Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3067 Redes

Laboratorio 3

Algoritmos de Enrutamiento

1 Antecedentes

Conociendo a dónde enviar los mensajes para cualquier router se vuelve trivial el envío de mensajes. Únicamente es necesario conocer el destino final y se reenvía al vecino que puede proveer la mejor ruta al destino. Toda esa información es almacenada en las tablas de enrutamiento.

No obstante, con el dinamismo con el que se espera que pueda funcionar el Internet es necesario que dichas tablas puedan actualizarse y acomodarse a cambios en la infraestructura. Los algoritmos con los que se actualizan estas tablas son conocidos como algoritmos de enrutamiento.

2 Objetivos

- Conocer los algoritmos de enrutamiento utilizados en las implementaciones actuales de Internet.
- Comprender cómo funcionan las tablas de enrutamiento.
- Implementar los algoritmos de enrutamiento y probarlos en una red simulada sobre un protocolo de capa superior (XMPP o similar).
- Analizar el funcionamiento de los algoritmos de enrutamiento.

3 Desarrollo

Los algoritmos de enrutamiento funcionan sobre nodos interconectados entre sí, donde cada nodo conoce únicamente cuáles son los vecinos que tiene. Dicha información inicial será proporcionada para cada nodo.

A partir de ello, se levantarán varias instancias (nodos) de las implementaciones de los algoritmos que se describen más adelante, para proceder a enviar mensajes y simular una red.

La primera fase se enfoca en los algoritmos, por lo que el medio será nuestra red local (nuestra computadora) y probaremos los algoritmos usando Sockets. La segunda fase busca escalar sobre la primera y estaremos utilizando un servidor XMPP o similar como el medio para nuestra red (el cual será proveído a ustedes), por lo que cada nodo corresponderá a un usuario en tal servidor.

Para tales efectos se conformarán grupos de 3 o 4 integrantes donde se implementarán los algoritmos según descrito más adelante.

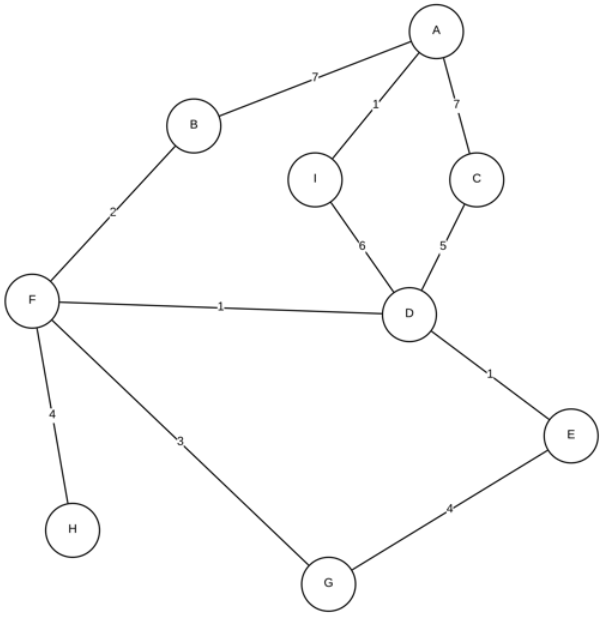

Imagen 1: Mapa de conexiones entre nodos.

En esta propuesta cada uno de los nodos corresponde a un cliente el cuál puede enviar o recibir mensajes para conformar su tabla de enrutamiento y posteriormente enviar un paquete de un nodo origen a un nodo destino utilizando dicha tabla. Cada nodo corresponde a un proceso independiente que se comunica vía Sockets (parte 1) o via el envío de mensajes hacia el servidor (parte 2).

El laboratorio consiste de dos partes principales: (1) la implementación de los algoritmos y (2) la interconexión y prueba de esos algoritmos.

3.1 Implementación de algoritmos

Por cada uno de los integrantes del grupo se debe implementar un algoritmo (el lenguaje de programación es libre, y puede ser el mismo para los *n* algoritmos). Los algoritmos para implementarse son (en negrita los obligatorios):

1. **Dijkstra**
2. **Flooding**
3. **Link state routing**
4. Distance vector routing

Las implementaciones deben de estar debidamente identificadas y publicadas en un repositorio privado (el día de la entrega lo hacen público), junto con sus requerimientos para su uso e instalación. Las implementaciones idealmente deben de poder ejecutarse en distintas plataformas de forma sencilla (makefiles, python venv, npm run, .jar).

Dependiendo del algoritmo, tendremos distintos inputs requeridos por cada uno para funcionar adecuadamente. Lo que requiere cada algoritmo incluye:

- Dijkstra: Topología (nodos, aristas)
- Flooding: conocimiento de sus vecinos solamente.
- Link State Routing: Las Tablas de los demás Nodos (de ella se deriva la Topología)
- Distance Vector: Las Tablas de los Vecinos.

Nótese que Dijkstra y Flooding se usan en LSR, por lo que deben manejar alta modularidad en sus Clases y archivos. Además, sus programas deben ser capaces de correr Flooding y Dijkstra como el algoritmo de la red, independientemente de su uso en LSR (o sea, levantar los nodos en modo "flooding", y enviarnos mensajes así, o levantarlo en modo Dijkstra y probarlo así aunque sepamos que es estático).

3.2 Conexión y Pruebas de los algoritmos

La segunda parte del laboratorio consiste en conectar y probar nuestros algoritmos implementados. Usaremos como "red" nuestro servidor XMPP, por lo que cada nodo corresponde a un usuario/recurso en la red.

Naturalmente, para poder probar y desarrollar sus algoritmos deberan conectarlos de alguna forma. Durante tal parte podrían utilizar sockets TCP temporalmente para agilizar el desarrollo de sus algoritmos (para poder probarlo todo localmente en sus computadoras, de forma offline, de ser necesario).

Sin embargo, la conexión oficial de la segunda parte y la que se probara y entregarán DEBE hacerse via XMPP y el servidor que levantaremos la siguiente semana; el uso de sockets es temporal y para agilizar el desarrollo local, y poder enfocarse más en los algoritmos en la "primera parte".

Como toda Red, debemos definir protocolos y formatos estándares para comunicarnos. La base es la siguiente, pudiendo agregar elementos si así lo consideran (**ojo, deben ponerse de acuerdo entre todos los grupos si modifican el protocolo. Esto significa que debería haber interoperabilidad entre codigos de distintos grupos en un mismo algoritmo**). La estructura será tipo JSON y será de la siguiente forma:

```
{
        "proto" : "dijkstra|flooding|lsr|dvr|...",
        "type" : "message|echo|info|hello|...",
        "from" : "foo@bar.com/123",
        "to" : "yolo@bar.com/777",
        "ttl" : 5,
        "headers" : [{"opcional" : "foo"}, {"alguna_optimizacion_suya" : "bar"}, ...],
        "payload" : "el contenido del paquete dependiendo de su tipo. Si es un mensaje de
                     usuario seria algo como este texto y se debe forward a destino o print
                     si somos nosotros destino. Si es un mensaje con info de tablas o
                     enlace, aca iría ese contenido que cada nodo puede extraer del
                     payload y utilizar para sus cálculos y tablas."
}
```

3.3 Otros detalles para las pruebas y conexiones

El día de la entrega probaremos los algoritmos en clase. Para ello, se estará asignando una dirección/nombre a cada uno de los alumnos (nodos), quienes usarán su usuario “oficial” del servidor como ID y credenciales (mas detalle cuando entremos a esa fase la siguiente semana). Se estará brindando un archivo con la distribución de nombres (ver Anexo para tal formato).

Los algoritmos a probar son Flooding, Distance Vector y Link State Routing (Dijkstra y Flooding son utilizados en LSR, Dijkstra no se probará directamente).

Adicional, se establecerán mapas de conexiones entre nodos similar al de la Imagen 1. Se estará brindando un archivo con tal topología mencionada al momento de las pruebas (ver Anexo para el formato), la cual deben utilizar para configurar sus nodos y solamente para eso. Cualquiera de los nodos debe de tener la capacidad de enviar y/o recibir un mensaje.

Al iniciar un nodo, este obtendrá la configuración y procederá a descubrir a sus vecinos. El nodo tendrá dos procesos/hilos en simultáneo: el forwarding y el routing. Todo debe correr en paralelo/asíncrono mediante el uso de hilos, procesos etc. Cada servicio se encarga de cosas específicas, como por ejemplo:

- Forwarding
    - Manejo de paquetes entrantes
        - Paquetes de Datos: forward o print si es para nosotros
        - Paquete de Info: dependiendo del algoritmo, como el Vector de Distancias o el LSP. Recibirlos y pasarlos al proceso de Ruteo.
        - Paquete de Hello/Ping: Descubrimiento de nodos y medición de distancia hacia ellos.
    - Manejo de paquetes salientes
        - Forward messages
        - Forward Flooding
        - Forward DV/LSP/INFO
        - Send Hello/Ping
        - Confirmaciones de recepción, etc.
- Routing
    - Inicializar la Tabla de Ruteo
    - Armar paquetes de Info
    - Consultar paquetes de Info y nuevos nodos entrantes
    - Utilizar paquetes de info para resolver y actualizar las tablas según cada algoritmo hace.

Siguiendo el formato establecido, deberán enviarse y definir distintos tipos de mensajes para el funcionamiento de los algoritmos. Se sugiere un paquete tipo HELLO/PING, para medir delays entre nodos e inicializar nodos. Se sugiere un paquete *DATA/MESSAGE*, el cual contenga data de usuario (mensajes) en su payload. Se sugiere un paquete *TABLE/INFO*, el cual contenga información de tablas, ruteo, vecinos, etc.. Pueden agregar otros tipos si así desean y les sirve.

El objetivo es lograr que los algoritmos se estabilicen y los mensajes pasen por los nodos que corresponden a la ruta óptima, así como el poder responder o adaptarse a nuevos nodos, nodos caídos, etc..

4 Rúbrica de evaluación

Elemento		Ponderación	
Código		**75%**	
	Documentación, orden, comentarios, limpieza, legibilidad/funcionalidad balanceada, etc..		5%
	Implementación de los Algoritmos, de forma eficiente y optimizada.		70%
Reporte Escrito		**25%**	
	Encabezado, Ortografía, Formato Adecuado, Descripción de la Práctica		2.5%
	Descripción de los Algoritmos Utilizados y su Implementación		10%
	Resultados		5%
	Discusión		5%
	Conclusiones + Comentarios + Referencias		2.5%

** Una inasistencia injustificada anula la nota del laboratorio.

Entregar en Canvas:

1. Archivo .pdf con su reporte en grupo
2. Codigo utilizado para el Laboratorio, en un rar si es necesario
3. Link a su repositorio, el cual es privado hasta antes de la entrega

5 Anexo

El formato de los archivos de configuración será acorde al siguiente (se pondrán ejemplos en Canvas):

- Topología de la Red:
  - archivo: topo-*.txt
  - contenido: JSON
  - formato ejemplo:

    {“type”:”topo”,

    “config”: **{**“A”: [‘B’ , ’C’],

    “B”: [‘A’],

    …, ”D”:**[]**, ... **}**,

    }

- Asignamiento de ID de Nodo:
  - archivo: names-*.txt
  - contenido: JSON
  - formato ejemplo:

    {“type”:”names”,

    “config”: **{**“A”:”foo@bar.com”, “B”:”yolo@bar.com”,...**}**,

    }

NOTA IMPORTANTE: No está permitido usar los archivos de configuración para nada más excepto el configurar su propio nodo y descubrir vecinos. A excepción de Dijkstra puro naturalmente y los procesos de Init() que usan esta información, no pueden usar eso para nada mas… por ejemplo, obtener toda la topología y resolver trivialmente de forma estática… en la realidad los routers tienen cables a sus vecinos pero desconocen el resto de la topología, y como pueden ver, los algoritmos a implementar todos son dinámicos.

De igual manera, hacer eso resultará en errores, ya que le quita robustez al algoritmo (cambiamos algo del archivo de nombres, y corremos de nuevo y habrá problemas…). Es el equivalente a que se caiga un nodo, o que se conecte uno nuevo, o que al rato regrese el que se había caído.