Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

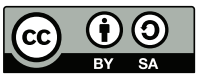
TRABAJO FIN DE GRADO

SIMULADOR DE REDES DE ORDENADORES EN REALIDAD AUMENTADA

Autor : Miguel Cerdeño Gutiérrez

Tutor : Jesús María González Barahona

Curso académico 2025/2026



Dedicado a mis padres y mi hermana

II

Agradecimientos

Quiero agradecer, en primer lugar, a mis padres, ellos son los que han hecho posible que pueda finalizar este grado. Admiro su paciencia conmigo, sus ganas de ayudarme y apoyarme en todo lo posible y todos los sacrificios que han realizado por mí. Han sido fundamentales durante todo el proceso. En segundo lugar quiero agradecer a mi hermana, por su paciencia infinita y por ser un apoyo desde hace años para todo lo que he necesitado. También agradezco a mis amigos, tanto del grado como de fuera, que han servido de vía de escape y desconexión. Y por último, pero no menos importante, a Jesús María González Barahona, tutor de este trabajo, quien lo ha conducido de la mejor manera, siempre entendiendo los tiempos que he necesitado, con paciencia y disponible para ayudar en todo lo posible, la mejor elección.

Resumen

El análisis del tráfico de red es una competencia clave para un ingeniero de telecomunicaciones, sin embargo, las herramientas tradicionales de inspección de tráfico presentan la información de una manera textual, que dificulta la comprensión espacial y temporal del comportamiento de los paquetes en una topología de red.

En este Trabajo de Fin de Grado se presenta el diseño e implementación de una aplicación software para la visualización del flujo de paquetes en topologías de red mediante tecnologías de realidad extendida. La solución procesa automáticamente ficheros JSON de capturas de tráfico y su topología, que se pueden obtener de otras herramientas, para representar en un entorno tridimensional la estructura de la red y la evolución temporal de los paquetes transmitidos.

Esta herramienta permite reproducir capturas de tráfico sobre topologías arbitrarias, visualizar rutas de paquetes, consultar información relevante de la captura y controlar temporalmente la reproducción de la simulación, así como funcionalidades interactivas para el usuario. La herramienta propuesta ha sido implementada empleando A-Frame como motor de visualización XR, lo que permite usarla tanto en dispositivos XR como en un navegador de escritorio.

Como resultado, se obtiene un prototipo orientado a la docencia y demostración de conceptos de redes de ordenadores.

Summary

Network traffic analysis is a key skill for a telecommunications engineer; however, traditional traffic inspection tools present the information in text form, which makes it difficult to understand the spatial and temporal behavior of packets within a network topology.

This Final Degree's Project presents the design and implementation of a software application for visualizing packet flow in network topologies using extended reality technologies. The solution automatically processes JSON files containing traffic captures and their topology – which can be obtained from other tools – to represent the network structure and the temporal evolution of transmitted packets in a three-dimensional environment.

This tool allows traffic captures to be played back over arbitrary topologies, packet routes to be visualized, relevant information from the capture to be viewed, and the playback of the simulation to be controlled over time, as well as providing interactive features for the user. The proposed tool has been implemented using A-Frame as the XR visualization engine, enabling it to be used on both XR devices and in a desktop browser.

The result is a prototype designed for teaching and demonstrating computer networking concepts.

Índice general

Índice de figuras

Capítulo 1

Introducción

Este Trabajo de Fin de Grado, cuyo repositorio en GitHub es https://github.com/mcerdeno2021/SIMULADOR-REDES-ORDENADORES-XR, trata de abordar una nueva forma de entender la arquitectura de red y el flujo de paquetes que existe en cualquier red.

Para ello, el usuario podrá diseñar una topología de red y una captura de tráfico cualquiera en formato JSON, siguiendo las especificaciones que se comentan más adelante, la cual podrá ser visualizada en realidad virtual, controlando en todo momento la reproducción de la captura.

Previo a esta herramienta, el comportamiento habitual sería recurrir a herramientas de simulación y captura de tráfico que permiten observar el intercambio de paquetes entre dispositivos y analizar el funcionamiento de protocolos de comunicación en distintos niveles de la pila de red. No obstante, es difícil entender algo tan complejo como el flujo de paquetes en la red con una herramienta que lo representa en texto en 2D.

Por ello, se ha trabajado en una aplicación que permita representar capturas, probar distintas topologías y, mediante la repetición y el uso, entender cómo funcionan las redes de ordenadores y su tráfico de paquetes.

Esta aplicación se encuentra desplegada en su propia página web, mediante GitHub Pages, en la que se pueden utilizar demos, descargar la memoria o acceder al repositorio; pudiendo ver además el estado de la herramienta en cada sprint. Enlace a la página web: https://mcerdeno2021.github.io/SIMULADOR-REDES-ORDENADORES-XR/.

1.1. Contexto

Este trabajo nace de una necesidad en nuestro grado y en todas las áreas de las telecomunicaciones, que es visualizar y, en consecuencia, poder entender como funcionan las redes de ordenadores que hacen, hoy en día, más accesible para una gran parte de la población el acceso a toda la información, conocimientos y entretenimiento. Se pretende con este proyecto dejar de ver las redes de ordenadores y todos sus elementos como una caja negra.

Paralelamente, el auge de las tecnologías de realidad extendida (XR) ha abierto nuevas posibilidades en la visualización de información compleja y en el desarrollo de entornos inmersivos orientados al aprendizaje, la simulación y la interacción avanzada con datos espaciales.

Por ello, parece casi una evolución natural pensar que las aplicaciones que se aprovechan de toda esta nueva tecnología serán las que puedan ayudar más a la docencia en este ámbito.

1.2. Motivación

En el ámbito docente, la comprensión de capturas de red y del recorrido de los paquetes a través de una topología constituye una de las principales dificultades para los estudiantes en asignaturas relacionadas con redes de computadores. Aunque otras herramientas permiten inspeccionar exhaustivamente cada paquete, la interpretación espacial y temporal de dicha información requiere un alto nivel de abstracción.

Actualmente, no existen herramientas ampliamente extendidas que permitan visualizar de forma directa el flujo de paquetes capturados en una topología de red dentro de un entorno XR, integrando simultáneamente la información topológica y temporal de la red. Esta carencia representa una oportunidad para desarrollar una solución innovadora orientada a facilitar el aprendizaje y análisis del comportamiento de redes.

La integración de una herramienta propia de modelado como NetGUI-Kathará, que permite, como se menciona en el apéndice sobre la misma, crear ficheros JSON propios, unida a su uso previo en la docencia, permite crear un flujo de trabajo lógico dentro de las clases usando un visor XR para poder mejorar la experiencia en el aula.

1.3. Estructura de la memoria

Esta memoria describe el proceso completo de desarrollo y validación de una herramienta de visualización de tráfico de red mediante tecnologías de realidad extendida. La memoria se ha organizado en distintos capítulos tratando de que se entienda no solo cómo está estructurada, sino también el proceso que se ha seguido desde que nace la idea hasta que se valida. Se abarcan desde los fundamentos tecnológicos necesarios hasta los resultados obtenidos con la aplicación desarrollada.

El Capítulo 2, *Tecnologías relacionadas*, se presentan las herramientas y conceptos tecnológicos en los que se basa el proyecto. En este apartado se introducen las tecnologías web y lenguajes de programación empleados para el desarrollo de la aplicación, como A-Frame, Three.js, WebGL, WebXR, JavaScript, Python y HTML, así como otras herramientas fundamentales utilizadas durante el desarrollo, entre las que se encuentran JSON, Wireshark, NetGUI-Kathará y GitHub. Además, se describen los dispositivos XR utilizados para la validación de la aplicación y se analiza la importancia del proyecto WireXRk, que constituye una de las referencias más próximas a lo que se propone en este trabajo, así como el trabajo de un compañero de la escuela, Enrique Estebaranz, que es otra referencia utilizada para ciertas etapas del proyecto.

El Capítulo 3, *Resultados*, presenta el resultado final alcanzado tras la implementación del proyecto. En primer lugar, se presenta la arquitectura general e interna de la aplicación, describiendo la forma en que interactúan los distintos módulos que componen el sistema. Después, se muestran las capacidades de la aplicación desarrollada, así como un manual de usuario, en el que se muestran los pasos que se pueden seguir para un correcto funcionamiento y aprovechamiento de la aplicación; por último, el capítulo muestra al detalle los componentes que usa la aplicación internamente, explicando la relevancia de cada uno. Asimismo, este capítulo da visibilidad al potencial de la herramienta para la docencia, ya que tiene unos pasos muy sencillos a seguir y es capaz de facilitar la comprensión de conceptos relacionados con redes de ordenadores, teniendo mucho margen de crecimiento.

El Capítulo 4, *Diseño e implementación*, junto con el anterior, constituyen el núcleo principal de la memoria. En este, se describe el proceso de desarrollo seguido durante el proyecto mediante una organización *Agile*, la cual está basada en sprints. En cada uno, se define cuál es el objetivo marcado, las tareas realizadas y una serie de conclusiones/resultados. Este esquema

basado en sprints muestra cómo se ha desarrollado el trabajo, las ideas que han surgido y hace más comprensible el diseño de la herramienta.

Finalmente, el Capítulo 5, *Conclusiones*, resume las principales aportaciones realizadas durante el Trabajo Fin de Grado. Se evalúa el grado de cumplimiento de los objetivos planteados inicialmente, se debate sobre los conocimientos adquiridos y aplicados en el proceso y se muestran algunas de las dificultades encontradas. Además, se presentan posibles líneas de trabajo futuras como la integración con tráfico en tiempo real o la extensión a escenarios más complejos.

1.4. Objetivos

1.4.1. Objetivo principal

El objetivo principal de este Trabajo de Fin de Grado es diseñar e implementar una herramienta software capaz de representar de forma visual e inmersiva el flujo de paquetes en una red de ordenadores mediante tecnologías de realidad extendida (XR), permitiendo la reproducción de capturas de tráfico cualesquiera

La herramienta debe facilitar la comprensión espacial y temporal del flujo de paquetes de la red, proporcionando una representación intuitiva de estos, de los dispositivos y de las comunicaciones entre ellos.

1.4.2. Objetivos instrumentales

Para alcanzar el objetivo principal es necesario cumplir una serie de objetivos instrumentales:

- Profundizar en el uso de tecnologías web empleadas en el desarrollo de aplicaciones XR, especialmente HTML, JavaScript y A-Frame.
- Estudiar el funcionamiento de las tecnologías WebXR y los mecanismos de interacción disponibles en dispositivos inmersivos.
- Analizar la estructura de los datos de topología y capturas de tráfico exportadas para permitir su procesamiento automático.

- Diseñar una arquitectura software que permita integrar la información de la topología de red y las capturas de paquetes en un entorno tridimensional.
- Desarrollar una interfaz de usuario que permita controlar la reproducción de las capturas y consultar información relevante de la simulación.
- Integrar un servidor local encargado de detectar y distribuir automáticamente los datos generados durante la simulación.
- Proporcionar una herramienta de apoyo para demostraciones docentes, laboratorios y presentaciones académicas.

1.4.3. Objetivos secundarios

Como objetivos complementarios al desarrollo principal se consideran los siguientes:

- Mejorar la experiencia de aprendizaje en asignaturas relacionadas con redes de computadores mediante una representación más intuitiva del tráfico de red.
- Facilitar la comprensión del recorrido de los paquetes a través de una topología mediante la combinación de información espacial y temporal.
- Evaluar la viabilidad de las tecnologías XR como herramienta de apoyo en la enseñanza de conceptos relacionados con redes de ordenadores.
- Establecer una base tecnológica extensible que permita añadir futuras funcionalidades.

Capítulo 2

Tecnologías relacionadas

El desarrollo de la aplicación que se expone en este Trabajo de Fin de Grado ha requerido la integración de tecnologías software y hardware.

Para llevar a cabo algunas implementaciones concretas se exige combinar tecnologías muy diferentes. La interoperabilidad entre todas las tecnologías es una de las partes que más destacan del proyecto.

En este capítulo se describen las principales herramientas, lenguajes y plataformas empleadas durante el diseño e implementación del sistema, incluyendo el papel que desempeñan dentro de la arquitectura final del proyecto.

2.1. A-Frame

A-Frame [1] es un framework opensource orientado al desarrollo de experiencias de realidad virtual y extendida sobre tecnologías web.

Está construido sobre Three.js y WebGL; en este punto es importante diferenciar estas tecnologías ya que se va a hablar más adelante de las tres en contextos similares. Por orden, la de nivel más bajo es WebGL, esta es una API gráfica de bajo nivel del navegador para renderizar gráficos 2D/3D usando la GPU; después, se encuentra Three.js, que es una librería JavaScript que abstrae WebGL. Por último, la mencionada A-Frame, que permite la creación de escenas tridimensionales mediante una sintaxis declarativa basada en HTML y construida sobre Three.js, simplificando el desarrollo de aplicaciones XR.

Una escena XR se construye mediante entidades de A-Frame, sobre las que se registran los

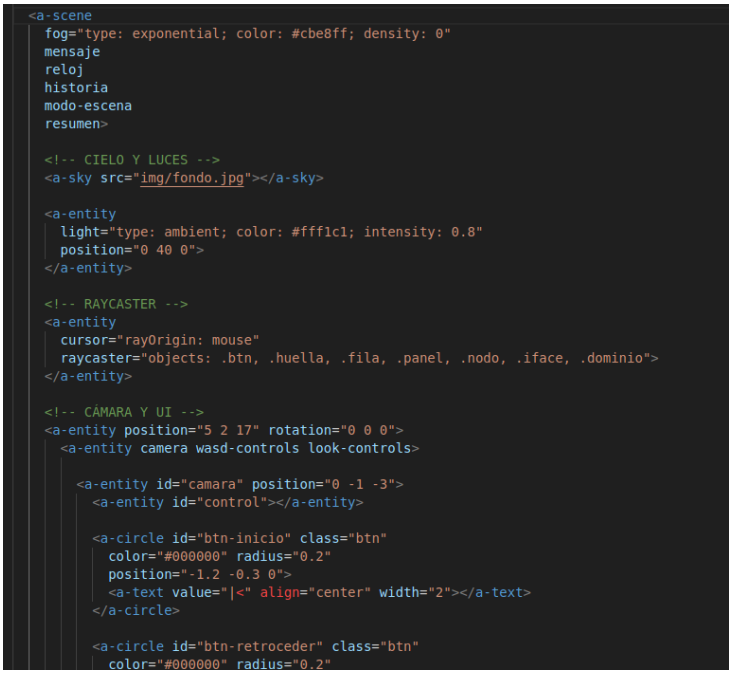


Figura 2.1: A-Frame

componentes implementados durante el desarrollo.

- Las entidades son objetos creados con la etiqueta *a-entity* a los que se les asignan atributos como posición o forma, y componentes. También se pueden usar primitivas, que son elementos HTML que actúan como un atajo para crear objetos 3D complejos.

```
<a-entity id="camara" position="0 -1 -3">
  <a-entity id="control"></a-entity>

  <a-circle id="btn-inicio" class="btn"
    color="#000000" radius="0.2"
    position="-1.2 -0.3 0">
    <a-text value="|<" align="center" width="2"></a-text>
  </a-circle>
```

Figura 2.2: Entidad A-Frame y primitiva *a-circle*

- Los componentes son bloques que definen los atributos de una entidad y que son reutilizables, se escriben en JavaScript. Están ceñidos al sistema ECS (Entity Component System). Toda la lógica de las aplicaciones se organiza mediante componentes registrados con *AFRAME.registerComponent()*, lo que permite separar las funcionalidades del

desarrollo. Para todo esto, se tiene una integración con la API del DOM para manipular elementos cambiando atributos, eliminándolos, etc. Se inicializan con *init()* y pueden tener parámetros de entrada en el *schema*.

```
// Registering component in foo-component.js
AFRAME.registerComponent('foo', {
  schema: {},
  init: function () {},
  update: function () {},
  tick: function () {},
  remove: function () {},
  pause: function () {},
  play: function () {}
});
```

Figura 2.3: Componente A-Frame

La elección de A-Frame frente a otras alternativas se debe primero, a la experiencia previa adquirida en esta herramienta, y, en lo técnico, a su capacidad para hacer más sencilla e intuitiva la programación de entornos inmersivos, permitiendo centrar el desarrollo en la lógica funcional. Además, su compatibilidad directa con dispositivos XR, de las que se hablará más adelante, facilita la validación sobre hardware XR real en desarrollos.

2.2. Three.js

Three.js [10] es una biblioteca JavaScript de gráficos 3D que abstrae el uso directo de WebGL y proporciona una API de más alto nivel; constituye el motor gráfico sobre el que se construye A-Frame (se encarga del renderizado).

Three estructura la escena de una manera jerárquica, en la que se encuentran los objetos, cuya raíz es la escena (*Scene*). Cada objeto puede contener hijos que heredan las transformaciones como traslaciones o rotaciones. Esta organización jerárquica facilita la gestión de modelos más complejos y permite aplicar transformaciones sobre grupos completos de elementos que mantengan relación.

Una escena en Three.js se compone, como mínimo, de tres elementos fundamentales:

- La escena, que actúa como contenedor de todos los objetos tridimensionales.

- La cámara, que define el punto de vista desde el que se observa la escena.

- El renderizador, encargado de recorrer los objetos siguiendo la estructura jerárquica y enviar la información correspondiente a WebGL.

```
AFRAME.registerComponent('slider', {
  init: function () {
    this.min = 0;
    this.step = 0.1;
    this.width = 8;
    this.dragging = false;
    this.max = 1;

    this.el.sceneEl.addEventListener('max', e => {
      this.max = e.detail;
    });

    this.bar = document.createElement('a-plane');
    this.bar.setAttribute('width', this.width);
    this.bar.setAttribute('height', 0.12);
    this.bar.setAttribute('color', '■#dddddd');
    this.el.appendChild(this.bar);

    this.progress = document.createElement('a-plane');
    this.progress.setAttribute('width', 0.001);
    this.progress.setAttribute('height', 0.12);
    this.progress.setAttribute('color', '■#666666');
    this.progress.setAttribute('position', `${-this.width/2} 0 0.01`);
    this.el.appendChild(this.progress);

    // Cuando el reloj avanza solo
    this.el.sceneEl.addEventListener('set-time', e => {
      this.setValue(e.detail);
    });
  },

  setHandlePosition(localX) {
    const progressWidth = localX + this.width/2;
    this.progress.setAttribute('width', progressWidth);
    this.progress.setAttribute('position', `${-this.width/2 + progressWidth/2} 0 0.01`);
  },

  emitValue() {
    const localX = this.handle.object3D.position.x;
    const ratio = (localX + this.width / 2) / this.width;
    let value = this.min + ratio * (this.max - this.min);
    value = Math.round(value / this.step) * this.step;
```

Figura 2.4: Three.js

Los objetos se construyen mediante geometrías y materiales.

- Las geometrías definen la forma de los modelos mediante conjuntos de vértices, aristas y caras.

- Los materiales determinan la apariencia, con propiedades como color o sombreado. La unión genera una malla (*Mesh*), que es la unidad básica de representación.

Gracias a su elevado nivel de simplificación de la generación de objetos, Three.js constituye la base sobre la que se construyen frameworks de mayor nivel como A-Frame.

2.3. WebGL

WebGL [4] es una API gráfica que permite renderizar gráficos 2D y 3D directamente desde el navegador web.

El funcionamiento de WebGL sigue este flujo:

1. La aplicación envía a la GPU la información geométrica de los objetos a representar.

2. El procesamiento de los vértices se realiza mediante los *vertex shaders*. Estos transforman las coordenadas de cada vértice desde el espacio del objeto hasta el espacio de la cámara y de la pantalla.

3. Posteriormente, se convierten las primitivas geométricas en píxeles de la imagen final. Cada uno de ellos es procesado por un *fragment shader*, encargado de calcular el color definitivo.

4. Finalmente, la imagen calculada se almacena en el framebuffer y es presentada en pantalla por el navegador.

A diferencia de bibliotecas de más alto nivel, como Three.js, WebGL no proporciona primitivas. Estas se deben implementar manualmente mediante matrices de transformación y operaciones matemáticas. Lo que lo hace muy flexible pero menos intuitivo y sencillo.

2.4. WebXR

WebXR [12] es el estándar del W3C que define la API utilizada por los navegadores para acceder a dispositivos de realidad virtual y realidad aumentada. Gracias a esta API, las aplicaciones web detectan la disponibilidad de dispositivos XR, inician sesiones inmersivas y obtienen la información sobre la posición del usuario.

En su flujo, una aplicación solicita al navegador la creación de una sesión indicando el tipo de experiencia, ya sea inmersiva o no inmersiva. Una vez concedido el acceso, el navegador establece la comunicación con el dispositivo físico y le proporciona información sobre su estado. Entre estos datos, la posición y orientación del visor mediante sistemas de referencia (*Reference Spaces*).

Además del seguimiento del visor, la API da acceso a los controladores y otros dispositivos de entrada. Estos incluyen información sobre botones, gestos y la posición espacial de cada controlador, permitiendo la interacción tridimensional.

Su uso permite validar herramientas sobre los dispositivos XR, importante para ajustar la escala de los elementos de una escena, distancias de visualización de los paneles, tamaño de los textos, etc.

2.5. JavaScript

JavaScript [7] es el lenguaje de programación principal empleado en la implementación de la lógica funcional de esta aplicación.

El modelo de ejecución de JavaScript se basa en un hilo principal gestionado mediante *Event Loop*, que coordina la ejecución del código síncrono junto con las operaciones asíncronas, clave para mantener interfaces de usuario reactivas sin bloquear la ejecución de operaciones de entrada/salida o peticiones de red.

Gracias a estas capacidades y a su integración nativa con el navegador, JavaScript es uno de los pilares del ecosistema web actual.

Todos los componentes que se describen en el capítulo de implementación se han desarrollado como componentes personalizados de A-Frame escritos en JavaScript.

```
AFRAME.registerComponent('panel', {
  schema: {
    origenPos: {type: 'string'},
    origen: {type: 'string'},
    destinoPos: {type: 'string'},
    destino: {type: 'string'},
    activa: {type: 'boolean', default: false}
  },

  init: function () {
    const el = this.el;
    const { origenPos, destinoPos, origen, destino } = this.data;

    this.titulo = `${origen} -> ${destino}`;

    // Validación de strings
    if (!origenPos || !destinoPos) {
      console.warn("Panel sin posiciones válidas:", this.data);
      return;
    }
```

Figura 2.5: Creación de componentes en JavaScript

Gracias a esto, es usar un modelo basado en eventos, manipular el DOM de la escena mediante llamadas *setAttribute()* y gestionar estructuras de datos para almacenar la información de

la simulación.

Es clave, por último, destacar la integración nativa del lenguaje con A-Frame y con las APIs del navegador.

2.6. HTML

HTML (HyperText Markup Language) [14] es el lenguaje de marcado estándar utilizado para definir la estructura de páginas web. Su función es describir el contenido de una página mediante un conjunto de elementos organizados jerárquicamente, para que los navegadores lo interpreten y lo representen.

La versión utilizada es HTML5, que tiene soporte nativo para contenido multimedia, formularios avanzados y nuevas APIs orientadas al desarrollo de aplicaciones web.

El lenguaje incorpora semántica para describir el significado del contenido y su estructura. Etiquetas como *header*, *section*, *article* o *footer*.

Por su naturaleza declarativa, HTML se puede emplear para definir escenas XR mediante A-Frame. Elementos como *a-scene*, *a-entity*, *a-plane* o *a-camera* forman parte del propio documento HTML y sirven como base sobre la que se instancian los componentes.

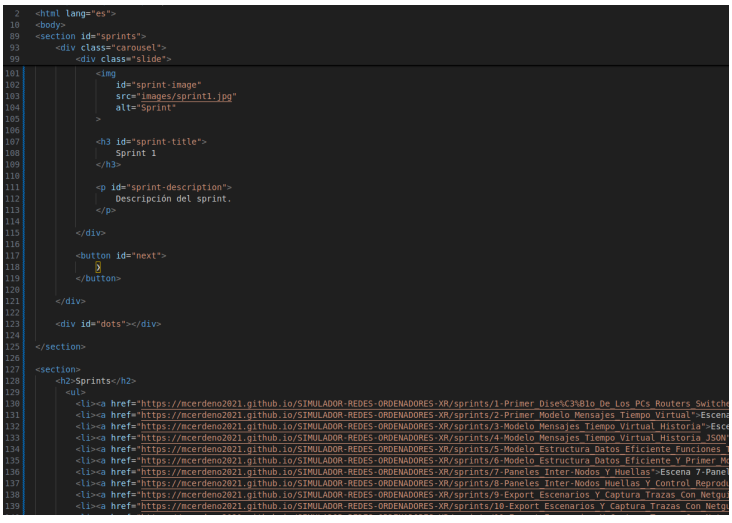


Figura 2.6: HTML

2.7. DOM

El Document Object Model [13] es la presentación estructurada, en memoria, del documento HTML que proporciona el navegador y permite acceder dinámicamente a cualquiera de sus elementos mediante JavaScript y manipularlo.

Cuando un navegador carga un documento HTML, construye internamente un árbol de nodos, el DOM. Cada etiqueta HTML se convierte en un nodo del árbol, organizado jerárquicamente. Con esta estructura se recorre el documento con relaciones de parentesco entre nodos. La API del DOM proporciona numerosos métodos para acceder a los nodos, crear nuevos elementos, eliminar otros existentes o modificar atributos, estilos y contenido textual.

El DOM incorpora un sistema de gestión de eventos, como clics que se capturan con (*Event Listeners*), dando como resultado un modelo de programación dirigido por eventos.

Relacionado con A-Frame, se pueden localizar entidades y modificar sus atributos mediante JavaScript, facilitando la comunicación entre la lógica de la aplicación y la representación gráfica.

```
var nuevoElemento = document.createElement('legend');
var nuevoContenido = document.createTextNode('Creando nuevo elemento');
nuevoElemento.appendChild(nuevoContenido);
```

Figura 2.7: Gestión del DOM desde JavaScript

2.8. NetGUI-Kathará

NetGUI-Kathará [8] es una interfaz gráfica para el sistema de emulación de redes Kathará. Cuenta con un panel con una barra de menús y una barra de botones que permite a los usuarios crear diagramas de red seleccionando elementos como ordenadores, routers, switches, routers o hubs. Posteriormente, permite a los usuarios iniciar y detener máquinas virtuales (nodos) e interactuar con sus consolas. Es una plataforma open-source desarrollada para facilitar el aprendizaje en el ámbito de las redes informáticas.

En su antigua versión proporcionaba una interfaz gráfica con las funcionalidades que se mencionan anteriormente. En esta nueva versión, desarrollada con Kathará, el cual es un software para la emulación de redes basado en Docker, se añaden nuevas funcionaliades como:

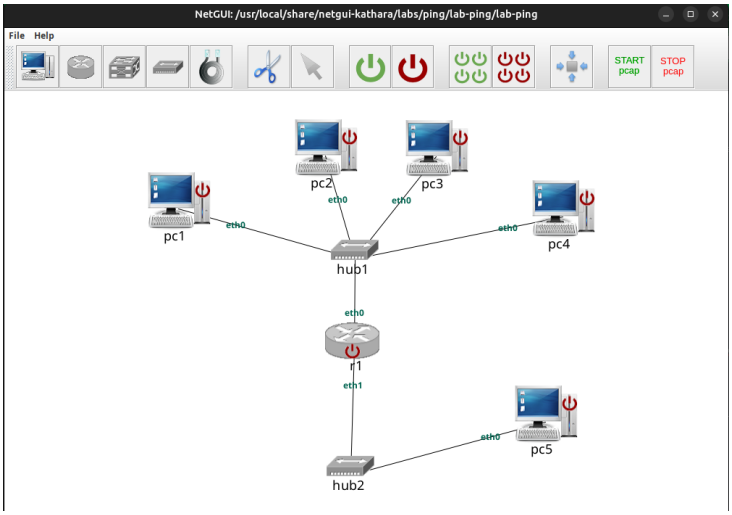


Figura 2.8: NetGUI-Kathará

- Capacidad de generar los archivos necesarios para visualizar el tráfico de red en XR: NetGUI-Kathará es capaz de exportar todas las capturas y la topología usada en ficheros JSON, que se pueden usar para iniciar la simulación con la herramienta diseñada; esto se detalla en el apéndice sobre Netgui-Kathará.

- Iniciar todas las máquinas al mismo tiempo.

- Capturar tráfico en todas las interfaces de la red directamente desde un botón.

2.9. JSON

JSON (JavaScript Object Notation) [2] de intercambio de datos basado en texto, diseñado para ser fácilmente interpretable.

La estructura de un documento JSON se basa en objetos y arrays:

- Los objetos representan colecciones de pares clave-valor, las claves son cadenas de caracteres y cada valor puede corresponder a distintos tipos de datos.

- Los arrays permiten almacenar colecciones ordenadas de elementos, pudiendo contener valores de distintos tipos e incluso anidar otros arrays.

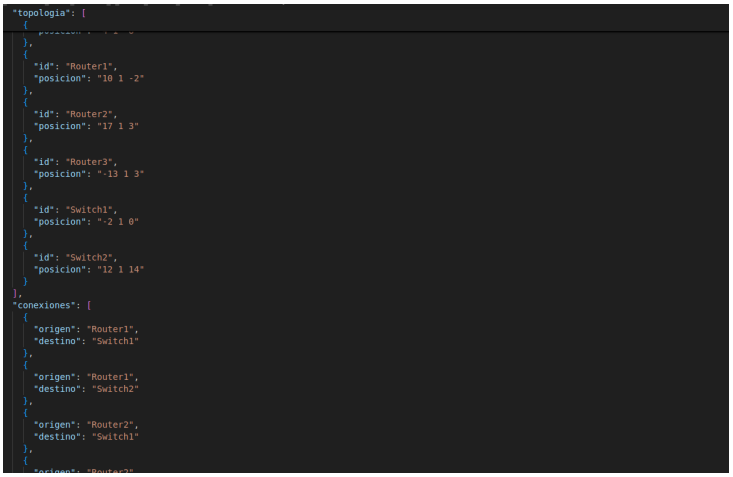


Figura 2.9: JSON

2.10. Wireshark

Wireshark [15] es una herramienta open-source; se define como un analizador de paquetes de red que se puede utilizar para realizar análisis y solucionar problemas en redes de comunicaciones y como una herramienta didáctica. Además, permite capturar el tráfico desde cualquier interfaz a la que pueda conectarse.

Esta herramienta representa una traza de paquetes mostrando uno por fila; estos contienen la información principal (puerto origen y destino, longitud, ...) sobre la misma fila, sin embargo, existe la opción de desplegarlo, apareciendo información sobre cada capa.

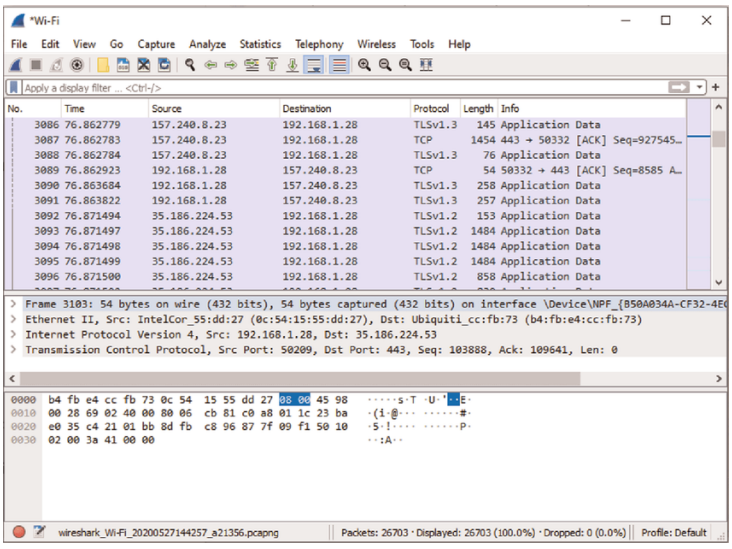


Figura 2.10: Interfaz de Wireshark

Tiene las siguientes características que la hacen diferencial y una de las más utilizadas en este ámbito:

- Decodificación profunda. Desglosa los paquetes de datos capa por capa basándose en estándares como el modelo OSI.
- Filtros avanzados: Permite aislar IPs, puertos o protocolos específicos.
- Análisis de estado: Reconstruye flujos de sesiones completas (como una petición HTTP o una transmisión TCP).

2.11. GitHub

GitHub [3] ha sido utilizado como plataforma de alojamiento remoto y control de versiones del código fuente del proyecto.

Permite usar ramas (*branches*), que son líneas de tiempo paralelas, útiles para experimentar sin alterar el proyecto principal.

Gracias a su integración con Git, permite mantener un historial completo del desarrollo y facilitar la gestión de versiones intermedias, lo que permite acceder a ficheros anteriores cuando sea necesario. Esto lo hace mediante comandos como:

- *commit*: permite crear un punto de guardado.
- *merge*: para fusionar el trabajo de dos ramas distintas.
- *push*: para subir el historial de commits al repositorio.

Además de ser útil durante el desarollo, es igual de útil para su publicación, ya que Github permite presentar el proyecto al resto de usuarios, en un formato que entienden y que permite tener acceso al código. El fichero *readme.md* permite dar una descripción básica de lo que se encuentra alojado en el repositorio.

2.12. VSCodium

VSCodium [11] es un entorno de desarrollo integrado (IDE) open-source basado en Visual Studio Code. A diferencia de este, VSCodium elimina los componentes propietarios, pero

mantiene las mismas funcionalidades desarrollo.

El editor incorpora un sistema de depuración para establecer puntos de ruptura, ejecutar el código paso a paso, etc.

Otra característica destacable es su integración con Git; permitiendo visualizar diferencias entre versiones y gestionar ramas desde la propia interfaz gráfica.

Uno de sus elementos más importantes es el sistema de extensiones, implementado mediante una API que permite añadir distintas funcionalidades. De aquí, se ha explotado la utilidad de Live Server, una extensión que permite ejecutar un servidor web local durante el desarrollo de aplicaciones web, permitiendo visualizar el desarrollo directamente desde el navegador, además de actualizar automáticamente los cambios realizados en el código.

2.13. Meta Quest 3

Las Meta Quest 3 [6] son un visor de realidad extendida (XR) autónomo desarrollado por Meta. El dispositivo integra capacidades de realidad virtual y realidad mixta, permitiendo ejecutar aplicaciones inmersivas.

Destacan entre sus características:

- Para determinar continuamente la posición del usuario, utilizan un sistema de seguimiento que combina varias cámaras RGB y sensores infrarrojos cuya información permite estimar la posición y orientación del usuario en tiempo real sin necesidad de sensores externos.
- El sistema incorpora reconocimiento espacial del entorno, reconstrucción tridimensional de superficies y detección automática de obstáculos.
- Con los altavoces integrados con sonido estéreo y espacial 3D se obtiene un mayor rango de graves y un posicionamiento perfecto del audio

2.14. Python

Python [9] es un lenguaje de programación de alto nivel, muy utilizado en desarrollo de aplicaciones, automatización de tareas y procesamiento de datos. Posee una biblioteca estándar

Figura 2.11: Meta Quest 3

muy extensa y una gran cantidad de módulos que permiten hacerlo mu útil para desarrollar utilidades auxiliares. Todo esto, sumado a su amplio número de usuarios, permite tener una documentación y una cantidad de dudas resueltas que lo hacen mucho más sencillo de utilizar en implementaciones complejas en comparación a otros lenguajes. Todo esto es consecuencia de su legibilidad y simplicidad sintáctica, que facilita la escritura de código claro y lógico.

En Python el tipo de cada variable se determina durante la ejecución del programa. La organización del lenguaje se fundamenta en módulos y paquetes, permitiendo estructurar aplicaciones componentes reutilizables.

Python admite programación orientada a objetos. Su sintaxis, basada en la indentación elimina favorece una escritura uniforme y mantenible.

```
def vigilar_captura(ruta, stop_event):
            ultima_fecha = fecha

        time.sleep(1)


# ---------- HANDLER ----------
class MiHandler(http.server.SimpleHTTPRequestHandler):
    ruta_captura = None

    def __init__(self, *args, **kwargs):
        super().__init__(*args, directory=WEB_DIR, **kwargs)

    def do_GET(self):

        if self.path == "/escenarios":
            self.listar_escenarios()

        elif self.path.startswith("/usr/"):
            self.servir_lab()

        elif self.path.startswith("/seleccionar"):
            self.seleccionar_escenario()

        elif self.path == "/estado":
            self.enviar_estado()

        else:
            super().do_GET()

    # listar escenarios
    def listar_escenarios(self):
        carpetas = [
            d for d in os.listdir(LABS_DIR)
            if os.path.isdir(os.path.join(LABS_DIR, d))
        ]

        self.responder_json(carpetas)
```

Figura 2.12: Ejemplo de script en Python

2.15. LaTeX

LaTeX [5] es un sistema de composición de documentos utilizado en entornos académicos, tiene una gran capacidad para generar documento con elementos como tablas, figuras, referencias bibliográficas e índices. Su proceso de edición se divide en dos fases: la escritura del texto en un editor de texto plano y su compilación para generar el documento final.

Como plataforma de edición en línea se ha usado Overleaf, este simplifica la escritura y posee una funcionalidad de compilar para tener una vista previa cada vez que se necesite y poder descargar las versiones en formato PDF, agilizando mucho el proceso, destacando así sobre otras opciones como Visual Studio.

La gestión bibliográfica suele realizarse mediante BibTeX, que permite mantener bases de datos de referencias fuera del documento principal.

2.16. WireXRk

WireXRk es un proyecto de código abierto alojado en el repositorio de GitHub de Netgui-kathará, orientado a la visualización de tráfico de red mediante tecnologías de realidad extendida, desarrollado por otros docentes del grado.

Esta herramienta se desarrolla gracias a Netgui-Kathará y funciona de la siguiente manera:

- Netgui-Kathará es capaz de generar unos ficheros JSON que cubren la topología de la red generada y la captura que se obtenga
- Con esto se genera la visualización y se reproduce la captura, permitiendo controlar la temporalidad de esta y con paneles informativos que van explicando cómo se desarrolla la captura
- Tiene unas demos con una serie de escenarios concretos.

Debido a la similitud entre sus objetivos y los de este Trabajo de Fin de Grado, ha servido como fuente de inspiración durante el diseño y desarrollo de la aplicación propuesta. Asimismo, el análisis de su arquitectura y funcionalidades ha permitido identificar soluciones técnicas y enfoques de visualización para este proyecto.

Sin embargo, tiene algunas diferencias conceptuales:

Figura 2.13: Interfaz de WireXRk

- Esta herramienta está pensada para representar escenarios y capturas arbitrarios, que se pueden obtener de herramientas reales fuera de Netgui-Kathará (siguiendo el formato de ficheros JSON que se explica en el siguiente capítulo).

- No pretende explicar la captura de una manera textual, aunque también contenga carteles informativos, sino que busca que, de forma interactiva, con los paneles internodos que se explican en el siguiente capítulo, se pueda entender el flujo temporal del tráfico de una red.

- Por otro lado, siguiendo el punto de los paneles, sobre estos se deja un rastro que representa el paso de los paquetes y que forma un diagrama temporal del flujo que permite entender la cronología de la captura.

Todos estos detalles de la implementación se exponen extensamente en el siguiente capítulo.

2.17. TFG Enrique Estebaranz

Este TFG, publicado en el año 2024, con el título ”SIMULADOR DE REDES DE ORDENADORES EN REALIDAD EXTENDIDA”, ha sido una referencia importante durante algunas fases del proyecto.

El autor, *Enrique Estebaranz Redondo*, describe su trabajo de la siguiente manera: .Este simulador permite ver la evolución de como los paquetes se mueven en la red a lo largo del tiempo, en la que cualquier usuario puede interactuar con la red de una manera fácil, intuitiva y visualmente rica..Este trabao, además, se ha desarrollado utilizando A-Frame.

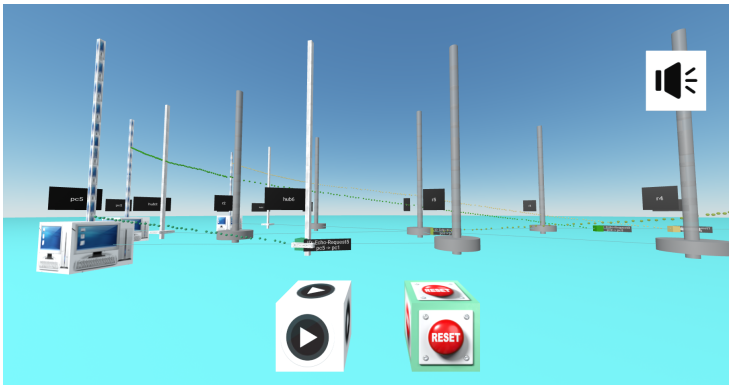


Figura 2.14: Visualización TFG Enrique Estebaranz

Ha sido desarrollado bajo la supervisión de Jesús María González Barahona, tutor de este proyecto, y mantiene semejanzas en bastantes partes del desarrollo, lo que ha permitido solventar más rápidamente problemas que ya se habían detectado. Esto ha hecho que partes de la arquitectura y la planificación de componentes tengan una base fundada.

Sin embargo, este trabajo no es más que una base sobre la que entender lógicas de programación de la aplicación y obtener inspiración para mejorar puntos del proyecto. Son varias las diferencias, pero la principal es el enfoque de cómo ayudar en la docencia con esta herramienta; hay ideas comunes como la del rastro de los paquetes mediante huellas, sin embargo, se ha pretendido ir más alla en la interacción del usuario con la herramienta y en las facilidades que se le proponen para entender mejor el tráfico de la red, por ejemplo con:

- Control de la reproducción temporal.
- Paneles internodo que permiten aislar conexiones.
- Visor estilo Wireshark.

Todos estos detalles de la implementación se exponen extensamente en el siguiente capítulo.

Capítulo 3

Descripción del resultado

Este capítulo trata de explicar en profundidad cuál es el resultado final de esta aplicación.

Primero, en la siguiente sección, se explica el funcionamiento completo de la herramienta y se expone el resultado final del desarollo, describiendo las distintas características y funcionalidades que incorpora, así como los elementos que conforman la interfaz y las posibilidades de interacción para al usuario.

Posteriormente, se describe la implementación de la herramienta desde un punto de vista técnico. Se hace un análisis de la arquitectura interna de la aplicación, describiendo los componentes A-Frame que desarrollan la escena y cómo colaboran entre sí. El objetivo de esta parte es mostrar las decisiones de diseño adoptadas y el funcionamiento interno del sistema.

3.1. Funcionamiento completo de la aplicación

3.1.1. Visión general del resultado obtenido

El resultado obtenido es una herramienta que permite visualizar de forma inmersiva la evolución temporal del tráfico de sobre una topología tridimensional.

A partir de unos ficheros JSON que describen la topología y la captura de tráfico, la aplicación genera la escena XR con los dispositivos y enlaces de la red, procesa los paquetes y reproduce su recorrido. Durante la simulación, el usuario puede aprovecharse de funcionalidades como:

- El control de la reproducción temporal.

- La visualización individual de comunicaciones.
- La representación del camino seguido por los paquetes
- Información relevante como interfaces de red o tablas de encaminamiento.
- Un visor de paquetes inspirado en Wireshark.

.

Todo esto combinado permite analizar la captura mediante diferentes funcionalidades.

A lo largo de esta sección, se va a describir qué hace la herramienta, que funcionalidades posee y un flujo de uso. Para ello, se van a tratar distintas características y funcionalidades que cubren prácticamente en su totalidad lo que es capaz de hacer la aplicación.

3.1.2. Arquitectura general

La Figura 3.1 muestra el flujo de información desde la generación de una simulación hasta su representación inmersiva.

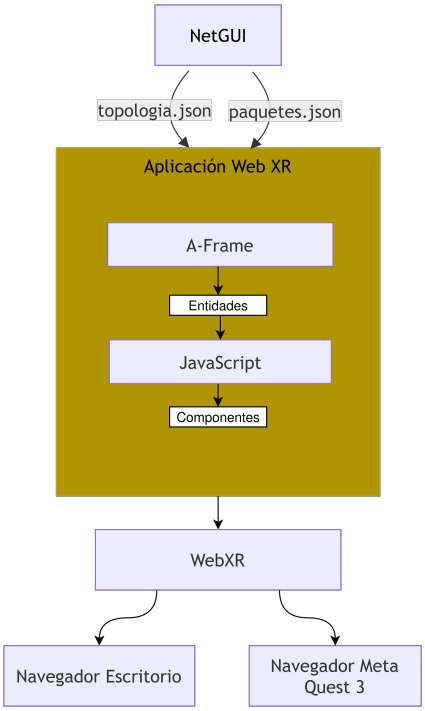


Figura 3.1: Arquitectura general del sistema

Como se puede observar, el usuario comienza introduciendo en la herramienta los ficheros JSON que contienen la descripción tanto de la topología como de la captura; en el flujo de la imagen se incluye Netgui como herramienta para generarlos. Estos ficheros son procesados por la aplicación desarrollada en A-Frame, encargada de generar la representación tridimensional mostrada al usuario.

Esta aplicación se crea con A-Frame, desde el cual se manejan entidades *a-entity*, y JavaScript, con el que se crean componentes, capaces de manipular las entidades y generar lógica funcional. El final del flujo se genera gracias a WebXR, el cual permite con permite a los navegadores acceder a dispositivos de realidad virtual y realidad aumentada, generando la posibilidad de que el usuario explore la aplicación mediante un dispositivo XR o directamente en el navegador de escritorio.

3.1.3. Representación de la topología

Al acceder al escenario, aparece representada la topología completa de la red. Todos los dispositivos se encuentran ya posicionados automáticamente según la información obtenida desde los ficheros JSON, mientras que las conexiones entre ellos aparecen dibujadas mediante enlaces tridimensionales.

Los dispositivos de red se representan mediante elementos tridimensionales distribuidos dentro del escenario virtual. Cada nodo (sea pc, router o switch) mantiene una posición concreta y se conecta visualmente con el resto de dispositivos mediante cables, los cuales también están definidos en el fichero de la topología.

La representación mantiene un equilibrio entre ser visualmente atractiva pero simple y tener alta capacidad de interpretación.

La Figura 3.2 muestra un ejemplo de la representación generada automáticamente para una topología de red.

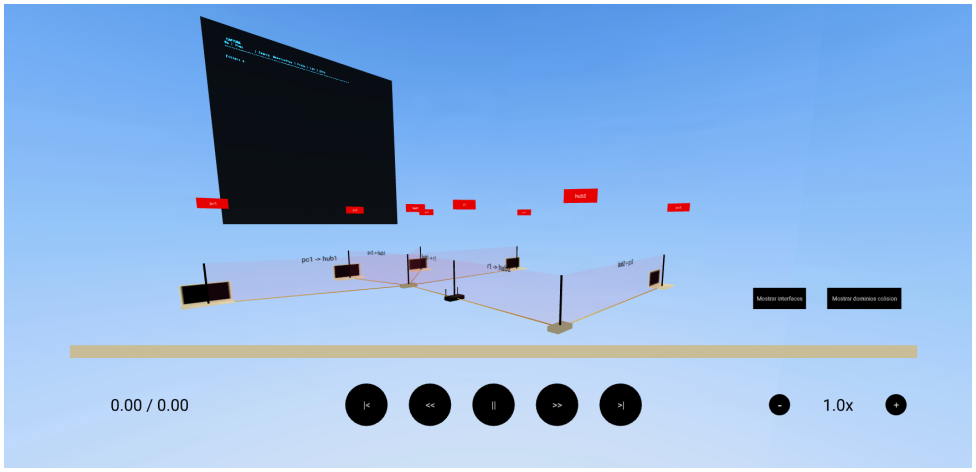


Figura 3.2: Representación tridimensional de una topología de red

3.1.4. Representación de paquetes

Los paquetes 3.3, que se obtienen del fichero de captura, se representan mediante objetos dinámicos, en este caso, entidades *a-box* de A-Frame, que se desplazan entre los diferentes nodos de la red.

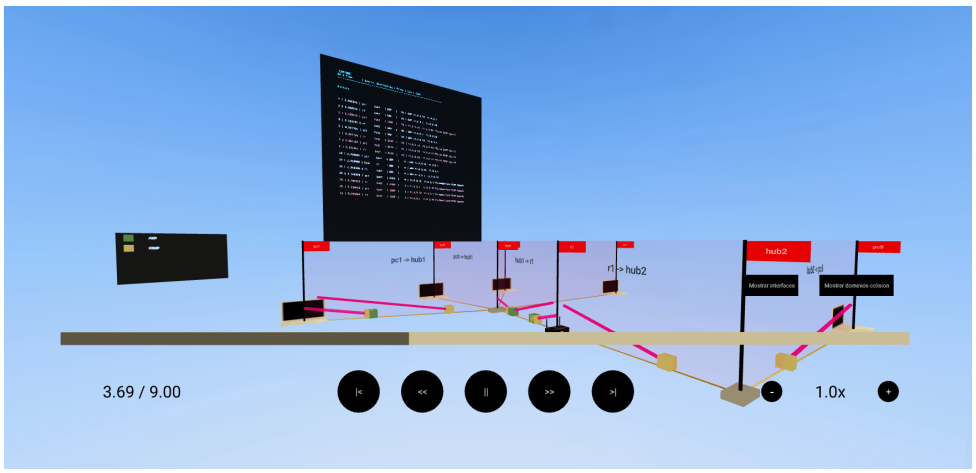


Figura 3.3: Representación tridimensional de los paquetes

Cada paquete sigue la trayectoria correspondiente a la comunicación observada durante la captura, desplazándose los paquetes por el camino que marca el cable que une los nodos. Gracias a esta representación es posible identificar visualmente el origen, destino y recorrido seguido por cada transmisión.

Esta representación con cajas animadas tan simple permite comprender de manera mucho

más intuitiva conceptos de la comunicación entre dispositivos, especialmente en topologías con múltiples nodos, ya que es posible acercarse al detalle que interese.

3.1.5. Paneles internodo

Los paneles son uno de los fuertes dentro de la representación de la topología. Cada enlace de la red dispone de un panel interactivo asociado, es decir, entre un router y un pc, cuyo camino traza un cable, se encuentra uno de estos paneles.

Estos tienen una función principal, que es delimitar la conexión y mostrar el tránsito que siguen los paquetes, ya que estos se mueven a la altura del suelo, donde se encuentra el cable, pero dejando un rastro en forma de pequeñas esferas de color rosa, que muestran el camino que ha seguido un paquete 3.4.

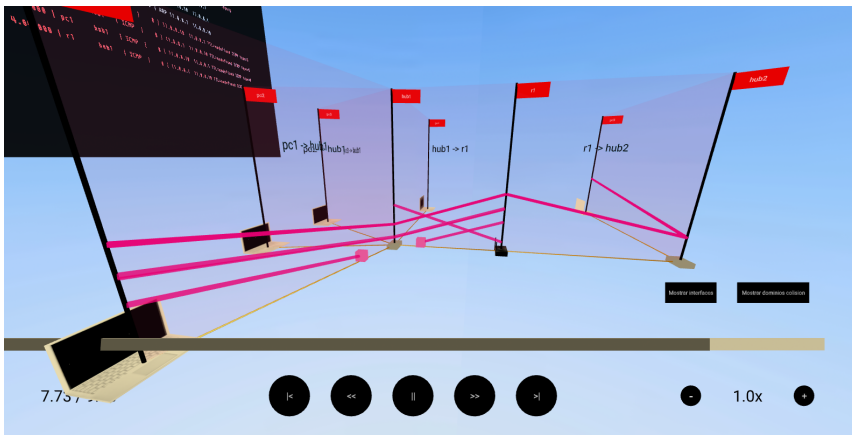


Figura 3.4: Paneles internodos y rastro de los paquetes

Lo más interesante de esto, es que los paneles van creciendo en altura a medida que avanza el tiempo de simulación, y con ellos, las huellas; así, se puede comprender la variable temporal, siendo los rastros más altos los primeros en haber ocurrido.

Además, el panel es interactivo y puede seleccionarse mediante un clic. Al hacerlo, la aplicación activa el modo de visualización por conexión, ocultando el resto de elementos de la topología y mostrando únicamente los nodos conectados, el cable que los conecta y los paquetes que circulan junto a las huellas que generan 3.5. Esta funcionalidad es muy útil para realizar un análisis más detallado de una comunicación concreta.

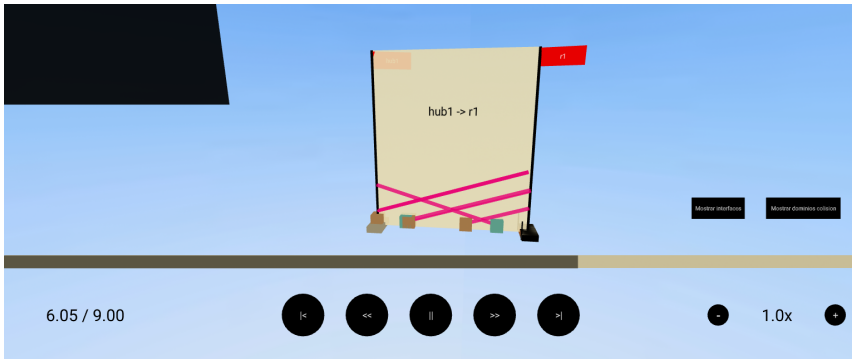


Figura 3.5: Aislamiento de una conexión durante la simulación

3.1.6. Control temporal de la simulación

Con el objetivo de facilitar el análisis de las capturas, la aplicación incorpora diferentes mecanismos para acercarnos más al detalle, en este caso, se habola de los de control temporal.

El usuario usando la funcionalidad del control de reproducción interactivo 3.6, puede iniciar, detener, reiniciar o finalizar la reproducción de la captura en cualquier momento; de la misma manera, es posible modificar la velocidad de reproducción. Todas estas funcionalidades son muy útiles para observar con mayor detalle determinadas situaciones o simplificar el análisis de escenarios complejos.

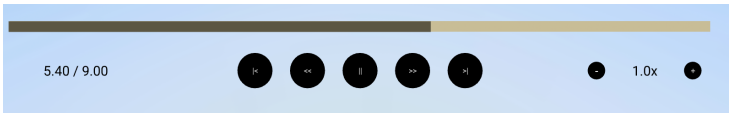


Figura 3.6: Control de reproducción temporal

Por otro lado, pensando en el ámbito docente, estas funcionalidades permiten adaptar la visualización a distintos contextos y facilitan la explicación de anomalías u otros casos concretos; asimismo, permite, por ejemplo, preparar ejercicios de análisis.

3.1.7. Carga de ficheros

La herramienta incorpora mecanismos para cargar automáticamente dos ficheros JSON; estos contienen la descripción de la topología, por un lado, y la información asociada a los paquetes capturados, por otro.

Esta funcionalidad permite adaptar la implementación según la experiencia que se requiera, ya que, respetando el formato del que se habla en el apéndice del manual de usuario, se puede introducir cualquier tipo de simulación que se quiera visualizar. Además, esta separación permite que una misma topología pueda reutilizarse con distintas capturas, lo que hace más sencillo realizar pruebas sobre diferentes escenarios de comunicación.

El fichero de la topología cumple con la estructura de un ".nkp", fichero de exportación de topologías NetGUI; mientras que el de la captura se podría obtener pasando una captura pcap, como las que se obienen en Wireshark, por la biblioteca Pyshark de python. Es importante asegurarse de que los ficheros que se introducen son coherentes (por ejemplo, que haya los mismos routers con los mismos nombres en ambos ficheros).

Ya con los archivos correspondientes, el sistema realiza una validación para comprobar si la información recibida contiene los parámetros esperados. Si los datos son correctos, se procede a generar el escenario; en caso contrario, se verá un escenario vacío.

En este punto, el programa dispone de toda la información necesaria para reproducir la simulación en un entorno XR.

3.1.8. Visualización mediante dispositivos XR

El otro mecanismo, y uno de los aspectos diferenciales de la herramienta es su capacidad para ejecutarse sobre dispositivos de realidad extendida.

La utilización de gafas Meta Quest 3 (soporta otros modelos, pero este es sobre el que se ha validado) permite al usuario situarse dentro del escenario generado y explorar la topología desde diferentes perspectivas. Esta capacidad incrementa significativamente la percepción espacial de la red y facilita la comprensión de las relaciones existentes entre dispositivos.

Además de esto, permite funcionalidades que lo hacen aún más avanzado, como la interacción directa con la simulación; más allá de los paneles, de los que se ha hablado anteriormente en la sección, de aquí se puede destacar:

- **Dominios de colisión**: Los dominios de colisión son el conjunto de dispositivos que no pueden transmitir una señal al mismo tiempo sin causar una colisión; que ocurre cuando dos o más señales diferentes se transmiten en el mismo segmento de cable al mismo tiempo. Los switches de red separan diferentes dominios de colisión. Pulsando este botón

se permite visualizar los distintos dominios de colisión que existen en la topología 3.7. Al activar esta opción se resaltan las zonas correspondientes y, al seleccionar una de ellas, se muestra un panel con el identificador del dominio y los dispositivos que lo componen.

- **Interfaces de red**: Las interfaces de red son las conexiones físicas entre un nodo y una red. Pulsando el botón, se muestran dichas interfaces físicas para cada enlace de la topología. Cada interfaz puede seleccionarse individualmente para consultar información detallada, como la dirección MAC, IP o máscara de red.

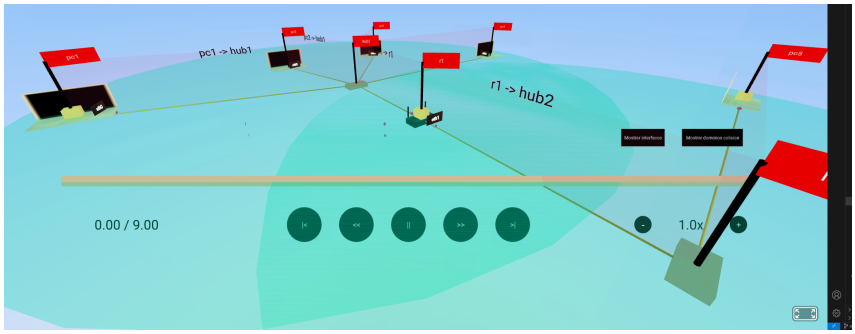


Figura 3.7: Ejemplo con dominios de colisión e interfaces de red

- **Panel de captura de paquetes**: Incorpora un visor inspirado en el diseño del frontal de Wireshark, donde se registran todos los paquetes de la simulación en tiempo real. Cada fila es un paquete diferente, donde se muestra el instante de captura, nodos origen y destino, protocolo utilizado, etc. Además, es posible desplegar cualquiera de los paquetes para ver información extra, esta información es la que se obtiene gracias a la librería Pyshark.

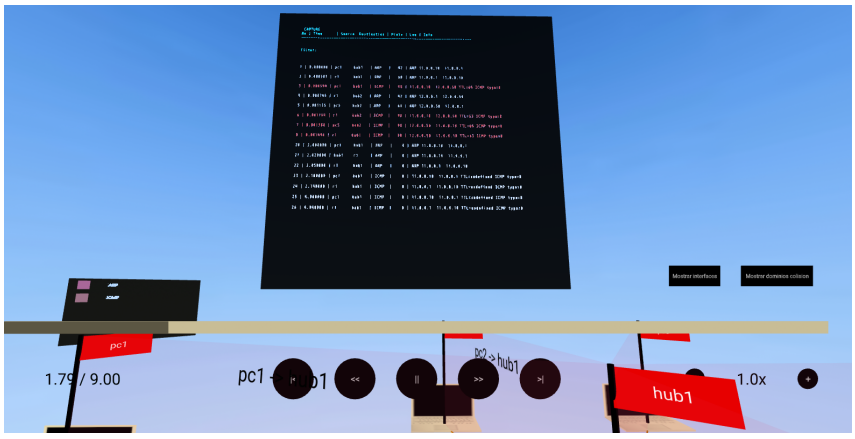


Figura 3.8: Ejemplo del visor de la traza

- **Leyenda de protocolos**: La aplicación genera una leyenda que asocia un color a cada protocolo detectado durante la simulación, actualizándose automáticamente a medida que aparecen nuevos protocolos. Esta leyenda facilita la identificación visual de los paquetes en función de su protocolo. Usa la misma lógica que Wireshark, es decir, indica el protocolo de más alto nivel del paquete; si, por ejemplo, tiene un protocolo de la capa de aplicación, se identificará con este.

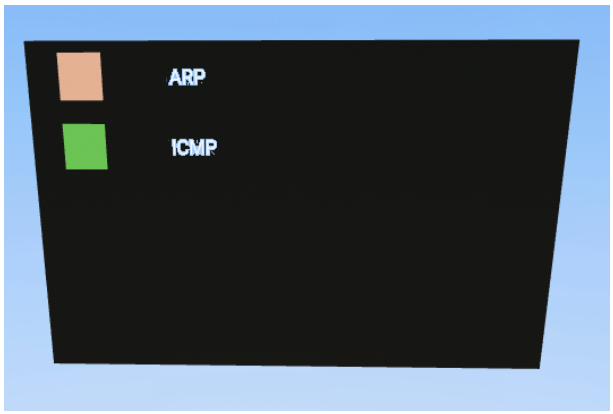


Figura 3.9: Ejemplo de leyenda de protocolos

- **Carteles informativos**: Estos cuadros de texto aparecen encima de cada nodo de red de la topología, indicando el nombre que tiene dentro de la simulación. Una característica diferencial de estos es que permanecen orientados hacia la posición de la cámara, permitiendo una correcta visualización independientemente del lugar desde el que el usuario esté viendo la escena.

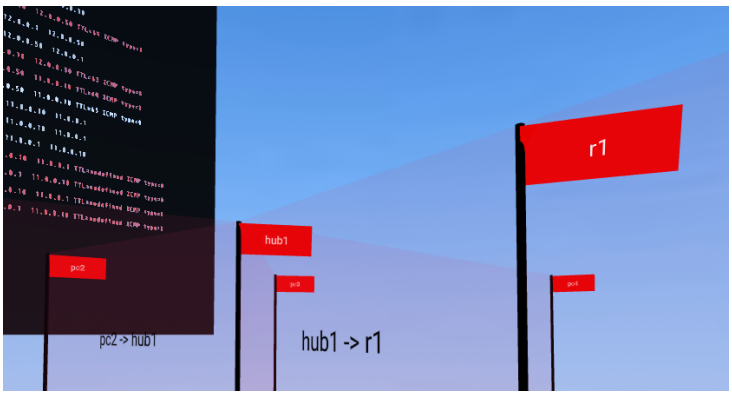


Figura 3.10: Ejemplo de banderín informativo

Además de la ejecución mediante dispositivos XR, la aplicación puede utilizarse con todas

sus funcionalidades desde un navegador.

3.1.9. Caso de uso

Para entender el funcionamiento se halla en el anexo un manual de usuario explicando paso a paso cómo usar la herramienta para explotar al máximo sus funcionalidades.

Por ahora, se va a explicar un caso de uso para un usuario final. Como ejemplo de utilización, se ha considerado una topología sencilla compuesta por varios 5 PCs conectados a través de 2 hubs y un router intermedio; mientras que la captura se compone de distintos paquetes que representan un intercambios de mensajes ping entre los PCs 3.11.

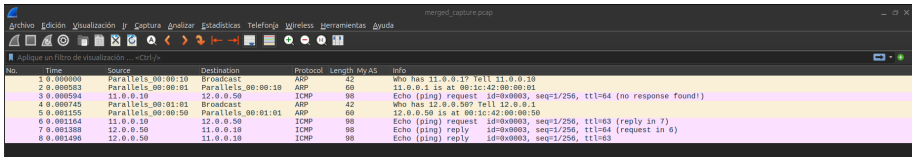


Figura 3.11: Captura para la simulación de ping

Tras generar los ficheros necesarios, el usuario inicia la aplicación desarrollada y carga los datos correspondientes a la simulación, usando los ficheros JSON de la captura de tráfico por un lado, y de la topología del escenario, por otro. A partir de ese momento, la herramienta genera automáticamente la representación de la topología en A-Frame 3.12 y comienza la reproducción de la captura.

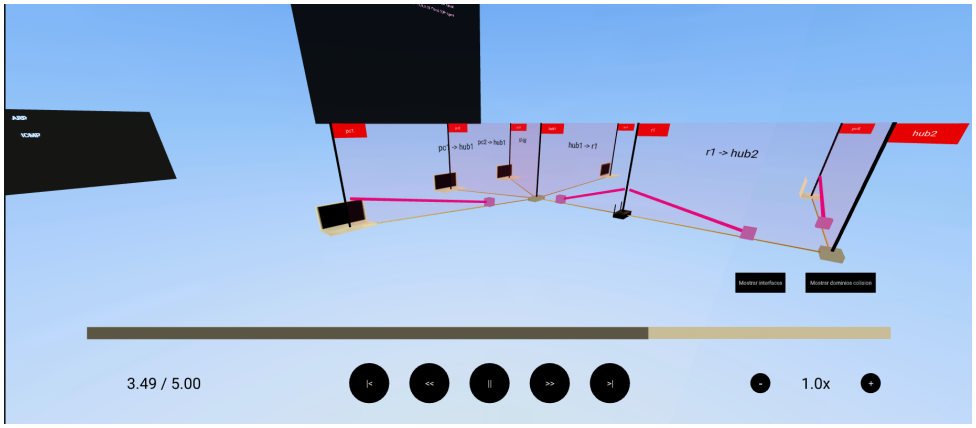


Figura 3.12: Escenario Ping

Durante la ejecución, el usuario puede observar la aparición de los paquetes y su desplaza-

miento a través de los enlaces de la red. A partir de aquí, el usuario puede interactuar con la representación como necesite, usando todas las funcionalidades que se han descrito antes. En este caso, podemos aislar, por ejemplo, un panel, como puede ser el de la conexión de r1 con el hub2 y pausar la reproducción, con ello podemos observar los paquetes que han viajado gracias a las huellas 3.13.

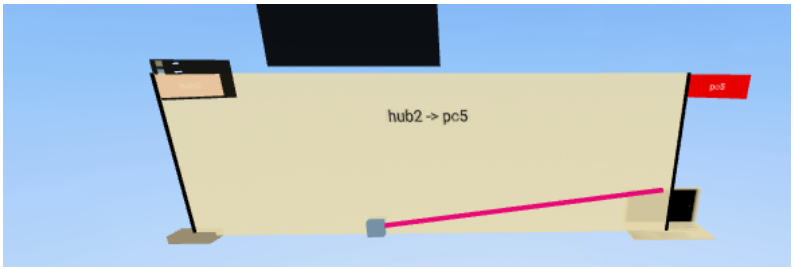


Figura 3.13: Escenario Ping con el panel aislado

También es útil fijarse en el visor con la traza completa, en él, podemos ver todos los paquetes que hay en ese momento en la traza, así que viajamos hasta el final del tiempo de reproducción y observamos con detalle los mensajes capturados de esta simulación 3.14.

Figura 3.14: Visor de la traza completa para el escenario Ping

Con todo esto, la aplicación permite al usuario entender visualmente conceptos que requieren un nivel de abstracción superior cuando se utilizan herramientas de análisis de tráfico que muestran cada paquete como una fila más.

3.2. Descripción de la implementación

Aunque para el usuario la aplicación se presenta como una única herramienta, internamente está formada por varios componentes A-Frame que colaboran entre sí. Diseñarlos y delimitar sus funciones correctamente es la clave de la aplicación, ya que esto definirá si la aplicación puede tener un techo u otro.

3.2.1. Arquitectura interna

La Figura 3.15 representa los componentes de la aplicación desarrollados durante el proyecto y cómo se relacionan entre sí.

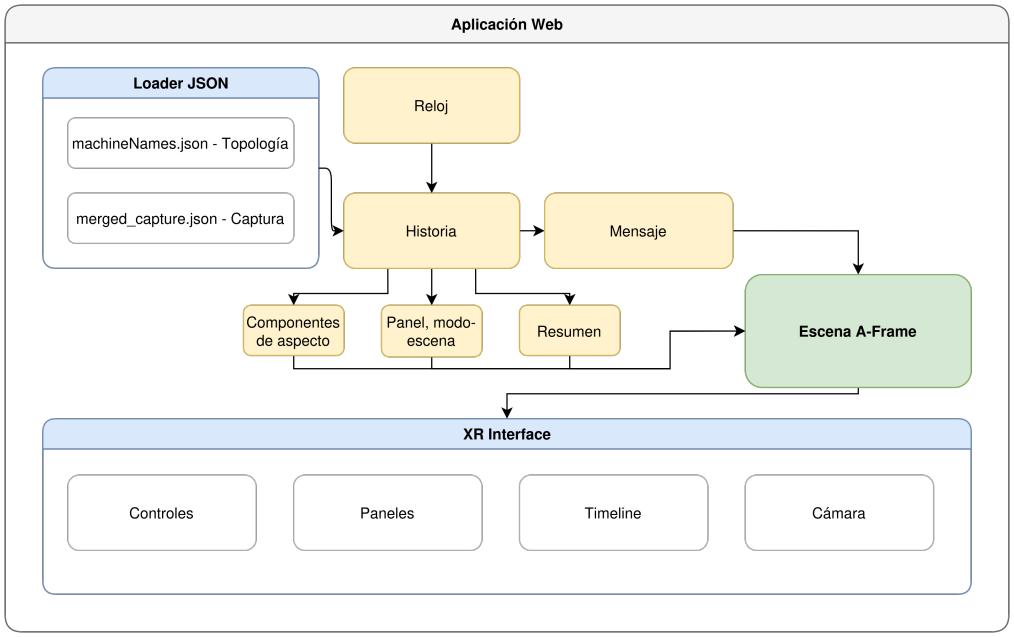


Figura 3.15: Arquitectura interna de la aplicación

Aunque la aplicación se encuentra dividida en varios componentes separados, ninguno de ellos funciona de manera aislada. En la imagen se puede comprobar que todos se relacionan entre sí; cada uno tiene una responsabilidad concreta, y es esencial la comunicación mediante eventos y referencias a entidades A-Frame. Cuanta más individualización exista en las funciones del programa, es decir, que haya un componente para cada responsabilidad, más sencillo es mantener el código e incorporar nuevas funcionalidades sin modificar el resto del sistema.

El punto de entrada de toda la aplicación es el componente *historia*; durante su ejecución

carga los ficheros JSON y construye toda la topología, asignando mediante *setAttribute* los componentes correspondientes a cada entidad, estableciendo así la relación entre *historia* y los componentes encargados del aspecto visual (*ordenador*, *router*, *switch*, *hub* y *cable*).

Una vez creada la escena, la comunicación ocurre entre los componentes *reloj* e *historia*. El componente reloj mantiene el tiempo interno de la simulación y, en cada tick generado, emite un evento con el instante temporal actual. Historia lo recibe y ejecuta la función *gestionarMensajes()*, donde consulta la estructura generada por *generarPaquetesPorTiempo()* para determinar qué mensajes deben actualizarse durante ese instante.

Tras definir el estado de cada paquete, se envía la información al componente *mensaje*, indicando si el paquete debe crearse, desplazarse o finalizar su recorrido. Con esto, la lógica temporal se concentra en Historia, mientras que Mensaje solo se encarga de las entidades tridimensionales.

Cuando el usuario solicita información sobre un dispositivo, una interfaz o un dominio de colisión, Historia consulta las estructuras de datos que tiene y envía al componente *panel* la información necesaria para construir los carteles correspondientes. Del mismo modo, Historia actualiza la altura de los ejes temporales y de los paneles modificando directamente los atributos de dichas entidades.

El componente *panel* tiene la información de que elementos forman su conexión, cuando el usuario selecciona uno de los paneles asociados a un enlace, se envía un evento para activar el modo de visualización individual de este. El componente *modo-escena* recorre entonces las entidades de la escena y los modifica para mostrar únicamente los elementos relacionados con la conexión.

El componente *resumen* utiliza la información procesada por Historia para construir la tabla HTML inspirada en Wireshark. Su funcionamiento es completamente independiente de la escena de A-Frame, este solo genera y actualiza elementos del DOM convencional.

El diseño de los componentes tiene el objetivo de minimizar el número de modificaciones posibles sobre el DOM.

3.2.2. Instanciación de los componentes

El punto de entrada del sistema es el fichero *index.html*, donde se define la estructura inicial de la aplicación y se cargan todos los recursos necesarios.

En el *header* del documento se importan las bibliotecas A-Frame y Three.js, así como todos los ficheros JavaScript que implementan los componentes de los que se va a hablar. Durante la carga de estos scripts, cada uno de ellos registra su componente mediante la llamada a *AFRAME.registerComponent()*.

Una vez registrados, el propio *index.html* define la escena inicial mediante la entidad *a-scene*, sobre la que se instancian los componentes encargados de gestionar el funcionamiento global de la aplicación. Concretamente, los atributos *mensaje*, *reloj*, *historia*, *modo-escena* y *resumen* hacen que A-Frame cree una instancia de cada uno de estos componentes asociada a la escena completa. De este modo, todos pueden acceder a cualquier entidad que forme la simulación.

```
  <script src="https://aframe.io/releases/1.5.0/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-look-at-component@1.0.0/dist/aframe-look-at-component.min.js"></script>

  <script src="mensaje.js" defer></script>
  <script src="reloj.js" defer></script>
  <script src="historia.js" defer></script>
  <script src="aspecto.js" defer></script>
  <script src="panel.js" defer></script>
  <script src="visor.js" defer></script>
</head>

  <a-scene
    fog="type: exponential; color: #cbe8ff; density: 0"
    mensaje
    reloj
    historia
    modo-escena
    resumen>
```

Figura 3.16: Instanciación de componentes en el HTML

Otros componentes se instancian sobre entidades específicas del documento HTML, como ocurre con el componente *slider*, que se asigna a una entidad independiente dentro de la interfaz del reproductor.

Por otro lado, hay componentes que no aparecen definidos en el *index.html*. Estos componentes son creados dinámicamente durante la ejecución por el componente *historia*, utilizando *setAttribute()*. Es el caso de los componentes de aspecto (*ordenador*, *router*, *switch*, *hub* y *cable*), así como de los paneles asociados a los enlaces.

Cuando el navegador contruye el DOM, A-Frame recorre todas las entidades. Para cada atributo que corresponde a un componente crea una instancia y ejecuta su *init()*.

```
const e = document.createElement('a-entity');

// Tipo real ahora viene en JSON
if (nodo.type === 'host') e.setAttribute('ordenador', '');
else if (nodo.type === 'router') e.setAttribute('router', '');
else if (nodo.type === 'switch') e.setAttribute('switch', '');
else if (nodo.type === 'hub') e.setAttribute('hub', '');
```

Figura 3.17: Instanciación de componentes en el componente *historia*

3.2.3. Componente Historia

Responsabilidad

El componente *historia* es el núcleo de la aplicación y es el encargado de coordinar prácticamente toda la lógica de la simulación. Su responsabilidad principal se podría resumir en que transforma los ficheros JSON en una representación interna útil para el resto de componentes. Por este motivo, es el componente que gestiona las estructuras de datos y el que hace de interlocutor entre la mayoría del resto de componentes de la aplicación.

Inicialización

Toda la inicialización del componente se realiza en el método *init()*; durante esta fase se cargan los dos ficheros JSON, que son los parámetros de entrada del componente. La primera tarea consiste en procesar la topología. Para ello, recorre la lista de nodos almacenada en el fichero JSON y, para cada uno de ellos, crea dinámicamente una entidad de A-Frame cuyo aspecto depende del tipo de dispositivo representado; en función de si el nodo corresponde a un ordenador, un router, un switch o un hub, se le aplica con *setAttribute* el elemento correspondiente, el cual creará el aspecto con el componente aspecto que se comenta más adelante. Además de asociar la representación gráfica, durante este proceso también se almacenan como atributos de cada entidad datos asociados al dispositivo, como las interfaces de red, por ejemplo. Así se evita tener que consultar nuevamente el fichero JSON más adelante.

Una vez creados todos los nodos, se tratan las conexiones existentes entre ellos; con cada enlace se generan tres elementos. En primer lugar, el cable, con el mismo proceso que los nodos. Después, se crea un panel transparente sobre el enlace, que posteriormente se convierte en los paneles internodos comentados antes. Finalmente, se generan unos ejes asociados a cada nodo, que se utilizan para representar la evolución temporal conforme avanza la reproducción y que

los paneles crezcan acorde a estos.

Ya con la escena construida, el componente procesa la captura de tráfico. Para ello recorre cada paquete almacenado en el fichero y extrae de él la información necesaria como el instante temporal, los nodos origen y destino y otra información adicional A partir de estos datos se construye una estructura interna común para todos los mensajes de la simulación, que consiste en una lista (*this.historia*s), con cada paquete, el cual se estructura con un diccionario con cada clave y su valor.

Aparte de esto, se calcula el intervalo temporal durante el cual el mensaje debe ser visible en la simulación. Para ello se determina el instante de aparición y el que el paquete llega al nodo destino. Ambos valores sirven para calcular posteriormente la posición exacta del mensaje en cualquier instante, evitando almacenar todas las posiciones intermedias.

Funciones implementadas

Una vez procesados los ficheros, los paquetes se reorganizan en la función *generarPaquetesPorTiempo()*. En lugar de seguir usando la lista secuencial de mensajes que se comenta antes, el componente construye un diccionario cuyo índice es el instante temporal de la simulación. Cada entrada contiene aquellos paquetes que deben actualizarse en ese instante. Con esta organización no resulta necesario recorrer la captura de nuevo, únicamente acceder al instante actual.

Con los ticks recibidos del componente reloj, se ejecuta la función *gestionarMensajes()* 3.18, que es responsable de determinar el estado de los paquetes activos. Los recorre y calcula su progreso relativo respecto a su recorrido. Con esto se saca la conclusión para el paquete; si aparece por primera vez, si debe desplazarse a una nueva posición o finalizar su recorrido. En función del estado se emite el evento correspondiente.

Otra de las funciones es la de reconstruir el estado de la simulación cuando el usuario realiza un salto temporal. Es la función *forzarEstadoPorSaltoTemporal()*, que reinicia todos los elementos, ocultando los mensajes existentes, reiniciando las trazas sobre los paneles y restaurando la altura tanto de los paneles como de los ejes. Por último, se simulan internamente todos los ticks hasta el instante seleccionado por el usuario.

Además, el componente implementa varias de las funcionalidades interactivas de la aplicación aprovechando la lista con la información de los mensajes. Entre ellas, la creación de los

```
// CREAR
if (progreso >= 0 && p.ultimoProgreso === null) {
  this.el.emit('mensaje', {
    id: p.id,
    x, y, z,
    estado: 'Crear',
    conexion: p.conexion,
    visible,
    protocolo : p.protocolo
  });
}

// MOVER
if (progreso >= 0) {
  this.el.emit('mensaje', {
    id: p.id,
    x, y, z,
    estado: 'Mover',
    conexion: p.conexion,
    visible,
    protocolo : p.protocolo
  });
}

// FINAL
if (progreso >= 1 && (p.ultimoProgreso === null || p.ultimoProgreso < 1)) {
  this.el.emit('mensaje', {
    id: p.id,
    x, y, z,
    estado: 'Acabar',
    conexion: p.conexion,
    visible,
    protocolo : p.protocolo
  });

  this.el.emit('mensaje-llegado', {
    id: p.id,
    destino: p.destinoNom,
    tiempo
  });
}
```

Figura 3.18: Código de la función *gestionarMensajes()*

carteles correspondientes a las interfaces de red, los dominios de colisión, los nombres de cada dispositivo y las tablas de encaminamiento. Todas estas funcionalidades se crean dinámicamente utilizando la información almacenada en *this.historias* y creando entidades *a-plane* con el texto delante.

Finalmente, se encarga de actualizar la altura de los ejes temporales y de los paneles conforme avanza el reloj. Esto lo hace calculando la altura con el tiempo de simulación y modificándola en cada entidad con un *setAttribute*.

3.2.4. Componente Reloj

Responsabilidad

El sistema temporal de la aplicación está implementado mediante dos componentes, *reloj* y *slider*, que trabajan juntos para controlar la reproducción. El primero, reloj, mantiene el tiempo interno.

Inicialización

El componente tiene una serie de variables internas: el instante actual de la simulación, el estado de reproducción y la velocidad. Cuando se inicia la reproducción se crea un temporizador que genera eventos periódicos, es decir, funciona con tiempo discreto; en cada uno de estos eventos se emite un evento con el instante actual, que posteriormente es recibido por el componente *historia*. De esta forma, el reloj únicamente mantiene el tiempo y no contiene ninguna lógica relacionada con los mensajes.

La lógica del componente se incluye también en el *init()*, donde se obtienen las referencias a los distintos botones del reproductor, que se definen en el HTML como entidades A-Frame, al igual que el deslizador temporal, que se define como un *a-plane*. Cada uno de estos elementos registra un *event listener*, así, se ejecuta la función correspondiente cuando el usuario interactúa con él y recibe un evento.

```
// Botones
document.querySelector('#btn-pausa').addEventListener('click', () => {
  this.pausado = !this.pausado;
});

document.querySelector('#btn-avanzar').addEventListener('click', () => {
  this.direccion = 1;
});

document.querySelector('#btn-retroceder').addEventListener('click', () => {
  this.direccion = -1;
});

document.querySelector('#btn-inicio').addEventListener('click', () => {
  this.tiempo = 0;
  this.ultimoTick = 0;
  this.el.emit('set-time', this.tiempo);

  this.el.sceneEl.emit('salto-temporal', { tiempo: 0 });
});

document.querySelector('#btn-final').addEventListener('click', () => {
  this.tiempo = this.maxTiempo;
  this.el.emit('set-time', this.tiempo);

  this.el.sceneEl.emit('salto-temporal', { tiempo: this.maxTiempo });
});

document.querySelector('#btn-vel-mas').addEventListener('click', () => {
  this.velocidad = Math.min(this.velocidad + 0.1, 5);
});

document.querySelector('#btn-vel-menos').addEventListener('click', () => {
  this.velocidad = Math.max(this.velocidad - 0.1, 0.1);
});
```

Figura 3.19: EventListeners() del componente *reloj*

3.2.5. Componente Slider

Responsabilidad

El componente slider, es la segunda pata del control de la reproducción temporal; proporciona la representación gráfica de dicho tiempo. Cada vez que el reloj modifica el instante de simulación, actualiza la posición del deslizador mediante *setAttribute*; esto lo hace calculando el progreso relativo respecto al tiempo total de la simulación.

Esta vista no la forman más que un *a-plane* rectangular, junto a un *a-circle*, del cual se cambia su atributo posición dinámicamente.

3.2.6. Componente Mensaje

Responsabilidad

El componente *mensaje* es el encargado de representar en la escena todos los paquetes que aparecen durante la simulación. A diferencia de Historia, no tiene que decidir sobre el estado de los mensajes, sino que únicamente ejecuta los estados que recibe.

Inicialización

Cuando llega un estado Çrear", el componente comprueba si ya existe una entidad asociada al mensaje; de lo contrario, crea una nueva entidad *a-sphere*, configura sus atributos visuales mediante varias llamadas a *setAttribute* y la almacena en un diccionario llamado *this.entidades*.

En las siguientes actualizaciones, en las que llega un estado "Mover", ya no es necesario volver a crear la entidad, simplemente se modifica su posición utilizando los valores calculados por Historia. Cuando el mensaje finaliza su recorrido (."Acabar") no se elimina, sino que se oculta modificando su visibilidad. Para que se pueda reutilizar si vuelve a ser necesario.

Además de mover los paquetes, también genera las huellas que aparecen sobre los paneles. Cada vez que un mensaje avanza se crea una entidad *a-sphere* en la posición correspondiente.

Aparte de esto, conforme aparecen nuevos tipos de paquetes, el sistema comprueba si el protocolo al que pertenece el paquete ya dispone de un color asociado. Si todavía no existe, genera uno nuevo, lo almacena en un diccionario (*this.coloresProtocolos* y actualiza la leyenda.

```
// Crear traza (una sola vez)
crearTrazaSiNoExiste: function (id, conexion, x, y, z) {
  if (this.trazas[id]) return this.trazas[id];
  if (!this.paneles) return null;

  const panel = this.paneles.find(p => this.mismaConexion(p.id, conexion));
  if (!panel) return null;

  const cilindro = document.createElement('a-cylinder');
  cilindro.setAttribute('radius', 0.03);
  cilindro.setAttribute('height', 0.01);
  cilindro.setAttribute('color', '#ff00aa');
  cilindro.setAttribute('opacity', 0.8);

  panel.appendChild(cilindro);

  // Guardamos el punto inicial EN LOCAL DEL PANEL
  const worldStart = new THREE.Vector3(x, y, z);
  const localStart = panel.object3D.worldToLocal(worldStart.clone());

  this.trazas[id] = {
    panel: panel,
    cilindro: cilindro,
    inicio: localStart
  };

  return this.trazas[id];
}
```

Figura 3.20: Código encargado de crear las huellas

3.2.7. Componentes de Aspecto

Responsabilidad

La representación tridimensional de cada elemento de la topología se gestiona mediante varios componentes especializados.

Inicialización

Cuando Historia crea una entidad correspondiente a un nodo, únicamente le asigna el componente correspondiente mediante *setAttribute* (ordenador, router, switch, hub). Entonces, el propio componente es el que, durante su función *init()*, genera las primitivas necesarias para construir el modelo. Dependiendo del dispositivo se crean diferentes combinaciones de cajas, cilindros y planos junto con las texturas correspondientes 3.21.

Para cada elemento cambia el diseño, que suele combinar una serie de planos rotados, en vez de un único cubo, por ejemplo, para poder así, personalizar cada cara del elemento, permitiendo añadir texturas; también son distintas las alturas, anchos, etc.

El componente *cable* sigue un funcionamiento similar, aunque tiene mayor complejidad. Durante la inicialización recibe las posiciones de los dos nodos que debe conectar y calcula el vector que une ambos puntos; con este vector obtiene la longitud, la posición del punto medio y

```
AFRAME.registerComponent('ordenador', { // Crea los ordenadores
  schema: {
    ancho: {type: 'number', default: 1},
    fondo: {type: 'number', default: 0.5},
    altoPantalla: {type: 'number', default: 0.5}
  },
  init: function () {
    const el = this.el;

    const {ancho, fondo, altoPantalla} = this.data;

    const base = document.createElement('a-box');
    base.setAttribute('depth', fondo);
    base.setAttribute('width', ancho);
    base.setAttribute('height', 0.02);
    base.setAttribute('position', `0 0 0`);
    base.setAttribute('material', 'src: img/teclado.jpg');
    base.classList.add('nodo');
    el.appendChild(base);

    const front = document.createElement('a-plane');
    front.setAttribute('material', 'src: img/pantalla.jpg');
    front.setAttribute('width', ancho);
    front.setAttribute('height', altoPantalla);
    front.setAttribute('position', `0 ${altoPantalla/2} ${-fondo/2+0.01}`);
    front.setAttribute('rotation', '0 0 0');
    el.appendChild(front);

    const sides = [ // Cada lado que no tiene imagen
      {pos: `0 ${altoPantalla/2} ${-fondo/2}`, rot: '180 0 0', w: ancho, h: altoPantalla},
      {pos: `${ancho/2} ${altoPantalla/2} ${-fondo/2+0.005}`, rot: '0 90 0', w: 0.01, h: altoPantalla},
      {pos: `${-ancho/2} ${altoPantalla/2} ${-fondo/2+0.005}`, rot: '0 -90 0', w: 0.01, h: altoPantalla},
      {pos: ` 0 ${altoPantalla/2} ${-fondo/2+0.005}`, rot: '90 0 0', w: ancho, h: 0.01},
    ];

    sides.forEach(side => {
      const plane = document.createElement('a-plane');
      plane.setAttribute('color', '■#ffffff');
      plane.setAttribute('width', side.w);
      plane.setAttribute('height', side.h);
      plane.setAttribute('position', side.pos);
      plane.setAttribute('rotation', side.rot);
      plane.setAttribute('material', 'side: double');
      el.appendChild(plane);
    });
  }
});
```

Figura 3.21: Código encargado de crear el aspecto de los PCs

la orientación del enlace. Con todo esto se configura entidad mediante llamadas a *setAttribute*.

Esta organización permite modificar la apariencia de un único dispositivo sin afectar al resto de componentes de la aplicación, ya que toda la información relacionada con su representación permanece encapsulada. Todos estos componentes se encuentran en el fichero *aspecto.js*.

3.2.8. Componente Panel

Responsabilidad

Cada panel se crea dinámicamente durante la construcción de la topología y queda asociado al enlace correspondiente.

Inicialización

Durante la inicialización, el componente *Panel*, registra distintos eventos de interacción sobre la entidad. Cuando el usuario selecciona uno de estos paneles se emite un evento que es recibido por el componente *modo-escena*, que se encarga de modificar la visualización del escenario.

La implementación de este modo consiste en recorrer todas las entidades de la escena y modificar su visibilidad mediante *setAttribute*, incluyendo los paquetes; los únicos elementos que permancen son los pertenecientes a la conexión.

Además de los paneles, el componente genera todos los carteles informativos utilizados por la aplicación. Para ello crea dinámicamente entidades *a-plane*, configura su tamaño, posición y texto mediante atributos y se encarga de ir actualizando el contenido cuando le llega información nueva desde *historia*. Esto incluye las interfaces de red, las tablas de encaminamiento, los dominios de colisión y la leyenda de protocolos.

3.2.9. Componente Visor

Responsabilidad

El componente *resumen* implementa el visor de paquetes inspirado en Wireshark; la mayor parte de su funcionamiento se crea usando el DOM de HTML en lugar de entidades A-Frame.

Inicialización

Durante la inicialización obtiene la referencia a la tabla donde se va a ubicar el texto y construye cada fila recorriendo la lista de mensajes creada por Historia. Para cada paquete crea los elementos HTML correspondientes a cada campo, insertándolos posteriormente mediante operaciones sobre el DOM.

Cada fila incorpora un segundo bloque oculto con la información completa de cada capa del protocolo. Está preparado para que cuando el usuario seleccione una fila, se modifique su estado mediante clases CSS, mostrando u ocultando la información.

Para reconocer el paso del ratón por un paquete y resaltarlo se crea el componente *cursor-listener*, de apenas unas líneas, que usa escucha los eventos *mouseenter* y *mouseleave*.

Capítulo 4

Desarrollo del proyecto

Este capítulo describe el proceso seguido durante el desarrollo del proyecto. Con el objetivo de organizar el trabajo de forma progresiva y controlar la incorporación de nuevas funcionalidades, se adoptó una metodología Agile basada en sprints, que son ciclos de duración corta en los que se proponen una serie de objetivos y tareas necesarias para alcanzarlos. Cada sprint tuvo una duración aproximada de tres semanas, aunque en determinadas fases del proyecto se realizaron iteraciones más cortas debido a la intensidad del desarrollo.

Aunque en el repositorio se pueden ver hasta 14 carpetas distintas, cada nueva carpeta corresponde a una reunión con el tutor para repasar el estado del proyecto. Estas reuniones se han producido en periodos más cortos que los sprints, es decir, un sprint puede contener más de una reunión.

La evolución del sistema no fue lineal. Muchas de las ideas planteadas inicialmente fueron modificadas o sustituidas conforme aparecían nuevos requisitos y limitaciones técnicas. Por este motivo, cada sprint no solo incorporó nuevas funcionalidades, sino que también permitió replantear decisiones de diseño tomadas en fases anteriores.

En las siguientes secciones se describen los objetivos perseguidos en cada sprint, las tareas realizadas y los principales resultados obtenidos.

4.1. Sprint 1: Diseño conceptual del escenario XR

4.1.1. Objetivos

Esta fase marca el incio del proyecto y, más allá de los primeros objetivos técnicos, el principal objetivo de esta primera fase fue comprender el alcance real del proyecto y explorar la viabilidad de utilizar tecnologías XR para representar topologías de red y capturas de tráfico de forma intuitiva, en resumidas cuentas, ver si era viable y, en caso positivo, cuál podía ser un objetivo realista.

En lo técnico, lo principal fue familiarizarse con las herramientas seleccionadas, era necesario encontrar una propuesta visual capaz de resultar atractiva para el usuario sin comprometer el rendimiento de la aplicación.

También se pretendía definir una primera idea de cómo representar dispositivos de red, conexiones y elementos interactivos dentro de un espacio tridimensional.

4.1.2. Tareas realizadas

Se estudiaron diferentes alternativas para representar routers, switches, ordenadores y enlaces dentro de un entorno XR. Para ello se tomaron múltiples referencias como el trabajo realizado previamente por Enrique Estebaranz, la propia interfaz gráfica de NetGUI, videojuegos, buscando una visualización moderna pero intuitiva.

En estas primeras pruebas empezó a definirse la idea de representar cada tipo de dispositivo en componentes independientes de A-Frame, lo que se convertiría posteriormente en los componentes *ordenador*, *router*, *switch*, *hub* y *cable* en *aspecto.js*.

También se exploraron diferentes formas de navegación dentro del escenario, posibles mecanismos de interacción y alternativas para mostrar información adicional asociada a cada dispositivo.

4.1.3. Resultados y conclusiones

El resultado principal de este sprint fue una propuesta inicial de interfaz capaz de representar topologías de red dentro de un entorno tridimensional de forma comprensible.

Aunque el diseño obtenido todavía estaba lejos del resultado final, permitió adquirir experiencia con las tecnologías seleccionadas y establecer una dirección clara para el resto del desarrollo. Además, sirvió para identificar las primeras limitaciones relacionadas con el rendimiento y la representación visual.

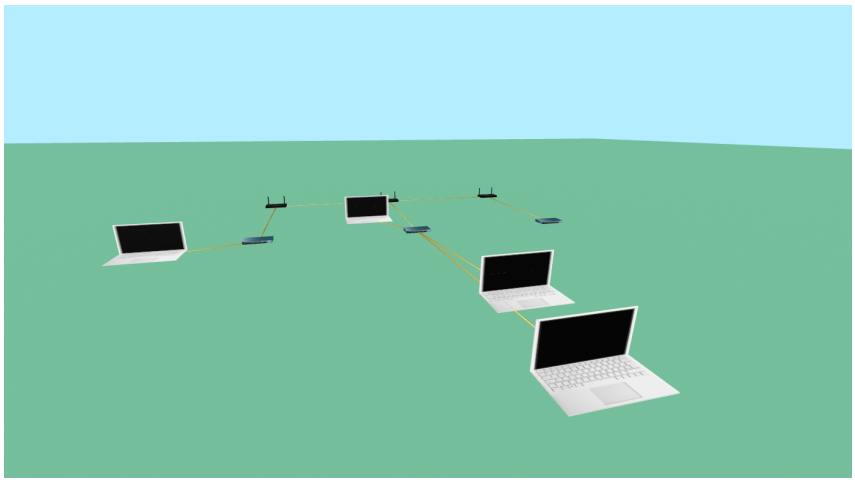


Figura 4.1: Resultado obtenido durante el Sprint 1

4.2. Sprint 2: Primer modelo temporal y representación de paquetes

4.2.1. Objetivos

Una vez definida una propuesta visual inicial, el siguiente objetivo consistía en representar el comportamiento de los paquetes en la red. Era necesario encontrar una forma de modelar paquetes, sincronizar eventos temporales y almacenar la información que fuera necesaria sobre la evolución de los mensajes.

4.2.2. Tareas realizadas

Se desarrolló un primer prototipo funcional basado en tres componentes fundamentales: *mensaje*, *reloj* e *historia*.

- En esta versión el componente *mensaje* se encargaba de representar visualmente los paquetes dentro del escenario, creando una nueva entidad A-Frame en cada actualización

recibida, lo que es muy poco escalable.

- El componente *reloj* gestionaba la evolución temporal mediante una escala discreta basada en ticks, cada uno de estos ticks correspondiendo a un momento concreto en el que el paquete tendría que hacer la acción correspondiente.

- El componente historia almacenaba los eventos asociados a cada mensaje en una única lista.

Se implementaron las primeras animaciones de movimiento entre nodos y se realizaron pruebas para verificar la sincronización entre la variable temporal y la representación gráfica.

4.2.3. Resultados y conclusiones

Este sprint permitió validar la viabilidad de utilizar una línea temporal controlada por el usuario para reproducir capturas de tráfico, todavía, sin embargo, en una fase muy inicial.

También fue una fase muy importante porque permitió descubrir de primera mano los primeros problemas relacionados con la gestión temporal de los eventos, especialmente cuando varios paquetes coexistían simultáneamente dentro del sistema. Estas dificultades motivarían posteriormente importantes cambios en la arquitectura interna.

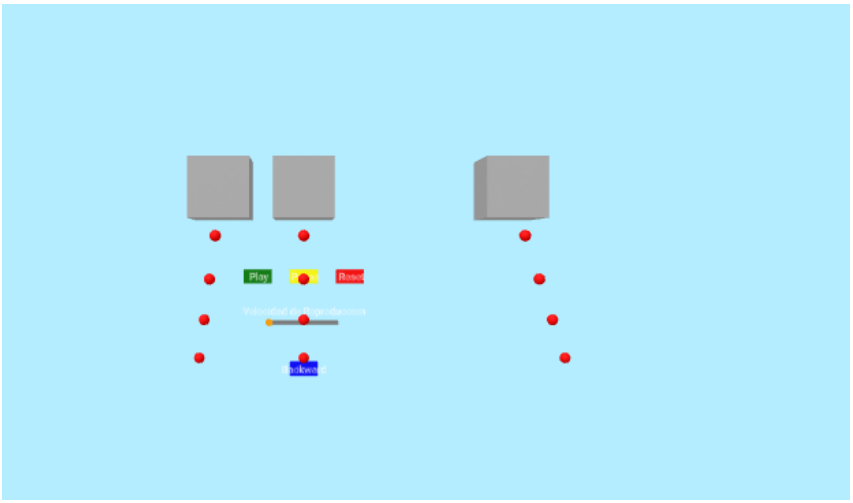


Figura 4.2: Resultado obtenido durante el Sprint 2

4.3. Sprint 3: Modelo avanzado de mensajes e historial

4.3.1. Objetivos

Pensando en llegar a escenarios más complejos y con muchos paquetes coexistiendo al mismo tiempo, el objetivo principal que se definió en este sprint fue reorganizar la arquitectura desarrollada hasta el momento, para desacoplar la representación gráfica de la gestión temporal de eventos.

4.3.2. Tareas realizadas

Se redefinieron las responsabilidades de los componentes existentes y se creó un modelo basado en parámetros de entrada para la generación y movimiento de los mensajes. Para esto, se separó la función *gestionarMensajes()*, encargada de decidir el estado de cada paquete en cada instante, y el componente *mensaje*, responsable únicamente de ejecutar lo que reciba de *historia*.

El componente *historia* pasó a almacenar y organizar la información de los paquetes con la lista *this.historias*; y a suministrar la información temporal para la creación de paquetes, de la que se seguiría encargando el componente *mensaje*. Además, se incorporó soporte para tiempos continuos y discretos.

Con ello, se implementaron los estados *Crear*, *Mover* y *Acabar*, posteriormente gestionados por *gestionarMensajes()*.

4.3.3. Resultados y conclusiones

Este sprint estableció las bases de la arquitectura que, posteriormente, evolucionaría hasta convertirse en la versión final del sistema.

La separación de responsabilidades permitió simplificar el desarrollo de nuevas funcionalidades y facilitó la gestión de eventos temporales complejos.

Fue uno de los momentos claves del proceso, ya que permitió pasar de pensar en una lista reducida de paquetes a convertirse en una aplicación con una gran escalabilidad.

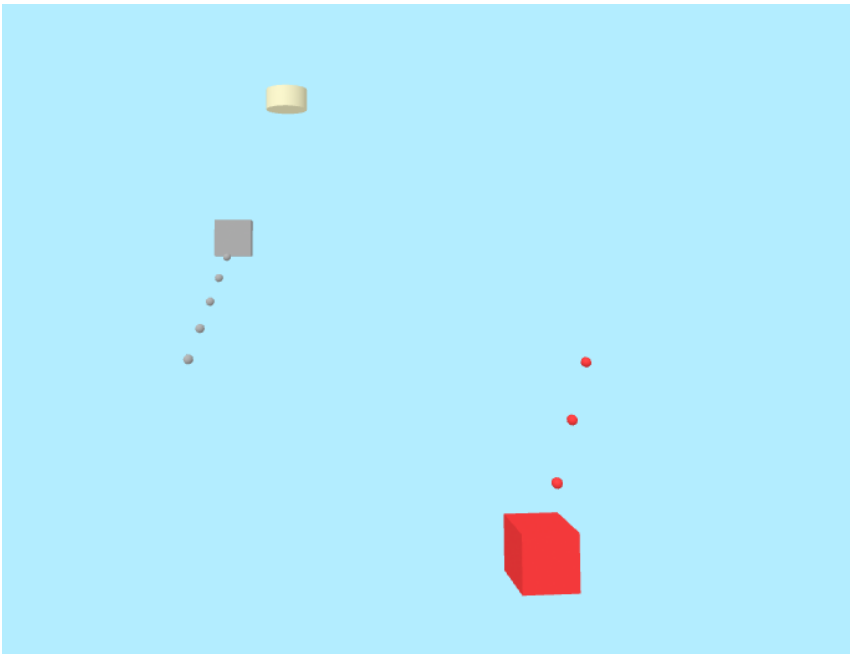


Figura 4.3: Resultado obtenido durante el Sprint 3

4.4. Sprint 4: Integración de datos mediante JSON y optimización de estructuras de datos

4.4.1. Objetivos

Eliminar la necesidad de usar datos definidos manualmente y permitir que la aplicación trabajase con escenarios configurables mediante ficheros externos, formato JSON.

También se planteó mejorar la eficiencia del sistema todo lo posible, esta mejora constante estará presente en la mayoría de sprints posteriores.

4.4.2. Tareas realizadas

Se diseñó una estructura basada en JSON para representar tanto la topología como los paquetes capturados.

Para ello, se modificó el método *init()* de *historia*, incorporando el método *fetch()* para cargar los ficheros JSON y el procesamiento tanto de la topología como de la captura.. No fue un cambio radical, ya que los datos con los que se trabajaba anteriormente tenían una estructura parecida.

Se creó la función *generarPaquetesPorTiempo()*, que reorganizaba la lista de mensajes almacenada en *this.historias* para indexarlos por instante temporal. Con esto, si llegaba el tick de reloj marcando X instante, las funciones necesarias se encargarían de ver qué le tocaba hacer al paquete en ese momento.

Paralelamente, se crearon scripts en Python para generar automáticamente escenarios de prueba con diferentes cargas de tráfico.

También se implementaron mecanismos de interpolación lineal para calcular posiciones intermedias durante las animaciones; tarea de la que después se encargaría *gestionarMensajes()*.

4.4.3. Resultados y conclusiones

La incorporación de JSON aumentó considerablemente la flexibilidad del sistema y permitió reutilizar escenarios. Mientras que las optimizaciones introducidas redujeron significativamente el coste computacional asociado a la gestión de eventos.

Aunque todavía existían limitaciones de escalabilidad, se consiguió un comportamiento mucho más eficiente que en versiones anteriores.

Estos cambios marcaron el inicio de la transición desde un prototipo experimental hacia una herramienta más cercana al uso real.

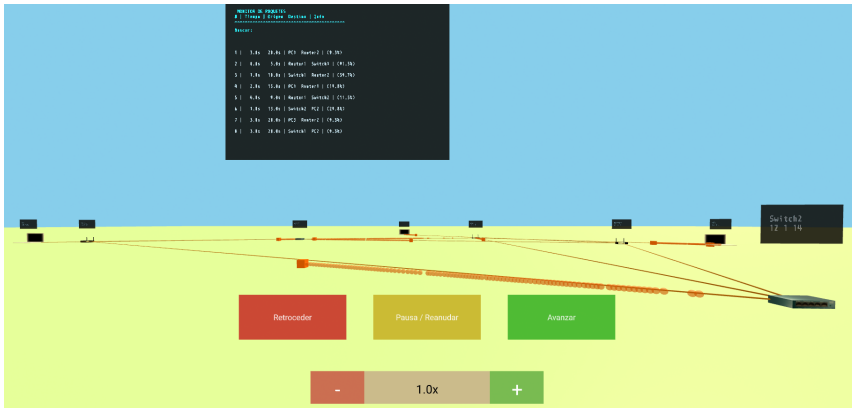


Figura 4.4: Resultado obtenido durante el Sprint 4

4.5. Sprint 5: Primera interfaz XR funcional

4.5.1. Objetivos

Hasta este momento, la herramienta era funcional para ejecutar una traza sobre un escenario, sin embargo, todavía era poco intuitiva, que es uno de los objetivos primordiales de la herramienta; así, se planteó transformar el prototipo técnico desarrollado hasta el momento en una herramienta interactiva mucho más enfocada a su uso desde dispositivos XR.

Esto incluye detalles como mejorar la comprensión visual de la evolución temporal del tráfico de red o incorporar mecanismos de navegación temporal que permitieran analizar la captura de tráfico de forma interactiva.

4.5.2. Tareas realizadas

Se diseñó una interfaz basada en una distribución espacial donde la topología se representaba sobre el suelo mientras que la dimensión temporal se visualizaba verticalmente.

Se introdujo el concepto de huella temporal, implementada dentro del componente *mensaje*, que crea entidades *a-sphere* conforme recibe eventos de movimiento.

Se creó el componente *panel*, encargado de generar los paneles asociados a cada enlace y de actualizar su altura mediante eventos emitidos por *historia* cada vez que recibe un tick. Capaces de mostrar visualmente la evolución de los paquetes a lo largo del tiempo. Más tarde, fue necesario rediseñar parte de la arquitectura para permitir que las huellas evolucionasen simultáneamente al crecimiento de dichos paneles.

Se implementaron además, funciones para:

- Reproducción. Pausa. Avance. Retroceso. Salto temporal.

Fue en este punto donde se comenzaron a realizar pruebas de usabilidad iniciales utilizando dispositivos XR, con el fin de evaluar la comprensión de la información mostrada.

4.5.3. Resultados y conclusiones

Las huellas temporales junto a los paneles demostraron ser una solución mucho más intuitiva para representar la dimensión temporal que otras alternativas exploradas anteriormente.

La aplicación pasó de ser una prueba tecnológica a convertirse en una herramienta con capacidad real de interacción, permitiendo al usuario sentir realmente que tiene poder sobre la escena y que esta le proporciona facilidades para entender las capturas.

Las pruebas realizadas permitieron identificar mejoras necesarias en la representación temporal y en la interacción con el usuario.

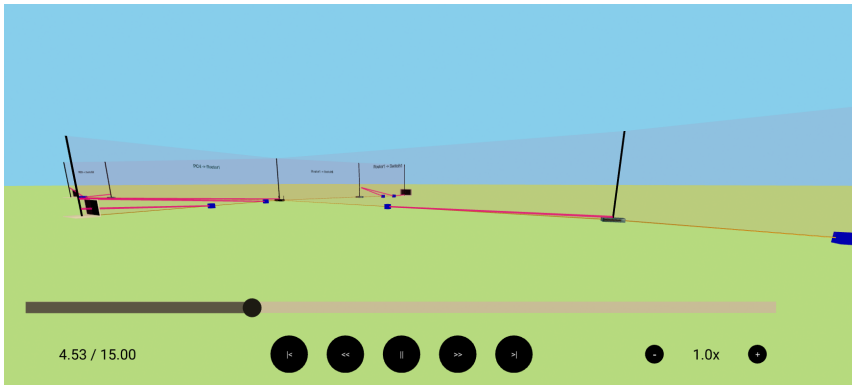


Figura 4.5: Resultado obtenido durante el Sprint 5

4.6. Sprint 6: Integración con ficheros JSON

4.6.1. Objetivos

Sustituir los escenarios sintéticos utilizados durante el desarrollo por datos reales procedentes de simulaciones de red, ya implementando la interoperabilidad con ficheros JSON propios.

4.6.2. Tareas realizadas

Se estudió el formato generado por NetGUI-Kathará, que genera automáticamente ficheros JSON, y se desarrollaron mecanismos para adaptarlo al modelo de datos utilizado por la aplicación, pensando en su uso con herramientas reales. Para ello, se amplió el procesamiento en el método *init()* de *historia* para construir automáticamente la escena mediante llamadas a *setAttribute()*.

También se realizaron pruebas utilizando capturas reales obtenidas mediante Wireshark.

4.6.3. Resultados y conclusiones

La integración permitió validar el sistema en escenarios mucho más próximos a un contexto docente real. A partir de este momento, la herramienta dejó de depender de datos generados artificialmente.

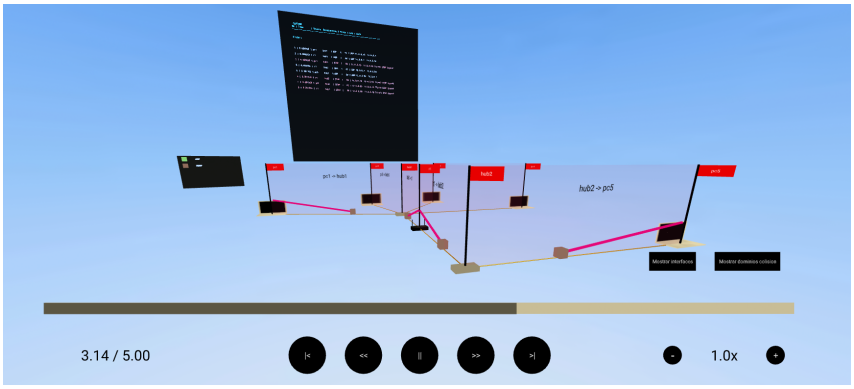


Figura 4.6: Resultado obtenido durante el Sprint 6

4.7. Sprint 7: Automatización, validación y despliegue

4.7.1. Objetivos

Preparar el sistema para su demostración final y automatizar el flujo completo de trabajo.

4.7.2. Tareas realizadas

Se completaron funcionalidades auxiliares implementadas en *historia*, como la creación dinámica de carteles de interfaces o de dispositivos; al igual que el componente *resumen*, que se desarrolló en una versión más avanzada.

Se verificó la comunicación basada en eventos entre *reloj*, *historia*, *mensaje*, *slider*, *panel* y *resumen*.

Se desarrolló un servidor Python encargado de servir la aplicación y monitorizar la aparición de nuevas capturas, pensado además, para futuras implementaciones en tiempo real.

También se realizaron pruebas finales utilizando el navegador de las Meta Quest 3, permitiendo validar el escenario en aspectos como la escala o la legibilidad del texto.

Finalmente, se preparó toda la documentación técnica y los materiales necesarios para la presentación del proyecto, como es la web que se usará para mostrar el proyecto.

4.7.3. Resultados y conclusiones

El resultado final fue un prototipo funcional capaz de representar capturas reales de tráfico de red dentro de un entorno XR interactivo.

Este sprint marcó la transición desde una versión experimental hacia una herramienta demostrable y suficientemente estable para ser evaluada como un Trabajo de Fin de Grado.

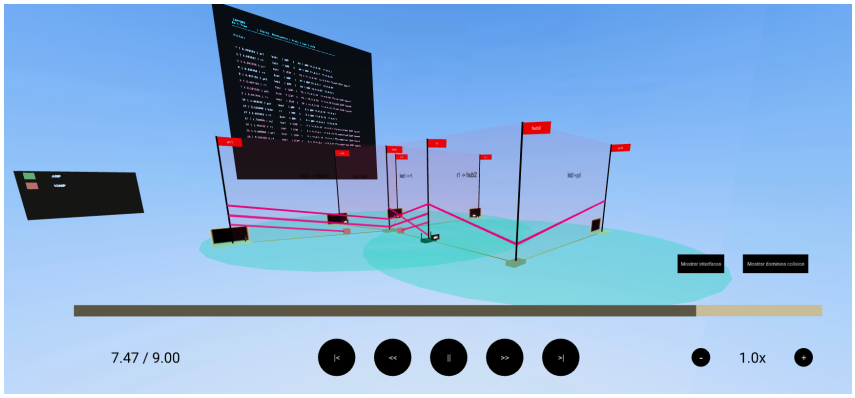


Figura 4.7: Resultado obtenido durante el Sprint 7

Capítulo 5

Conclusiones

5.1. Consecución de objetivos

El objetivo principal que se plantea al inicio de este Trabajo de Fin de Grado es desarrollar una herramienta capaz de representar visualmente el flujo de paquetes en una topología de red utilizando tecnologías de realidad extendida.

A lo largo del desarrollo del proyecto se han implementado los mecanismos necesarios para transformar la información procedente de los ficheros JSON iniciales en una representación tridimensional interactiva.

Los resultados finales demuestran que la aplicación genera escenarios XR a partir de topologías cualesquiera, reproduce capturas y despliega funcionalidades de interacción para facilitar el análisis de la simulación.

Además del objetivo principal, también se han alcanzado los objetivos instrumentales que se definen al incio de la memoria. Que se centran en aprendizajes adquiridos durante el trabajo, esto se expone más adelante con detalle.

Por otra parte, los objetivos secundarios, que se centran en la experiencia docente, han sido logrados satisfactoriamente; ya que la herramienta permite presentar conceptos de redes de ordenadores de una forma más intuitiva, facilitando la comprensión espacial y temporal.

Por tanto, a modo de conclusión, se puede afirmar que los se han alcanzado en su gran mayoría los objetivos planteados al inicio del proyecto han sido alcanzados.

5.2. Esfuerzo y recursos dedicados

El desarrollo del proyecto ha requerido una dedicación continuada durante varios meses.

Al tratarse de un proyecto basado en tecnologías no tan habituales dentro de los contenidos del grado, una parte importante del esfuerzo inicial estuvo dedicada a recordar y ampliar los conocimientos necesarios.

La mayor parte del tiempo dedicado durante la fase de desarrollo e implementación de la herramienta se ha basado en la programación.

En las siguientes subsecciones se presenta una estimación aproximada del esfuerzo realizado durante las distintas etapas del proyecto.

5.2.1. Distribución temporal por sprints

La planificación seguida durante el desarrollo se basa en la metodología Agile, diviendo el trabajo en varios sprints para incorporar las nuevas funcionalidades.

La Tabla 5.1 muestra una estimación muy aproximada del tiempo dedicado a cada fase.

Cuadro 5.1: Estimación de horas dedicadas por sprint

Sprint	Actividad principal	Horas
1	Investigación inicial y pruebas XR	20
2	Diseño conceptual	30
3	Gestión de paquetes, historial y almacenamiento	70
4	Interfaz de usuario	45
5	Escenario final	45
6	Integración y automatización	25
7	Validación y pruebas	15
8	Redacción de memoria	40
	Total	290

Mientras que la tabla 5.2 identifica en qué semanas se ha estado trabajando en cada sprint, contando la semana 1 como la primera de septiembre.

Cuadro 5.2: Estimación del tiempo dedicadas por sprint utilizando las semanas reales de trabajo

Sprint	Actividad principal	Semanas
1	Investigación inicial y pruebas XR	1-4
2	Diseño conceptual	4-9
3	Gestión de paquetes, historial y almacenamiento	8-17
4	Interfaz de usuario	16-22
5	Escenario final	21-27
6	Integración y automatización	25-29
7	Validación y pruebas	28-30
8	Redacción de memoria	30-44

5.2.2. Tiempo de aprendizaje

Una parte del esfuerzo estuvo relacionada con el aprendizaje de tecnologías que no habían sido abordadas en profundidad durante la titulación.

Es cierto que el desarrollo de aplicaciones XR mediante WebXR y A-Frame, junto con la programación avanzada en JavaScript orientada a entornos tridimensionales, sí formaban parte del plan de estudios del grado, sin embargo, esta aplicación lleva muchos de los conceptos aprendidos a un nivel más elevado. Por ello, ha sido necesario, no solo un tiempo de adaptación, sino también una parte de ampliar conceptos.

También fue necesario profundizar en aspectos relacionados con la comunicación entre aplicaciones web y servidores locales, el procesamiento de datos en formato JSON y la automatización de flujos de trabajo relacionados con simulaciones de red.

5.2.3. Redacción de la memoria

La elaboración de la memoria ha supuesto una fase adicional del proyecto que ha requerido un esfuerzo significativo. Desde la perspectiva concreta de este trabajo, ha sido la parte más dura, ya que la fase de programación y desarrollo de la herramienta fueron más llevaderas, sin embargo, se ha hecho más cuesta arriba la parte de redactar, ya que en el grado no se acostumbra realizar redacciones de este calibre.

Además de documentar las decisiones de diseño adoptadas, ha sido un reto estructurar adecuadamente toda la información recopilada durante estos meses, donde después de cada reunión se sacaron unos apuntes que han sido de gran utilidad para el capítulo del desarrollo.

El uso de LaTeX ha permitido generar un documento con formato académico profesional, aunque también ha requerido tiempo adicional para familiarizarse con el formato que se usa.

5.2.4. Estimación global

Considerando todas las actividades realizadas durante el proyecto, incluyendo aprendizaje, desarrollo, reuniones de seguimiento y redacción de la memoria, se estima una dedicación aproximada de 300 horas distribuidas a lo largo de 10 meses.

Esta dedicación viene marcada por el resto de responsabilidades que se intercalaban con el proyecto, como la preparación de algunas asignaturas pendientes o la realización de prácticas académicas.

5.3. Lecciones aprendidas y problemas resueltos

5.3.1. Caraácter multidisciplinar

Uno de los grandes retos de este trabajo es que ha requerido combinar conocimientos de distintas áreas, como los propios de redes, desarrollo web, representación tridimensional y realidad extendida. Esto ha permitido también obtener la visión global de un proceso completo de creación de aplicaciones tecnológicas.

5.3.2. Diseño previo de la arquitectura

Una de las principales lecciones aprendidas ha sido la importancia de escoger una arquitectura y diseño adecuados antes de comenzar la implementación. Aunque varias veces fue necesario replantear soluciones debido a limitaciones detectadas, las primeras ideas ya estaban muy enfocadas en una escalabilidad alta.

5.3.3. Aprendizaje de tecnologías XR

EL principal reto del desarrollo de aplicaciones XR es adaptar la aplicación para una correcta disposición espacial de elementos, la interacción tridimensional o la comodidad visual del usuario, ya que no se corría el riesgo de hacer algo que el usuario no entendiera.

Para esto ha sido necesario profundizar en el funcionamiento de WebXR, entendiendo los sistemas de referencia, controladores e interacción espacial. Además, se ha profundizado la arquitectura interna de A-Frame y de Three.js, entendiendo cómo se construyen las escenas y se transforman las entidades tridimensionales

5.3.4. Representación espacial de topologías de red y sincronización temporal de paquetes

Otro de los problemas técnicos ha sido la representación de la variable temporal, ya que había que mostrar y hacer entender la secuencia temporal de los paquetes.

La resolución, usando paneles y huellas, es uno de los puntos innovadores de este proyecto; no es solo un apaño, es realmente una solución definitiva pensando en modificaciones del trabajo a futuro.

5.3.5. Diseño iterativo y resolución de problemas

A lo largo del proyecto se ha destacado la importancia de tener un enfoque iterativo, ya que muchas de las decisiones iniciales se iban revisando y aparecían nuevas necesidades no calculadas originalmente.

Esto ha permitido entender que el desarrollo de aplicaciones no sigue un camino lineal y que es necesario tener la capacidad para adaptarse a los cambios..

5.3.6. Uso de JavaScript

El proyecto ha permitido adquirir un conocimiento mucho más avanzado de JavaScript del que habitualmente se utiliza en aplicaciones web. Entre ellos:

- la programación basada en componentes.
- La gestión de eventos personalizados

- El uso intensivo del DOM para generar interfaces dinámicas.

También se ha profundizado en el modelo de componentes de A-Frame, aprendiendo a separar funcionalidades independientes y a comunicarlas entre si.

5.3.7. Conceptos de ingeniería del software

Finalmente, el desarrollo ha servido para profundizar en conceptos de ingeniería del software. Ya que ha sido necesario diseñar una arquitectura escalable, optimizar el rendimiento y realizar una validación continua y refinamiento hasta alcanzar una herramienta funcional y estable.

5.4. Relación con los estudios cursados

El desarrollo de este Trabajo de Fin de Grado ha permitido utilizar una buena cantidad de conocimientos adquiridos durante el grado; de la misma manera, ha permitido identificar áreas tecnológicas en las cuales el grado es más limitado.

5.4.1. Conocimientos aplicados

Gran parte de la base necesaria para este trabajo viene de asignaturas cursadas durante el grado. Aplicando conjuntamente los conocimientos adquiridos a lo largo de varias asignaturas del grado, integrando competencias de redes de computadores, programación y gráficos por computador.

Por un lado, las asignaturas relacionadas con redes de ordenadores, como Arquitectura de Internet, y la asignatura de Sistemas Telemáticos para Medios Audiovisuales, son uno de los pilares del proyecto, con conocimientos clave sobre protocolos de comunicación, direccionamiento IP o análisis de tráfico. Es clave para entender cómo necesita visualizar el escenario el usuario para entender la captura de tráfico.

Por otro lado, las asignaturas de programación han sido la otra base para implementar los componentes que se mencionan en capítulos anteriores, aprendida principalmente en las dos asignaturas de infórmatica que tiene el grado. Destacan los conocimientos sobre estructuras de datos, programación orientada a objetos y depuración, organizando el sistema en componentes

independientes que se relacionan mediante eventos. Del mismo modo, la experiencia previa en JavaScript ha facilitado la implementación de la lógica de la simulación y la gestión dinámica de entidades A-Frame.

Aparte de esto, la utilización de dispositivos Meta Quest, usados previamente en las aulas, ha sido de una gran ayuda durante el desarrollo, entre otras cosas para la validación del sistema, esto, junto al uso más avanzado de A-Frame, Three.js y JavaScript, se han aprendido en la asignatura de Gráficos y Visualización en 3D, muy recomendable.

Por último, durante el proyecto se han aplicado competencias generales del grado, como la planificación de un proyecto software, la resolución de problemasy el control de versiones mediante Git.

5.5. Planificación, presupuesto y seguimiento

Aunque el proyecto se ha desarrollado siendo un Trabajo de Fin de Grado, se ha intentado mantener una metodología de trabajo parecida a las de proyectos reales.

Así, se ha establecido una planificación basada en objetivos organizados mediante sprints (Agile). Algunas tareas han tenido variaciones respecto a lo que se estimó al arrancarlas; a pesar de esto, la gran ventaja de la organización por sprints es que permite dividir un proyecto complejo en partes más asumibles y sencillas.

5.5.1. Diagrama de Gantt

La Figura 5.1 muestra una representación temporal aproximada de las distintas fases del proyecto.

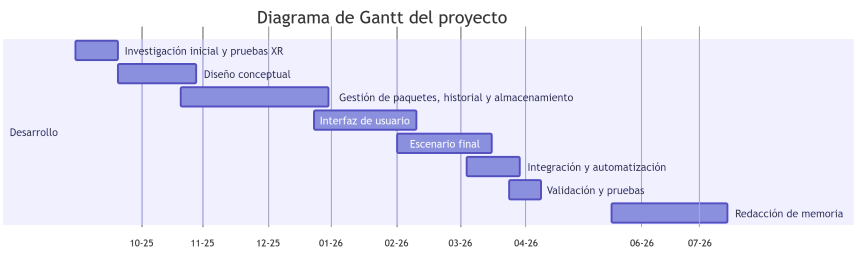


Figura 5.1: Diagrama de Gantt del desarrollo del proyecto.

Viendo este diagrama se puede apreciar claramente cómo la organización por sprints permite el desarrollo progresivo de la aplicación pero siguiendo un orden y sin pisar tareas.

Visto tras finalizar la parte de desarrollo, la utilización de esta metodología iterativa ha resultado adecuada para este proyecto,por su componente experimental, ya que algunas decisiones se han tenido que revisar conforme aumentaba el conocimiento de las herramientas utilizadas.

5.5.2. Recursos utilizados

La siguiente lista enumera los recursos utilizados para llevar a cabo el proyecto:

- Meta Quest
- Equipo de desarrollo
- Conexión y servicios de red
- Software de desarrollo
- NetGUI-Kathará
- GitHub
- VSCodium
- LaTeX

Destaca la ausencia de costes directos asociados a software, los cuales, o son propios del grado, como NetGUI, o son Open Source; por otro lado, las Meta Quest 3 son parte de los recursos que la universidad pone a disposición del alumno.

Además de los recursos materiales, el principal coste asociado al proyecto corresponde al tiempo dedicado al desarrollo.

5.6. Trabajos futuros

Aunque la aplicación cumple los objetivos planteados y se puede hablar de ella como una herramienta funcional, existen muchas líneas de mejora para ampliar sus capacidades y su utilidad pensando principalmente en entornos docentes.

Una de las principales extensiones es incorporar tráfico en tiempo real. Es una línea sobre la que se ha debatido durante el desarrollo y que, por su dificultad, se ha decidido no seguir, y usar tráfico en tiempo discreto; sin embargo, la posibilidad de representar los paquetes según son capturados es un salto muy grande.

También es muy interesante la idea de que el usuario pueda realizar filtrados y búsquedas, permitiendo seleccionar únicamente determinados protocolos, direcciones IP, ... Pensando sobre todo en capturas de gran tamaño.

Desde el punto de vista de la experiencia del usuario en XR, existen múltiples opciones de interacción; entre estas, se incluyen selección mediante gestos o comandos de voz, que harían una aplicación mucho más adaptada a la actualidad tecnológica.

Otra línea de desarrollo interesante es la posibilidad de hacer escenarios multiusuario; con lo que varios estudiantes visualizan simultáneamente la misma topología desde diferentes dispositivos XR.

Finalmente, como no puede ser de otra manera, una posible evolución a largo plazo es integrar análisis mediante inteligencia artificial integrada; esto ayuda a detectar anomalías y a poder hacer prompts de cada escenario concreto representado.

5.7. Declaración de uso de Inteligencia Artificial

- **Herramientas utilizadas:** Se ha empleado la herramienta *ChatGPT-4o de OpenAI*.
- **Alcance de la aplicación:** El uso de dichas herramientas se ha limitado a las siguientes tareas:

    1. Generación de bloques de código concretos e ideas para la lógica del mismo.
    2. Explicaciones detalladas de fragmentos de código.
    3. Generación de scripts auxiliares como pruebas de carga.
    4. Generación de bloques de texto con ideas preliminares para la redacción de capítulos.
    5. Corrección gramatical y ortográfica.

- **Responsabilidad y Autoría:** El autor manifiesta que la IA ha actuado únicamente como un asistente de apoyo. Toda la argumentación, el análisis crítico, el diseño experimental y las conclusiones finales son de elaboración propia y original.

- **Verificación de contenido:** Se garantiza que toda la información, datos y referencias bibliográficas generadas o sugeridas por las herramientas de IA han sido verificadas manualmente mediante fuentes académicas primarias para evitar errores o falsedades.

Apéndice A

Manual de usuario

Este apéndice describe el funcionamiento de la herramienta desde el punto de vista de un usuario que no conoce la aplicación, pero que posee conocimientos básicos sobre redes de computadores. Desarrolla paso a paso cómo se maneja la aplicación, para entender a fondo cada característica y funcionalidad del proyecto.

A.0.1. Descargar el repositorio y abrirlo en un IDE

El primer paso consiste en obtener los ficheros sobre los que se ejecuta la aplicación, para ello, se debe descargar el repositorio desde el siguiente enlace: https://github.com/mcerdeno2021/SIMULADOR-REDES-ORDENADORES-XR.

Para ello usaremos el botón de descarga A.1, al que se llega clickando en code y en *Download ZIP*, con el que obtendremos un *.zip* que, una vez extraído en una carpeta, desplegará todo el contenido del repositorio.

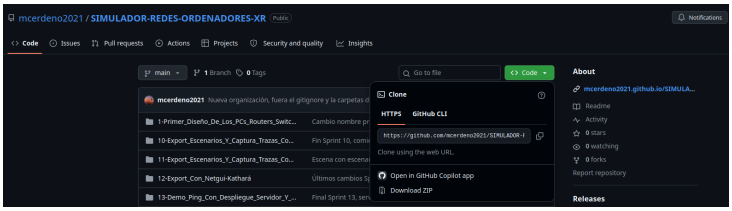


Figura A.1: Botón para descargar el repositorio

Tras ello, se puede observar que todas las actualizaciones se organizan en carpetas por lo que lo idóneo es usar la última versión.

A.0.2. Inicio de la aplicación

Lo primero es generar los ficheros. Estos ficheros JSON, como se detalla en el capítulo 3, tienen que seguir una estructura concreta A.2, el usuario ejecuta la aplicación.

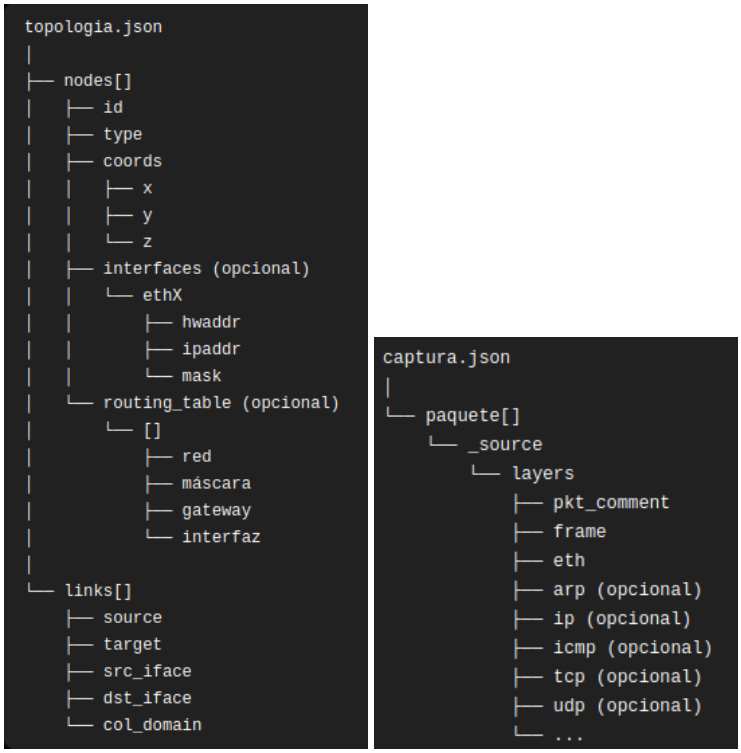


Figura A.2: Estructura del fichero JSON de captura

En el repo, los ficheron JSON de ejemplo se llaman *machineNames.json* y *merged_capture.json*, que son el fichero de topología y el de la captura, respectivamente.

Después, en el programa *historia.js*, donde se cargan los ficheros, hay que indicar en qué ruta se encuentran. Ahora mismo se procesan dos ficheros que van incluidos en el repositorio, basta con sustituirlos por los que interese reproducir.

```
22    fetch(`../machineNames.json`)
23      .then(r => r.json())
24      .then(datos => {
```

Figura A.3: Ubicación de los ficheros JSON de entrada en *historia*

Para iniciar la aplicación la opción principal es:

- Desplegar manualmente un servidor que sirva el index de la aplicación o instalar alguna extensión que despliegue un servidor, como lo hace *LiveServer.index.html*, dentro de la carpeta *aframe* que contiene la aplicación A-Frame, y se queda escuchando en las rutas correspondientes para cargar los ficheros JSON, tanto con los datos de la topología como de la información temporal asociada a los paquetes capturados.

A.0.3. Generación automática del escenario

Al acceder al escenario aparece representada la topología completa de la red A.4. Todos los dispositivos se encuentran ya posicionados automáticamente según la información obtenida desde el fichero de topología, mientras que las conexiones entre ellos aparecen dibujadas mediante enlaces tridimensionales.

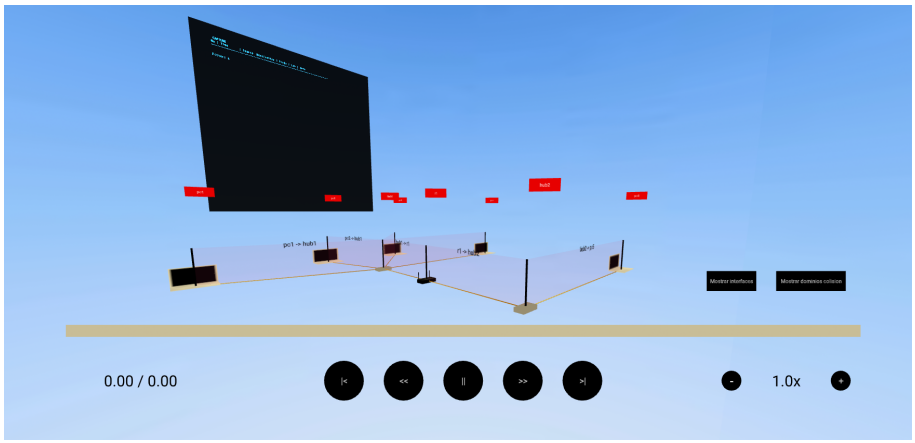


Figura A.4: Inicio de la simulación

Es entonces cuando comienza la simulación desde el instante inicial de la captura. Además de los dispositivos de red, también se muestran los paneles temporales asociados a cada conexión, que posteriormente irán almacenando la información correspondiente al paso de los paquetes.

A.0.4. Interacción con la simulación

En este punto, se explican las interacciones posibles del usuario con el escenario. Hay que tener en cuenta que la simulación se puede contemplar desde un ordenador o desde un dispositivo XR, por tanto, los siguientes puntos tratan de diferenciar ambos casos.

Movimiento por el escenario

La aplicación permite desplazarse libremente por el escenario para observar la simulación desde cualquier punto de vista.

Cuando la aplicación se ejecuta desde un navegador convencional, el movimiento se realiza utilizando los controles habituales empleados en aplicaciones tridimensionales:

- Las teclas *W*, *A*, *S* y *D* permiten avanzar, retroceder y desplazarse lateralmente.
- El movimiento del ratón modifica la orientación de la cámara, permitiendo observar el escenario en cualquier dirección, ya que se puede rotar esta clickando y, sin soltar, arrastrando.

Cuando la aplicación se ejecuta mediante unas gafas Meta Quest 3, la orientación de la cámara queda controlada automáticamente por los movimientos de la cabeza del usuario, proporcionando una percepción espacial mucho más natural. Se puede, además, usar el raycasting para apuntar y clickar en elementos de la escena gracias a los mandos.

En ambos casos el usuario puede acercarse a cualquier dispositivo de la red para observar con mayor detalle la evolución del tráfico o alejarse para obtener una visión global del escenario.

Panel de control temporal

La parte más importante de la interacción con la aplicación corresponde al control temporal de la simulación A.5. Es el propio usuario quien decide cómo desea recorrer la captura de paquetes.

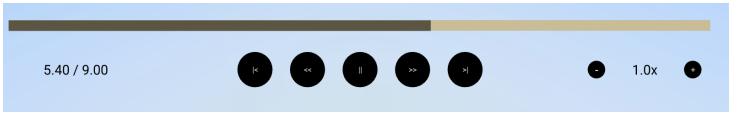


Figura A.5: Panel de control temporal

El reproductor incorpora diferentes controles que permiten modificar el estado temporal del sistema:

- Reproducir y pausar: El botón de reproducción inicia el avance del reloj interno de la aplicación.

En cualquier instante el usuario puede pausar la simulación con el botón de pausa. Al hacerlo, tanto los paquetes como todas las animaciones quedan completamente congelados, permitiendo analizar con detalle el estado de la red en ese instante concreto.

Esta funcionalidad resulta especialmente útil pensando en prácticas docentes, ya que permite detener una comunicación concreta para explicar el comportamiento de un protocolo o pararse a analizar el recorrido de un determinado paquete.

- Avance y retroceso temporal: Además de reproducir la simulación de forma continua, el sistema permite modificar manualmente el sentido temporal representado.

  El usuario puede avanzar o retroceder la simulación para volver a observar una comunicación determinada o desplazarse rápidamente hasta una zona concreta de la captura.

  Cuando el tiempo cambia, la aplicación reconstruye automáticamente el estado completo de la simulación para ese instante. Esto implica que los paquetes visibles, sus posiciones y las huellas temporales se actualizan de forma automática sin necesidad de volver a cargar la captura. No es necesario pausar la escena para cambiar el sentido de la reproducción.

  Desde el punto de vista del usuario, esta transición resulta completamente transparente.

- Ir al inicio e ir al final: El reproductor también incorpora accesos directos para situarse inmediatamente en el comienzo o en el final de la simulación.

  Al volver al inicio desaparecen todos los paquetes representados y únicamente permanece visible la topología de red. Los paneles vuelven a su altura original y sus huellas se borran.

  Por el contrario, al desplazarse al final de la captura el usuario puede observar todas las huellas temporales generadas durante la ejecución, obteniendo una visión global con todo el flujo de paquetes ya finalizado.

- Velocidad de reproducción: La velocidad de reproducción puede modificarse para adaptar la simulación a las necesidades del análisis. Esta función se representa con dos botones a cada lado, para aumentar o disminuir la velocidad y, entre ambos, el número que indica la velocidad actual A.6; este valor aumenta o disminuye en saltos de 0.1, siendo 0.1 su límite inferior y 5 el superior.

  Una velocidad reducida facilita observar el recorrido seguido por cada paquete, mientras

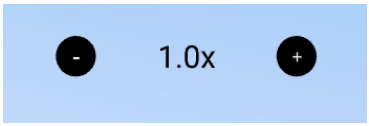


Figura A.6: Control de la velocidad de reproducción

que velocidades superiores permiten recorrer rápidamente capturas largas sin necesidad de esperar a que transcurra todo el tiempo real de la simulación.

Internamente, esta modificación únicamente afecta al avance del reloj de la aplicación, cambiando la escala temporal de los ticks, manteniéndose siempre el orden cronológico de los eventos registrados durante la captura.

Paneles temporales

Cada enlace de la red dispone de un panel asociado encargado de representar la dimensión temporal de la comunicación. Estos paneles constituyen uno de los elementos más característicos de la aplicación.

Durante la reproducción, estos paneles van aumentando progresivamente su altura conforme avanza el tiempo de la simulación, haciendo así, que las huellas más altas sean las que ocurrieron hace más tiempo, mientras que las nuevas huellas aparecen junto al cable de la conexión, a la altura del suelo, donde se desplaza el paquete.

Acercándose a un panel y pulsando sobre él, desaparecerán el resto de elementos de la escena, dejando tan solo los dos elementos que forman la conexión, junto a su cable y el propio panel, aislando así este para poder visualizarlo con claridad. Los paquetes de esta conexión seguirán mostrándose A.7.

Gracias a esta representación, el usuario no solamente observa dónde se encuentra un paquete en un instante determinado, sino que también puede reconstruir visualmente todo el historial de comunicaciones ocurrido anteriormente.

Visualización de interfaces

La aplicación permite visualizar las interfaces de red asociadas a cada dispositivo de la topología. Esta funcionalidad facilita comprender cómo se establecen físicamente las conexiones entre los distintos equipos y qué interfaz participa en cada comunicación. Se puede activar con

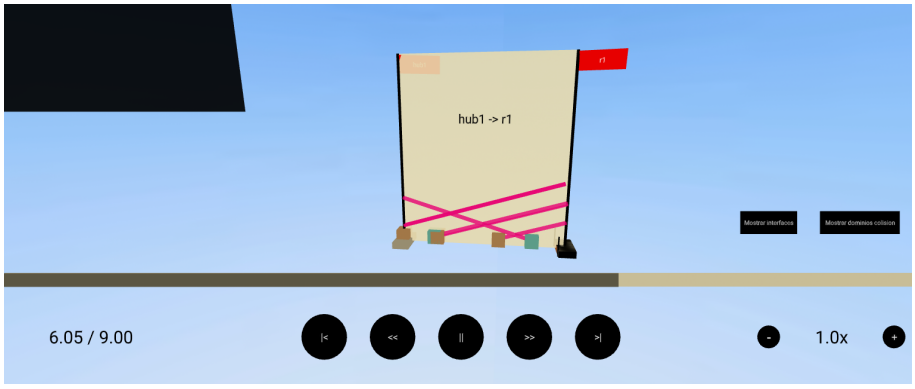


Figura A.7: Panel aislado

un botón que se encuentra a la derecha en la cámara, justo encima del reproductor temporal A.8.

Figura A.8: Botón para mostrar las interfaces y los dominios de colisión

Cuando el usuario activa esta opción, sobre cada extremo de una conexión aparece la identificación de la interfaz, utilizando la nomenclatura habitual en redes, como por ejemplo *eth0* o *eth1*, en función de la información disponible en la simulación.

Esta visualización puede activarse o desactivarse pulsando el botón de *mostrar interfaces* A.9 en cualquier momento sin afectar al resto de elementos mostrados en el escenario.

Visualización de dominios de colisión

Al lado del botón de interfaces, la aplicación incorpora un modo de representación de dominios de colisión A.8.

Al activar esta funcionalidad, las conexiones pertenecientes a un mismo dominio aparecen agrupadas mediante una superficies circular de color A.9. Esta representación permite observar cómo cambia la organización de los dominios al modificar la topología.

También en este caso el usuario puede activar o desactivar esta representación en cualquier instante sin afectar al resto de elementos.

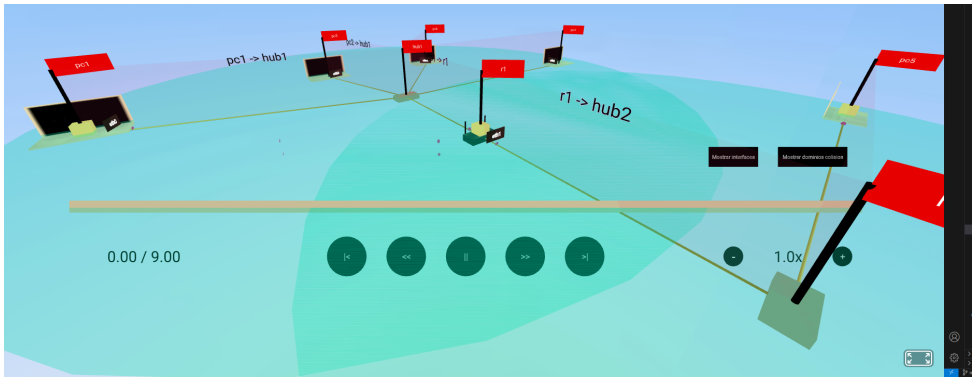


Figura A.9: Escenario con los dominios de colisión y las interfaces activadas

Panel de inspección de paquetes

Además de la representación tridimensional de los paquetes, la aplicación incorpora un panel de información inspirado en la herramienta de análisis de tráfico *Wireshark*.

El usuario verá una fila por cada paquete con datos como el instante temporal de aparición, el origen y destino de la comunicación, el protocolo utilizado, la dirección IP de ambos extremos, los puertos implicados cuando existen y el resto de información obtenida durante el proceso de captura.

Para desplegar un paquete y obtener más información, el usuario puede pulsar en cualquiera de las filas, permitiéndole así ver más información A.10.

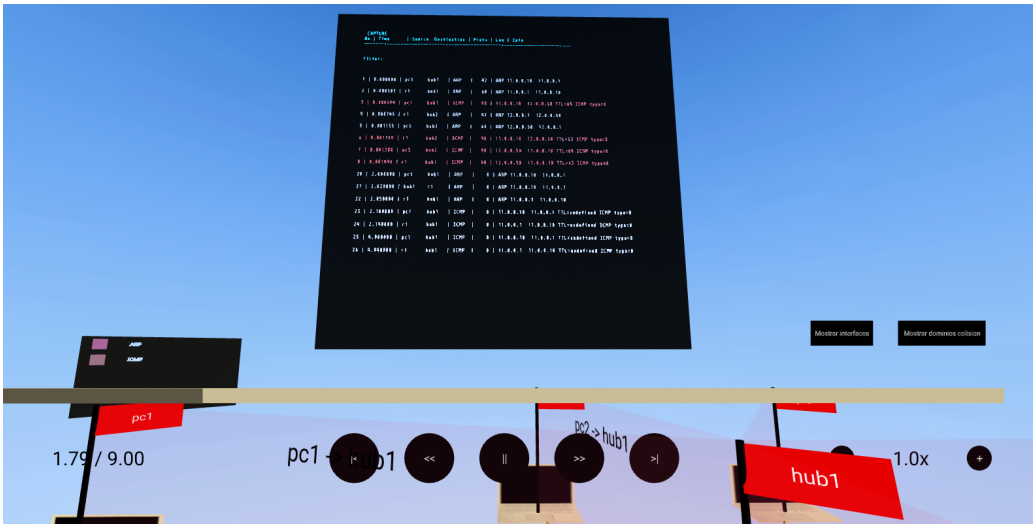


Figura A.10: Panel de inspección de paquetes desplegado

Este panel permite complementar la representación visual con información técnica detalla-

da, evitando que el usuario tenga que abandonar el entorno XR para consultar herramientas externas.

Cambio de capturas con la misma topología

La aplicación permite cargar diferentes escenarios sin necesidad de modificar el código.

Basta con sustituir el archivo JSON correspondiente a la captura de tráfico para generar automáticamente un nuevo escenario con la misma topología. Para ello, habrá que ir a la ruta en la que el servidor HTTP está escuchando el JSON de la captura JSON A.11, y depositarlo ahí.

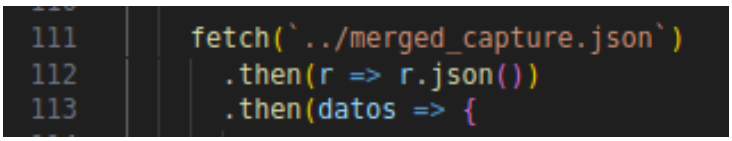


Figura A.11: Ruta en la que la aplicación espera la captura

A.0.5. Utilización mediante XR

Finalmente, la aplicación puede visualizarse utilizando dispositivos XR como Meta Quest 3. Para este caso, toda la interacción descrita anteriormente se mantiene, aunque adaptada a un entorno inmersivo.

Aunque la aplicación también puede utilizarse desde un navegador, la experiencia inmersiva es el principal objetivo del proyecto y el entorno para el que se ha diseñado la interfaz.

Esto permite al usuario desplazarse por la topología y observar los paquetes desde diferentes perspectivas.

Apéndice B

Uso de la herramienta con Netgui-Kathará

En este apéndice se explica cómo se puede utilizar la herramienta Netgui-Kathará para crear los ficheros JSON que use la aplicación.

Es clave destacar que para los alumnos es mucho más cómodo partir de una herramienta ya conocida como NetGUI, aporta un punto extra pensando en el uso de la aplicación para la docencia.

B.0.1. Descargar e instalar Netgui-Kathará

Lo primero que hay que aclarar es que los comandos que se van a utilizar están pensados para realizarse sobre un sistema operativo Linux.

Para comenzar, se debe ir al siguiente enlace: https://gitlab.eif.urjc.es/rura/netgui-kathara/-/tree/main. Desde ahí, se han de seguir los siguientes pasos, proporcionados para este trabajo por una de las desarrolladoras de la herramienta, Eva María Castro:

- PASO 1: Instalar docker. Ejecutar los siguientes comandos, paso a paso, en una ventana de terminal:

Listing B.1: Instalación de Docker en Ubuntu

```
sudo apt update

sudo apt install ca-certificates curl
```

```
sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/
   gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/ubuntu
Suites: $(. /etc/os-release && echo "${UBUNTU_CODENAME:-
   $VERSION_CODENAME}")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt update

sudo apt install docker-ce docker-ce-cli containerd.io
   docker-buildx-plugin docker-compose-plugin
```

- PASO 2: Instalar Kathará. Ejecutar los siguientes comandos, paso a paso, en una ventana de terminal:

Listing B.2: Instalación de Kathará

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-
   keys 21805A48E6CBBA6B991ABE76646193862B759810

sudo add-apt-repository ppa:katharaframework/kathara

sudo apt update

sudo apt install kathara
```

- PASO 3: Instalar NetGUI-kathará: Descarga el paquete de NetGUI-kathará y guardarlo en la carpeta de Descargas Ejecuta en una ventana de terminal:

Listing B.3: Instalación de Netgui-kathará

```
sudo apt install ~/Descargas/netgui-kathara-1.0.5.deb
```

- PASO 4: Salir de la sesión y volver a entrar.

- Extras: Para que las capturas queden anotadas en el formato que usa la aplicación, hay que instalar el siguiente paquete: https://gitlab.com/eva.castro/netgui-kathara/-/blob/main/deb_pkg/annotatePcap-1.0.0.deb?ref_type=heads

  Para que se muestren los botones de captura de tráfico se debe modificar el fichero /.config/netgui.conf para que contenga la siguiente linea: withWireXRk=true

B.0.2. Diseño de la topología

El proceso comienza en NetGUI-Kathará, al cual se accede desde este icono de aplicación B.1

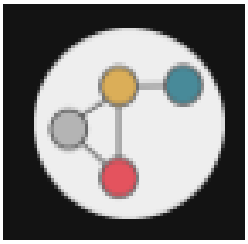


Figura B.1: Icono NetGUI

Ahí, se llega a la interfaz clásica de Netgui B.2, desde la cual el usuario diseña una topología de red utilizando los elementos habituales disponibles en la herramienta, tales como ordenadores, routers, switches, hubs y cables, que se pueden añadir pulsando un botón. La topología puede ser tan sencilla o compleja como se desee, dependiendo del escenario que se quiera estudiar. Durante esta fase no existe ninguna diferencia respecto al flujo de trabajo habitual utilizado en las prácticas de redes.

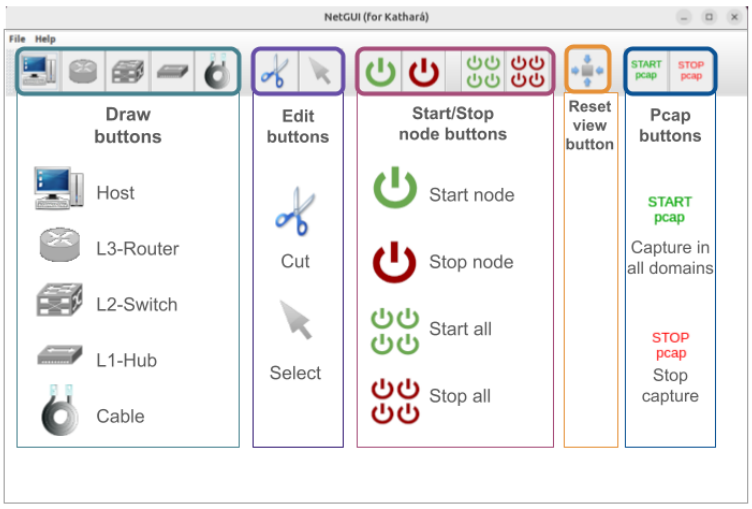


Figura B.2: Botones de la interfaz de NetGUI

B.0.3. Generación del tráfico

Una vez creada la topología, se encienden las máquinas con el botón de *start all* con lo que se abrirá un terminal diferente para cada elemento.

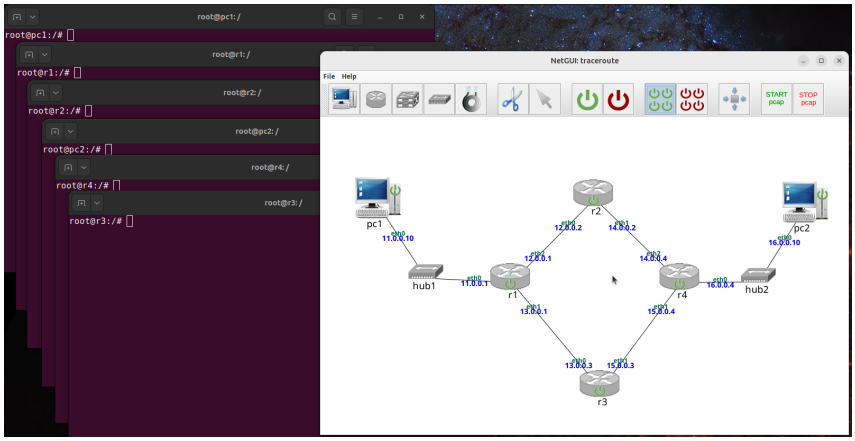


Figura B.3: Escenario con las máquinas arrancadas

Con esto abierto, se puede comenzar a ejecutar las simulaciones necesarias para generar tráfico entre los distintos dispositivos. Para ello, lo primero es iniciar la captura, con el botón de start pcap B.4, una vez lanzados los comandos requeridos, se detendrá la captura, lo que generará en la carpeta *shared* del escenario, el JSON de esta.

Durante la ejecución pueden realizarse pruebas de conectividad, transferencias de datos,

Figura B.4: Botones de arrancar/detener la captura

intercambios de mensajes ICMP o cualquier otro escenario que permita generar paquetes dentro de la red.

El objetivo de esta fase es obtener una captura representativa del comportamiento que posteriormente será visualizado dentro del entorno XR.

B.0.4. Exportación de los datos

Tras finalizar la simulación, NetGUI-Kathará genera los ficheros necesarios para describir tanto la topología como las capturas realizadas.

Estos datos son exportados en formato JSON, lo que permite que puedan ser interpretados automáticamente por la aplicación desarrollada.

En este punto el usuario dispone de toda la información necesaria para reproducir la simulación dentro del entorno XR.

Bibliografía

[1] A-Frame. A-frame documentation. https://aframe.io/docs/, 2025. Último acceso: julio de 2026.

[2] ECMA. The json data interchange syntax. https://www.ecma-international.org/publications-and-standards/standards/ecma-404/, 2017.

[3] GitHub, Inc. Github documentation. https://docs.github.com/, 2025. Consultado: 2026-07-11.

[4] Khronos Group. Webgl specification. https://www.khronos.org/webgl/, 2025. Último acceso: julio de 2026.

[5] L. Lamport. LaTeX: A document preparation system, 1994.

[6] Meta Platforms, Inc. Meta quest 3 documentation. https://developers.meta.com/horizon/documentation/, 2025. Consultado: 2026-07-11.

[7] Mozilla Foundation. Javascript guide. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide, 2025. Último acceso: julio de 2026.

[8] NetGUI. Netgui-kathará repo. https://gitlab.eif.urjc.es/rura/netgui-kathara/, 2025. Último acceso: mayo de 2026.

[9] Python Software Foundation. Python documentation. https://docs.python.org/3/, 2025. Consultado: 2026-07-11.

[10] Three.js. Three.js documentation. https://threejs.org/docs/, 2025. Último acceso: julio de 2026.

[11] VSCodium Project. Vscodium documentation. https://vscodium.com/, 2025. Consultado: 2026-07-11.

[12] W3C. Webxr device api. https://www.w3.org/TR/webxr/, 2025. Último acceso: julio de 2026.

[13] WHATWG. Dom living standard. https://dom.spec.whatwg.org/, 2025. Consultado: 2026-07-11.

[14] WHATWG. Html living standard. https://html.spec.whatwg.org/, 2025. Consultado: 2026-07-11.

[15] Wireshark Foundation. Wireshark user's guide. https://www.wireshark.org/docs/wsug_html_chunked/, 2025. Último acceso: julio de 2026.