Desarrollo de un demonio RIPv2

Objetivo

Desarrollar, a partir de la librería `librawnet`, una torre de protocolos IP que implemente los siguientes protocolos: Ethernet, ARP [1], IPv4 [2] y UDP [3]. Sobre dicha torre de protocolo, y utilizando el servicio de envío y recepción de datagramas ofrecido por el protocolo de transporte UDP desarrollado, se implementará un demonio RIPv2 [4] que debe ser capaz de procesar tanto mensajes *Request* de un cliente, como mensajes *Response* generados por otros routers RIPv2.

Especificaciones

Primero se pide desarrollar, utilizando el lenguaje de programación C, una pila de protocolos modular, estructurada en capas, donde cada protocolo sea independiente y oculte su funcionalidad a los demás, y tan sólo utilice los servicios proporcionados por la capa inmediatamente inferior. Por lo tanto los protocolos desarrollados no pueden realizar ninguna suposición acerca del protocolo superior que va a utilizarlos (p.e. fijar en la capa IP el campo protocolo de la cabecera IPv4 a 0x11), aunque sí es posible hacer asunciones sobre los protocolos inferiores (p.e. calcular el tamaño máximo de los paquetes en función de la MTU de Ethernet).

Cada protocolo debe ser implementado en un módulo independiente, que consistirá, al menos, en un archivo `.h` con la definición de las funciones que puede utilizar el protocolo superior, así como un segundo archivo `.c` que contenga la implementación de dichas funciones, así como cualquier otra estructura o función necesaria para el correcto funcionamiento del mismo. Para la capa ARP se recomienda utilizar el código desarrollado en la práctica ARP realizada previamente.

Para comprobar el correcto funcionamiento de la torre de protocolos, se recomienda ir probando su funcionalidad capa a capa, esto es, no empezar a desarrollar un protocolo de una capa superior hasta que no se ha probado completamente el correcto funcionamiento de la capa inferior. De este modo es mucho más sencillo aislar los fallos y saber en qué capa se encuentra el problema, que si se prueba toda la torre de protocolos simultáneamente al final del ciclo de desarrollo. Para ello será necesario desarrollar un cliente y/o servidor de Eco para cada uno de los protocolos implementados. El programa cliente (p.e. `ipv4_client.c`) debe enviar un mensaje con datos al servidor, y esperar respuesta, mientras que el programa servidor (p.e. `ipv4_server.c`) debe estar preparado para recibir datos de los clientes y responderles de vuelta con los mismos datos.

Una vez desarrollada la torre de protocolos indicada, se procederá a implementar un programa servidor RIPv2 (`RIP_server.c`) que debe ser capaz de procesar tanto mensajes *Request* (recibidos de un cliente RIP) como mensajes *Response* (generados por otro demonio RIPv2 estándar).

Los mensajes RIP *Request* serán generados por un programa cliente (`RIP_client.c`), que también debe ser desarrollado por el grupo de prácticas, y enviados en unicast a la dirección IPv4 especificada por línea de parámetros. Dichos mensajes RIP *Request*

pueden solicitar toda la tabla de rutas del demonio RIPv2 destino, o tan sólo parte de ella, a discreción de los propios alumnos. El cliente debe esperar asimismo respuesta del servidor RIP, que puede ser tanto un servidor RIPv2 estándar (p.e. el que implementan los routers Linksys empleados en la asignatura) como el demonio RIP desarrollado, e imprimir por pantalla el contenido más significativo de la respuesta recibida.

En cuanto el servidor RIP, este debe ser capaz de recibir mensajes *Request* enviados por el cliente y responder con un mensaje *Response* adecuado al tipo de consulta realizado. Además debe ser capaz de recibir y procesar mensajes *Response* generados por otros demonios RIPv2 estándar, y a partir de los mismos generar y mantener actualizada una tabla de rutas de acuerdo a las reglas del susodicho protocolo de encaminamiento de vector distancia RIPv2. Para ello es necesario almacenar en dicha tabla de rutas la ruta óptima en cada momento (indicando la métrica de dicha ruta) para llegar a cualquiera los destinos aprendidos, así como borrar las entradas que no han sido actualizadas recientemente o tienen una métrica infinita. Por cada mensaje *Response* procesado que genere algún cambio en la tabla de rutas, debe imprimirse, al menos, el estado final de la misma una vez aplicados todos los cambios.

Nótese que para cumplir con la funcionalidad básica solicitada no es necesario que el demonio envíe mensajes *Response* periódicamente, sino tan sólo que los reciba. El anuncio de rutas por parte del servidor RIP desarrollado se considera una mejora opcional.

Mejoras del proyecto básico

Esta es una lista, no exhaustiva, de posibles mejoras que pueden realizarse sobre proyecto básico arriba indicado para mejorar la nota final del mismo:

- Añadir bytes de relleno en tramas Ethernet de tamaño mínimo.
- Caché ARP:
    - Múltiples entradas.
    - Temporizador para sobreescribir las entradas menos utilizadas.
    - Consultas ARP en *unicast.*
- Servidor ARP:
    - Interacción con la cache ARP
- Implementar el *Checksum* de UDP
- Comprobación de los *checksums* de IP y/o UDP (incluyendo la generación aleatoria de errores para probar dicha funcionalidad).
- Implementación de los dos tipos de operación RIP *Request* permitiendo solicitar cualquier número de rutas individuales.
- Envío de mensajes RIP *Request broadcast*/*broadcast* de subred/*multicast* para solicitar la tabla de rutas al iniciar el servidor RIP.
    - Mapeo de direcciones IP *broadcast* en direcciones MAC *broadcast*.
    - Mapeo de direcciones IP *multicast* en direcciones MAC *multicast*.
- Sincronizar la tabla de rutas de RIP con la tabla de reenvío de IP.
    - Obtener rutas directamente conectadas de la tabla de IP (configurable).
    - Obtener rutas estáticas de la tabla de IP (configurable).
    - Añadir el concepto de distancia administrativa en IP.
- Envío periódico de mensajes RIP *Reply broadcast*/*broadcast* de subred/*multicast* con la tabla de rutas actualizada.
    - Implementar el temporizador de *garbage-collection*.
    - Implementar la técnica de *Split Horizon*.

    - Implementar la técnica de *Poison Reverse*.
    - Implementar la técnica de *Triggered Updates*.
- Permitir tablas de rutas RIP con más de 25 entradas.

Referencias

[1] “RFC826: *An Ethernet Address Resolution Protocol*”. Noviembre 1982.
[2] “RFC791: *Internet Protocol*”. Septiembre 1981.
[3] “RFC768: *User Datagram Protocol (UDP)*”. Agosto 1980.
[4] “RFC2453: *RIP Version 2*”. Noviembre 1998.