estará unido al elemento con `"dpid"` = 0000000000000002.

Con esto tendremos toda la información que necesitaremos para generar nuestra escena.

4.7. Generación de la escena a partir de los archivos JSON descargados

Una vez que conocemos la estructura de los archivos JSON que hemos recuperado y sabemos la información que necesitamos extraer vamos a pasar a crear la lógica definitiva para crear una escena en el navegador a partir de esa información. Para ello creamos un archivo JavaScript[13].

Como se ve en la figura 4.26 vamos a crear un componente llamado red, el esquema de este componente se basará simplemente en la ruta del archivo que vamos a recuperar, una vez que tengamos esa información pasaremos a crear las entidades.

Al iniciar el componente en el navegador inicializamos las variables que vamos a necesitar como se puede observar en la figura 4.27.

- `data = this.data`: lo vamos a utilizar para acceder a la información que se proporciona en el esquema del componente

[13] `https://github.com/albertoabades/TFG-AFrame/blob/main/assets/js/Generar_Red.js`

```
init: function () {
  let data = this.data
  let array_links =[];
  let array_switches =[];
  let array_switches_final = [];

  const request = new XMLHttpRequest();
  const requestURL = data.file_links;

  request.open('GET', requestURL);
  request.responseType = 'text';
  request.send();
```

Figura 4.27: Esquema del componente para crear la red

- `array_links =[]`: es un array en el que vamos a guardar diferentes arrays con los `"dpid"` de todos pares origen-destino.
- `array_switches =[]`: es un array en el que vamos a guardar los `"dpid"` de los diferentes switches que va a tener nuestra red.
- `array_switches_final = []`: es un array en el que vamos a guardar diferentes arrays con el `"dpid"` y las coordenadas que vamos a generar para cada uno.

Utilizaremos estos arrays para obtener la información que vamos a utilizar para generar nuestra escena.

Una vez que tenemos inicializadas las variables que vamos a necesitar preparamos la llamada para recuperar el archivo JSON. Para ello tenemos que inicializar un objeto `request = new XMLHttpRequest()`, este objeto nos facilitará obtener información de un path sin necesidad de tener que recargar la página.

También necesitaremos el path sobre el que hacer la llamada, este información la recuperamos del parámetro que se pasa al componente en su creación, el que hemos definido en el esquema. Por eso utilizamos `data.file_links`.

Ahora tenemos que iniciar la llamada utilizamos el método `open()` al que le pasamos el parámetro `GET`, que es el método HTTP que vamos a usar para la llamada y el parámetro `requestURL` que es el path sobre el que vamos a realizar la llamada. Definimos el tipo de respuesta que vamos a obtener, en este caso, lo definimos como un texto plano utilizando el método

`responseType`[15] de `XMLHttpRequest()`. Ya tenemos definido todo lo necesario, por lo que lo único que nos queda es enviar la petición con el método `send()`.

Tras comprobar que la petición ha ido correctamente se accede al método `onload`[13] y empezamos a trabajar con la información recibida. Lo primero que hacemos es guardar la respuesta con el método `response`[14] y lo parseamos con el método `JSON.parse()` para convertirlo en un objeto de JavaScript y poder trabajar con él.

4.7.1. Guardar la información de los switches y enlaces

Lo primero que hacemos es guardar la información que obtenemos del JSON en los arrays que hemos inicializado al principio, como se puede ver en la figura 4.28. Lo hacemos generando un bucle para poder acceder a toda la información.

Para ello tenemos que acceder al objeto de JavaScript en el que hemos convertido la respuesta JSON, guardamos el `"dpid"` del switch origen en una variable accediendo a él con la sentencia `links[i].src.dpid`, del mismo modo guardamos en otra variable el `"dpid"` del destino accediendo a él con la sentencia `links[i].dst.dpid`. Como necesitamos conocer la dupla origen-destino para poder generar los enlaces vamos a inicializar el array `origen_destino = []`. Estos arrays los iremos guardando en el array `array_links` que luego utilizaremos para generar los enlaces entre los switches.

Una vez que tenemos el `"dpid"` del origen y del destino lo guardamos en el array `array_switches` asegurándonos que sólo se guarda el valor una única vez y no están repetidos. La información de este array la usaremos luego para generar las coordenadas de cada switch.

4.7.2. Generar las coordenadas de cada switch

Ya conocemos el número de switches y de enlaces que vamos a tener que generar, ahora hay que definir las coordenadas que va a tener cada switch, como no conocemos el número de switches que vamos a tener generamos las coordenadas X y Z de manera aleatoria como se puede observar en la figura 4.29.

Creamos un bucle para recorrer el array `array_switches` en el que tenemos guardados los `"dpid"` de cada switch. Lo primero que hacemos en el bucle es inicializar el array `switch_coordenadas = []` en el que vamos a guardar el `"dpid"` y las coordenadas X,

```
for (let i = 0; i < links.length; i++) {
    let origen = links[i].src.dpid;
    let destino = links[i].dst.dpid;
    let origen_destino = [];
    origen_destino.push(origen);
    origen_destino.push(destino);
    array_links.push(origen_destino);

    const existe_origen = array_switches.includes(origen);
    if (existe_origen == false){
        array_switches.push(origen);

    }
    const existe_destino = array_switches.includes(destino);
    if (existe_destino == false){
        array_switches.push(destino);
    }

}
```

Figura 4.28: Completar los arrays con la información del JSON

```
for(let i = 0; i < array_switches.length; i++){
    let switch_coordenadas = [];
    switch_coordenadas.push(array_switches[i]);
    switch_coordenadas.push(Math.floor(Math.random() * ((array_switches.length+3) - (-3)) + (-3)));
    switch_coordenadas.push(0.5);
    switch_coordenadas.push(Math.floor(Math.random() * ((array_switches.length+3) - (-3)) + (-3)));
    array_switches_final.push(switch_coordenadas);
}
```

Figura 4.29: Generar las coordenadas de cada switch

Y, Z. Ese array que hemos creado lo incluimos en el array `array_switches_final` que inicializamos al principio.

Para generar las coordenadas X, Z lo hacemos con las funciones matemáticas `Math.floor()` y `Math.random()`. La primera se utiliza para redondear el número obtenido con la segunda. Para que nos dé un abanico más amplio de valores la función `Math.random()` utilizamos como valor máximo el número de switches que tenemos y lo ampliamos en tres unidades y el mínimo le ponemos un menos tres. Esto nos da un rango de más seis sobre el número de switches que tenemos.

El valor de la coordenada Y será siempre el mismo, ya que en este caso queremos que estén todos a la misma altura del plano.

4.7.3. Generar los switches e incluirlos en la escena

Ya tenemos toda la información que necesitamos para generar nuestra escena, por tanto vamos a comenzar a hacerlo. Lo primero que vamos a incluir en nuestra escena son los switches.

```
for (let i = 0; i < array_switches_final.length; i++) {
    var scene = document.querySelector('a-scene');
    var router = document.createElement('a-entity');
    var_x = array_switches_final[i][1];
    var_y = array_switches_final[i][2];
    var_z = array_switches_final[i][3];
    router.setAttribute('ID',array_switches_final[i][0]);
    router.setAttribute('geometry', {primitive: 'box'});
    router.setAttribute('material', 'color', 'blue');
    router.setAttribute('position', {x: var_x, y: var_y, z: var_z});
    router.setAttribute('scale', {x: 0.5, y: 0.5, z: 0.5});
    scene.appendChild(router);
}
```

Figura 4.30: Configuración de los atributos de cada switch

Como se puede ver en la figura 4.30 generamos un bucle sobre el array `array_switches_final` para ir sacando toda la información.

Lo primero que hacemos es recuperar la escena del DOM para incluir las diferentes entidades que vamos a crear. Tras esto creamos una entidad genérica a la que luego le vamos a configurar los diferentes atributos para definirla. Recuperamos del array `array_switches_final` las coordenadas X, Y, Z y guardamos cada una en una variable. Recuperamos el valor de `"dpid"` que hemos guardado en la primera posición de cada uno de los arrays que componen el array `array_switches_final` y utilizando el método `setAttribute` lo guardamos en el atributo `ID`.

Utilizamos el mismo método para configurar una geometría primitiva en la entidad, en este caso, una caja, que será la forma que le daremos a nuestros switches en la escena. También le asignaremos un color, en lugar de darle un valor hexadecimal hemos optado por pasarle el texto de uno de los colores básicos `blue` al atributo `Color`

Ahora es cuando utilizamos las variables en las que hemos guardado las coordenadas para asignárselas al atributo `Position`. Para que las cajas que vamos a crear no sean demasiado grandes vamos a asignarle al atributo `Scale` el valor 0.5 para cada coordenada, esto hará que el valor por defecto que tiene ese atributo sea la mitad.

Ahora que tenemos configurados todos los atributos de la entidad que hemos creado la incluimos en la escena como un hijo, ya que el DOM trata los elementos con una estructura de árbol.

4.7.4. Generar los enlaces e incluirlos en la escena

Ahora que hemos tenemos todos los switches incluidos en nuestra escena vamos a incluir los enlaces entre ellos, como se puede ver en la figura 4.31, generamos un bucle sobre el array `array_links` en el que tenemos los pares de origen-destino, accedemos al `"dpid"` del origen con la sentencia `switch_origen = array_links[i][0]` y del destino con la sentencia `switch_destino = array_links[i][1]`. Una vez que tenemos los `"dpid"` hacemos dos bucles sobre `array_switches_final` para obtener las coordenadas X, Y, Z de los dos switches y las guardaremos por separado en diferentes variables.

Para los enlaces vamos a utilizar una entidad especial que se llama `tube`, esta entidad no entra dentro de las entidades primitivas sino que pertenece a las entidades que han ido creando los usuarios para ampliar las capacidades que puede ofrecer A-Frame.

Para generar esta entidad necesitamos las coordenadas donde empieza y las coordenadas donde termina, así como la longitud del radio. Al ser una entidad especial las coordenadas no se guardan en un atributo haciendo diferencia entre X, Y, Z sino que hay que generar una cadena de texto con las coordenadas de origen y otra cadena de texto con las coordenadas de destino. Para ello concatenamos las coordenadas que hemos recuperado en el array `array_switches_final` con las sentencias que se guardan en las variables `coordOrigen` y `coordDestino`.

Extraemos la escena del DOM para poder incluir en ella los diferentes enlaces que vamos a crear. Una vez hecho creamos una identidad genérica que configuraremos como una entidad `tube`. En la asignación del tipo `tube` debemos rellenar la propiedad `path` con un array al que le pasamos como atributos las variables donde hemos guardado las cadenas de texto con las coordenadas de origen y las coordenadas de destino. También definimos la longitud del radio intentando que no sea una longitud muy grande para que no nos queden unos enlaces demasiado gruesos.

Ya sólo nos queda definir el color y la opacidad del enlace, de nuevo para el color optamos por pasarle como parámetro el texto de uno de los colores básicos (red, black, cyan...) en lugar de pasarle un valor hexadecimal. Hemos decidido que el enlace sea totalmente opaco, cuanto menor sea el valor más translúcido será la entidad, en este caso, el enlace.

Ahora que tenemos configurados todos los atributos de la entidad que hemos creado la incluimos en la escena como un hijo. Lo que dejará nuestra escena completamente creada.

```
for(let i = 0; i < array_links.length; i++){
    let switch_origen = array_links[i][0];
    let switch_destino = array_links[i][1];
    for(let i = 0; i < array_switches_final.length; i++){
        if(switch_origen == array_switches_final[i][0]){
            coordXOrigen = array_switches_final[i][1];
            coordYOrigen = array_switches_final[i][2];
            coordZOrigen = array_switches_final[i][3];
        }
        if(switch_destino == array_switches_final[i][0]){
            coordXDestino = array_switches_final[i][1];
            coordYDestino = array_switches_final[i][2];
            coordZDestino = array_switches_final[i][3];
        }
    }
    coordOrigen = String(coordXOrigen) + " " + String(coordYOrigen) + " " + String(coordZOrigen);
    coordDestino = String(coordXDestino) + " " + String(coordYDestino) + " " + String(coordZDestino);
    var enlace = document.createElement('a-entity');
    var scene = document.querySelector('a-scene');
    enlace.setAttribute('tube', {'path': [coordOrigen, coordDestino] , 'radius': '0.05' });
    enlace.setAttribute('material', 'color', 'red');
    enlace.setAttribute('material', 'opacity', '1');
    scene.appendChild(enlace);
}
```

Figura 4.31: Configuración de los atributos de cada enlace

4.8. Un archivo HTML para cada SDN

Como hemos tenido que descargar los diferentes archivos JSON para cada una de las SDN que hemos preparado necesitamos un HTML diferente para representar cada escena, ya que tenemos que pasar la ruta de cada uno de los archivos JSON.

Los archivos que HTML que hemos creado son los siguientes:

- Para la SDN tree,depth=3 hemos creado el archivo HTML[14] que se puede ver en la figura 4.32.
- Para la SDN tree,depth=4 hemos creado el archivo HTML[15] que se puede ver en la figura 4.33.
- Para la SDN linear,4 hemos creado el archivo HTML[16] que se puede ver en la figura 4.34.
- Para la SDN linear,8 hemos creado el archivo HTML[17] que se puede ver en la figura 4.35.

[14] https://github.com/albertoabades/TFG-AFrame/blob/main/red_final_tree_depth3.html

[15] https://github.com/albertoabades/TFG-AFrame/blob/main/red_final_tree_depth4.html

[16] https://github.com/albertoabades/TFG-AFrame/blob/main/red_final_linear4.html

[17] https://github.com/albertoabades/TFG-AFrame/blob/main/red_final_linear8.html

```
<html>
  <head>
    <script src="../assets/js/aframe.min.js"></script>
    <script src="../assets/js/aframe-extras.min.js"></script>
    <script src="../assets/js/Generar_Red.js"></script>
  </head>
  <body>
    <a-scene inspector="https://cdn.jsdelivr.net/gh/aframevr/aframe-inspector@master/dist/aframe-inspector.min.js">
      <a-entity red="file_links: links_topo-tree-depth3.json"></a-entity>
    </a-scene>
  </body>
</html>
```

Figura 4.32: HTML para representar la SDN tree,depth=3

```
<html>
  <head>
    <script src="../assets/js/aframe.min.js"></script>
    <script src="../assets/js/aframe-extras.min.js"></script>
    <script src="../assets/js/Generar_Red.js"></script>
  </head>
  <body>
    <a-scene inspector="https://cdn.jsdelivr.net/gh/aframevr/aframe-inspector@master/dist/aframe-inspector.min.js">
      <a-entity red="file_links: links_topo-tree-depth4.json"></a-entity>
    </a-scene>
  </body>
</html>
```

Figura 4.33: HTML para representar la SDN tree,depth=4

- Para la SDN torus,3,3 hemos creado el archivo HTML[18] que se puede ver en la figura 4.36.

Como se puede apreciar todos los archivos son iguales, la única diferencia es el path que le pasamos al componente `red` para hacer la llamada `GET`. Para la versión final de nuestros HTML hemos decidido descargar y guardar los archivos `aframe.min.js` y `aframe-extras.min.js`, así no dependemos de la conexión a internet para poder cargar nuestras escenas con A-Frame y podemos cargarlas de manera offline desde nuestro servidor.

Aprovechando este cambio hemos creado una estructura en la que guardar todos nuestros archivos de JavaScript y así tener todo mucho más ordenado[3]. En esa misma carpeta hemos guardado el archivo JavaScript que utilizamos para generar nuestros switches y enlaces e in-

[18] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_final_torus33.html`

```
<html>
  <head>
    <script src="../assets/js/aframe.min.js"></script>
    <script src="../assets/js/aframe-extras.min.js"></script>
    <script src="../assets/js/Generar_Red.js"></script>
  </head>
  <body>
    <a-scene inspector="https://cdn.jsdelivr.net/gh/aframevr/aframe-inspector@master/dist/aframe-inspector.min.js">
      <a-entity red="file_links: links_topo-linear-4.json"></a-entity>
    </a-scene>
  </body>
</html>
```

Figura 4.34: HTML para representar la SDN linear,4

```
<html>
  <head>
    <script src="../assets/js/aframe.min.js"></script>
    <script src="../assets/js/aframe-extras.min.js"></script>
    <script src="../assets/js/Generar_Red.js"></script>
  </head>
  <body>
    <a-scene inspector="https://cdn.jsdelivr.net/gh/aframevr/aframe-inspector@master/dist/aframe-inspector.min.js">
      <a-entity red="file_links: links_topo-linear-8.json"></a-entity>
    </a-scene>
  </body>
</html>
```

Figura 4.35: HTML para representar la SDN linear,8

```
<html>
  <head>
    <script src="../assets/js/aframe.min.js"></script>
    <script src="../assets/js/aframe-extras.min.js"></script>
    <script src="../assets/js/Generar_Red.js"></script>
  </head>
  <body>
    <a-scene inspector="https://cdn.jsdelivr.net/gh/aframevr/aframe-inspector@master/dist/aframe-inspector.min.js">
      <a-entity red="file_links: links_topo-torus3-3.json"></a-entity>
    </a-scene>
  </body>
</html>
```

Figura 4.36: HTML para representar la SDN torus,3,3

cluirlos en la escena.

4.9. Archivo HTML para integrar con Mininet

Una vez que hemos comprobado que nuestra lógica funciona y hemos visto que cada una de las escenas se genera correctamente, debemos integrarlo con Mininet, que es el objetivo de este proyecto. Poder representar cualquier SDN con un único archivo HTML.

Para ello hemos creado el archivo HTML[19] que se puede observar en la figura 4.37. En este archivo HTML vemos que la ruta que le pasamos es a una API que nos devuelve un archivo JSON en el puerto donde hemos dejado escuchando al controlador `(http://localhost:8080/v1.0/topology/links)`.

Con esto hacemos que la información sea la de la SDN que está arrancada en ese momento, por tanto, no hay nada predefinido sino que podemos representar cualquier tipo de SDN. Por lo tanto con este pequeño cambio integramos nuestra representación de SDN en A-Frame con Mininet donde arrancamos las diferentes SDN.

[19] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_final.html`

```
<html>
  <head>
    <script src="aframe.min.js"></script>
    <script src="aframe-extras.min.js"></script>
    <script src="Generar_Red.js"></script>
  </head>
  <body>
    <a-scene inspector="https://cdn.jsdelivr.net/gh/aframevr/aframe-inspector@master/dist/aframe-inspector.min.js">
      <a-entity red="file_links: http://localhost:8080/v1.0/topology/links"></a-entity>
    </a-scene>
  </body>
</html>
```

Figura 4.37: HTML genérico para representar cualquier SDN

Capítulo 5

Experimentos y validación

Para validar nuestro proyecto hemos probado a generar la visualización de 5 escenas diferentes. Para generar las 5 escenas hemos debido arrancar las diferentes SDN en Mininet y el controlador de RYU. Tras esto abrimos el navegador y cargamos el archivo HTML sobre el puerto en el que hemos puesto a escuchar al controlador.

Esas 5 escenas son las siguientes:

1. En la figura 5.1 podemos ver la representación de una red lineal con 4 switches.

2. En la figura 5.2 podemos ver la representación de una red lineal con 8 switches

3. En la figura 5.3 podemos ver la representación de una red torus de 3x3

4. En la figura 5.4 podemos ver la representación de una red en forma de árbol con 3 niveles

5. En la figura 5.5 podemos ver la representación de una red en forma de árbol con 4 niveles

Lo bueno que tienen estas representaciones es que podemos movernos a través de ellas en el navegador con los botones de dirección, por lo que podemos aproximarnos a los diferentes switches y seguir los enlaces para ver a qué otros switches están conectados. Lo mismo podemos, hacer de una manera mucho más inmersiva, si utilizamos unas gafas de realidad virtual.

Para ver la diferencia y la mejora que supone generar las redes en 3D vamos a comparar una de las redes que hemos generado en el navegador en 3D con la representación en 2D que genera el visualizador del controlador RYU, que se puede observar en la figura 5.6.

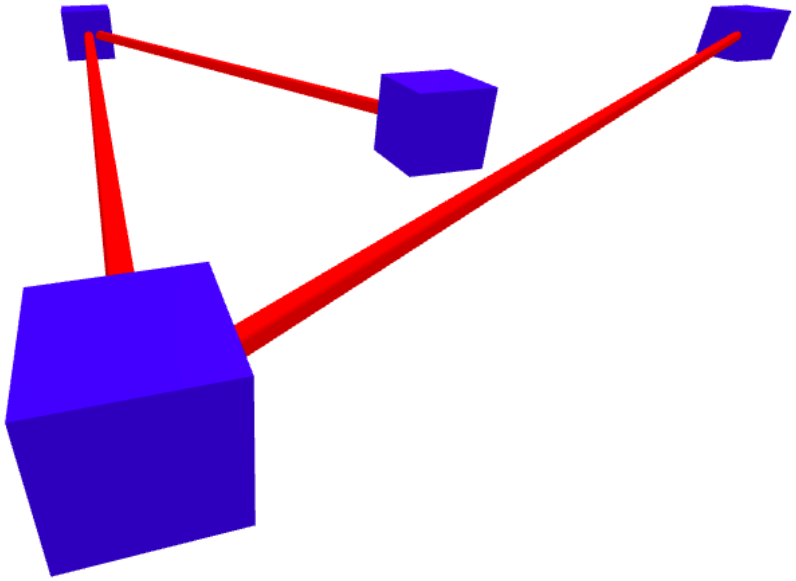


Figura 5.1: Representación de una red lineal con 4 switches

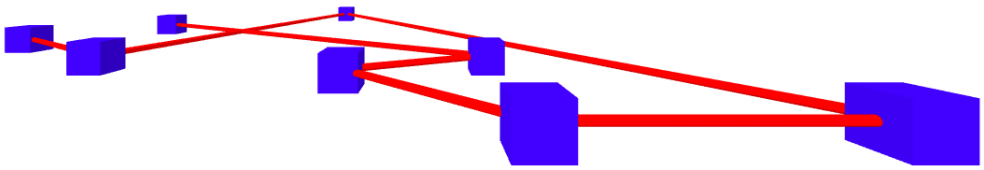


Figura 5.2: Representación de una red lineal con 8 switches

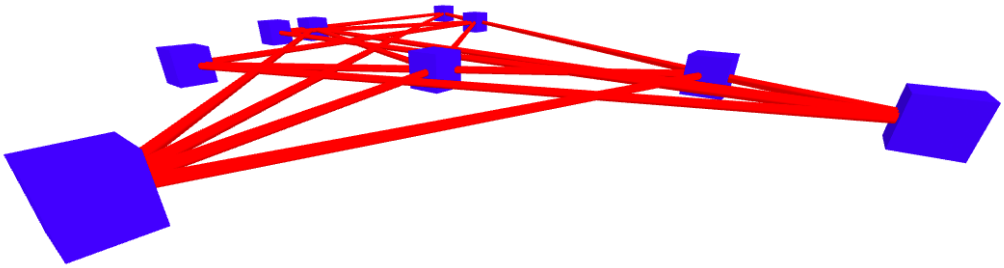


Figura 5.3: Representación de una red torus 3x3

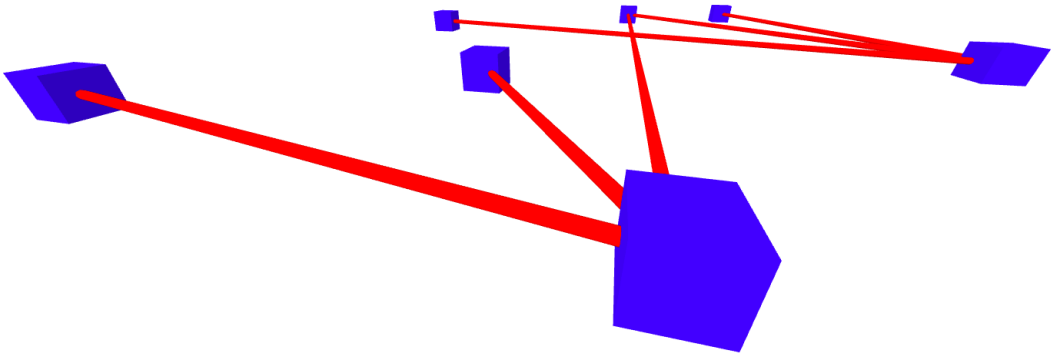


Figura 5.4: Representación de una red con forma de árbol con 3 niveles

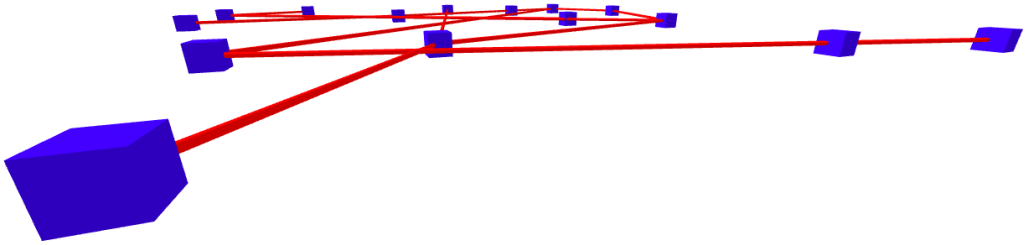


Figura 5.5: Representación de una red con forma de árbol con 4 niveles

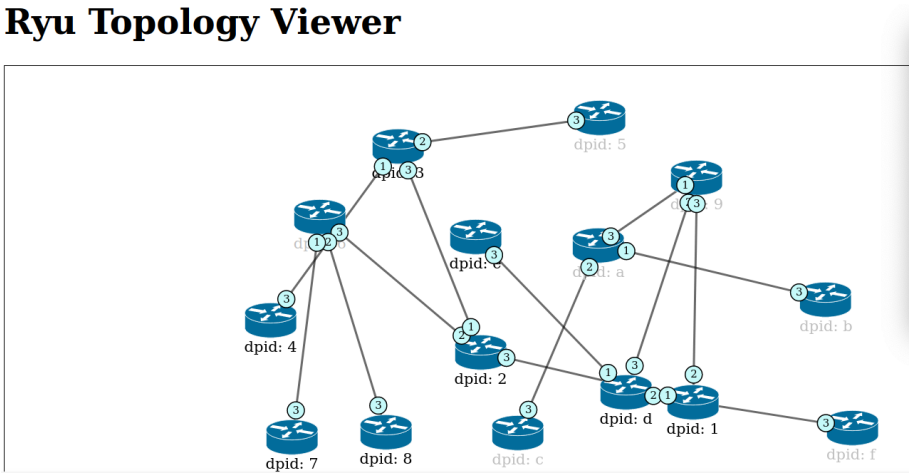


Figura 5.6: Red tree,depth=4 vista desde el visualizador de RYU

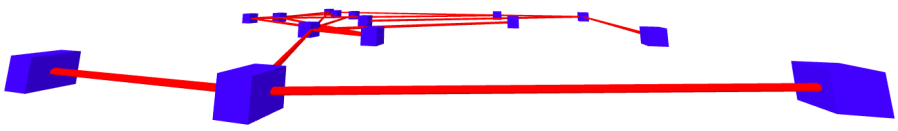


Figura 5.7: Red tree,depth=4 vista en 3D

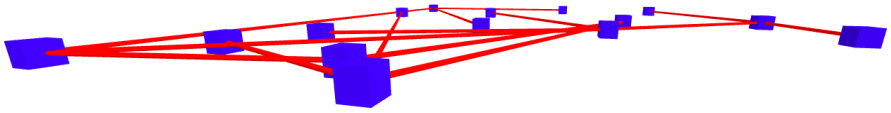


Figura 5.8: Red tree,depth=4 vista en 3D desde otra perspectiva

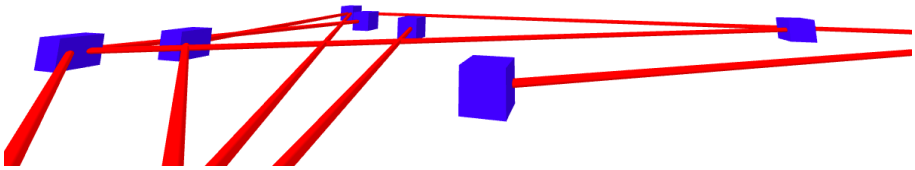


Figura 5.9: Aproximación a un elemento de la Red tree,depth=4

La figura 5.7 es la misma red generada en 3D.

Como se puede observar en la figura 5.8 otra de las ventajas que tiene generar las redes en 3D es que podemos movernos hacia cualquier punto para ver la red desde una perspectiva diferente.

Además en la red que hemos generado en 3D podemos movernos a través de ella para acercarnos a alguno de los elementos y seguir los enlaces para ver más fácilmente a qué otros elementos está conectado, como se puede observar en la figura 5.9.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

En este proyecto teníamos dos objetivos principales.

El primero de ellos era representar una red de switches en el navegador utilizando la tecnología A-Frame para poder llevarlo también a la Realidad Virtual.

Este objetivo ha quedado alcanzado con creces ya que con el mismo código hemos representado 5 redes diferentes a partir de la información de varios archivos JSON que contenían información de esas redes. Para esto hemos tenido que crear cinco archivos HTML diferentes porque teníamos que llamar por separado los diferentes archivos.

El segundo objetivo que tenía este proyecto era integrar la representación de las redes SDN con Mininet.

Este objetivo también se ha alcanzado puesto que una vez arrancamos en Mininet cada una de las SDN podemos representarlas en el navegador con el mismo HTML, recuperando la información de la red del puerto en el que está escuchando el controlador que hemos arrancado previamente.

6.2. Aplicación de lo aprendido

Para abordar este trabajo partía de la base de lo aprendido en las diferentes asignaturas de programación que he cursado a lo largo del grado. Todas me han servido ya que me han dado los conocimientos básicos de programación para poder utilizar cualquier tipo de lenguaje,

ya sea un lenguaje orientado a objetos o un lenguaje secuencial. Pero la asignatura de programación que más me ha ayudado es la asignatura de DESARROLLO DE APLICACIONES TELEMATICAS en la que nos enseñaron a modificar el DOM, ya fuera desde un archivo HTML, un archivo JavaScript o incluso poder darle una mejor apariencia con CSS.

También han sido fundamentales las asignaturas de análisis y creación de redes, en concreto la asignatura de SISTEMAS TELEMATICOS que me dio los conocimientos necesarios de análisis de redes y de protocolos, así como identificar la información de los diferentes switches y enlaces para saber para qué sirve cada uno de los parámetros de configuración de las redes.

6.3. Lecciones aprendidas

A lo largo de este proyecto he podido profundizar en una tecnología como es A-Frame, antes de abordar el proyecto esta tecnología era totalmente desconocida para mí por lo que todo ha sido adquisición de nuevos conocimientos. He podido comprobar de primera mano que A-Frame es una tecnología muy potente, con unas posibilidades enormes pero a su vez con una estructura muy sencilla y muy fácil de aplicar. Con unas pocas líneas de código se pueden conseguir funcionalidades complejas y que son fácilmente escalables.

Además no sólo sirve para representar cosas sobre el navegador, si no que se puede extrapolar a la realidad virtual dando funcionalidad específica para ese mundo de manera sencilla y también compatible con la funcionalidad dentro del navegador.

Este proyecto también me ha servido para conocer dos tecnologías como son Mininet y RYU. La primera de ellas se utiliza para generar y trabajar con diferentes SDN, no he tenido la oportunidad de profundizar en exceso dentro de la tecnología ya que no he generado desde cero las diferentes SDN pero si he podido trabajar con ellas, analizando y modificando las diferentes SDN que iba a representar.

El controlador RYU me ha servido para poder poner a escuchar en un puerto y así recuperar la información de las diferentes SDN con las que he trabajado. RYU también tiene un visualizador que me ha servido para poder comparar la representación que hace de las SDN con las que he realizado con la tecnología A-Frame y así comprobar que mi lógica generaba el número correcto de switches y que estaban conectados entre sí de la manera correcta.

6.4. Trabajos futuros

Algunas de las evoluciones sencillas que creo que podría tener este proyecto sería cambiar la visualización para que no sea tan aséptico, es decir, en lugar de utilizar geometrías primitivas de A-Frame utilizar imágenes de tipo GLTF con algún tipo de animación, poner alguna imagen en el entorno para que no tenga un aspecto más sofisticado o poner algún tipo de cartel en los switches para poder identificarlos sin necesidad de acceder al inspector.

Otra evolución, en este caso más compleja, sería recuperar información sobre el intercambio de paquetes entre los switches y pintarlo en la escena, así podría verse el camino que llevan los paquetes desde un switch a otro.

También podría desarrollarse algún tipo de lógica para que se puedan parar o arrancar los switches directamente desde el navegador, en lugar de tener que ir al terminal de Mininet donde tenemos arrancada la SDN. Esto generaría una reconfiguración de la red y, uniéndolo con la posible mejora explicada anteriormente, se podría ver cómo cambian de camino los diferentes paquetes al haber cambiado la topología de la red de switches activos.

Apéndice A

Manual de usuario

Hemos trabajado con Mininet y RYU sobre una máquina virtual. Por tanto lo primero sería configurarla.

Una vez que lo tenemos disponible y funcionando debemos incluir nuestro código en la ruta donde escucha el controlador de RYU.

En nuestro caso la ruta es:
`/Escritorio/venv/lib/python3.8/site-packages/ryu/app/gui_topology/html/`

En esa ruta debemos incluir los tres archivos JavaScript que hacen funcionar nuestra lógica y el archivo HTML[1] en el que incluimos la información para hacer la llamada al controlador de RYU. Debido a los permisos de navegación que tienen las carpetas de la ruta en la que tenemos que guardar nuestros archivos no podemos generar una jerarquía de carpetas como sí hacemos en local.

Los archivos JavaScript que tenemos que incluir en esa ruta son:

- `aframe-extras.min.js`[2]
- `aframe.min.js`[3]
- `Generar_Red.js`[4]

[1] `https://github.com/albertoabades/TFG-AFrame/blob/main/red_final.html`
[2] `https://github.com/albertoabades/TFG-AFrame/blob/main/aframe-extras.min.js`
[3] `https://github.com/albertoabades/TFG-AFrame/blob/main/aframe.min.js`
[4] `https://github.com/albertoabades/TFG-AFrame/blob/main/Generar_Red.js`

Una vez que tenemos todo nuestro código en la ruta correcta activamos el controlador con los siguientes comandos en el terminal

- `source /Escritorio/venv/bin/activate`
- `cd /Escritorio/venv/lib/python3.8/site-packages`
- `ryu run --observe-links ryu/app/gui_topology/gui_topology.py`

Ya tenemos el controlador escuchando en el puerto 8080 por lo que sólo nos queda arrancar las diferentes redes en Mininet. Sólo una cada vez. Para ello usamos los siguientes comandos en un terminal diferente al que estamos usando para tener arrancado el controlador.

- `sudo mn --controller remote --topo tree,depth=3`
- `sudo mn --controller remote --topo tree,depth=4`
- `sudo mn --controller remote --topo linear,4`
- `sudo mn --controller remote --topo linear,8`
- `sudo mn --controller remote --topo torus,3,3`

Al ser un comando `sudo` nos solicitará la contraseña de administrador.

Si queremos apagar una red para arrancar otra lo único que tenemos que hacer es acudir al terminal donde la tenemos arrancada y escribir el comando `exit`. Después ejecutamos de nuevo uno de los comandos de la red que queremos arrancar.

Ya sólo nos queda arrancar el navegador para visualizar la red y para ello tenemos dos opciones

- La primera opción es escribir la URL `http://localhost:8080/` esto nos mostrará la red con el visualizador de RYU.
- La segunda opción es escribir la URL `http://localhost:8080/red_final.html` lo cual hará que se muestre la visualización en 3D la red que tenemos arrancada.

Bibliografía

[1] Three.js fundamentals, Consultada por última vez el 15/7/21.
https://threejsfundamentals.org/threejs/lessons/threejs-fundamentals.

[2] DeuSens. 10 hitos en la historia de la vr, Consultada por última vez el 15/7/21.
https://www.deusens.com/hitos-historia-realidad-virtual/.

[3] J. M. González-Barahona. A playground for learning about a-frame, Consultada por última vez el 15/7/21.
https://jgbarah.github.io/aframe-playground/.

[4] S. Lau. The spinning cube of potential doom. *COMMUNICATIONS OF THE ACM*, 2004.

[5] E. L. Malécot, M. Kohara, Y. Hori, and K. Sakurai. Interactively combining 2d and 3d visualization for network traffic monitoring. 2006.

[6] Mozilla. Javascript, Consultada por última vez el 15/7/21.
https://developer.mozilla.org/es/docs/Web/JavaScript.

[7] Mozilla. Json.parse(), Consultada por última vez el 15/7/21.
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse.

[8] Mozilla. Política same-origin, Consultada por última vez el 15/7/21.
https://developer.mozilla.org/es/docs/Web/Security/Same-origin_policy.

[9] Mozilla. A web framework for building 3d/ar/vr experiences, Consultada por última vez el 15/7/21.
https://aframe.io/.

[10] Mozilla. Webgl, Consultada por última vez el 15/7/21. https://developer.mozilla.org/es/docs/Web/API/WebGL_API.

[11] Mozilla. Webxr device api, Consultada por última vez el 15/7/21. https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API.

[12] Mozilla. Xmlhttprequest, Consultada por última vez el 15/7/21. https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest.

[13] Mozilla. Xmlhttprequesteventtarget.onload, Consultada por última vez el 15/7/21. https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequestEventTarget/onload.

[14] Mozilla. Xmlhttprequest.response, Consultada por última vez el 15/7/21. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/response.

[15] Mozilla. Xmlhttprequest.responsetype, Consultada por última vez el 15/7/21. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseType.

[16] J. Nicholls, D. Peters, A. Slawinski, T. Spoor, S. Vicol, J. Happa, M. Goldsmith, and S. Creese. Netvis: a visualization tool enabling multiple perspectives of network traffic data. *EG UK Theory and Practice of Computer Graphics*, 2013.

[17] Y. Okada. Network data visualization using parallel coordinates version of time-tunnel with 2dto2d visualization for intrusion detection. 2013.

[18] L. Shi, Q. Liao, X. Sun, Y. Chen, and C. Lin. Scalable network traffic visualization using compressed graphs. 2013.

[19] D. W. H. Ten, S. Manickam, S. Ramadass, and H. A. A. Bazar. Study on advanced visualization tools in network monitoring platform. 2009.

[20] WHATWG. Html living standard, Consultada por última vez el 15/7/21. https://html.spec.whatwg.org.