UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SIMULADOR GRÁFICO DE PROTOCOLOS DE TRANSPORTE PARA EL APOYO A LA DOCENCIA EN REDES COMPUTACIONALES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

JOSÉ IGNACIO VERGARA PALOMINO

PROFESOR GUÍA:
JOSÉ PIQUER GARDNER

MIEMBROS DE LA COMISIÓN:
NANCY HITSCHFELD KAHLER
ALEJANDRO HEVIA ANGULO
RODRIGO ARENAS ANDRADE

SANTIAGO DE CHILE
2021

Resumen

Los protocolos de repetición automática de peticiones, son un conjunto de métodos que permiten la transmisión confiable de datos sobre canales de comunicación deficientes, a través del uso de confirmaciones y tiempos de espera. Estos se estudian en el curso "Redes" (CC4303) de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, como parte de la teoría detrás de los protocolos de la capa de transporte. En este curso, para mostrar el funcionamiento de estos métodos, se utiliza una aplicación web que permite configurar e interactuar con una simulación y su visualización, donde la simulación consiste en el intercambio de paquetes de datos entre un emisor y un receptor.

La aplicación mencionada posee diversos problemas. Su interfaz gráfica no se adapta a dispositivos móviles, y la ejecución de la simulación sufre de inconsistencias relacionadas a la inactividad de la pestaña en la que se ejecuta. Además, existen problemas con la distribución, orden, documentación y mantenibilidad general del código. Esta memoria pretende rediseñar, reimplementar, y reestructurar completamente el simulador de protocolos, abordando desde un inicio los problemas mencionados, que afectan a las experiencias de usuario y desarrollo.

Para llevar a cabo la solución se utilizan tecnologías web como HTML, CSS y JavaScript; herramientas y servicios de versionamiento y gestión de proyectos como Git y GitHub; *frameworks* y librerías para la implementación de pruebas unitarias como Mocha y Chai; JSDoc, una herramienta para la generación de documentación; y Chart.js, una librería que permite generar gráficos. Destaca también, el uso de tecnologías como *Web Components*, que permite la implementación de componentes reutilizables que encapsulan lógica y estilo gráfico; SVG, un lenguaje para la descripción de gráficos vectoriales; y la API de animaciones web, interfaz de JavaScript provista por el navegador para la programación de animaciones.

En este trabajo, se crea una librería en JavaScript que permite la ejecución de simulaciones para los protocolos mencionados. Para implementar dicha librería, se utiliza desarrollo guiado por pruebas, programación basada en eventos, y programación orientada a objetos, empleando herencia de clases y el patrón de diseño *plantilla* (*template pattern* en inglés). Por otra parte, para la implementación de la interfaz gráfica se utiliza programación basada en eventos, junto a las tecnologías mencionadas anteriormente.

El resultado de esta memoria es una aplicación web[1], que permite configurar la simulación, e interactuar con su visualización. Esta, posee una interfaz gráfica que otorga una mejor experiencia de usuario, y es adaptable a distintos tamaños de pantalla. Y por otra parte, el proyecto[2] carece de los problemas mencionados inicialmente, proporcionando una implementación modular, testeada, documentada, y por lo tanto mejor y más mantenible.

[1]Código disponible en: `https://joseivp.github.io/transport-protocols-simulator/`

[2]Repositorio del proyecto: `https://joseivp.github.io/transport-protocols-simulator/`

Tabla de Contenido

Capítulo 1

Introducción

1.1. Contexto

En redes computacionales, los protocolos de transporte son los encargados de transmitir o comunicar la información entre procesos que pueden estar en máquinas diferentes. Estos protocolos implementan reglas para manejar los paquetes de información intercambiados. Y gracias a dichos conjuntos de reglas, los protocolos de transporte pueden proveer abstracciones para la comunicación confiable entre procesos.

Un ejemplo de estos protocolos, y una parte muy importante de cómo funciona la Internet en estos días, es TCP (Transport Control Protocol, por sus siglas en inglés). TCP permite una comunicación confiable, secuencial y ordenada entre procesos, sobre una red no confiable, es decir, una donde se pueden perder, dañar o alterar paquetes de datos. Si le enviamos información a un proceso a través de TCP, este la recibirá de forma ordenada (respecto al orden de envío) e inalterada. Esto es posible gracias a que TCP implementa un conjunto de reglas que permiten el control de errores, flujo, y congestión, de los paquetes circulantes en la red. De esta forma, se verifica la validez de la información recibida, se controla la cantidad de paquetes que un proceso remitente puede enviar a un proceso receptor, y la cantidad de paquetes introducidos a la red.

Dado el rol fundamental de los protocolos de transporte, es muy importante que los estudiantes de computación formen un entendimiento, al menos básico, del funcionamiento de éstos. Con este fin, en los cursos de redes computacionales se enseña sobre distintos métodos para el manejo de paquetes que se han propuesto a lo largo de la historia, como *Stop-and-wait* (SW), *Go-Back-N* (GBN), y *Selective Repeat* (SR), donde los dos últimos son más similares a TCP [5]. Y por la naturaleza asíncrona de estos métodos, es muy útil usar visualizaciones gráficas para mostrar su funcionamiento a los alumnos.

En particular, el curso "Redes" (CC4303) del Departamento de Ciencias de la Computación (DCC) de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, ha introducido en los últimos semestres un simulador gráfico que permite visualizar la ejecución de los tres métodos mencionados anteriormente. Dicho simulador, posee extensiones hechas

por el profesor del DCC José Miguel Piquer[1], ampliando la implementación del código original de Johannes Kessler[2]. El simulador permite ver el intercambio de paquetes entre un remitente y un receptor, a través de una animación que muestra los paquetes listos para enviarse, los que están en camino, aquellos que se han perdido, confirmado, etc; permitiendo al usuario configurar los parámetros de la comunicación.

1.2. Motivación

A pesar de que el simulador mencionado es una muy buena herramienta para mostrar el funcionamiento de los protocolos, todavía hay lugar para mejorarlo en términos de interfaz de usuario, usabilidad, y claridad de la información presentada. Además, el código original del simulador fue desarrollado en base a tecnologías web del año 2012, y a la fecha estas han tenido grandes mejoras que, si se usaran, aportarían a la claridad, mantenibilidad y extensibilidad del proyecto. En esta memoria se pretende reimplementar el simulador, tomando en cuenta los aspectos anteriores y, por sobre todo, el potencial pedagógico de una herramienta de este tipo, que permite concretizar las abstracciones de los conceptos estudiados.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo general de la memoria consiste en rediseñar y reimplementar el simulador gráfico de protocolos de transporte. Este trabajo apunta a mejorar la interacción de los usuarios con la aplicación y la experiencia de desarrollo de quien esté a cargo de la mantención del proyecto. Lo que se traduce en una mejor herramienta para complementar el estudio teórico de los protocolos de transporte con una experiencia más práctica del funcionamiento de estos.

1.3.2. Objetivos específicos

A continuación se especifica una lista de metas que apoyan al objetivo general del proyecto:

1. Diseñar e implementar una librería que permita simular los protocolos *SW*, *GBN* y *SR* para un emisor y un receptor.
2. Diseñar y ejecutar test unitarios para las funciones de la librería.
3. Diseñar e implementar una interfaz gráfica de usuario que consuma las abstracciones de la librería anterior y que permita al usuario correr experimentos y obtener resultados. Para esto, se tienen las siguientes sub-metas:
   (a) Reimplementar los formularios que permiten al usuario configurar la ejecución del experimento.
   (b) Reimplementar la obtención de estadísticas en tiempo real de la simulación.

[1]Simulador con extensiones (José M. Piquer): `https://users.dcc.uchile.cl/~jpiquer/srgbn4.html`
[2]Código original (Johannes Kessler): `https://www.ccs-labs.org/teaching/rn/animations/gbn_sr/`

(c) Investigar y estudiar la manera de realizar animaciones en los navegadores web. En particular, cómo lidiar con los problemas que surgen al cambiar de pestaña en los navegadores, y que ralentizan la simulación, generando errores.

(d) Reimplementar las animaciones que permiten visualizar el intercambio de paquetes entre un emisor y un receptor.

4. Generar una documentación del código que ayude al desarrollo y mantención del proyecto.

1.4. Metodología

La metodología para llevar a cabo esta memoria consiste en la separación de tareas a partir de los objetivos mencionados anteriormente y de los requisitos derivados discutidos en la sección 3.4, para luego realizar una implementación secuencial de éstas. La metodología también involucra el estudio y experimentación con distintas tecnologías que pueden ayudar a implementar la aplicación. De esta manera, se mantiene en todo momento una lista de las tareas tipo Kanban, con tareas por hacer, en proceso, y completadas.

1.5. Contenido de la memoria

En el capítulo 2 se discute sobre los protocolos para los que se pretende implementar las simulaciones. Y se analizan las tecnologías web y librerías consideradas para la realización de la aplicación que permite a un usuario ejecutar dichas simulaciones. Luego, se estudian algunas de las tecnologías disponibles para la implementación de animaciones que permitan visualizar la ejecución.

En el capítulo 2 también se discute la metodología usada en el desarrollo de la librería de protocolos. Se habla sobre la arquitectura de esta librería y de la aplicación que permite su uso. Y además, se estudia la aproximación para el diseño y desarrollo de la interfaz gráfica de usuario.

Los problemas del simulador de protocolos existente, y los desafíos y limitaciones a enfrentar en la nueva implementación son discutidos en el capítulo 3. Ahí también se analiza la relevancia de la solución y los requisitos que esta debe cumplir. Y por último, se establecen criterios sobre la calidad del software a realizar.

El capítulo 4 habla sobre el diseño de la interfaz gráfica de usuario y cómo esta es dividida en componentes que facilitan su disposición espacial e implementación. Revisa además, la arquitectura del software implementado, y los beneficios de un diseño modular, con programación basada en eventos. Y finalmente, presenta el diseño de las clases de la librería de protocolos, y el de las clases de los componentes de la interfaz gráfica.

En el capítulo 5 se habla sobre cómo la solución propuesta resuelve el problema general de crear una nueva versión del simulador de protocolos. Por otra parte, se discute cómo la solución resuelve el problema del uso de la aplicación en distintos tamaños de pantalla. Y además, se habla sobre el rendimiento de la aplicación en distintos escenarios.

Por último, el capítulo 6 presenta una breve recapitulación del trabajo realizado. Habla también sobre los objetivos alcanzados y los aprendizajes obtenidos. Y finaliza proponiendo trabajo futuro para mejorar el simulador.

Capítulo 2

Estado del arte

En este capítulo se discute sobre los protocolos para los que se pretende implementar las simulaciones. Se analizan las tecnologías web y librerías consideradas para la realización de la aplicación, que permite a un usuario ejecutar dichas simulaciones. Y además, se estudian algunas de las tecnologías disponibles para la implementación de animaciones que permitan visualizar la ejecución.

Por otra parte, se discute la metodología usada en el desarrollo de la librería de protocolos. También, se habla sobre la arquitectura de esta librería y de la aplicación que permite su uso. Y además, se estudia la aproximación para el diseño y desarrollo de la interfaz gráfica de usuario.

2.1. Protocolos de transporte

Los protocolos que se quieren simular en este proyecto pertenecen al conjunto de protocolos ARQ (Automatic Repeat Query, por sus siglas en inglés), que son métodos para el control de errores en transmisiones de datos. Estos protocolos utilizan confirmaciones y tiempos de espera para lograr un intercambio de datos confiable, es decir, libre de errores, sobre conexiones deficientes entre un emisor y un receptor. En esta clase se encuentran los protocolos *Stop-and-wait* (SW), *Go-Back-N* (GBN), y *Selective Repeat* (SR) [5].

2.1.1. Stop-and-wait

Este protocolo es el más simple de los mencionados. Tal como su nombre lo indica, consiste en parar y esperar. Cada vez que el emisor envía un paquete de datos, debe esperar la confirmación del paquete por parte del receptor para enviar el siguiente. Además, el emisor inicia un tiempo de espera por la confirmación cada vez que envía un paquete; si la confirmación no llega antes de que el tiempo acabe, entonces el emisor reenvía el paquete de datos y reinicia el tiempo de espera.

En cuanto al receptor, este espera a que le llegue el paquete con el número de secuencia correcto. Si recibe un paquete con otro número de secuencia o si el paquete está dañado,

entonces responde con una confirmación para la secuencia anterior. Por otra parte, si recibe el número de secuencia esperado, responde con una confirmación y procede a esperar el próximo número de secuencia.

2.1.2. Go-Back-N

GBN introduce el concepto de ventana deslizante, un buffer en el emisor, que le permite enviar tantos paquetes como sea el tamaño de la ventana. Aquí, el emisor también inicia un tiempo de espera cada vez que envía el primer paquete de la ventana, esperando por la confirmación de este. Si la confirmación no llega antes de que termine el tiempo, el emisor reenvía todos los paquete de la ventana que hayan sido enviados; por el contrario, si la confirmación es recibida, entonces se mueve la ventana deslizante, comenzando en el siguiente paquete que no ha sido enviado o confirmado.

Por otra parte, el receptor de GBN se comporta de manera similar al de SW, esperando recibir los paquetes en orden, por el número de secuencia de estos. Además, introduce un concepto adicional: confirmaciones acumulativas. Con éstas, cada confirmación que envíe el receptor para un paquete, también corrobora todos los anteriores.

2.1.3. Selective Repeat

El emisor de SR también utiliza una ventana deslizante, pero a diferencia de GBN, inicia y mantiene un tiempo de espera por cada paquete enviado. Si la confirmación de un paquete enviado no es recibida antes de que termine su tiempo de espera, se reenvía el paquete y se reinicia su tiempo de espera. Y al igual que en GBN, cuando el primer paquete o secuencia de la venta es confirmada, esta se mueve para comenzar en la próxima secuencia que no ha sido enviada o confirmada.

Por el lado del receptor, ahora también se usa una ventana deslizante. Esta le permite al receptor recibir tantos paquetes como sea el tamaño de la ventana, sin importar el orden de llegada por número de secuencia, mientras esta se encuentre dentro de la ventana y el paquete no esté dañado. Por otra parte, el receptor ignora los paquetes dañados y reenvía una confirmación para cada paquete duplicado. Finalmente, cuando la primera secuencia de la ventana es recibida correctamente, la venta se desplaza para comenzar en el próximo paquete o secuencia cuya confirmación no haya sido enviada.

Una variación de SR, incluye confirmaciones acumulativas, que corroboran la secuencia o paquete al que corresponden, y todos los anteriores. Estas confirmaciones son enviadas con el número de secuencia anterior a la primera secuencia de la ventana del receptor, luego de cada vez que este desplace su ventana (al recibir un paquete correcto) o reciba un paquete incorrecto.

2.2. Librerías, frameworks y herramientas de desarrollo web

En esta sección se discute sobre el software considerado para llevar a cabo la implementación del proyecto. Se da una pequeña definición de cada tecnología, y se discuten sus ventajas y desventajas. Esta revisión permite argumentar las decisiones tomadas respecto a las tecnologías finalmente escogidas.

2.2.1. Librerías y frameworks de desarrollo web

Al diseñar la solución para el problema de esta memoria, se estudian varias opciones de software que pueden ayudar a implementarla. Entre estas opciones surgen *frameworks* de desarrollo web como React[1], Vue[2] y Angular[3], y librerías que permiten graficar datos como D3[4], Chart.js[5] y Plotly[6]. Por otra parte, también se consideran nuevas tecnologías que se han incorporado a los navegadores como *Web Components*[7] y *ES Modules*[8], para usarlas en reemplazo a los *frameworks* mencionados.

React, Vue y Angular son *frameworks* escritos en JavaScript, que permiten crear aplicaciones web de página única (del inglés *single-page web applications*). Estos *frameworks* necesitan del entorno de ejecución de JavaScript Node.js, para el desarrollo de las aplicaciones. Además, actualmente son algunas de las opciones más populares para la creación de aplicaciones web.

Los *frameworks* mencionados, se basan principalmente en la posibilidad de crear componentes o *widgets* reutilizables, los cuales permiten encapsular la lógica y el estilo gráfico de las distintas partes que compongan una aplicación o interfaz. Entre las ventajas de React, Vue y Angular, se encuentran su escalabilidad, flexibilidad, y la amplia documentación existente y comunidad de desarrollo que los utilizan. Por otra parte, para hacer uso efectivo y correcto de ellos se debe aprender una cantidad considerable de conceptos, estilos de programación, y configuraciones de las herramientas que integran.

En el último tiempo, los navegadores han comenzado a implementar tecnologías que llenan parte del vacío que los *frameworks* intentan ocupar actualmente. Este conjunto de tecnologías es llamado *Web Components*, que de forma similar a los *frameworks*, permite crear componentes reutilizables que encapsulan lógica y estilo gráfico, pero de forma nativa en los navegadores. Junto a los *ES Modules* (ESM) o módulos de ECMAScript (el modelo estándar de JavaScript) - tecnología también implementada recientemente en los navegadores y que permite importar y exportar código a partir de archivos de JavaScript [4] - permiten crear aplicaciones web de manera fácil y ordenada, sin el uso de los *frameworks* mencionados.

Las tecnologías nombradas han sido implementadas por los navegadores más usados, entre

[1] `https://reactjs.org/`
[2] `https://vuejs.org/`
[3] `https://angular.io/`
[4] `https://d3js.org/`
[5] `https://www.chartjs.org/`
[6] `https://plotly.com/javascript/`
[7] `https://developer.mozilla.org/en-US/docs/Web/Web_Components`
[8] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules`

los que se encuentran Chrome, Safari, Firefox y Edge[9]. Una de las ventajas de usar *Web Components* junto a ESM en lugar de algún *framework*, es que permite al programador concentrarse en la implementación de la aplicación, en lugar de la configuración del *framework* utilizado. Por otra parte, esta alternativa carece de la escalabilidad, optimizaciones y robustez que ofrecen los *frameworks*, para la creación de grandes aplicaciones.

Otras tecnologías estudiadas para la implementación de la aplicación corresponden a librerías para la generación de gráficos a partir de datos como D3, Chart.js, y Plotly. De estas, Plotly es la más completa en términos de la cantidad de opciones y configuraciones, pero debido a esto, es también la más complicada y pesada de las tres (3.5 MB minificada, mientras que D3 y Chart.js no sobrepasan los 300 KB). Por otra parte, D3 es una librería de bajo nivel para la creación de visualizaciones en SVG y la manipulación de documentos HTML, que es también una dependencia de Plotly. Por último, Chart.js es una librería simple para la creación de visualizaciones de datos, que utiliza la tecnología *canvas*[10] de HTML 5.

2.2.2. Herramientas para la generación de pruebas unitarias y documentación

Por otra parte, en este trabajo se buscan librerías para la generación de pruebas unitarias y herramientas para la producción de documentación. Entre las primeras se encuentran las librerías de JavaScript Jest[11] y Mocha[12], mientras que entre las herramientas de documentación se encuentra JSDoc[13]. Estas librerías y herramientas se utilizan dentro del entorno de ejecución de JavaScript Node.js[14].

Jest es un *framework* de pruebas unitarias muy fácil de utilizar, compatible con otros *frameworks* de desarrollo web como Vue, React o Angular. Muy bien documentado, es utilizado por una gran comunidad de desarrolladores. Por otro lado, Mocha es un *framework* de pruebas más complejo que Jest, pero también más flexible, lo que le permite incluso ser ejecutado en el navegador. Debido a esto último, Mocha puede ser una gran alternativa en cuanto a la creación de pruebas para una librería cuyo principal entorno de ejecución será el navegador.

En cuanto a documentación, JSDoc es una de las herramientas más utilizadas para la elaboración de estos recursos en el desarrollo de aplicaciones en JavaScript. Esta herramienta permite crear un sitio web a partir de comentarios especiales en el código fuente. JSDoc requiere de mínimas configuraciones y funciona muy bien con editores de texto o entornos de desarrollo como Visual Studio Code[15], que entiende las anotaciones y provee de autocompletado de código.

[9] Según el sitio de estadísticas `https://gs.statcounter.com/`, éstos cubren alrededor de un 90 % del mercado actual.

[10] `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API`

[11] `https://jestjs.io/`

[12] `https://mochajs.org/`

[13] `https://jsdoc.app/`

[14] `https://nodejs.org/es/`

[15] `https://code.visualstudio.com/`

2.2.3. Herramientas de versionamiento y administración de proyectos

Finalmente, se buscan herramientas y software que permitan el control de versiones, el alojamiento de código fuente, distribución de contenido web, y el manejo de tareas e hitos para el proyecto. Para lo primero se utiliza Git[16], un software de control de versiones libre. Y para lo que resta se utiliza GitHub[17], un servicio gratuito de alojamiento de código fuente, que permite de forma fácil la publicación de una página web a partir de éste, e incorpora herramientas para el manejo de equipos y proyectos.

2.3. Métodos de dibujo y animación

En esta memoria también se estudian diferentes maneras de realizar visualizaciones y animaciones en dos dimensiones, a partir de tecnologías web implementadas en los navegadores. Entre las alternativas consideradas para la creación de visualizaciones se encuentran el uso de HTML[18] en conjunto con CSS[19], utilizar SVG[20] en combinación con CSS, y el uso de la API[21] de dibujo de *canvas*. Por otra parte, para la animación de dichas visualizaciones se consideran las transiciones y animaciones de CSS, la API de animaciones web[22], y el método *requestAnimationFrame()*[23] de la API del navegador.

Las visualizaciones hechas con HTML y CSS utilizan elementos del documento HTML, estilizados a través de CSS para representar las distintas partes de un dibujo. A pesar de que el estilizado y animaciones de elementos HTML están bien soportados por los navegadores más usados, HTML tiene más bien un fin semántico y no esta hecho para la creación de dibujos o visualizaciones. Otra desventaja de este método, es que es un poco más difícil mantener el posicionamiento de un elemento relativo a las coordenadas de origen del dibujo y la relación de aspecto del dibujo.

Por otra parte, las visualizaciones hechas con SVG y CSS utilizan los elementos semánticos de SVG, hechos para representar elementos en dibujos, en conjunto con la cómoda sintaxis de CSS para otorgar atributos visuales a estos elementos. SVG tiene la ventaja de que todas las posiciones de los elementos son relativas al origen de coordenadas del dibujo y es fácil mantener una relación de aspecto definida. Por otro lado, el último estándar (SVG 2) que permite utilizar CSS en conjunto con SVG no ha sido implementado completamente por todos los navegadores más usados y aún está en desarrollo.

Un tercer método de dibujo en la web corresponde al uso de la API de *canvas*. Esta interfaz permite una programación imperativa de los elementos visuales, a diferencia de la programación declarativa de los métodos anteriores. Esto tiene la desventaja de que la programación puede ser más verbosa y difícil de entender. Además, las optimizaciones para renderizar lo

[16] `https://git-scm.com/`
[17] `https://github.com/`
[18] HyperText Markup Language
[19] Cascading Style Sheets
[20] Scalable Vector Graphics
[21] Application Programming Interface
[22] `https://developer.mozilla.org/en-US/docs/Web/API/Web_Animations_API`
[23] `https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame`

que se dibuja dependen más del programador y menos del navegador.

En cuanto a las animaciones, un primer método son las transiciones de propiedades CSS. Este método requiere del uso de CSS, por lo que no puede ser usado en conjunto con la API de *canvas*. Para animar una propiedad visual de un elemento, se especifica a través de CSS cuál es esta propiedad, la duración de la animación y la función de sincronización; el navegador es entonces el encargado de detectar los cambios en el valor de la propiedad y de computar las transiciones necesarias entre el valor previo y el nuevo. Una ventaja de este método es que su programación es simple y hace uso de las optimizaciones que el navegador pueda tener, sin embargo, es difícil obtener un control más refinado de la animación.

Un método de animación más elaborado son las animaciones de CSS. Estas permiten un mejor control sobre las etapas de una animación, especificando las propiedades que el elemento animado debe tener en cada uno de los fotogramas clave. Por otro lado, controlar la pausa y reanudación de las animaciones solo se puede realizar a través de trucos.

Otro método de animación, aún más completo, es el uso de la API de animaciones web a través de JavaScript. Al igual que el método anterior, este hace uso de las propiedades CSS de un elemento, especificando los valores que deben tener en los fotogramas clave de la animación. Adicionalmente, cuenta con interfaces para el inicio, pausa, reanudación, cancelación y terminación de las animaciones. Sin embargo, esta API solo tiene una implementación funcional en algunos de los grandes navegadores, como Chrome, Firefox y Edge (versión basada Chromium).

Por último, *requestAnimationFrame()* es un método de la API de JavaScript proporcionadada por el navegador, que permite la creación de animaciones fotograma a fotograma. Este método puede ser utilizado tanto para cambiar las propiedades CSS de un elemento, como para dibujar con la API de *canvas*. Aquí las optimizaciones para la animación dependen del programador y no del navegador, lo que podría ser una desventaja.

2.4. Desarrollo guiado por pruebas

El desarrollo guiado por pruebas o TDD (sigla en inglés para Test Driven Develpment) es una metodología de desarrollo de software, en la que los requisitos de software son convertidos a casos de prueba, y estos son escritos antes de cualquier implementación. TDD se basa en dos principios básicos [1]: escribir código solo si alguna de las pruebas automáticas falla, y eliminar la duplicación de código. Lo anterior genera algunas implicancias técnicas, como: el diseño del software debe generar componentes cohesivos, y débilmente acoplados, que faciliten la ejecución de pruebas; la respuesta del ambiente de desarrollo a pequeños cambios debe ser rápida; y el código funcional debe proveer retroalimentación entre la toma de decisiones.

Los dos principios básicos mencionados también implican un orden en las tareas de desarrollo. Estas pueden ser resumidas en el "mantra" de TDD: red/green/refactor (o traducido desde el inglés rojo/verde/refactoriza). Así, rojo corresponde a escribir y ejecutar una pequeña prueba que no funcione; verde, a hacer rápidamente que la prueba funcione; y refactoriza, a eliminar la duplicación de código o implementar patrones de diseño que mejoren la legibilidad de éste.

2.5. Programación basada en eventos

La programación basada en eventos es un modelo computacional en el que el flujo de ejecución es determinado a partir de eventos generados por la acción del usuario, sensores o mensajes provenientes de otros hilos de ejecución o procesos. Esta forma de programación es la más usada para la implementación de interfaces gráficas de usuario interactivas. Una aplicación que utiliza este modelo por lo general posee un bucle principal que "escucha" los eventos y ejecuta procedimientos a través de llamadas de funciones o *callback functions* en inglés.

En "Event-based programming"[3], Faison define un evento como una condición detectable que puede desencadenar una notificación. Una notificación se define como una señal provocada por un evento, enviada a un receptor definido en tiempo de ejecución. Los eventos son la causa, mientras que las notificaciones son el efecto.

Una de las razones para usar la programación basada en eventos es reducir el acoplamiento de los componentes en un sistema. El acoplamiento en un sistema es asociado a la presencia de interdependencias entre clases y componentes. Éste introduce complejidad, haciendo que un sistema sea más difícil de entender, testear y mantener.

2.6. Diseño e implementación de interfaces gráficas de usuario

En la última década el diseño y la implementación de interfaces gráficas se han visto muy beneficiados por la gran cantidad de herramientas y tecnologías que han surgido o se han mejorado. Por otra parte, el estudio del diseño y la interacción han proporcionado mejores prácticas para lograr que una interfaz permita al usuario cumplir con su objetivo. Como consecuencia, la Web se ha vuelto más amigable para las personas, sobre todo en dispositivos móviles.

Herramientas para la generación de prototipos como Adobe XD[24], Figma[25], Sketch[26] y muchas otras permiten el rápido diseño de interfaces gráficas. Varias de estas herramientas pueden generar además código CSS para los componentes diseñados, aunque por lo general este dista de lo necesario para crear interfaces adaptables a distintos tamaños de pantalla o que cumplan con interacciones específicas. Otra característica de algunas de estas herramientas es que permiten el trabajo colaborativo en tiempo real, con varios usuarios editando el mismo diseño.

El lenguaje de estilos de la Web, CSS, también ha sido mejorado enormemente en la última década, proporcionando nuevos métodos para la disposición espacial de los elementos en una página web, como *Flexbox*[27] y *Grid*[28]. *Flexbox* es un método unidimensional para distribuir elementos en filas o columnas, mientras que *Grid* es un método bidimensional que

[24] `https://www.adobe.com/la/products/xd.html`
[25] `https://www.figma.com/`
[26] `https://www.sketch.com/`
[27] `https://developer.mozilla.org/en-US/docs/Glossary/Flexbox`
[28] `https://developer.mozilla.org/en-US/docs/Glossary/Grid`

permite disponer a los elementos en ambas, filas y columnas. Estos métodos permiten generar fácilmente más y mejores diseños que aquellos implementados antiguamente con elementos flotantes o posicionamiento absoluto.

2.7. Tecnologías escogidas

Ya que la aplicación a desarrollar es pequeña, se decide no utilizar *frameworks* de desarrollo web. En lugar de éstos se ocupan las tecnologías relacionadas a *Web Components*. Esta decisión permite incrementar la velocidad de desarrollo y experimentación con las tecnologías de animaciones, dado que la curva de aprendizaje para el uso de *Web Components* es mucho menor a la de los *frameworks*.

Respecto a las librerías para graficar datos, se opta por Charts.js, ya que es la más simple de utilizar y la menos pesada en cuanto al tamaño de sus archivos. El proyecto no requiere de gráficos tan elaborados (solo gráficos de línea) como para necesitar de las capacidades de las otras librerías. Y además, Charts.js posee suficientes configuraciones para obtener el aspecto visual deseado (por ejemplo: colores de los elementos, tipos de fuente, configuración de los ejes del gráfico, etc).

Para las pruebas unitarias se escoge Mocha en lugar de Jest, principalmente porque Mocha puede ser utilizado directamente en el navegador. Esto permite asegurar el funcionamiento correcto de la librería a desarrollar, en el entorno al que va dirigido. En cambio, Jest solo permite la ejecución de las pruebas dentro del entorno de ejecución Node.js.

En cuanto a la documentación de las clases y funciones de la librería de protocolos, se utiliza JSDoc, ya que es la herramienta más popular, es de fácil uso, y está bien documentada. Por otra parte, para el versionamiento de código se utiliza Git junto con GitHub, pues ya se tiene experiencia con éstos, y además GitHub posee una aplicación que permite organizar las tareas del proyecto.

De entre los métodos de dibujo mencionados se escoge SVG junto a CSS, ya que permiten incorporar las figuras directo en el código HTML, y la escalabilidad de los gráficos SVG facilita la implementación de una interfaz que se adapte a distintos tamaños de pantalla. Por otro lado, para las animaciones se escogen diferentes métodos, dependiendo del control requerido sobre éstas, lo que se ve en la sección 4.9.1.

Por último, para el diseño de la interfaz gráfica se utiliza Figma, ya que se puede obtener una licencia de estudiante con el correo institucional de la Universidad de Chile. Otra razón es que Figma puede ser utilizada en cualquier navegador, a diferencia de los otros programas de diseño mencionados, que solo tienen aplicaciones de escritorio.

Capítulo 3

Problema

En este capítulo se discuten los problemas del simulador de protocolos existente, y los desafíos y limitaciones a enfrentar en la nueva implementación. También se analiza la relevancia de la solución y los requisitos que debe cumplir. Y por último, se establecen criterios sobre la calidad del software a realizar.

3.1. Problemas de la implementación existente

El simulador de protocolos existente fue originalmente implementado en 2012, fecha en la que gran parte de los estándares web hoy disponibles aún estaban en desarrollo o todavía no habían sido desplegados en la mayoría de los navegadores más usados. Una de las adiciones más importantes a los navegadores fue la publicación del estándar de JavaScript ECMAScript 2015, que incluyó declaración de clases, la importación de módulos, declaración local de variables, expresiones *arrow function*, nuevos tipos de colecciones como diccionarios, conjuntos y diccionarios de referencia débil, promesas - un nuevo tipo de objeto para la programación asíncrona -, y muchas otras características [4]. Todas estas adiciones permiten generar código más legible y mantenible para la web.

En relación a lo mencionado anteriormente, el simulador existente posee una implementación monolítica en la que los componentes mantienen una fuerte dependencia entre sí. La mayoría del código esta agrupado en un único archivo, y este carece de documentación o comentarios en su mayor parte. Esto hace que la implementación de la aplicación sea difícil de entender, y la introducción de cambios puede fácilmente generar problemas.

El simulador posee además, un problema donde la simulación actúa de forma incorrecta una vez que el usuario deja la pestaña en la que se ejecuta. Esto se debe a que el simulador utiliza la función de la API del navegador *setTimeout()*[1] - que ejecuta una función dada luego de un tiempo definido - para simular el retraso de los paquetes, y los navegadores incluyen optimizaciones que hacen que los procesos y *timeouts* en aquellas pestañas que no están activas sean ralentizados. Así, cuando el usuario vuelve a la pestaña en la que se ejecuta la aplicación, el estado de la simulación es erróneo y éste debe detenerla o incluso refrescar la

[1] `https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout`

página.

Otro problema es que el diseño de la interfaz gráfica de usuario para el simulador actual[2] es poco intuitiva (ver figuras 1 y 2 del anexo), y un tanto desordenada. Esta, no aprovecha las capacidades que brinda CSS para crear una interfaz clara y atractiva visualmente. Además, la interfaz no es adaptable a tamaños de pantalla más pequeños, considerando que gran parte de los estudiantes utilizan dispositivos móviles para acceder a internet [3].

3.2. Desafíos y limitaciones para la nueva implementación

La nueva implementación del simulador debe abordar los mismos problemas anteriores, es decir, debe utilizar las nuevas sintaxis provistas por JavaScript para generar código modular, fácil de entender y mantener. Debe lidiar también con las limitaciones en el uso de *setTimeout()* y otras restricciones que los nuevos navegadores puedan presentar. Y además, debe hacer uso de las nuevas herramientas de CSS y JavaScript para crear una interfaz intuitiva y atractiva.

Otro desafío para la implementación del nuevo simulador es crear una interfaz y que se comporte de manera aceptable en distintos dispositivos, móviles o de escritorio. Para esto, la interfaz debe ser adaptable a varios tamaños de pantalla, haciendo que los elementos visuales sean reconfigurados dependiendo del espacio disponible. Por otra parte, distintos tamaños de dispositivos también implican distintas capacidades de procesamiento, por lo que el simulador debe comportarse de manera aceptable en dispositivos con menos recursos.

3.3. Sobre la relevancia de la solución

Resolver este problema es relevante, por al menos dos motivos. El primero de estos es que el simulador es una herramienta útil para comprender las diferencias entre los protocolos que se estudian, y entender estos protocolos es relevante para tener una base del conocimiento sobre cómo funcionan las redes de transporte. Por otra parte, es necesario realizar estas mejoras para que más personas puedan acceder al simulador a través de distintos dispositivos y tengan un recurso adicional que les ayude en su estudio.

3.4. Requisitos del software

La nueva implementación del simulador debe cumplir varias características para ofrecer funcionalidades similares al simulador actual e incorporar soluciones a los problemas discutidos anteriormente. A continuación se ofrece una lista de funcionalidades que el simulador debe cumplir:

[2]URL del simulador: `https://users.dcc.uchile.cl/~jpiquer/srgbn4.html`

[3]De acuerdo a mediciones de la plataforma educativa EOL de la Universidad de Chile (eol.uchile.cl), entre febrero y octubre de 2020, un 38,97 % del tráfico de esta corresponde a dispositivos móviles, mientras que un 61,03 % corresponde a dispositivos de escritorio (estadística facilitada por José M. Piquer, Vicerrector de Tecnologías de la Información de la Universidad de Chile).

1. La interfaz del simulador debe permitir configurar la simulación escogiendo entre los protocolos *Stop-and-Wait*, *Go-Back-N* y *Selective Repeat* (incluyendo confirmaciones acumulativas).
2. El simulador debe permitir configurar el retraso de los paquetes, el tiempo de espera para retransmisión, el tamaño de ventana, la cantidad de paquetes enviados por minuto, y la probabilidad de pérdida de los paquetes.
3. El simulador debe proveer de una visualización que muestre el intercambio de paquetes entre un emisor y un receptor.
4. La visualización debe diferenciar entre paquetes por enviar y confirmados, paquetes en camino y repuestas o confirmaciones, paquetes a la espera de ser recibidos, recibidos y perdidos.
5. La visualización debe mostrar las ventanas deslizantes del emisor y receptor dependiendo del protocolo, y debe mostrar los tiempos de espera correspondientes.
6. El usuario debe poder hacer *click* sobre los paquetes que están en camino para eliminarlos o perderlos.
7. El simulador debe mostrar estadísticas en vivo sobre el intercambio de paquetes, con medidas como por ejemplo la cantidad de paquetes enviados por segundo y paquetes confirmados por segundo.
8. El simulador debe evitar los problemas de ejecución producto de que la pestaña de la aplicación se vuelve inactiva.
9. El simulador debe proporcionar una interfaz adaptable a distintos tamaños de pantalla.
10. El simulador debe tener un rendimiento aceptable en distintos dispositivos.

3.5. Sobre la calidad del software y criterios de aceptación

Para que la solución proporcionada sea aceptable, esta debe ser implementada conforme a ciertos criterios de calidad. Para empezar, la solución debe cumplir con todos los requisitos mencionados anteriormente. Por otra parte, se espera que la implementación cumpla con parámetros sobre los que se habla en los párrafos siguientes.

La implementación debe tener una arquitectura clara, con bajo acoplamiento entre sus componentes. Debe también, presentar documentación transparente sobre sus clases y funciones. Y además, debe incluir pruebas unitarias que aseguren el comportamiento correcto de las simulaciones.

Por otra parte, la interfaz de la aplicación debe poder adaptarse de forma fluida a tamaños de pantalla pequeños, con un tamaño de 320 píxeles de ancho por 568 píxeles de alto (dimensiones de pantalla de iPhone 5/SE) como objetivo de dimensiones mínimas a soportar. Además, el rendimiento de la visualización debe ser aceptable tanto en dispositivos de escritorio como móviles, apuntando a una tasa de alrededor de 30 fotogramas por segundo. Y por último, la interfaz debe poder ser usable en dispositivos con pantallas táctiles, proporcionando botones y controles de tamaños adecuados.

Capítulo 4

Solución

En este capítulo se discute el diseño de la interfaz gráfica de usuario y cómo esta es dividida en componentes que facilitan su disposición espacial e implementación. Se revisa además, la arquitectura del software implementado, y los beneficios de un diseño modular, con programación basada en eventos. Y por último, se presenta el diseño de las clases de la librería de protocolos, y el de las clases de los componentes de la interfaz gráfica.

4.1. Metodología de trabajo

La metodología de trabajo utilizada involucra desarrollo guiado por pruebas, y el uso de herramientas provistas por GitHub para la gestión de proyectos[1]. Así, el flujo de trabajo parte por crear un prototipo de la aplicación en Figma, para luego comenzar la implementación de la librería de protocolos utilizando TDD (ver sección 2.4), y posteriormente el desarrollo de la interfaz gráfica. Para mantener un orden en las tareas, se utiliza la herramienta de manejo de proyectos de GitHub, la que posee integración con el repositorio de Git[2] y permite automatizar los cambios de estado de las tareas.

4.2. Entorno de desarrollo y estructura del proyecto

El entorno de desarrollo utilizado consta de los navegadores Google Chrome y Firefox, para ejecutar y testear la aplicación, en el sistema operativo Ubuntu 20.04. Para escribir el código se utiliza el editor de texto Visual Studio Code, junto al complemento Live Server[3], que permite servir de forma local la página web desarrollada y recargarla automáticamente cada vez que se detecta un cambio en el código. Se usa también Node.js 14 para la generación de documentación con JSDoc, y la instalación local de Mocha.

La estructura de los archivos del proyecto se divide en 5 directorios y 4 archivos principales descritos brevemente a continuación:

[1] `https://github.com/features/project-management/`

[2] Repositorio del proyecto: `https://github.com/JoseIVP/transport-protocols-simulator`

[3] `https://github.com/ritwickdey/vscode-live-server`

1. **components**: contiene un directorio para cada componente de la interfaz gráfica. Cada uno de estos directorios contiene un archivo de JavaScript para la lógica del componente y un archivo CSS para su estilo.
2. **docs**: contiene el sitio web[4] generado por JSDoc con la documentación de la librería de protocolos.
3. **fontawesome**: contiene los archivos de FontAwesome[5], una librería de íconos utilizados en la interfaz gráfica.
4. **jsdoc-template**: es un sub-repositorio de Git con la plantilla utilizada para la generación de documentación de JSDoc.
5. **lib**: contiene dos subdirectorios, **src** y **tests**, donde el primero corresponde al código fuente para las clases de la librería de protocolos y el segundo contiene las pruebas unitarias de Mocha.
6. **index.html**: es el archivo HTML de entrada para la aplicación, contiene la estructura semántica de la interfaz general y de cada uno de los componentes.
7. **main.js**: es el archivo de entrada a la lógica de la aplicación, referido por **index.html**.
8. **main.css**: archivo principal de CSS que configura los aspectos estilísticos expuestos por los componentes para integrarlos al diseño de la página, y también es referido por **index.html**.
9. **README.md**: archivo Markdown del repositorio, que explica cómo configurar un entorno para trabajar en el desarrollo de la aplicación.

4.3. Diseño de la interfaz gráfica

La interfaz gráfica de usuario es diseñada con Figma, una herramienta visual para la generación de prototipos. Esta herramienta permite crear componentes que luego pueden ser reutilizados en cualquier lugar del diseño. Esto, junto a la posibilidad de crear y mantener una paleta de colores, diferentes estilos de texto y restricciones en el posicionamiento de los elementos, permiten fácilmente lograr una interfaz consistente.

En el diseño de la interfaz (ver figuras 3 y 4 en el anexo) se decide dividir o agrupar las partes de la aplicación en distintos paneles o tarjetas. Así, cada panel tiene una utilidad específica, lo que da un orden a la interfaz y facilita su implementación a partir de componentes. Inicialmente, los componentes diseñados incluyen: una tarjeta que contenga los controles y el formulario para configurar la simulación; un panel para la visualización de la simulación; una tarjeta para la presentación de estadísticas; y una sección que presenta una visualización adicional para SW.

Por otra parte, el diseño construido inicialmente sufrió algunas modificaciones durante el desarrollo (ver figuras 5, 6 y 7 en el anexo), dejando algunos elementos fuera de la aplicación. Estos elementos corresponden al botón para descargar la información de la simulación, el botón que permite la configuración a partir de un archivo, la sección para la visualización de SW y los botones que permiten obtener información o ayuda sobre una sección. Estos elementos se dejaron fuera de la implementación para dar prioridad a nuevas ideas que surgieron

[4] `https://joseivp.github.io/transport-protocols-simulator/docs/`
[5] `https://fontawesome.com/`

durante el desarrollo, como la pausa manual de la simulación, un gráfico para mostrar estadísticas, la capacidad de dañar paquetes manualmente, y ajustes o refinamientos generales del proyecto.

Todas las tarjetas de la interfaz poseen una estructura similar, con un encabezado que contiene el título de la sección y un menú de controles o botones, y luego el contenido de la tarjeta. A continuación se discuten los detalles de las diferentes secciones presentes en el diseño original y también aquellos agregados durante la implementación.

4.3.1. Configuración de la simulación

La tarjeta de configuración de la simulación presenta en su encabezado botones que cambian de estado según sean accionados. El primero de estos botones permite cambiar la distribución de las tarjetas en la pantalla y ocupar todo el espacio disponible. Los dos botones siguientes permiten controlar la ejecución de la simulación (iniciar/detener y pausar/reanudar). Y el último botón permite abrir y cerrar la tarjeta de configuración (oprimir el título de la tarjeta también sirve), para así dejar más espacio en la pantalla a las demás secciones.

En su contenido, esta tarjeta posee controles para escoger el protocolo de la simulación, y configurar el retraso de los paquetes, el tiempo de espera para retransmisión, la tasa de paquetes enviados por minuto, el tamaño de la ventana deslizante, y las probabilidades de pérdida y daño. El control escogido para configurar la mayoría de las propiedades anteriores es un control deslizante de rango. El control mencionado es mucho más amigable en dispositivos táctiles que utilizar un teclado para ingresar uno por uno los parámetros.

4.3.2. Visualización

La tarjeta para esta sección presenta en su contenido la visualización para el intercambio de paquetes entre un emisor (parte superior de la visualización) y un receptor (parte inferior). Esta visualización muestra "pistas" por las que viajan los paquetes, donde cada pista corresponde a un número de secuencia. Aquí, el usuario puede presionar los paquetes que viajan para eliminarlos, utilizando adicionalmente la tecla *control* del teclado para dañarlos. Adicionalmente, el contenido de la tarjeta muestra una leyenda con referencias a los colores utilizados en la visualización.

Esta tarjeta posee en su encabezado un botón que permite mostrar o esconder las pistas adicionales por las que viajan los paquetes. Esto permite visualizar de mejor manera aquellas configuraciones que utilicen una ventana deslizante de gran tamaño. Así, podemos también mostrar o no, una visualización más grande dependiendo de lo que le acomode al usuario, según las dimensiones de la pantalla que este usando.

4.3.3. Estadísticas en vivo

Esta tarjeta muestra en su contenido una tabla con estadísticas en vivo sobre la simulación y una visualización que grafica algunas de estas estadísticas a lo largo del tiempo. Además, en su encabezado posee un botón para cambiar el orden de las figuras mostradas. Lo anterior le permite, nuevamente, al usuario disponer los elementos de la página en la forma que más le acomode.

4.4. Arquitectura de la aplicación

La aplicación desarrollada puede ser dividida en dos partes, la implementación de la interfaz de usuario, y la librería de protocolos. Y ambas partes, poseen también una estructura divisible, o modular. Esta división surge por varios motivos sobre los que se hablan a continuación.

Para comenzar, la modularización y separación de responsabilidades permite establecer de forma más fácil una serie de pasos o tareas a realizar para lograr una implementación. Por otra parte, esta separación ayuda también a facilitar la implementación y ejecución de pruebas unitarias. Y además, ayuda a la mantención y el crecimiento del proyecto, ya que facilita el reemplazo y la adición de componentes.

La arquitectura de la aplicación es guiada por la programación basada en eventos utilizada para su implementación. Por una parte, la interfaz de usuario, que se divide en tres componentes - uno por cada sección o tarjeta -, escucha los eventos generados por el usuario, y los comunica, cuando corresponda, a los objetos provenientes de la librería de protocolos. Y por otro lado, los objetos de la librería de protocolos - encargados de la simulación - generan eventos relacionados a la simulación y los comunican, en este caso, a los componentes de la interfaz de usuario.

En la implementación, lo anterior se puede visualizar en el código del archivo principal o de entrada a la aplicación **main.js**. Aquí, se configuran los eventos sobre los que se desea escuchar y las acciones que se deben llevar a cabo en cada caso.

4.5. Diseño de clases para la librería de protocolos

Las clases de la librería de protocolos fueron diseñadas de forma modular tanto para facilitar el desarrollo de pruebas unitarias como para representar, de manera muy general y agrandes rasgos, los elementos de la red que participan en el funcionamiento de un protocolo. Así, se diseña una clase para representar los paquetes de información, clases que representan nodos emisores y receptores de paquetes, y una clase para representar a los canales de comunicación entre los nodos (el anexo contiene diagramas UML de las clases aquí descritas).

Además de las clases anteriores, se diseña una clase especial para el control de tiempos de espera - o *timeouts* en inglés -, y una clase para la clasificación de paquetes y el computo de estadísticas. La primera de estas, es muy importante para el funcionamiento de la simulación y la aplicación, ya que permite pausar y reanudar la ejecución de estas. Mientras que la segunda, permite el computo de estadísticas sin interferir en la implementación de los protocolos.

4.5.1. La clase Packet

Esta clase es la más simple de todas. Su misión es representar un paquete de la red, conteniendo toda la información necesaria para el funcionamiento de los protocolos. Esta información incluye: una referencia al nodo emisor y otra al nodo receptor, el número de secuencia y el número de confirmación del paquete, propiedades booleanas que indican si es una confirmación, confirmación acumulativa, o si el paquete está dañado, y una propie-

dad especial - que no es utilizada para el funcionamiento de los protocolos, pero si para la visualización y el computo de estadísticas - que indica si el paquete es un reenvío.

Todas las propiedades anteriores, a excepción de la que indica el daño del paquete, pueden ser asignadas a través del constructor. Por otra parte, la clase posee un método *damage()* para dañar el paquete, después de cuyo uso la propiedad correspondiente indica que el paquete esta dañado.

4.5.2. La clase Channel

Esta clase representa a la red o el canal de comunicación entre dos nodos. Su objetivo es manejar la entrega de paquetes y la sincronización con la que esto se realiza. Y por otra parte, se encarga de administrar la pérdida y el daño de los paquetes que son enviados a través de este.

Los objetos de esta clase mantienen propiedades que indican el retraso con el que viajan los paquetes, y las probabilidades de pérdida y daño. Estas propiedades pueden ser asignadas a través del constructor de la clase, pero también son públicas. La clase se diseña así, para permitir simulaciones con un comportamiento dinámico de la red.

Los métodos públicos correspondientes a esta clase, permiten enviar un paquete, dañar y perder paquetes que se encuentren en el canal, obtener un iterador sobre los paquetes que viajan, y pausar, reanudar o detener la ejecución de eventos correspondientes al canal. Parte de estos eventos, como la pérdida y daño de paquetes, o la interrupción de la entrega de estos, son comunicados a través de *callbacks* correspondientes, asignadas por el usuario de la clase.

4.5.3. Las clases de los protocolos

Las clases de los nodos, que representan a los emisores y receptores de los protocolos, poseen un diseño basado en herencia de clases. Todas estas clases tienen como ancestro común a la clase abstracta AbstractArqNode. Esta clase implementa la lógica y define las interfaces comunes a todos los nodos.

4.5.3.1. AbstractARQNode

AbstractARQNode define métodos públicos comunes para enviar y recibir paquetes. Por otra parte, define también *callbacks* a través de las que el usuario puede obtener información sobre la ocurrencia de estos eventos. Los métodos mencionados, son las principales interfaces a través de las que el usuario interactua con este tipo de objetos.

Por otro lado, AbstractARQNode define métodos abstractos, como parte de la lógica de los métodos anteriores. Y además, define métodos protegidos, como útilidades comúnes usadas por las clases herederas. Así, esta clase es extendida directamente por aquellas que definen la base de emisores y receptores, AbstractARQSender y AbstractARQReceiver respectivamente.

4.5.3.2. AbstractARQReceiver

Esta es una clase muy simple, cuyo fin es agrupar las clases receptoras. Y por otra parte, inhibe la funcionalidad de enviar paquetes que no sean confirmaciones. Así, las clases herederas - SimpleReceiver, AbstractSequentialReceiver y SRReceiver - deben ocuparse únicamente de recibir paquetes y procesarlos para enviar confirmaciones.

4.5.3.3. SimpleReceiver

SimpleReceiver surge como una clase utilitaria para facilitar la implementación de pruebas unitarias. Su única función es recibir paquetes y comunicar su recepción a través del *callback* correspondiente asignado por el usuario. De esta manera, puede actuar como receptor de paquetes o confirmaciones en pruebas donde no se espera que esten implementadas ambas partes de un protocolo.

4.5.3.4. AbstractSequenctialReceiver

AbstractSequentialReceiver agrupa las clases receptoras de los protocolos SW (SWReceiver) y GBN (GBNReceiver), e implementa toda la lógica de estas partes. Esto sucede ya que ambos receptores se comportan de la misma manera, difiriendo únicamente en el rango de números de secuencia utilizados. Así, dichas clases solo deben extender a ésta e invocar su constructor con los parámetros adecuados.

4.5.3.5. SRReceiver

Esta clase implementa la lógica del receptor de SR. Incluye lo necesario para el uso de una ventana deslizante, agregando la definición de un *callback* que comunique el evento relacionado al desplazamiento de aquella. Y además, incluye la lógica opcional para el uso de confirmaciones acumulativas.

4.5.3.6. AbstractARQSender

AbstractARQSender agrupa a las clases emisoras, SWSender y AbstractWindowedSender, e implementa la funcionalidades y utilidades comunes entre estas. Dichas utilidades incluyen métodos protegidos que permiten la activación y desactivación de tiempos de espera, y las definiciones de los *callbacks* que comunican estos eventos, además de la confirmación de paquetes. Y por otra parte, las funcionalidades incluyen métodos para la pausa, reanudación y cancelación de los tiempos de espera activos.

4.5.3.7. SWSender

Esta clase implementa la lógica para el emisor de SW. En su estructura define propiedades de solo lectura, que transparentan el estado del nodo. Y por otra parte, implementa en los métodos abstractos heredados la lógica que le corresponde.

4.5.3.8. AbstractWindowedSender

Esta clase abstracta agrupa aquellas clases emisoras que utilizan una ventana deslizante. Además, implementa las propiedades, utilidades y funcionalidades comunes entre estas. Esto

incluye la definición de propiedades que mantienen y transparentan el estado del nodo, y métodos para el manejo de la ventana deslizante y la comunicación de su desplazamiento. Así, las clases que la extienden, GBNSender y SRSender, solo deben implementar sus lógicas correspondientes en los métodos abstractos heredados.

4.6. Diseño y desarrollo de pruebas unitarias para la librería de protocolos

Para desarrollar las pruebas unitarias de la librería de protocolos se utiliza el *framework* Mocha, y la librería de aserciones Chai[6]. El trabajo en conjunto de ambos módulos de software permiten la ejecución automatizada de pruebas en el navegador. Este último punto es decisivo para escoger entre Mocha y Jest, ya que el segundo solo admite la ejecución de pruebas en el entorno de ejecución Node.js, y la librería que se desarrolla apunta primariamente a su uso en el navegador.

Se crean pruebas unitarias para los métodos públicos más importantes de cada clase. También se crean pruebas de integración para la interacción entre el emisor y receptor de cada protocolo, simulando una comunicación donde se pierden y dañan paquetes. Las pruebas pueden ser ejecutadas de forma automática cada vez que se cambia el código fuente, gracias a que Mocha se ejecuta en el navegador en un página web servida localmente a través de Live Server (ver sección 4.2).

El desarrollo de pruebas se lleva a cabo por módulos, testeando primero el módulo de utilidades, que incluye en si mismo una función para facilitar el desarrollo de pruebas asíncronas, y el objeto que permite el manejo de tiempos de espera. Luego, se tiene pruebas para la clases Packet, SimpleReceiver y Channel. Y finalmente, se tienen las pruebas para cada uno de los módulos correspondientes a los protocolos SW, GBN y SR.

Como se mencionó anteriormente, el módulo de utilidades incluye una función para facilitar la implementación de pruebas. A esta función se le llama *sleep()* y su objetivo es reemplazar - mediante el uso de promesas - el uso de la función *setTimeout()* proporcionada por el navegador. Esto, ya que la sintaxis para el uso de *setTimeout()* ofusca la implementación de las pruebas, mientras que *sleep()* hace uso del azúcar sintáctico provisto por JavaScript - que incluye las palabras reservadas *async* y *await* [4] - para escribir código asíncrono de forma síncrona.

Para testear las clases de los protocolos, se recrean escenarios en los que se envían o reciben paquetes y confirmaciones. Para cada parte, las pruebas se enfocan primero en testear el envío y recepción de paquetes, y luego en la ejecución correcta de los métodos o *callbacks* que comunican los distintos eventos que ocurren en el intercambio de paquetes. Aquí, se hace gran uso de la clase SimpleReceiver, que actúa como la contraparte de un nodo en el intercambio de paquetes y ayuda a asegurar que estos sean efectivamente enviados o recibidos.

La última prueba de cada módulo de protocolos, se asegura de que emisor y receptor actúen conjuntamente de manera correcta. Para esto se recrea un escenario en el que el canal

[6] `https://www.chaijs.com/`

de comunicación daña o pierde algunos de los paquetes enviados. Así, se prueba que los nodos sean capaces de avanzar en el intercambio de paquetes en un canal de comunicación no confiable.

4.7. Detalles de implementación de la librería de protocolos

En esta sección se discuten algunos de los detalles importantes de la implementación de la librería, para algunas de sus clases.

4.7.1. La clase Timeout

Esta clase funciona como un envoltorio de los métodos del navegador *setTimeout()* y *setInterval()*, permitiendo la pausa y reanudación de tiempos de espera e intervalos. Su implementación utiliza estados, de manera interna, para saber si debe pausar, reanudar, o terminar el tiempo de espera o intervalo. Aquí, se puede usar un patrón de diseño de estado o *state pattern*[2] en inglés. Sin embargo, no se utiliza debido a la simpleza de las transiciones.

Un detalle importante de la clase, es que cada vez que se inicia o reanuda el tiempo de espera o intervalo se utiliza *setTimeout()* y luego se realiza una transición a *setInterval()*, si es que corresponde. La razón de esto es que *setTimeout()* permite ejecutar el tiempo restante para terminar el periodo de espera completo, o el ciclo actual del intervalo luego de una pausa. Aquí, se puede ocupar siempre *setTimeout()*, en lugar de pasar a *setInterval()*, pero no se implementa de esta forma, ya que *setTimeout()* debe iniciar un hilo de ejecución aparte cada vez que se llama para iniciar el tiempo de espera, y además, se estarían desaprovechando otras potenciales optimizaciones que el navegador pudiera ofrecer con *setInterval()*.

4.7.2. La clase Channel

Para el manejo de los paquetes que se encuentran viajando por el canal, los objetos de esta clase mantienen un diccionario. Este diccionario, permite relacionar los paquetes a objetos especiales que agrupan la información sobre la sincronización de eventos, como la entrega, daño o pérdida del paquete correspondiente.

La información mencionada corresponde a objetos de la clase Timeout. Estos son creados en el momento en que un paquete es enviado a través del canal con el método *send()* de la clase. Ahí, se utilizan las probabilidades de perdida y daño para decidir si ocurrirán los eventos correspondientes para el paquete.

Cuando se computa lo anterior, también se decide en qué momento ocurrirá la pérdida o daño de estos. Para esto, se calcula un momento aleatorio entre uno y tres cuartos del tiempo de retraso o viaje de los paquetes. Esto permite que dichos eventos no ocurran justo al inicio o final del trayecto del paquete en la visualización, y así el usuario pueda ver de mejor manera lo que ocurre.

4.7.3. Sobre las clases de los protocolos

4.7.3.1. Lógica general de recepción y emisión de paquetes

Para llevar a cabo la lógica de los nodos se utiliza una implementación simple del patrón de diseño plantilla o *template pattern*[2] en inglés. Así, la lógica general de la emisión y recepción de paquetes se desarrolla en la clase AbstractARQNode, y las clases que la extienden deben implementar los métodos abstractos correspondientes. Esto permite separar de manera clara las distintas partes en la lógica de los protocolos.

La lógica general para la recepción de paquetes es muy simple, empleando tres métodos abstractos. Al recibir un paquete, primero se utiliza *checkReceivedPacket()* que retorna verdadero si el paquete es correcto y falso de lo contrario, y esta información se pasa junto con el paquete al *callback* que comunica la llegada de un paquete. Luego, dependiendo del valor retornado anteriormente, se ejecuta *processExpectedPacket()* o *processUnexpectedPacket()* para un paquete correcto o incorrecto, respectivamente.

Por otra parte, la lógica general para el envío de paquetes (no confirmaciones) es muy simple también, usando dos métodos abstractos. Primero se verifica que el nodo pueda emitir un paquete con *canSend()*. Y luego, si un paquete puede ser emitido, se llama a *processSending()*, para ejecutar la lógica particular de emisión.

4.7.3.2. Manejo de números de secuencia

Los protocolos ocupan a lo más una cantidad de números de secuencia igual al doble del tamaño de la ventana deslizante que utilizan (en el caso de SW este sería 1, por lo que se usan 0 y 1 como números de secuencia). De esta manera, se obtiene una sucesión circular de números para usar en los paquetes, computando la próxima secuencia, o la previa, con el operador módulo. Y para facilitar esto, AbstractARQNode implementa dos métodos utilitarios que obtienen la secuencia previa y la siguiente dado un número de secuencia actual.

El uso de una cantidad finita de números de secuencia en los protocolos, posibilita el empleo de arreglos para el manejo de objetos relacionados a estos. Esto se utiliza tanto en la implementación de los mismos protocolos, como en la implementación de la visualización, que se discute más adelante.

4.7.3.3. Manejo de tiempos de espera

Dado que todas las clases emisoras de los protocolos utilizan tiempos de espera, AbstractARQSender - la clase que los agrupa - implementa métodos utilitarios para el manejo de tiempos de espera asociados a números de secuencia. Estos permiten crear, eliminar y comprobar la existencia de los tiempos.

Para el manejo de los tiempos, AbstractARQSender utiliza un diccionario (Map) privado. Sin embargo, ya que los protocolos usan a lo más un tiempo de espera por número de secuencia en todo momento, se podría emplear un arreglo de acuerdo a lo discutido en la sección anterior. Esto, podría mejorar un poco el rendimiento de los métodos relacionados.

4.8. Diseño de clases de la interfaz de usuario

En esta sección se habla de manera general sobre las propiedades y métodos proporcionados por las clases de la interfaz de usuario. Al ser componentes web, todas estas clases extienden a la clase HTMLElement, provista por el navegador. Las clases mencionadas corresponden a SettingsCard (la clase para la sección de configuración), PacketVisualization y PacketTrack (correspondientes a la visualización y las pistas dentro de esta, respectivamente), y StatisticsCard (la tarjeta que muestra las estadísticas y el gráfico en vivo). En el anexo pueden verse los diagramas UML de las clases aquí descritas.

4.8.0.1. SettingsCard

SettingsCard mantiene propiedades privadas para el manejo del estado de ejecución, y referencias a elementos de su interfaz gráfica como botones y la tarjeta en sí misma. Además, proporciona métodos que permiten cambiar el estado de ejecución mantenido por el componente, es decir, permiten iniciar, pausar, resumir y detener la simulación. Y por otra parte, define *callbacks* para la comunicación de estos eventos y la configuración asignada por el usuario al momento de iniciar.

4.8.0.2. StatisticsCard

Esta clase posee propiedades privadas que guardan referencias a las distintas celdas donde se muestran los valores de las estadísticas, y arreglos para la información mostrada en el gráfico - del que también mantiene una referencia. Y por otra parte, provee de métodos públicos para la actualización de las estadísticas y la recuperación del estado inicial o vacío del componente.

4.8.0.3. PacketTrack

El objetivo de esta clase es representar las pistas por las que viajan los paquetes entre el emisor y el receptor en la visualización. Esta mantiene referencias privadas a los elementos visuales que representan los paquetes en el lado del emisor y el receptor, aquellos paquetes que viajan, y la representación del tiempo de espera, además de las animaciones relacionadas. Y también, proporciona métodos para iniciar las animaciones relacionadas a la transmisión, daño y pérdida de paquetes, y al transcurso del tiempo de espera.

Por otra parte, esta clase provee de métodos para pausar y reanudar las animaciones mencionadas. Define también un método para restablecer el estado inicial del componente, lo que permite reutilizarlo. Y además define un *callback* para la comunicación de los *clicks* realizados sobre los paquetes que viajan.

4.8.0.4. PacketVisualization

Esta clase se ocupa del manejo general de la visualización del intercambio de paquetes. Para esto, se compone de un objeto de la clase PacketTrack por cada pista de la visualización. Y además, los objetos de la clase deben mantener propiedades privadas para la posición de la visualización, las ventanas del emisor y receptor, arreglos para mantener las pistas, y otras propiedades que posibilitan mostrar más o menos pistas en el diseño de la visualización.

Por otra parte, esta clase proporciona métodos públicos para el control de la visualización. Estos permiten iniciar animaciones para el envío de paquetes o confirmaciones, y el daño, pérdida, confirmación o recepción de estos. Además, entrega métodos para controlar la reproducción de dichas animaciones. Y finalmente, a través de esta clase se asignan los *callbacks* a los objetos de la clase PacketTrack, para obtener la información de los *clicks* sobre los paquetes.

4.9. Detalles de implementación de la interfaz de usuario

4.9.1. Elementos y animaciones de la visualización

Cada objeto de la clase PacketTrack se encarga de mantener el estado de sus propias animaciones. Así, para tener control sobre la reproducción de estas, la clase utiliza la API de animaciones web. Para esto, también asocia cada paquete a su representación y animación correspondiente, y los mantiene en un diccionario.

Por otra parte, PacketVisualization se encarga de decidir qué instancia de PackeTrack utilizar para comenzar una animación. Esto lo hace manteniendo las pistas en orden, una referencia a la siguiente pista para usar, y un arreglo para la asociación de números de secuencia a pistas. La siguiente pista se asocia a un número de secuencia cada vez que un paquete es enviado por primera vez (la propiedad del paquete que indica que es un reenvío es falsa), mientras que un paquete reenviado indica que se debe usar una pista ya asociada.

PacketVisualization también se encarga del desplazamiento de las ventanas deslizantes y de la visualización. Para esto proporciona métodos públicos, utilizados en conjunto con los *callbacks* proporcionados por las clases de los protocolos, para saber cuándo y cuánto mover las ventanas. Y por otro lado, el desplazamiento de la visualización se realiza cada vez que la ventana del emisor alcanza el final de la visualización.

Lo anterior, también es parte de un proceso de reutilización o rotación de las pistas. Así, aquellas pistas que desaparecen - al ser la visualización desplazada - son reubicadas al final de la visualización. Las pistas son reutilizadas, ya que existe una cantidad finita de estas, por motivos de rendimiento y uso de memoria. Así, se utiliza una cantidad de pistas igual a la suma del número máximo de pistas visibles (42) y el número máximo de pistas desplazadas fuera de la visualización (40).

Adicionalmente, se utiliza una optimización para el proceso de reutilización descrito. Ya que se tiene un arreglo con las pistas - donde para modificar la posición de la pista se modifica su atributo SVG y no su posición en el arreglo -, se guardan además los índices en dicho arreglo de la pista en el extremo izquierdo de la visualización y de la primera pista visible desde ese extremo. Esto permite recorrer el arreglo de manera circular, pasando solo por aquellas pistas de la izquierda que ya han quedado fuera de la vista, para reubicarlas.

Por otro lado, para realizar las animaciones de los desplazamientos mencionados PacketVisualization utiliza transiciones de CSS. Se usa dicho tipo, ya que para estas animaciones no se necesita tener control sobre su reproducción. Así, en su implementación basta con cambiar la posición de las ventanas o la visualización, configurar la duración de la transición, y el

navegador se encarga del resto.

Respecto a cómo se dibujan los elementos de la visualización, se utiliza SVG junto a CSS para todos estos. Sin embargo, la implementación posee una sutileza, ya que cada pista es un componente web y SVG no reconoce estos elementos. Así, para solucionar aquello, se utiliza el elemento especial de SVG <foreignObject> - que permite incluir elementos externos - para cada pista de la visualización.

Otros detalle importante se relacionan a los elemento utilizados para el desplazamiento de la visualización. Inicialmente, las pistas - contenidas en elementos <foreignObject> - se posicionan dentro de un elemento <svg> adicional al elemento <svg> raiz de la visualización, para así cambiar la posición de dicho elemento en lugar de la posición de cada pista. Sin embargo, a pesar de que la posibilidad de animar la posición del elemento <svg> a través de CSS es parte del estándar SVG 2, esta todavía no se encuentra implementada en Chrome, a diferencia de Firefox. Así, para poder animar el desplazamiento se utiliza nuevamente <foreignObject>, para el que la animación a través de CSS si se encuentra implementada en ambos navegadores, Chrome y Firefox.

4.9.2. Interfaz gráfica adaptativa

Para implementar una interfaz gráfica que se adapte a distintos tamaños de pantalla, se hizo un fuerte uso de las nuevos métodos incorporados a CSS, como *Flexbox* y *Grid*. Además, se utilizaron en menor medida - en parte gracias al uso de los métodos anteriores - *CSS media queries*[7], una tecnología que permite aplicar estilos al cumplirse condiciones definidas - como por ejemplo, que la pantalla supere ciertas dimensiones -, pero cuyo uso puede incrementar rápidamente la complejidad del código. Y por otra parte, también se utilizaron propiedades personalizadas o *CSS custom properties*[8], única forma de ofrecer acceso a través de CSS, a modificar los estilos de un componente web desde afuera de sus definiciones internas.

La implementación del diseño de cada componente es realizada de tal manera que el tamaño de estos cambie de acuerdo al tamaño de su contenedor. Esto es importante, ya que en CSS tanto el uso de porcentajes (%) - relativos al tamaño del contenedor -, como el de unidades de ancho de vista (vw) - relativas al ancho de la vista - permiten crear interfaces adaptables al tamaño de pantalla, pero el uso de porcentajes posibilita una mayor flexibilidad. Así, se facilita bastante el cambio de diseño general cuando el usuario escoge la disposición de los elementos en forma de columna desde el botón ofrecido en los controles de la sección de configuración, de tal forma que el código solo debe cambiar el flujo estático por el uso de *Grid* (ver figura 10 en el anexo).

Otro aspecto a aclarar, es el uso de la librería Chart.js para la implementación del gráfico que muestra algunas de las estadísticas a lo largo del tiempo. Se decide finalmente utilizar esta librería, por su facilidad de uso y pequeño tamaño respecto a Plotly y D3. Cabe mencionar, que para mantener un aspecto consistente del gráfico, este se renderiza nuevamente, cada vez que la ventana del navegador cambia de tamaño, escuchando el evento correspondiente y evitando así que el gráfico se vuelva borroso.

[7] `https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries`
[8] `https://developer.mozilla.org/en-US/docs/Web/CSS/--*`

Finalizando, en la interfaz de la aplicación se utilizaron íconos de terceros, y para los controles del formulario de configuración se utilizaron propiedades de CSS específicas a los navegadores. Los íconos utilizados corresponden a la librería de íconos FontAwesome 5[9]. Y por otra parte, los controles modificados corresponden a los rangos numéricos, para los que se creó un módulo de CSS aparte, que es importado al estilo del componente correspondiente, y donde se usan *prefijos* específicos de los motores de renderizado de Chrome y Firefox. Lo anterior, permite mantener una consistencia en el diseño entre los navegadores.

4.10. Despliegue de la aplicación

Ya que la aplicación web desarrollada utiliza simple HTML, CSS y JavaScript, sin compilación o transpilación[10] de código, el proceso para servirla como un sitio web es bastante fácil. Para esto se utiliza GitHub Pages[11], un servicio de GitHub que permite crear un sitio web a partir del código alojado en un repositorio de Git. Esto permite también, que las características implementadas puedan ser rápidamente desplegadas[12].

[9] `https://fontawesome.com/`

[10] La transpilación es un proceso que permite traducir un lenguaje de programación a otro, y en desarrollo web se utiliza para traducir entre distintas versiones de JavaScript, permitiendo el uso de nuevas características del lenguaje en proyectos utilizados en navegadores que aún no las soportan.

[11] `https://pages.github.com/`

[12] URL de la aplicación: `https://joseivp.github.io/transport-protocols-simulator/`

Capítulo 5

Validación

En este capítulo se habla sobre cómo la solución propuesta resuelve el problema general de crear una nueva versión del simulador de protocolos. Por otra parte, se discute cómo la solución resuelve el problema del uso de la aplicación en distintos tamaños de pantalla. Y por último, se habla sobre el rendimiento de la aplicación en distintos escenarios.

5.1. Aplicación

La aplicación desarrollada permite al usuario configurar y ejecutar una simulación de los protocolos *Stop-and-Wait*, *Go-Back-N* y *Selective Repeat*. Permite además visualizar el intercambio de paquetes que ocurre entre el emisor y receptor en la ejecución del protocolo. Y por último, permite obtener información estadística en vivo sobre la ejecución de la simulación.

De esta forma, un usuario que utilice la aplicación primero puede configurar la simulación en la sección de configuración. Aquí se pueden cambiar los parámetros que indican el protocolo a utilizar, el retraso de los paquetes, el tiempo de espera para retransmisión, la tasa de emisión de paquetes por minuto, el tamaño de la ventana deslizante, y las probabilidades de pérdida y daño de paquetes. Luego el usuario puede utilizar los controles que se encuentran en el encabezado de la sección de configuración para iniciar, pausar, reanudar o detener la simulación.

Luego de iniciada la simulación, el usuario que utiliza la aplicación puede interactuar con la simulación en la sección de visualización. Aquí el usuario puede ver qué paquetes son enviados, recibidos, confirmados, dañados o perdidos, y cuál es el estado de las ventanas deslizantes. También aquí, el usuario puede interactuar con los paquetes que viajan para eliminarlos o dañarlos, haciendo *click* sobre ellos.

Mientras la simulación es ejecutada, el usuario puede también ver información estadística sobre lo que ocurre con ésta en la sección de estadísticas en vivo. Aquí se muestra una tabla que muestra las tazas reales sobre el envío de paquetes, y la confirmación de estos, además de la cantidad total de paquetes enviados, reenviados y recepcionados, confirmaciones, etc. Y por otro lado, se muestra un gráfico que permite visualizar cómo han cambiado las tazas mencionadas en los últimos 30 segundos.

La aplicación detecta cuando el usuario a cambiado a otra pestaña del navegador, pausando así la ejecución de la simulación, y evitando los problemas causados por las optimizaciones del navegador cuando esto sucede. De esta forma, la aplicación le permite al usuario reanudar la ejecución cuando vuelva a la pestaña correspondiente a la simulación.

La solución es validada, por una parte, a través de la ejecución y éxito de las pruebas unitarias para la librería de protocolos. Y por otra parte, se valida a través de la inspección visual de la visualización de la simulación. Lo anterior se facilita por la interactividad de la visualización, que permite eliminar o dañar paquetes y crear situaciones para las que se espera un resultado de acuerdo al estudio anterior del comportamiento de los protocolos.

5.2. Interfaz gráfica adaptativa

El diseño e implementación de la interfaz gráfica, permite al usuario utilizar la aplicación en un gran rango de dimensiones de pantalla, desde dispositivos de escritorio, con típicas dimensiones de 1920 por 1080 píxeles, hasta dispositivos móviles como tabletas o teléfonos inteligentes, con dimensiones de 320 por 568 píxeles (ver figuras 8 y 9 en el anexo) o incluso menos. La aplicación aprovecha el espacio de pantalla disponible, cambiando la disposición de los elementos, lo que permite - cuando sea posible - ver más información al mismo tiempo. Y por último, el diseño de la interfaz provee controles de tamaños adecuados, en la mayoría de los casos (si la pantalla es muy pequeña se dificulta presionar los paquetes), tanto para dispositivos controlados por ratón y teclado, como para aquellos controlados por tacto.

5.3. Rendimiento de la visualización

Para probar el rendimiento de la aplicación se utilizan las herramientas de desarrollo del navegador Google Chrome 86, en específico las herramientas de monitoreo de rendimiento. Esto, en un computador con sistema operativo Ubuntu 20.04, procesador Intel Core i7-4720HQ con 4 núcleos a 2.60GHz, 12 GB de RAM, tarjeta gráfica NVIDIA GTX 960M y resolución de pantalla de 1920 por 1080 píxeles. El rendimiento de la aplicación se mide en base a la cantidad de *layouts* por segundo generados por el navegador para la página, utilizando el procesador a capacidad normal y también a un sexto de su capacidad (disminución emulada por el navegador).

El rendimiento de las animaciones de la visualización es probado escogiendo escenarios arbitrarios en los que se percibe un gran intercambio simultaneo de paquetes. Estos escenarios podrían ser problemáticos (en términos de rendimiento o suavidad de las animaciones) en dispositivos con menos poder de procesamiento. Lo anterior es importante debido al aumento en el uso de dispositivos móviles y a la diferencia en el poder de procesamiento de estos respecto a los computadores convencionales.

Las pruebas consisten en la ejecución de los protocolos GBN y SR + CACK, con una ventana deslizante de tamaño 40 (la configuración máxima en la aplicación), un retraso de 1000 ms, tiempo de espera de 2500 ms, taza de emisión de 120 paquetes por segundo (configuración máxima), probabilidad de pérdida de 0.3, y probabilidad de daño 0. Así, para la ejecución de la simulación durante al menos un minuto con procesador a capacidad normal

se obtienen lecturas de alrededor de 60 o más *layouts* por segundo, la mayoría del tiempo, en ambos protocolos. Mientras que en la ejecución a un sexto de la capacidad del procesador se obtinen lecturas de alrededor de 30 *layouts* por segundo, la mayoría del tiempo, en ambos protocolos.

Adicionalmente se testea la aplicación en un teléfono inteligente de gama baja, con sistema operativo Android 8.1 Go, procesador de 4 núcleos a 1.5GHz, 1 GB de RAM y resolución de pantalla de 720 por 1280 píxeles. Las pruebas corresponden a las mismas descritas en el párrafo anterior, ejecutadas a la capacidad normal del procesador, y el rendimiento, en este caso, es juzgado únicamente por la percepción visual de las animaciones. Así, se percibe que las animaciones deben correr a una taza de 20 fotogramas por segundo, que sin embargo, disminuye dramáticamente cuando la cantidad de paquetes visibles aumenta.

De acuerdo a lo anterior, se espera que la aplicación tenga un rendimiento aceptable en un gran rango de dispositivos de gama alta y media. Por otra parte, en dispositivos de gama baja, con menos recursos computacionales, se espera una experiencia degradada en cuanto a la fluidez de las animaciones en la visualización.

Capítulo 6

Conclusiones y trabajo futuro

El trabajo de esta memoria consiste en el diseño e implementación de una nueva versión de un simulador de protocolos de transporte utilizado para apoyar la docencia en un curso de redes computacionales. Para esto, se utilizan herramientas para la creación de prototipos, tecnologías de desarrollo web, herramientas para el versionamiento y el manejo de proyectos, y herramientas para la ejecución de pruebas automatizadas y la generación de documentación. El trabajo se lleva a cabo utilizando metodologías como el desarrollo guiado por pruebas y la programación basada en eventos, dando como resultado un software diseñado de forma modular, que consta de una librería para la ejecución de simulaciones y una interfaz que permite visualizarlas e interactuar con estas.

Con el trabajo que se realiza, se logra alcanzar el objetivo general de crear una herramienta que pueda complementar el estudio teórico de los protocolos de transporte. Para esto, se cumplieron los objetivos específicos del proyecto, como el diseño y la implementación de una librería en JavaScript que permita la ejecución de simulaciones para distintos protocolos, la creación de pruebas unitarias que aseguren un correcto desempeño de las simulaciones, y el diseño e implementación de una interfaz gráfica de usuario que permita visualizar la ejecución y consumir los resultados de la simulación.

Por otra parte, se logra cumplir todos los requisitos de software, implementando una interfaz gráfica de usuario que permite la configuración y visualización interactiva de las simulaciones. Además, dicha interfaz es adaptable a distintos tamaños de pantalla, y el rendimiento de sus animaciones debiera ser aceptable en una gran variedad de dispositivos. Sin embargo, para dispositivos con escasos recursos computacionales se espera una experiencia degradada en cuanto al rendimiento de las animaciones.

Esta nueva implementación del simulador, permite al usuario tener una experiencia más amigable con el software, sobre todo en dispositivos móviles. Y por otra parte, también mejora la experiencia de usuario en dispositivos de escritorio y laptops, permitiendo ver de manera cómoda más información sobre la simulación al mismo tiempo. Y además, este trabajo ofrece un poco más de seguridad en cuanto a la correctitud de la simulación, debido a las pruebas unitarias ejecutadas sobre el código. Todo lo anterior favorece bastante al uso docente de esta herramienta.

Por otro lado, este trabajo permite aprender algunas lecciones sobre el desarrollo de este tipo de software. Lo primero, es que la programación basada en eventos ayuda enormemente a la separación de responsabilidades de los objetos de código y a la modularización. Segundo, es que uno de los primeros pasos para la implementación de visualizaciones animadas a partir de simulaciones, es - aunque suene obvio - identificar los elementos a animar y los eventos de la simulación que deben iniciar estas animaciones. Y tercero, en la implementación de animaciones web es muy importante escoger los métodos adecuados de dibujo y animación, para obtener la mejor combinación de control y rendimiento que el navegador puede ofrecer.

Finalmente, el trabajo ofrecido en esta memoria puede ser extendido y mejorado. En primer lugar, el proyecto podría ser portado, módulo a módulo, a TypeScript[1], lo que aportaría bastante a la robustez, sobre todo de la librería de protocolos. Por otra parte, se podría intentar realizar optimizaciones a la implementación para obtener un mejor rendimiento de las animaciones, como ejecutar la simulación en un hilo separado, que era una de las intenciones de esta implementación. Y además, se podrían realizar mejoras a la experiencia de usuario de la aplicación, incorporando secciones explicativas o de ayuda, mejorando el soporte para discapacidades relacionadas a la percepción de colores, agregando traducción al español, o incluso permitiendo instalar la aplicación de forma local para su uso sin internet.

[1]TypeScript es un superconjunto de JavaScript que agrega definición estática de tipos, permitiendo comportamientos más estrictos gracias a la verificación de éstos en tiempo de compilación. El código en TypeScript es compilado a JavaScript, pudiendo ser ejecutado en cualquier lugar que admita JavaScript.

Bibliografía

[1] Kent Beck. *Test Driven Development: By Example.* A Kent Beck Signature Book;The Addison-Wesley Signature Series. Addison-Wesley Professional; Pearson Education (US), 2002.

[2] Ralph Johnson John M. Vlissides Erich Gamma, Richard Helm. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1 edition, 1994.

[3] Ted Faison. *Event-Based Programming.* Apress, 1 edition, 2006.

[4] David Flanagan. *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language.* O'Reilly Media, 7 edition, 2020.

[5] Keith W. Ross James F. Kurose. *Computer Networking: A Top-Down Approach.* Addison-Wesley, 6 edition, 2013.

Apéndices

Apéndice A. Problema

En este apéndice se muestran capturas de pantalla del simulador actual de protocolos. Las capturas permiten apreciar la disposición de los elementos de la interfaz y comparar el simulador con la nueva versión.

Figura 1: Porción superior del simulador actual.

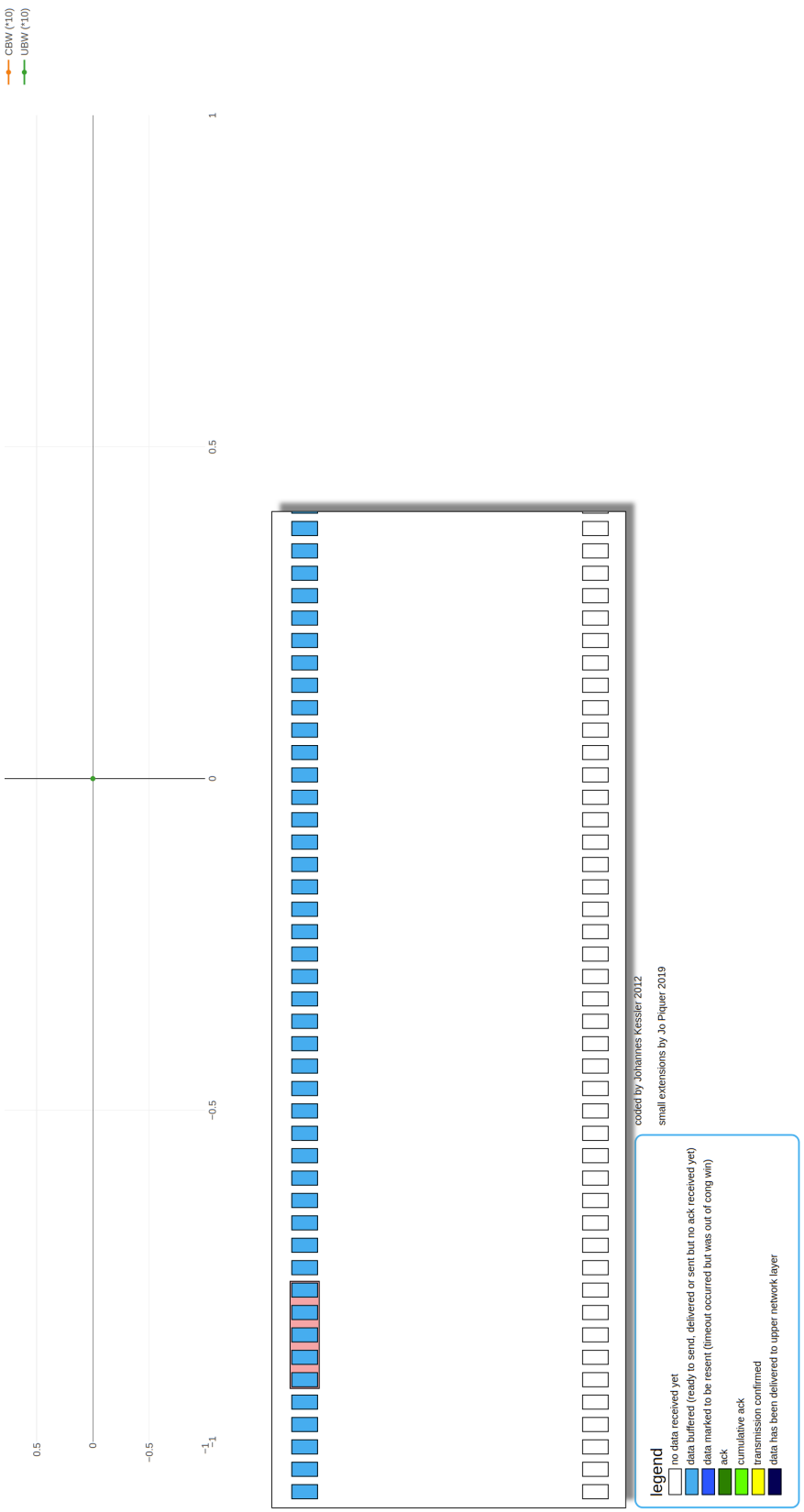

Figura 2: Porción inferior del simulador actual.

Apéndice B. Solución

Este apéndice muestra las figuras asociadas al capítulo 4. Estás permiten realizar una comparación entre el diseño del simulador actual, el prototipo elaborado en Figma, y la implementación final del nuevo simulador. Adémas, se muestran los diagramas UML de las clases de la librería de protocolos y de las clases que componen a la interfaz gráfica.

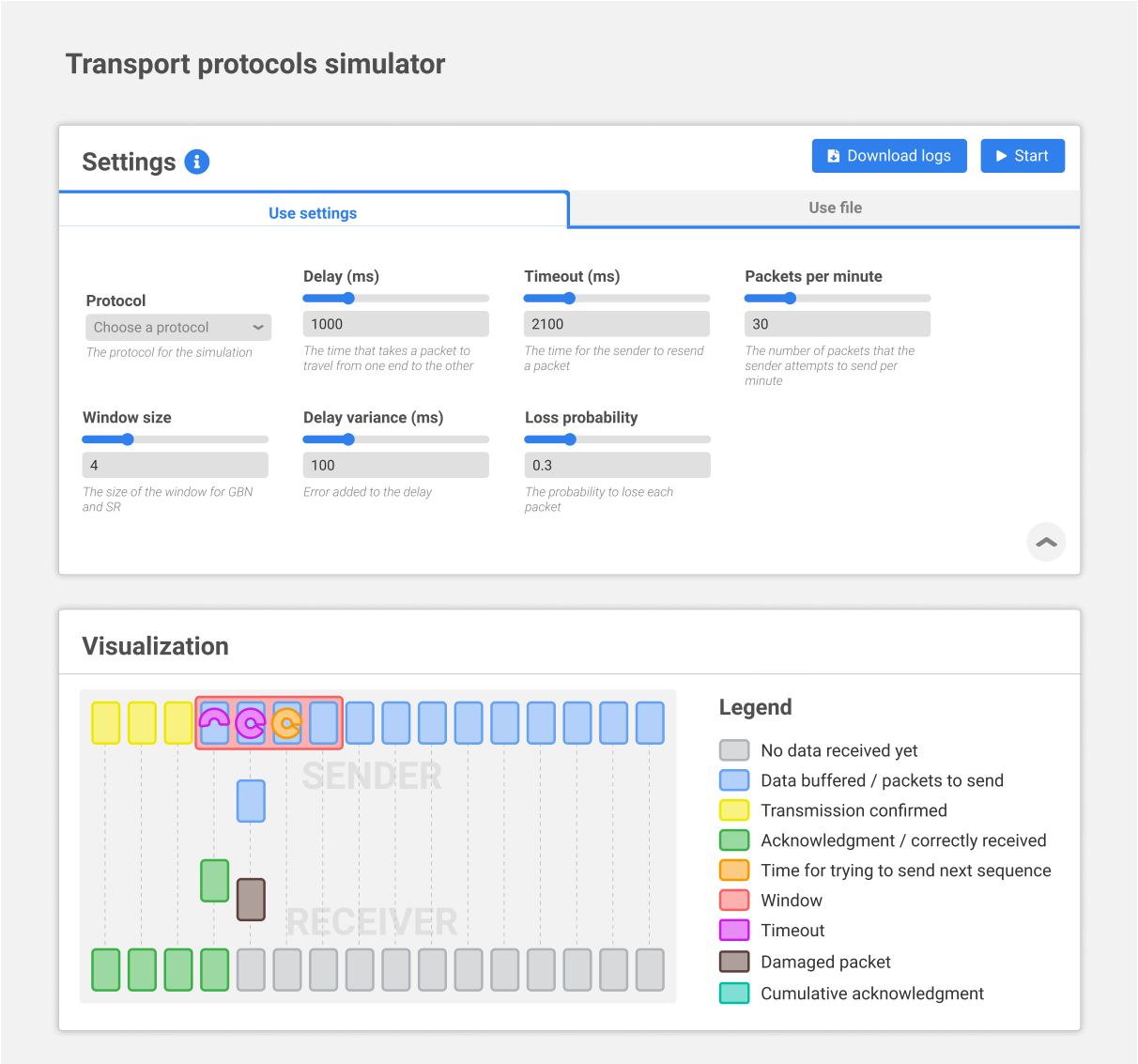

Figura 3: Prototipo desarrollado en Figma (porción superior de la página).

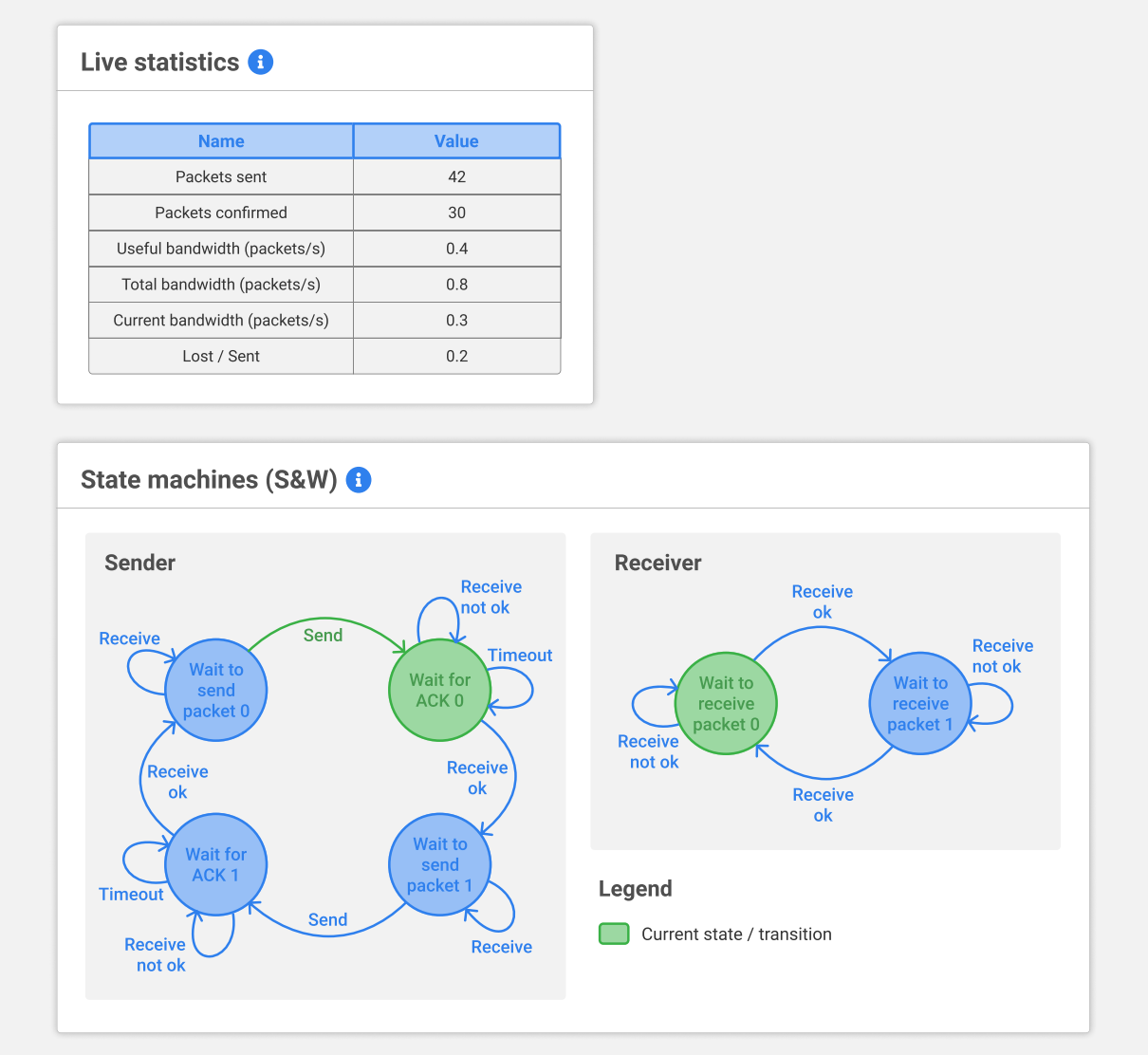

Figura 4: Prototipo desarrollado en Figma (porción inferior de la página).

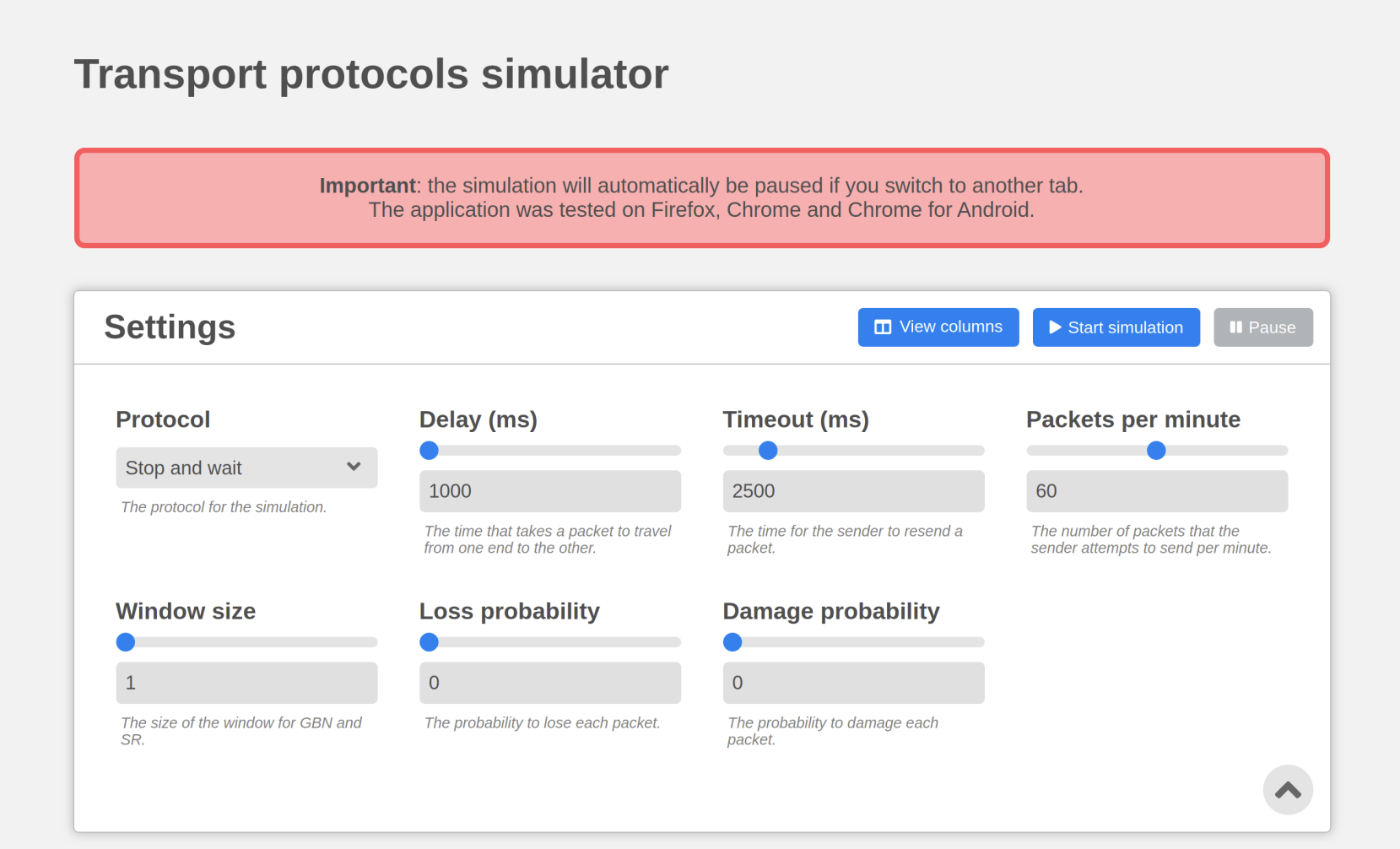

Figura 5: Interfaz real (sección de configuración).

Figura 6: Interfaz real (sección de visualización).

Measure	Value
Packets sent / s (last 2 s)	0
Packets sent / s (from start)	0
Packets confirmed / s (last 2 s)	0
Packets confirmed / s (from start)	0
Acks received ok / s (last 2 s)	0
Acks received ok / s (from start)	0
Packets sent	0
Packets confirmed	0
Packets re-sent	0
Packets received	0
Packets received ok	0
Acks sent	0
Acks received	0
Acks received ok	0

Figura 7: Interfaz real (sección de estadísticas).

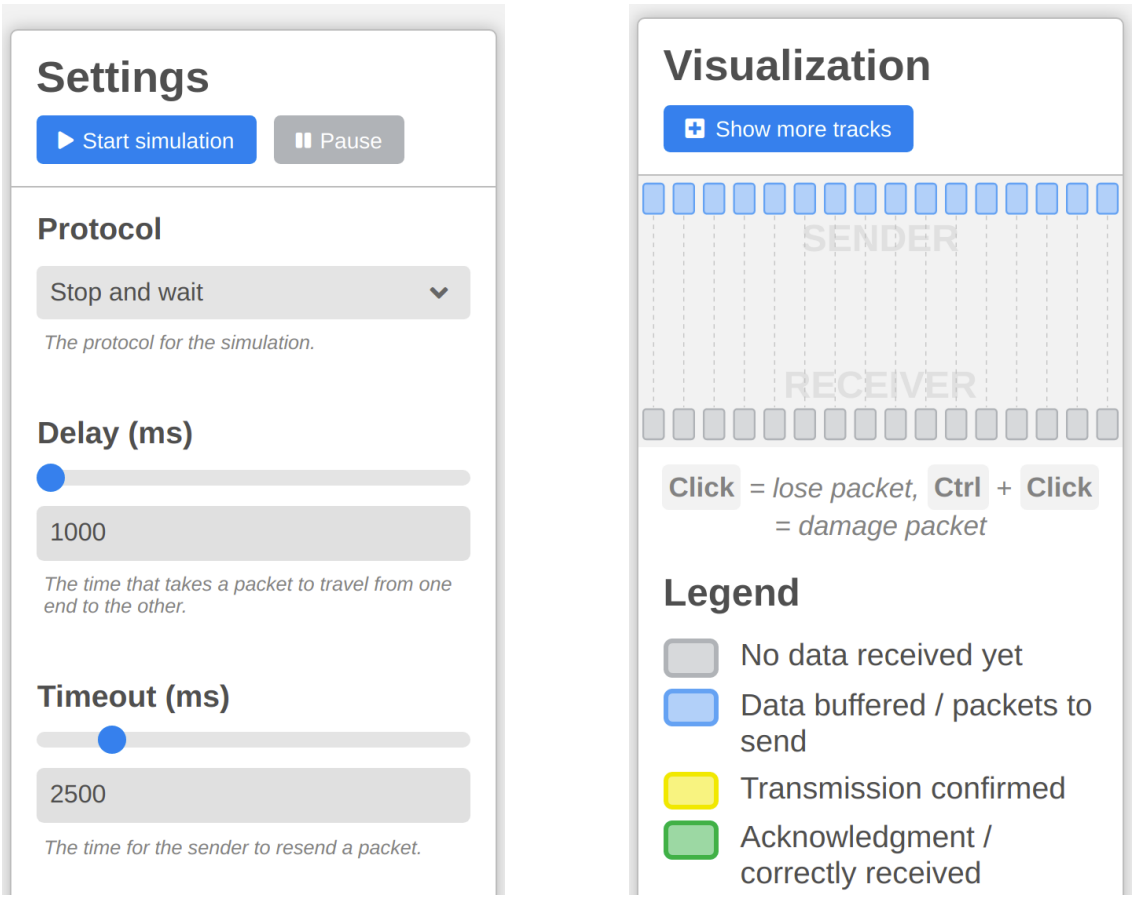

Figura 8: Porciones de las secciones de configuración y visualización, en una pantalla de 320 por 568 píxeles.

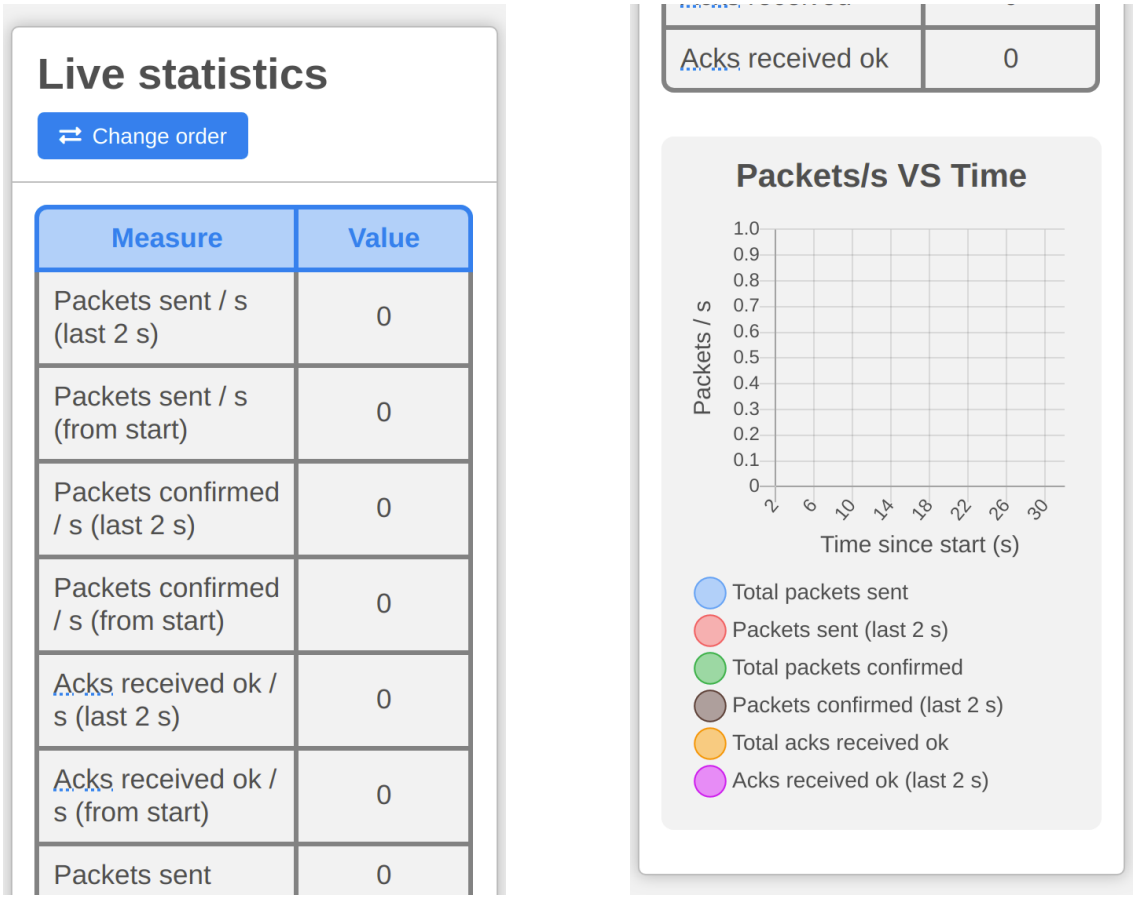

Figura 9: Porciones de la sección de estadísticas, en una pantalla de 320 por 568 píxeles.

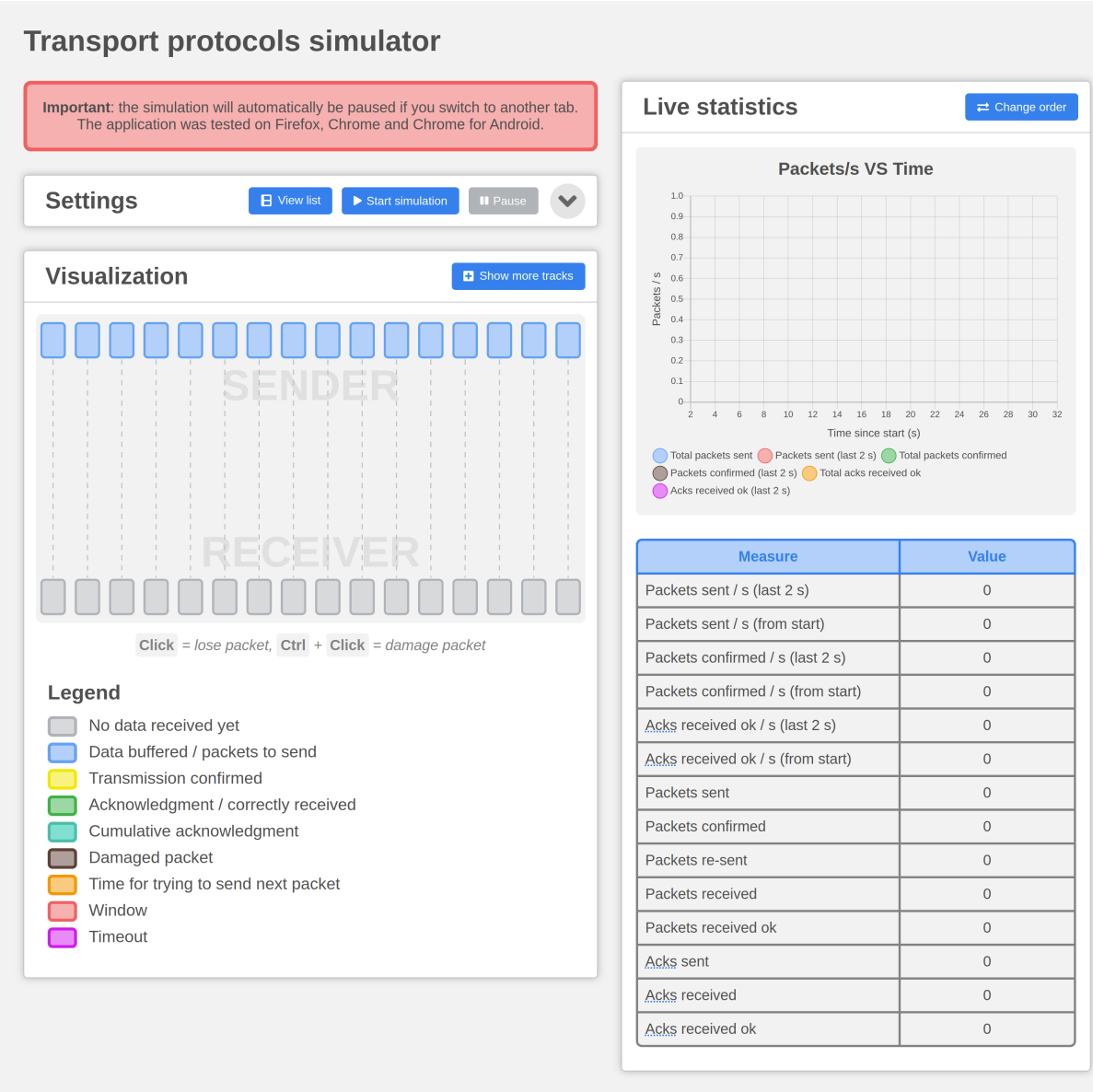

Figura 10: Interfaz real dispuesta en forma de columnas.

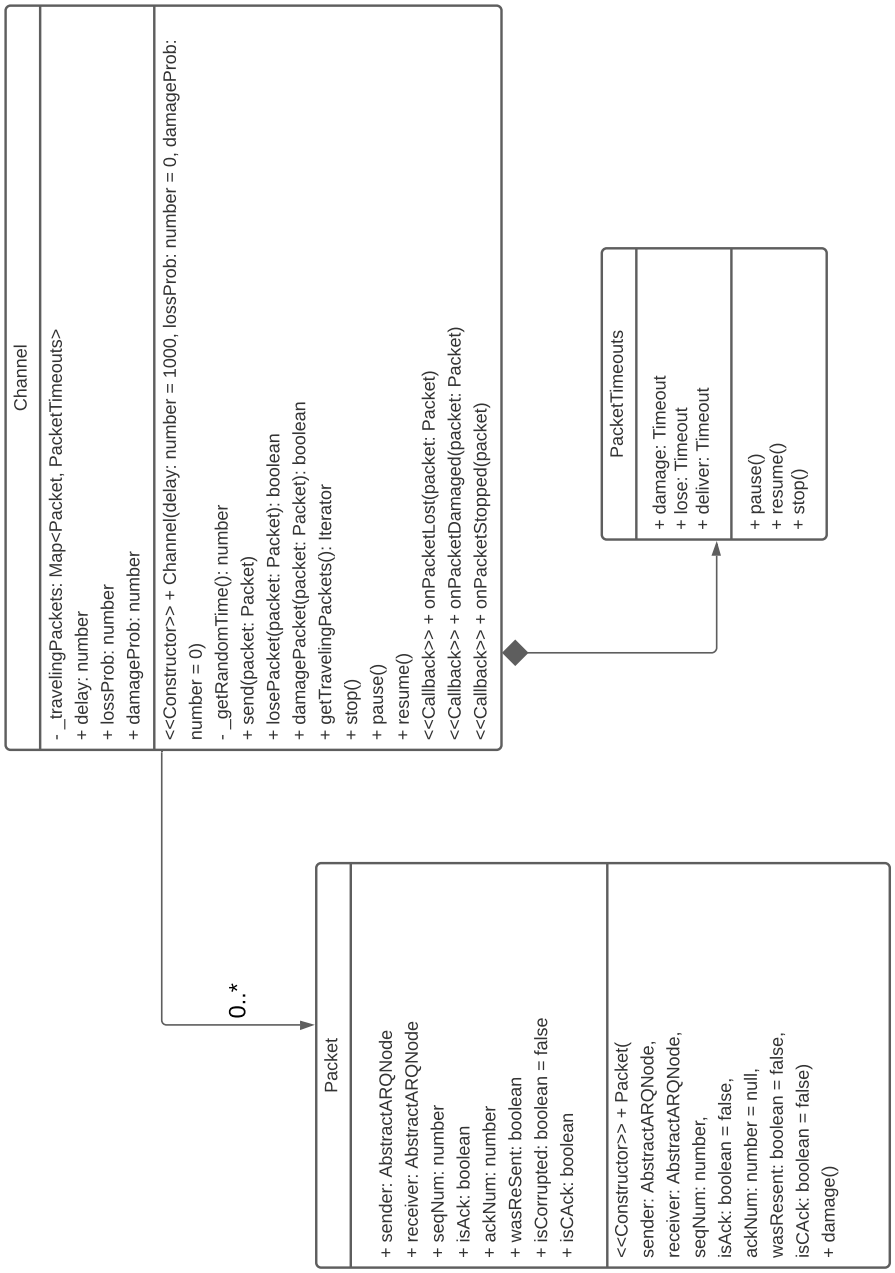

Figura 11: Clases Channel y Packet.

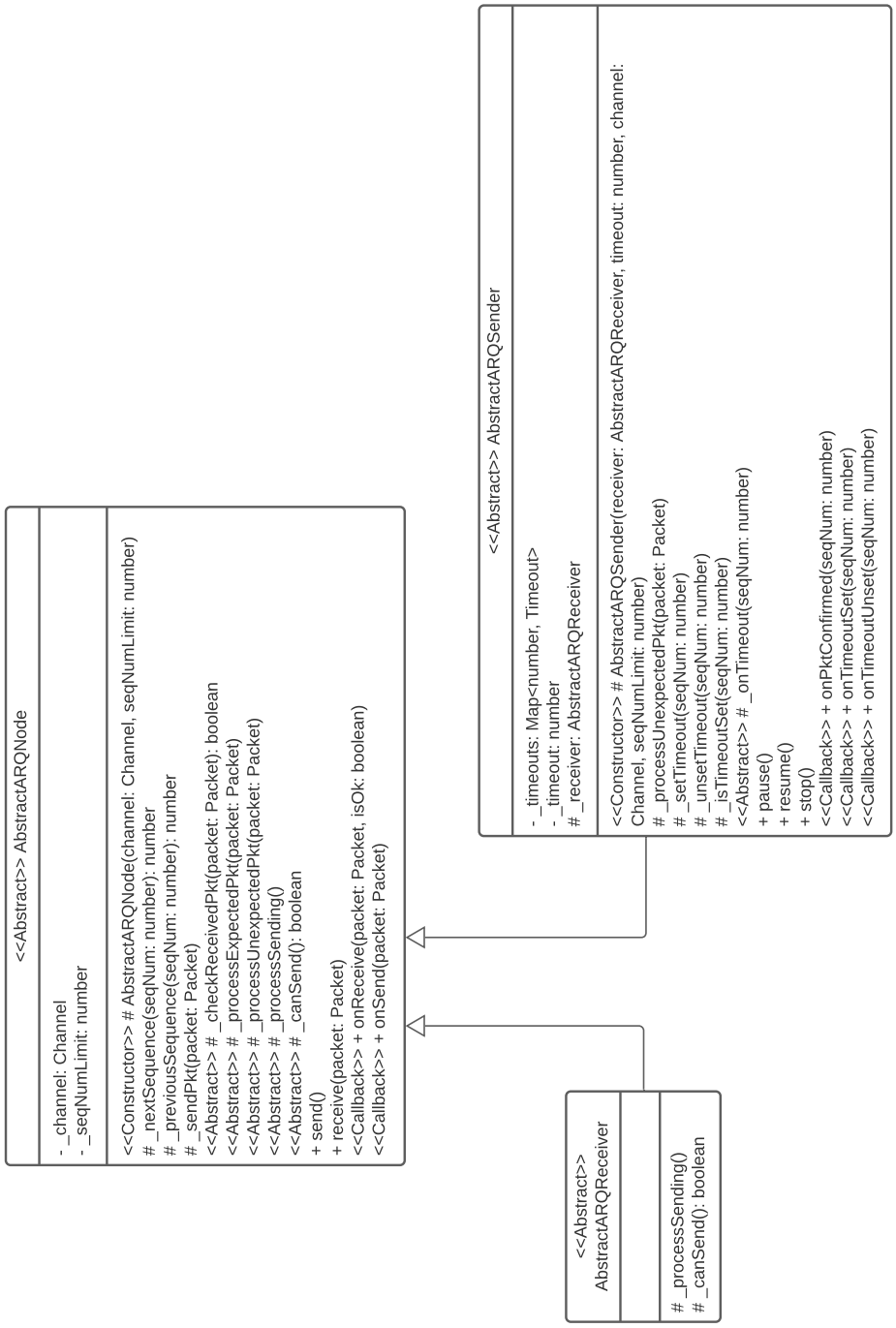

Figura 12: Clases AbstractARQNode, AbstractARQSender y AbstractARQReceiver.

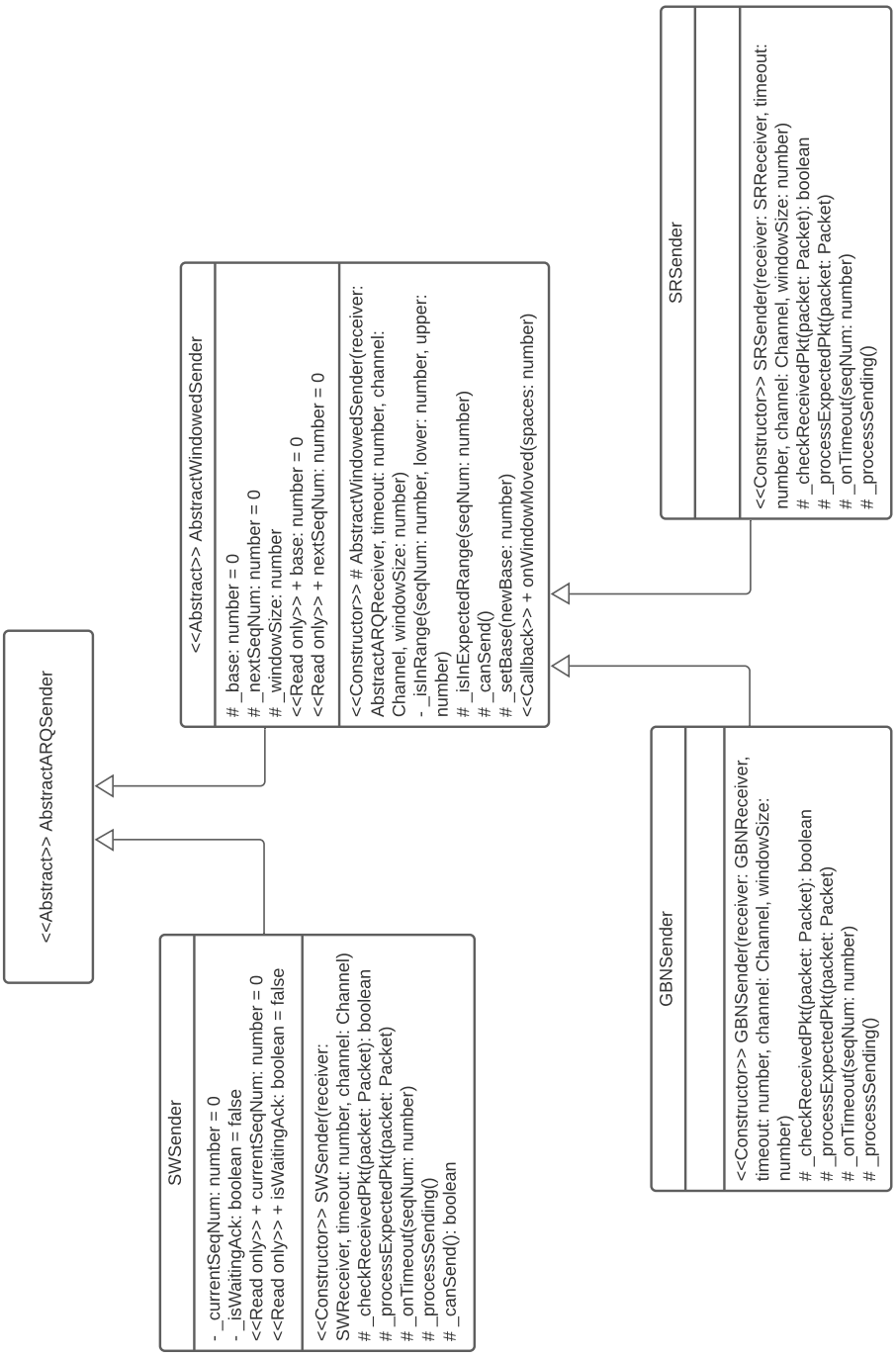

Figura 13: Clases que extienden a AbstractARQSender.

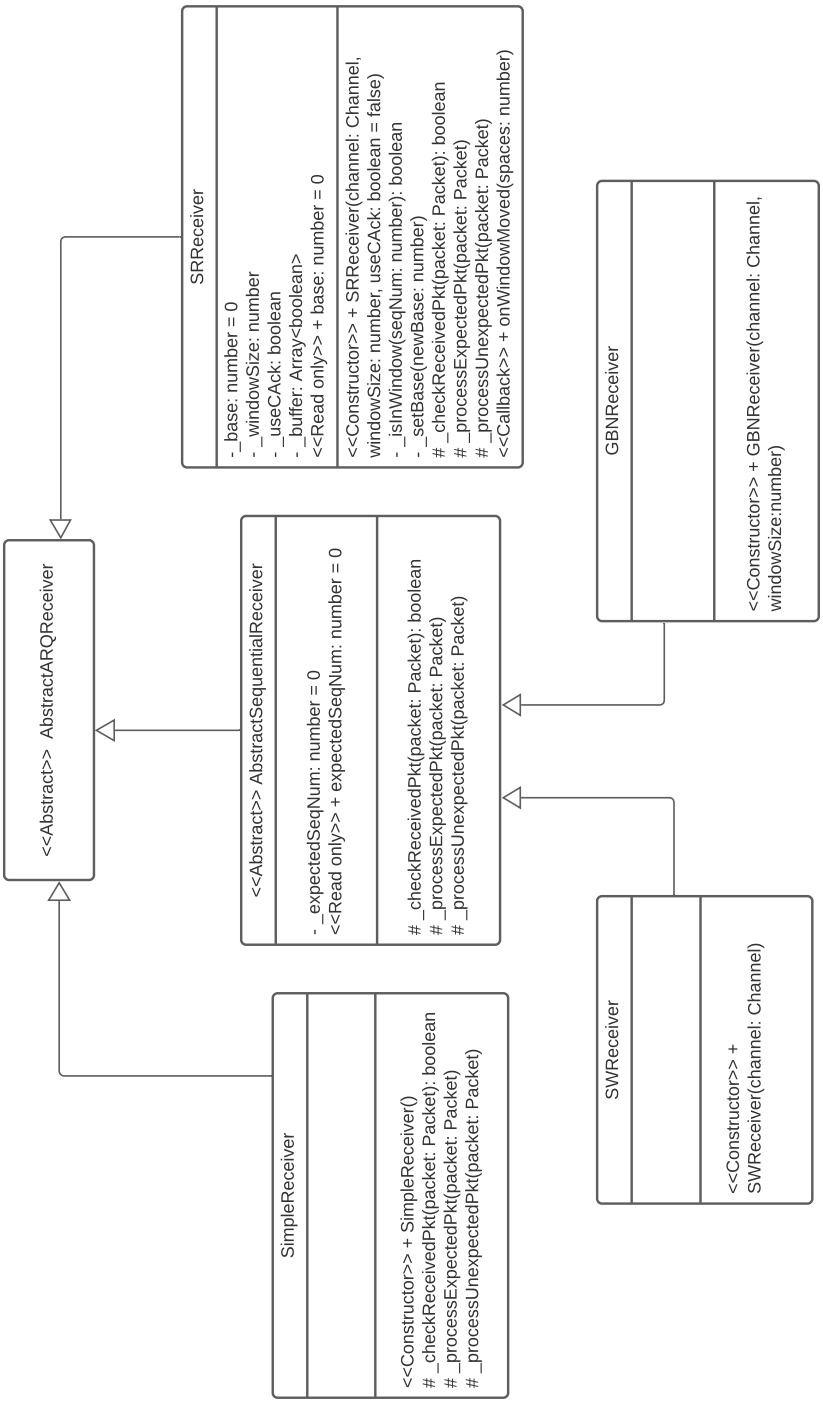

Figura 14: Clases que extienden a AnstractARQReceiver.

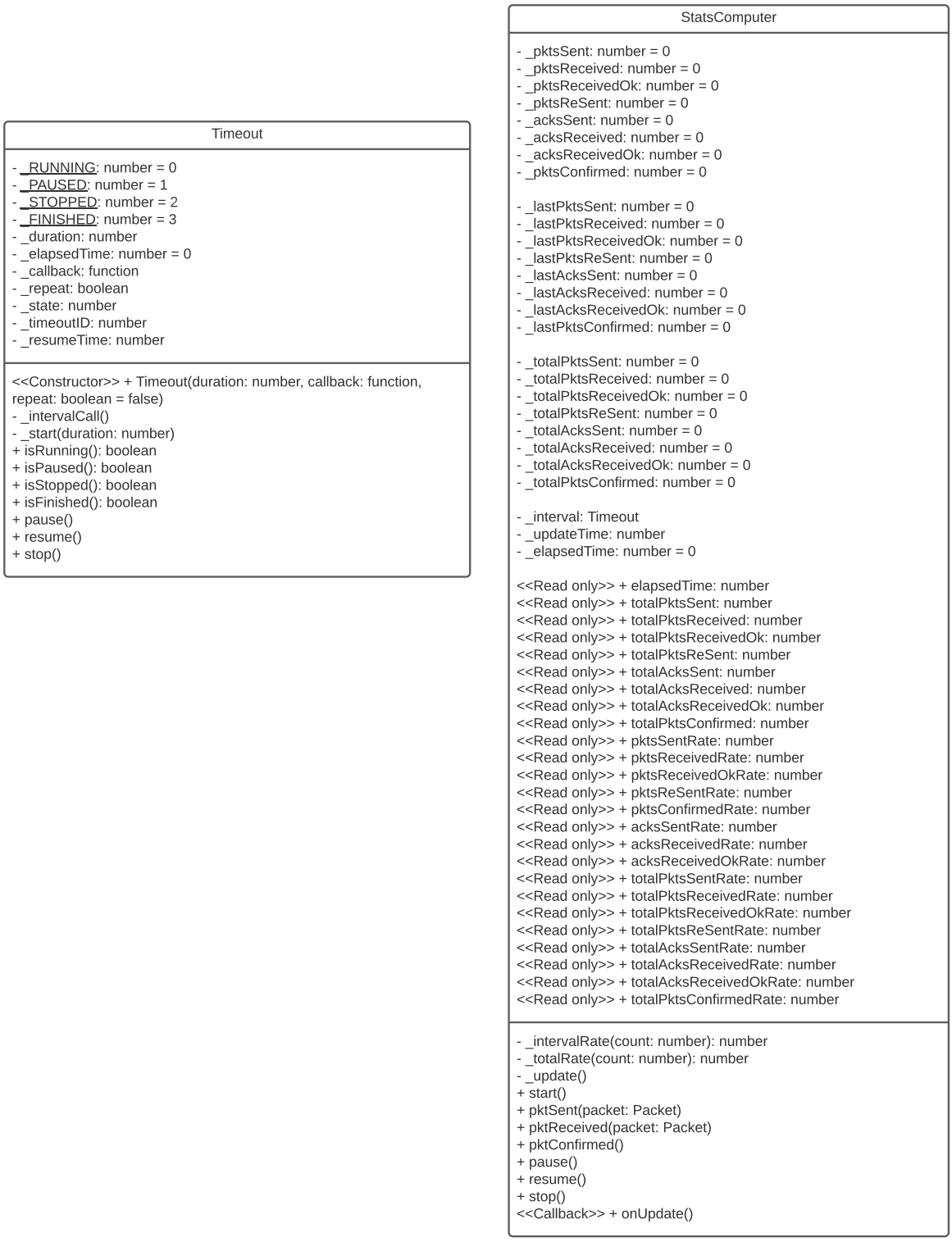

Figura 15: Clases Timeout y StatsComputer.

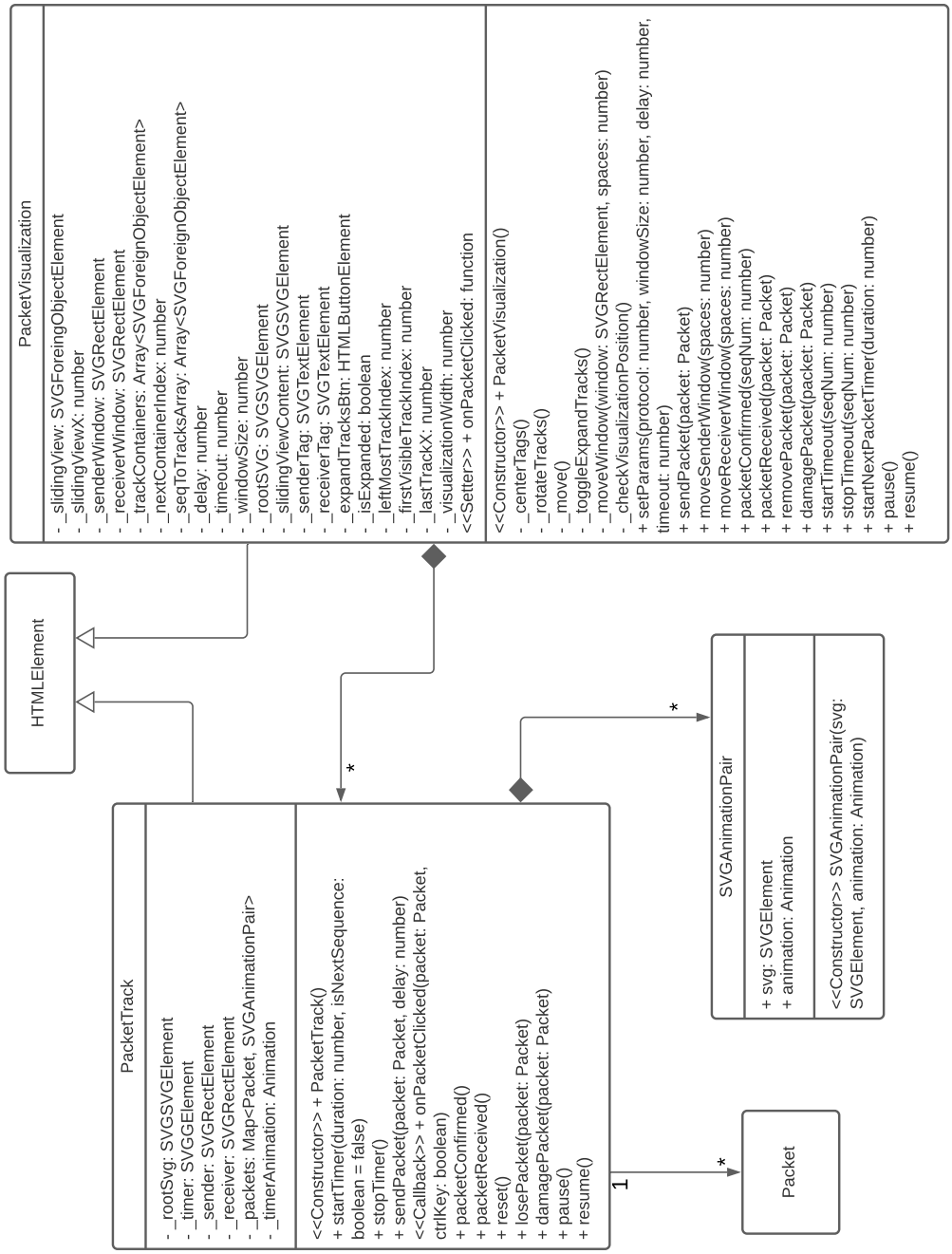

Figura 16: Clases de los componentes PacketTrack y PacketVisualization.

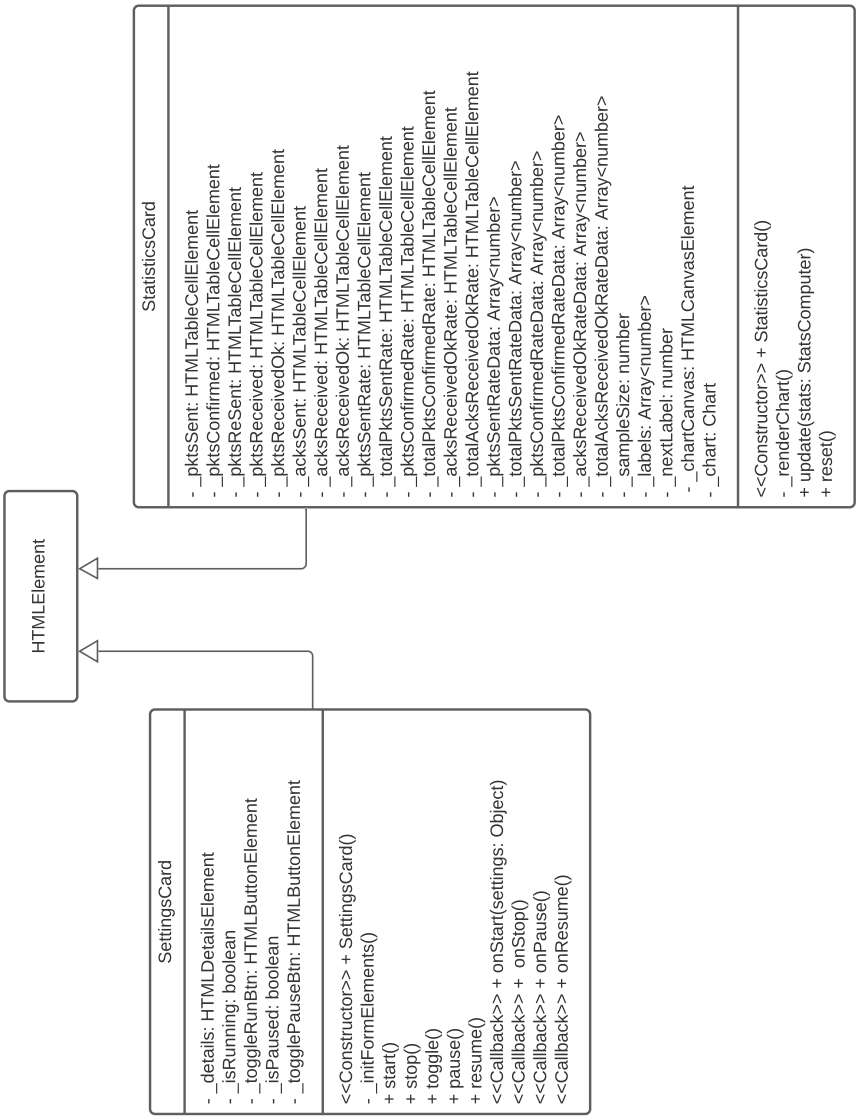

Figura 17: Clases de los componentes StatisticsCard y SettingsCard.