UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SIMULACIÓN DE SISTEMAS DE FEDERATED LEARNING EN REDES MÓVILES 5G

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

JOSÉ OMAR ANTONIO PAVEZ COLLADO

PROFESOR GUÍA:
ALBERTO CASTRO ROJAS

MIEMBROS DE LA COMISIÓN:
CESAR AZURDIA MEZA
JORGE SANDOVAL ARENAS

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: **JOSÉ OMAR ANTONIO PAVEZ COLLADO**
FECHA: 2023
PROF. GUÍA: ALBERTO CASTRO ROJAS

SIMULACIÓN DE SISTEMAS DE FEDERATED LEARNING EN REDES MÓVILES 5G

El aprendizaje federado permite el entrenamiento colaborativo de modelos de *machine learning* entre una gran cantidad de dispositivos sin la necesidad de compartir sus datos a un servidor central. Esto se puede mediante el entrenamiento local de modelos en cada cliente, los cuales se envían a un servidor para ser integrados a un modelo global. Este método de aprendizaje se utiliza para aplicaciones en los que se requiere mantener la privacidad de los datos, por ejemplo al entrenar modelos con datos personales del *smartphone* de un usuario. También se utiliza cuando se tiene una gran cantidad de datos para realizar un entrenamiento eficiente utilizando la capacidad computacional de los mismos clientes. Por otro lado, las redes 5G permiten la comunicación en tiempo real con un gran ancho de banda, lo que tiene diversas aplicaciones en internet de las cosas (IoT), inteligencia artificial, etc.

Sin embargo, el diseño, la implementación y el desarrollo de pruebas de estos sistemas posee grandes desafíos. Debido a que se necesita una gran cantidad de clientes para entrenar estos modelos, hacer pruebas con dispositivos físicos en la etapa de prototipado para diseñar una red puede ser costoso, lento y hasta inviable.

En este trabajo se propone la simulación de sistemas de aprendizaje federado considerando dos aspectos de simulación principales. En primer lugar, una simulación de red 5G desde la capa física en adelante. Y en segundo lugar, una simulación del entrenamiento de los modelos de aprendizaje federado utilizando librerías de *machine learning*, como si fuesen entrenados en esta red simulada. Con respecto a los resultados obtenidos, se establece un clasificador de imágenes para el CIFAR-10 el cual se quiere entrenar y se ponen a prueba tres casos distintos de red a simular. Se muestra que la inestabilidad y perdida de paquetes tiene un mayor efecto negativo en el rendimiento de la red que la congestión.

A mi familia y seres queridos.

Agradecimientos

En primer lugar, quiero agradecer a mi familia por apoyarme durante todo este tiempo en la universidad, a mi madre Elizabeth y a mi padre Gonzalo, que siempre me apoyaron para estudiar lo que quería a pesar de todas las dificultades, por su cariño, amor y por todo lo que me han entregado durante la vida para poder desarrollarme como persona. Sin ellos no hubiese podido lograr lo que he realizado hasta el día de hoy. A mis hermanos, Constanza y Nikola, por darme alegría en los momentos difíciles. A mis abuelos, José y Angélica, por recibirme con los brazos abiertos cuando llegué a Santiago y por todo su apoyo.

También agradecer a todos los profesores de la facultad que me enseñaron a lo largo de mi formación por su tiempo y dedicación. En especial a mi profesor guía, Alberto Castro, por ayudarme a la realización de este trabajo y por todos sus consejos en todas nuestras reuniones. Finalmente, agradecer a los profesores de la comisión, Cesar Azurdia y Jorge Sandoval, por aceptar ser parte de la comisión y darse el tiempo de revisar y corregir mi trabajo.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.0.1. Antecedentes

A diferencia de los métodos tradicionales de *machine learning*, en donde el proceso de entrenamiento se concentra en una sola máquina central, el *federated learning* aborda el entrenamiento desde un enfoque distinto, en donde, una multitud de dispositivos entrenan un modelo de manera colaborativa y decentralizada, utilizando sus datos locales sin la necesidad de compartir datos entre dispositivos. El *federated learning* tiene una gran cantidad de aplicaciones, por ejemplo, desde entrenar modelos con datos en celulares, pero respetando la privacidad de sus usuarios, hasta entrenar modelos con datos adquiridos en una masiva cantidad de sensores en donde no sería viable enviar todos esos datos a un servidor central.

1.0.2. Planteamiento del problema

Pero a pesar de los beneficios del *federated learning* en distintas aplicaciones, su propia naturaleza descentralizada hace difícil y costoso realizar pruebas en el sistema utilizando *hardware* real debido a la gran cantidad de dispositivos involucrados. Esta limitante hace que desarrollar sistemas de *federated learning* puede ser un proceso más lento o hasta inviable, comparado con los métodos de *machine learning* tradicional. Para resolver este problema, es necesario un entorno de simulación virtual, el cual permita hacer pruebas del sistema en la etapa de desarrollo, antes de que se despliegue en la realidad. Sin embargo, no existe una solución diseñada para simular con fidelidad todas las capas de red que componen los sistemas de *federated learning*, en especial, hay una deficiencia para modelos de *reinforcement learning* ya que, si bien existen simuladores de redes generales, estos no cubren las necesidades particulares de los sistemas descritos anteriormente.

1.1. Objetivos

1.1.1. Objetivo General

El objetivo central del trabajo es desarrollar y optimizar sistemas de *federated learning* en redes móviles 5G mediante un entorno de simulación.

1.1.2. Objetivos Específicos

Para completar el objetivo general propuesto, los objetivos específicos a realizar son:

- Diseñar e implementar un simulador de redes móviles 5G
- Diseñar e implementar un simulador de modelos FL
- Integrar ambos simuladores a través de un protocolo de comunicación
- Evaluar la arquitectura propuesta en un escenario realista y entrenar un modelo FL

Capítulo 2

Marco Teórico y Estado del Arte

2.1. Marco teórico

2.1.1. Definiciones

A continuación se presentan ciertas definiciones necesarias para la lectura de esta memoria.

Definición 2.1 *El federated machine learning (FML) es un sistema que permite a múltiples participantes entrenar un modelo de machine learning sin tener que compartir los datos que le pertenecen a los participantes. Este concepto se explica con mayor detalle en la subsección siguiente.*

Definición 2.2 *Un modelo FML es el resultado del entrenamiento en un sistema FML . El modelo entrenado puede ser utilizado para realizar tareas de inferencia en datos que no han sido observados previamente, por ejemplo, tareas de clasificación, predicción, recomendación, etc.*

Definición 2.3 *El coordinador o 'aggregator' es el servidor central que construye el modelo FML a partir de combinar los sub-modelos entrenados por los distintos colaboradores.*

Definición 2.4 *El auditor es el usuario responsable de comprobar el funcionamiento del modelo FML y evaluar su rendimiento para verificar que pueda ser utilizado por otros usuarios.*

Definición 2.5 *Una característica o 'feature' es un subconjunto de propiedades medibles en un conjunto de datos.*

Definición 2.6 *Un 'dataset' es una colección de datos, los cuales consisten en una lista de features, etiquetas y un identificador único (ID)*

Definición 2.7 *El dataset de entrenamiento es el dataset utilizado para entrenar el modelo FML . En un sistema FML este dataset esta distribuido entre los distintos colaboradores.*

Definición 2.8 *El dataset de validación corresponde al dataset que se usa para evaluar el rendimiento de un modelo FML ya entrenado.*

Definición 2.9 *La precisión se define como el porcentaje de resultados correctos(RC) de una cantidad de resultados totales (RT),* $\frac{RC}{RT}$ *, ya sean verdaderos positivos o negativos.*

Definición 2.10 *Dada una muestra de resultados, con una cantidad de verdaderos positivos VP y una de verdaderos negativos VN, se define el 'recall' como* $\frac{VP}{VP+VN}$

Definición 2.11 *Una llamada a procedimiento remoto (en inglés, Remote Procedure Call, RPC) es cuando un programa computacional ejecuta un procedimiento en un espacio de direcciones distinto al del programa (usualmente en otro computador a través de la red) de tal manera que parezca ejecutado localmente.*

2.1.2. Modelos de Machine Learning

El concepto de modelo se puede definir como una manera de representar una función con múltiples entradas, la cual produce una o múltiples salidas. Esta función se modifica a través de un proceso de aprendizaje, la cual modifica sus parámetros basándose en una métrica medible, ya sea el error cuadrático medio, entropía cruzada, etc. Después de su etapa de entrenamiento, se espera que el modelo sea capaz de inferir cual es la salida correcta a datos de entrada que no han sido presentados previamente. Este modelo puede tener distintas representaciones ya sean redes neuronales, arboles de decisión, cadenas de Markov, etc.

2.1.2.1. Tipos de aprendizaje

Existen distintos métodos para abordar la etapa de aprendizaje en un modelo. A veces, la forma de aprendizaje esta relacionada estrechamente como el tipo de modelo. Además, la forma en que los datos de entrenamiento estén disponibles puede hacer más ventajosa o desventajosa cierta manera de aprender. Los métodos más utilizados son los siguientes[1]:

- **Aprendizaje supervisado:** en este tipo de aprendizaje, se tiene un dataset con información de las entradas y las salidas correspondientes. Aquí el modelo intenta minimizar el error entre las salidas del dataset (*ground truth*) y las generadas por el modelo. Se utiliza en general para tareas de clasificación y regresión.

- **Aprendizaje no supervisado:** en este caso solo se cuenta con los datos de entrada, pero no se tiene información a priori sobre la salida. Problemas de este tipo son el *clustering*, detección de anomalías y reducción de dimensionalidad.

- **Aprendizaje semi-supervisado:** es una combinación del aprendizaje supervisado y no supervisado, ocurre cuando el dataset de entrenamiento tiene solo algunas etiquetas de salida. Usualmente, se aborda utilizando alguna heurística para etiquetar las salidas faltantes.

- **Transfer learning:** esta técnica se utiliza cuando se tiene un modelo entrenado para cierto dominio y se quiere adaptar para solucionar un problema en otro dominio similar. Por ejemplo pasar de un modelo entrenado en una simulación a uno que se aplique a la realidad.

- **Reinforcement learning:** el *reinforcement learning*[2] es un método de aprendizaje en el cual un agente, controlado por un modelo, interactúa con un entorno. A este agente se le premia o castiga si ocurre algo deseado o indeseado a través de una función que asigne valores a los distintos resultados. La búsqueda y selección de modelos óptimos puede ser estocástica (Monte Carlo) o utilizando algoritmos genéticos. Difiere en el aprendizaje supervisado en que este no intenta regular las acciones del agente directamente, sino que las regula basándose en los resultados que resultan de estas acciones

2.1.2.2. Arquitecturas para el aprendizaje

El proceso de aprendizaje se puede abordar de distintas maneras. Aquí, se divide entre dos aspectos de un proceso de entrenamiento, el aspecto temporal y el espacial. En el aspecto temporal se pueden encontrar dos formas de entrenar:

- **Aprendizaje offline o batch:** en este caso el modelo es entrenado con un dataset elegido previamente; una vez entrenado el modelo es desplegado para su uso. Una vez es desplegado el modelo se mantiene estático y no cambia a lo largo del tiempo. Para mejorar el modelo se debe detener su uso, entrenar y volver a desplegarlo. Este tipo de aprendizaje se usa para modelos que no necesiten adaptarse a situaciones nuevas ya que el dominio es estático. Además, permite un mejor rendimiento en su uso puesto que solo tiene que inferir a partir del modelo ya creado.

- **Aprendizaje online:** se entrena el modelo inicial de la misma forma que en el modo offline, pero en este caso una vez desplegado el modelo, este sigue aprendiendo utilizando la nueva información que recibe, modificando el modelo inicial continuamente. Este tipo de aprendizaje se utiliza para sistemas dinámicos en donde los parámetros necesitan adaptarse constantemente debido a los cambios ambientales.

Otro aspecto a considerar es cómo se distribuye el aprendizaje, ya sea en un solo lugar o en varios; las opciones posibles son:

- **Aprendizaje centralizado:** la forma más utilizada en machine learning, el modelo se entrena en un solo dispositivo, útil cuando se tiene un gran poder computacional. Pero no se puede respetar la privacidad de los datos y no puede escalar horizontalmente cuando los datos llegan a los limites de la memoria y almacenamiento del servidor.

- **Aprendizaje distribuido:** el entrenamiento se distribuye entre varios dispositivos utilizando técnicas de procesamiento paralelo. Esto permite extender la capacidad computacional más allá de la capacidad de una sola máquina.

- **Aprendizaje federado:** el entrenamiento es colectivo igual que en el caso distribuido, pero en el aprendizaje federado los datos no se comparten entre los distintos nodos, sino que cada colaborador entrena su propio modelo para luego unirlo a un modelo final. Este tipo de aprendizaje se explicará con mayor detalle en la siguiente sección.

2.1.3. Aprendizaje Federado

Un concepto central en este trabajo es el de *federated learning* (FL) o aprendizaje federado. Es una técnica de computación distribuida y descentralizada para el entrenamiento de modelos de *machine learning*. Consiste en un grupo colectivo de dispositivos que funcionan de manera independiente. Estos clientes entrenan un modelo local utilizando su propia base de datos adquirida localmente. Este dataset puede ser, por ejemplo, un sensor adquiriendo datos de sus mediciones, o un smartphone adquiriendo datos de su usuario. Luego de realizar el entrenamiento, el modelo es enviado a un servidor central, el cual se encarga de combinar todos los aportes de los colaboradores en un modelo central. La combinación se hace a través de un algoritmo determinado (por ejemplo, *Federated Averaging* (FedAvg)).

Un *framework* de FL, como el que se puede ver en la Figura 2.1[3], consiste en usuarios, datos y modelos. Los datos, están separados de manera distribuida a través de los distintos nodos colaboradores. Estos datos son utilizados para entrenar sub-modelos FML , los cuales son integrados en un solo modelo FML de manera segura, manteniendo la privacidad.

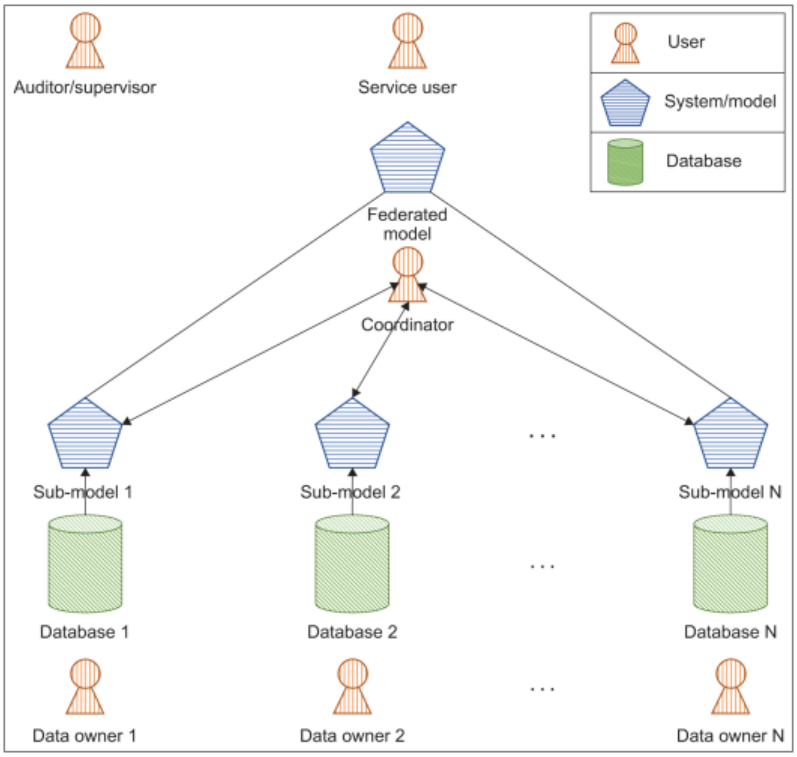

Figura 2.1: Diagrama de un framework de aprendizaje federado.

2.1.4. Promedio Federado (FedAvg)

Dentro del aprendizaje federado, existen distintos algoritmos para formar un modelo global central a partir de los modelos entrenados por cada cliente, en este sentido solo se considera el aprendizaje federado centralizado[4], que depende de un servidor para unir los modelos (otros esquemas descentralizados están fuera del enfoque de este trabajo). Uno de estos algoritmos es el promedio federado o FedAvg[5], este se deriva del *Federated Stochastic Gradient Descent* (FSGD). En FSGD, se promedia el gradiente de cada cliente para actualizar los *weights* del modelo, bajo la siguiente formula:

$$w_{t+1} = w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$$

Donde n_k es la cantidad de datos del cliente k, con una tasa de aprendizaje η, para K clientes, ahora como el modelo inicial es el mismo para todos los clientes, una manera equivalente de actualizar el modelo es:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$$

Esto es lo que se conoce como promedio federado, es decir el servidor central promedia los *weights* de cada modelo local al final de la ronda, este modelo local siendo actualizado por el cliente mismo, a diferencia de promediar las gradientes y que el servidor realice todo el proceso computacional.

2.1.5. Redes móviles 5G

La quinta generación de telefonía móvil es un nuevo estándar especificado por la 3GPP[6], el cual apunta a ser una plataforma que no solo otorga un mayor ancho de banda para los usuarios, sino que además permite el funcionamiento de nuevas tecnologías, como el internet de las cosas (IoT), edge computing y aplicaciones de tiempo real como la conducción autónoma[7]. La quinta generación es un avance respecto a la cuarta en varios aspectos[8]:

- **Enhanced Mobile Broadband (eMBB):** Se ofrecen mayores anchos de banda. Para el *downlink* hasta 50 Mbps en el exterior y 1 Gbps en interiores (5GLAN). Y la mitad de lo anterior para *uplink*. Otros casos de estudio en donde la tecnologia eMBB puede ayudar es por ejemplo en el caso de la aviación[9] para la comunicación tierra-aire.

- **Critical Communications (CC) y Ultra Reliable and Low Latency Communications (URLLC):** Para ciertas aplicaciones se necesita una fiabilidad extramadamente alta. Por ejemplo, para el control remoto de automatización de procesos[10] se espera una fiabilidad del 99,9999 %. Esto es posible a través de las capacidades del *Edge computing*.

- **Massive Internet of Things (mIoT):** Los sistemas 5G pueden funcionar en situaciones de alto trafico con una gran cantidad de dispositivos lo que lo hace apto para el internet de las cosas y compatibles con un amplio espectro de aplicaciones y dispositivos IoT.

2.2. Estado del arte

Dentro de la literatura actual podemos identificar distintos entornos de simulación, cada uno se focaliza en ciertos aspectos de una simulación dependiendo el uso para el cual esté destinado.

2.2.1. GNS3

GNS3[11] es un simulador de redes que permite crear topologías de red a través de una interfaz gráfica; permite correr nodos con máquinas virtuales usando desde *firmware* de *routers* hasta imágenes de Linux. En la Figura 2.2 se puede ver un ejemplo de simulación. Esto permitiría, potencialmente, simular el servidor y los clientes para correr cualquier modelo de aprendizaje federado. Sin embargo, una de las principales desventajas de este simulador es que no cuenta con simulaciones de redes inalámbricas en el contexto 5G de esta memoria. Además, el entorno de simulación debe cargar una máquina virtual por cada nodo, considerando que cada nodo debe correr un modelo *machine learning* esta manera de simular se vuelve ineficiente y demanda muchos recursos lo que limita la cantidad de nodos que se puedan añadir.

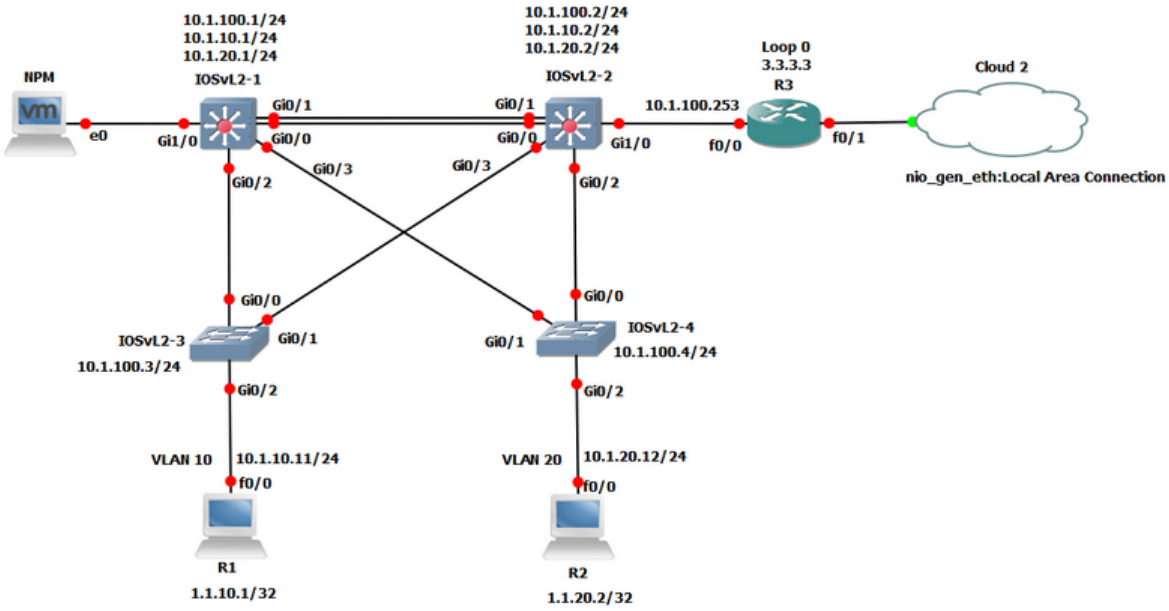

Figura 2.2: Ejemplo de una topología de red en GNS3.

2.2.2. TensorFlow

Tensorflow es una de las librerías para Python de *machine learning* más populares. Esta cuenta con un *framework* llamado 'TensorFlow Federated'[12][13] creado específicamente para el desarrollo y la investigación de sistemas de aprendizaje federado. Las simulaciones se hacen exclusivamente desde la perspectiva del modelo y cómo evoluciona el aprendizaje en cada ronda. Sin embargo, esto no considera ni el aspecto de red ni mucho menos lo que es la capa física, por lo que para una simulación correcta es necesario añadir una red virtual que pueda emular todo lo que ocurre bajo la capa de aplicación.

2.2.3. OpenFL

Framework de aprendizaje federado[14] desarrollado por Intel. Este permite utilizar cualquier librería de *machine learning* (TensorFlow, PyTorch, etc.) y crear experimentos con distintos flujos de trabajos dependiendo de las necesidades del desarrollador. Específicamente, el esquema de cómo funcionan se puede ver en la Figura 2.3. Se definen dos tipos de flujos de trabajo, basado en un director, en el cual los nodos se mantienen en ejecución, para correr una serie de experimentos asignados por un director, lo que es útil en etapas de prueba iniciales, y basado en un agregador, en donde el experimento se define a priori y los nodos se terminan al terminar el experimento. Al igual que *Tensorflow Federated*, las simulaciones solo se hacen a nivel de los modelos, dejando de lado simulaciones de capa física o de red.

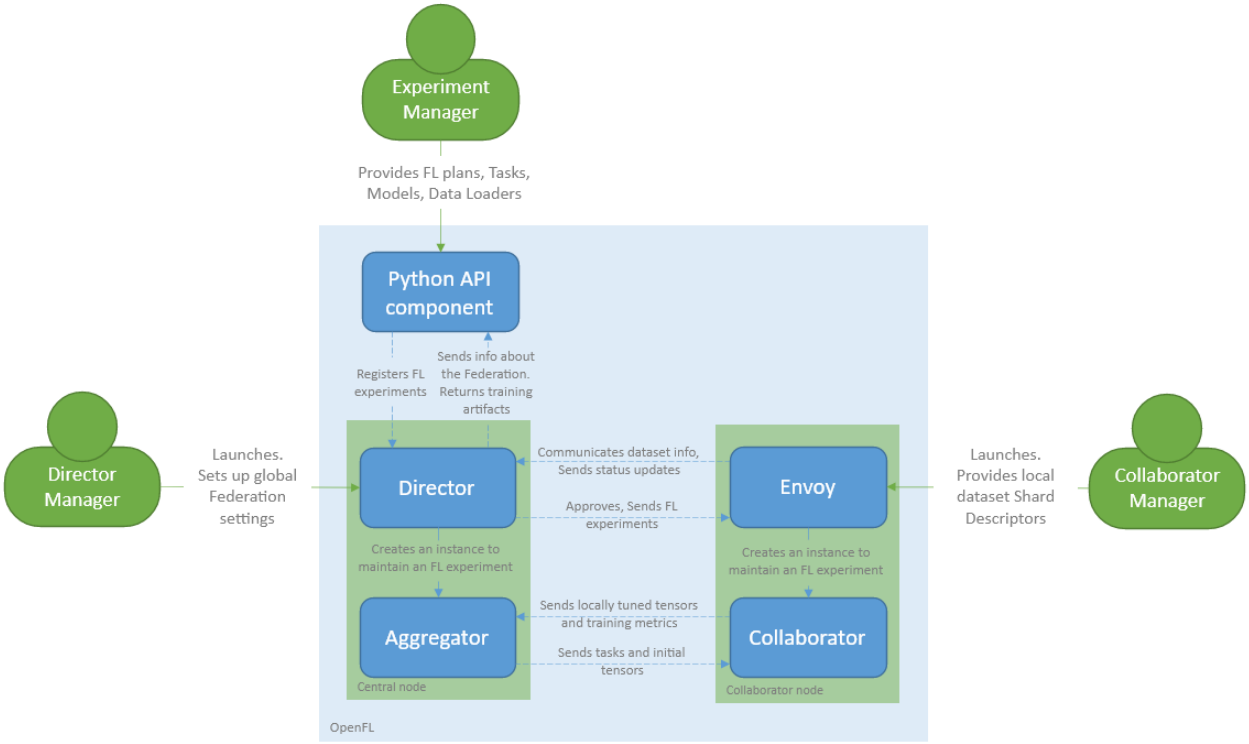

Figura 2.3: Esquema de la estructura de OpenFL.

2.2.4. FedEdge

FedEdge[15] es un entorno experimental para redes multi-hop inalámbricas de *edge computing FL*. Un esquema de sus componentes se puede ver en la Figura 2.5. Utiliza un modelo 'Multi-Agent Reinforcement Learning' (MA-RL) para encontrar el enrutamiento óptimo en la red inalámbrica entre los dispositivos. Además, cuenta con un simulador[16] de redes inalámbricas, lo que permite crear una simulación correcta desde la capa física. Para esto utiliza el driver mac80211_hwsim que permite crear una interfaz de radio virtual, la que luego es controlada por el módulo *netlink*. En la Figura 2.4 se puede observar un diagrama de sus componentes, en donde se puede ver como interactúan los drivers en el kernel con las aplicaciones del *user-space*. Además, se utilizan modelos físicos para la propagación de ondas y la interferencia, modificando la señal a partir del signal-to-noise ratio (SNR) calculado. Sin embargo, no existe una implementación de FedEdge disponible de manera pública, por lo que será necesario desarrollar una implementación, al menos, del simulador. Además, la aplicación que se dio a este sistema es específica por lo que será necesario generalizarlo y adaptarlo a nuestras necesidades.

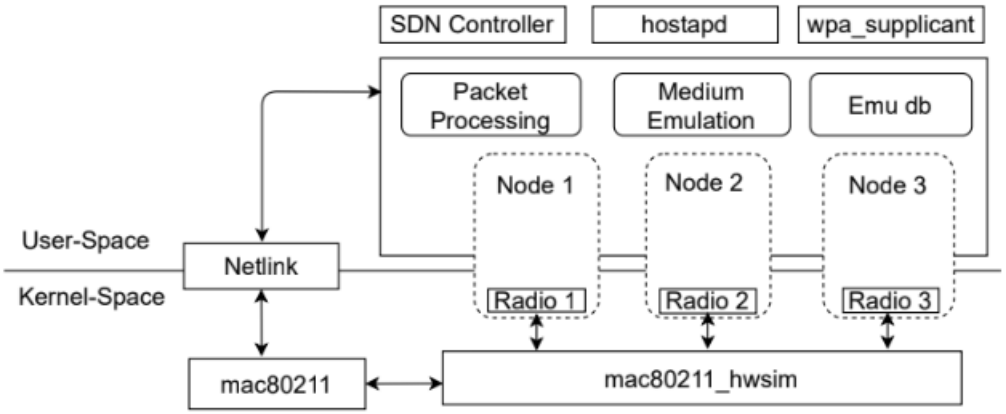

Figura 2.4: Diagrama de los componentes del simulador de FedEdge dentro del sistema operativo.

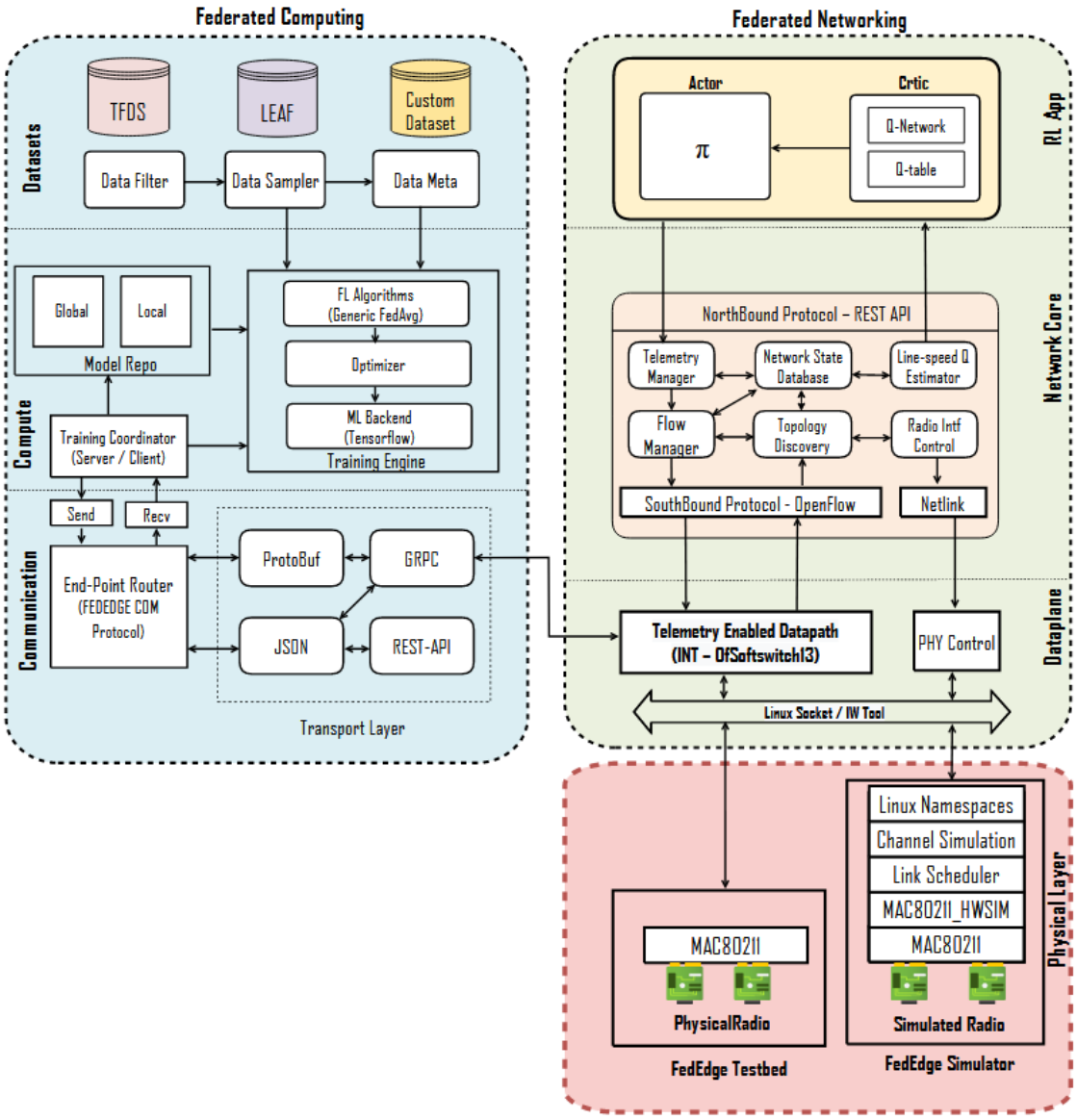

Figura 2.5: Diagrama completo del entorno FedEdge.

2.2.5. NS3-fl

NS3-fl[17] es un simulador de aprendizaje federado. Este toma en consideración desde la simulación de red, hasta los algoritmos y datos. Se compone de dos simuladores: NS3 para la simulación de red y *flsim* para la ejecución de los modelos. Además, simula el tiempo computacional, poder y energía utilizados basándose en mediciones experimentales. Si bien, este modelo es más completo que los anteriores con lo que respecta a las distintas capas de simulación, el simulador solo permite utilizar redes Wi-Fi o Ethernet. Por lo que no nos sirve en el contexto de redes móviles. Además, requiere necesariamente utilizar el *framework* de flsim para los modelos, por lo que no es posible utilizar otras librerías de interés como Tensorflow.

Capítulo 3

Metodología

A continuación, se presenta cual debiera ser la metodología utilizada para el desarrollo de la memoria. Primero, se explica el funcionamiento de un modelo de *federated learning*, mostrando los componentes y procesos relevantes para el proceso de aprendizaje de modelos de federated learning. De este funcionamiento se deriva cómo se estructura el simulador.

Luego, se muestra un esquema general de el entorno de simulación dando una vista conceptual de sus componentes. Como se verá a continuación, el simulador, principalmente, consiste en dos partes, la primera es la simulación de red y la segunda es la ejecución de modelos FL. Una vez definido el simulador, se procede a detallar el simulador de red, explicando los módulos que lo componen y una documentación del software desarrollado. Después de esto, se pasa a presentar el simulador de aplicaciones FL y se define un protocolo para la comunicación entre el simulador de red y la aplicación.

Finalmente, se prueba el entorno de simulación en distintas aplicaciones para ver el rendimiento de la red en diferentes situaciones. El objetivo es ser capaces de simular un entorno federado y entrenar modelos en el sistema de una manera que sea consistente con condiciones reales de operación.

3.1. Visión general del simulador

3.1.1. Proceso de entrenamiento FL

En primer lugar, se presenta el proceso de aprendizaje federado desde una perspectiva general, visualizado principalmente desde el punto de vista de la red y todos los nodos que la componen. Esta estructura sera crucial para el diseño del simulador puesto que permite entender el flujo de datos para el simulador de red. Este proceso aplica para todo algoritmo FL puesto que la arquitectura es independiente del modelo de *machine learning* utilizado por los dispositivos. En la Figura 3.1[18] se puede ver un diagrama del proceso de aprendizaje, los pasos del proceso son los siguientes:

- Se selecciona una cantidad de clientes para la ronda de aprendizaje, estos puede ser elegidos de manera aleatoria, elegir a todos los clientes, etc.
- Se envía el modelo global actual desde el servidor a los clientes.
- Los clientes, utilizando los datos que les pertencen, entrenan su propio sub-modelo a partir del modelo global recibido.
- Una vez entrenado, el cliente envía su modelo entrenado de vuelta al servidor
- El servidor integra todos los modelos recibidos en uno solo a partir de un algoritmo definido, por ejemplo, el promedio (FedAvg)
- Se actualiza el modelo global, listo para ser utilizado en una nueva ronda.

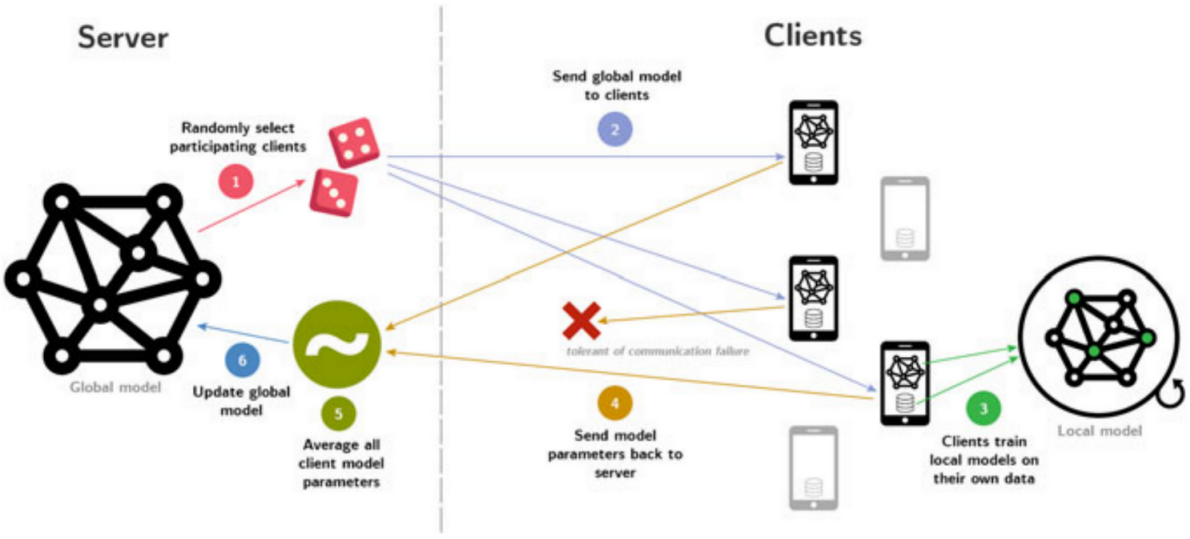

Figura 3.1: Diagrama del proceso de aprendizaje.

3.1.2. Estructura general del simulador

A continuación, se presenta una visión general de cómo esta estructurado el sistema completo de simulación que consiste de dos partes principales. La primera, es el modelo FL, este programa corre el modelo desde un punto de vista de la aplicación, ejecutando los algoritmos de aprendizaje sin considerar las simulaciones de red y la comunicación entre nodos, por lo que se podría considerar como una simulación puramente desde la capa de aplicación. Sin embargo, se comunica con el otro simulador para primero establecer la topología de red deseada a simular y, luego, para delegar las tareas de simulación de red, como se puede observar en la Figura 3.2. Utiliza Tensorflow como librería de machine learning y esta desarrollado en Python.

La segunda parte del entorno de simulación, consiste en el simulador de red. Este se basa en la librería de simulación de red NS3 y se desarrolla en C++. Se encarga de simular todas las capas excepto la de aplicación. Utilizando el módulo 5G-LENA para la simulación de redes móviles 5G, permite simular desde la propagación de onda hasta los sockets para la comunicación en un entorno FL. La comunicación entre ambas partes se realiza por RPC definidas que se ejecutan en el servidor NS3 a través de una conexión TCP. Además, es una comunicación síncrona. El modelo pide ejecutar el simulador y es necesario esperar por sus resultados. Como el tiempo que toma procesar la simulación puede ser bastante largo, en especial para simulaciones suficientemente grandes, la conexión no podría esperar a que termine la simulación para entregar los resultados ya que puede exceder el *timeout*, por lo que es el simulador el que debe enviar un nuevo mensaje una vez su tarea ha sido completada.

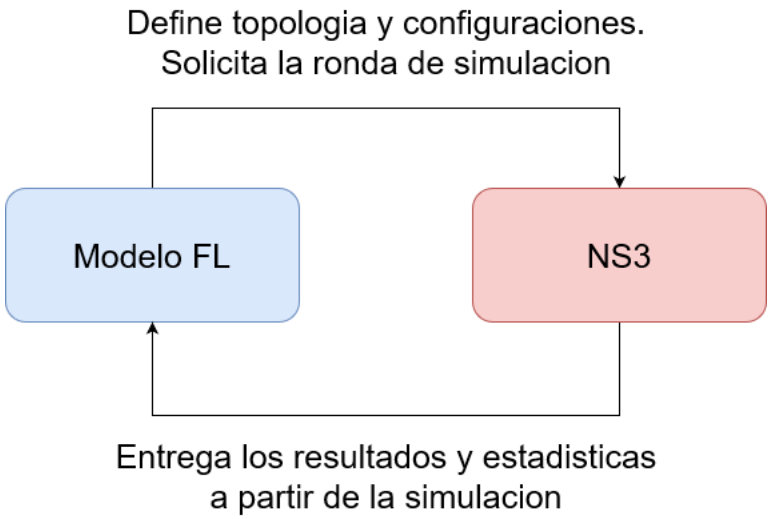

Figura 3.2: Esquema general del simulador.

3.1.3. Proceso de comunicación del simulador

Para solicitar las peticiones, el modelo FL se comunica con el simulador a través de un socket. En la Figura 3.3 se puede ver un esquema del proceso de comunicación. Primero, se inicia el proceso de NS3, luego se envían los datos de configuración de el entorno que se quiere simular. Una vez inicializado todos los parámetros del sistema, se envía un mensaje para iniciar la simulación. Desde este punto se hacen correr todas las rondas de entrenamiento establecidas hasta el final en donde se recopilan las estadísticas y datos entregados por la simulación. Luego, el simulador envía un mensaje de aviso al modelo para que reciba los resultados, en donde, finalmente, el modelo analiza los datos y da una orden de termino al proceso NS3.

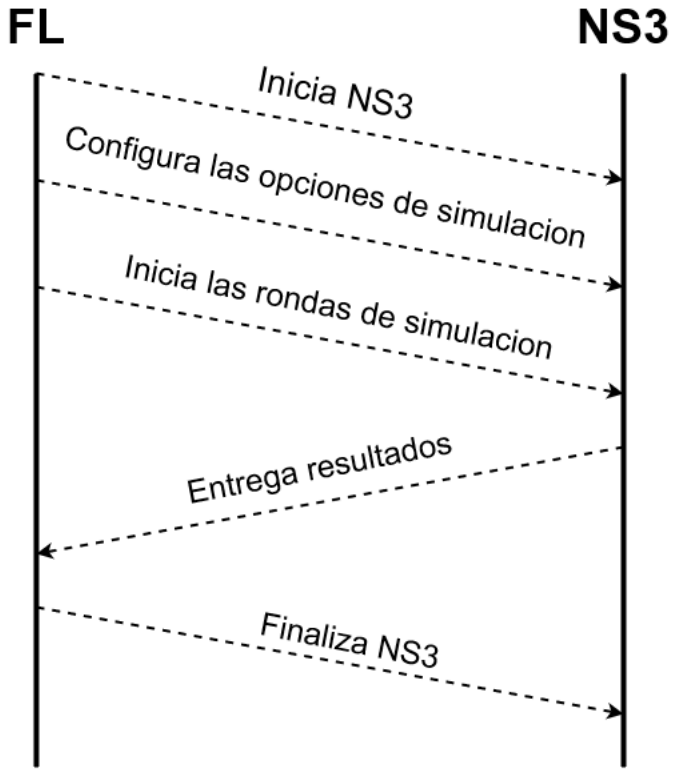

Figura 3.3: Esquema general del simulador.

3.1.4. Parámetros de configuración

La simulación puede ser configurada con distintos parámetros, lo que permite representar distintos escenarios, distintos modelos de propagación, configurar el modelo FL y probar topologías de red diferentes. Estos parámetros se almacenan en un archivo de configuración; este archivo debe estar en formato JSON. Al iniciar el programa, se lee el archivo y se ajustan las variables correspondientes en el simulador. Las opciones a configurar son las siguientes:

- **frequency:** Frecuencia de la portadora base.
- **bandwidth:** Ancho de banda disponible.
- **txpower:** Poder del transmisor de la estación base.
- **hbs:** Altura de la antena de la estación base.
- **hut:** Altura de la antena de los clientes.
- **segmentsize:** Tamaño de los segmentos TCP.
- **scenario:** Escenario de propagación de ondas.
- **nodes:** Numero de clientes.
- **radius:** Distancia radial entre clientes y la estación base.
- **modelsize:** Tamaño en bytes del modelo FL.
- **rounds:** Cantidad de rondas de entrenamiento.

Código 3.1: Ejemplo de un archivo de configuración JSON.

```
{
    "frequency": 28e9,
    "bandwidth": 100e6,
    "txpower": 40,
    "hbs": 25,
    "hut": 1.5,
    "segmentsize":1500,
    "scenario": "RMa",
    "nodes": 20,
    "radius": 5.0,
    "modelsize":100000,
    "rounds":3
}
```

3.2. Simulador de redes móviles 5G basado en NS3

En esta sección se detalla el funcionamiento del simulador de red basado en NS3 y todos sus componentes. Este simulador es el encargado de recrear una conexión móvil 5G de varios clientes con una estación base, simulando desde la capa física, usando modelos de propagación de onda, hasta la capa de red. La simulación se compone de la siguiente forma (Figura 3.4), la estación base (gNB) se ubica al centro, solo se considerara una estación (puede variar la cantidad de antenas), los clientes (UEs) están conectados a una distancia radial de la estación definida en la configuración. Esta distancia es igual para todos los usuarios, esto para ver mas claramente el efecto de la distancia en la red. Además, el servidor FL esta conectado a la estación directamente. En el contexto de este simulador, las transacciones ejecutadas por el modelo FL son básicamente conexiones TCP por donde se envían los modelos, ya sea desde el servidor a los clientes como del cliente al servidor para actualizar el modelo.

El simulador en si funciona como un servidor, recibiendo las peticiones de simulación del modelo FL. El funcionamiento de la simulación de red consiste en primer lugar inicializar todos los componentes necesarios utilizando los parámetros otorgados por el archivo de configuración. Para todas las simulaciones se considera una estación base y los clientes posicionados alrededor de esta a una distancia configurable. Luego se inician las rondas de simulación. Esta simulación corre de manera secuencial con el modelo FL, es decir, primero se ejecuta la simulación de red completa para luego pasar a el entrenamiento del modelo utilizando Tensorflow. Esto se hace para otorgar mayor flexibilidad a los modelos y ser capaces de utilizar otros modelos sin cambiar la interacción entre NS3 y los modelos FL.

Una vez se da inicio a las rondas de entrenamiento, el flujo de ejecución de la simulación esta basado en callbacks, esto es porque al ser NS3 un simulador basado en eventos si se fuera a realizar la ejecución de manera secuencial, se realizarían todos los envíos al mismo tiempo lo que no seria realista en absoluto y además estos envíos podrían ir en el orden incorrecto. El simulador debe saber la cantidad de bytes recibidos por TCP para poder verificar que el modelo fue recibido correctamente, y una vez se hayan enviado todos los modelos se puede continuar con la siguiente ronda, con un delay definido que se asume por concepto del tiempo de procesamiento que le toma a los computadores procesar y entrenar los modelos antes del reenviarlos. .Esto quiere decir que si existe un error en la transmisión de los modelos la simulación no va a poder continuar y se va a detener. Sin embargo, se considera que esto es preferible para poder detectar fallas y posibles errores en el diseño de la topología, ya que si el sistema falla silenciosamente existe el riesgo de que el problema pase desapercibido al usuario.

Si la simulación se realiza exitosamente, el programa guarda los registros de todos los envíos de paquete a través de la red móvil (archivos .pcap), además de otras estadísticas relacionadas al flujo de paquetes. Desde el punto de vista del simulador de red su tarea es solo recopilar y almacenar los datos, para luego ser analizados por la aplicación principal (Modelo FL), esto para utilizar la flexibilidad y el poder de Python y sus librerías para el procesamiento de datos.

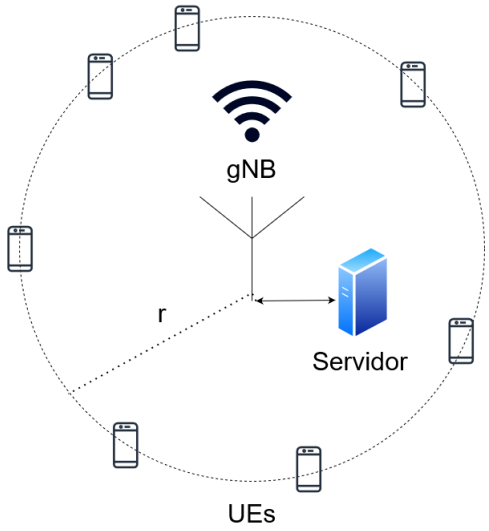

Figura 3.4: Topología de la simulación.

3.2.1. Componentes

A continuación se mencionan las librerías y componentes utilizados en el programa y como estos interactúan en la aplicación.

- **NS3:** NS3 es un simulador de red de eventos discreto, es el principal componente en el cual se construye nuestra simulación. Permite emular las conexiones de red y sus protocolos de transporte, además posee una gran cantidad de módulos para todo tipo de tecnologías de red. Permite crear aplicaciones que se ejecuten por encima del stack de red simulado.

- **5G-LENA:** Modulo de NS-3 diseñado para la simulación de redes móviles New Radio (NR) 5G[19], es la evolución del simulador mmWave para redes LTE/EPC. Implementa modelos de propagación de onda y los componentes de una red 5G como se describen en la especificación de 3GPP. En la Figura 3.5 se puede ver un esquema general de los componentes de LENA, con un host remoto conectado a una estación base (gNB) y esta con conexiones a dispositivos de los usuarios (UE),

- **EasySocket:** Librería para sockets en C++, compatible en Windows y Linux con un enfoque simple y minimalista (*single header*) para crear conexiones TCP de manera sencilla, permite la conexión entre el modelo FL y el simulador de red.

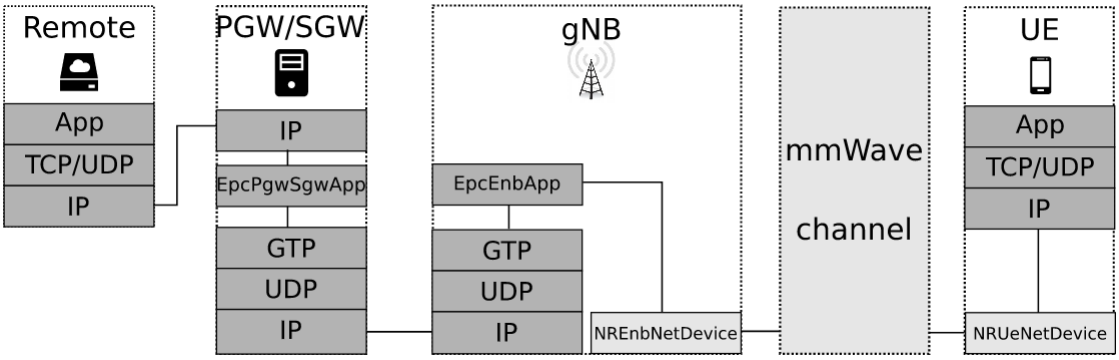

Figura 3.5: Diagrama de los componentes principales de LENA.

3.2.2. Estructura y clases del programa

El simulador de red se compone de distintos objetos que se encargan de diversas funciones en el programa, en esta sección se muestran todas las clases presentes con una descripción detallada de sus especificaciones.

3.2.2.1. FLServerApp

Clase encargada del funcionamiento del servidor, se hereda de la clase ns3::Application y se instala en el nodo de la estación base. Controla a los clientes y organiza las rondas de entrenamiento a través de callbacks. Los clientes que no logren transmitir dentro de un *timeout* determinado son deshabilitados. En la Figura 3.6 podemos ver un diagrama general del funcionamiento de la clase.

Métodos

- **Setup:** Inicializa la configuración del servidor y prepara la aplicación para ser iniciada
- **AddClientApp:** Asigna una aplicación a un cliente especifico.
- **BroadcastModel:** Envía el modelo FL global a todos los clientes disponibles.
- **RequestModels:** Método encargado de solicitar los modelos entrenados por los clientes.
- **RunRound:** Función que inicia una ronda de entrenamiento.
- **OnModelReceived:** Callback activado cuando se recibe el modelo de un cliente.
- **OnClientReceivedModel:** Callback activado cuando un cliente recibe el modelo satisfactoriamente.

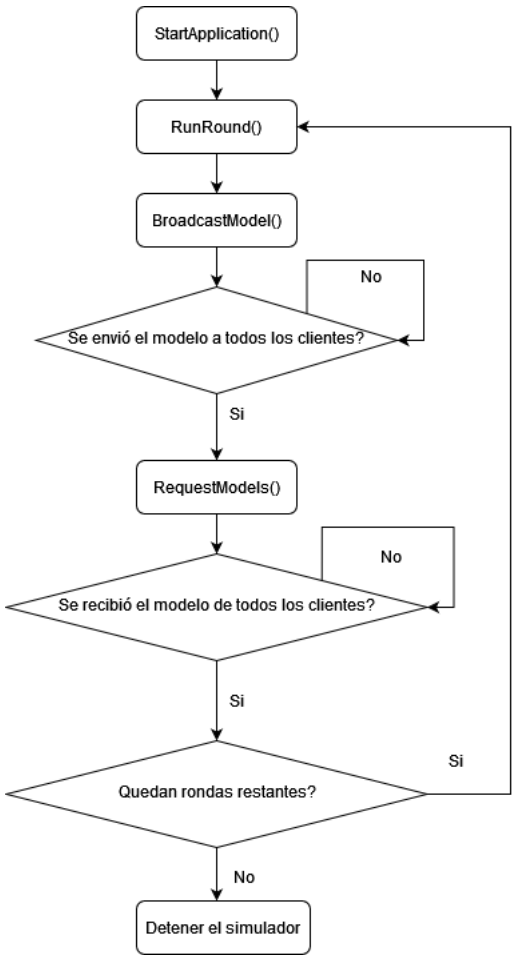

Figura 3.6: Diagrama del flujo de ejecución de FLServerApp.

3.2.2.2. FLClientApp

Aplicación encargada de controlar los nodos de los clientes, es supervisada por la aplicación del servidor la cual solicita las peticiones al cliente. Representa el funcionamiento de un cliente en un sistema de aprendizaje federado. Si bien en una red de aprendizaje federado real el intervalo de tiempo entre los envíos de modelos entre cliente y servidor puede ser de horas o incluso días, ya que el cliente debe adquirir información adicional para entrenar su modelo, por simplicidad el modelo entrenado se envía de vuelta al servidor casi inmediatamente con un pequeño retardo. Esto para que los registros de la captura de paquetes solo contengan información relevante respecto al funcionamiento del envió de modelos.

Métodos

- **Setup:** Configura los parámetros del cliente antes del inicio de la simulación.
- **SendModel:** Envía el modelo entrenado de vuelta al servidor.
- **OnModelRecieved:** Callback activado al recibir el modelo FL desde el servidor.

3.2.2.3. PyConnection

Interfaz entre el simulador NS3 y el modelo FL, se encarga de administrar la comunicación TCP y escuchar los comandos ejecutados por el simulador central.

Métodos

- **Listen:** Escucha el puerto designado para la conexión en espera de un comando que ejecutar.
- **HandleData:** Procesa los comandos y datos enviados por el modelo FL para su ejecución.
- **AlertEnd:** Envía un mensaje de aviso al modelo FL cuando la simulación termina.

3.2.2.4. BulkSocket

Esta clase permite enviar grandes cantidades de datos entre los nodos. Utiliza sockets TCP y se dedica a mantener el correcto funcionamiento de las conexiones servidor-cliente. Su uso principal es en el envió del modelos FL entre los nodos.

Métodos

- **Setup:** Establece los parámetros del envío como la IP del nodo emisor, el destinatario, el tamaño del segmento TCP y la cantidad total de bytes a enviar.
- **Init:** Comienza a enviar los datos luego de haber configurado los parámetros previamente con 'Setup'

3.3. Simulador de modelos FL

Ya visto el simulador de red, en esta sección se presenta el simulador de modelos FL. Este simulador se encarga se realizar el entrenamiento de el modelo FL solicitado, enfocándose en el funcionamiento de los algoritmos y evaluando el rendimiento una vez entrenado los modelos. Además, este programa es el encargado de controlar al simulador de red, para ejecutar la comunicación que se haría en un modelo FL implementado en la realidad. A continuación se muestran los componentes del simulador y el modelo de machine learning que se utilizara para realizar los experimentos.

3.3.1. Flower

Flower[20] es un *framework* de machine learning federado, permite utilizar el aprendizaje federado en cualquier tipo de modelo machine learning y tiene soporte para distintas librerías como TensorFlow, PyTorch, MXNet, etc. El algoritmo que se utilizara para unir los modelos locales es el promedio federado (FedAvg).

3.3.2. CIFAR-10

El CIFAR-10[21] es un dataset compuesto de 60000 imágenes a color de 32x32 píxeles, clasificadas en 10 categorías mutuamente exclusivas. Se utilizará este dataset para entrenar un clasificador de manera federada. El modelo clasificador será una red neuronal convolucional, en la Figura 3.7 se puede ver un diagrama con su arquitectura.[22]

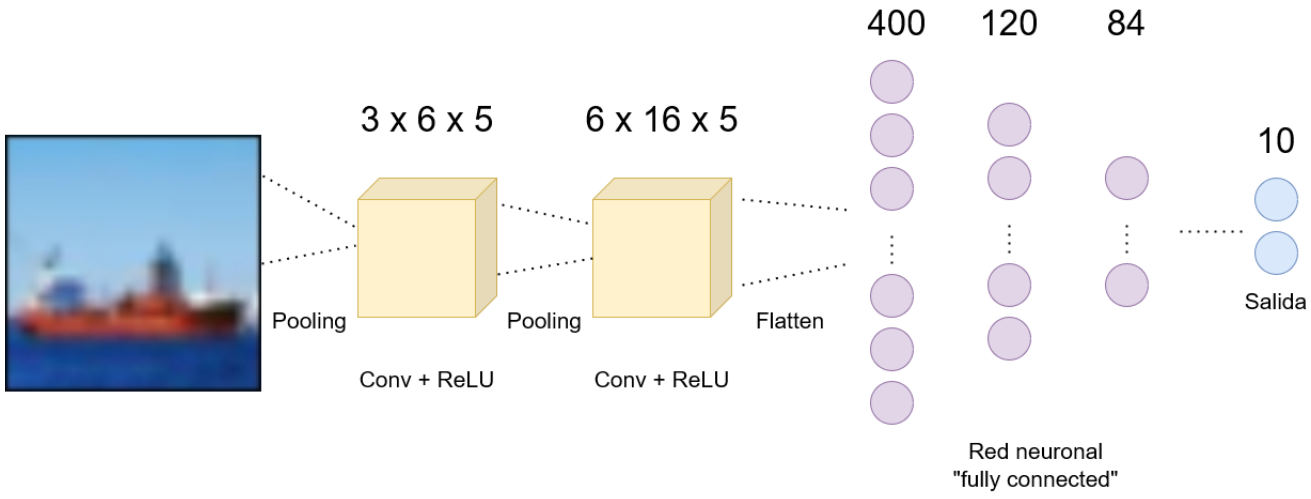

Figura 3.7: Diagrama de la red neuronal convolucional.

3.4. Experimentos

Para evaluar el rendimiento del simulador, se realizaron ciertas simulaciones, variando la configuración de los parámetros del simulador. La idea es observar el comportamiento de los sistemas FL y la red en entornos diversos que ejemplifiquen situaciones de redes reales. Luego se analizarán las estadísticas generadas por el simulador para determinar como los distintos parámetros afectan el rendimiento del sistema. Se evaluaran tres casos que representan ciertas características en la red, estos casos son las siguientes. En la Tabla 3.1 se pueden ver la opciones de configuración utilizadas en cada caso, una explicación mas detallada de cada opción se puede encontrar en la sección 3.1.4 Parámetros de configuración.

- **Caso base:** Representa el funcionamiento normal de la red, con solo 10 clientes y suficiente banda ancha para la transferencia de los modelos, entrenando 4 rondas.

- **Caso congestión:** En este caso se aumenta la cantidad de clientes a 50. Además se reduce el ancho de banda de la estación base, esto se hace para representar un exceso de clientes intentando comunicarse, esto se podría hacer subiendo la cantidad de clientes aun mas, pero para este trabajo no se cuenta con la capacidad computacional para simular una mayor cantidad de nodos por lo que se utilizaran estas limitantes equivalentes.

- **Caso larga distancia:** En este escenario se mantiene la cantidad de clientes del caso base pero se aumenta 5 veces la distancia entre los clientes y la estación base. Esto para estudiar los efectos de la distancia en el sistema federado.

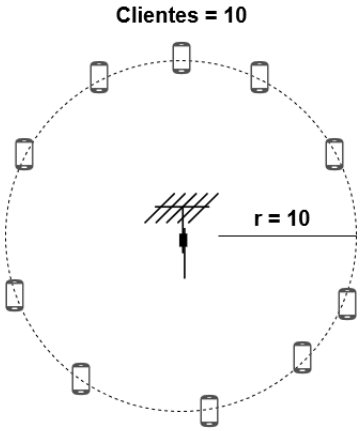

Figura 3.8: Diagrama del experimento, caso base.

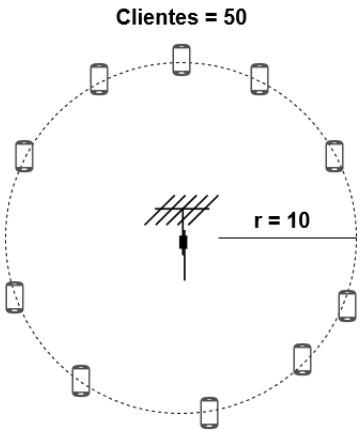

Figura 3.9: Diagrama del experimento, caso congestión.

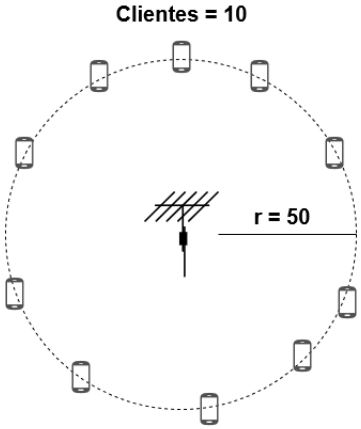

Figura 3.10: Diagrama del experimento, caso larga distancia.

Tabla 3.1: Opciones de configuración para cada caso.

Propiedad	**Caso base**	**Caso congestión**	**Caso larga distancia**
frequency	4e9	4e9	4e9
bandwidth	100e6	50e6	100e6
txpower	50	50	50
hbs	25	25	25
hut	1.5	1.5	1.5
segmentsize	1500	500	1500
scenario	UMa	UMa	UMa
nodes	10	50	10
radius	10	10	50
modelsize	250466	250466	250466
rounds	4	4	4

Capítulo 4

Resultados

Una vez realizadas las simulaciones se tienen los resultados a continuación. En la tabla 4.1 podemos ver la latencia promedio en cada caso, se observa que la congestión causa un aumento de 19.04 ms en el caso base a unos 43.43 ms. Para analizar esto en mas detalle en las Figuras 4.1,4.2 y 4.3 se puede observar el histograma de latencia de paquetes. La latencia se mide como el tiempo de ida y vuelta (RTT), en el caso de una perdida de paquete este no se considera en la medición.

Tabla 4.1: Latencia medida en cada caso

Caso	**Latencia promedio [ms]**	σ
Base	19.04	3.7
Congestión	43.43	19.31
Larga distancia	19.1	3.87

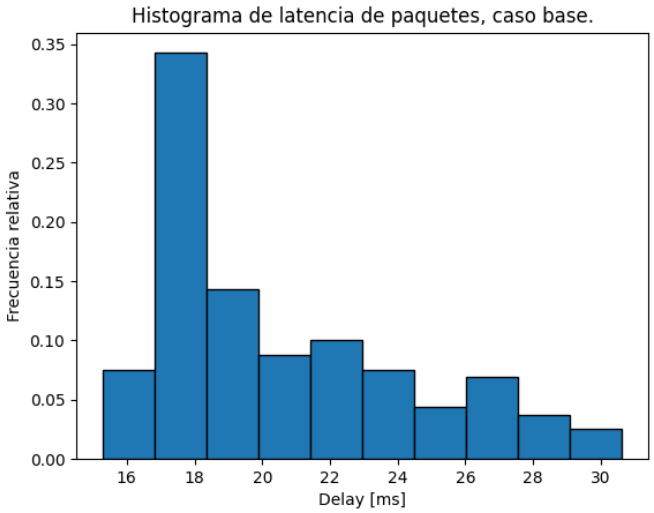

Figura 4.1: Histograma latencia medida, caso base.

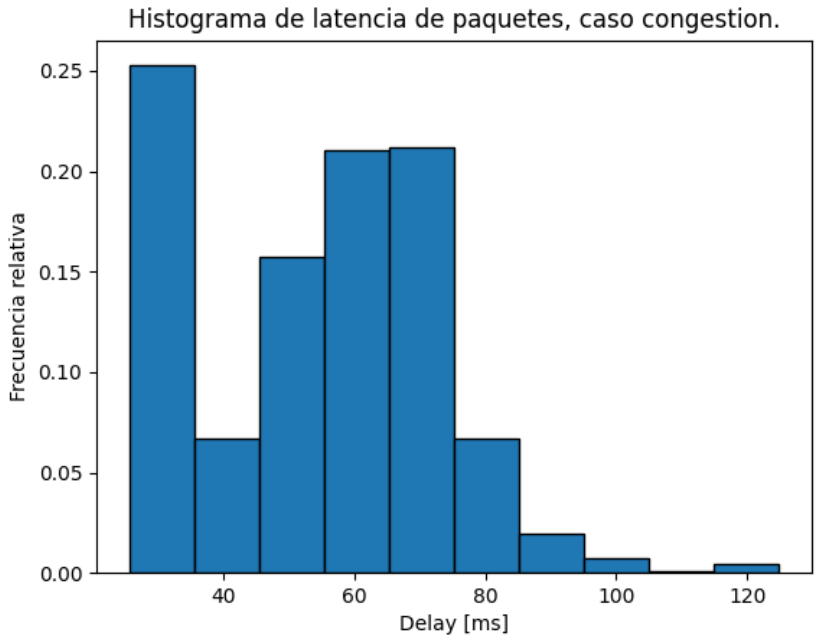

Figura 4.2: Histograma latencia medida, caso congestión.

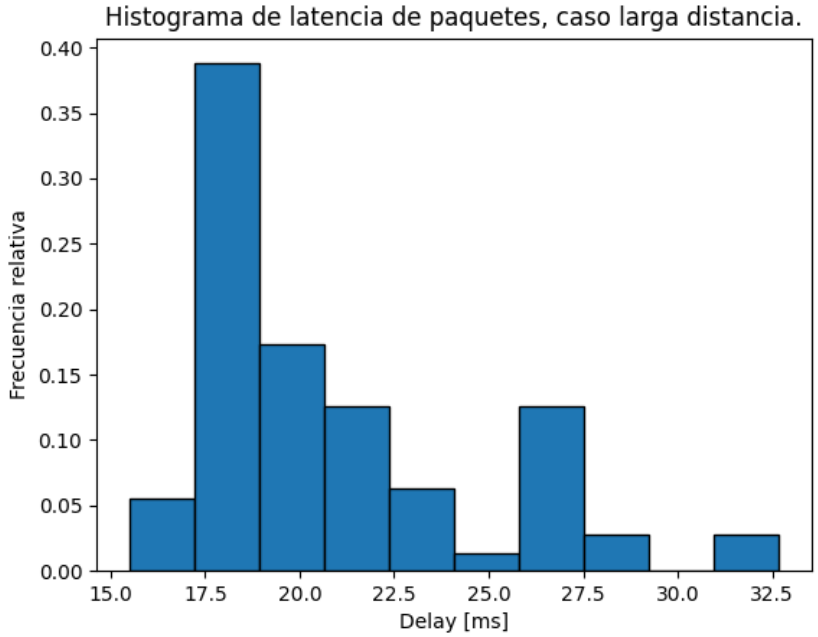

Figura 4.3: Histograma latencia medida, caso larga distancia.

Las mediciones de jitter se pueden encontrar en la Tabla 4.2 y en las Figuras 4.4.4.5 y 4.6.

Tabla 4.2: Jitter medido en cada caso

Caso	**Jitter promedio [ms]**	σ
Base	0.52	0.16
Congestión	1.27	0.62
Larga distancia	0.50	0.15

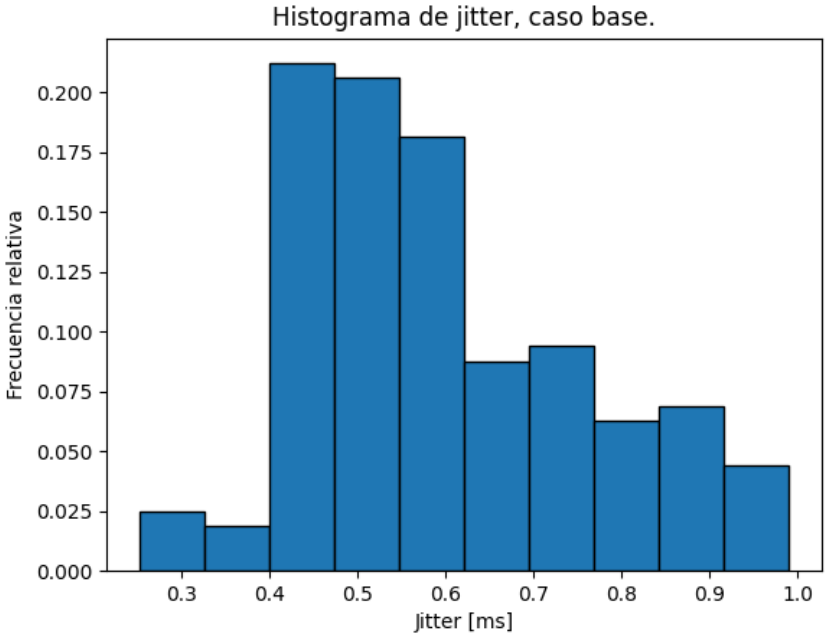

Figura 4.4: Histograma jitter medido, caso base.

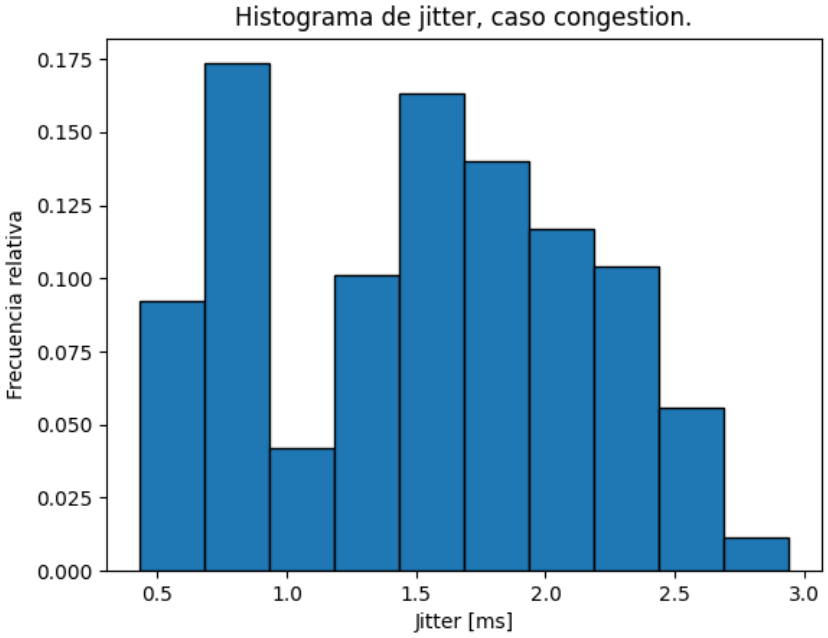

Figura 4.5: Histograma jitter medido, caso congestión.

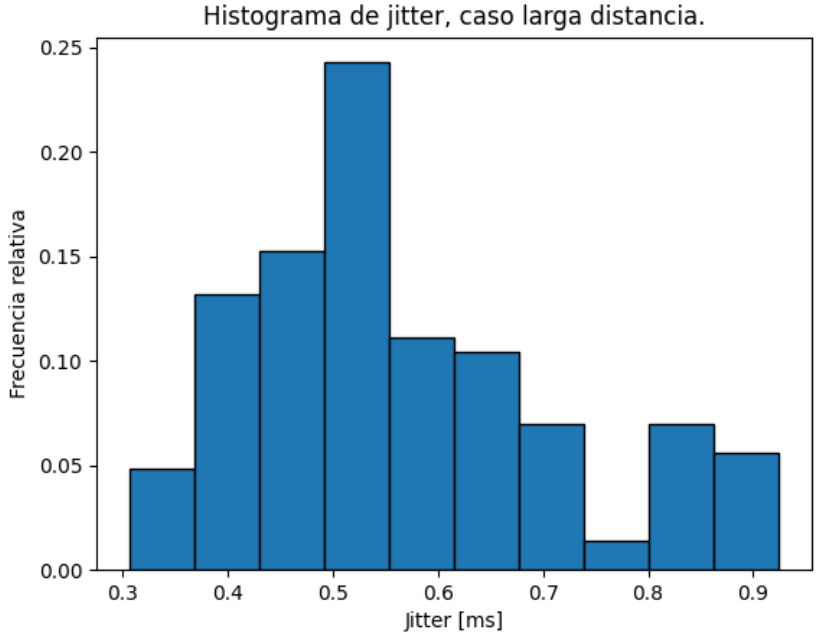

Figura 4.6: Histograma jitter medido, caso larga distancia.

En la Tabla 4.3 se presenta la cantidad de retransmisiones TCP totales, los paquetes totales transmitidos y para un mejor análisis se calcula el porcentaje entre ambos. En la Tabla 4.4 se observa el tiempo de ejecución que le toma a la red en enviar todas las rondas de entrenamiento para cada caso, en segundos. El progreso de la ejecucion se puede visualizar con mas detalle en la Figura 4.7 en donde se gráfica para cada caso el progreso del entrenamiento

que se representa por $(\frac{bytes\ enviados}{bytes\ totales})$ a lo largo del tiempo. En la Figura 4.8 se observa el aumento de la precisión en el modelo FL por cada ronda de entrenamiento.

Tabla 4.3: Retransmisiones TCP

Caso	**Retransmisiones**	**Paquetes totales**	**Porcentaje**
Base	0	27120	0 %
Congestión	8548	419891	2.03 %
Larga distancia	5	24516	>0 %

Tabla 4.4: Tiempo total de ejecución del entrenamiento.

Caso	**Tiempo [s]**
Base	1.728
Congestión	9.564
Larga distancia	11.597

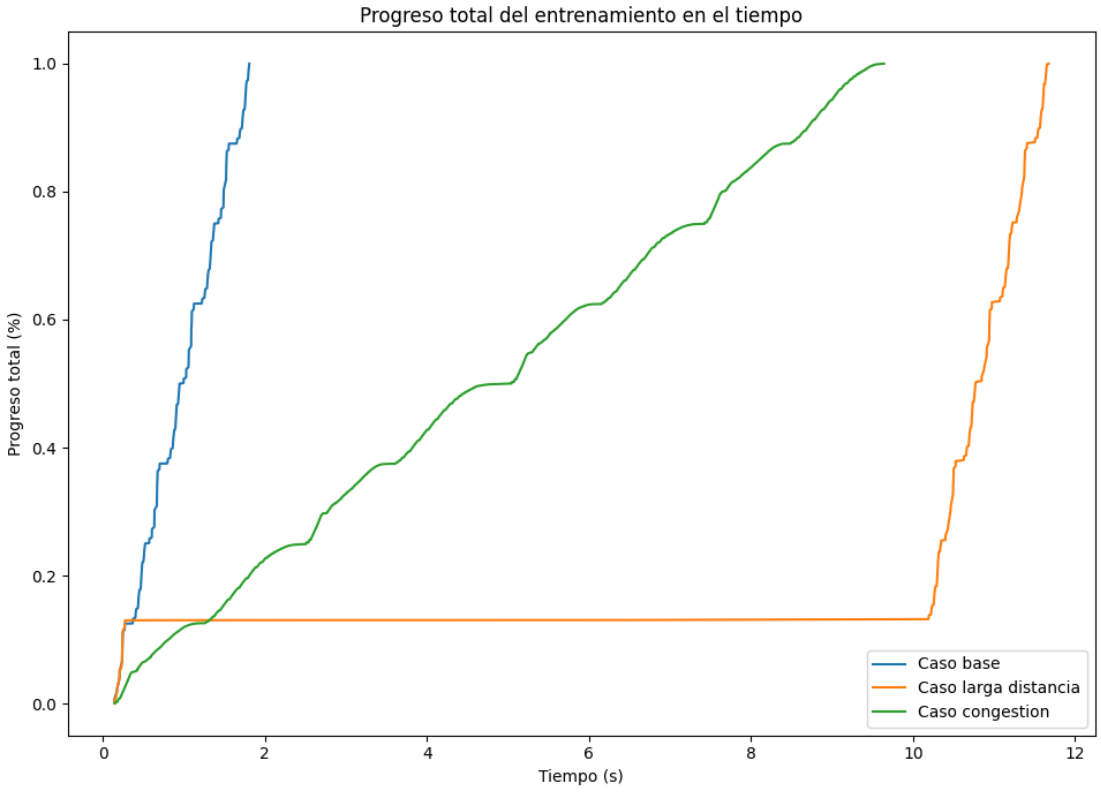

Figura 4.7: Progreso del proceso de entrenamiento federado en el tiempo.

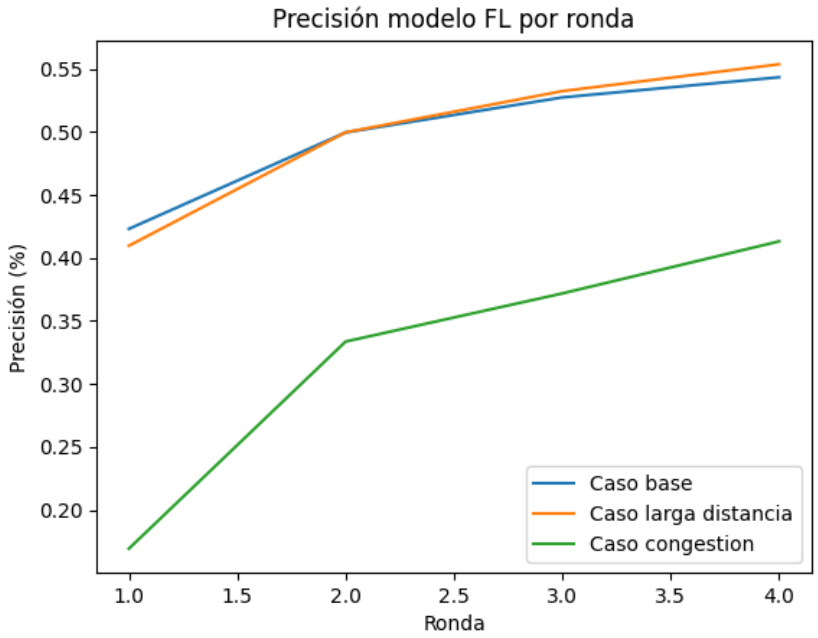

Figura 4.8: Precisión del modelo global por ronda de entrenamiento.

Capítulo 5

Conclusiones

El objetivo propuesto en este trabajo es el desarrollo de un simulador para sistemas de *federated learning* para redes móviles 5G. Sobre el entorno de simulación presentado, se muestra el desarrollo del simulador de red que permite emular las interacciones de red entre los dispositivos de los clientes y una torre base, lo que nos entrega registros y estadísticas relevantes de la red para evaluar su desempeño. Además, se pudo desarrollar una simulación de modelos FL basada en el *framework* Flower que permite entrenar los modelos solicitados en la simulación. Finalmente, se pudieron integrar estos sistemas en un entorno de simulación en el cual se emularon distintos escenarios de simulación y evaluar sus rendimientos.

Con respecto a la latencia, como se puede ver en los histogramas de las Figuras 4.1,4.2 y 4.3, para el caso base y larga distancia se observa a la mayoría de los paquetes en un rango de 17 a 19 ms en donde la latencia máxima llega a los 30 ms. Mientras que en el caso congestión la latencia aumenta considerablemente y se observa una distribución bimodal, con la mayoría de los paquetes en el rango de 50 a 70 ms y otro grupo con una latencia de entre 30 a 40 ms, siendo la latencia máxima hasta 120 ms. Esto muestra el efecto directo que causa la congestión en la red. Como es esperado, aumentar la distancia no aumenta la latencia significativamente. La congestión causa que el jitter promedio aumente mas del doble que en el caso base, como se observa en la Tabla 4.2. En los histogramas se puede observar un patrón similar al de la latencia. Además, el caso de larga distancia resultó ser el que se demoró mas tiempo en ejecutar, incluso cuando mostraba estadísticas similares al caso base que fue el mas rápido. Lo que se explica debido a la falla de conexión en uno de los clientes el cual tardó 10 segundos en reconectarse. Incluso cuando el aprendizaje federado tiene una naturaleza descentralizada, este esquema de aprendizaje requiere unir todos los modelos antes de continuar, por lo que con un solo cliente que presente un problema de conexión es suficiente para detener el progreso del modelo. Si bien se pudo desconectar al cliente en el primer *timeout*, eso hubiera implicado una perdida de datos por lo que no correspondería al escenario que se quería simular. En la Tabla 4.3 se ve que el caso de larga distancia solo tuvo 5 retransmisiones, sin embargo, esto tuvo un mayor impacto que las mas de 4000 retransmisiones en la congestión. Esto se explica debido a que en el caso de la larga distancia, al haber una perdida de conexión el sistema espera el *timeout* (configurado por defecto en 2 segundos) para pedir una retransmisión por lo que estas tienen un costo en el rendimiento considerable. Esto muestra la importancia de tener una conexión robusta ya que las perdidas de conexión por una perdida de señal implica un mayor costo en rendimiento que incluso una congestión.

Trabajos futuros

Dado el trabajo presentado en esta memoria, se presentan ciertas alternativas para expandir el simulador. Esto se enfoca a añadir una mayor cantidad de funcionalidad para facilitar el uso en simulación de aplicaciones reales y de mayor complejidad. Algunas posibilidades para extender el simulador son:

Expandir la configuración de topologías

Extender la configuración del modelo para aceptar una topología de red mas compleja que la actual es fundamental para simular la mayoría de las redes que usualmente se implementarían. Aquello implica permitir agregar una mayor cantidad de estaciones base y poder especificar las posiciones de las estaciones base y los clientes, puesto que la gran mayoría de redes 5G están compuestas de varias antenas debido a las perdidas por la alta frecuencia a la que opera. Además, generalizar el modelo FL para permitir cualquier modelo de *machine learning* y algoritmo federado. Esto no requeriría grandes cambios ya que la estructura del software se hizo para ser extensible, por lo que se puede utilizar un modelo distinto a través de crear una nueva clase que herede de la clase base actual.

Implementar arquitecturas de redes 5G *Non-standalone*

Se les denomina arquitecturas *non-standalone* o NSA, por sus siglas en ingles, a aquellas redes 5G que son implementadas dentro de una infraestructura LTE o de cuarta generación. Este tipo de redes tiene la ventaja de que son mas rápidas de implementar y pueden utilizar la infraestructura que ya fue instalada, necesitando una menor inversión. Estas redes se utilizan en aplicaciones reales para mejorar las capacidad de la red 4G existente, por lo que seria interesante ser capaz de simular redes que no sean puramente 5G. En el simulador, el módulo 5G LENA fue desarrollado a partir del módulo 4G *mmWave*, con el cual comparte similitudes. Para realizar este cambio, seria necesario refactorizar el código de inicialización de la simulación, tal que se contenga en una clase la cual pueda ser extendida para agregar nuevos tipos de redes como las redes LTE.

Glosario

FL Federated Learning.

FML Federated Machine Learning.

FSGD Federated Stochastic Gradient Descent.

NSA Non-standalone Architecture.

RPC Remote Procedure Call.

UE User Equipment.

Bibliografía

[1] D. Verma, *Federated AI for Real-World Business Scenarios*. 07 2021.

[2] F. Al-Turjman, *Real-Time Intelligence for Heterogeneous Networks: Applications, Challenges, and Scenarios in IoT HetNets*. Springer International Publishing, 2021.

[3] "Ieee guide for architectural framework and application of federated machine learning," *IEEE Std 3652.1-2020*, pp. 1–69, 2021.

[4] P. Kairouz, H. B. McMahan, and e. a. Brendan Avent, "Advances and open problems in federated learning," *CoRR*, vol. abs/1912.04977, 2019.

[5] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016.

[6] 3GPP, "5g; procedures for the 5g system (3gpp ts 23.502 version 15.2.0 release 15)," 2018. https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/15.02.00_60/ts_123502v150200p.pdf.

[7] G. Velez, Martín, G. Pastor, and E. Mutafungwa, "5g beyond 3gpp release 15 for connected automated mobility in cross-border contexts," *Sensors*, vol. 20, no. 22, 2020.

[8] 3GPP, "5g system overview," 2022. https://www.3gpp.org/technologies/5g-system-overview.

[9] X. Lin, "An overview of 5g advanced evolution in 3gpp release 18," *CoRR*, vol. abs/2201.01358, 2022.

[10] J. Torsner, K. Dovstam, G. Miklós, B. Skubic, G. Mildh, T. Mecklin, J. Sandberg, J. Nyqvist, J. Wang, B. Zhang, C. Martinez, and J. Neander, "Industrial remote operation: 5g rises to the challenge," vol. 93, pp. 54–69, 01 2016.

[11] L. SolarWinds Worldwide, "Gns3." https://www.gns3.com/, 2022.

[12] I. Kholod, E. Yanaki, D. Fomichev, E. Shalugin, E. Novikova, E. Filippov, and M. Nordlund, "Open-source federated learning frameworks for iot: A comparative review and analysis," *Sensors*, vol. 21, no. 1, 2021.

[13] L. Google, "Tensorflow federated." https://www.tensorflow.org/federated, 2022.

[14] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, J. Martin, B. Edwards, M. J. Sheller, S. Pati, P. N. Moorthy, S. han Wang, P. Shah, and S. Bakas, "Openfl: An open-source framework for federated learning," 2021.

[15] P. Pinyoanuntapong, T. Pothuneedi, R. Balakrishnan, M. Lee, C. Chen, and P. Wang, "Sim-to-real transfer in multi-agent reinforcement networking for federated edge com-

puting," 2021.

[16] T. Pothuneedi, "Ai based realistic multi-hop wireless simulation," 2021.

[17] E. Ekaireb, X. Yu, K. Ergun, Q. Zhao, K. Lee, M. Huzaifa, and T. Rosing, "Ns3-fl: Simulating federated learning with ns-3," in *Proceedings of the 2022 Workshop on Ns-3*, WNS3 '22, (New York, NY, USA), p. 97–104, Association for Computing Machinery, 2022.

[18] M. Rehman and M. Gaber, *Federated Learning Systems: Towards Next-Generation AI*. Studies in Computational Intelligence, Springer International Publishing, 2022.

[19] CTTC, "5g-lena simulator," 2022. https://5g-lena.cttc.es/.

[20] G. Adap, "Flower: A friendly federated learning framework." https://flower.dev/, 2022.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[22] G. Adap, "Pytorch - from centralized to federated." https://flower.dev/docs/example-pytorch-from-centralized-to-federated.html#example-pytorch-from-centralized-to-federated, 2022.