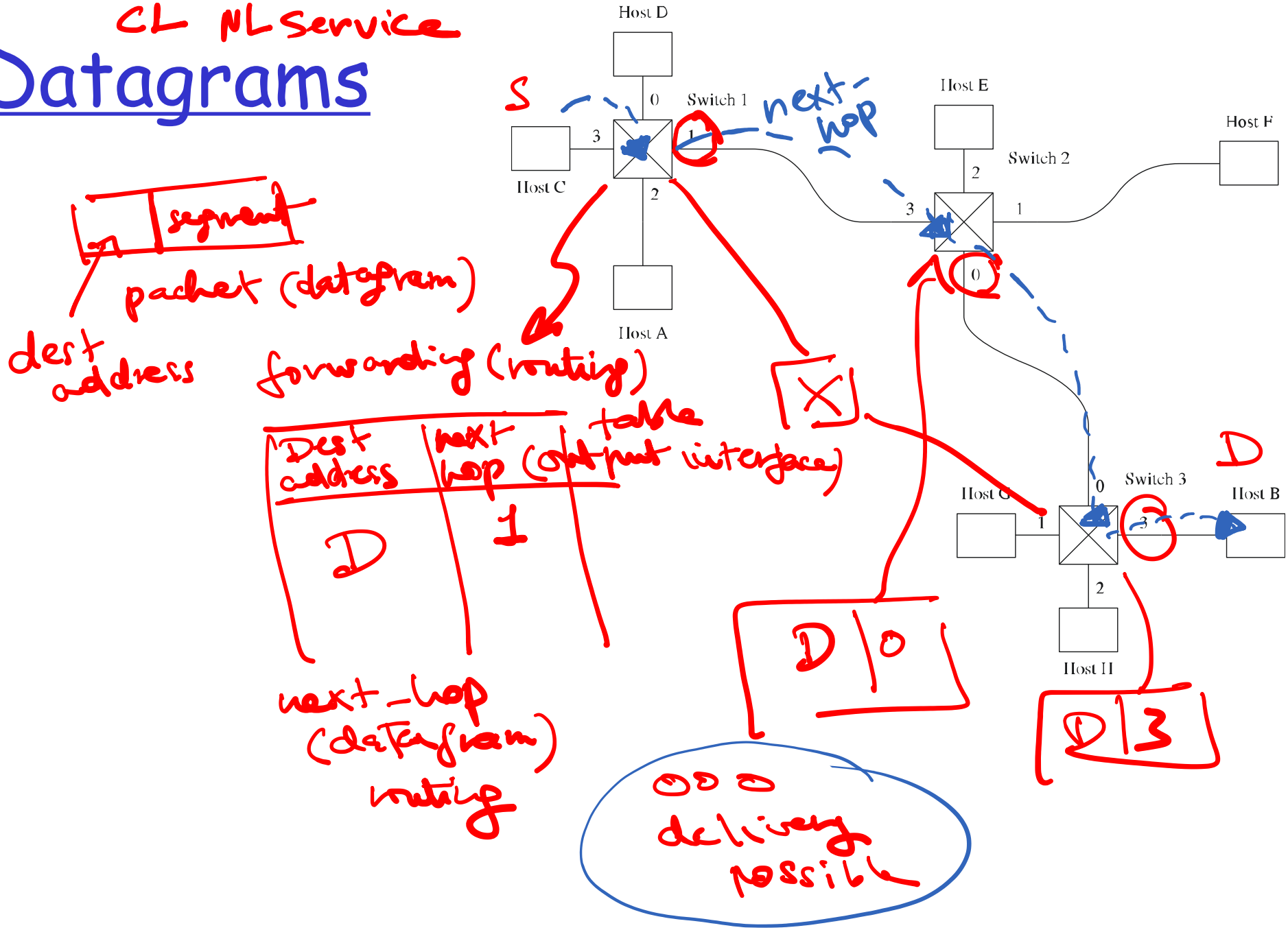
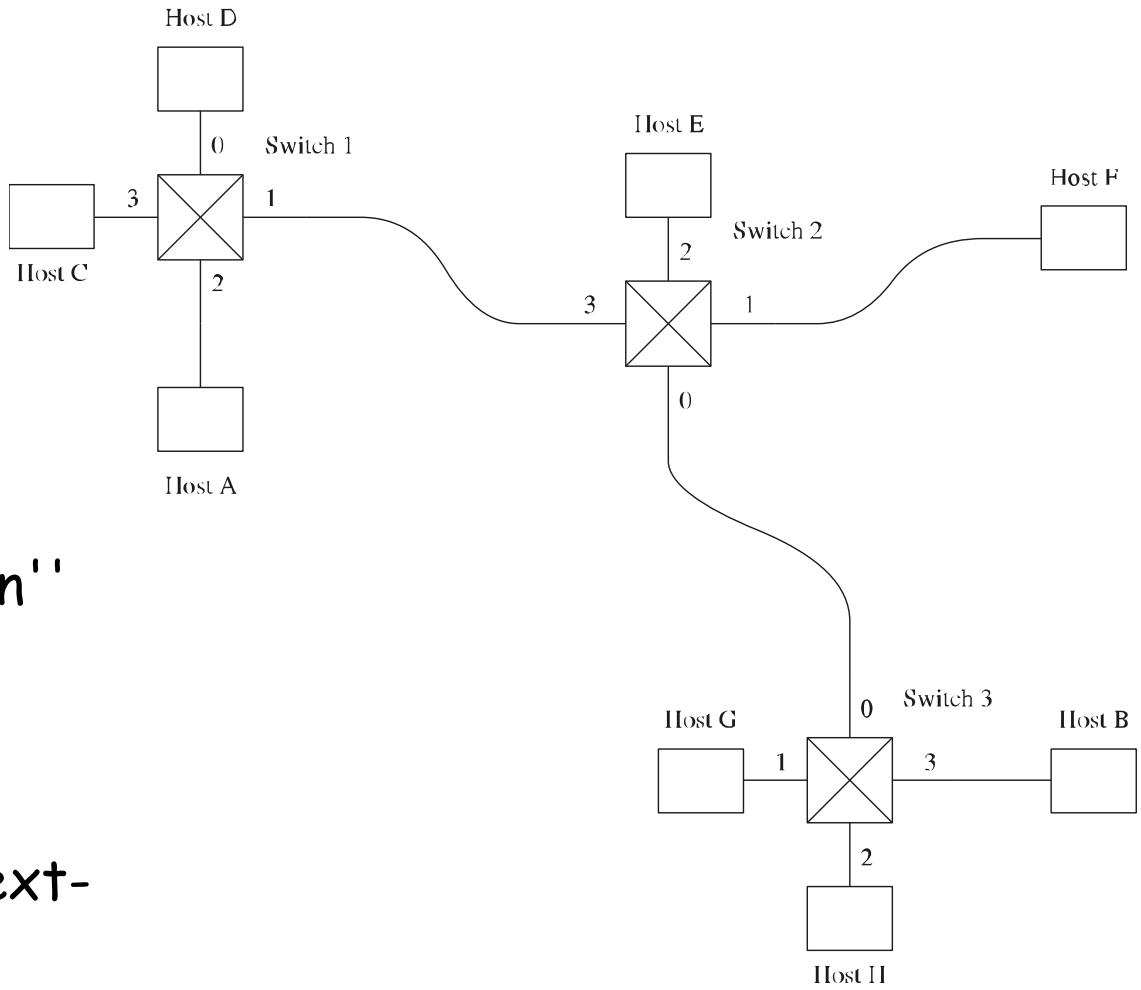
CL NL service

Datagrams



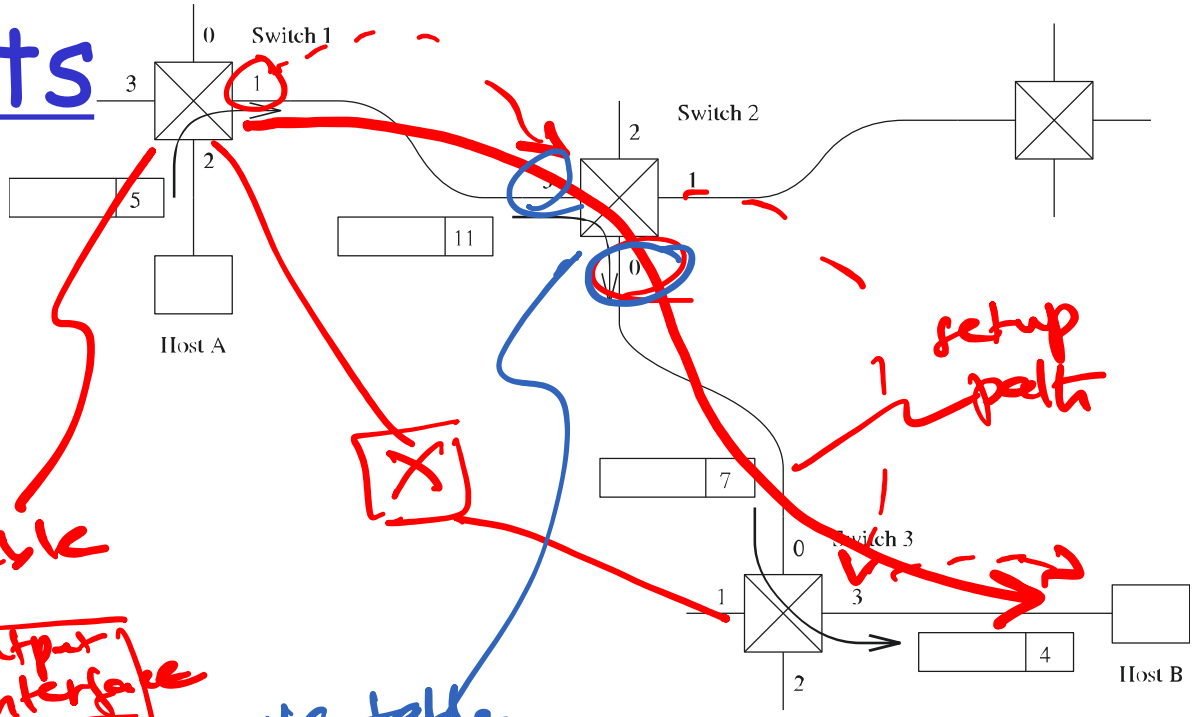
Datagrams

- No connection setup phase
- Each packet forwarded independently based on destination address
- Each packet in ``connection'' may follow different path
- Analogy: postal system
- Sometimes called *connectionless* model (*or* next-hop routing)
- Each switch maintains a forwarding (routing) table: {destination address, output port (or next-hop)}



CO NL service

Virtual Circuits



dest address

setup packet next-hop routed

setup path

VC table

VCI	output interface
X	1

globally unique VC #

VCI x

data packet

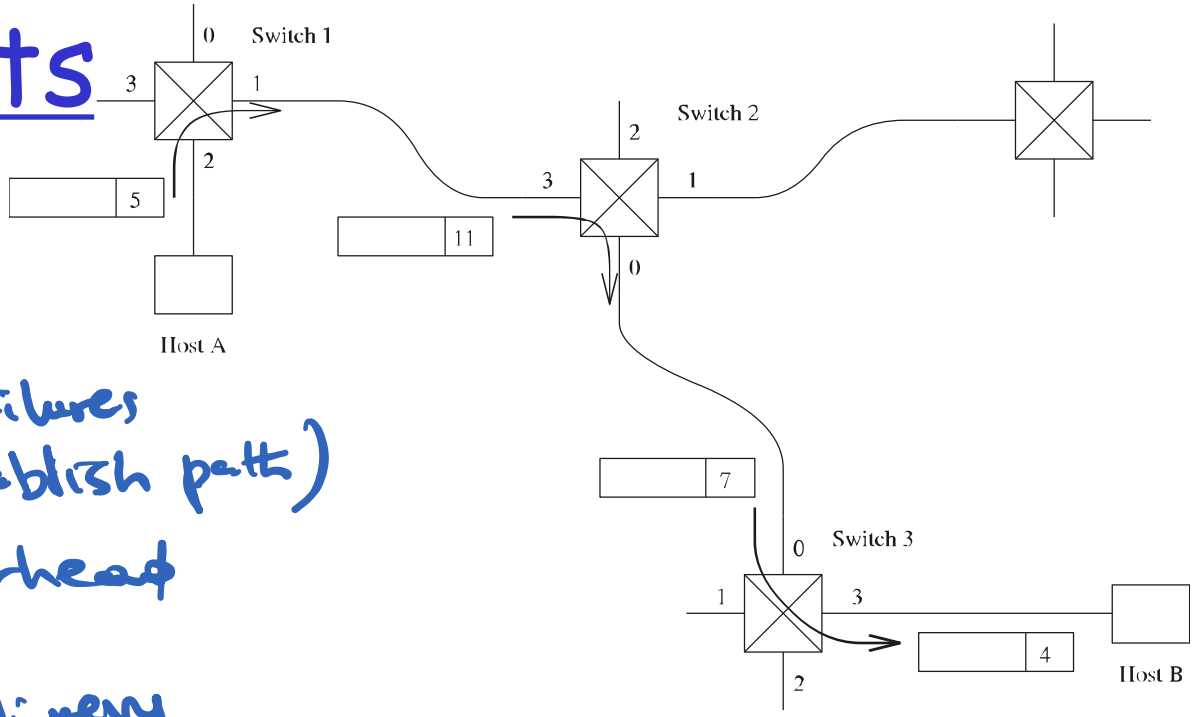
VC table

VCI in	i/p interface	VCI out	o/p interface
z	3	y	0

locally unique

VCI/label swapping

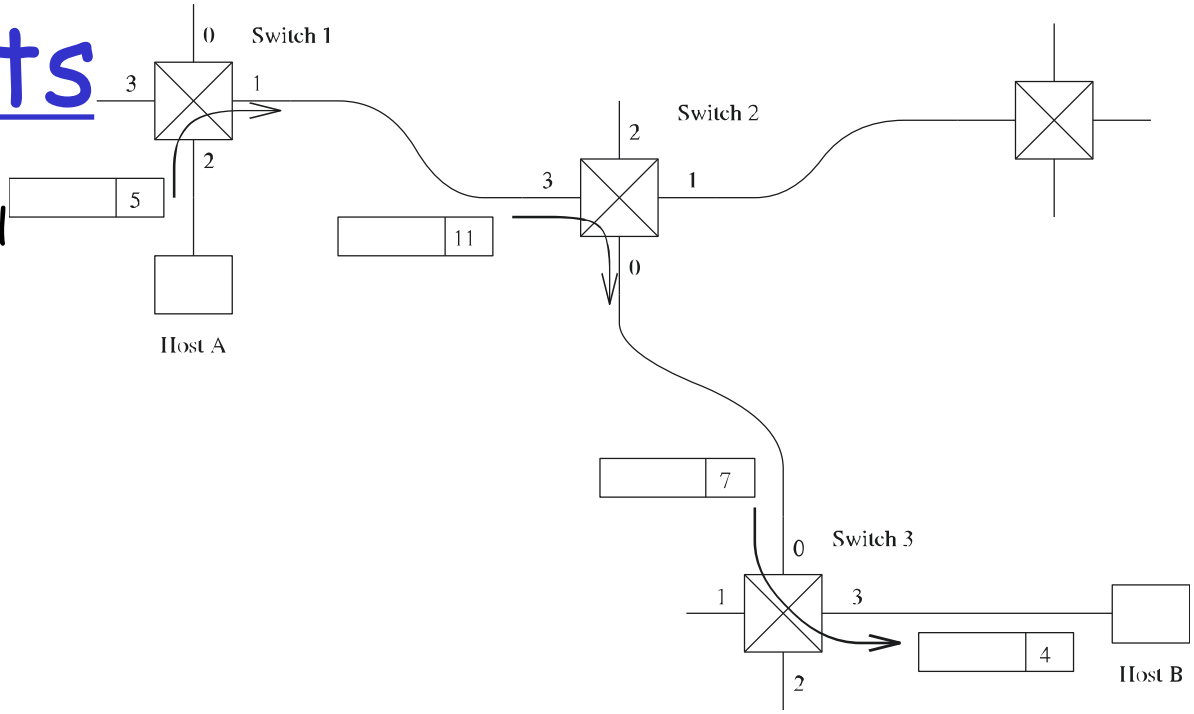
Virtual Circuits



- wait for RTT
- not resilient to failures (need to reestablish path)
- VC tables overhead

+ in-order delivery
+ performance guarantees (QoS)
+ per data packet overhead smaller

Virtual Circuits



- Explicit connection setup (and tear-down) phase
- Setup packet is routed using next-hop routing
- Subsequent packets follow same route, thus arrive in order
- Analogy: phone call
- Sometimes called ***connection-oriented*** model (*or* VC routing)
- Each switch maintains a VC table: {input VCI, input port, output VCI, output port}
- VCI in header is interchanged by nodes as they forward data packet

Virtual Circuits versus Datagrams

Virtual Circuit Model:

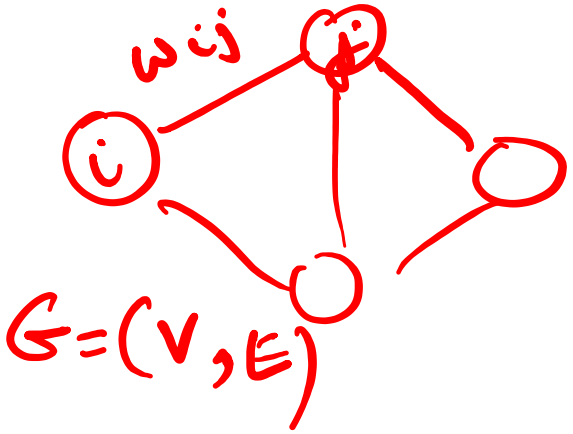
- Typically wait full RTT for connection setup before sending first data packet
- While the connection request contains the full address of destination, each data packet contains only a small identifier, making the per-packet header overhead small
- Need per-connection state at switches
- If a switch or a link in a connection fails, the connection is broken and a new one needs to be established
- Connection setup provides an opportunity to reserve resources (to guarantee QoS)

Virtual Circuits versus Datagrams (cont'd)

Datagram Model:

- There is no round trip time delay waiting for connection setup; a host can send data as soon as it is ready
- Since every packet must carry the full address of the destination, the overhead per packet is higher than for the connection-oriented model
- No connection state in switches
- Since packets are treated independently, it is easier to route around link and node failures
- Source host has no way of knowing if the network is capable of delivering a packet or if the destination host is even up

How do Switches Maintain Routing Tables?

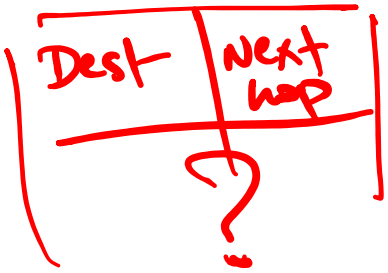


SP problem

Find shortest path P

s.t. $w(P) = \sum_{(i,j)\in P} w_{ij}$ is min

$w_{ij} = \begin{cases} 1 & \text{min-hop routing} \\ \text{delay}_{ij} & \text{min-delay routing} \\ \text{cost}_{ij} & \text{min-cost routing} \end{cases}$

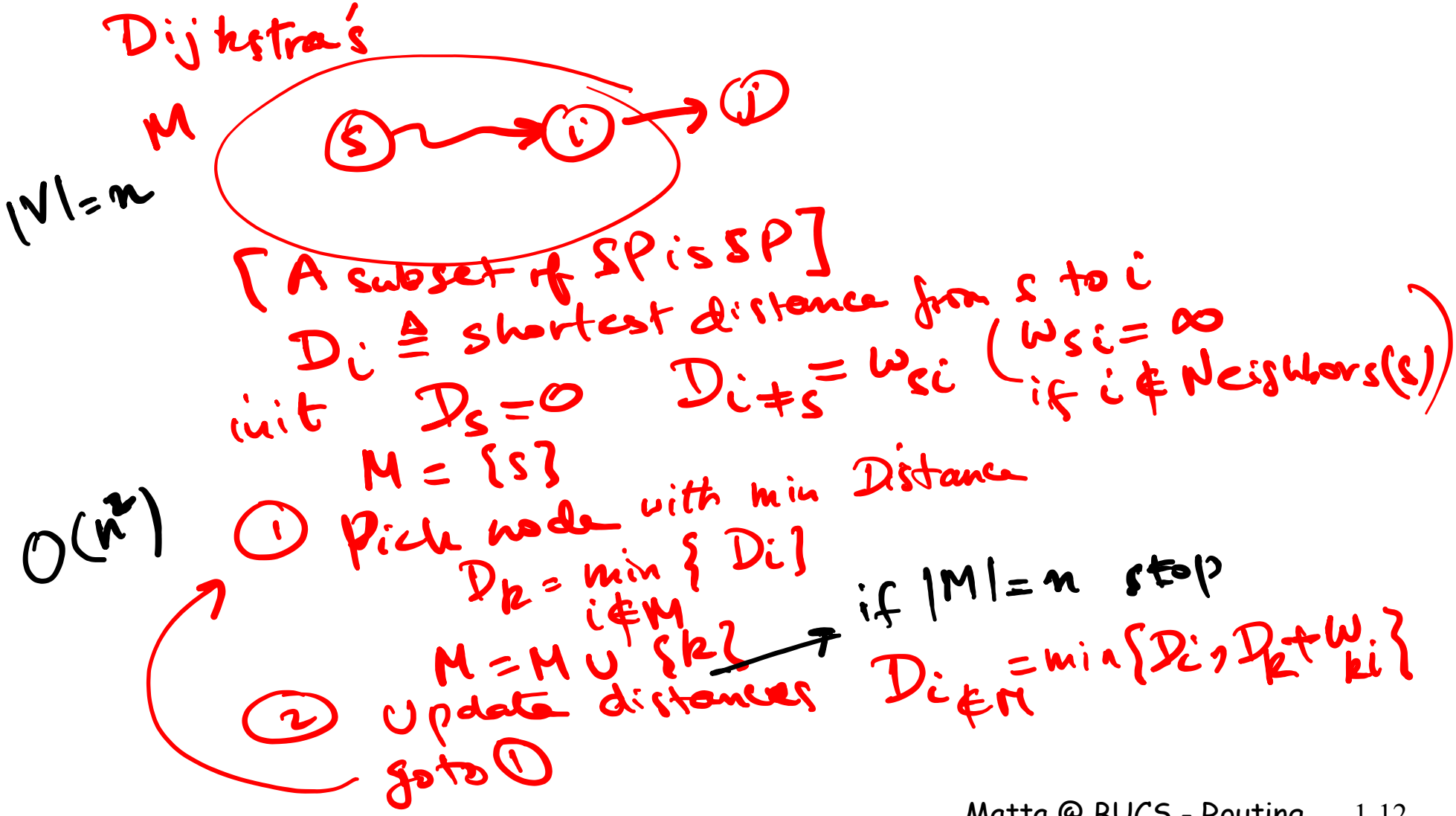


Distributed routing protocols

- Link-state (Dijkstra's)
- Distance vector (Bellman-Ford)

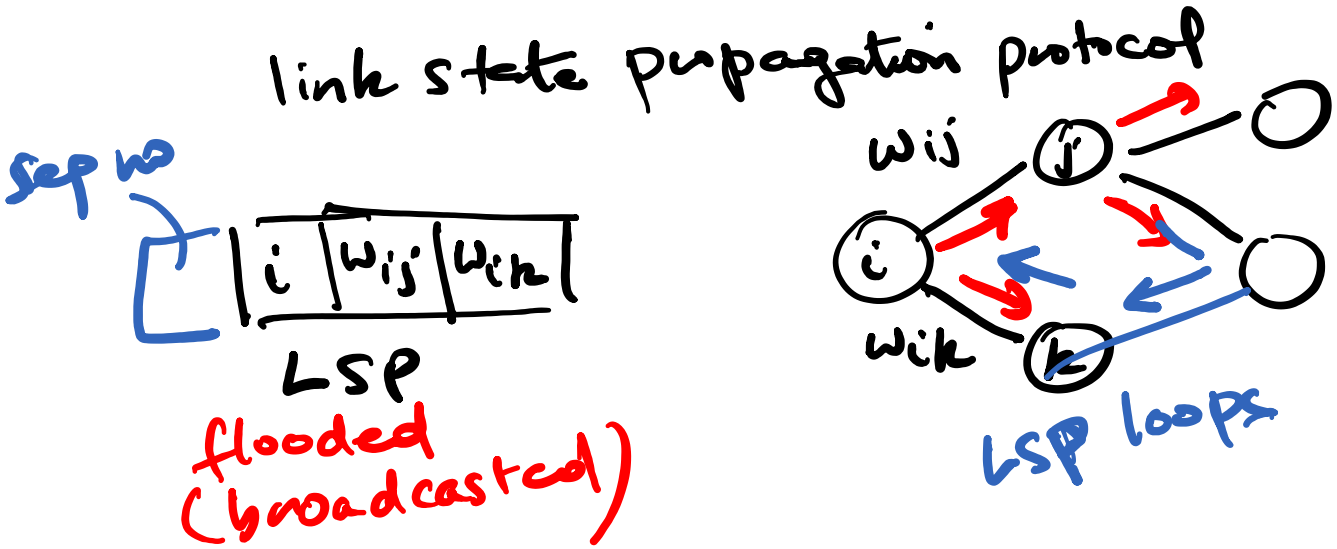
How do Switches Maintain Routing Tables? (cont'd)

Link-state (SPF -- Shortest Path First)



How do Switches Maintain Routing Tables? (cont'd)

Link-state (SPF -- Shortest Path First)

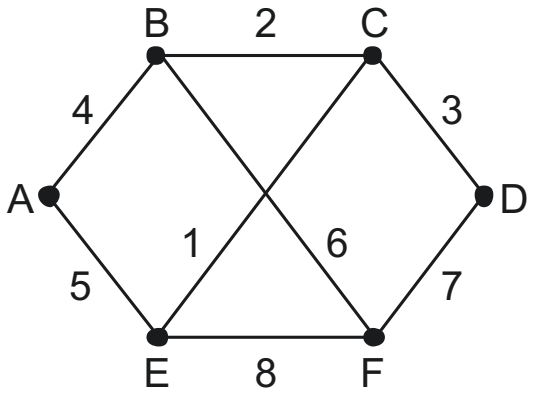


How do Switches Maintain Routing Tables? (cont'd)

Link-state (SPF -- Shortest Path First)

- Each node maintains a complete view of the topology: each node periodically broadcasts to all nodes (not just neighbors) the costs/status of its outgoing links (not entire routing table)
  - Link State Packet (LSP)
    - id of the node that created the LSP
    - cost of link to each directly connected neighbor
    - sequence number (SEQNO)
    - age or time-to-live (TTL) for this packet
- Reliable Flooding
  - store most recent LSP from each node
  - forward LSP to all neighbor nodes but one that sent it
  - generate new LSP periodically; increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP; discard when TTL=0
- When a link state message arrives and the view changes accordingly, the node recomputes routes by applying ``**Dijkstra's shortest path algorithm**''

LS Propagation Protocol



(a)

Link State Packets

A
Seq.
Age
B	4
E	5

B
Seq.
Age
A	4
C	2
F	6

C
Seq.
Age
B	2
D	3
E	1

D
Seq.
Age
C	3
F	7

E
Seq.
Age
A	5
C	1
F	8

F
Seq.
Age
B	6
D	7
E	8

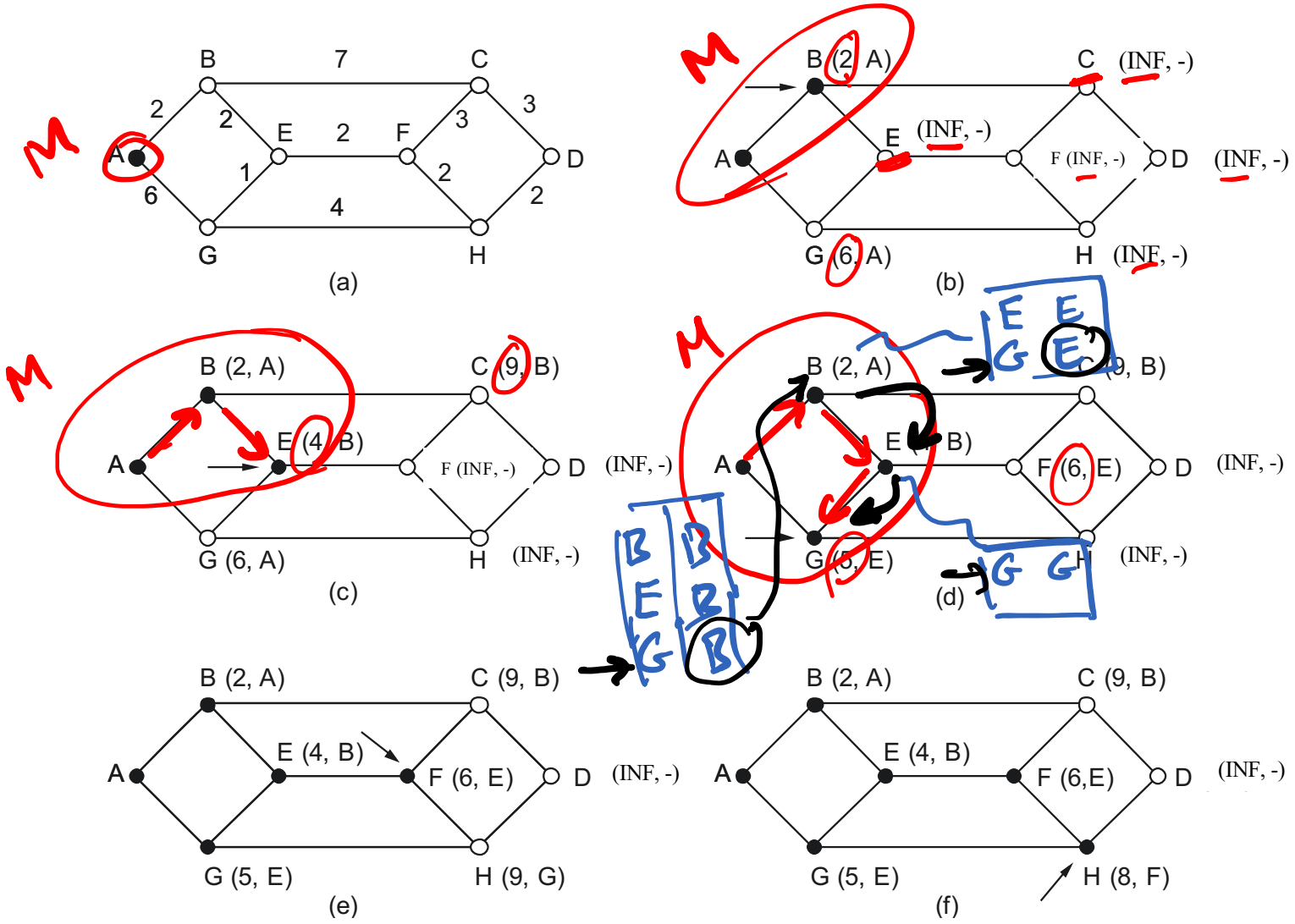
(b)

Route Calculation in Link State

- Dijkstra's shortest path algorithm
- N denotes set of nodes in the graph
- l(i,j) denotes non-negative cost (weight) for edge (i,j)
- s in N denotes this node
- M denotes the set of nodes incorporated so far
- C(n) denotes cost of the path from s to node n

```
M = {s}
for each n in N - {s}
        C(n) = l(s,n)
while (M != N)
        M = M U {w}  such that
                C(w) is the minimum for all w in (N-M)
        for each n in (N-M)
                C(n) = MIN(C(n), C(w)+l(w,n))
```

Dijkstra's Algorithm in action



How do Switches Maintain Routing Tables?

Distance-vector (Bellman-Ford)

$D_i^h \triangleq$ shortest distance from i to some dest s.t. length of path $\leq h$ links/hops

init $D_{dest}^h = 0$ $\quad D_{i \neq dest}^0 = \infty$

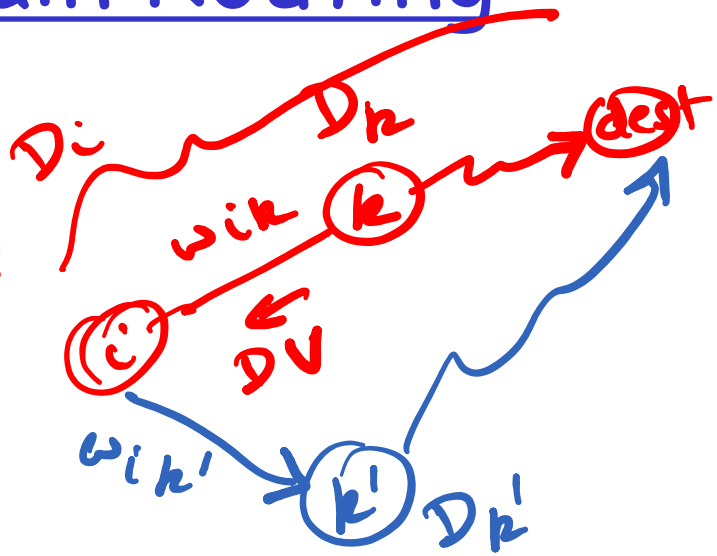
for $h = 1, 2, 3, \ldots, n-1$

$|V| = n$ $\quad D_i^h = \min_{k \in \text{Neighbors}(i)} \{ D_k^{h-1} + w_{ik} \} \; \forall i$

$O(n^3)$

How do Switches Maintain Routing Tables?

Distance-vector (Bellman-Ford)



Periodically, each node sends a DV = {(dest, cost)}

Upon receipt of DV, node i updates its routing table if

1) unknown dest, take "k" as next hop
2) better distance, switch to k as next hop
3) k is next hop, update your distance

How do Switches Maintain Routing Tables?

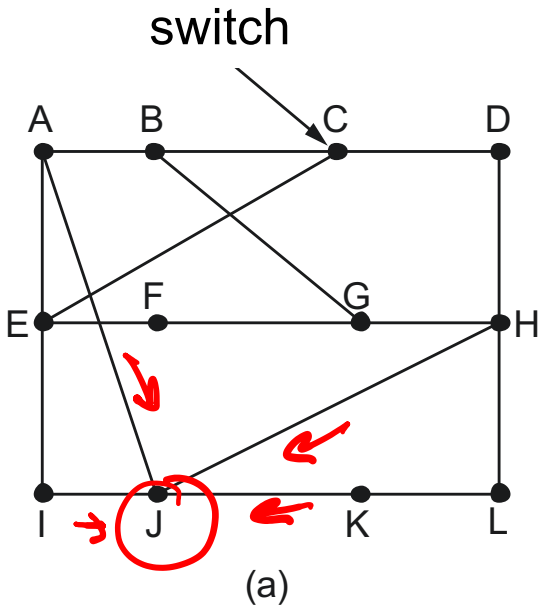
Distance-vector (Bellman-Ford)

- Each node keeps in its routing table its current best distance (cost) to each destination node: (Destination, Cost, NextHop)
- Initially, the routing table of each node contains its neighbor nodes and distances to them
- Each node sends a copy of its routing information {(Destination, Cost)} to each of its neighbor nodes
  - periodically (on the order of several seconds)
  - whenever its table changes (called *triggered* update)
- Upon receiving a message from a neighbor J, a node K updates its routing table as follows:
  - If node J knows a shorter path to some destination, node K updates its corresponding entry with J as the new next-hop
  - If node J reports a distance for an unknown destination, node K adds a new entry for it with J as the next-hop
  - If node K is using node J as its next-hop to some destination node and J reports a change in its distance to it, K updates the corresponding entry

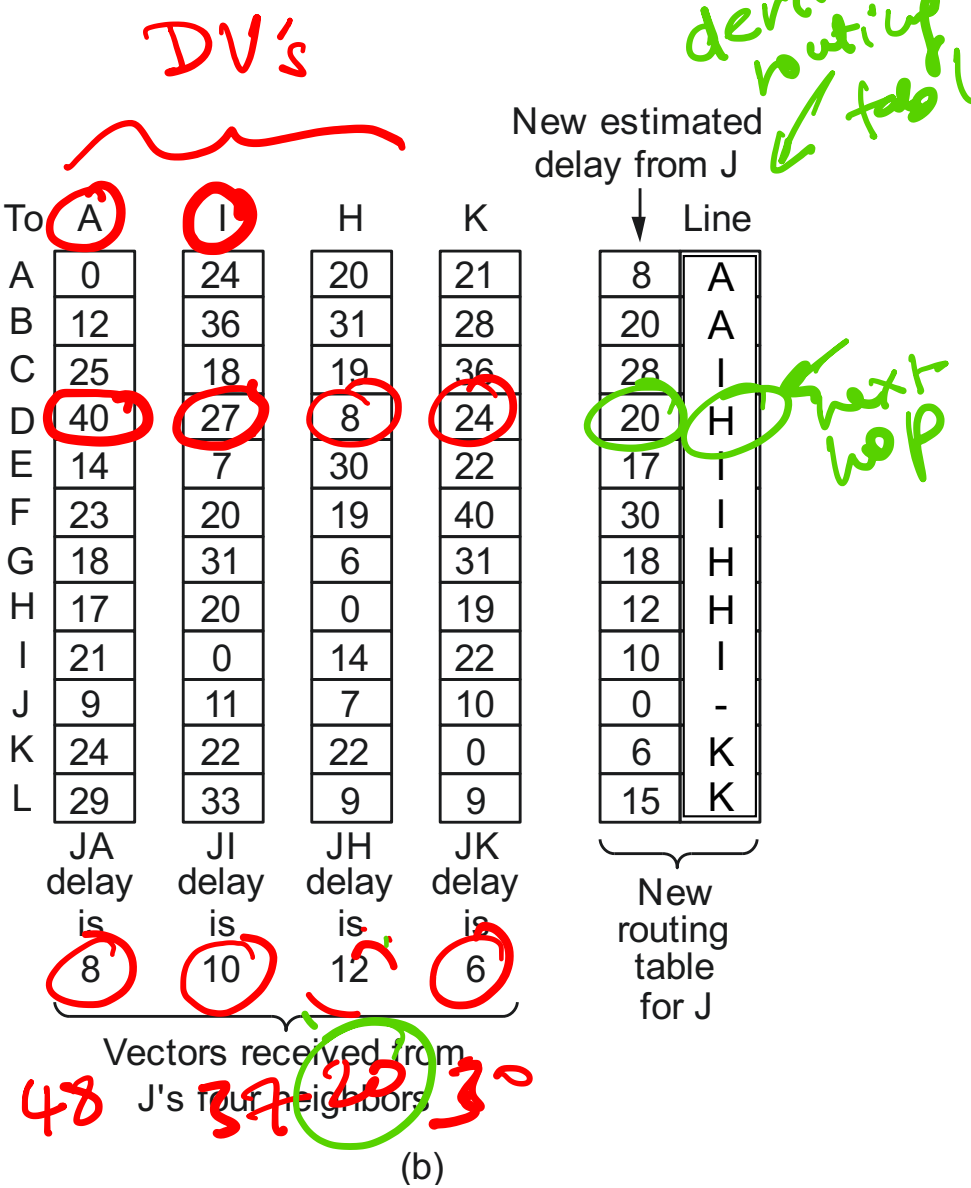
Route Calculation in Distance Vector

```
void mergeRoute (Route *new) {
    int i;
    for (i = 0; i < numRoutes; ++i)  {
        if (new->Dest == rt[i].Dest) {
                if (new->Cost + 1 < rt[i].Cost)  // min-hop routing
                        break;
                else if (new->NextHop == rt[i].NextHop)
                        break;
                else
                        return;
        }
    }
    rt[i] = *new;
    rt[i].TTL = MAX_TTL;
    ++rt[i].Cost;
    if (i == numRoutes)
        ++numRoutes;
}
```

DV Routing



(a)



(b)

Distance Table
distance to each dest via each neighbor

dest D [48 37 29 30]