Introducción

Actualmente el tema de anonimato se encuentra cada vez más en el plano principal de discusión en el mundo. Esto debido a que en el pasado, pueblos han sufrido y su historia se ha visto manchada.

Debido a ello se han promulgado acuerdos internacionales en pro de los derechos humanos. Y uno en especial (Artículo 19) de la libertad de expresión. A pesar que hoy 2021 nos encontramos en un momento privilegiado en la historia, hay países donde aún se siguen teniendo problemas de libertad de expresión. Desde el no poder comunicarse, debido aque las TELCOS locales no les interesa invertir en pueblos donde tiene pérdidas, lugares donde el gobierno calla a sus habitantes y hasta lugares donde la privacidad es una cuestión de vida o muerte. Es por todo esto, que es una buena idea producir nuevas alternativas de comunicación. Por ejemplo, las redes mesh de proyectos como guifi.net y openmesh. Por lo anterior, el objetivo del proyecto es conocer distintas soluciones que nos permitan acercarnos más a esa libertad de expresión sin excepciones.

Para esto, hemos implementado un cliente y un servidor IRC, esto para poder comunicarnos a través de un medio físico, para el cual hemos escogido Morse como medio de comunicación, siendo capaz de emitir un mensaje (o bien, la información de un paquete) a través del sonido gracias a Morse y poder recibirlo en otros puntos de nuestra red modelando el comportamiento de una red Mesh.

Ambiente de desarrollo

- Oracle Virtual Box:
  Sistema emulador de Ubuntu para utilizar como sistema operativo.
- Sublime y Spyder:
  Editor de texto para los códigos fuente escritos en Python.
- GitHub:
  Servicio de repositorio y control de versiones.
  https://github.com/Emanlui/WaveNet
- Bibliotecas:
  scapy
  sounddevice
  pydub (ffprobe, ffmpeg, frei0r-plugins)
  openssl

scipy
sockets
morseToText
pulseAudio
pavuControl

Estructuras de datos usadas y funciones

En la clase de sonido tenemos:
* encrypt: recibe un mensaje (string), lo cifra a Morse y lo retorna
* increase_volume: recibe un nombre de archivo de audio para subirle los db.
* Class Sonido: Instancia del objeto con las funcionalidades principales
* recordAudio: recibe los segundos durante los cuales se van a grabar audio y se genera el "output.wav"
* decodeAudio: se encarga de descifrar el contenido de un archivo de audio en formato wav que contenga morse.
* buildAudio: emite los sonidos Morse a partir de un string dado, emite el sonido en consola y lo guarda en "emittedMessage.wav"
* menu: Se encarga de la mini-interfaz y los inputs del usuario.

* Clase CPPM: es el protocolo creado con Scapy, se implementaron las funciones validateChecksum() y
setCheksum(), se encargan de la lógica de validación del checksum del mensaje y poder verificar las consistencia de los datos enviados en el paquete.
* Clase Service: posee funciones como createPacket(), sendPacket(), packetToBytes(), bytesToPacket(), encryptPacket(), decryptPacket().
Estas funciones se encargan de manejar paquetes de Scapy y realizer tareas necesarias para el procesamiento de los paquetes al momento de enviar y recibir estos.

* IRC
- serverManagement()
Es la función encargada de manejar el IRC, realizar la conexión.
- messageManagementIRC()
Es la función encargada de manejar los mensajes que viajan al IRC.

Instrucciones para ejecutar el programa

Cliente de sonidos
Para ejecutar el cliente, solo deberemos realizar en la terminal
* python3 sound.py
Con esto, veremos tres opciones para el menu principal, que consisten en
* 1) Emitir Mensaje
* 2) Grabar mensaje
* 3) Decodificar Mensaje

Seguidamente, al emitir un mensaje solo debemos poner un texto cualquiera (evitar el exceso de caracteres especiales, por que no todos estan contemplados por Morse) y seguidamente, se empezará a reproducir el audio respectivo. Al finalizar, será guardado en "emittedMessage.wav" para su futura revisión.
En la opción de grabar un mensaje, deberemos indicar cuantos segundos queremos que nuestra terminal grabe sonido, para esto usamos el PulseAudio que nos permite escuchar los sonidos de la computadora y no solo del micrófono a la hora de hacer una grabación (recomiendo silenciar el micrófono ya que podría agregar estática a la grabación), al finalizar, el audio grabado será guardado en "output.wav" y será decodificado, escribiendo el mensaje cifrado en "output.txt"
En la opción de decodificar un mensaje, tendremos tres opciones
* 1) output.wav
* 2) emittedMessage.wav
* 3) Personalizado
Las primeras dos opciones nos permiten analizar rapidamente audios generados previamente y la opción personalizada es cuando deseamos analizar un archivoo ajeno, para esto solo debemos ingresar el nombre del archivo y lo buscará para ser decodificado.

Cliente y el server

python3 server.py [ip del server] [puerto del server] [nombre del IRC]
python3 client.py [ip local del cliente] [puerto local del cliente] [ip del server] [puerto del server] [nombre]

Todos los puertos deben de ser diferentes.

Actividades realizadas por estudiante

Nota: Se asume que todas las actividades descritas en esta sección fueron desarrolladas en partes iguales por cada uno de los integrantes del grupo.

- Mayo 10: 2 horas
Investigación de estructuras a utilizar y librerías de utilidad, todo muy general.

- Mayo 15: 3 horas
Primeros ejemplos de código para conocer el funcionamiento general de las estructuras investigadas y comenzar a planear una solución al problema a resolver.

- Mayo 20: 4 horas
Definicion de issues para enfocar areas iniciales de trabajo.

- Mayo 25 - 31: 3 horas
Trabajo individual por parte de cada uno en los issues definidos anteriormente, alrededor de un par de horas por día.

- Mayo 4: 6 horas
Resolución de errores y correción de problemas en cada área.

- Mayo 5: 6 horas
Unión de cada una de las partes y estudio de errores resultantes para proponer una nueva solución.

- Mayo 6: 6 horas
Solución y unión casi exitosa del programa.

Comentarios finales(estado del programa)

Server IRC:
El server funciona bien, recibe los mensajes de los clientes, asi como cuando se une o se sale un cliente este mismo avisa a las demas instancias cuales clientes están activos o no.
Se utilizó el servicio de DigitalOcean para tenerlo siempre corriendo en la nube.

Medio Físico: Sonido
Se logra implementar un cliente capaz de emitir, grabar y decodificar sonidos.
A la hora de emitir un sonido, se creará un archivo que conserva el total del sonido emitido.
A la hora de grabar un sonido, se inicia una grabación para escuchar a la otra terminal que emite el mensaje y lo guarda en un archivo local.
A la hora de decodificar, se ofrecen tres opciones, las primeras dos son para decodificar los archivos comunes generados por la emisión y grabación, pero también se incluye una tercera opción para decodificar archivos específicos con solo ingresar el nombre del archivo.
Desafortunadamente, por alguna razón de bitrate o codecs, a la hora de emitir un mensaje en una terminal, el audio generado no es correctamente interpretado por el decodificador, tampoco al ser grabado por la otra terminal, sin embargo al generar un audio morse (ejemplo, desde https://morsedecoder.com/ el audio es decodificado exitosamente) esto a pesar de los audios tener la misma frecuencia, unidades de periodo y volumen/sonidos.

Para el desarrollo del protocolo mediante la utilización de Scapy, algunos problemas encontrados al inicio del desarrollo fueron los siguientes:
- ¿Cómo enviar el paquete?
- ¿Cómo recibir el paquete?
- Una vez recibido el paquete ¿Cómo saber si el paquete es del tipo del paquete nuestro?
- ¿Cómo recuperar los datos que contiene el paquete?
- ¿Qué atributos necesita el protocolo custom para resolver el problema?

Conclusiones

- Emanuelle:
Este proyecto me llamó bastante la atención porque pudimos ser capaces de recrear un tipo de topología estilo onion, donde todo es completamente anónimo, permitiéndonos a nosotros poder tener a mano un sistema para poder mandar paquetes de manera anónima. El uso del sonido fue algo bastante interesante, especialmente a la hora de buscar soluciones para poder contrarrestar el ruido de ambiente.

- Fabrizio:
Me pareció interesante el conocer más a profundidad él funcionamiento de creación de paquetes y protocolos, para que con estos elementos poder formar una abstracción de red (junto con servers claro) y así permitir la comunicacion entre diferentes puntos mediante el medio fisico implementado (sonido)
Adicionalmente creo que si es un poco compleja la abtracción y el funcionamiento de la red en sí, por lo que personalmente si sentí la curva de aprendizaje un poco alta, pero con su debida invesytigación se aclaró bien.

- Kevin:
Logré aprender mucho sobre las opciones que se ofrecen a la hora de establecer comunicaciones y como las abstracciones de comunicar información tienen demasiadas aplicaciones en el mundo real. Aparte de eso, la interacción para enviar un simple mensaje posee una complejidad bastante elaborada, que hoy en día damos por asentado gracias a otros servicios de terceros que lamentablemente no sabemos que pueden llegar a hacer con esa información y hasta espiarnos. A futuros estudiantes, les aconsejo indagar a fondo sobre todas las propiedades de su medio de comunicación (en mi caso, los audios) por que algo tan pequeño como bitrate o codecs, pueden alterar el contenido de su información.

- Marco:

Aunque al inicio me costó lograr entender el problema mi primera impresión del proyecto es que iba a ser relativamente sencillo de desarrollar, pero conforme se va avanzando con el desarrollo aparecen problemas que son complicados de resolver, tanto problemas de código como a nivel físico, siento que pudimos resolver muchos de estos problemas de manera eficiente y creativa, esto es satisfactorio y estoy contento con lo aprendido y realizado.

Bibliografía
* https://code.google.com/archive/p/morse-to-text/source/default/source
* https://python-sounddevice.readthedocs.io/en/0.4.1/
* https://github.com/jiaaro/pydub

* https://www.scipy.org/
* https://pypi.org/project/playsound/
* https://www.freedesktop.org/wiki/Software/PulseAudio/
* https://scapy.readthedocs.io/en/latest/build_dissect.html
* https://scapy.readthedocs.io/en/latest/build_dissect.html#layers