Instituto Tecnológico de Costa Rica
Redes de Computadoras
Profesor: Kevin Moraga
Proyecto 1: TEC Land
Estudiantes:
Daniel Solís Méndez
Melvin Elizondo Pérez
I Semestre 2016

Introducción

El proyecto consiste en realizar una reimplementación de algunas de las funciones de varias capas del modelo OSI, generando un protocolo propio para el envío de mensajes a través de la red y definiendo un RFC para el mismo.

Capa 1: Medio Físico

En el proyecto el medio físico por utilizar es la luz. Para su transmisión se hará uso de una Raspberry Pi, un LED, un receptor de luz, un capacitor y algunos cables. El objetivo es convertir los paquetes en un binario que será representado con la presencia o la ausencia de luz, e interpretar estas señales en el otro extremo pudiendo reconstruir el mensaje y obtener cada uno de los caracteres que forman parte del texto.

Capa 2 y 3: TEC-land

Consiste en una red que funciona sobre TCP/IP mediante la cual se realiza el envío de mensajes de texto de un nodo a otro. Los diferentes nodos centrales se encargan de realizar el ruteo de los paquetes a través de los distintos dispositivos que se encuentran asociados. De esta manera se pueden usar nombres de usuario que representen Host de la red, para enviar y recibir mensajes.

Capa 4: Aplicación

El propósito de TEC-land es ser una red de chat que permita la comunicación entre personas en distintas localizaciones. Esto se da a nivel interno de la red, permitiendo que las comunicaciones sean privadas. Existe también la opción de enviar mensajes que salgan al "clear-net" y sean accesibles por todo el público, como también se pueden mandar "broadcast" a lo interno de la red.

Además en esta capa se debe realizar la instalación de un servidor IRC y un servidor de noticias NNTP, para que los mensajes catalogados como públicos pueden ser vistos por las personas, y para poder brindar la posibilidad de publicar entradas en NNTP.

Ambiente de desarrollo

Para la implementación del proyecto se utilizaron una serie de herramientas que se describen a continuación:

Python: es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Twisted: es un framework de red para programación dirigida por eventos escrito en Python y licenciado bajo la licencia MIT. Twisted proporciona soporte para varias arquitecturas (TCP, UDP, SSL/TLS, IP Multicast, Unix domain sockets), un gran número de protocolos (incluidos HTTP, XMPP, NNTP, IMAP, SSH, IRC, FTP), y mucho más.

PyCharm: entorno de desarrollo integrado (IDE) que se utiliza para la programación en Python. Proporciona análisis de código, un depurador gráfico, un medidor de unidad integrada, integración con sistemas de control de versiones (prensas de tornillo), y apoya el desarrollo web con Django. PyCharm es desarrollado por la empresa checa JetBrains.

UnrealIRCd: es un servidor de IRC de código abierto, sirviendo a miles de redes desde 1999. Se ejecuta en Linux, OS X y Windows y es actualmente el IRCd más implementada con una cuota de mercado superior al 50%. UnrealIRCd es un IRCd muy avanzado con un fuerte enfoque en la modularidad y un avanzado y altamente configurable archivo de configuración.

Raspberry Pi 2 Modelo B: es un ordenador de placa reducida (SBC) de bajo costo desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Para el proyecto se utilizó este dispositivo en el desarrollo del medio de transmisión (luz).

Raspbian: es una distribución del sistema operativo GNU/Linux y por lo tanto libre basado en Debian Wheezy (Debian 7.0) para la placa computadora Raspberry Pi. Su lanzamiento inicial fue en junio de 2012.

Estructuras de datos usadas y funciones

Principales estructuras

Listas: utilizadas para almacenar los routers bien conocidos utilizados por un host para establecer conexión o por otro router cuando se necesita transmitir algún paquete a un nodo que no se encuentra entre sus conexiones directas.

Diccionarios: utilizados principalmente para la transmisión de mensajes. Se envían objetos 'Diccionario' que consisten de diferentes llaves y valores. Estos permiten a los host y router categorizar los mensajes y distinguir entre la función que cada uno de estos tiene. Por ejemplo se tiene los tipos "m", "r", "q", "fw", que permiten respectivamente enviar un mensaje a otro usuario, registrar a un usuario dentro de la red, hacer una consulta a los demás routers para el balance de conexiones y hacer "forward" de un mensaje a un

usuario conectado directamente a un router.

Principales funciones

`ConnectionFinder::look_for_router()`: este método se encarga de buscar cuál es el router apropiado para cuando un host desea conectarse. El host indica que quiere unirse a la red, por lo que por medio de este método se usa la lista de routers bien conocidos para ver cuál se encuentra disponible. Una vez que se encuentra alguno, se le realiza la consulta de a cuál router debe conectarse.

`ChatClient::dataReceived(data)`: método del cliente de chat que implementa lo que se va a realizar cuando el reactor de Twisted atiende un evento de datos recibidos. En este caso lo que se necesita hacer es notificar al usuario que un mensaje ha llegado, por lo que se imprimen en su pantalla el remitente y contenido del mensaje.

`HostManager::exists(username)`: verifica la existencia de un usuario en un router, mediante la revisión del archivo de hosts.csv (cada router posee su propio archivo).

`HostManager::register(host)`: recibe la información del host que desea conectarse y la almacena en el archivo hosts.csv para finalizar con la conexión del host en el router.

`HostManager::delete(username)`: elimina a un usuario de la lista de hosts.csv cuando este se desconecta de la red.

`HostManager::get_users()`: devuelve una lista con la información de los usuarios pertenecientes a un router.

`RouterConnection::dataReceived(data)`: este es una de las funciones más importantes de RouterConnection ya que es aquí donde se reciben los datos desde los clientes y se decide que hacer con ellos.

`RouterConnection::parse_data(data)`: método que asigna una función como respuesta a cada uno de los tipos de solicitud, dependiendo de la función que se deba realizar para brindar una respuesta al cliente.

`RouterConnection::send_message(data)`: función que se encarga de enviar un mensaje que viene desde un Host y va hacia otro Host, por medio de un nombre de usuario. Se determina si el host esta dentro de las conexiones del router que esta procesando la solicitud, si es así se envía de forma directa el mensaje; de lo contrario el Router pregunta a los demás Router si alguien tiene al usuario en sus conexiones, si alguno lo tiene, se hace forward hacia ese Router, en otro caso no existe el usuario en la red y se notifica al Host emisor.

`RouterConnection::register_user(data)`: el Router se encarga de registrar un usuario en la red. Primero se encarga de verificar si el nombre de usuario esta tomado y indica al Host que debe eligir otro nombre. Si el nombre esta disponible entonces responde de forma positiva al Host.

`RouterConnection::is_local_user(data)`: verifica si el usuario dado existe en el archivo de conexiones del Router que esta procesando la solicitud. Esto se utiliza para saber si es una conexión directa o si pertenece a otro Router.

`IRCClient::send(msg)`: envía un mensaje que contiene el hashtag "public" al canal elegido en el servidor IRC para que este sea visible por cualquier persona.

`SocketClient::send(data)`: se encarga de enviar un mensaje de manera genérica y esperar por una respuesta del Router.

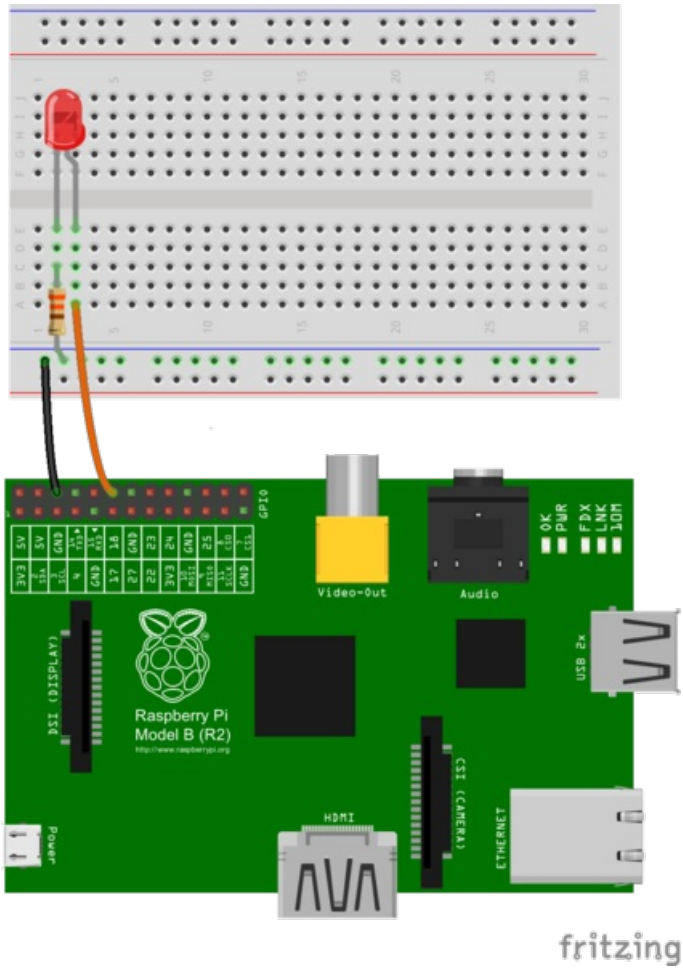
Instrucciones para ejecutar el programa

Para correr TEC-land se tienen 2 archivos principales que corresponden con los dispositivos Host y Router. Por otra parte se encuentran los archivos correspondientes a los bot: IRC, NNTP y medio de transmisión, que también deben ser ejecutados para que estas características estén disponibles. Una descripción de cada uno es brindada en la siguiente lista:

- host/main.py: se encarga de correr la aplicación host en un nodo. Este tiene como funciones principales la recepción y el envío de mensajes a través de la red.
- router/router.py: archivo principal de los nodos centrales, también llamados routers, que permite a estos ponerse disponibles y empezar a funcionar como punto de enlace para los siguientes host que se integren a la red. Este es el que se encarga de realizar todas las operaciones de ruteo en la red.
- irc/main.py: cliente IRC que se conecta a algún router que aún no contenga un bot, y se queda activo esperando por mensajes que deban ser publicados en el servidor IRC (UnrealIRC en este caso).
- nntp/main.py: cliente NNTP que de igual forma busca un router que no contenga bots para establecer su conexión y quedar a disposición de la red para cuando un mensaje deba publicarse en el servidor NNTP.
- transmission/main.py: bot que esta corriendo y recibe los mensajes que se dirigen hacia la comarca de Cartago, por lo que deben ser transmitidos por medio de luz utilizando la Raspberry Pi.

Para el medio de transmisión se deben realizar las conexiones en la Raspberry Pi que permitan el paso de corriente eléctrica a través de la placa reducida y hasta el LED de manera que se pueda encender y apagar cuando se necesite. El circuito por realizar consiste básicamente de una fuente de poder (Raspberry Pi), un LED y una resistencia que limita el flujo de corriente.

Se necesitará usar uno de los pines GND (ground) para que actue como el extremo negativo de una batería. El extremo positivo de la batería sera proveído por uno de los pines GPIO. En este caso se usará el pin 18, que emitirá 3.3 voltios para hacer que el LED se encienda. El diagrama del circuito se muestra a continuación.



Para información más detalladas se puede consultar: [turn LED with Raspberry Pi](https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins

Algunas aspectos que se deben tomar en cuenta antes de ejecutar los archivos mencionados son:

- Tanto los dispositivos Host como los Router deben pertenecer a la misma red y estar comunicados por medio de una conexión TCP/IP.
- Al ejecutar el Host ya debe estar registrado al menos un Router en la red. Esto para que el Host pueda encontrar un punto al cuál conectarse.
- En los Host debe existir el archivo de routers bien conocidos, para ver cual de ellos esta disponible y le puede responder la consulta que pregunta por la ip y puerto del Router menos cargado.

Cuando se tomaron en cuenta las consideraciones anteriores se procede a ejecutar los archivos principales según la función que se desee. Se presentan las lineas:

```
$ python2 host/main.py
$ python2 router/router.py
$ python2 irc/main.py
$ python2 nntp/main.py
$ python2 transmission/main.py
```

Finalmente se digita la información requerida para su inicio/funcionamiento.

Actividades realizadas por el estudiante

Jueves a viernes 01 de abril

Actividad	Tiempo
Lectura de la especificación.	20 min
Reunión con el profesor para aclarar puntos de la tarea.	1 hora
Investigacion de P2P, APIs para redes mesh en Python, servidores IRC (Unreal IRC) y clientes (XChat IRC), Firechat	3 horas
Logramos encender el LED de la Raspberry y convertir caracteres a su binario para transmitirlos por medio de luz.	1.5 horas
Instalación de Unrealircd (servidor IRC).	15 min

Lunes 11 de abril

Actividad	Tiempo
Creación del repositorio.	15 min
Diseñando la red y como conectar nodos.	30 min
Investigación de Twisted. Lectura de creación de clientes y servidores básicos.	30 min
Creación de la clase HostManager. Encargada de administrar los host conectados a cada Router.	1.5 horas
Estructura de la clase Router.	1.5 horas
Agregando estructuras para cliente y servidor. Clases Protocol y Factory para realizar el listenTCP y el connectTCP.	1.5 horas

Martes 12 de abril

Actividad	Tiempo
Separación de clase Router en dos clases Router y RouterConnection. RouterConnection registra los usuarios.	1 hora
Investigación sobre objetos Diferred de Twisted.	2 horas
Pruebas con objetos Diferred.	2 horas
Nueva estructura de los componentes del proyecto.	20 min
Creación de clase ConnectionFinder para revisar la lista de routers bien conocidos y establecer contacto con alguno con el fin de obtener la ip y el puerto del router con el cuál establecer la conexión.	1.5 horas

Jueves 14 de abril

Actividad	Tiempo
Consultar a los router vecinos para realizar el balanceamiento de carga (asignar host al menos cargado).	20 min
Trabajos en la comunicación entre los router.	1 hora
Creación de clase SocketClient para manejar las conexiones cliente/servidor y realizar envío de mensajes.	30 min
Creación de clase Host, para envío de mensajes.	30 min
Investigación extensa del servidor de Twisted para ver si se podía crear un evento para enviar mensajes.	2 horas
Envío de mensaje hasta router para que este se encargue de hacerle "forward".	20 min
Creación de "thread" para que muestre el chat preguntando a quien va dirigido el mensaje y su contenido.	20 min
Clases ChatClient, HostFactory y su estructura básica.	1 hora

Viernes 15 de abril

Actividad	Tiempo
Modificaciones en la clase HostManager.	30 min
Verificación de usuarios en toda la red.	1.5 horas
Redireccionamiento de mensajes cuando no se encuentra el usuario en la red local.	30 min
Creación de clase ChatClient.	20 min
Agregación de un atributo status en los mensajes para notificar al usuario (host) el estado de la petición.	10 min
Creación del archivo MarkDown para la documentación, creación de las diferentes secciones y trabajo en la Introducción, Herramientas Usadas y Bitácora.	1.5 horas

Sábado 16 de abril

Actividad	Tiempo
Modificación de la clase RouterConnection para realizar broadcast de un mensaje a la red (se modificaron algunos parámetros).	30 min
Documentación parcial de código.	1 hora

Jueves 19 abril

Actividad	Tiempo
Actualización de la documentación	30 min

Domingo 22 de abril

Actividad	Tiempo
Trabajo en RFC basado en el de HTTP/1.1	1 hora
Agregación de términos y tipos de mensajes.	1 hora

Miercoles 25 de abril

Actividad	Tiempo
Investigación de los servidores IRC para implementralo junto con el cliente	1 hora
Investigación de los clientes IRC de Twisted	30 min
Codificación del cliente de iRC con biblioteca de Twisted	1 hora
Cambiando el código en las diferentes clases y archivos para revisar si el status de las respuestas es negativo y mostrar el mensaje de error.	2horas
Creando nueva clase para separar las conexiones del router. Clase Linker.	2 horas
Agregando contenido al RFC en la sección de terminología	2 horas
Cambio en el identificador de los mensajes de IRC	5 min

Jueves 26 de abril

Actividad	Tiempo
Agregando comentarios a las clases	30 min
Agregando terminología y en la sección de operación general	1 hora

Sábado 7 de mayo

Actividad	Tiempo
Terminando documentación del proyecto.	2 horas
Completando últimas secciones del RFC.	2 horas
Comentarios en clases varias del código.	1 hora

Comentarios finales

El estado final del proyecto es satisfactorio ya que la mayor parte de las funciones se pudieron implementar de manera correcta. El envío de mensajes a través de la red se puede realizar sin ningún problema siempre y cuando el usuario destino exista dentro de TEC-land. El broadcast es funcional y permite que un mensaje sea enviado a todos los nodos conectados a la red.

Los bot para IRC y NNTP funcionan de manera correcta y ejecutan su labor cada vez que reciben un mensaje. El cliente IRC se conecta correctamente con el servidor UnrealIRCd y es capaz de publicar los mensajes en un canal para que puedan ser accedidos públicamente. Sin embargo se tuvieron problemas con la parte del servidor para NNTP por lo que esta conexión entre el cliente y el servidor no se pudo concretar en el proyecto.

Por el lado del medio de transmisión se experimentaron una serie de factores limitantes debido a la naturaleza de esta parte del proyecto, que es en cierto modo ajena a nuestra área de expertiz. Se tuvieron muchas pruebas para poder obtener resultados, no obstante estos no son buenos en todos los casos ya que la forma en que la fotoresistencia interpreta la luz y la oscuridad tiende a ser un poco difusa y no se puede determinar a ciencia cierta cuando es una y cuando es la otra. Por esto algunas de las pruebas fallaron porque en algunos casos los rangos usados para distinguir un 0 de un 1 variaban haciendo que se mal interpretara y con ello afectando totalmente al mensaje final.

Conclusiones

Trás la implentación de la red TEC-land se logró comprender mejor como funciona una red, cuáles son las tareas que le corresponden a cada de las capas (física, red, aplicación...) y como estas interactuan entre sí. Las redes son contrucciones complejas que requieren de un diseño correcto acorde a las necesidades que se tengan.

Se tuvo una experiencia de toma de decisiones para la creación del protocolo y el diseño que se utilizarían en la red. En este paso se realizó la escogencia entre los distintos "dispositivos" que se iban a utilizar y como iba a ser la jerarquía entre ellos, por lo que representó un ejercicio ingenieril donde se tuvo que justificar el porqué de las decisiones.

También se pudo apreciar como cambios pequeños complican o varían considerablemente el envío de paquetes a través de una red. Las conexiones que se escojan y las funciones que se deleguen a cada uno de los dispositivos representan factores importantes que modifican como se va a comportar el flujo de datos desde un punto a otro.

Además, hay que destacar que las jerarquías son puntos determinantes en la red, ya que establecen controles o puntos de comunicación entre dos o más dispositivos que tienen el mismo rol. Esto permite a

los paquetes subir por los niveles de la red y delegar la escogencia del camino a un dispositivo que pertenece a el nivel actual, tomando en cuenta toda la información del ambiente que se posea.

Se pudo fortalezar también el conocimiento sobre el funcionamiento entre las relaciones cliente/servidor, al cada nodo de la red tener que escuchar y al mismo tiempo conectarse a otro dispositivo para enviar un mensaje. Esto representa un modelo ampliamente utilizado para muchas de las tareas que se realizan en computación por lo que tener claro su implementación y algunas aplicaciones representa un insumo importante.

Otro de los beneficios obtenidos trás el proyecto fue el conocimiento de nuevos temas en los que no se tenía experiencia como lo son el protocolo NNTP y el acercamiento a los arduinos para la creación del medio de transmisión. Este es un campo que normalmente no se explota mucho en computación pero resulta interesante realizar pruebas en la capa física y obtener resultados más tangibles en comparación a como se acostumbra en la mayoría de cursos de la carrera.

Bibliografía

1. Twisted Network Programming Essentials 2nd Edition
2. The PiHut. Turning on an LED with your Raspberry Pi's GPIO Pins. June 11, 2015. Disponible en https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins
3. UnrealIRCd. UnrealIRCd 4 documentation. Disponible en https://www.unrealircd.org/docs/UnrealIRCd_4_documentation.