REDES DE COMPUTADORAS

CURSO 2019

GRUPO 51

Informe - Obligatorio 3

Autores:

Tatiana Rischewski
Manuel Freire
Juan Ferrand

Supervisores:

Martín Giachino
Federico Rodriguez

13 de octubre de 2019

Contenido

Introducción

Este trabajo se enfocó en la implementación de una de las dos principales funcionalidades de la capa de red: el routing. Para esto se debía utilizar el algoritmo de Dijkstra pero partiendo de hipótesis distintas a las que parte este, ningún nodo sabe a priori ni el número total de nodos ni ninguna información sobre la topología de la red. Esto plantea dificultades añadidas para utilizar el algoritmo en el hecho de recabar (y mantener actualizada) la información de la red. En este trabajo se usó un algoritmo de routeo de tipo “link state” este tipo de algoritmo se basa en tres pilares: conocimiento de toda la red, el uso de floodings controlados y el intercambio de información con toda la red (todos los nodos saben todo).

Problemas

Para la resolución se debieron superar dos problemas del funcionamiento de esta red. El primero era la capacidad de obtener la información necesaria para ejecutar el algoritmo, esto es en definitiva crear una matriz de adyacencia de toda la red conocida. El segundo es el mantenimiento de la información, en una red los costos de los enlaces pueden cambiar haciendo que las decisiones que fueron óptimas en un momento no sean en el siguiente por lo que no puede verse la red como algo estático si no con la capacidad de cambiar.

Obtención de la información:

Para esta parte se utilizó flooding controlado que es “la estrategia en la que cada router envía los paquetes recibidos a todos sus vecinos a excepción del emisor”. Para este uso en particular se eligió uno controlado por número de secuencia, aquí cada router solo reenvía el paquete si “no lo ha visto nunca”, en caso contrario lo descarta. Para implementarlo se añadió información al vector de costos reservando un índice para agregarle un número de secuencia, la estrategia implementada asume un reconocimiento acumulativo (cada nodo mantiene el número de secuencia más grande recibido de cada otro nodo y reenvía solo si el recibido es mayor que este).

Pseudocódigo:

```
reciveUpdate(pkt){
   em = emisor(pkt)
   if(desconocido (em) )
         agregarVectorAMatriz(pkt)
         for each node in vecinos
            if(node <> em)
                  reenviar(pkt)
         flooding()
   else if(pkt.numSeq > actual(em))
         actualizarVector(em, pkt)
```

```
        for each node in vecinos
            if(node <> em)
                reenviar(pkt)
        endfor
    else
        descartar(pkt)
    endif
    if(algo_cambio())
        actualizarTabla()
}
```

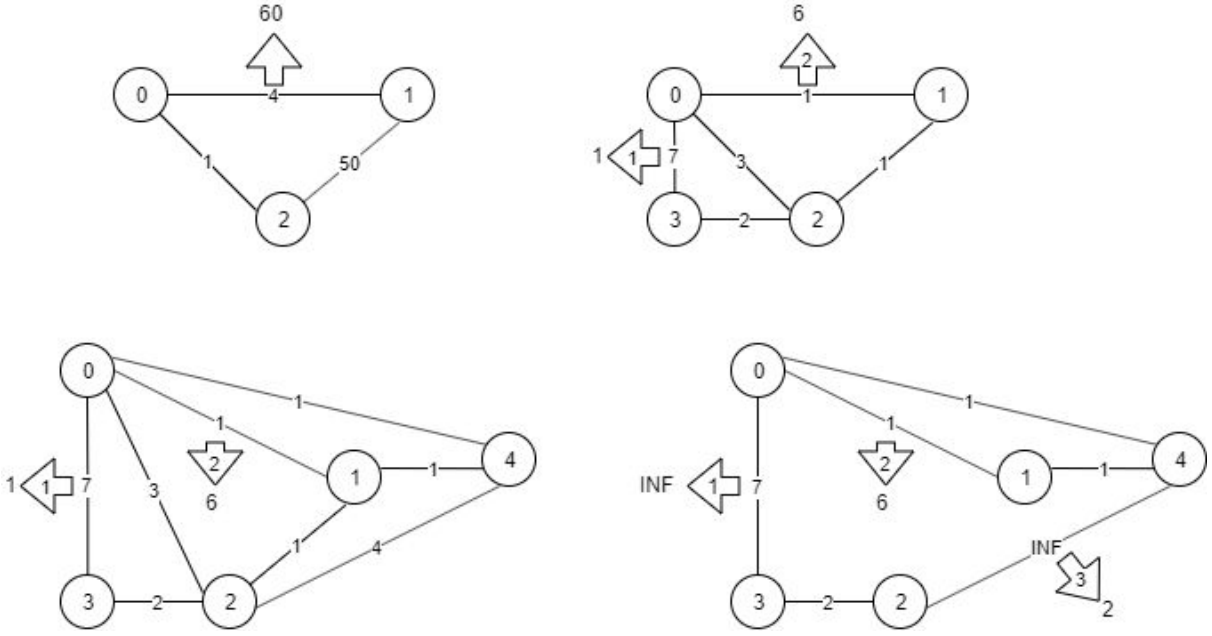
Para finalizar resulta interesante comentar la razón del primer if. Una de las decisiones importantes en el diseño del algoritmo es decidir cuándo se debe hacer flooding pues si bien podría hacerse ante cualquier cambio “por las dudas” eso generaría una enorme sobrecarga en la red. También es evidente que es necesario hacerse por lo menos cuando un nodo recibe un updateLinkCost() ya que debe actualizar las matrices de adyacencia de todos los nodos de la red y solo él tiene esa información. Sin embargo en este algoritmo hay un caso más en el que se hace flooding y este es cuando se recibe información sobre un nodo que hasta el momento era desconocido. La razón detrás de esta decisión es que con la implementación actual cuando un nodo “conoce” a otro no se puede garantizar que este sepa de su existencia (o que lo sabrá en un futuro) por esto es necesario hacer un nuevo flooding.

Coherencia de la información

El otro gran asunto para tener en cuenta es la coherencia de la información, por ejemplo ante la caída de un enlace que divida la red entre dos subgrafos disjuntos es necesario eliminar toda la información con respecto a nodos del otro subgrafo (es decir eliminar las entradas de la matriz de adyacencia respectivas a las aristas desde esos nodos al resto). De no hacer esto la red podría estabilizarse en estados incoherentes o que no representen la realidad. Un ejemplo sería una arista que se borra pero posteriormente se recupera, de haber cambios en las aristas internas de cada subgrafo en el tiempo entre ambos eventos no sería reflejado en el otro subgrafo. Esto es porque la información se distribuiría en la interna de cada subgrafo y no tendría forma de “salir” hacia el otro al ser disjuntos pero al conectarse nuevamente como ya “se conocen” no se generarían nuevos floodings. Por esto en la ejecución del algoritmo de dijkstra para actualizar la tabla de routing se agregó una etapa de actualización de la matriz que elimine todas las aristas no alcanzadas.

Pruebas realizadas

Se probó el funcionamiento del algoritmo con cuatro grafos distintos, los tres provistos por los docentes en la sección tests más uno de trabajo propio que busca probar el caso de una red que se parte en dos y luego se une. El valor INF en una arista implica que esta no existe mientras tiene ese valor.



Conclusión

El algoritmo de ruteo de utilizando Dijkstra está lejos de ser una solución óptima en redes (como es el caso de estas) que cambian su topología constantemente ya sea en valores de aristas o nodos nuevos/que se van. El gasto de recursos para mantener una matriz de adyacencia de la parte conocida de la red ($O(k^2)$ de memoria siendo k la cantidad de nodos conocidos y por tanto k<=n). no se justifica. Esto es porque la matriz debe ser modificada cada vez que hay un cambio en la red y ejecutar nuevamente un Dijkstra. Comparándolo con el método de distance vector vemos que ambos tienen un comportamiento similar (ambos guardan en memoria información sobre todos los otros nodos y ante cada actualización deben revisarla) sin embargo el link state lo hace en un orden n (simplemente al recibir una actualización para cada nodo que recibe debe calcular si el costo supera al que tenía) y las mejores implementaciones de dijkstra son en orden mlogn siendo m la cantidad de aristas. En un grafo con m parecido a n el uso de este algoritmo pierde el sentido pues tiene un consumo mayor o igual de memoria y un costo de actualización en tiempo mayor o igual al distance vector