Laboratorio 4

Programación de Red virtual y simulación de tráfico
Laboratorio de Redes 2012, UTFSM

Valparaíso, 25 de junio de 2012

Índice

1. Contextualización

En esta experiencia se deberá programar un entorno de de red de routers virtuales sobre la cual se deben encontrar y registrar, a través de la simulación de tráfico, las rutas más cortas entre cada uno de ellos . Una vez terminada la programación, el alumno se encontrará apto para responder una serie de preguntas relacionadas con la programación y materia de cátedra para posteriormente escribir un informe técnico.

Es de suma importancia tomar en cuenta las consideraciones que se establecen al final de este documento además de los siguientes conocimientos.

- Concepto de una *tabla de ruteo*.
- Lenguaje de programación *Python* o *Java*.
- Estructuras de datos.

1.1. Objetivo General

Utilizando *"tablas de ruteo"*, determinar los *mínimos-costos* entre las rutas de la red.

1.2. Objetivos Específicos

- Programar un *entorno de red routers simulado*.
- Aplicar estructura de datos en la programación con *Python* o *Java*.
- Determinar los *mínimos-costos* entre las rutas a cada router utilizando el concepto de *tablas de ruteo*.
- Comprender la utilidad del cálculo de mínimos-costos entre las rutas de la red.
- Realizar un informe técnico con análisis y resultados referentes a la programación, e incorporar las respuestas a las preguntas planteadas al final de este documento.
- Cumplir con las indicaciones establecidas en la sección *Consideraciones*.

2. Descripción del Laboratorio

En esta Experiencia, usted escribirá un conjunto de procedimientos "distribuidos"que implementan una rutina *vector-distancia distribuida y asincrona* para la red de planteada en la figura 1.

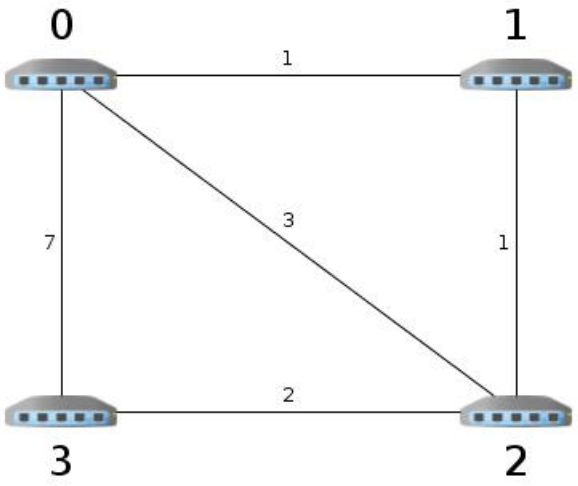

Figura 1: *Topología de red y costos de enlaces.*

Como indica la figura anterior, los routers son identificados como:

router 0
router 1
router 2
router 3

Usted escribirá las siguientes rutinas que se "ejecutarán" asincronamente dentro del entorno emulado que se proveerá durante el desarrollo de la misma tarea. Para el *router 0*, escribirá las siguientes rutinas:

- *rtinit*0(): Esta rutina será llamada una vez al comienzo de la emulación. *rtinit*0() no tiene argumentos. Esta debe inicializar su *tabla de distancias* en el *router 0* para "reflejar" los costos 1, 3 y 7 a los routers 1, 2 y 3 respectivamente. En la figura 1, todos los enlaces son bidireccionales y los costos en ambas direcciones son idénticos. Después de inicializar la *tabla de distancias* y cualquier otra estructura de datos que necesiten las rutinas de su *router 0*, este debererá enviar directamente a sus vecinos conectados (en este caso, 1, 2 y 3) el costo de sus rutas de *mínimos-costos* a todos los otros routers de la red. Esta informacion de *minimos-costos* es enviada a los routers vecinos en un *paquete de actualizacion de ruteo* por medio de una *rutina de llamada* conocida como *tolayer*2.

- *rtupdate0(struct rtpkt *rcvdpk)*: Esta rutina será llamada cuando el *router 0* reciba un *paquete de ruteo* que fue le fue enviado por un vecino directamente conectado. El parámetro $*rcvpkt$ es

un puntero al paquete que fue recibido.

*rtupdate*0 es el corazón del algoritmo *vector-distancia*. Los valores son recibidos en un paquete de actualización de ruteo desde algún router *i* que contiene sus actuales costos de las *rutas mas cortas* a todos los otros routers de la red. *rtupdate0()* usa esos valores recibidos para actualizar su propia *tabla de distancias* (como especifica el algoritmo *vector-distancia*). Si su propio costo hacia otro router cambia como resultado de una actualización, el *router 0* informa directamente a sus vecinos de este cambio de *minimos costos* enviandoles un *paquete de ruteo*. Recordar que en el algoritmo *vector-distancia*, solo los routers conectados directamente intercambiaran *paquetes de ruteo*. Por lo tanto, *routers 1 y 2* se comunicaran con cada uno de los otros, pero los *routers 1 y 3* no se comunicaran con esos otros.

- *tolayer2(struct rtpkt pkt2send)*: Donde *rtpkt* es una siguiente estructura como la siguiente:

```
extern struct rtpkt {
  int sourceid;     /* id of node sending this pkt, 0, 1, 2,
                       or 3 */
  int destid;       /* id of router to which pkt being sent
                       (must be an immediate neighbor) */
  int mincost[4];   /* min cost to node 0 ... 3 */
  };

Note that tolayer2() is passed a structure, not a pointer to a
structure.
```

Como ya se mencionó antes *tolayer*2, envia la información de *mínimos-costos* a los routers vecinos en un *paquete de actualizacion de ruteo*.

Las tablas de distancias de cada router son las principales estructuras de datos usadas por su *algoritmo vector-distancia*. Encontrará conveniente declararla como un arreglo de 4x4 que almacenará enteros, donde la entrada [i,j] de la tabla del *router 0* son los actuales costos del router 0 al *router i* calculados via su vecino directo *router j*.
Recuerde que solo los routers conectados directamente pueden comunicarse.

La figura 2 muestra conceptualmente la relación de los procedimientos dentro del *router 0*.

Rutinas similares deben ser definidas para los *routers 1, 2 y 3*. Por lo tanto, deberá escribir 8 procedimientos en total: *rtinit0(), rtinit1(), rtinit2(), rtinit3(), rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3()*. Juntas esas rutinas implementaran *computación distribuida, asincrónica* sobre las *tablas de distancias* para la topología y los costos mostrados en la figura 1.

Deberá escribir sus procedimientos para los *routers 0* en 4 archivos llamados *router0,...,router3*. No se permite el uso de variables globales que sean visibles fuera de un archivo *java*. (por ejemplo, cualquier variable global que se defina en *router*0.*java* debe solo ser accedida por *router*0.*java*).

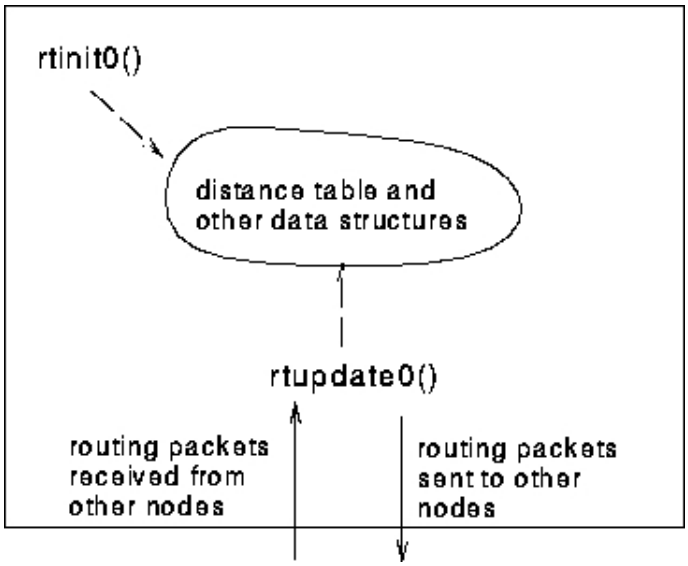

Figura 2: *Relación entre procedimientos dentro del* router 0.

3. Preguntas

1. Explique qué utilidad podría ofrecer el conocer las rutas menos costosas en una red, y dé un ejemplo pensando en algún servicio de red especifico (p.ej. Servidor de archivos).
2. ¿Qué parte del desarrollo (código) ud. considera que es asincróno y que parte sincrónico?
3. ¿Qué elemento(s) del código le(s) permite(n) la comunicación entre los nodos? ¿Qué mejora podría agregarle?
4. Determine, según usted, cuál sería el comportamiento o qué sucedería si se agregase o eliminase un nodo de la red.
5. Comente cuales fueron las principales complicaciones durante el desarrollo de la tarea.

4. Código

- El programa deberá sólo mostrar por pantalla, la matriz final con los caminos más cortos a cada nodo.
- La corrección se realizará en los computadores del LabIT, se recomienda probar su código en ellos antes de la entrega.
- El archivo a subir debe ser un tarball que contenga un directorio llamado **GrupoXX**. Además el nombre de este tarball debe ser **GrupoXX.tar.gz**, donde XX es el número de su grupo asignado.
- El directorio debe contener un `Makefile` que debe generar el ejecutable **mincost** y un `README` donde se explique brevemente la solución implementada.

5. Consideraciones

1. Si cree necesario implementar además otras rutinas es libre de hacerlo, siempre y cuando argumente el por qué de la decisión.
2. Recuerde respetar los identificadores y la presentación de sus códigos.
3. Comente su código pero no abuse de ello.

6. Reglas

- El ayudante encargado es: Jean Pierre Guiñez *jpguinez@alumnos.inf.utfsm.cl*
- La tarea debe ser realizada por los grupos ya establecidos por el laboratorio.
- La programación de la tarea debe ser desarrollada en *Python* o *Java*.
- Se debe escribir un informe técnico elocuente y ordenado.
- Incorpore en la sección *Experiencia* el template del informe, sus resultados y respectivos análisis, además de las respuestas a las preguntas planteadas en el presente documento.
- Si utiliza Referencias en el informe, éstas deben ser indicadas por página y al final del documento.
- La entrega del Trac será permitida hasta el Jueves 12 de Julio a las 23:59 hrs.
- La entrega del informe será permitida hasta el día 13 de Julio antes de las 19:00 hrs. en la casilla del LabIT (Piso 3, edificio F3) impreso en hojas de tamaño carta y escrito en LaTeX.
- Si Trac y/o informe no se entregan en el día y hora determinada se evaluará con nota final 0 en el informe.