Universidad de Chile
Facultad de Cs. Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC4303 Redes

Tarea 3 - 2 protocolos de ruteo

Profesor: Catalina Álvarez

Auxiliar: Eduardo Riveros

Objetivos

La finalidad de esta tarea es que comparen los funcionamientos de los dos protocolos de ruteo interno que se vieron en el curso: RIP y OSPF. Dado que es bastante trabajo que implementen ambos algoritmos desde cero, se les da bastante trabajo avanzado. Junto a esta tarea viene el código fuente de una implementación de RIP sobre la cual pueden trabajar; además, muchas de las herramientas (routers, router_port, etc) son válidas también para OSPF.

Implementación entregada

La tarea viene junto con el código fuente (en Python) de una implementación de RIP funcional (pero a la que quizás deban agregar cosas para realizar sus comparaciones).

Sobre la implementación:

La implementación consta de cinco archivos base:

1. **send_packet**: Contiene una sola función, del mismo nombre, que sirve para enviar un paquete a un cierto puerto. Se les deja para poder realizar pruebas más fácilmente. Es importante notar que send_packet no realiza ningún formateo de los paquetes (eso queda a responsabilidad suya).

2. **topology**: Clase que modela una topología (conjunto de router), tiene dos métodos principales, start (que dado un archivo de topología crea los routers necesarios, levantando threads como sea necesario) y stop (para terminar los threads correctamente). Además, contiene métodos para cambiar los costos de un enlace.

3. **router**: Clase que modela un router. Un router tiene puertos que utiliza para comunicarse con otros routers (estos puertos se modelan usando la clase RouterPort). Las decisiones internas de qué hacer con los paquetes las toma el router con sus funciones:

- **_success**: el paquete es para el router que lo recibe, se imprime la data que lleva en consola.
- **_new_packet_received**: cuando se recibe un nuevo paquete de un puerto, se revisa el destino del paquete: si es para el router, se llama a **_success**, sino, se elige a quien forwardear el paquete.

4. **router_port**: Clase que modela el puerto de un router; se modela como un thread que es levantado por el Router. Un router por un determinado puerto puede recibir y mandar tráfico simultáneamente, lo que no es posible (o al menos recomendado) con sockets.
   - Para modelar este comportamiento, un puerto usa dos puertos del sistema operativo, uno para enviar los datos (output) y otro para recibir los datos (input).
   - El RouterPort guarda datos que debe mandar en una cola, y revisa constantemente si esta cola tiene algo para mandar. Cuando tiene un paquete que mandar, crea un socket con el puerto de output y manda el paquete, cerrando el socket cuando termina.
   - Para recibir los datos constantemente, cada RouterPort levanta un thread hace binding del socket de entrada y está escuchando por nueva data. Cuando recibe un paquete, lo envía al router usando el método de callback definido (por defecto en la implementación, **_new_packet_received**), para que decida que hacer con él.
   - Cuando se tiene un paquete que enviar, el Router lo pone en una cola que controla el RouterPort.

Los mensajes que se envían los routers son diccionarios stringificados usando el clásico método json.dumps de Python. Los mensajes tienen dos campos: "data" (la información que se desea mandar) y "destination" (a quien va dirigido el mensaje).

Los archivos de topología siguen una estructura similar a la del archivo de ejemplo entregado:

1. Los routers se entregan en una lista, cuya llave es "routers".
2. Cada router tiene dos parámetros: nombre (identificador del router) y ports (puertos o "salidas" del router, lo que en clases conocemos como interface), en forma de una lista.
3. Los puertos de los routers se modelan como dos sockets: uno de entrada (input) y uno de salida (output).

Es posible configurar varios parámetros del sistema usando el archivo `settings/settings.py`. Los parámetros relevantes son:

1. LOG: controla el logging del sistema.
2. TOPOLOGY_CREATION_TIMEOUT: Tiempo que se espera entre que se levanta el sistema y se buscan los vecinos de cada router.
3. UPDATE_TIME: Cada cuanto tiempo se envía el vector de distancia a los vecino.
4. PROGRAM_UPDATE: Tiempo que se espera entre modificar un enlace y recalcular los vectores de distancia.

Implementación pedida

La implementación pedida es simple. Deben implementar OSPF (pueden o no usar como base la implementación de RIP entregada) y comparar ambos protocolos en los siguientes parámetros:

1. Número de paquetes (total en el sistema) necesarios para lograr la convergencia.
2. Tiempo entre el inicio del sistema y que las tablas sean estables.
3. Tiempo de cálculo (a nivel de un router) de las tablas de ruta.
4. Tiempo de estabilización de las tablas de rutas cuando se sube y baja algún costo de un enlace.
5. Tiempo de estabilización de las tablas de rutas cuando se elimina un enlace (no probar con un enlace que dejaría dos grupos separados de routers).

Informe

Detalle importante: **deben entregar un README explicando como correr su tarea**, y aclarando sus supuestos. Se les repite **su tarea debe poder correrse sin complicaciones por parte del corrector**. Por favor consideren que mientras mejor documentada su tarea, más fácil su evaluación y menos problemas tendrán.

A diferencia de las tareas anteriores, acá deberán escribir un poco más en el informe. Deben correr pruebas para obtener datos para las comparaciones pedidas en la sección anterior, y luego tabularlas. En total, sus pruebas deben incluir un mínimo de 5 topologías

distintas; para estas pruebas pueden usar las tres topologías que se les da de ejemplo y al menos dos más (creadas por ustedes), que deben tener un mínimo de 6 nodos y 10 enlaces.

Restricciones

Como la base entregada está en Python para esta tarea están restringidos completamente a este lenguaje.

Cualquier duda o pregunta o reporte de bugs, dirigirse al foro de U-cursos.

Evaluación

Esta tarea será evaluada tomando en cuenta los siguientes puntos:

1. Funcionalidad de implementación de OSPF:
   - *a*) Lee los archivos de topología igual que RIP.
   - *b*) Hace flooding de la información de conectividad de la red.
   - *c*) Es capaz de calcular las tablas de ruta usando el algoritmo de Dijkstra.
   - *d*) Es capaz de recalcular las tablas de ruta correctamente luego de cambiar el costo de un enlace (subir o bajar), o de eliminar un enlace.
2. Entrega de resultados de comparaciones entre RIP y OSPF, usando las mismas topologías de prueba.