ELEN 4017

Network Fundamentals

Lecture 25 & 26

Purpose of lecture

Chapter 4: Network Layer

- Routing algorithms
- Link State Routing Algorithm

Terminology

- **Default router/first hop router**: - this is the router to which a host is connected.
- **Source router** is the first-hop router connected to the source host.
- **Destination router** is the first-hop router connected to the destination host.
- The purpose of a routing algorithm is to find a ‘**good**’ path from source to destination router.

Graph theory

- A graph is used to formulate routing problems.
- Graph G = (N,E)
  - N is the set of nodes
  - E is the set of edges, where each edge is a **pair of nodes** from N.
- For network layer routing, the **nodes are the routers** and the **edges are the physical links** connecting the routers.

Definitions

- N = {u,v,w,x,y,z}
- E = {(u,x), (u,v), …}
- Each **edge has a cost** associated with it, denoted by **c(x,y)**
- If the pair (x,y) does not belong to E, then the cost is given as infinite → **c(x,y) = ∞**
- Graphs are un-directed → c(x,y) = c(y,x)
- Node y is said to be a **neighbour** of node x if E contains (x,y)

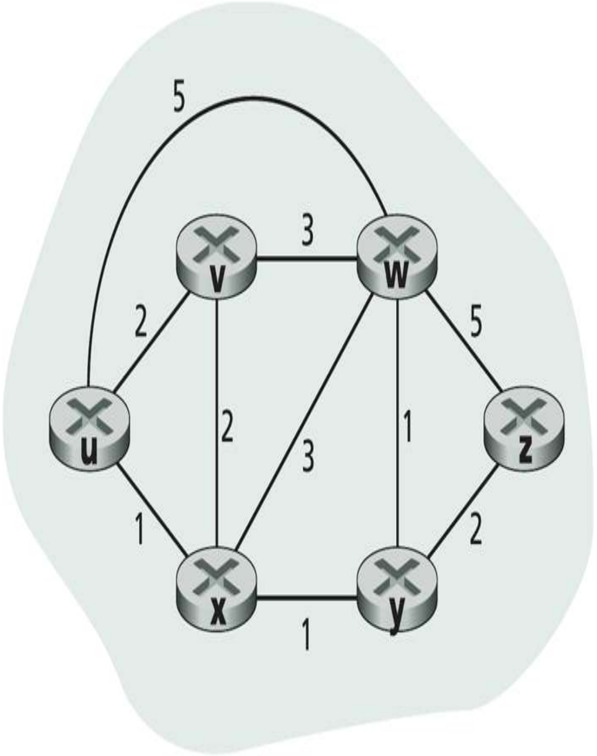


Figure 4.25 ♦ Abstract graph model of a computer network

Least cost path

- A goal of a routing algorithm is to find a **least-cost path**.
- A path in a graph G = (N,E) is a **sequence of nodes ($x_1, x_2, \ldots, x_p$)** such that the **pairs (x_1, x_2), (x_2, x_3), … (x_{p-1}, x_p) are edges in E**.
- The cost of the path is **$c(x_1, x_2)$** + **$c(x_2, x_3)$** + …

- What is the least cost path from **u** to **w** ?
- How did you calculate it ?

Your calculation is an example of a centralized (global) routing algorithm → all calculations are done in one place and has access to the state of entire system.

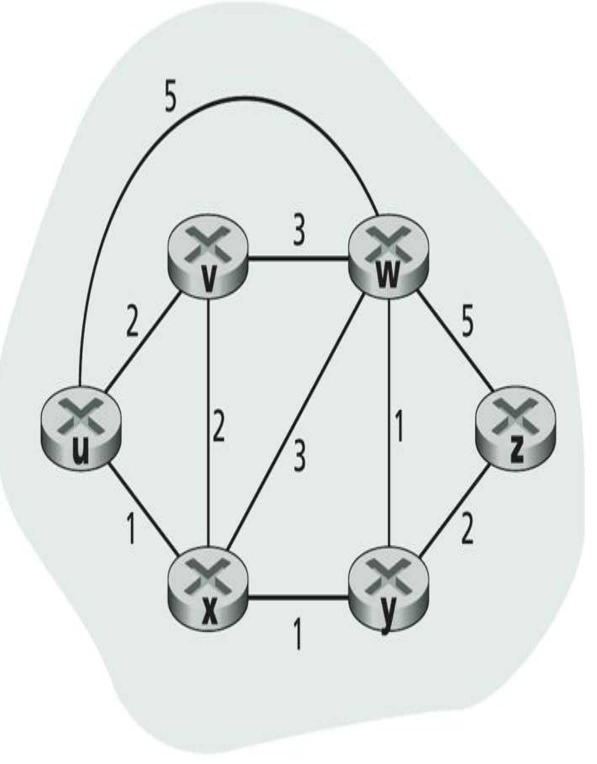


Figure 4.25 ♦ Abstract graph model of a computer network

Classifying algorithms

- Global vs decentralized
- Static vs dynamic
- Load sensitive vs insensitive

Global vs Decentralized

- **Global**: computes least cost path using complete, global knowledge of the network. These are referred to as **link-state algorithms** since the state of all links must be known.
- **Decentralized**: calculation done iteratively and distributed.
  - No node has complete information about all network links.
  - Each node begins with knowing the cost of its direct links.
  - Through an iterative process of calculation and information exchange with its neigbouring nodes, a node gradually calculates the least cost path.
  - One class of decentralized algorithms are the **distance vector (DV) algorithms.**

Static vs Dynamic

- Static refers to cases where routes don’t change at all, or very infrequently due to human configuration.
- Dynamic refers to the case where routes are updated based on network topology and load. It can run periodically on in response to an event.

Load sensitive

- Load sensitive refers to the ability to adjust the link costs based on current congestion level.
- There have been attempts to implement load sensitive algorithms, but they are problematic.
- Todays Internet algorithms are not load sensitive.

Purpose of lecture

Chapter 4: Network Layer

- Routing algorithms
- Link State Routing Algorithm

Dijkstra's algorithm

- This algorithm requires states of all other nodes to be known.
- This is accomplished by having each node **broadcast link-state packets** to **all** other nodes.
- We will consider Dijkstra's algorithm:
  - It is iterative.
  - It has the property that **after the kth iteration**, the least cost paths **are known to k destination nodes**.
  - Among the least-cost paths to all destination nodes, these k paths will have the k smallest costs.

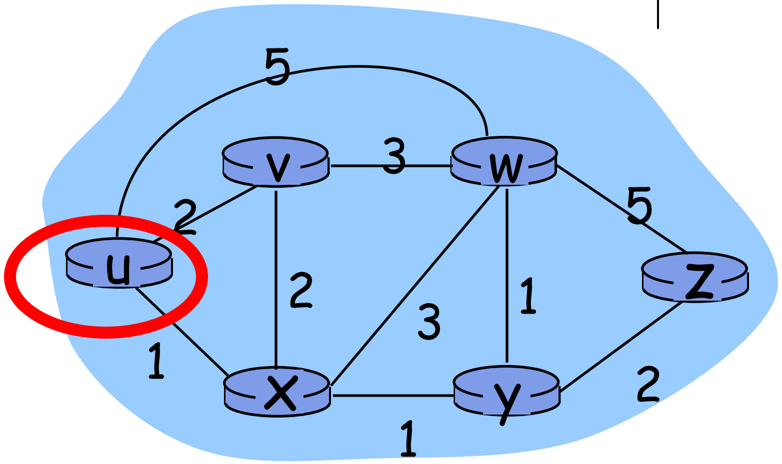
Dijkstra's algorithm

- D(v) : **cost of the least-cost path** from the source node to destination **node v** as of this iteration of the algorithm
- p(v) : **previous node** (neighbour of v) along the current least-cost path from the source to v.
- N' : **subset of nodes**.
  - v is in N' if the least-cost path from the source to v is definitely known.

Dijkstra's algorithm

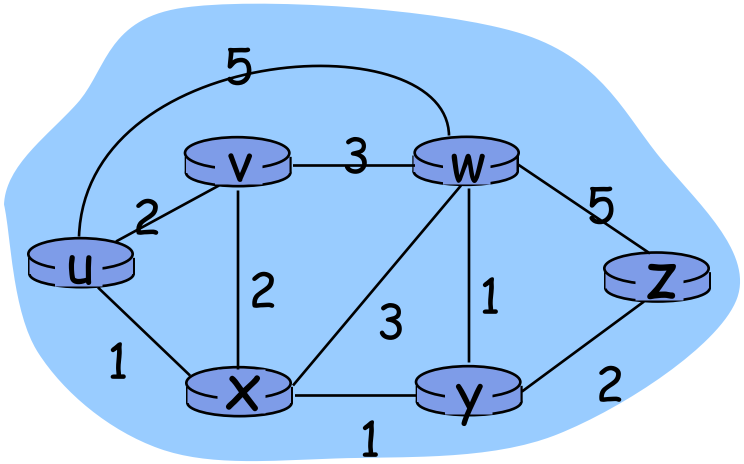
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
   find w not in N' such that D(w) is a minimum
   add w to N'
   update D(v) for all v adjacent to w and not in N' :
      D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
    shortest path cost to w plus cost from w to v */
until all nodes in N'
```



Dijkstra's algorithm example

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



- Initialize values from u to neighbours v,x,w
- Look at nodes not in N' and add the least cost. In this it is x.
- Recompute all costs.
- In case of multiple paths with same cost, choose arbitrarily.
- When algorithm terminates, for each node **we have the predecessor along least cost path**.

Applet

- http://www.unf.edu/~wkloster/foundations/DijkstraApplet/DijkstraApplet.htm