Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Herramienta informática de apoyo a la docencia en la asignatura Fundamentos de Redes de Ordenadores

Grado en Ingeniería Mecánica

ALUMNO: Juan Antonio Ramírez Peña

DIRECTORES: Norberto Fernández García

Milagros Fernández Gavilanes

CURSO ACADÉMICO: 2019-2020

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Herramienta informática de apoyo a la docencia en la asignatura Fundamentos de Redes de Ordenadores

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

RESUMEN

En el quinto curso del Grado en Ingeniería Mecánica, como parte de la intensificación en Tecnología Naval, se imparte la asignatura “Fundamentos de Redes de Ordenadores”. Los conocimientos en arquitectura y funcionamiento de los modelos de redes son primordiales para los futuros Oficiales de la Armada debido a los continuos avances tecnológicos. De hecho, algunos de ellos serán responsables de destinos en los que se administren las redes de la Armada como puede ser un Centro de Comunicaciones (CECOM).

Esta asignatura puede resultar difícil de comprender para el alumnado ya que los conceptos impartidos en su temario, como pueden ser las diferentes capas en las que se dividen los diferentes modelos de red, la calidad de servicio y los aspectos relativos a seguridad son conceptos inexorablemente abstractos.

Es por esto por lo que se ha desarrollado una herramienta informática que represente de manera visual e interactiva algunos conceptos que se imparten en la materia. Se pretende con ello facilitar las labores de enseñanza del profesorado y las de comprensión del alumnado para los próximos cursos académicos.

PALABRAS CLAVE

Jupyter Notebook, Python, Aprendizaje, Redes de Ordenadores, Simulación

AGRADECIMIENTOS

Quiero agradecer, en primer lugar, a mis tutores Don Norberto Fernández García y Doña Milagros Fernández Gavilanes el continuo apoyo realizado a lo largo del desarrollo del proyecto.

De la misma manera, a los usuarios del foro *Stackoverflow* la resolución de dudas puntuales y errores surgidos al compilar.

También, al Centro Universitario de la Defensa todo el material y documentación que ha puesto a mi disposición para su realización.

Por último, agradecer a mis compañeros de la Promoción 420 / 150 y, en particular, a los de mi camareta (Francisco Márquez Lumpié, Jerónimo Domínguez Fernández-Núñez y Pablo González Álvarez) su paciencia y el apoyo y las opiniones aportados.

CONTENIDO

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS

En este capítulo se introducirá la asignatura de Fundamentos de Redes de Ordenadores, los conceptos que se imparten en ella y los objetivos y organización del presente trabajo fin de grado.

1.1 Contextualización

Una Red de Ordenadores [1] es un conjunto de dispositivos informáticos interconectados que permiten un intercambio de datos (Figura 1-1). Su finalidad es permitir, desde cualquier lugar que disponga de acceso a la red, la transmisión de información y ofrecer servicios a distancia. Gracias a los avances tecnológicos, los sistemas actuales pueden realizar un gran número de intercambio de datos de manera prácticamente instantánea, siempre teniendo en cuenta que el medio empleado como canal puede influir en la velocidad de transmisión.

Es por esto por lo que actualmente las redes de ordenadores han pasado a ser de interés común, resultando elementos cotidianos e imprescindibles y siendo utilizadas en diversos campos tales como:

- Negocios: un ejemplo conocido es el de una impresora compartida en una oficina.
- Aplicaciones domésticas: consultar información, comunicarse y comprar desde casa.
- Usuarios móviles: uso de Internet fuera de su domicilio u oficina.

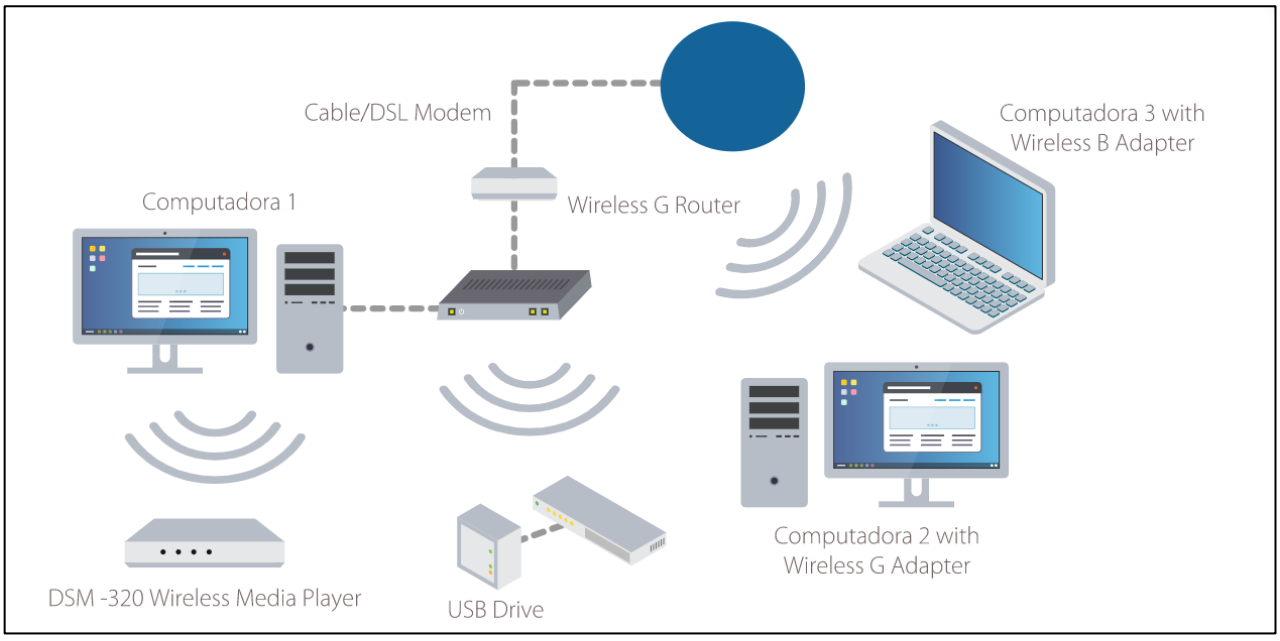


Figura 1-1 Redes de ordenadores [2]

Las redes de ordenadores se clasifican según [1]:

- Su área de cobertura: PAN (*Personal Area Network*), LAN (*Local Area Network*), MAN (*Metropolitan Area Network*) y WAN (*Wide Area Network*).
- Su ámbito de uso: Internet, Intranet y Extranet.
- Su relación funcional: Cliente-Servidor y *Peer to Peer*.
- Su topología: Malla, Estrella, Anillo, Bus, Árbol, etc.

1.2 Redes de Ordenadores en las Fuerzas Armadas

Las Fuerzas Armadas Españolas [3] disponen de dos redes de área extensa:

- Red de área extensa de propósito general: WAN PG.
- Red de área extensa de mando y control: WAN C2.

La red WAN PG es una red privada virtual proporcionada por la compañía Telefónica. Esta red interconecta más de 700 emplazamientos (Figura 1-2) ubicados tanto en territorio nacional como en el extranjero y permite el acceso a servicios de propósito general a unidades que se encuentran en zonas de operaciones.

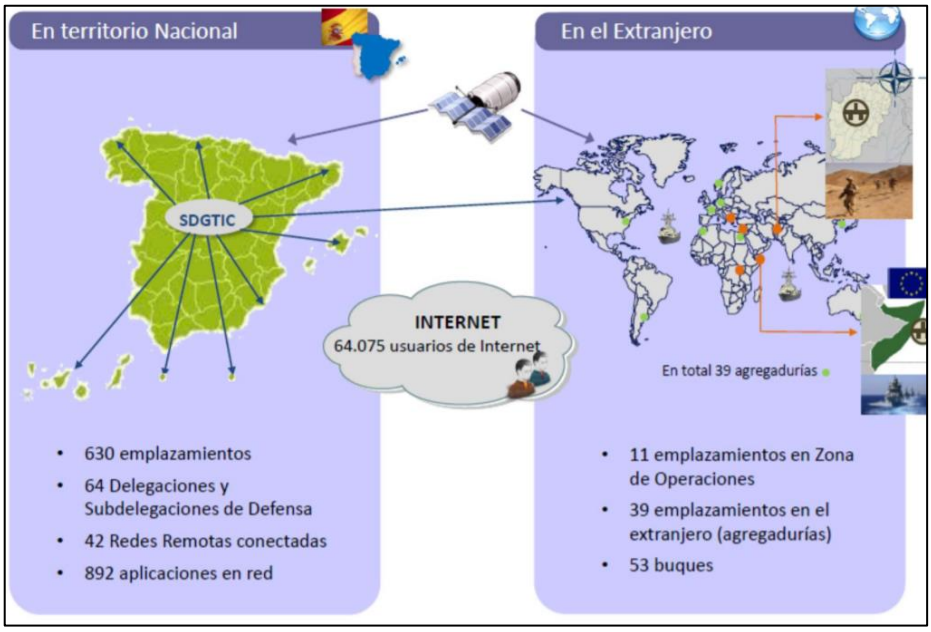


Figura 1-2 Red WAN PG [3]

En cada uno de los emplazamientos pueden existir además redes LAN PG que se conectan al enlace WAN (Figura 1-3), permitiendo el enlace con otros emplazamientos.

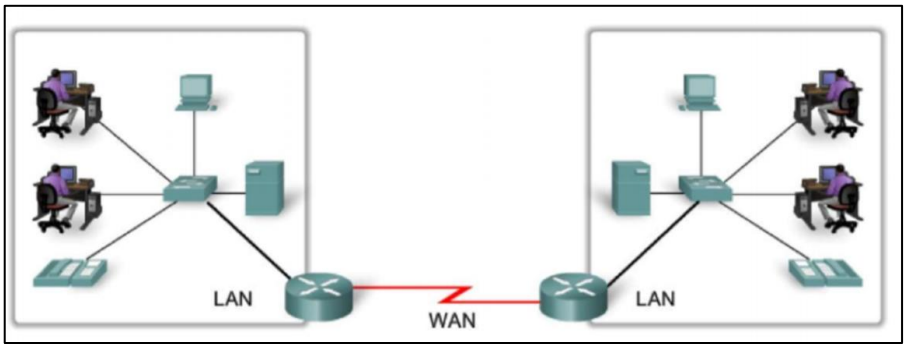


Figura 1-3 Redes LAN PG [3]

Por otro lado, la red WAN C2 (Figura 1-4) consiste en una red de área extensa cuyos elementos permiten al mando el desempeño de sus funciones. Esto es, el conocimiento de la situación para la toma de decisiones, la transmisión de órdenes y el control de su ejecución. Estos sistemas de mando y control de las Fuerzas Armadas tienen clasificaciones de seguridad de Confidencial y Reservado por lo que no se adjunta información sobre ellos.

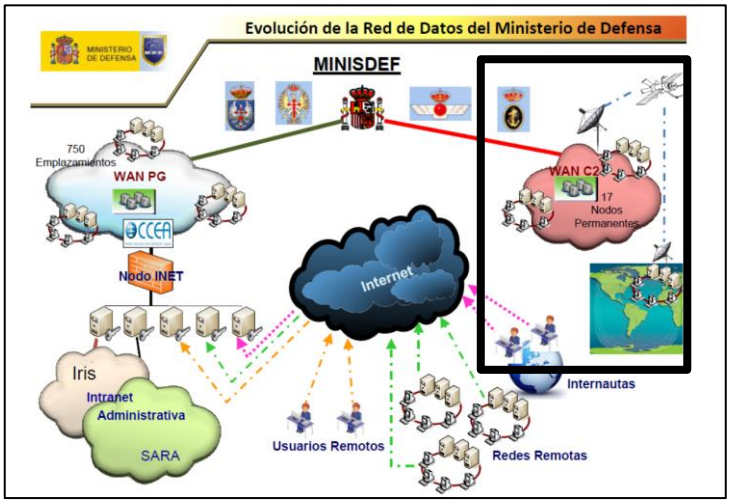


Figura 1-4 Red WAN C2 [3]

1.3 Historia de las Redes de Ordenadores

La historia de las Redes de Ordenadores [4] se remonta a finales de la década de 1950, durante la Guerra Fría, donde Estados Unidos consideró necesaria una red de mando y control para comunicarse de manera fiable y que fuese resistente a ataques nucleares, ya que por aquel entonces se hacía uso de la red telefónica pública.

Por aquel entonces, se decidió crear una única organización de investigación para la defensa, la Agencia de Proyectos de Investigación Avanzados (ARPA, *Advanced Research Projects Agency*). Tras varios intentos de diseño fallidos, es en 1967 cuando Wesley Clark, uno de los expertos a los que había consultado ARPA, propuso la creación de una subred de conmutación de paquetes conectando cada equipo a un enrutador. Se retomaba así alguna de las ideas que habían sido descartadas anteriormente.

Con esta idea, en 1968 ARPA le encargó la construcción de la red (que posteriormente se denominaría como ARPANET) a BBN, una empresa de consultoría (Figura 1-5). El sistema consistía en un conjunto de microcomputadoras conocidas como IMP (*Interface Message Processors*), conectadas como mínimo con otras dos por líneas de transmisión de 56 kbps. La red permitiría reencaminar automáticamente los mensajes por otras rutas si alguna fallaba, gracias a que sería de conmutación de paquetes (datagramas).

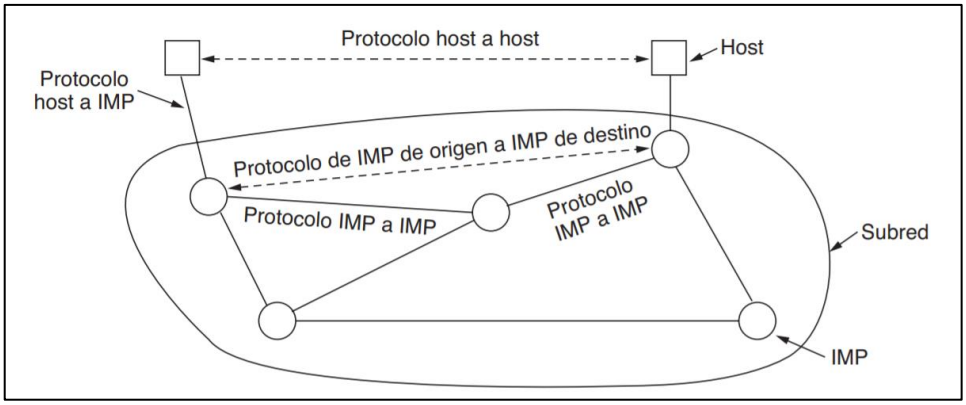


Figura 1-5 Diseño original de ARPANET [4]

A finales del 1969 [4] comenzaron pruebas en una red experimental formada por 4 nodos: UCLA (University of California, Los Angeles), SRI (Stanford Research Institute), UCSB (University of California, Santa Barbara*)* y UTAH (University of Utah) y se logró la transmisión del primer mensaje de nodo a nodo desde UCLA a SRI. Como muestra la Figura 1-6, la red fue creciendo rápidamente conforme se iban instalando más microcomputadoras, llegando a cubrir Estados Unidos en poco tiempo. Las fechas de cada una de las imágenes de la Figura 1-6 son:

a. Diciembre de 1969.
b. Julio de 1970.
c. Marzo de 1971.
d. Abril de 1972.
e. Septiembre de 1972.

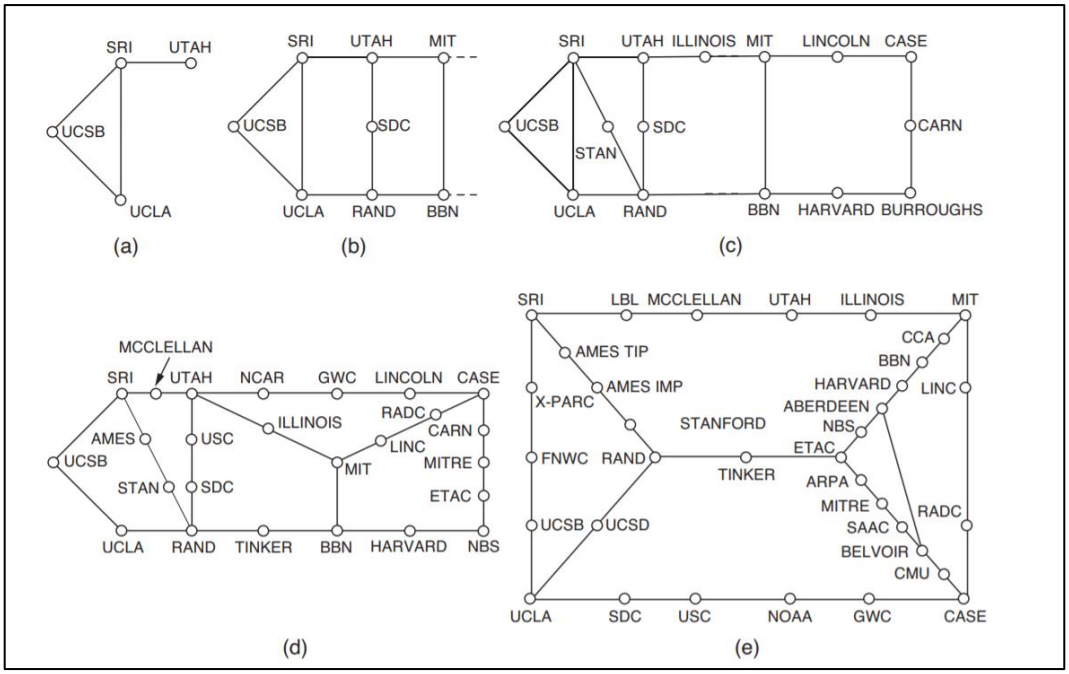


Figura 1-6 Crecimiento de ARPANET durante los tres primeros años [4]

Los protocolos del modelo de ARPANET no servían si se quería trabajar con diferentes redes, por lo que se diseñó el modelo y los protocolos TCP/IP (Transmission Control Protocol e Internet Protocol), cuyos pioneros fueron Cerf y Kahn en 1974, conocidos popularmente como los "padres de Internet". El concepto de Internet nace de la filosofía de este proyecto, de interconectar redes de distintas clases.

A finales de los años 70 [4], la NSF (National Science Foundation) había visto la repercusión de ARPANET en la investigación científica al permitir el intercambio de datos y la colaboración mutua en investigaciones. Como no tenían un contrato de investigación para poder entrar en ARPANET, la NSF decidió crear una red para conectar seis centros de supercomputadoras. Esto se realizó complementando cada supercomputadora con una microcomputadora, llamada Fuzzball, que se conectaron a líneas de 56 kbps para formar la subred como hacía ARPANET. Se patrocinaron cerca de 20 redes regionales que tenían conexión a la red troncal, permitiendo que universidades, laboratorios y bibliotecas tuvieran acceso. Esta red completa se llamó NSFNET (Figura 1-7) y tuvo un éxito instantáneo que más tarde absorbió a ARPANET.

De esta manera, se permitía la comunicación entre diferentes redes que se iban conectando a ARPANET. Sin embargo, al irse conectando más redes, surgía otro problema. El proceso de buscar equipos se hacía demasiado complejo. Esto culminó en 1984 con la creación del DNS (*Domain Name System*) organizando los equipos en dominios y resolviendo los nombres de equipos en direcciones IP.

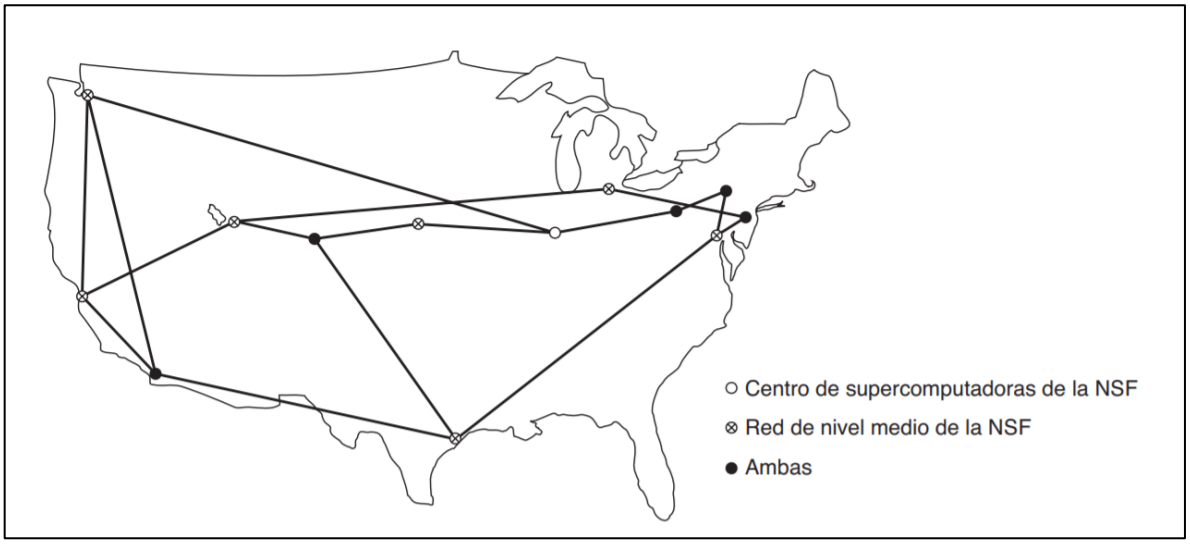


Figura 1-7 Red troncal de NSFNET [4]

Debido a la falta de financiación, se creó la ANS (Advanced Networks and Services), una corporación sin fines de lucro para dar paso hacia la comercialización. En este momento, NSFNET pasó a convertirse en ANSNET.

El éxito de ambas redes animó a que otros países se sumasen y construyesen otras redes de investigación que usualmente seguían los patrones de ARPANET o NSFNET. El crecimiento fue tan grande que en 1990 NSFNET ya contaba con 100 000 servidores [4]. Estas nuevas redes de libre acceso fueron uniéndose a NSFNET, construyendo el embrión de lo que actualmente conocemos como Internet.

Desde 1985, Internet ya era una tecnología establecida, pero conocida por pocos. Es a partir de 1990, cuando se produce la rápida expansión de Internet (Figura 1-8) gracias a la creación de la WWW (World Wide Web*)* por el CERN (Centro Europeo de Investigaciones Nucleares).

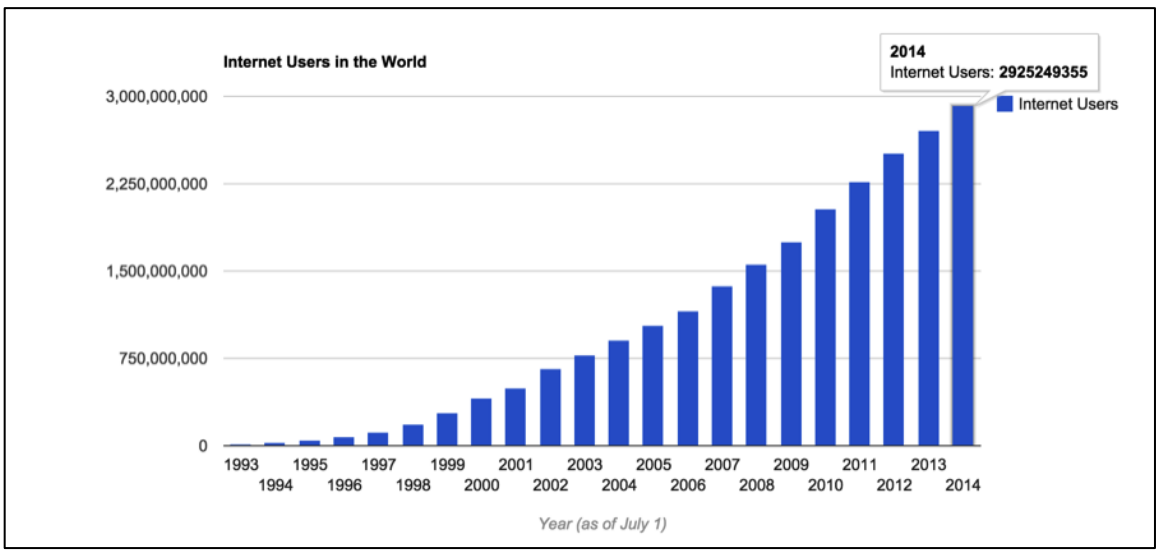


Figura 1-8 Crecimiento de Internet [5]

Aunque el modelo TCP/IP, de la familia de protocolos en los que se basa Internet, sea uno de los más difundidos a nivel práctico, existe otro modelo relevante. Para solucionar la incompatibilidad entre redes mencionada anteriormente, en 1984 se desarrolló paralelamente el modelo de referencia OSI por la ISO (Organización Internacional para la Estandarización). Este modelo de referencia es teórico. No se usa mucho en la práctica, ya que está pensado más bien para normalizar diseños.

1.4 Motivación y objetivos

La asignatura Fundamentos de Redes de Ordenadores es impartida por el Centro Universitario de la Defensa a los alumnos de 5º curso en la Escuela Naval Militar. La Guía docente de la asignatura [6] la encuadra dentro de la Intensificación en Tecnologías Navales y su objetivo es instruir al alumnado sobre conocimientos relativos a las redes de comunicación y servicios telemáticos. Para cumplir con los objetivos, el profesorado hace uso de las clases teóricas para impartir los conceptos del temario, las prácticas de laboratorio como complemento interactivo y el periodo de seminarios para la resolución de problemas y dudas del alumnado.

El temario de la asignatura engloba:

- Las diferentes capas del modelo de Internet: capa de enlace, capa de red, capa de transporte y capa de aplicación.
- Una introducción a los servicios y protocolos principales de cada capa.
- Aspectos relativos a la calidad de servicio ofrecido por la red.
- Conceptos relativos a ciberdefensa y ciberseguridad.
- Los sistemas de información y mando y control disponibles en la Armada Española.

Aunque el profesor haga un gran esfuerzo cuando da la clase, el alumno no siempre llega a comprender los conceptos impartidos. Esto se debe a que gran parte de los conceptos no pueden materializarse, llegando a ser irremediablemente abstractos y por lo tanto de difícil comprensión. Un ejemplo de esto puede ser el concepto de *Token Bucket* que, aunque pueda facilitarse su explicación gracias a su similitud a un cubo de agua con un agujero (Figura 1-9), no siempre llegan a comprenderlo todos los alumnos.

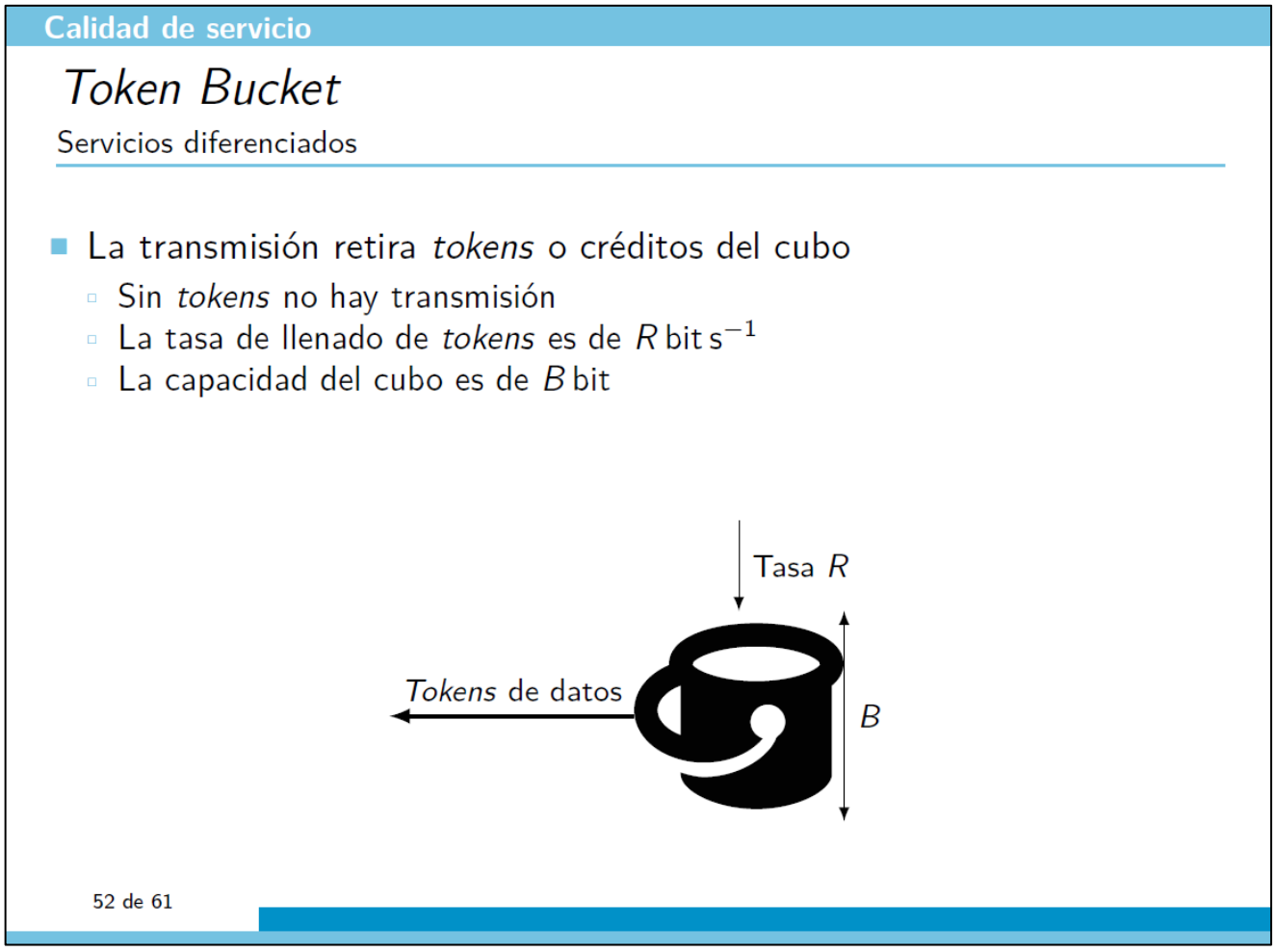


Figura 1-9 Dispositiva utilizada en la asignatura [7]

En el Nuevo Espacio Europeo de Educación Superior [8] se destaca la innovación en la docencia como uno de los pilares para la mejora de las clases presenciales. En concreto, se hace hincapié en el uso de herramientas interactivas para favorecer el aprendizaje del alumno. Leonardo Emiro Contreras Bravo, Julián Alfonso Tristancho Ortiz y Héctor Javier Fuentes López [9] explican el uso de herramientas informáticas como una estrategia pedagógica "*en el proceso de enseñanza-aprendizaje de la asignatura se hace necesario integrar la herramienta informática a la pedagogía de aula, esto es, el uso de computadores para realizar simulaciones y conexiones en red, con el fin de que el estudiante observe, comente, estudie y plantee dudas y/o sugerencias ante los diferentes ejercicios y proyectos de aplicación*". El hecho de que se estudie una asignatura con contenidos multimedia e interactivos publicados puede incrementar la motivación para el estudio de la misma.

El objetivo de este proyecto es el desarrollo de una herramienta informática formada por cuadernos realizados con Jupyter Notebook como apoyo a la docencia en la asignatura que facilite el aprendizaje de los conceptos de redes de ordenadores. De esta manera, con el desarrollo de este proyecto el alumno dispondrá en los próximos cursos académicos de un cuaderno informático en el que podrá interactuar con los conceptos que se han considerado críticos. El uso de esta herramienta resultará atractivo ya que permitirá su uso en los dispositivos personales. Para ello, se procederá a determinar cuáles son los conceptos de la asignatura más problemáticos apoyándose en las opiniones de los docentes de la asignatura y en una encuesta realizada a los alumnos que han cursado la asignatura este curso académico. Una vez determinados los conceptos que se simularán, se procederá a la instalación de las herramientas necesarias para su desarrollo y se desarrollarán los cuadernos.

1.5 Organización de la memoria

La presente memoria se estructura en los siguientes capítulos:

- Capítulo 1: Introducción y objetivos. Se introduce la asignatura para exponer los objetivos del trabajo fin de grado.
- Capítulo 2: Estado del arte. Herramientas similares al presente proyecto y herramientas empleadas para el desarrollo del mismo.
- Capítulo 3: Desarrollo del TFG. Instalación de las herramientas para el desarrollo del trabajo e implementación de la herramienta docente.
- Capítulo 4: Validación. Ejemplos de uso de la herramienta y análisis de los resultados que se obtienen con la misma.
- Capítulo 5: Conclusiones y líneas futuras de desarrollo del trabajo realizado.
- Anexo I: Códigos desarrollados en los notebooks. Se adjuntan los códigos completos de cada uno de los cuadernos.

2 ESTADO DEL ARTE

En este capítulo se describirán herramientas empleadas para el desarrollo del proyecto, así como las herramientas docentes similares a la que se pretende implementar. Aunque existan numerosas herramientas informáticas para docencia desarrolladas con Jupyter Notebook, no se encuentra casi ninguna que haya sido realizada con Jupyter Notebook para la enseñanza de redes de ordenadores; de ahí la propuesta del presente trabajo. Es por ello por lo que se han decidido presentar las herramientas similares a este trabajo categorizadas en diferentes apartados.

2.1 Herramientas utilizadas

2.1.1 Python

Python [10] es un lenguaje de programación interpretado que hace uso de una sintaxis legible, lo cual lo hace uno de los lenguajes de más fácil aprendizaje. Al contener muchas librerías, Python ayuda al programador a no tener que programar desde cero muchas tareas comunes que se vayan a realizar.

En Internet están publicados muchos gráficos como el de la Figura 2-1 sobre el crecimiento de este lenguaje. Aunque se le achaque que su tiempo de ejecución sea más lento (debido a que se trata de un lenguaje interpretado), Python está enfocado para acortar los tiempos de desarrollo de los programadores, facilitándoles su trabajo y de ahí su crecimiento. Además, hoy en día disponemos de grandes capacidades de computación a un precio asequible, por lo que el tiempo de ejecución no supone un gran problema.

También destaca por ser una de las mejores herramientas de *scripting* (capacidad de integrarse con otros lenguajes), *scraping* (extraer información de páginas web), *crawling* (revisar contenido y enlaces de una web e indexarlos) y desarrollo web (gracias a *frameworks* de desarrollo web muy potentes). Pero donde realmente Python es el rey es en *Data Science* y *Machine Learning* (de hecho, los cuadernos con Python que más abundan en Internet son de estos últimos). Esto se debe a que existen bibliotecas como *TensorFlow* y *Keras*, que contienen muchas funcionalidades de aprendizaje automático.

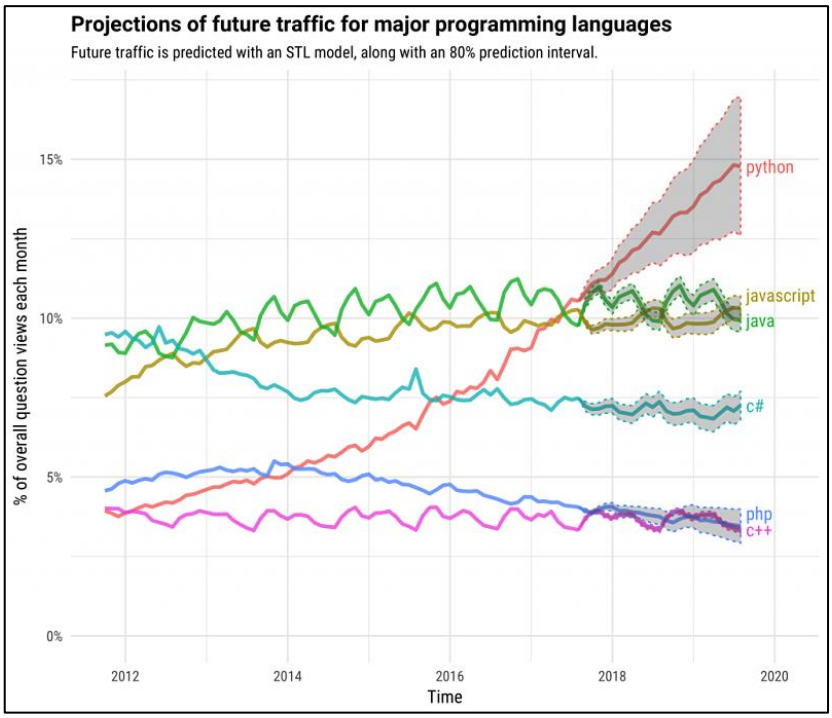


Figura 2-1 Predicción del uso de los lenguajes en el futuro [10]

Incluso una encuesta realizada por Google (Figura 2-2) muestra como más del 63% de las personas a las que entrevistó recomienda Python como primer lenguaje de programación a aprender.

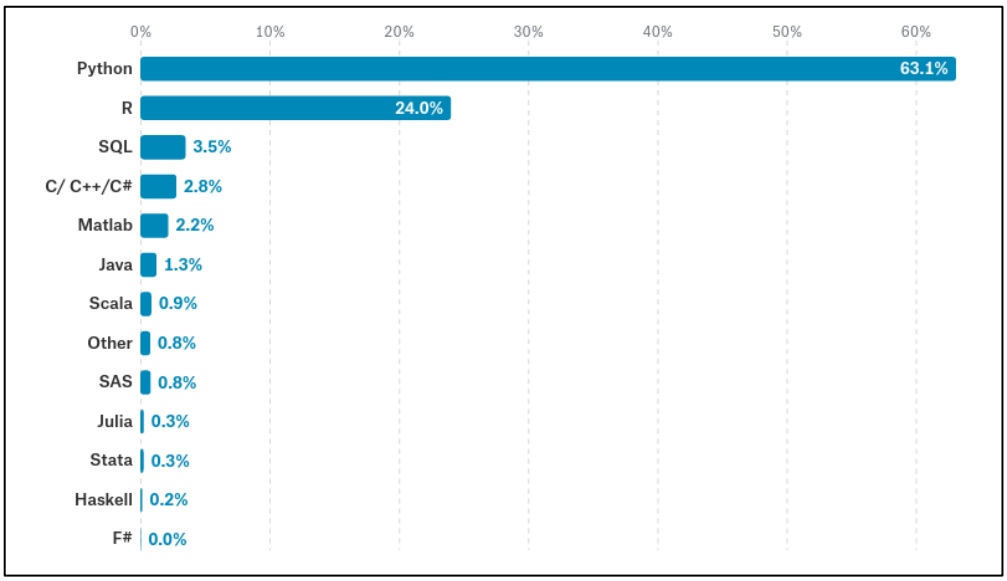


Figura 2-2 Encuesta realizada por Google [10]

Python también tiene un papel esencial en programación para redes de ordenadores [11]. Su biblioteca estándar tiene soporte para múltiples conceptos de redes como protocolos de red, codificación y decodificación de datos, haciendo más fácil el desarrollo de programas de red. Por lo tanto, solo con importar algunos módulos, se pueden desarrollar programas con Python que permitan interactuar con redes.

Un ejemplo de esto es el módulo *socket*. Un *socket* es el punto final de un canal de información, que permite crear clientes y servidores para realizar un intercambio de información entre programas de un mismo equipo o de diferentes equipos a través de un puerto específico (Figura 2-3). Python tiene una librería que nos permite trabajar directamente con este componente de red.

```python
# El programa cliente que se comunica con el Servidor Software que está escuchando en el Servidor Físico
import socket

# El host remoto. Sería "localhost" si el cliente está en la misma máquina que el servidor
HOST = "85.208.20.119"

# El puerto en el que está escuchando nuestro servidor
PORT = 50000


if __name__ == "__main__":
        print('[Cliente 1] Iniciando socket con el host "{}" al puerto {}'.format(HOST, PORT))
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as mi_socket:
                tupla_para_el_enlace = (HOST, PORT)

                print('[Cliente 2] Conectando a: {}'.format(tupla_para_el_enlace))
                mi_socket.connect(tupla_para_el_enlace)

                mantener_el_socket_abierto = True
                while mantener_el_socket_abierto:
                        dato_para_enviar_str = input("[Cliente 3] Escribe el dato a enviar al servidor: ")

                        if dato_para_enviar_str == "cerrar":
                                print("[Cliente 6] Cerrando socket desde cliente")
                                break

                        mi_socket.sendall(bytes(dato_para_enviar_str, encoding='utf8'))

                        print('[Cliente 4] Datos enviados, esperando respuesta del servidor ...')
                        dato_recibido_en_bytes = s.recv(1024)
```

Figura 2-3 Ejemplo de uso de la librería socket en un Script [12]

2.1.2 Jupyter Notebook

Jupyter Notebook [13] es una aplicación web que permite crear documentos, también llamados cuadernos (*notebooks)* con celdas *Code* para desarrollar código y celdas *Markdown* que contienen texto, ecuaciones y visualizaciones. Los cuadernos pueden ser editados sin conexión a Internet desde el navegador web seleccionado (Figura 2-4). Es muy utilizado para crear documentos para el campo de ciencia de datos: simulaciones, visualizaciones, aprendizaje automático, etc. Por lo general, los documentos se guardan con la extensión *.ipynb*, aunque también se pueden convertir a otros formatos como *PDF*, *HTML*, *Markdown* o *ReStructuredText*, lo que los hace fáciles de intercambiar por correo electrónico, nubes, páginas web…

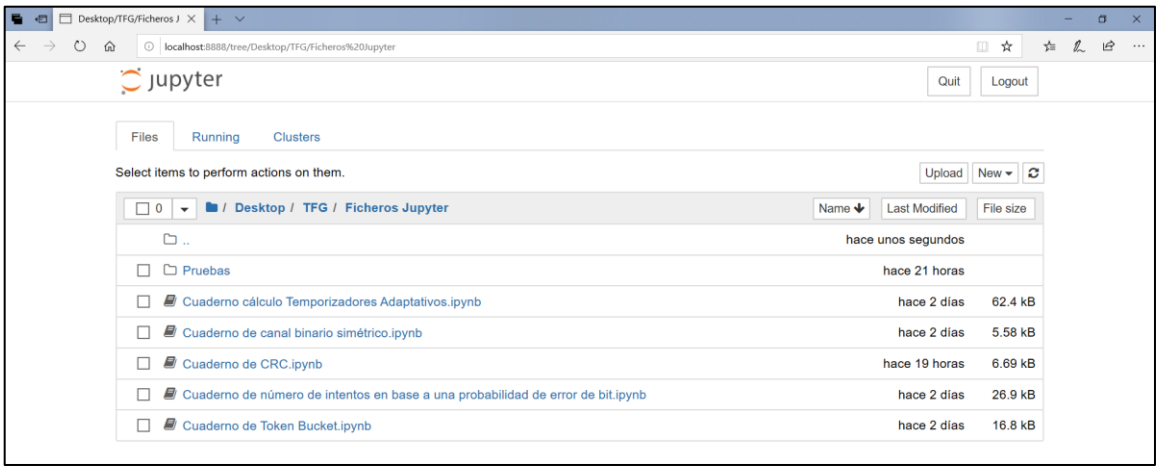


Figura 2-4 Aplicación Jupyter Notebook (elaboración propia)

Jupyter es el proyecto en el que se basa Google Colaboratory [14], proyecto de investigación lanzado por Google que nos permite compartir fácilmente cuadernos (Figura 2-5). Su objetivo es fomentar la educación y la exploración en el área del aprendizaje automático.

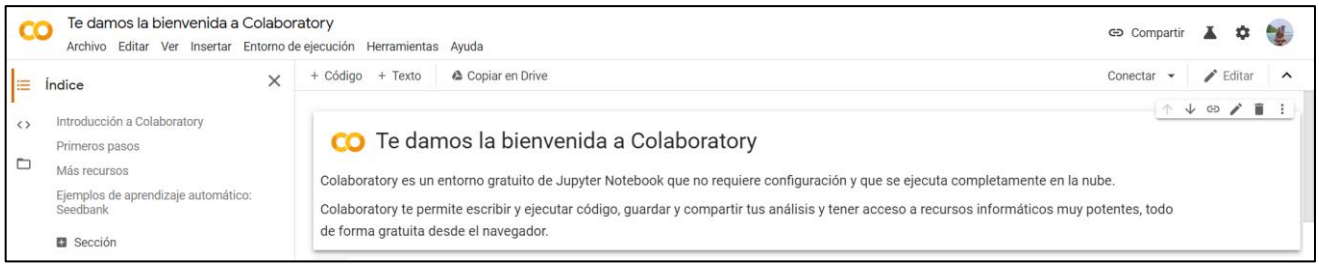


Figura 2-5 Google Colaboratory [14]

El empleo de cuadernos como apoyo en entornos docentes está creciendo. Esto se debe a la gran flexibilidad que ofrecen y la posibilidad de separar el documento del núcleo de cálculo, permitiendo abandonar la instalación local. El profesorado puede usar su propio ordenador para proporcionar a sus estudiantes presentaciones, ejercicios interactivos o lecciones y el alumno las visualiza. De esta manera se pueden generar cuadernos ricos en contenido que el alumno puede visualizar incluso en su tiempo libre. Para los estudiantes de ciencias es de gran utilidad ya que estos en algún momento van a tener que trabajar con programas de cálculo y un cuaderno puede servirle para que se vaya familiarizando con los conceptos.

2.2 Herramientas similares para la docencia de redes de ordenadores

Una herramienta muy utilizada [15] para el estudio de diseño de redes de ordenadores son los simuladores (Figura 2-6). Los simuladores permiten el estudio de conceptos y fundamentos de redes sin un gasto económico innecesario o una limitación de equipos. Los simuladores se pueden clasificar según si permiten el uso de diferentes topologías y, dentro de estos, si permiten el uso de diferentes protocolos. De entre los que trabajan con la topología que define el usuario, pero sólo trabajan con TCP/IP, destacan: **KivaNS** [16]**, Marionnet** y **PSimulator2.** Otros más complejos sí permiten y trabajan con un abanico más amplio de protocolos. Ejemplos de estos son **GNS3** [17] **y CORE** [18]**.**

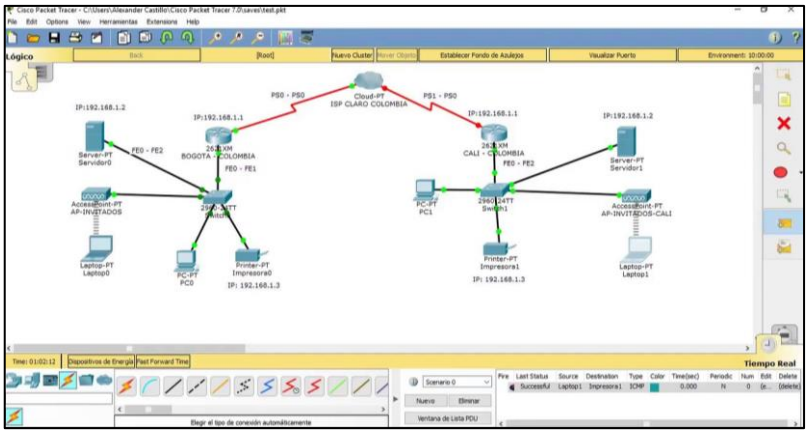


Figura 2-6 Diseño de redes de ordenadores con simulador [19]

2.2.1 CORE

CORE *Common Open Research Emulator* (emulador público de investigación) [18] es una herramienta para emular redes (Figura 2-7) que desarrolló un grupo de investigación de la empresa Boeing. Esta herramienta está escrita en Tcl/Tk [20] y ha sido desarrollada para FreeBSD y Linux. Consiste en una interfaz gráfica que permite dibujar topologías de máquinas virtuales y crear módulos de Python para desarrollar secuencias de comandos que ejecuten la simulación de la red. CORE también permite conectar estas redes emuladas a redes reales, ya que permite crear escenarios híbridos con las especificaciones de *hardware* real y nodos ficticios.

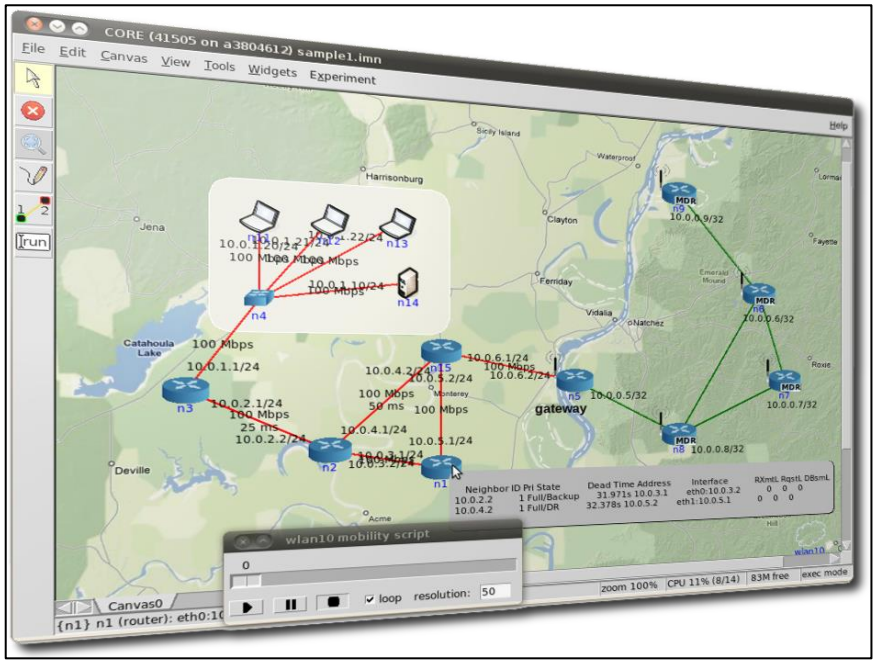


Figura 2-7 Simulación de redes con CORE [18]

2.2.2 GNS3

GNS3 [17] es un simulador gráfico desarrollado por Jeremy Grossman [21] con el que se pueden diseñar y configurar topologías de red complejas y realizar simulaciones (Figura 2-8), permitiendo el testeo sin la necesidad de *hardware* físico. Es utilizado como herramienta para la docencia (de hecho, es usado en las prácticas de laboratorio de la asignatura) ya que permite crear laboratorios virtuales que simulen los escenarios reales, de manera que el estudiante pueda reproducir los entornos desde su dispositivo personal de manera simple e intuitiva. También permite, al igual que CORE, conectar dispositivos virtuales con equipos reales. GNS3 es una plataforma de *software* libre escrito en Python y puede ser instalado en Windows, Mac OSX y Linux.

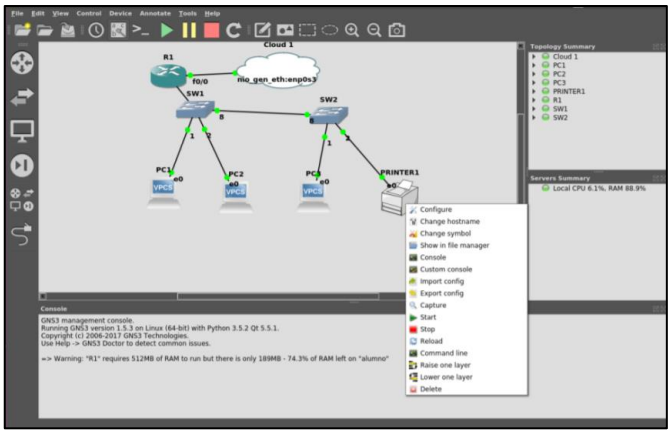


Figura 2-8 Diseño de topología de red con GNS3 [22]

2.2.3 KivaNS

KivaNS [16] es una aplicación de código abierto escrita en Java que permite simular el encaminamiento de paquetes a través de los esquemas de redes que se especifiquen. Su objetivo es ayudar a comprender cómo funcionan las redes con arquitectura TCP/IP. También permite el análisis de protocolos auxiliares como pueden ser ARP e ICMP (Figura 2-9). La ventaja de KivaNS frente a otros es la simplicidad de uso si no se quiere profundizar mucho en los conceptos.

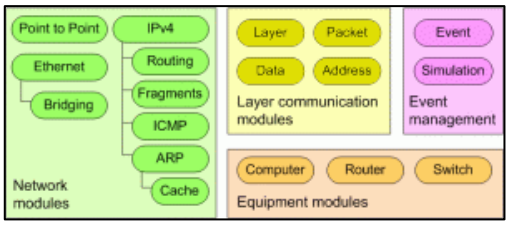


Figura 2-9 Módulos de KivaNS [23]

2.2.4 Juegos de ordenador

Existen juegos como alternativas docentes, ya que algunas universidades consideran el uso de simuladores como una actividad aburrida y desmotivadora para sus alumnos. Se pretende que los alumnos se enganchen a estos juegos a través del reto y la competición, logrando así una mayor retención de los conceptos. Miguel Arevalillo-Herráez, José M. Claver y Raúl Morán Gómez [24] han diseñado "El juego de las redes" (Figura 2-10) para conseguir estos objetivos en la Universidad de Valencia. Está basado en el juego de mesa Risk, que está basado a su vez en las guerras napoleónicas y está pensado para que el jugador desarrolle estrategias. Se ayuda así al aprendizaje de:

- Configuraciones: Asignar direcciones IP.
- Uso de los comandos *ifconfig* y *route* para asignar una IP y una puerta de enlace a una interfaz.
- Conocimiento del número de puerto estándar que suelen proporcionar los servicios comunes.

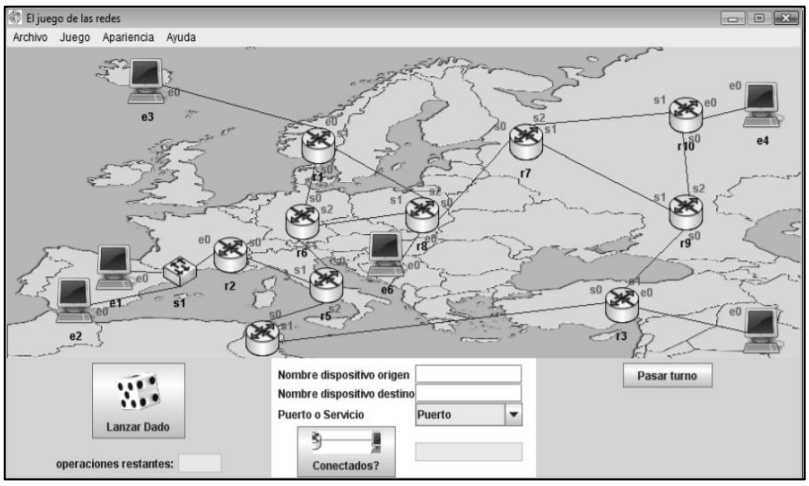


Figura 2-10 El juego de las redes [24]

La empresa Cisco, productora de encaminadores, también ha desarrollado una línea de juegos que se encuentran disponibles dentro de su plataforma Cisco Learning [25]. De estos juegos destacan Network Defenders y Cisco Aspire CCNA.

En Network Defenders [26] tienes que proteger la red de diferentes ataques que van incrementándose en número y complejidad conforme avanza el juego (Figura 2-11). Se facilita al usuario el aprendizaje de los diferentes ataques que existen.

En Cisco Aspire CCNA [27] se juega a ser un técnico que tiene que reparar los problemas de red de los ciudadanos (Figura 2-12). El técnico tiene acceso a la configuración de red, *hardware* y programas con un alto nivel de detalle técnico, de tal manera que ha de ser el usuario el que se las ingenie tomando diferentes soluciones para solucionar el problema. El juego permite realizar una evaluación del rendimiento y proporcionar una calificación del usuario.

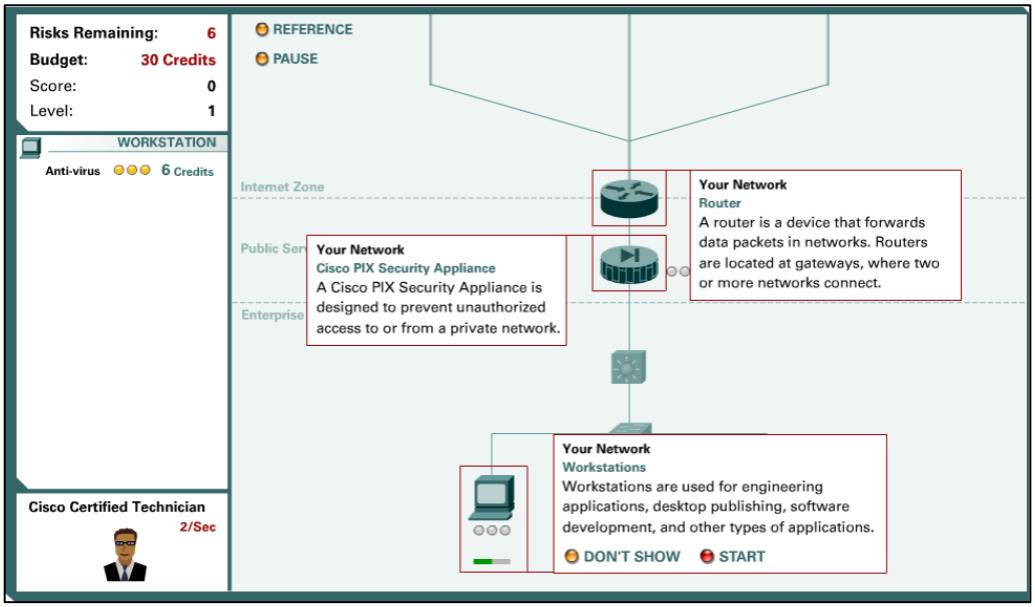


Figura 2-11 Network Denfenders [25]

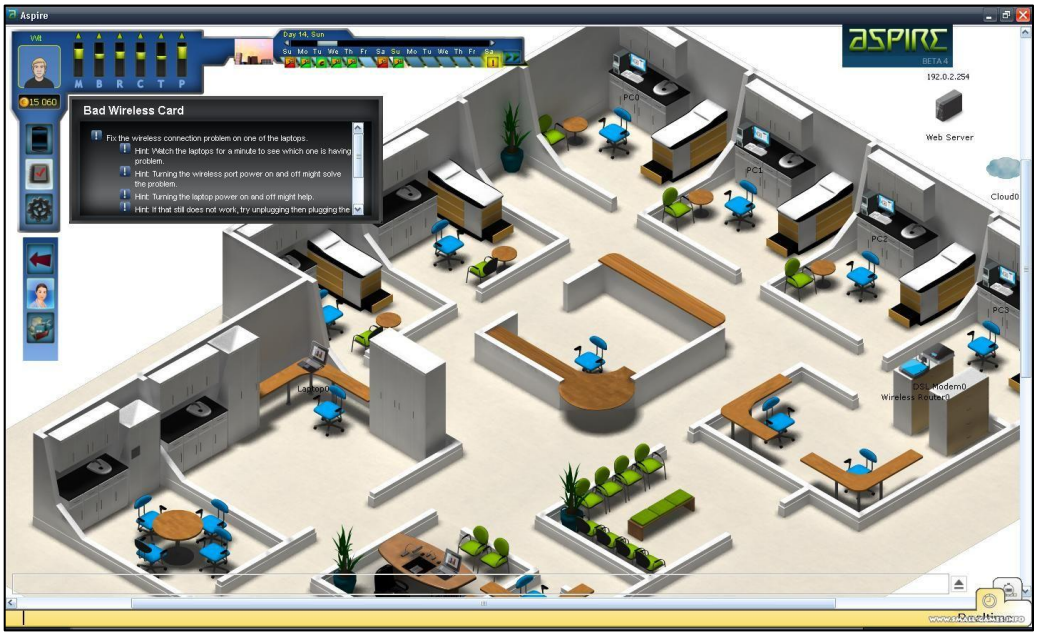


Figura 2-12 Cisco Aspire CNNA [25]

2.2.5 Wireshark

Wireshark es el analizador de protocolos más conocido y usado [28]. Un analizador de protocolos es un *sniffer* [29] (herramienta que analiza el tráfico de una red en tiempo real) al que se le han implementado funcionalidades que permiten entender y traducir los protocolos que se están empleando en la red, presentándolos en un formato legible para las personas (Figura 2-13).

La popularidad de esta herramienta reside en [30]:

- La diversidad de funcionalidades: soporte a una amplia variedad de protocolos.
- Interfaz de usuario: interfaz intuitiva.
- Coste: *software* de código libre y gratuito.
- Soporte: tutoriales, ayuda e información accesible.
- Disponibilidad: versiones para diferentes sistemas operativos (Linux, Windows y Mac OS).

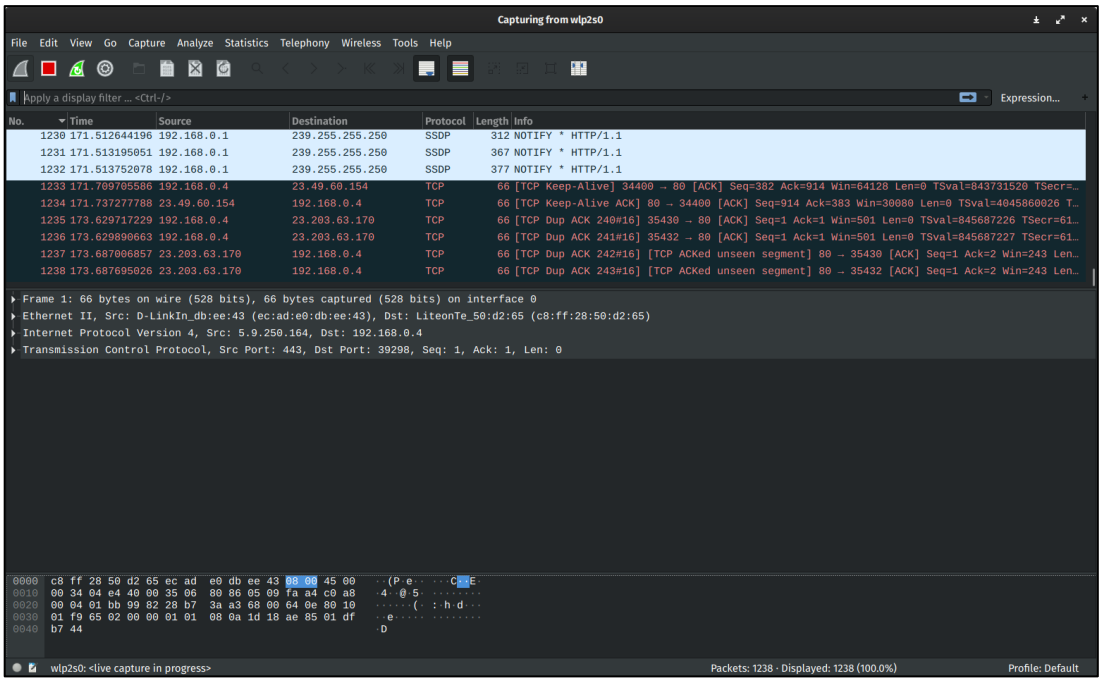


Figura 2-13 Análisis del tráfico de una red con Wireshark [31]

2.3 Herramientas docentes desarrolladas con Jupyter Notebook

En Internet podemos encontrar varias páginas web que han recopilado cuadernos para enseñanza desarrollados en Jupyter Notebook con Python por diferentes autores. El mayor abanico de cuadernos para docencia se encuentra en la página web GitHub.com [32]. En GitHub los podemos encontrar incluso categorizados según el campo: informática, estadística, matemáticas, física, biología, química, geología, lingüística, etc. Entre la colección, se han querido destacar los indicados en las siguientes secciones.

2.3.1 Notebooks para aprendizaje de Python

Los dos siguientes cuadernos, de autores desconocidos, describen de manera muy concisa todo lo necesario para aquellos usuarios que van a comenzar con Python desde cero. El primer cuaderno, ¿Cómo comenzar con Python? [33], dispone de cuatro apartados realizados con celdas *Markdown* en los que explica por orden las instalaciones necesarias y proporciona enlaces a tutoriales y vídeos para aprender el lenguaje (Figura 2-14).

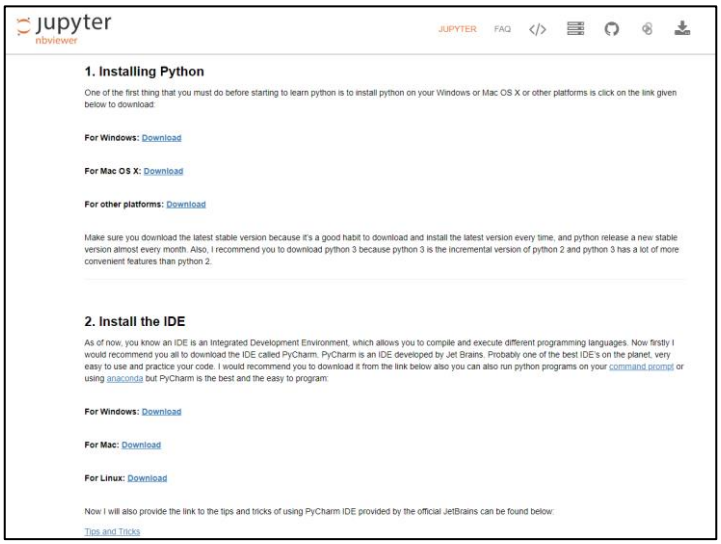


Figura 2-14 Cuaderno ¿Cómo comenzar con Python? [33]

El segundo cuaderno, Python [34], se ha desarrollado como tutorial para el aprendizaje de la sintaxis del lenguaje. Como muestra la Figura 2-15, está dividido mediante títulos en apartados en los que se describe cómo se emplea la aritmética, bucles, listas, etc. de Python alternando celdas *Markdown* y celdas *Code*. También dispone de apartados de ejercicios para que el usuario pueda poner en práctica sus conocimientos y simular un código relacionado con los apartados que se han estudiado.

Figura 2-15 Cuaderno Python [34]

2.3.2 Procesado de audio aplicando la transformada de Fourier

Esta colección de diez notebooks subidos a la página web GitHub por el usuario Caleb Madrigal [35] explica cómo se emplea la transformada de Fourier para casos básicos de procesado de sonido (Figura 2-16). Así, el usuario interesado parte desde una introducción de los conceptos de onda de sonido y de la Transformada de Fourier durante los tres primeros cuadernos, estudia diferentes nociones de procesado con los seis siguientes y finaliza con una conclusión de todos los conceptos.

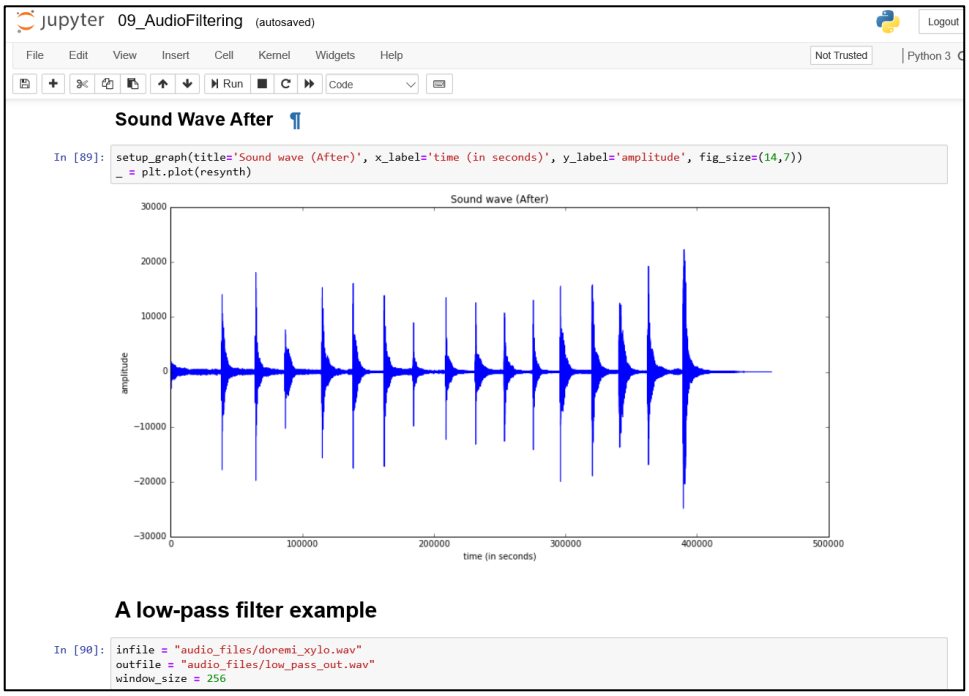


Figura 2-16 Cuaderno Procesado de audio [36]

Cada uno de los cuadernos describe mediante celdas *Markdown* cada uno de los conceptos que van a ser posteriormente graficados o simulados mediante códigos. También dispone de celdas que reproducen sonidos según el procesado de la señal. Para todo esto, hace uso de las librerías Spicy, Numpy, Matplotlib e Ipython y define reiteradamente funciones al principio de cada uno de los cuadernos para simplificar el código entre las posteriores celdas.

2.3.3 Introducción al análisis de ingeniería química

Introducción al análisis de ingeniería química [37] es una herramienta para el estudio de ingeniería química realizada por Jeff Kantor, profesor en la Universidad de Notre Dame. Consta de un repositorio de más de cuarenta cuadernos divididos según los capítulos de la asignatura: Unidades y cálculos, estequiometría, ajustes y balances de flujos, reacciones, gases y vapores, balances energéticos (Figura 2-17) y equilibrios de gases y vapores. Se han querido destacar dos: Propulsión de torpedos [38] y Balances energéticos de una turbina [39].

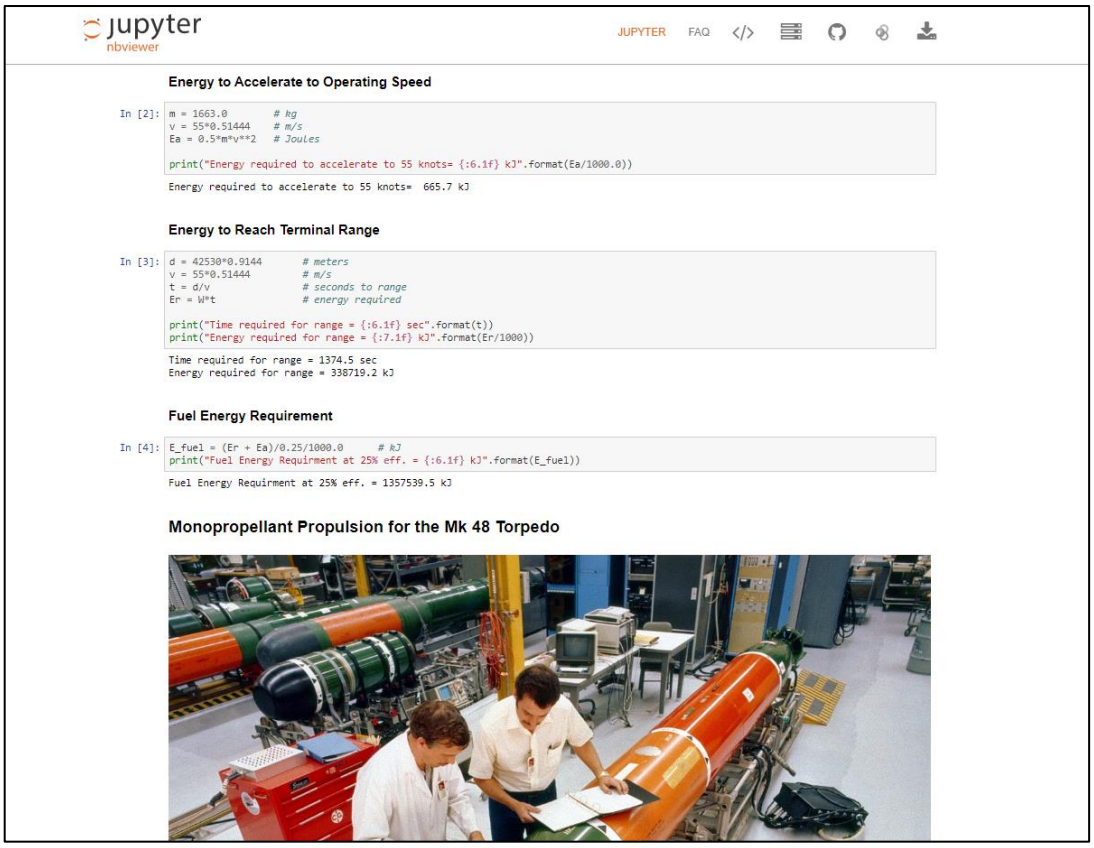


Figura 2-17 Cuaderno Propulsión de torpedos [38]

La sintaxis de los cuadernos es relativamente poco compleja, haciéndola más atractiva para los alumnos. Se hace uso de variables escalares y de listas con éstas, de funciones aritméticas creadas con los comandos *def* y *lambda* y de las librerías Math, Matplotlib, Numpy, Sympy y Spicy.

2.3.4 Números imaginarios

El matemático Caleb Fangmeier realizó el cuaderno Números imaginarios [40] con los conocimientos que había adquirido haciendo un curso de matemáticas aplicadas para ayudar al resto de compañeros a comprender los números complejos. Aunque Caleb haya hecho un uso de un lenguaje coloquial (incluso inadecuado) en algunas celdas *Markdown*, el cuaderno desarrollado es bastante completo y resulta muy atractivo para la persona interesada ya que se ayuda de numerosas gráficas para facilitar la comprensión y los comentarios mencionados terminan siendo graciosos (Figura 2-18). Caleb también ha realizado un cuaderno en Jupyter que ha llamado MatPotBoard [41]. Como bien indica su nombre, el cuaderno permite generar *HTMLs* de cuadros de mando con *Matplotlib*.

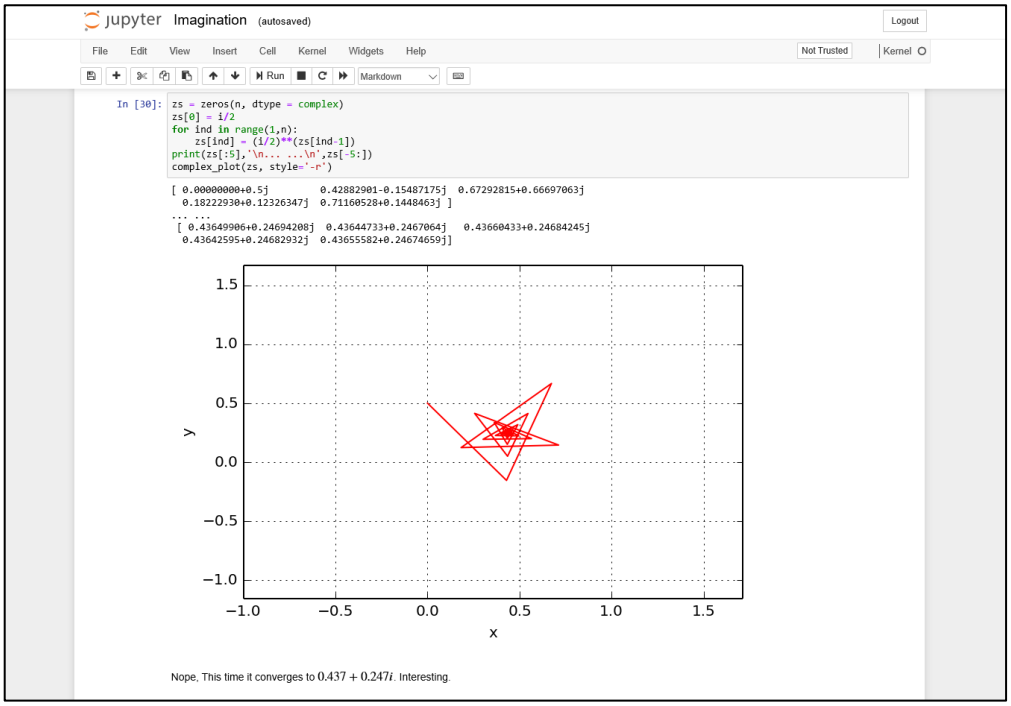


Figura 2-18 Cuaderno Números imaginarios [40]

2.3.5 Sensores y actuadores

Sensores y actuadores [42] es un conjunto de cuadernos realizados por Andrés Marrugo, profesor de la Universidad Tecnológica de Bolívar para los alumnos del curso IMTR 1713 de sensores. Aunque todavía está en desarrollo, ya cuenta con ocho cuadernos que abarcan los conceptos del curso, como son: sensores activos y pasivos, errores y características de los sensores (Figura 2-19) y pérdida de potencia en las fibras ópticas. No hay nada nuevo relacionado con el lenguaje con respecto a los anteriores.

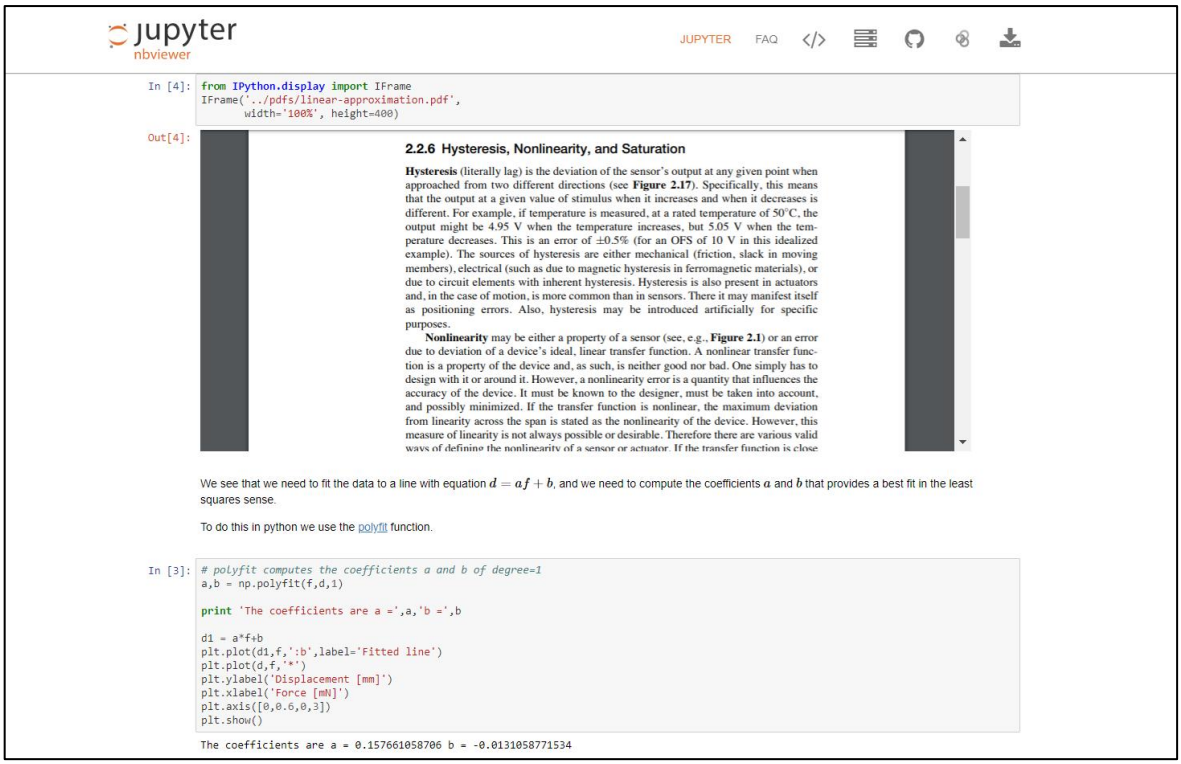
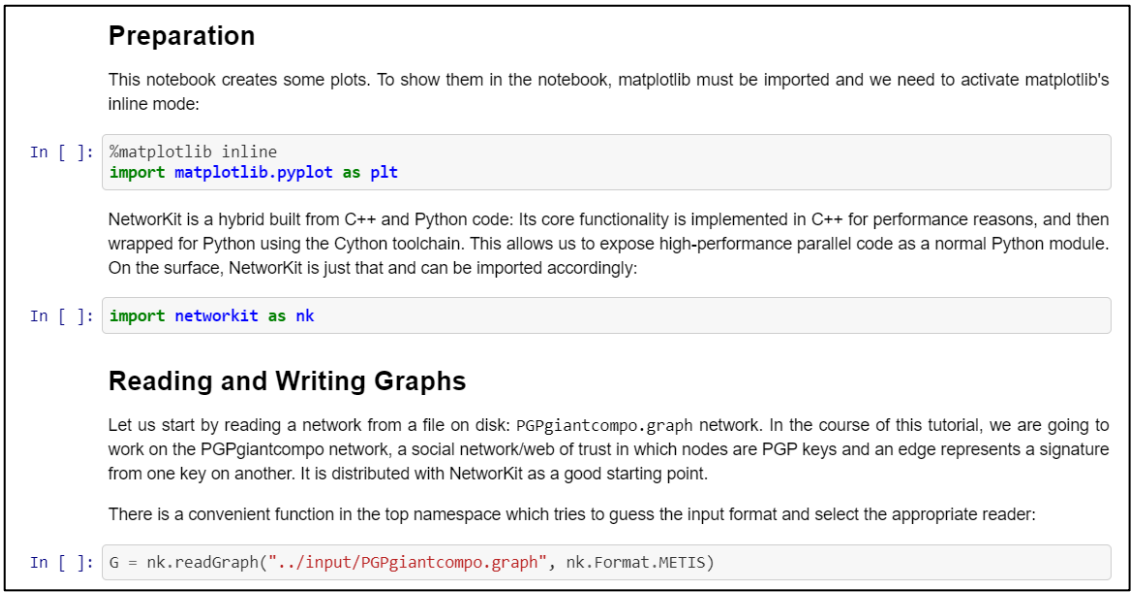


Figura 2-19 Cuaderno Función transformada de un sensor de posición [43]

2.4 Cuadernos relacionados con Redes

2.4.1 Guía de usuario de Networkit

Networkit [44] es un kit de herramientas de código abierto para el análisis de redes a gran escala. Las redes pueden llegar a ser muy complejas y resultan un objetivo desafiante ya que contienen miles de millones de conexiones. Para el análisis de estas redes, *Networkit* implementa algoritmos de grafos eficientes. En GitHub existen numerosos cuadernos de *Networkit* para el análisis de redes. Un ejemplo es Guía de usuario de Networkit [44], un cuaderno desarrollado como introducción interactiva a las características de *NetworKit* (Figura 2-20), que consta de texto y código ejecutable para explicar los conceptos ayudándose de gráficas.



Preparation

This notebook creates some plots. To show them in the notebook, matplotlib must be imported and we need to activate matplotlib's inline mode:

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

NetworKit is a hybrid built from C++ and Python code: Its core functionality is implemented in C++ for performance reasons, and then wrapped for Python using the Cython toolchain. This allows us to expose high-performance parallel code as a normal Python module. On the surface, NetworKit is just that and can be imported accordingly:

In [ ]:
```
import networkit as nk
```

Reading and Writing Graphs

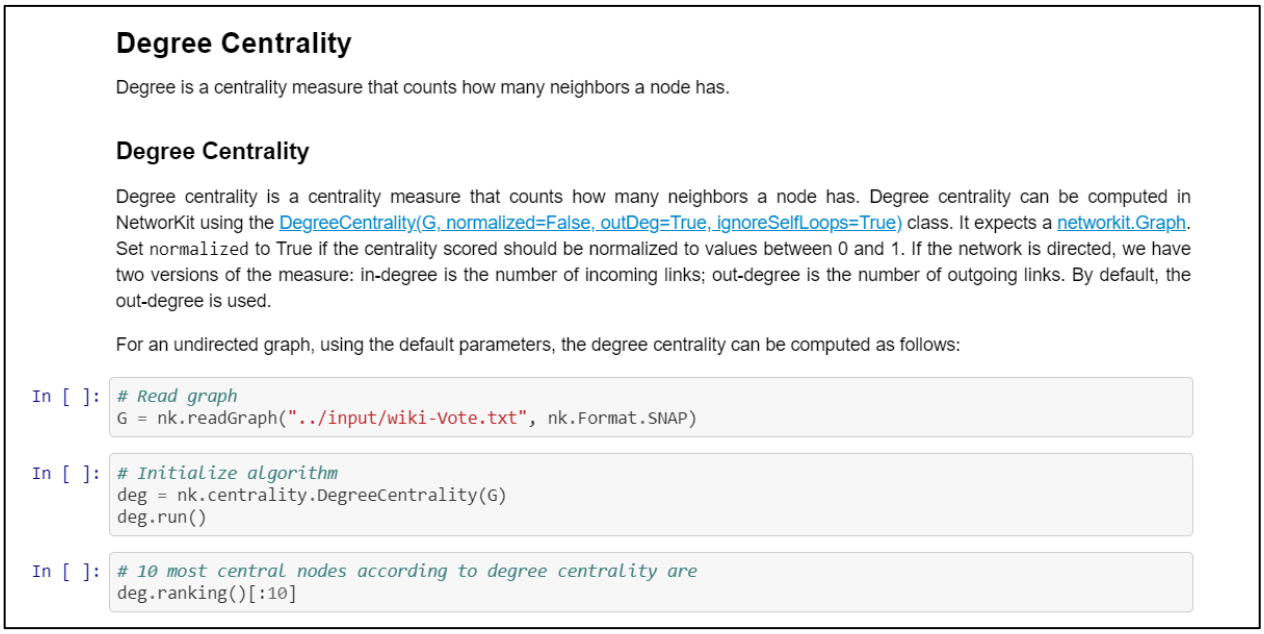
Let us start by reading a network from a file on disk: `PGPgiantcompo.graph` network. In the course of this tutorial, we are going to work on the PGPgiantcompo network, a social network/web of trust in which nodes are PGP keys and an edge represents a signature from one key on another. It is distributed with NetworKit as a good starting point.

There is a convenient function in the top namespace which tries to guess the input format and select the appropriate reader:

In [ ]:
```
G = nk.readGraph("../input/PGPgiantcompo.graph", nk.Format.METIS)
```

Figura 2-20 Cuaderno Guía de usuario de Networkit [44]

Otro cuaderno de *Networkit* es Tutorial de centralidad [45]. Este cuaderno cubre algunos de los algoritmos de centralidad implementados en *NetworKit* (Figura 2-21). La centralidad mide la importancia de un nodo dentro de un gráfico. El código para el análisis de centralidad en *NetworKit* se agrupa en el módulo de centralidad.



Degree Centrality

Degree is a centrality measure that counts how many neighbors a node has.

Degree Centrality

Degree centrality is a centrality measure that counts how many neighbors a node has. Degree centrality can be computed in NetworKit using the DegreeCentrality(G, normalized=False, outDeg=True, ignoreSelfLoops=True) class. It expects a networkit.Graph. Set `normalized` to True if the centrality scored should be normalized to values between 0 and 1. If the network is directed, we have two versions of the measure: in-degree is the number of incoming links; out-degree is the number of outgoing links. By default, the out-degree is used.

For an undirected graph, using the default parameters, the degree centrality can be computed as follows:

In [ ]:
```
# Read graph
G = nk.readGraph("../input/wiki-Vote.txt", nk.Format.SNAP)
```

In [ ]:
```
# Initialize algorithm
deg = nk.centrality.DegreeCentrality(G)
deg.run()
```

In [ ]:
```
# 10 most central nodes according to degree centrality are
deg.ranking()[:10]
```

Figura 2-21 Cuaderno Tutorial de centralidad [45]

2.4.2 NetworkX

NetworkX [46] es una biblioteca de Python para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas. Permite:

- Establecer clases para gráficos.
- Convertir grafos a o desde otros formatos.
- Construir grafos aleatorios o construirlos de manera incremental.
- Encontrar subgrafos y núcleos.
- Analizar adyacencia, grado, tamaño, etc.
- Dibujar redes en 2D y 3D.

Al igual que con *Networkit*, en GitHub podemos encontrar muchos cuadernos de *Networkx* para el estudio de redes. Un ejemplo de estos cuadernos es Análisis de redes con Networkx [47]. En la Figura 2-22 puede verse que, aunque es un cuaderno simple, es más que suficiente para introducirse y afianzar conceptos sobre como representar gráficamente con Python.

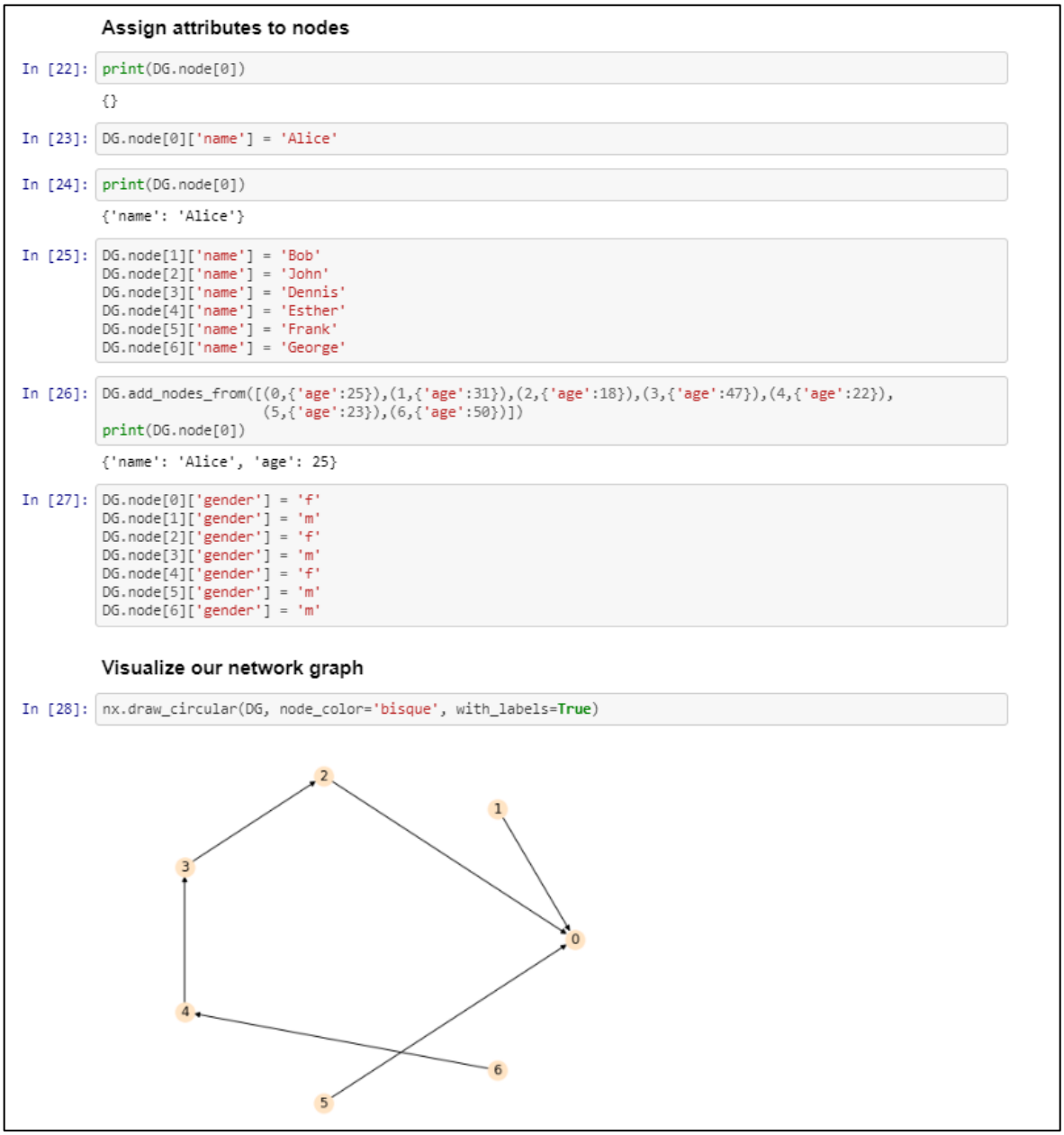


Figura 2-22 Cuaderno Network Analysis with NetworkX [47]

3 DESARROLLO DEL TFG

En este capítulo se describirá la instalación de las herramientas utilizadas, el desarrollo del conjunto de cuadernos que componen la herramienta de apoyo para la asignatura y se justificarán las decisiones tomadas durante su desarrollo.

3.1 Instalación de las herramientas con Anaconda

Aunque existan otras alternativas, la instalación de las herramientas que han sido utilizadas en este trabajo se ha realizado con Anaconda. Anaconda [48] es una plataforma de código abierto para aplicaciones de ciencia de datos con Python que puede ser descargada para diferentes sistemas operativos. Permite instalar y administrar paquetes y entornos de manera sencilla, lo que resulta más eficiente. Desde la página web oficial de Anaconda [49] podemos descargar el fichero .exe (para sistemas operativos Windows). Como aparece indicado en la Figura 3-1, para el desarrollo de este proyecto se ha descargado la versión de Python 3.7.

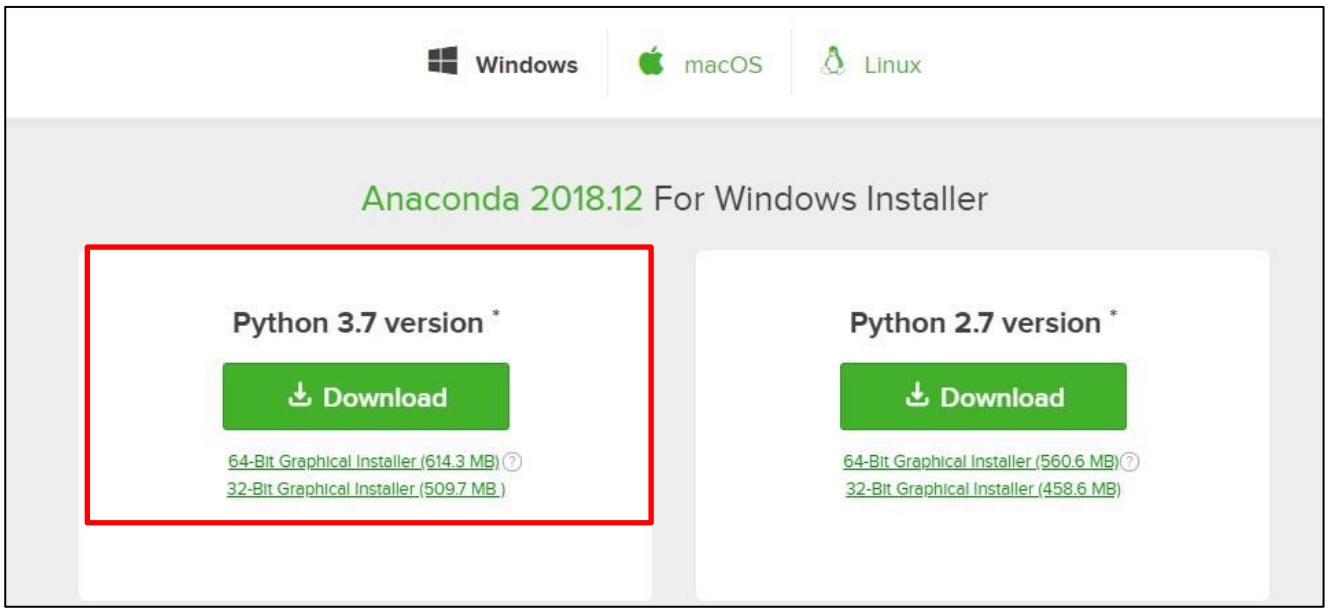


Figura 3-1 Descarga de Anaconda [49]

Tras la descarga del fichero *.exe* de la versión seleccionada, sólo con ejecutarlo bastará. El *setup* se encarga de toda la instalación (Figura 3-2). La ubicación predeterminada de instalación en Windows es *C:/Anaconda*.

Figura 3-2 Instalación de Anaconda (elaboración propia)

La distribución de Anaconda (Figura 3-3) se divide en 4 partes: Conda, librerías de datos, Anaconda Project y Anaconda Navigator. Después de la instalación dispondremos de todas estas ellas ya configuradas, aunque también las podremos configurar desde la interfaz gráfica Anaconda Navigator.

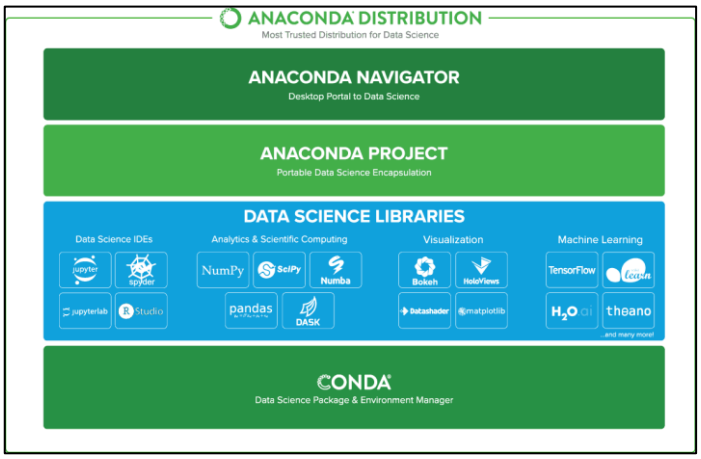


Figura 3-3 Anaconda Distribution [48]

La instalación de Jupyter Notebook es tan simple como abrir Anaconda Navigator (interfaz gráfica de usuario de escritorio) y hacer clic en la casilla de Jupyter Notebook, tal y como se muestra en la Figura 3-4. Tras su instalación y cada vez que se haga clic se abrirá una pestaña en el navegador web y se podrá comenzar a trabajar.

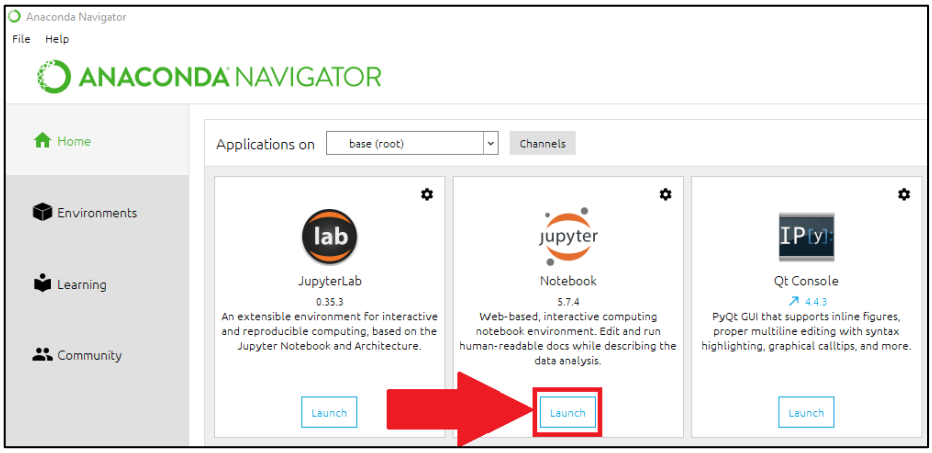


Figura 3-4 Instalación de Jupyter Notebook desde la consola [50]

3.2 Selección de los conceptos para realizar los notebooks

Debido a que la asignatura engloba un gran abanico de conceptos, se realizó una encuesta mediante Formularios de Google [51] a los alumnos de quinto curso para que seleccionasen los conceptos en los que consideraban que hubiese sido de utilidad para su comprensión una herramienta interactiva. Se obtuvieron un total de 24 respuestas, como se muestra en la Figura 3-5.

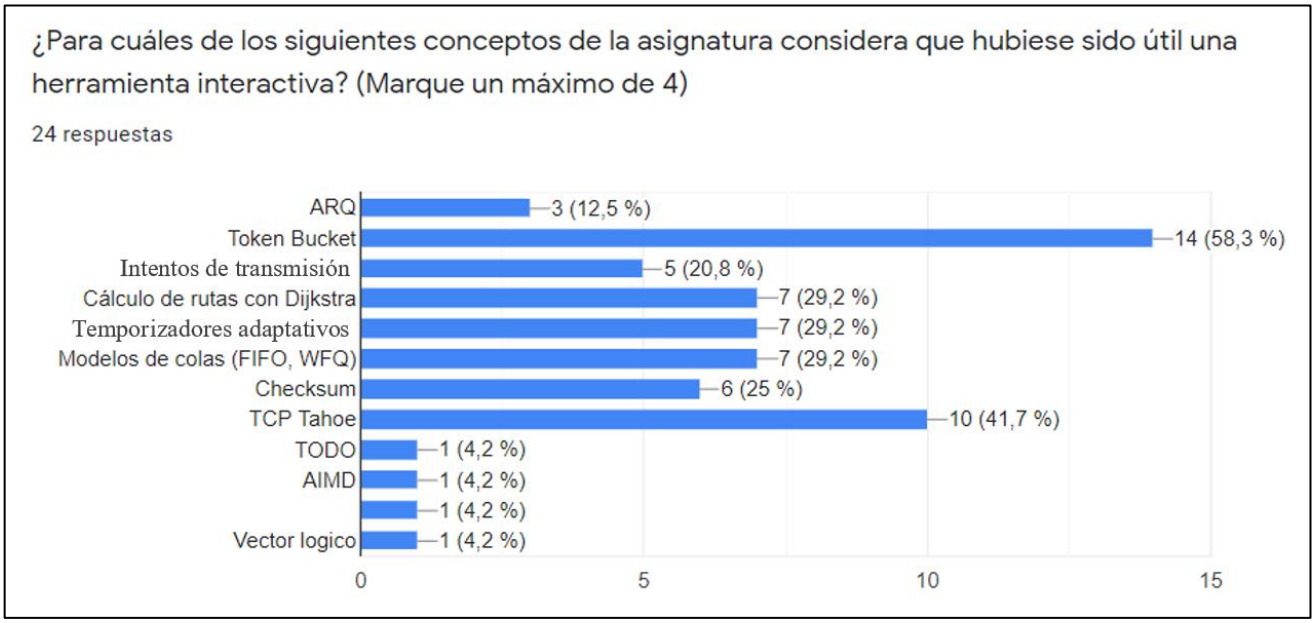


Figura 3-5 Encuesta realizada [51]

Se tomó como objetivo el desarrollo de cinco cuadernos. El profesorado propuso el desarrollo de un cuaderno de CRC (*Cyclic Redundancy Check*) y otro de *Canal Binario Simétrico* (los profesores de la asignatura han notado durante este curso académico que estos dos conceptos han sido algunos de los más problemáticos). A la vista de los resultados de la encuesta, aunque no llegasen a ser los más votados, se decidió realizar los cuadernos de *Token Bucket*, *Cálculo del RTO en temporizadores adaptativos* y *Número de intentos de transmisión en base a una probabilidad de bit*. La encuesta no se tomó como un factor determinante en la elección, sino como apoyo, combinando sus resultados con las opiniones de los tutores del trabajo, docentes ambos este curso de la asignatura.

3.3 Desarrollo de los notebooks

En este apartado se describirá el desarrollo de los cuadernos según su orden cronológico de realización. Todos los cuadernos realizados, constan, en el siguiente orden, de:

- Un título redactado en una celda *Heading*.
- Una explicación del concepto con texto e imágenes (introducidas con URLs de Internet) en celdas *Markdown*.
- Una celda código donde se ha desarrollado el código con Python para que el alumno interactúe introduciendo los datos y obtenga resultados.
- Una última celda con las referencias utilizadas.

3.3.1 Notebook del Canal Binario Simétrico

El primer cuaderno desarrollado ha sido el del Canal Binario Simétrico. El haber desarrollado este cuaderno en primer lugar se debe a que se consideró que la sintaxis del código de este cuaderno era la de mayor simplicidad y a que era uno de los cuadernos de mayor importancia para el profesorado ya que era uno de los conceptos que causó más problemas al ser impartido en clase.

En la Figura 3-6 se muestra la explicación del concepto de Canal Binario Simétrico que se ha redactado en el cuaderno.

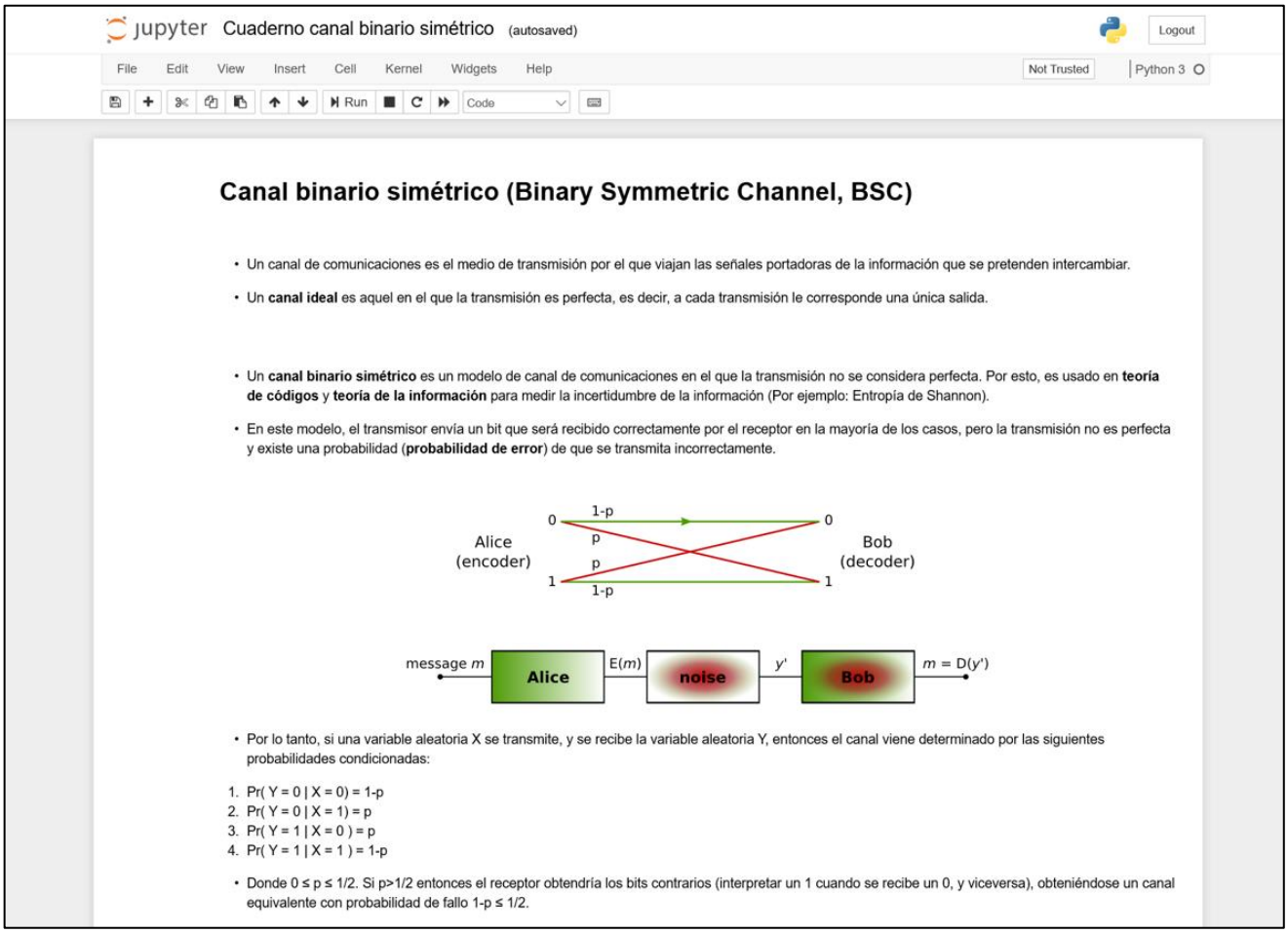


Figura 3-6 Explicación del concepto mediante celdas Heading y Markdown (elaboración propia)

El código desarrollado tiene una primera sección para importar las librerías utilizadas (Figura 3-7). En este caso, la única librería utilizada ha sido *random.* Esta librería ha sido empleada, como veremos más adelante, para generar errores de naturaleza aleatoria.

```
#Librerías
import random
```

Figura 3-7 Librería utilizada en el notebook Canal Binario Simétrico (elaboración propia)

La segunda parte del código (Figura 3-8) consta de dos secciones para que el usuario introduzca la trama a transmitir y la probabilidad de error del canal por el que se va a transmitir la información. En el caso de la probabilidad de error, se emplea la función float para convertir el texto de entrada a un número real y la función *input* para almacenar la probabilidad de error en la variable "Per". Se ha implementado un bucle con la función *while* para que le pida al usuario que introduzca de nuevo la probabilidad de error en el caso en el que este se equivoque e introduzca una incorrecta.

```
#Usuario introduce probabilidad de error correctamente
Per = float(input("Introduzca probabilidad de error: "))
while Per > 0.5 or Per < 0:
    print("P.error ha de tener un valor comprendido entre 0 y 1/2")
```

```
Per = float(input("Introduzca probabilidad de error: "))
```

Figura 3-8 Código para que el usuario introduzca probabilidad de error (elaboración propia)

Para la secuencia de bits que desee introducir (Figura 3-9), se ha usado la función *int* y la función *input*, indicándole al final de la función *int* que el número que se va a introducir es binario (base 2), de tal manera que lo almacene como un entero en la variable "entrada". Se ha creado un bucle *for* para que genere la lista "bits" con el número en binario que había introducido el usuario convirtiendo el valor entero de la variable "entrada" a dígitos en binario.

```
#Datos de entrada
print()
entrada = int(input("Introduzca manualmente secuencia de bits: "),2)
bits = [int(x) for x in bin(entrada)[2:]]
```

Figura 3-9 Código para que el usuario introduzca secuencia de bits (elaboración propia)

La siguiente sección (Figura 3-10) crea una lista cuyos valores son generados en base a la probabilidad que el usuario ha introducido. Para ello, se recorren cada uno de los elementos (bits) de la lista "bits" y se introducen aleatoriamente errores en base a la probabilidad especificada en la lista nueva "bitsconerror". Esto simula la transmisión de la secuencia de bits por el canal con ruido. Si la lista "bitsconerror" es diferente a la lista "bits" es porque se ha producido un error en la transmisión. Cuanto mayor sea la probabilidad de error o número de bits introducidos, mayor serán las posibilidades de que la lista nueva sea diferente. Para todo esto, primero se ha generado la variable "q" que es la probabilidad de que se transmita correctamente un bit y la lista "bitsconerror". Para generar los valores aleatorios y añadirlos, se ha implementado un bucle *for* que recorre la lista "bits". Dentro del bucle *for*, si el número aleatorio generado con la función *random.random* es menor que la probabilidad de acierto, se añade el bit correcto a la lista "bitsconerror". En el caso contrario, si el número generado es mayor que "q", se introduce un error. Esto simula el bit en el que se produce error.

```
#Errores en el canal de transmisión
errores = 0
q = 1 – Per
bitsconerror = []
for bit in bits:
    if random.random() < q:
        bitsconerror.append(bit)
    else:
        if bit == 0:
            bitsconerror.append(1)
        else:
            bitsconerror.append(0)
        errores = errores + 1
```

Figura 3-10 Código para generar secuencia con errores (elaboración propia)

El resto del código muestra en pantalla todo el proceso. En el primer bloque se ha creado una nueva variable que se ha llamado “salida”, para unir los bits de la lista “bitsconerror”. Se imprime en pantalla con la función *print* la secuencia que el usuario había introducido y la secuencia de datos (“salida”) con el posible error. Mediante la condición *if*, se comparan las listas (Figura 3-11). Si las listas son iguales se muestra en pantalla un mensaje que indica que no hubo error y, en el caso contrario, un mensaje con que sí lo hay. Se imprime en pantalla una lista encima de la otra para que le sea al usuario más fácil ver los errores y el número de errores.

```
#Mostramos en pantalla al usuario lo que ha sucedido

salida = ''.join(map(str, bitsconerror))

print("Tras la transmisión de",bin(entrada),"por el canal, la salida queda:",salida)

if bits == bitsconerror:

    print("La secuencia de bits no ha sido modificada al transmitirse por el canal")

else:

    print("La secuencia de bits ha sido modificada al transmitirse por el canal")

print()

print("La secuencia a la entrada era:", bits)

print("La secuencia a la salida es  :",bitsconerror)

print()

print("Con un error de ",errores," bits")
```

Figura 3-11 Código para comparar y mostrar las secuencias de bits (elaboración propia)

Finalmente, el usuario dispone de una celda con las referencias utilizadas (Figura 3-12) tanto para el desarrollo del código como para la redacción del texto.

Referencias

1. https://es.wikipedia.org/wiki/Canal_binario_sim%C3%A9trico
2. http://ingenieriacognitiva.com/developer/cursos/TeoriaInf/Unidad6.pdf
3. https://stackoverflow.com/questions/59800154/how-to-split-a-binary-number-into-indiviual-digits-to-make-a-list-python

Figura 3-12 Referencias relacionadas con el canal binario simétrico (elaboración propia)

3.3.2 Notebook del número de intentos de transmisión

El siguiente cuaderno realizado permite al usuario calcular el número de intentos de media que se tendrían que realizar para enviar una trama sin error. A mayor probabilidad de error de transmisión, mayor número medio de intentos. Como este número medio sigue una distribución geométrica, en la introducción del cuaderno se da una explicación tanto de la probabilidad de error de bit como de la distribución geométrica (Figura 3-13).

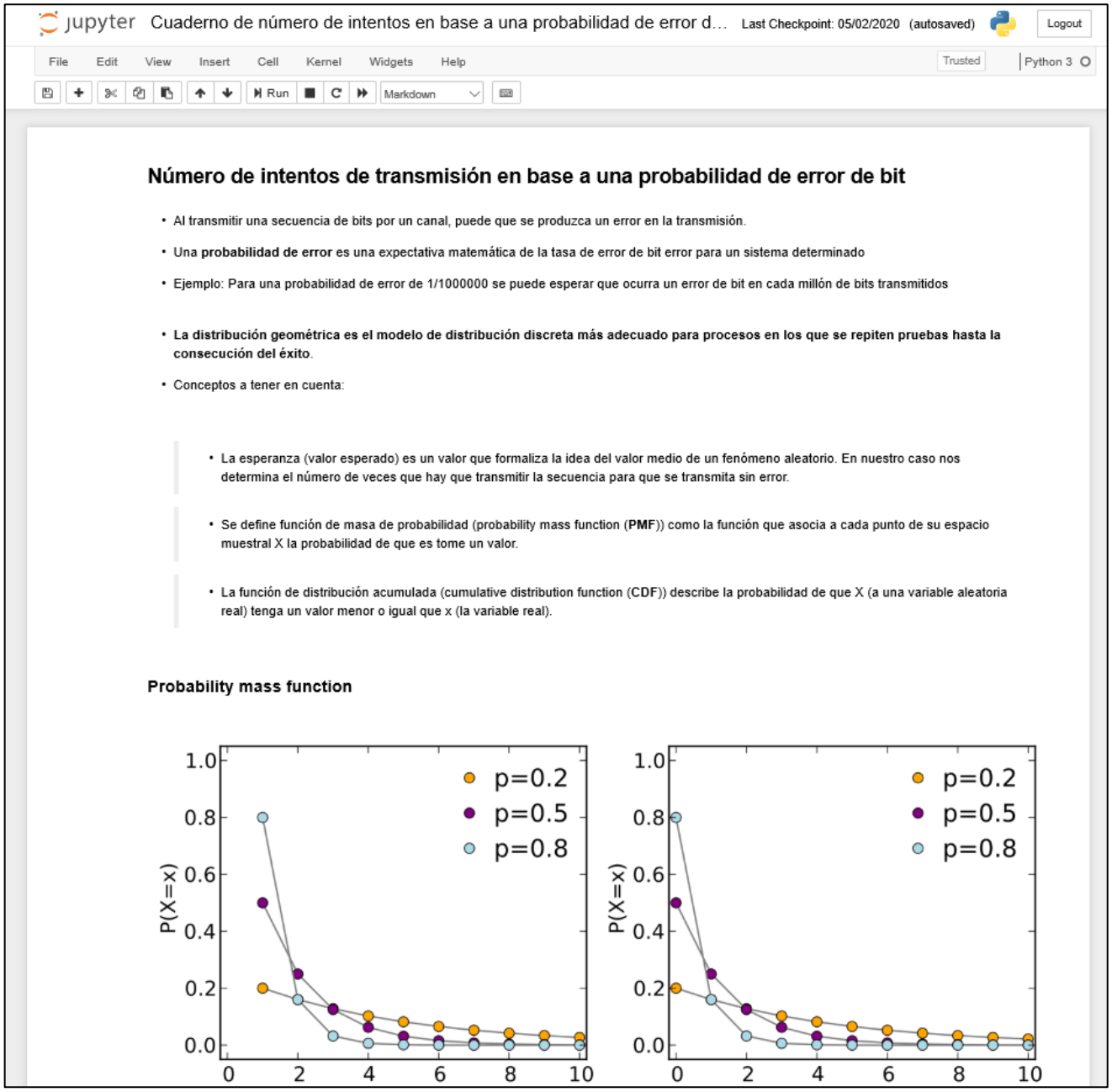


Figura 3-13 Explicación de la probabilidad de error y de la distribución geométrica (elaboración propia)

La celda código comienza con una primera relativa a librerías. Las librerías utilizadas (Figura 3-14) son *matplotlib.pyplot* para la representación de las gráficas y *numpy* para trabajar con matrices. Las abreviaturas utilizadas para facilitar el uso de las librerías son *np* y *plt* respectivamente.

```
#Librerías

import numpy as np

import matplotlib.pyplot as plt
```

Figura 3-14 Librerías del notebook del número de intentos de transmisión

El segundo bloque de código (Figura 3-15) es para que el usuario introduzca la probabilidad de transmitir el mensaje correctamente y definir las listas necesarias. La probabilidad de transmisión será almacenada en la variable "pexito" mediante los comandos *float* e *input*. Se ha añadido un bucle *while*

por si el usuario introduce datos erróneos. Se han creado las listas “listapmf” y “listacdf” para almacenar cada uno de los valores calculados de la función de masa de probabilidad (PMF) y la función de distribución acumulada (CDF).

```
#Datos y listas necesarias
pexito = float(input("Introduzca probabilidad de transmisión de mensaje correcta: "))
while pexito < 0 or pexito > 1:
        pexito = float(input("Introduzca probabilidad de transmisión de mensaje correcta: "))
listapmf = []
listacdf = []
```

Figura 3-15 Código para datos y listas

En la siguiente sección (Figura 3-16) se calcula y se imprime el número medio de intentos de transmisión del mensaje.

```
#Cálculo número de intentos
intentos = 1/pexito
redondeo = round(intentos)
print()
print("El número medio de intentos para la transmisión del mensaje es", intentos)
```

Figura 3-16 Código para el cálculo del número medio de intentos de transmisión

Si el usuario introduce una probabilidad de transmisión correcta relativamente alta, el número de intentos será muy bajo. Para que el usuario pueda observar en las gráficas los intentos que serían necesarios con esa probabilidad, se ha establecido en 10 el número mínimo de valores a representar (Figura 3-17).

```
if redondeo < 10:
   redondeo = 10
```

Figura 3-17 Código para el redondeo (elaboración propia)

En el siguiente bloque del código, se calcula la función de masa de probabilidad (PMF) y la función de distribución acumulada (CDF) (Figura 3-18). Para generar los valores del PMF y CDF, se han desarrollado dos bucles *for* que realicen las operaciones correspondientes “redondeo” veces. La ecuación de la PMF viene determinada como $PMF = ((1-p)^{i-1})p$ y la del CDF viene como $CDF = 1-(1-p)^{i}$ donde “p” es la probabilidad de transmisión del mensaje correcta, “q” la probabilidad de transmisión errónea e “i” cada uno de los intentos. Cada uno de los valores calculados con estas ecuaciones se almacenan en las variables “pmf” y “cdf”, que van siendo añadidos a sus listas correspondientes mediante el uso de la función *append*.

```
#Cálculo PMF y CDF
for i in range(redondeo):
    pmf = (((1-pexito)**(i-1))*pexito)
    listapmf.append(pmf)
```

```
    cdf = (1-((1-pexito)**i))

    listacdf.append(cdf)
```

Figura 3-18 Código para el cálculo del PMF y CDF

La última parte del código (Figura 3-19) es para la representación de los resultados con gráficas. Para poder representar las gráficas se ha preferido trabajar con matrices, por lo que se han creado los vectores "arraycdf" y "arraypmf" mediante la función *array* a partir de sus listas correspondientes. Con la función *arange(0, n, 1)* definimos el eje x como el número de bits, desde el primer hasta el bit "n" que introdujo el usuario. Para colocar una gráfica sobre la otra y que sea más visual se ha utilizado *subplot* y para el título y ejes, *title* y *label* respectivamente. Se han indicado los valores de ejes a representar junto con que el tipo de elemento para marcar los valores sea un punto seguido de una línea que una los valores con la función *plot.* Finalmente, la función *show* representará ambas gráficas.

```
#Representación de gráficas
arraycdf = np.array(listacdf)
arraypmf = np.array(listapmf)
x1 = np.arange(1, redondeo, 1)
x2 = np.arange(1, redondeo, 1)

plt.subplot(2, 1, 1)
plt.plot(x1, arraypmf, '.-')
plt.title('Intentos de transmisión en base a una probabilidad de error de mensaje')
plt.ylabel('PMF')

plt.subplot(2, 1, 2)
plt.plot(x2, arraycdf, '.-')
plt.xlabel('Intentos')
plt.ylabel('CDF')

plt.show()
```

Figura 3-19 Código para representación de las gráficas (elaboración propia)

Al igual que en el caso anterior, se finaliza con las referencias relacionadas con el cuaderno (Figura 3-20).

Referencias

1. https://en.wikipedia.org/wiki/Geometric_distribution
2. https://stackoverflow.com/questions/60081128/x-and-y-must-have-same-first-dimension-but-have-shapes-50-and-10?noredirect=1#comment106261150_60081128
3. https://matplotlib.org/gallery/subplots_axes_and_figures/subplot.html#sphx-glr-gallery-subplots-axes-and-figures-subplot-py
4. https://cursos.faitic.uvigo.es/moodle3_1920/pluginfile.php/60310/mod_resource/content/1/Soluciones.pdf
5. http://www.geocities.ws/rosa_virgen_sm/Comunicaciones/Com_Dig/Apuntes/Prob_de_error_2.pdf

Figura 3-20 Referencias relacionadas con el número de intentos de transmisión (elaboración propia)

3.3.3 Notebook del cálculo de RTO de temporizadores adaptativos TCP

El tercer cuaderno desarrollado permite calcular y representar gráficamente el intervalo de vencimiento de temporizador con el algoritmo de retransmisión adaptativo que utiliza TCP. Se comienza como puede verse en Figura 3-21 con una breve introducción al concepto.

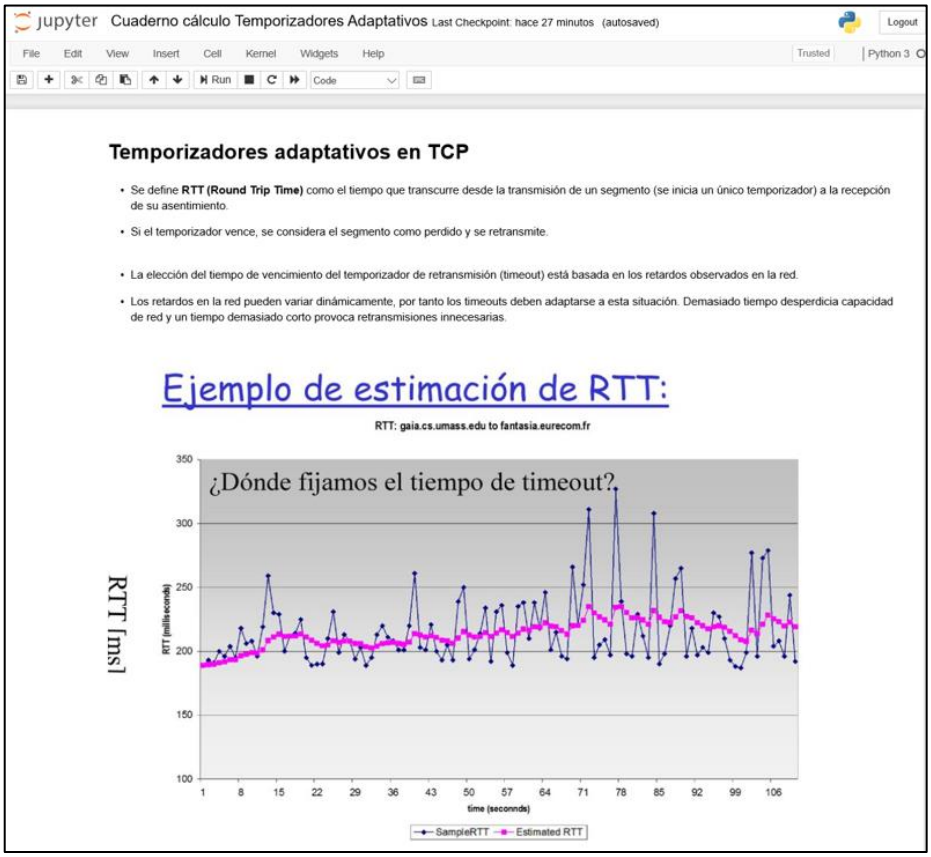


Figura 3-21 Explicación del concepto de RTO (elaboración propia)

La celda código comienza importando las librerías (Figura 3-22) con las que se va a trabajar. En este caso, se ha requerido el uso de *random* y *numpy* para generar valores según una distribución, de *math* para obtener el valor absoluto de una operación y de *matplotlib.pyplot* para la representación de gráficas.

```
#Librerías necesarias

import random

import math

import numpy as np

import matplotlib.pyplot as plt
```

Figura 3-22 Librerías utilizadas en el notebook del cálculo del RTO (elaboración propia)

El RTO (*Retransmission Timeout*) es tiempo de vencimiento del temporizador de retransmisión, es decir, el tiempo que espera el emisor a que se reciba el asentimiento. El RTO ha de ser mayor que el RTT *Round-Trip Time* (tiempo de ida y vuelta). Si el temporizador vence, se considera el segmento como perdido y se desecha. El cálculo del RTO sigue una media exponencialmente ponderada que

viene definida en el estándar RFC 6298. Para ello, se calcula el SRTT (*Smoothed Round-Trip Time*), que nos indica el valor esperado del RTT y la SVAR (el valor estimado de la varianza).

El estándar RFC 6298 establece [52]:

- "*(2.2) When the first RTT measurement R is made, the host MUST set*
    - *SRTT <- R*
    - *RTTVAR <- R/2*
    - *RTO <- SRTT + max (G, K*RTTVAR)*
    - *where K = 4*"
- "*(2.3) When a subsequent RTT measurement R' is made, a host MUST set*
    - *RTTVAR <- (1 - beta) * RTTVAR + beta * |SRTT - R'|*
    - *SRTT <- (1 - alpha) * SRTT + alpha * R'*
    - *RTO <- SRTT + max (G, K*RTTVAR)*
    - *where K = 4*"

Los valores de α y β recomendados por el estándar 6298 son 0.125 y 0.25 respectivamente. Es por esto por lo que en el siguiente módulo del código (Figura 3-23) se le recomienda al usuario el uso de esos dos valores, pero se le permite que introduzca cualquier valor de ponderación (siempre y cuando sea un valor válido). Para ello, se le pide que introduzca los valores de α y β gracias a las funciones *float* e *imput*. Como en el resto de casos anteriores, se acompaña de un bucle *while* para el caso en el que el usuario introduzca valores erróneos. En el texto que incluye la función *input* es dónde se añade la recomendación de los valores.

```
#Ponderaciones

α = float(input("Introduzca valor de α ""(se recomienda 0.125)"": "))

while α < 0:

    α = float(input("Introduzca valor de α ""(se recomienda 0.125)"": "))

β = float(input("Introduzca valor de β ""(se recomienda 0.25)"": "))

while β < 0:

    β = float(input("Introduzca valor de β ""(se recomienda 0.25)"": "))
```

Figura 3-23 Código para introducir valores de las ponderaciones (elaboración propia)

En este cuaderno el usuario va a poder escoger entre introducir uno a uno los valores de RTT o generarlos mediante una distribución. Para almacenar estos valores y los calculados, se han inicializado las cuatro listas y las dos variables que se muestran en Figura 3-24. La lista "SRTT" almacena los valores de los SRTT, la lista "RTT" los valores de RTT que introduce el usuario o que son generados según la distribución, la lista "Varianzas" los valores de SVAR y la lista "Temporizadores" los de RTO. Se inicializa la variable "n" con valor 0, que realizará la función de contador y "tempRTT" sin valor, que almacenará valores temporalmente.

```
#Listas y variables necesarias

print()

SRTT = []

RTT = []

Varianzas = []

Temporizadores = []

n = 0

tempRTT = ()
```

Figura 3-24 Código para definir listas y variables (elaboración propia)

Para que escoja entre introducir los valores o generarlos, se ha creado la variable "respuesta" y con los comandos *input* y *str* se piden datos de entrada (Figura 3-25). La respuesta para cada uno de los casos se indica en el texto de la función *input.* Si desea introducirlos manualmente deberá introducir la palabra "introducir" y si quiere generarlos la palabra "generar". El módulo se acompaña de un bucle *while* para el caso en el que se introduzca mal el comando. Si se introduce mal, se imprimirá en pantalla un mensaje con texto que indicará que la palabra introducida no es correcta y se volverá a pedir la respuesta.

```
#Usuario decide como introducir valores
respuesta = str(input("¿Desea introducir manualmente los valores de RTT o que sean generados? (introducir/generar): "))
while (respuesta != 'introducir') and (respuesta != 'generar'):
    print()
    print("Respuesta incorrecta")
    print()
    respuesta = str(input("¿Desea introducir manualmente los valores de RTT o que sean generados? (introducir/generar): "))
```

Figura 3-25 Código para que el usuario decida si genera o introduce los valores (elaboración propia)

La parte del código que se muestra en la Figura 3-26 es para el caso en el que el usuario haya decidido introducir los valores del RTT manualmente. Con la condición *if* si el valor de respuesta que introdujo es igual a "introducir", se ejecutará toda la parte de código adjunta a esta condición. Para que el usuario vaya introduciendo valores, se ha creado un bucle *while* que se mantendrá ejecutándose hasta que el usuario introduzca un 0 para finalizar (un RTT es imposible que tenga este valor). En el bucle *while* se le pide al usuario con los comandos *float* e *input* que vaya introduciendo los valores, que se irán almacenando en la variable "tempRTT". Dentro de este bucle se ha creado otro bucle *while* para los casos en los que el usuario se equivoque e introduzca valores negativos. En este caso, no se almacenará el valor y se volverá a pedir el valor del RTT. Al final de este módulo se almacenan los valores de la variable "tempRTT" en la lista "RTT".

```
#Se introducen valores de RTT manualmente
if respuesta == 'introducir':
    while tempRTT != 0:
        print()
        tempRTT = float(input("Introduzca valor de RTT: "))
        while tempRTT < 0:
            print()
            print("Valor de RTT no válido")
            print()
            tempRTT = float(input("Introduzca valor de RTT: "))
        RTT.append(tempRTT)
```

Figura 3-26 Código para introducir valores de RTT (elaboración propia)

Para realizar los cálculos, se ha desarrollado el código de la Figura 3-27 en el que se hace uso de las condiciones *if* y *elif*. Como la variable que actúa como contador "n" tiene valor 0, se ejecutará en primer lugar la parte de código adjunta al comando *if*. En esta parte se realizan los cálculos para la primera medición, como viene especificado en el estándar. Por lo que se toma el primer valor que introduce como SRTT y se almacena directamente en la lista "SRTT" con la función *SRTT.append(tempRTT)*. La división por dos de este valor se toma como SVAR (también definido por el estándar) y se almacena como primer valor en la lista "Varianzas" mediante la función *Varianzas.append(SVAR)*. Con estos dos primeros valores, ya se puede realizar el cálculo del primer RTO. El valor del RTO se almacena en la variable "TemporizadorTCP" que se añadirá a la lista "Temporizadores" gracias a *Temporizadores.append(TemporizadorTCP)*. El resto de esta parte de código imprime en pantalla los valores obtenidos, un mensaje que le indica al usuario que para finalizar de introducir valores ha de introducir un 0 y se incrementa el valor del contador en una unidad para salir de esta parte del código y tener un valor numérico de la cantidad de valores almacenados en cada lista.

```
#Cálculos para primer valor de RTT introducido según estándar
    if n == 0:
        SRTT.append(tempRTT)
        SVAR = SRTT[0]/2
        Varianzas.append(SVAR)
        TemporizadorTCP = SRTT[0] + 4*Varianzas[0]
        Temporizadores.append(TemporizadorTCP)
        print()
        print("Se estima que el valor del SRTT es", tempRTT)
        print("Se estima que el valor del RTTVAR es", SVAR)
        print("El intervalo de tiempo de vencimiento del temporizadorTCP es",
```

```
        TemporizadorTCP," segundos")
        print()
        print("Si deseea finalizar introduzca un 0")
        n = n + 1
```

Figura 3-27 Código para el cálculo del RTO con la primera medición de RTT (elaboración propia)

El bloque de la Figura 3-28 se ha desarrollado para las siguientes mediciones del RTT. Como el valor de la variable "n" se incrementó en una unidad, se ejecuta la parte de código adjunta al condicionante *elif*. Con las fórmulas establecidas en el estándar que siguen una media móvil exponencialmente ponderada (para las que hacen falta los valores que ya se tienen de las ponderaciones α y β y de los primeros SRTT y SVAR), se van calculando, almacenando en sus listas correspondientes e imprimiendo en pantalla los valores de SRTT, SVAR y RTO con las herramientas mencionadas anteriormente.

```
    #Cálculo para los siguientes RTT
    elif n > 0:
        SVAR = ((1-β)*Varianzas[n-1])+(β*(math.fabs(tempRTT-tempSRTT)))
        Varianzas.append(SVAR)
        tempSRTT = ((1-α)*SRTT[n-1])+(α*tempRTT)
        SRTT.append(tempSRTT)
        TemporizadorTCP = SRTT[n] + 4*Varianzas[n]
        Temporizadores.append(TemporizadorTCP)
        print()
        print("Se estima que el valor del SRTT es", tempSRTT)
        print("Se estima que el valor del RTTSVAR es", SVAR)
        print("El intervalo de tiempo de vencimiento del temporizadorTCP es",
        TemporizadorTCP," segundos")
        print()
        print("Si desea finalizar introduzca un 0")
        n = n + 1
```

Figura 3-28 Código para el cálculo del RTO con las siguientes mediciones de RTT (elaboración propia)

Una vez cerrada la condición *if* para el caso en el que el usuario quería introducir los datos manualmente, se pasaría al bloque de código para la representación de los resultados como veremos más adelante en la Figura 3-31.

Para el caso en el que el usuario desee generar los valores de RTT automáticamente según una distribución, se ha desarrollado otro bloque similar al anterior. Para que se ejecute la parte de código de la Figura 3-29, la palabra que se introducía y se almacenaba en la variable "respuesta" ha de ser "generar". Para generar los valores se ha implementado una parte de código que genera una matriz de valores aleatorios. Con las funciones *float* e *input* se le pide al usuario el valor máximo y mínimo de los RTT generados y con *int* e *input* el número de valores de RTT a generar. Los valores introducidos

son almacenados en las variables “maxi” “mini” y “Num” respectivamente. Los valores de estas variables son utilizados en la función *numpy.random.uniform(mini,maxi,Num)* para generar la matriz “RTT” a partir de una distribución uniforme con esos valores. Esta parte de código se acompaña de un bucle *while* para el caso en el que el usuario introduzca datos erróneos.

```
#Se generan valores de RTT según una distribución uniforme
elif respuesta == 'generar':
    print()
    maxi = float(input("Introduzca valor máximo de RTT: "))
    mini = float(input("Introduzca valor mínimo de RTT: "))
    Num = int(input("Introduzca número de valores a generar: "))
    while maxi <= 0 or mini <= 0 or Num <= 0:
        print()
        print("Uno o varios valores introducidos son erróneos")
        print()
        maxi = float(input("Introduzca valor máximo de RTT: "))
        mini = float(input("Introduzca valor mínimo de RTT: "))
        Num = int(input("Introduzca número de valores a generar: "))
    RTT = np.random.uniform(mini,maxi,Num)
```

Figura 3-29 Código para generar los valores según una distribución uniforme

Al igual que en la parte de código de la Figura 3-27 y de la Figura 3-28 , se realizan los cálculos según el estándar. Para realizar los cálculos (Figura 3-30) se recorre la matriz generada “RTT” elemento a elemento mediante un bucle *for* y se realizan las operaciones correspondientes. Para el caso de la primera medición, se selecciona el primer elemento de la matriz generada como primer valor de SRTT y con él realizar las mismas operaciones que ya se han mencionado. Para el caso de los siguientes valores, la dinámica también es la misma.

```
#Cálculos
    primera_vez = 1
    for i in RTT:
        #Cálculos para primer valor de RTT generado según estándar
        if primera_vez == 1:
            tempRTT = RTT[0]
            SRTT.append(tempRTT)
            RTTVAR = (RTT[0])/2
            Varianzas.append(RTTVAR)
```

```
            TemporizadorTCP = SRTT[0] + 4*Varianzas[0]
            Temporizadores.append(TemporizadorTCP)
            primera_vez = 0
        #Cálculo para los siguientes RTTs generados
        else:
            tempSRTT = ((1-α)*SRTT[n-1])+(α*i)
            SRTT.append(tempSRTT)
            SVAR = ((1-β)*Varianzas[n-1])+(β*(math.fabs(i-tempSRTT)))
            Varianzas.append(SVAR)
            TemporizadorTCP = SRTT[n] + 4*Varianzas[n]
            Temporizadores.append(TemporizadorTCP)
            print()
            print("Se estima que el valor del siguiente RTT es", tempSRTT)
            print("Se estima que el valor del siguiente SVAR es ",SVAR)
            print("El intervalo de tiempo de vencimiento del temporizadorTCP es",
            TemporizadorTCP," ms")
```

Figura 3-30 Código para el cálculo del RTO a partir de la matriz de RTTs generada (elaboración propia)

La última parte del código (Figura 3-31) es para representar gráficamente los valores del RTT, SRTT y RTO y que el usuario analice los resultados obtenidos. Con la función *plot* de *matplotlib* se han generado un gráfico de líneas que representará los resultados de cada una de las listas. Se han añadido título, etiquetas a los ejes y una leyenda con las funciones *title*, *label* y *legend* respectivamente. Se cierra el código representando en pantalla el diagrama con la función *show*.

```
#Representación de RTT, SRTT y RTO
line_chart1 = plt.plot(range(0,n), SRTT)
line_chart2 = plt.plot(range(0,n), RTT)
line_chart3 = plt.plot(range(0,n), Temporizadores)
plt.title('Temporizadores adaptativos TCP')
plt.xlabel('Muestras')
plt.ylabel('RTT (ms)')
plt.legend(['SRTT', 'RTT', 'RTO'], loc=4)
plt.show()
```

Figura 3-31 Código para la representación del diagrama

Se cierra el cuaderno con las referencias relacionadas con el cálculo del RTO (Figura 3-32).

Referencias

1. https://www.youtube.com/watch?v=Xk_16aK665g
2. https://www.fdi.ucm.es/profesor/rubensm/ASOR/Trasparencias/Tema%203-%20Conceptos%20Avanzados%20del%20Protocolo%20TCP.pdf
3. https://www.tdx.cat/bitstream/handle/10803/7040/04AMCA04de15.pdf;sequence=4
4. https://w3.ual.es/~vruiz/Docencia/Apuntes/Networking/Protocols/Level-4/05-TCP/index.html
5. https://matplotlib.org/gallery/lines_bars_and_markers/cohere.html#sphx-glr-gallery-lines-bars-and-markers-cohere-py

Figura 3-32 Referencias relacionadas con el cálculo del RTO (elaboración propia)

3.3.4 Notebook del código de verificación por redundancia cíclica

El cuaderno de código de verificación por redundancia cíclica CRC (*Cyclic Redundancy Check)* permite simular la transmisión de datos y comprobar por medio de este código de detección de errores, si existen errores en dicho mensaje cuando este llega al receptor. Para ello, se introduce el concepto con una explicación paso a paso del procedimiento que realiza este código para comprobar si existe error, como puede verse en la Figura 3-33.

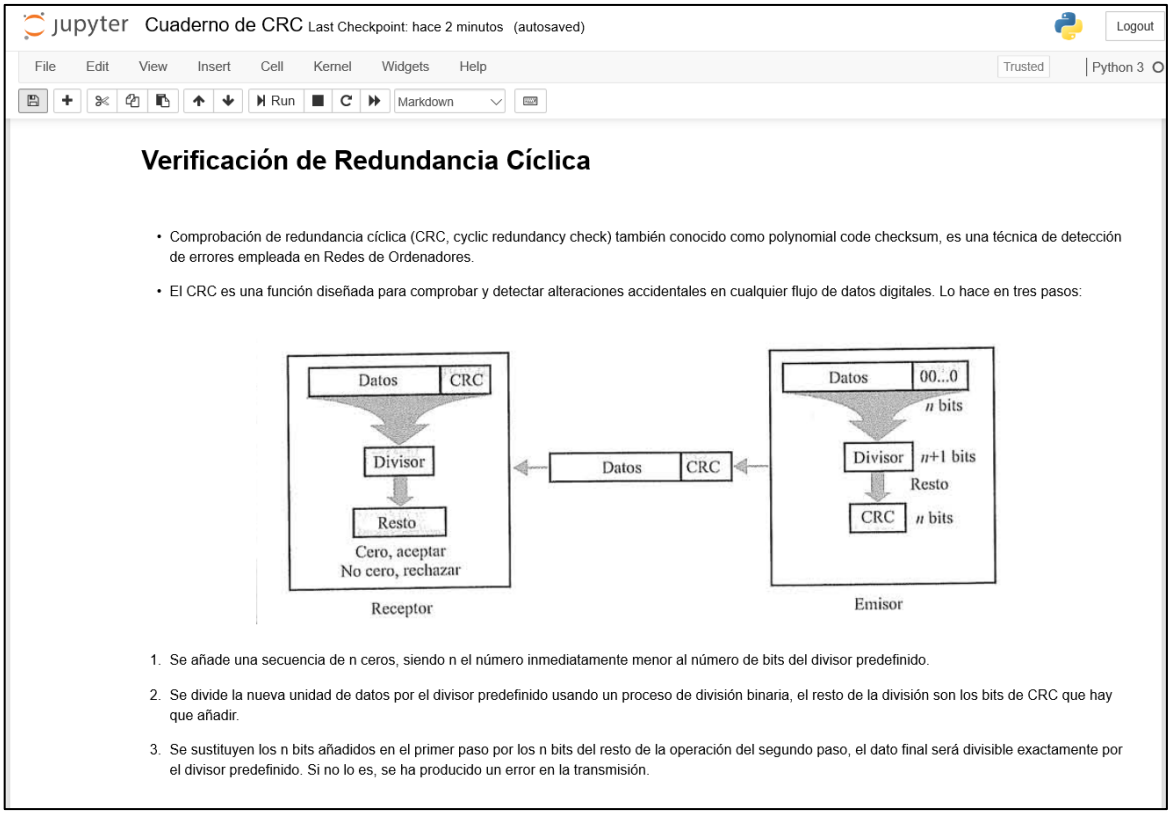


Figura 3-33 Explicación del concepto de CRC (elaboración propia)

Las librerías utilizadas en este cuaderno (Figura 3-34) son *random* y *array*. El empleo de *random* es para trabajar con números aleatorios para simular los posibles errores en la transmisión y el de *array* para almacenar y trabajar con los datos del mensaje.

```
import random
from array import array
```

Figura 3-34 Librerías utilizadas en el código de CRC (elaboración propia)

Se comienza el código definiendo la función que calcula el CRC (Figura 3-35) que se empleará en otros bloques de código que se explicarán posteriormente. Definimos con el comando *def* la función "crc8" que se aplicará a los datos de entrada para obtener el valor del resto resultante de dividir los mensajes por el polinomio generador "0x1D" (00011101).

```
# Usando el polinomio 0x1D de CRC-8
def crc8(datos):
    resto = 0
    for byte in datos:
        resto = resto ^ byte
        for j in range(0,8):
            if (resto & 0x80):
                resto = (resto << 1) ^ 0x1D
            else:
                resto = (resto << 1)
        resto = resto & 0xFF
    # Descomentar para imprimir los restos parciales
    # print hex(resto)
    return resto
```

Figura 3-35 Código para definir la función de cálculo de CRC (elaboración propia)

Con *str*, *float* e *input* pedimos al usuario que introduzca el mensaje que se va a transmitir y la probabilidad de error de mensaje, que se almacenaran en las variables "texto" y "prob_error" (Figura 3-36). Creamos un vector de caracteres (tipo B) con la palabra codificada que se introdujo anteriormente.

```
texto = str(input("Introduzca mensaje a transmitir: "))

prob_error = float(input("Introduzca probabilidad de error: "))

datos = array('B', texto.encode())
```

Figura 3-36 Código para introducir probabilidad y datos a transmitir (elaboración propia)

Para convertir posteriormente los caracteres a binario, definimos una función simple "to_bin" con *lambda* (Figura 3-37). Se pasarán a binario los caracteres eliminando los dos caracteres iniciales (0b) que Python establece predeterminadamente al inicio.

```
# Función de conversión a binario
to_bin = lambda x: str(bin(x)[2:].zfill(8))
```

Figura 3-37 Código para definir función de conversión a binario (elaboración propia)

Se imprimen en pantalla los datos del mensaje en binario, el resultado del resto de la codificación con CRC (realizado con la función "CRC8") y los datos con el CRC añadido al final (Figura 3-38).

```
print("\n======== TRANSMISOR: DATOS ==========")
print("|".join(map(to_bin, datos)))

# Calculamos el CRC (se hace en el transmisor)
print("\n======== TRANSMISOR: CALCULO CRC8 ==========")
resto = crc8(datos)
print("Resto: " + str(to_bin(resto)))

# Añadimos el resto (CRC) al mensaje
print("\n======== TRANSMISOR: DATOS CON CRC ==========")
datos.append(resto)
print("|".join(map(to_bin, datos)))
```

Figura 3-38 Código para imprimir en pantalla el mensaje y su codificación con CRC (elaboración propia)

El bloque de código de la Figura 3-39 simula la transmisión de los datos por el canal. Cuando el número generado con la función *random.random* sea menor que la probabilidad de error introducida, se generará un error en los datos. A mayor probabilidad de error, más probabilidad existe de que el número generado sea menor que ésta y, por lo tanto, mayor probabilidad de que se genere el error.

Para generar el error, se selecciona aleatoriamente con el comando *random.choice* un byte del mensaje y dentro de ese byte, un bit. Se almacenan estas posiciones en las variables "posicion_byte" y "posicion_bit" y se modifica el valor del bit. Se cierra esta parte de código imprimiendo en pantalla el número del byte y de bit en los que se ha producido el error y la secuencia de bits con una "X" en el bit erróneo. Si el número generado era mayor que la probabilidad de error, la parte de código mencionada anteriormente no se ejecuta y se imprime en pantalla un mensaje que indica que no hubo error en la transmisión.

```
# Añadimos aleatoriamente un error en un bit con una probabilidad dada
print("\n======== CANAL ==========")
if random.random() < prob_error:
    # Hay que introducir error, elegimos la posición al azar
    posicion_byte = random.choice(range(len(texto)))
    posicion_bit = random.choice(range(8))
    datos[posicion_byte] = datos[posicion_byte] ^ (0x80 >> posicion_bit)
    # Sumamos uno por claridad: la primera posición es en realidad la 0
    # Tanto para el byte como para el bit se empieza a contar por la izquierda
    print("Error en el byte " + str(posicion_byte+1) + " y bit " +  str(posicion_bit+1))
    bits = list("".join(map(to_bin, datos)))
    bits[posicion_byte*8+posicion_bit] = 'X'
```

```
        print("".join(bits))
    else:
        print("No se introduce un error")
```

Figura 3-39 Código para simular el error generado al transmitirse el mensaje (elaboración propia)

Se cierra la celda código (Figura 3-40) imprimiendo en pantalla los bytes que recibe el receptor, imprimiendo los vectores "datos" y "resto" que han podido ser sido modificados como consecuencia del código adjunto en la Figura 3-39 ("datos") y Figura 3-40 ("resto"). Se imprime además un mensaje que indica si ha habido error en la transmisión (el CRC calculado en recepción es distinto de 0) o no (el CRC en recepción es 0).

```
print("\n======== RECEPTOR: DATOS RECIBIDOS ==========")

print("|".join(map(to_bin, datos)))


# Verificamos el CRC (esto lo hace el receptor)

print("\n======== RECEPTOR: VERIFICAR CRC8 ==========")

resto = crc8(datos)

if resto == 0:

    print("Se han recibido los datos sin error")

else:

    print("Se ha producido un error en la transmisión de datos")

    print("Resto: " + str(to_bin(resto)))
```

Figura 3-40 Código para imprimir en pantalla los datos recibidos en el receptor.

Las referencias utilizadas para el desarrollo son las que aparecen en la Figura 3-41.

Figura 3-41 Referencias relacionadas con el código de CRC

3.3.5 Notebook del algoritmo de Token Bucket

El último cuaderno de la herramienta es para simular el algoritmo de conformación de tráfico *Token Bucket* [7]. Como en el resto de cuadernos, se introduce el concepto en las primeras celdas y se desarrolla el código que simula el algoritmo. En la Figura 3-42 se muestra cómo se introduce teóricamente el concepto.

Figura 3-42 Explicación del concepto de Token Bucket (elaboración propia)

Las librerías utilizadas para el desarrollo del cuaderno son las que se muestran en la Figura 3-43.

```
import numpy as np
import matplotlib.pyplot as plt
```

Figura 3-43 Librerías utilizadas para el cuaderno del algoritmo de Token Bucket (elaboración propia)

El código de la Figura 3-44 ha sido desarrollado para que el usuario vaya introduciendo y almacenando, alternativamente, los valores de los periodos de tiempo en los que se está recibiendo una ráfaga de datos y los valores de los periodos en los que no. Para ello, se han creado las listas y variables que pueden verse en la Figura 3-44. La lista "Tiempos" almacenará los valores de tiempo que introduzca el usuario por medio de la variable "T" y las funciones *float* e *input*. El resto de variables inicializadas son para que se alternen las peticiones de los datos.

```
#Código para que el usuario introduzca la duración de las ráfagas que entran


Tiempos = []
T = float(input("Introduzca tiempo 1ª ráfaga de datos: "))
Tiempos.append(T)
T = float(input("Introduzca 1º tiempo sin transmitir: "))
Tiempos.append(T)
```

```
if T != 0:
    x = 2
    q = 2
    w = 1
    while T != 0:
        if w%2 != 0:
            T = float(input("Introduzca tiempo en segundos" + str(x) +"ª ráfaga de datos: "))
            x = x + 1
            w = w + 1
            Tiempos.append(T)
        else:
            T = float(input("Introduzca " + str(q) +"º tiempo en segundos sin transmitir: "))
            q = q + 1
            w = w + 1
            Tiempos.append(T)
```

Figura 3-44 Código para introducir los valores de los periodos de tiempo (elaboración propia)

De manera similar, se ha desarrollado el bloque de la Figura 3-45 para que se introduzcan las tasas de transmisión de cada una de las ráfagas. La lista "Tasas" almacenará los valores de las tasas que introduzca el usuario mediante la variable "Tr". Se generan valores de tasa 0 para los periodos de tiempo en los que no se transmite alternando, de la misma manera que con los tiempos, los valores que introduce el usuario con ceros. El resto de variables inicializadas actúan como contadores.

```
#Código para que el usuario introduzca la velocidad de transmisión


Tasas = []
print()
Tr = float(input("Introduzca tasa de transferencia 1ª ráfaga de datos en Mbps: "))
Tasas.append(Tr)


if Tr != 0:
    k = 2
    p = 0
    while Tr != 0:
        if w%2 != 0:
            Tr = float(input("Introduzca tasa de transferencia " + str(k) +"ª ráfaga de datos en Mbps: "))
            k = k + 1
            Tasas.append(Tr)
            w = w + 1
```

```
        else:
            Trr = 0
            Tasas.append(Trr)
            w = w + 1
            p = p + 1
```

Figura 3-45 Código para que se introduzca la tasa de transmisión (elaboración propia)

Se repite el proceso con un bucle *while* si el número de tasas no se corresponde con el número de tiempos de transmisión Figura 3-46.

```
#Repetimos todo si hay error al introducir el número de datos


while k != x:


    Tiempos = []
    T = float(input("Introduzca tiempo en segundos 1ª ráfaga de datos: "))
    Tiempos.append(T)
    T = float(input("Introduzca 1º tiempo en segundos sin transmitir: "))
    Tiempos.append(T)

    if T != 0:
        x = 2
        q = 2
        w = 1
        while T != 0:
            if w%2 != 0:
                T = float(input("Introduzca tiempo en segundos " + str(x) +"ª ráfaga de datos: "))
                x = x + 1
                w = w + 1
                Tiempos.append(T)
            else:
                T = float(input("Introduzca " + str(q) +"º tiempo en segundos sin transmitir: "))
                q = q + 1
                w = w + 1
                Tiempos.append(T)


    Tasas = []
```

```
print()
Tr = float(input("Introduzca tasa de transferencia 1ª ráfaga de datos en Mbps: "))
Tasas.append(Tr)


if Tr != 0:
    k = 2
    p = 0
    while Tr != 0:
        if w%2 != 0:
            Tr = float(input("Introduzca tasa de transferencia " + str(k) +"ª ráfaga de datos en
            Mbps: "))
            k = k + 1
            Tasas.append(Tr)
            w = w + 1
        else:
            Trr = 0
            Tasas.append(Trr)
            w = w + 1
            p = p + 1


if k != x:
    print()
    print("ERROR, NÚMERO DE TIEMPOS O DE TASAS INCORRECTO")
```

Figura 3-46 Código para repetir el proceso (elaboración propia)

Para el cálculo de la tasa media de transmisión (Figura 3-47), se ha creado la lista "Datos". Mediante un bucle *for* y la función *zip* multiplicamos los valores de cada tiempo con su tasa correspondiente. Los periodos de tiempo durante los cuales no se transmite se multiplicarán por un valor nulo de tasa de transmisión, por lo que no influirán en el resultado de la tasa media. Cada uno de los valores calculados se almacena en la lista "Datos". Para el cálculo de la tasa media se suman con la función *sum* todos los valores de esta lista "Datos" y se dividen por la suma de todos los tiempos (también sumados mediante la función *sum*). Finalmente se muestra el resultado en pantalla.

```
#Cálculo de tasa media de transmisión
Datos = []
z = 0
v = 0
for i,j in zip(Tasas,Tiempos):
    Dato = i*j
```

```
        Datos.append(Dato)


suma = sum(Datos)
tiempototal = sum(Tiempos)
Tasamedia = suma/tiempototal
print()
print("La tasa media de transmision es:",Tasamedia)
```

Figura 3-47 Código para el cálculo de la tasa media de transmisión (elaboración propia)

Los valores de los periodos de tiempo que se le han ido pidiendo al usuario no son valores acumulados, por lo que hay que ir sumando los valores de la lista y almacenarlos en otra para poder representar estos tiempos en el eje de abscisas. Para realizar esta operación se ha desarrollado la parte que puede verse en la Figura 3-48, creando para ello la lista "sumatiempos". Para que el gráfico en escalera de *matplotlib* comience a representar los valores desde el origen, se ha añadido con el comando *append* un cero como primer valor de la nueva lista. Mediante un bucle *for*, se toma un valor de la lista "Tasas" y se suma con el valor anterior almacenado en la lista "sumatiempos".

```
#Suma de tiempos para representación
sumatiempos = []
sumatiempos.append(0)
time = ()
c = 1
for i in Tiempos:
    if c == 1:
        time = i
        sumatiempos.append(time)
        c = c + 1
    else:
        time = i + sumatiempos[c-1]
        sumatiempos.append(time)
        c = c + 1
```

Figura 3-48 Código para crear lista con los valores acumulados del tiempo (elaboración propia)

Al igual que en el bloque anterior, también se crea una nueva lista "representación" (Figura 3-49) y se le añade un 0 como primer valor. Los valores para representar las tasas no son acumulativos, por lo que a la lista "representación" se le añaden directamente los valores de la lista "Tasas" después del cero. Se ha creado un bucle *for* para obtener el valor más alto de las tasas, que será el que delimite el margen derecho del gráfico.

Mediante la función *array* se crean los vectores "x" e "y" con los valores de las listas "Tiempos" y "Tasas" respectivamente. Con la función *suptitle* de *Matplotlib* añadimos el título, con *axis* los valores de los márgenes y con *step* graficamos un diagrama de escalera de los valores de los vectores "x" e "y".

```
#Representación
maximo = Tasas[0]
for i in Tasas:
    if i > maximo:
        maximo = i

representacion = []
representacion.append(0)
for i in Tasas:
    representacion.append(i)

x = np.array(sumatiempos)
y = np.array(representacion)
plt.suptitle("Tasas de entrada")
plt.axis([0,sumatiempos[c-1],0,maximo+1])
plt.step(x, y)
plt.axhline(y=Tasamedia, xmin=0, xmax=maximo, color = "green", linestyle = "dashed")
plt.show()
```

Figura 3-49 Código para representar las tasas (elaboración propia)

Tras la representación de las gráficas, se pide que se introduzcan los valores de capacidad y tasa de salida que tendrá el Token Bucket (Figura 3-50). Con el valor de la capacidad y cada uno de los valores de tiempo y de tasa de cada una de las ráfagas, se calculan los excesos de estas. El exceso de una ráfaga viene definido como $Exceso = (Tasa_i - Capacidad) \times Tiempo_i$. Los valores de exceso calculados se almacenan en la lista "excesos" y son representados en pantalla sólo si son positivos por medio del bucle *for* que se encuentra al final de la parte de código de la Figura 3-50.

```
#Cálculo excesos
print()
capacidad = float(input("Introduzca capacidad del bucket: "))
salida = float(input("Introduzca tasa de salida del bucket: "))

excesos = []
s = 0
for h in Tasas:
    if h != 0:
        exceso = (h-capacidad)*Tiempos[s]
        excesos.append(exceso)
        s = s + 2

e = 0
for y in excesos:
    if y >= 0:
        print("El exceso en la rágafa",e+1,"es",excesos[e])
```

```
    e = e+1
  else:
    print("En la rágafa",e+1,"no hay exceso")
    e = e+1
```

Figura 3-50 Código para el cálculo de excesos

Finalmente, se calcula el tiempo que se podría transmitir en base a la tasa de la ráfaga cuando hay exceso (Figura 3-51). Se ha creado la lista "Tasas1" que almacenará únicamente los valores de las tasas de las ráfagas por medio de un bucle *for*. El tiempo que se puede estar transmitiendo cuando hay exceso viene determinado por $Tiempo = \frac{R}{Tasa_i - B(t)}$ donde R es la tasa de salida y B es la capacidad de fichas (tokens) a cursar en un instante de tiempo. Se realiza esta operación para cada uno de los valores de la lista "Tasas1" y se imprimen en pantalla.

```
#Cálculo tiempos en base a la tasa
tx = ()
nume1 = 0
Tasas1 = []

for i in Tasas:
  if i > 0:
    Tasas1.append(i)

for i in Tasas1:
  if i > capacidad:
    tx = (salida/(i-capacidad))
     print("El tiempo que se puede estar transmitiendo a la tasa de la",nume1+1,"ª ráfaga es:",tx)
    nume1 = nume1 + 1
  else:
    nume1 = nume1 + 1
```

Figura 3-51 Código para el cálculo del tiempo máximo de transmisión en base a la tasa (elaboración propia)

Se finaliza el desarrollo del cuaderno con las referencias utilizadas (Figura 3-52).

Referencias

1. https://image.slidesharecdn.com/algoritmos-de-control-de-congestin-1226586258550402-8/95/algoritmos-de-control-de-congestin-7-728.jpg?cb=1226557458
2. https://cursos.faitic.uvigo.es/moodle3_1920/pluginfile.php/64537/mod_resource/content/0/7_QoS.pdf

Figura 3-52 Referencias relacionadas con el algoritmo Token Bucket

En el *Anexo I: Códigos desarrollados en los notebooks* se adjuntan los códigos completos de cada uno de los cuadernos.

4 VALIDACIÓN

En el siguiente apartado se muestran los resultados obtenidos y las pruebas realizadas a la herramienta desarrollada en este proyecto.

4.1 Resultados obtenidos

4.1.1 Resultados obtenidos en el cuaderno Canal Binario Simétrico

Introduciendo una probabilidad de error de 0.2 y una secuencia de 16 bits, se han obtenido los siguientes resultados:

```
Introduzca probabilidad de error: 0.2


Introduzca manualmente secuencia de bits: 1011000110111101


Tras la transmisión de 0b1011000110111101 por el canal, la salida queda:
1111000110111100


La secuencia de bits ha sido modificada al transmitirse por el canal


La secuencia a la entrada era: [1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1,
0, 1]
La secuencia a la salida es  : [1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1,
0, 0]


Con un error de 2 bits
```

Figura 4-1 Resultados obtenidos con una probabilidad baja (elaboración propia)

Con una probabilidad más alta de 0.8 y una secuencia de 16 bits:

```
Introduzca probabilidad de error: 0.8
```

```
Introduzca manualmente secuencia de bits: 1011000110111101


Tras la transmisión de 0b1011000110111101 por el canal, la salida queda:
0101111010101001
La secuencia de bits ha sido modificada al transmitirse por el canal


La secuencia a la entrada era: [1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1,
0, 1]
La secuencia a la salida es  : [0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0,
0, 1]


Con un error de 9 bits
```

Figura 4-2 Resultados obtenidos con una probabilidad alta (elaboración propia)

Como última prueba se introduce una probabilidad de error igual a cero:

```
Introduzca probabilidad de error: 0


Introduzca manualmente secuencia de bits: 1111111111111111


Tras la transmisión de 0b1111111111111111 por el canal, la salida queda:
1111111111111111


La secuencia de bits no ha sido modificada al transmitirse por el canal


La secuencia a la entrada era: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1]
La secuencia a la salida es  : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1]


Con un error de 0 bits
```

Figura 4-3 Resultados obtenidos con una probabilidad nula (elaboración propia)

Los resultados parecen indicar el buen funcionamiento del código, ya que se ha observado que:

- Al aumentarse la probabilidad de error de bit aumenta el número de bits erróneos, al disminuirse, disminuyen y cuando la probabilidad es cero, no los hay.
- La secuencia de salida tiene el mismo número de bits que la secuencia de entrada.
- El mensaje de texto que indica si se ha modificado o no la secuencia funciona correctamente.
- El contador del número de bits funciona correctamente al contar el número de bits erróneos en ambas listas.

4.1.2 Resultados del cuaderno del número de intentos de transmisión

En un primer momento, este cuaderno se desarrolló de otra manera. Mediante las funciones *float* e *input* se pedía al usuario la probabilidad de error de bit y el número de bits que se iban a transmitir. A partir de estos datos, se calculaba la probabilidad de transmitir el mensaje correctamente y el número medio de intentos para lograrlo. La probabilidad de transmisión del mensaje correctamente viene

definida por $Pm = (1-p)^n$, donde "p" es la probabilidad de error y "n" el número de bits. Como el número de intentos viene determinado por $Intentos = \frac{1}{Pm}$, cuando se introducía un número de bits elevado Python redondeaba P a 0 y daba el error "*float división by zero*" al calcular el número de intentos y aunque existan funciones como *Decimal* o *Fraction* para tratar estos asuntos, no se logró solucionar el problema. Es por esto por lo que se ha optado por pedirle directamente al usuario la probabilidad de transmisión correcta del mensaje, que no es redondeada y no surgen errores al ejecutarlo.

Introduciendo una probabilidad de transmisión del mensaje correcta de 0.3678776, que corresponde a una probabilidad de error de bit 10^{-5} y a un número de 100.000 bits transmitidos, se han obtenido los siguientes resultados:

```
Introduzca probabilidad de transmisión de mensaje correcta: 0.3678776


El número medio de intentos para la transmisión del mensaje es
2.7182954330462086
```

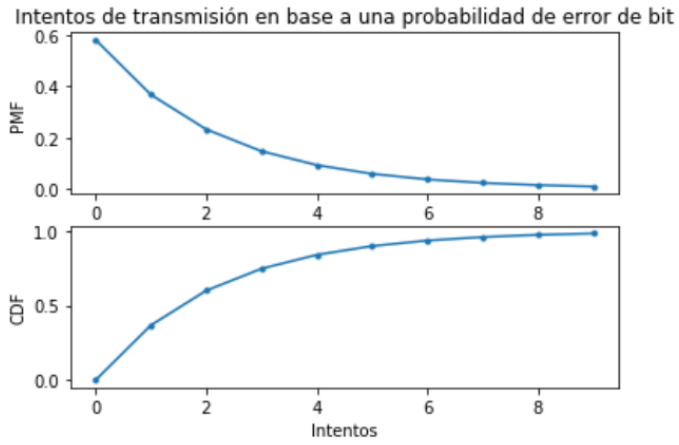


Figura 4-4 Resultados con una probabilidad de transmisión de 0.36 (elaboración propia)

Con una probabilidad de transmisión correcta muy alta:

```
Introduzca probabilidad de transmisión de mensaje correcta: 0.95


El número medio de intentos para la transmisión del mensaje es 1.05263157
89473684
```

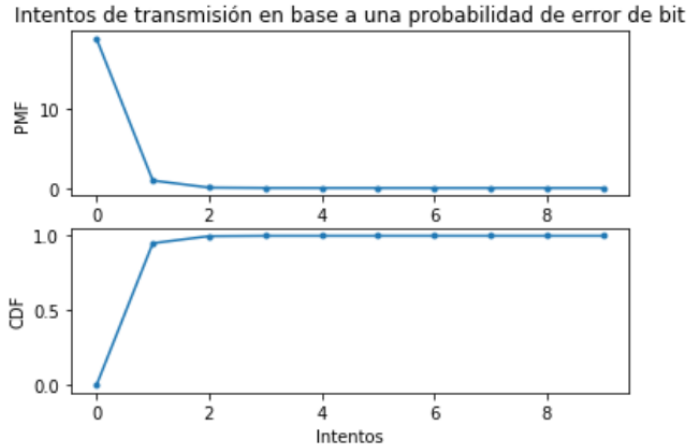


Figura 4-5 Resultados con una probabilidad de transmisión de mensaje muy alta (elaboración propia)

Como última prueba, se introduce una probabilidad de transmisión de mensaje muy baja:

```
Introduzca probabilidad de transmisión de mensaje correcta:
0.00000000000000000000000000000000001

El número medio de intentos para la transmisión del mensaje es 1e+35
```

Figura 4-6 Resultados con una probabilidad de transmisión de mensaje muy baja (elaboración propia)

En los casos en los que la probabilidad es muy pequeña, no se presenta en pantalla el gráfico, se muestra un error en memoria RAM y el programa se bloquea. No obstante, al ser resultados muy pequeños y, por lo tanto, incoherentes, no se han considerado relevantes.

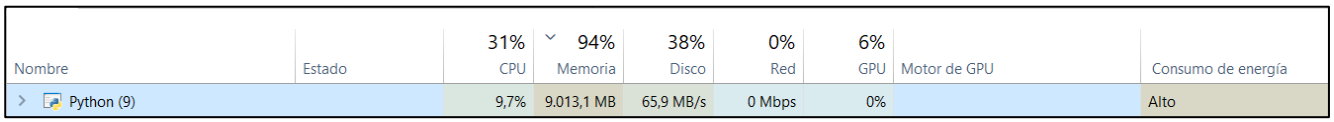


Figura 4-7 Error surgido al ejecutar el código con probabilidades muy bajas (elaboración propia)

Por otra parte, se ha comprobado que, si el usuario introduce valores numéricos no válidos, se vuelve a pedir el dato de entrada:

```
Introduzca probabilidad de transmisión de mensaje correcta: 1.2
Introduzca probabilidad de transmisión de mensaje correcta: -0.2
Introduzca probabilidad de transmisión de mensaje correcta:
```

Figura 4-8 Resultados al introducir datos erróneos (elaboración propia)

A la vista de los resultados, salvo por las excepciones ya mencionadas, se observa que el código funciona correctamente para casos con valores coherentes, que son los valores con los que se deberían trabajar normalmente.

4.1.3 Resultados del cuaderno del cálculo de temporizadores adaptativos en TCP

Un resultado resumido que se ha obtenido introduciendo valores con un patrón completamente aleatorio para el caso en el que el usuario escoja la opción de introducir los valores manualmente es el siguiente:

```
Introduzca valor de α (se recomienda 0.125): 0.125
Introduzca valor de β (se recomienda 0.25): 0.25

Introduzca valor de RTT (ms): 21.7

Se estima que el valor del SRTT es 21.7
Se estima que el valor del RTTVAR es 10.85
El intervalo de tiempo de vencimiento del temporizadorTCP es 65.1 ms

Si desea finalizar introduzca un 0

Introduzca valor de RTT (ms): 17.3

Se estima que el valor del SRTT es 21.150000000000002
```

```
Se estima que el valor del RTTSVAR es 9.237499999999999
El intervalo de tiempo de vencimiento del temporizadorTCP es
58.099999999999994 ms

Si desea finalizar introduzca un 0

Introduzca valor de RTT (ms): 29.3

Se estima que el valor del SRTT es 22.168750000000003
Se estima que el valor del RTTSVAR es 8.965625
El intervalo de tiempo de vencimiento del temporizadorTCP es 58.03125 ms

Si desea finalizar introduzca un 0

Introduzca valor de RTT (ms): 8.4

Se estima que el valor del SRTT es 20.447656250000005
Se estima que el valor del RTTSVAR es 10.16640625
El intervalo de tiempo de vencimiento del temporizadorTCP es 61.11328125
ms

Si desea finalizar introduzca un 0

Introduzca valor de RTT (ms): 27.3

Se estima que el valor del SRTT es 21.304199218750007
Se estima que el valor del RTTSVAR es 9.337890624999998
El intervalo de tiempo de vencimiento del temporizadorTCP es
58.65576171875 ms

Si desea finalizar introduzca un 0

...
```

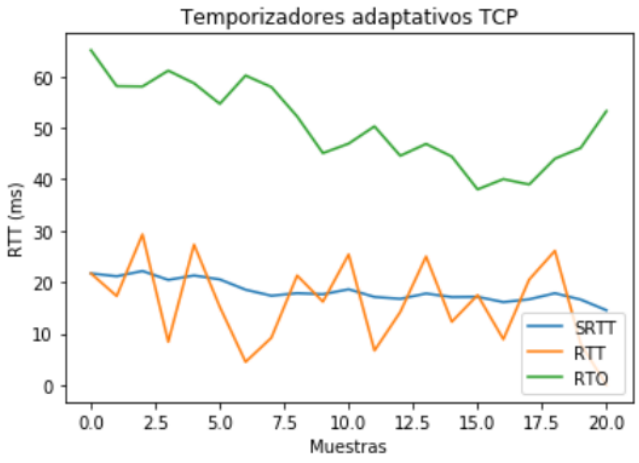


Figura 4-9 Resultado obtenido al introducir los datos manualmente

Para el caso en el que los valores de RTT son generados, un resultado obtenido es:

```
Introduzca valor de α (se recomienda 0.125): 0.125
Introduzca valor de β (se recomienda 0.25): 0.25

¿Desea introducir manualmente los valores de RTT o que sean generados? (introducir/generar): generar

Introduzca valor máximo de RTT: 25
Introduzca valor mínimo de RTT: 5
Introduzca número de valores a generar: 40

Se estima que el valor del siguiente RTT es 9.300209246782632
Se estima que el valor del siguiente SVAR es 3.6111506886565805
El intervalo de tiempo de vencimiento del temporizadorTCP es 23.744812001408953 ms



Se estima que el valor del siguiente RTT es 10.176612444863583
Se estima que el valor del siguiente SVAR es 4.4611694126543355
El intervalo de tiempo de vencimiento del temporizadorTCP es 28.021290095480925 ms



Se estima que el valor del siguiente RTT es 11.420028549871795
Se estima que el valor del siguiente SVAR es 5.832709269507174
El intervalo de tiempo de vencimiento del temporizadorTCP es 34.750865627900495 ms

...
```

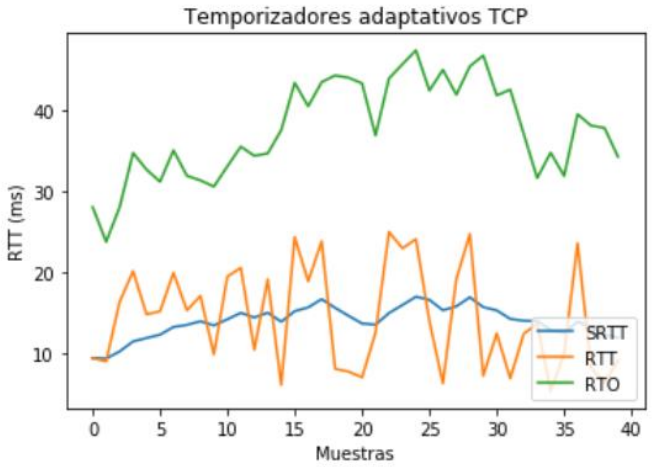


Figura 4-10 Resultados obtenidos generando los valores de RTT

A la vista de los resultados, puede concluirse el buen funcionamiento del código ya que se observa que:

- Como puede verse en la Figura 4-10, aunque se tomen valores de RTT que se salen de la normalidad (picos naranjas), la estimación del RTT no se ve demasiado afectada (picos azules).
- Tras la inicialización, el gráfico del RTO se encuentra siempre por encima de los RTTs que llegan.
- Si se introduce un número consecutivo de valores de RTT que se encuentran por debajo de los estimados, los valores de los estimados van disminuyendo y viceversa.
- Los valores de RTT que son generados se encuentran entre los valores que ha especificado el usuario y el número de estos también es el que se introdujo.

4.1.4 Resultados del cuaderno de CRC

Introduciendo como mensaje una palabra de corta longitud como puede ser "Hola" y una probabilidad de error pequeña de 0.2, se espera, como puede verse en el resultado obtenido en la Figura 4-11, que no se produzcan errores en la transmisión.

```
Introduzca mensaje a transmitir: Hola
Introduzca probabilidad de error: 0.2

======== TRANSMISOR: DATOS ==========
01001000|01101111|01101100|01100001

======== TRANSMISOR: CALCULO CRC8 ==========
Resto: 10111111

======== TRANSMISOR: DATOS CON CRC ==========
01001000|01101111|01101100|01100001|10111111

======== CANAL ==========
No se introduce un error

======== RECEPTOR: DATOS RECIBIDOS ==========
01001000|01101111|01101100|01100001|10111111

======== RECEPTOR: VERIFICAR CRC8 ==========
Se han recibido los datos sin error
```

Figura 4-11 Resultados obtenidos con un mensaje corto y una probabilidad baja (elaboración propia)

Si se introduce una probabilidad más alta y un mensaje de mayor longitud, es más probable que se produzca un error. Como puede verse en la Figura 4-12, introduciendo el mensaje "Hola Caracola" y una probabilidad de error de 0.9, se obtiene un resultado en el que se ha producido error en la transmisión.

```
Introduzca mensaje a transmitir: Hola Caracola
```

```
Introduzca probabilidad de error: 0.9

======== TRANSMISOR: DATOS ==========
01001000|01101111|01101100|01100001|00100000|01000011|01100001|01110010|0
1100001|01100011|01101111|01101100|01100001

======== TRANSMISOR: CALCULO CRC8 ==========
Resto: 10010011

======== TRANSMISOR: DATOS CON CRC ==========
01001000|01101111|01101100|01100001|00100000|01000011|01100001|01110010|0
1100001|01100011|01101111|01101100|01100001|10010011

======== CANAL ==========
Error en el byte 1 y bit 7
010010X0011011111011011000110000100100000010000110110000010111001001100001
1100011011011110110110001100001100010011

======== RECEPTOR: DATOS RECIBIDOS ==========
01001010|01101111|01101100|01100001|00100000|01000011|01100001|01110010|0
1100001|01100011|01101111|01101100|01100001|10010011

======== RECEPTOR: VERIFICAR CRC8 ==========
Se ha producido un error en la transmisión de datos
Resto: 00011111
```

Figura 4-12 Resultados obtenidos con probabilidad de error más alta (elaboración propia)

Analizando los resultados obtenidos, se observa que:

- La posibilidad de que aparezca un error durante la transmisión de mensaje aumenta conforme se aumenta la probabilidad de error o el número de caracteres.
- Se codifica y decodifica correctamente el mensaje. Si se produce una alteración en el resto, el resultado ha de indicar que se ha producido un error.
- El número de bytes se corresponde con el número de caracteres y si alguno de estos se repite, los correspondientes bits correspondientes a esos caracteres son iguales.
- El byte y el bit que se indica que tienen el error, se corresponde con dicho error.
- Los mensajes de si existe error o no se adecuan con los resultados.

Con estas observaciones, se puede decir que el código funciona correctamente.

4.1.5 Resultados del cuaderno de Token Bucket

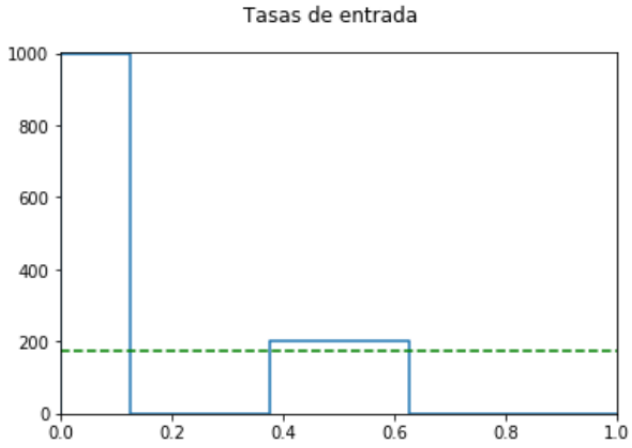
Para comprobar el funcionamiento del código, se han introducido los datos que vienen en uno de los boletines de ejercicios de la asignatura, concretamente el primer ejercicio del Seminario 6.

```
Introduzca tiempo 1ª ráfaga de datos: 0.125
Introduzca 1° tiempo sin transmitir: 0.25
Introduzca tiempo 2ª ráfaga de datos: 0.25
```

```
Introduzca 2° tiempo sin transmitir: 0.375
Introduzca tiempo 3ª ráfaga de datos: 0

Introduzca tasa de transferencia 1ª ráfaga de datos: 1000
Introduzca tasa de transferencia 2ª ráfaga de datos: 200
Introduzca tasa de transferencia 3ª ráfaga de datos: 0

La tasa media de transmisión es: 175.0
```




```
Introduzca capacidad del bucket: 200
Introduzca tasa de salida del bucket: 128

El exceso en la rágafa 1 es 100.0
El exceso en la rágafa 2 es 0.0

El tiempo que se puede estar transmitiendo a la tasa de la 1 ª ráfaga es:
0.16
```

Figura 4-13 Resultados obtenidos con los datos del problema (elaboración propia)

Aunque el código desarrollado sólo permite un valor constante de tasa de transmisión para cada ráfaga, se pueden realizar ejercicios básicos como los que se llevan a cabo en clase. Al comparar los resultados con la resolución del problema y comprobar que estos son los mismos, se puede validar el buen funcionamiento del código desarrollado.

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este apartado se presentarán las conclusiones que se han obtenido tras el desarrollo de este proyecto y se propondrán líneas futuras de desarrollo del mismo.

5.1 Conclusiones

El resultado de este proyecto es una herramienta informática como apoyo a la docencia para la asignatura Fundamentos de Redes de Ordenadores para que los alumnos de la asignatura dispongan de ella tanto en las horas lectivas de clase como en su tiempo libre y puedan ver en la práctica el funcionamiento de las técnicas vistas en clase. Con esta herramienta, tanto los profesores como los alumnos no tienen que instalar *software* costoso, sino que podrán descargarlo gratuitamente desde la página web oficial de Anaconda. Además, los ficheros de los cuadernos son poco pesados (del orden de 30 KB). Por otro lado, es una herramienta novedosa para el alumnado ya que actualmente no dispone de ninguna similar para los conceptos que abarca esta herramienta. De esta manera, instalando simplemente el *software,* el alumno dispone de una herramienta más como apoyo, haciendo más entretenidas sus labores de aprendizaje.

Teniendo en cuenta la dificultad de comprensión de algunos conceptos de la asignatura, la falta de una herramienta como ayuda para explicar estos conceptos y el buen funcionamiento de esta herramienta, se puede hacer un balance positivo del desarrollo del proyecto.

5.2 Líneas futuras

Para la mejora de esta herramienta, se propone:

- Desarrollar e incluir otro lote de cuadernos con otros conceptos de la asignatura.
- Simplificar la sintaxis de los códigos de los notebooks. Una herramienta para esto podría ser el uso el comando *def*.
- Solucionar errores leves en los cuadernos como pueden ser:
    - Al introducirse por error cadenas de caracteres o ningún valor que la celda código no pare de ejecutarse y vuelva a pedir los datos.
    - Solucionar los problemas que surgen con los códigos cuando se introducen números decimales pequeños.

6 BIBLIOGRAFÍA

[1] Manuel Antelo Majado, «Redes de ordenadores e Internet,» [En línea]. Available: https://www.edu.xunta.gal/centros/iesblancoamorculleredo/aulavirtual2/pluginfile.php/25655/mod_page/content/30/RedesComunicacion_ManuelAntelo_AlvaroA%C3%B1on_DavidFernandez.pdf. [Último acceso: 24 Enero 2020].

[2] UNAM, «Fundamentos de redes de computadoras,» [En línea]. Available: https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/956/mod_resource/content/1/contenido/index.html. [Último acceso: 24 Enero 2020].

[3] Escuela Superior de las Fuerzas Armadas, La Armada Española, 2ª Edición Corregida, 2012.

[4] Alfonso Vidal Romero Elizondo, «Redes de Computadoras,» [En línea]. Available: https://bibliotecavirtualapure.files.wordpress.com/2015/06/redes_de_computadoras-freelibros-org.pdf. [Último acceso: 24 Enero 2020].

[5] Cristian R. Cappo, «Crecimiento en la utilización de Internet,» [En línea]. Available: https://www.researchgate.net/figure/Crecimiento-en-la-utilizacion-de-Internet-desde-el-ano-1993-Debido-a-su-alta-utilizacion_fig1_283290277. [Último acceso: 25 Enero 2020].

[6] Profesorado de la asignatura Fundamentos de Redes de Ordenadores, «GUÍA DOCENTE DE FUNDAMENTOS DE REDES DE ORDENADORES,» [En línea]. Available: https://cud.uvigo.es/images/Documentacion/guiasdocentes/primer_cuatrimestre/2019_20/fro.pdf. [Último acceso: 26 Enero 2020].

[7] Profesorado de la asignatura Fundamentos de Redes de Ordenadores, «Tema 7: Calidad de servicio,» [En línea]. Available: https://cursos.faitic.uvigo.es/moodle3_1920/pluginfile.php/64537/mod_resource/content/0/7_QoS.pdf. [Último acceso: 26 Enero 2020].

[8] Eduardo Cabrera Granado y Elena Díaz García, «MANUAL DE USO DE JUPYTER NOTEBOOK PARA APLICACIONES DOCENTES,» [En línea]. Available: https://eprints.ucm.es/48304/1/ManualJupyter.pdf. [Último acceso: 26 Enero 2020].

[9] Leonardo Emiro Contreras Bravo y Julián Alfonso Tristancho Ortiz, «Uso de las herramientas informáticas educacionales para la enseñanza de la resistencia de materiales,»

[En línea]. Available: https://revistavirtual.ucn.edu.co/index.php/RevistaUCN/article/viewFile/825/1343. [Último acceso: 26 Enero 2020].

[10] Manuel Zaforas, «¿Es Python el lenguaje del futuro?,» [En línea]. Available: https://www.paradigmadigital.com/dev/es-python-el-lenguaje-del-futuro/. [Último acceso: 11 Febrero 2020].

[11] Eneko, «Python en programación con redes,» [En línea]. Available: https://www.solvetic.com/tutoriales/article/396-python-programacion-con-redes/. [Último acceso: 11 Febrero 2020].

[12] Ramón Invarato, «Nuestro propio servidor Software (Socket con Python),» [En línea]. Available: https://jarroba.com/nuestro-propio-servidor-software-socket-con-python/. [Último acceso: 11 Febrero 2020].

[13] Christian Tutivén Gálvez, «Empezando a usar Jupyter Notebook para Python,» [En línea]. Available: https://medium.com/saturdays-ai/empezando-a-usar-jupyter-notebook-para-python-parte-1-instalaci%C3%B3n-94e97b4c5f37. [Último acceso: 12 Febrero 2020].

[14] «Google Colab,» [En línea]. Available: https://colab.research.google.com/notebooks/intro.ipynb. [Último acceso: 12 Febrero 2020].

[15] Juan Antonio Torres y Daniel Arias Figueroa, «“Herramientas de Software de Simulación para Redes de Comunicaciones",» [En línea]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/52857/Documento_completo.pdf-PDFA.pdf?sequence=1. [Último acceso: 29 Enero 2020].

[16] Teresa L. Fabuel, «¿Qué es KivaNS?,» [En línea]. Available: http://www.aurova.ua.es/kiva/. [Último acceso: 29 Enero 2020].

[17] Carlos Binker y Alejandro Pérez, «Emulación de elementos de networking interactuando con máquinas virtuales,» [En línea]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/56542/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y. [Último acceso: 27 Enero 2020].

[18] U.S. NAVAL RESEARCH LABORATORY, «Common Open Research Emulator (CORE),» [En línea]. Available: https://www.nrl.navy.mil/itd/ncs/products/core. [Último acceso: 29 Enero 2020].

[19] Alex Castillo, «Como crear red wan Cisco Packet Tracer,» [En línea]. Available: https://www.youtube.com/watch?v=eO2waH0D_Os. [Último acceso: 1 Marzo 2020].

[20] Jae Kim, «CORE: A real-time network emulator,» [En línea]. Available: https://www.researchgate.net/publication/261091924_CORE_A_real-time_network_emulator. [Último acceso: 29 Febrero 2020].

[21] Telectronika, «GNS3 Guía Introductoria: Características y Requerimientos Mínimos,» [En línea]. Available: https://telectronika.com/articulos/ti/que-es-gns3/. [Último acceso: 2 Marzo 2020].

[22] Profesorado de la asignatura Fundamentos de Redes de Ordenadores, «Simulador de redes GNS3,» [En línea]. Available: https://cursos.faitic.uvigo.es/moodle3_1920/pluginfile.php/59132/mod_resource/content/3/practica2.pdf. [Último acceso: 27 Enero 2020].

[23] Universidad de Alicante, «KivaNS,» [En línea]. Available: https://blogs.ua.es/redesitis/recursos-didacticos/. [Último acceso: 29 Enero 2020].

[24] Miguel Arevalillo-Herráez, «Un juego de ordenador para el aprendizaje la configuración de los componentes de red,» [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2099/11820/r50.pdf. [Último acceso: 30 Enero 2020].

[25] Cisco Systems, «Network Defenders,» [En línea]. Available: https://www.insite.net/cisco_games/game_page.html?userName=Guest&game=security. [Último acceso: 30 Enero 2020].

[26] Julio Jaime, «Conectate, vestite y juga con Cisco Systems,» [En línea]. Available: http://d3ny4ll.blogspot.com/2009/05/conectate-vestite-y-juga-con-cisco.html. [Último acceso: 30 Enero 2020].

[27] UPONIC PARA DISEÑADORES WEB, «JUEGO CISCO ASPIRE CCNA,» [En línea]. Available: http://ingsistemasdecomputadoras.blogspot.com/2014/11/juego-cisco-aspire-ccna.html. [Último acceso: 30 Enero 2020].

[28] awasthi7xenextt, «Introduction to Wireshark,» [En línea]. Available: https://www.geeksforgeeks.org/introduction-to-wireshark/. [Último acceso: 3 Febrero 2020].

[29] Universidad de Navarra, «Práctica3 - Analizadores de red: Wireshark y tcpdump,» [En línea]. Available: https://www.tlm.unavarra.es/~daniel/docencia/arss/arss10_11/practicas/practica3.pdf. [Último acceso: 3 Febrero 2020].

[30] Profesorado de la asignatura Fundamentos de Redes de Ordenadores, «Introducción a Wireshark,» [En línea]. Available: https://cursos.faitic.uvigo.es/moodle3_1920/pluginfile.php/62319/mod_resource/content/1/practica3.pdf. [Último acceso: 3 Febrero 2020].

[31] Wikipedia, «Wireshark,» [En línea]. Available: https://es.wikipedia.org/wiki/Wireshark. [Último acceso: 3 Febrero 2020].

[32] «A gallery of interesting Jupyter Notebooks,» [En línea]. Available: https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks#machine-learning-statistics-and-probability. [Último acceso: 4 Febrero 2020].

[33] «How to get started coding in Python notebook,» [En línea]. Available: https://nbviewer.jupyter.org/github/Tanu-N-Prabhu/Python/blob/master/How_to_get_started_coding_in_Python%3F.ipynb. [Último acceso: 8 Febrero 2020].

[34] «Learning to code with Python notebook,» [En línea]. Available: https://nbviewer.jupyter.org/urls/bitbucket.org/amjoconn/watpy-learning-to-code-with-python/raw/3441274a54c7ff6ff3e37285aafcbbd8cb4774f0/notebook/Learn%20to%20Code%20with%20Python.ipynb. [Último acceso: 8 Febrero 2020].

[35] Caleb Madrigal, «Sound Analysis with the Fourier Transform and Python,» [En línea]. Available: https://github.com/calebmadrigal/FourierTalkOSCON. [Último acceso: 8 Febrero 2020].

[36] Caleb Madrigal, «Audio Filtering,» [En línea]. Available:

https://github.com/calebmadrigal/FourierTalkOSCON/blob/master/09_AudioFiltering.ipynb. [Último acceso: 8 Febrero 2020].

[37] Jeff Kantor, «CBE20255,» [En línea]. Available: http://jckantor.github.io/CBE20255/. [Último acceso: 9 Febrero 2020].

[38] Jeff Kantor, «Torpedo propulsion notebook,» [En línea]. Available: https://nbviewer.jupyter.org/github/jckantor/CBE20255/blob/master/notebooks/08.07-Torpedo-Propulsion.ipynb. [Último acceso: 9 Febrero 2020].

[39] Jeff Kantor, «Energy balances for a steam turbine notebook,» [En línea]. Available: https://nbviewer.jupyter.org/github/jckantor/CBE20255/blob/master/notebooks/08.04-Energy-Balances-for-a-Steam-Turbine.ipynb. [Último acceso: 9 Febrero 2020].

[40] Caleb Fangmeier, «Imagination notebook,» [En línea]. Available: https://github.com/cfangmeier/Small/blob/9037dbd6d7e5843a60d7a92c6e4a713b3b685a74/Imagination.ipynb. [Último acceso: 9 Febrero 2020].

[41] Caleb Fangmeier, «Matplotboard notebook,» [En línea]. Available: https://github.com/cfangmeier/matplotboard. [Último acceso: 9 Febrero 2020].

[42] Andres Marrugo, «Sensors and actuators notebook,» [En línea]. Available: https://github.com/agmarrugo/sensors-actuators/tree/master/notebooks. [Último acceso: 10 Febrero 2020].

[43] Andrés Marrugo, «Position sensor notebook,» [En línea]. Available: https://github.com/agmarrugo/sensors-actuators/blob/master/notebooks/position-sensor.ipynb. [Último acceso: 10 Febrero 2020].

[44] Eugenio Angriman, «Networkit user guide notebook,» [En línea]. Available: https://github.com/networkit/networkit/blob/Dev/notebooks/User-Guide.ipynb. [Último acceso: 10 Febrero 2020].

[45] Charmaine Ndolo, «Centrality notebook,» [En línea]. Available: https://github.com/networkit/networkit/blob/7e88535260aa60f9e2b779afa6984a6379374f8a/notebooks/Centrality.ipynb. [Último acceso: 10 Febrero 2020].

[46] Wikipedia, «NetworkX,» [En línea]. Available: https://en.wikipedia.org/wiki/NetworkX. [Último acceso: 11 Febrero 2020].

[47] Stacy Meichle, «NetworkX notebook,» [En línea]. Available: https://github.com/smeichle/NetworkX-sandbox/blob/master/20180801-NetworkX-Social-Network.ipynb. [Último acceso: 11 Febrero 2020].

[48] Luigys Toro, «Anaconda Distribution: La Suite más completa para la Ciencia de datos con Python,» [En línea]. Available: https://blog.desdelinux.net/ciencia-de-datos-con-python/. [Último acceso: 12 Febrero 2020].

[49] «Anaconda,» [En línea]. Available: https://www.anaconda.com/. [Último acceso: 45 Febrero 2020].

[50] «How to Add Julia to Jupyter Notebook,» [En línea]. Available: https://datatofish.com/add-julia-to-jupyter/. [Último acceso: 12 Febrero 2020].

[51] Juan Antonio Ramírez Peña, «Herramienta informática de apoyo para la asignatura "Fundamentos de Redes de Ordenadores",» [En línea]. Available:

https://docs.google.com/forms/d/e/1FAIpQLSd14r_hKqyMxIv859NTNg3wWuxijdGMEt_Bwg5eqeGKpj1hXQ/viewform. [Último acceso: 12 Febrero 2020].

[52] Vern Paxson, «Computing TCP's Retransmission Timer,» [En línea]. Available: https://tools.ietf.org/html/rfc6298. [Último acceso: 2020 Marzo 02].

ANEXO I: CÓDIGOS DESARROLLADOS EN LOS NOTEBOOKS

El objeto de este anexo es adjuntar los códigos completos de cada uno de los cuadernos.

- Código del canal binario simétrico

```
#Librerías
import random

#Usuario introduce probabilidad de error correctamente
Per = float(input("Introduzca probabilidad de error: "))
while Per > 0.5 or Per < 0:
    print("P.error ha de tener un valor comprendido entre 0 y 1/2")
Per = float(input("Introduzca probabilidad de error: "))


#Datos de entrada
print()
entrada = int(input("Introduzca manualmente secuencia de bits: "),2)
bits = [int(x) for x in bin(entrada)[2:]]

#Errores en el canal de transmisión
errores = 0
q = 1 – Per
bitsconerror = []
for bit in bits:
      if random.random() < q:
          bitsconerror.append(bit)
      else:
         if bit == 0:
             bitsconerror.append(1)
         else:
             bitsconerror.append(0)
          errores = errores + 1
#Mostramos en pantalla al usuario lo que ha sucedido
salida = ''.join(map(str, bitsconerror))
print("Tras la transmisión de",bin(entrada),"por el canal, la salida queda:",salida)
if bits == bitsconerror:
    print("La secuencia de bits no ha sido modificada al transmitirse por el canal")
else:
    print("La secuencia de bits ha sido modificada al transmitirse por el canal")
print()
print("La secuencia a la entrada era:", bits)
```

```
print("La secuencia a la salida es  :",bitsconerror)
print()
print("Con un error de ",errores," bits")
```

Figura I-1 Código del canal binario simétrico

- Código del número medio de intentos de transmisión

```
#Librerías
import numpy as np
import matplotlib.pyplot as plt


#Datos y listas necesarias
pexito = float(input("Introduzca probabilidad de transmisión de mensaje correcta: "))
while pexito < 0 or pexito > 1:
        pexito = float(input("Introduzca probabilidad de transmisión de mensaje correcta: "))
listapmf = []
listacdf = []

#Cálculo número de intentos
intentos = 1/pexito
redondeo = round(intentos)
print()
print("El número medio de intentos para la transimión del mensaje es", intentos)

if redondeo < 10:
   redondeo = 10

#Cálculo PMF y CDF
    for i in range(redondeo):
        pmf = (((1-pexito)**(i-1))*pexito)
        listapmf.append(pmf)
        cdf = (1-((1-pexito)**i))
        listacdf.append(cdf)

#Representación de gráficas
arraycdf = np.array(listacdf)
arraypmf = np.array(listapmf)
x1 = np.arange(1, redondeo, 1)
x2 = np.arange(1, redondeo, 1)

plt.subplot(2, 1, 1)
plt.plot(x1, arraypmf, '.-')
plt.title('Intentos de transmisión en base a una probabilidad de error de mensaje')
plt.ylabel('PMF')
```

```
plt.subplot(2, 1, 2)
plt.plot(x2, arraycdf, '.-')
plt.xlabel('Intentos')
plt.ylabel('CDF')

plt.show()
```

Figura I-2 Código del número medio de intentos de transmisión

- Código del cálculo de temporizadores adaptativos TCP

```
#Librerías necesarias
import random
import math
import numpy as np
import matplotlib.pyplot as plt


#Ponderaciones
α = float(input("Introduzca valor de α ""(se recomienda 0.125)"": "))
while α < 0:
    α = float(input("Introduzca valor de α ""(se recomienda 0.125)"": "))
β = float(input("Introduzca valor de β ""(se recomienda 0.25)"": "))
while β < 0:
    β = float(input("Introduzca valor de β ""(se recomienda 0.25)"": "))


#Listas y variables necesarias
print()
SRTT = []
RTT = []
Varianzas = []
Temporizadores = []
n = 0
tempRTT = ()


#Usuario decide como introducir valores
respuesta = str(input("¿Desea introducir manualmente los valores de RTT o que sean generados? (introducir/generar): "))
while (respuesta != 'introducir') and (respuesta != 'generar'):
```

```
        print()
        print("Respuesta incorrecta")
        print()
        respuesta = str(input("¿Desea introducir manualmente los valores de RTT o que sean generados? (introducir/generar): "))

    #Se introducen valores de RTT manualmente
    if respuesta == 'introducir':
        while tempRTT != 0:
            print()
            tempRTT = float(input("Introduzca valor de RTT: "))
            while tempRTT < 0:
                print()
                print("Valor de RTT no válido")
                print()
                tempRTT = float(input("Introduzca valor de RTT: "))
            RTT.append(tempRTT)

    #Cálculos para primer valor de RTT introducido según estándar
        if n == 0:
            SRTT.append(tempRTT)
            SVAR = SRTT[0]/2
            Varianzas.append(SVAR)
            TemporizadorTCP = SRTT[0] + 4*Varianzas[0]
            Temporizadores.append(TemporizadorTCP)
            print()
            print("Se estima que el valor del SRTT es", tempRTT)
            print("Se estima que el valor del RTTVAR es", SVAR)
            print("El intervalo de tiempo de vencimiento del temporizadorTCP es", TemporizadorTCP," segundos")
            print()
            print("Si deseea finalizar introduzca un 0")
            n = n + 1

    #Cálculo para los siguientes RTT
        elif n > 0:
```

```
            SVAR = ((1-β)*Varianzas[n-1])+(β*(math.fabs(tempRTT-tempSRTT)))
            Varianzas.append(SVAR)
            tempSRTT = ((1-α)*SRTT[n-1])+(α*tempRTT)
            SRTT.append(tempSRTT)
            TemporizadorTCP = SRTT[n] + 4*Varianzas[n]
            Temporizadores.append(TemporizadorTCP)
            print()
            print("Se estima que el valor del SRTT es", tempSRTT)
            print("Se estima que el valor del RTTSVAR es", SVAR)
            print("El intervalo de tiempo de vencimiento del temporizadorTCP es",
            TemporizadorTCP," segundos")
            print()
            print("Si desea finalizar introduzca un 0")
            n = n + 1


    #Se generan valores de RTT según una distribución uniforme
    elif respuesta == 'generar':
        print()
        maxi = float(input("Introduzca valor máximo de RTT: "))
        mini = float(input("Introduzca valor mínimo de RTT: "))
        Num = int(input("Introduzca número de valores a generar: "))
        while maxi <= 0 or mini <= 0 or Num <= 0:
            print()
            print("Uno o varios valores introducidos son erróneos")
            print()
            maxi = float(input("Introduzca valor máximo de RTT: "))
            mini = float(input("Introduzca valor mínimo de RTT: "))
            Num = int(input("Introduzca número de valores a generar: "))
        RTT = np.random.uniform(mini,maxi,Num)
    #Cálculos
        primera_vez = 1
        for i in RTT:
            #Cálculos para primer valor de RTT generado según estándar
            if primera_vez == 1:
                tempRTT = RTT[0]
                SRTT.append(tempRTT)
                RTTVAR = (RTT[0])/2
```

```
            Varianzas.append(RTTVAR)
            TemporizadorTCP = SRTT[0] + 4*Varianzas[0]
            Temporizadores.append(TemporizadorTCP)
        #Cálculo para los siguientes RTTs generados
          else:
             tempSRTT = ((1-α)*SRTT[n-1])+(α*i)
              SRTT.append(tempSRTT)
              SVAR = ((1-β)*Varianzas[n-1])+(β*(math.fabs(i-tempSRTT)))
              Varianzas.append(SVAR)
              TemporizadorTCP = SRTT[n] + 4*Varianzas[n]
              Temporizadores.append(TemporizadorTCP)
              print()
              print("Se estima que el valor del siguiente RTT es", tempSRTT)
              print("Se estima que el valor del siguiente SVAR es ",SVAR)
              print("El intervalo de tiempo de vencimiento del temporizadorTCP es",
              TemporizadorTCP," ms")
              print()

    #Representación de RTT, SRTT y RTO
    line_chart1 = plt.plot(range(0,n), SRTT)
    line_chart2 = plt.plot(range(0,n), RTT)
    line_chart3 = plt.plot(range(0,n), Temporizadores)
    plt.title('Temporizadores adaptativos TCP')
    plt.xlabel('Muestras')
    plt.ylabel('RTT (ms)')
    plt.legend(['SRTT', 'RTT', 'RTO'], loc=4)
    plt.show()
```

Figura I-3 Código del cálculo de temporizadores adaptativos TCP

- Código del cálculo de CRC

```
import random
from array import array

# Usando el polinomio 0x1D de CRC-8
def crc8(datos):
    resto = 0
    for byte in datos:
        resto = resto ^ byte
        for j in range(0,8):
            if (resto & 0x80):
                resto = (resto << 1) ^ 0x1D
            else:
                resto = (resto << 1)
        resto = resto & 0xFF
    # Descomentar para imprimir los restos parciales
    # print hex(resto)
      return resto

texto = str(input("Introduzca mensaje a transmitir: "))
prob_error = float(input("Introduzca probabilidad de error: "))
datos = array('B', texto.encode())

# Función de conversión a binario
to_bin = lambda x: str(bin(x)[2:].zfill(8))

print("\n======== TRANSMISOR: DATOS ==========")
print("|".join(map(to_bin, datos)))

# Calculamos el CRC (se hace en el transmisor)
print("\n======== TRANSMISOR: CALCULO CRC8 ==========")
resto = crc8(datos)
print("Resto: " + str(to_bin(resto)))

# Añadimos el resto (CRC) al mensaje
print("\n======== TRANSMISOR: DATOS CON CRC ==========")
datos.append(resto)
print("|".join(map(to_bin, datos)))
```

```
# Añadimos aleatoriamente un error en un bit con una probabilidad dada
print("\n======== CANAL ==========")
if random.random() < prob_error:
    # Hay que introducir error, elegimos la posición al azar
    posicion_byte = random.choice(range(len(texto)))
    posicion_bit = random.choice(range(8))
    datos[posicion_byte] = datos[posicion_byte] ^ (0x80 >> posicion_bit)
    # Sumamos uno por claridad: la primera posición es en realidad la 0
   # Tanto para el byte como para el bit se empieza a contar por la izquierda
    print("Error en el byte " + str(posicion_byte+1) + " y bit " +  str(posicion_bit+1))
    bits = list("".join(map(to_bin, datos)))
    bits[posicion_byte*8+posicion_bit] = 'X'
    print("".join(bits))
else:
    print("No se introduce un error")

print("\n======== RECEPTOR: DATOS RECIBIDOS ==========")
print("|".join(map(to_bin, datos)))

# Verificamos el CRC (esto lo hace el receptor)
print("\n======== RECEPTOR: VERIFICAR CRC8 ==========")
resto = crc8(datos)
if resto == 0:
    print("Se han recibido los datos sin error")
else:
    print("Se ha producido un error en la transmisión de datos")
    print("Resto: " + str(to_bin(resto)))
```

Figura I-4 Código del cálculo del CRC

```
T = float(input("Introduzca tiempo 1ª ráfaga de datos: "))
Tiempos.append(T)
T = float(input("Introduzca 1º tiempo sin transmitir: "))
Tiempos.append(T)

if T != 0:
```

```
    x = 2
    q = 2
    w = 1
    while T != 0:
        if w%2 != 0:
T = float(input("Introduzca tiempo en segundos" + str(x) +"ª ráfaga de datos: "))
            x = x + 1
            w = w + 1
            Tiempos.append(T)
        else:
T = float(input("Introduzca " + str(q) +"º tiempo en segundos sin transmitir: "))
            q = q + 1
            w = w + 1
            Tiempos.append(T)

#Código para que el usuario introduzca la velocidad de transmisión
Tasas = []
print()
Tr = float(input("Introduzca tasa de transferencia 1ª ráfaga de datos en Mbps: "))
Tasas.append(Tr)

if Tr != 0:
    k = 2
    p = 0
    while Tr != 0:
        if w%2 != 0:
Tr = float(input("Introduzca tasa de transferencia " + str(k) +"ª ráfaga de datos en Mbps: "))
            k = k + 1
            Tasas.append(Tr)
            w = w + 1
        else:
            Trr = 0
            Tasas.append(Trr)
            w = w + 1
            p = p + 1
```

```
#Repetimos todo si hay error al introducir el número de datos
while k != x:

    Tiempos = []
    T = float(input("Introduzca tiempo en segundos 1ª ráfaga de datos: "))
    Tiempos.append(T)
    T = float(input("Introduzca 1º tiempo en segundos sin transmitir: "))
    Tiempos.append(T)

    if T != 0:
        x = 2
        q = 2
        w = 1
        while T != 0:
            if w%2 != 0:
T = float(input("Introduzca tiempo en segundos " + str(x) +"ª ráfaga de datos: "))
                x = x + 1
                w = w + 1
                Tiempos.append(T)
            else:
T = float(input("Introduzca " + str(q) +"º tiempo en segundos sin transmitir: "))
                q = q + 1
                w = w + 1
                Tiempos.append(T)

    Tasas = []
    print()
    Tr = float(input("Introduzca tasa de transferencia 1ª ráfaga de datos en Mbps: "))
    Tasas.append(Tr)

    if Tr != 0:
        k = 2
        p = 0
        while Tr != 0:
            if w%2 != 0:
```

```
        Tr = float(input("Introduzca tasa de transferencia " + str(k) +"ª ráfaga de datos en Mbps: "))
                k = k + 1
                Tasas.append(Tr)
                w = w + 1
            else:
                Trr = 0
                Tasas.append(Trr)
                w = w + 1
                p = p + 1

    if k != x:
        print()
        print("ERROR, NÚMERO DE TIEMPOS O DE TASAS INCORRECTO")

#Cálculo de tasa media de transmisión
Datos = []
z = 0
v = 0
for i,j in zip(Tasas,Tiempos):
        Dato = i*j
        Datos.append(Dato)

suma = sum(Datos)
tiempototal = sum(Tiempos)
Tasamedia = suma/tiempototal
print()
print("La tasa media de transmision es:",Tasamedia)

#Suma de tiempos para representación
sumatiempos = []
sumatiempos.append(0)
time = ()
c = 1
for i in Tiempos:
    if c == 1:
        time = i
```

```
            sumatiempos.append(time)
            c = c + 1
        else:
            time = i + sumatiempos[c-1]
            sumatiempos.append(time)
            c = c + 1


    #Representación
    maximo = Tasas[0]
    for i in Tasas:
        if i > maximo:
            maximo = i

    representacion = []
    representacion.append(0)
    for i in Tasas:
        representacion.append(i)
    x = np.array(sumatiempos)
    y = np.array(representacion)
    plt.suptitle("Tasas de entrada")
    plt.axis([0,sumatiempos[c-1],0,maximo+1])
    plt.step(x, y)
    plt.axhline(y=Tasamedia, xmin=0, xmax=maximo, color = "green", linestyle = "dashed")
    plt.show()

    #Cálculo excesos
    print()
    capacidad = float(input("Introduzca capacidad del bucket: "))
    salida = float(input("Introduzca tasa de salida del bucket: "))

    excesos = []
    s = 0
    for h in Tasas:
        if h != 0:
            exceso = (h-capacidad)*Tiempos[s]
            excesos.append(exceso)
            s = s + 2

    e = 0
    for y in excesos:
```

```
    if y >= 0:
        print("El exceso en la rágafa",e+1,"es",excesos[e])
        e = e+1
    else:
        print("En la rágafa",e+1,"no hay exceso")
        e = e+1

#Cálculo tiempos en base a la tasa
tx = ()
nume1 = 0
Tasas1 = []

for i in Tasas:
    if i > 0:
        Tasas1.append(i)

for i in Tasas1:
    if i > capacidad:
        tx = (salida/(i-capacidad))
print("El tiempo que se puede estar transmitiendo a la tasa de la",nume1+1,"ª ráfaga es:",tx)
        nume1 = nume1 + 1
    else:
        nume1 = nume1 + 1
```

Figura I-5 Código del algoritmo de Token Bucket