The Networking Layer

Routing

Quick Review: Routing Protocols

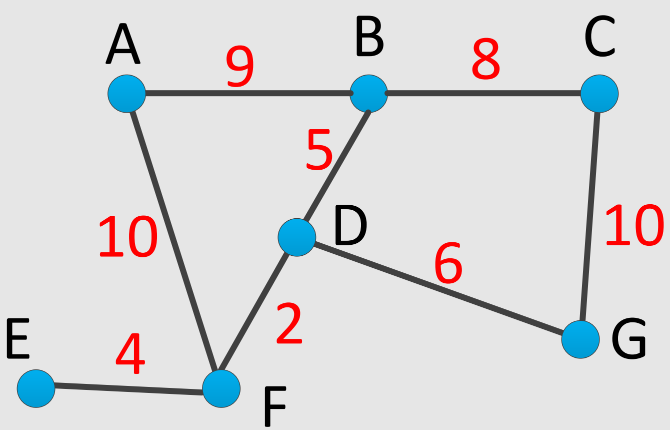
- Each node runs the routing algorithm (e.g., Dijkstra algorithm), then maintain its own routing table
- When packet arrives, forward to next hop …
- How is the routing table constructed?
- Distance vector vs. Link state
  - distributed methods for building routing tables that converge to the shortest path tables
  - 1) collect topology information; 2) run routing algorithm

Link State Protocols

- Each router distributes information it has to every other router
- Information is in the form of *Link State Announcements (LSAs)*
  - (myID, neighborID, link cost)
- Distribution is by controlled flooding
  - Distribute along each link but receiving one
- Now, each router has complete information
- OSPF is a prevalent example
  - SPF with LSA to find arc costs

Link State Routing

- Open Shortest Path First (OSPF)
- Nodes exchange link state advertisements with their neighbors
- Link State Advertisements: info about links connected to a node (LSA) (delay + node ID of other end of link)



G receives LSA from C and D:

C: [B, 8; G, 10]

D: [B, 5; F, 2; G, 6]

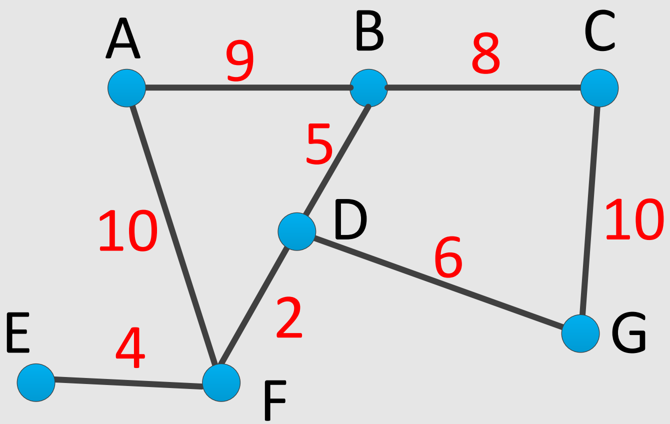
- When a node receives an LSA, it forwards it on all of its links

LSA Routing Algorithm

Each router does the following

1. "Meets the (immediately adjacent) neighbors" and learns their IDs
2. Builds an LSA containing IDs and distance to each of its neighbors
3. Transmits the LSA to all other routers
4. Stores the most recent LSA from every other router in the network
   – Not just from its immediate neighbors!
5. Creates a "map" of the network topology from LSAs
6. Computes routes (to store in forwarding table) from its local map of the topology

Link State Routing



Round 1:

G receives LSA from C and D:

C: [B, 8; G, 10]

D: [B, 5; F, 2; G, 6]

Round 2:

G receives LSA from B and F:

B: [A, 9; C, 8; D, 5]

F: [A, 10; D, 2; E, 4]

Round 3:

G receives LSA from A and E:

A: [B, 9; F, 10]

E: [F, 4]

What is the next after G knows everything?

Run Dijkstra algorithm directly

Generating LSAs

- A router generates LSAs *periodically* → background refresh rate
- A router also generates LSAs when its local environment changes
  - it has a new neighbor (router comes online)
  - a link goes down (indicated by absence of "Hello" packets)
  - the cost of a link to an existing neighbor has changed
- Limiting the overhead (network bandwidth) consumed by routing messages, particularly LSAs
  - set a minimum interval between successive updates

Link State Routing (Summary)

- Based on global knowledge
- Converges Faster
- No count to infinity problem

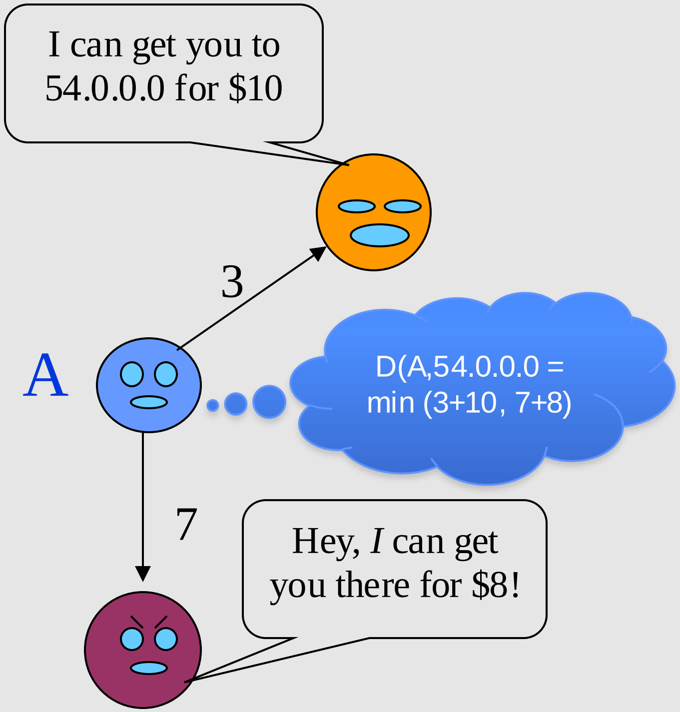
- But …
- Require more network resources (bandwidth)
- Heavy traffic due to flooding of packets
- Flooding may result in infinite looping which can be solved by using the Time to live (TTL) field

Distance Vector (DV) "Routing"

- Asynchronous, iterative distributed computation
  - exchange of routing information: (destination, min_distance)
  - Diff: exchange info with neighbors only
  - computation step: based on Bellman-Ford method
- **Routers do not store or compute the network topology; they only store distance / next hop information**
- Used by many protocols: RIP, BGP, etc.
- Advertise distances to route prefixes (networks, subnets, hosts), not routers

Bellman's Equation

- For any node A,
  - Edge costs $l(i,j)$ =
    - Cost of link from i to j, if exists
    - ∞, otherwise
- Beyond neighborhood, trust neighbors' claims
  - $D(i,j) = \min_k \{l(i,k) + D(k,j)\}$
  - $D(i,i) = 0$



- Node A forms vector of $D(A,j)$, distributes to neighbors
  - So does every other node
  - Nodes collectively and iteratively learn about better and better paths

Distance-Vector Algorithm (Each Router)

- Start with a distance vector consisting of the value
  - "0" for itself
  - "infinity" for every other destination
- Link cost to neighbors is available, through direct measurement, or administrator configuration
- Transmit its distance vector to each of its neighbors
  - when the link to the neighbor first comes up
  - whenever the information changes → triggered updates
  - periodically (even if there are no changes)
- Saves the most recent distance vector from each neighbor
- Calculates its own distance vector using Bellman's equation
- Iterate

$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\} = \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2+1\ ,\ 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

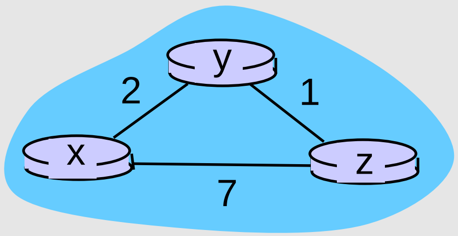
node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

time



Quiescent State
until link cost changes

Properties and Problems

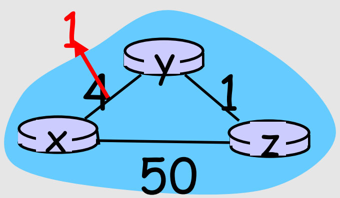
- DV algorithm eventually will converge on shortest paths
  - as long as state of the links / routers remains stable

- During convergence, non-shortest paths and loops may develop
  - "good news travels fast, bad news travels slowly"
  - count-to-infinity problem

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



Before the change DVs

$D_x = [0,4,5]$
$D_y = [4,0,1]$
$D_z = [5,1,0]$

"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors. → changes is DV to be $D_y = [1,0,1]$

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* $D_z = [2,1,0]$ sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *NOT* change, so *y* does *not* send a back message to *z*.

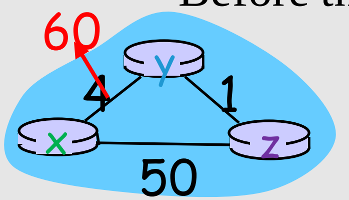
2 iterations needed to get to quiescent state

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- 44 iterations before algorithm stabilizes

Before the change DVs



$$D_x = [0,4,5]$$
$$D_y = [4,0,1]$$
$$D_z = [5,1,0]$$

bad news travels slow - "count to infinity" problem!

Before link cost changes

at node y
$c(y,x) = 4$ and $c(y,z) = 1$
$D_x(x) = 0$ and $D_z(x) = 5$

at node z
$c(z,x) = 50$ and $c(z,y) = 1$
$D_x(x) = 0$ and $D_y(x) = 4$

t_0: y detects the link cost change

$D_y(x) = \min\{c(y,x) + D_x(x),\ c(y,z) + D_z(x)\} = \min\{60, 6\}$ = 6 (Since $D_z(x)$ has not been updated yet, actually loop back to itself)

t_1 : y informs z of it's new cost to x

t_2 : z computes its new cost to x via y to be $D_z(x) =$ 6+1=7

t_3 : z informs y of it's new cost to x

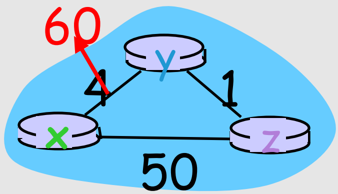
t_4 : y computes its new cost to x via z to be $D_y(x)$ =7+1=8

wrong!
Routing Loop

Notation: $D_x(y)$ is the estimate of least cost from count-to-infinity problem x to y

Distance vector: link cost changes

link cost changes: node detects local link cost change , 44 iterations before algorithm stabilizes



Before link cost changes

at y
y[4,0,1]
z[5,1,0]

Before link cost changes

at z
y[4,0,1]
z[5,1,0]

link cost detected at y

y[6,0,1]
z[5,1,0]

$D_y(x) = 6$

y[6,0,1]
z[7,1,0]

$D_z(x) = 7$

y[8,0,1]
z[7,1,0]

$D_y(x) = 8$

y[8,0,1]
z[9,1,0]

$D_z(x) = 9$

⋮

$D_y(x) = 50$

y[50,0,1]
z[50,1,0]

$D_z(x) = 50$

y[51,0,1]
z[50,1,0]

$D_y(x) = 51$

y[51,0,1]
z[50,1,0]

No more changes in DV

Solution: poisoned reverse:
If z routes through y to get to x then z tells y its (z's) distance to x is infinite (so y won't route to x via z)

What if the link x-y breaks (4->infinity)? … Count to infinity!

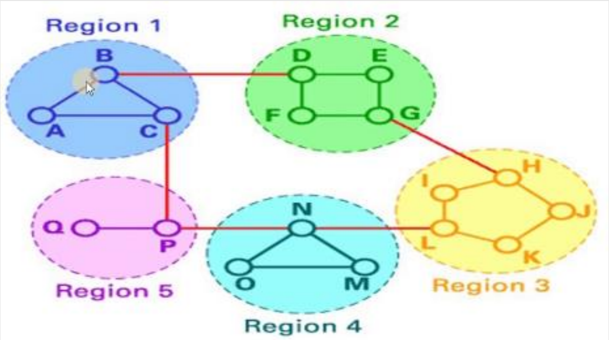
DV vs. LS

Characteristics	DV	LS
# entries per update	O(N)	O(# neighbors)
# propagated updates per round	O(# neighbors)	O(# links)
# rounds to finish update (till loop-freeness / optimality)	Count to infinity	1
# address-value pairs in storage	O(N * # neighbors)	O(N * # neighbors)
Conclusions	**Less overhead & storage** **Does not scale well**	**Loop-freeness, fast convergence** **Large overhead**

- Luckily, advantages are in different zones
- Small scale and simplicity – DV
- Large scale and optimality – LS

Hierarchical Routing

- Divide and conquer
- Group destinations into logical localities
- Break routing into two phases
  - Between groups
    - Groups may not be directly connected
  - Inside a group



- For both phases, use some other strategy
- Can be generalized into more levels
- Application in Ad Hoc routing

Internet Context

- Interior and Exterior
  - Interior: within a single network (ISP)
  - Hierarchy may be used simply for scalability
  - Exterior: between different ISPs
  - Hierarchy must be used to isolate policies

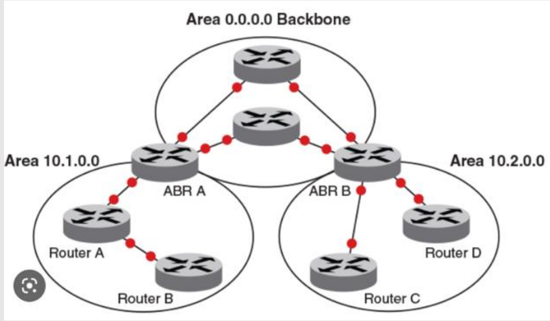
- Interior
  - OSPF itself accommodates areas
  - Scalability measure

- Exterior
  - Between multiple “Autonomous Systems”

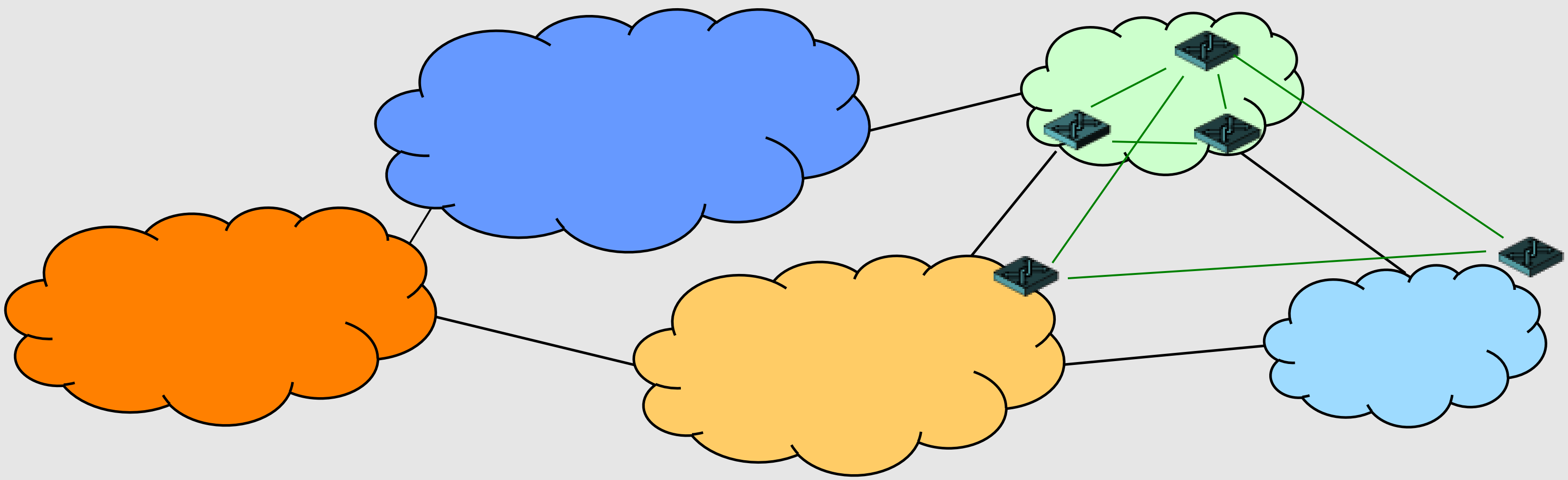
Autonomous Systems

- Autonomous System: a region of the Internet that is administered by a single entity and that has a unified routing policy
  - “Domain” – but not operationally equivalent to DNS domains

- Each autonomous system is assigned an Autonomous System Number (ASN)
  - NCSU’s campus network (AS11442)
  - BellSouth Business Systems (AS5002)
- AS numbers between 1 and 65,535 (two bytes)
  - Numbers greater than 64,511 are “private”
- AS numbers may be requested:
  - Global asn – from your regional internet registry
  - Private asn – from your upstream ISP
  - Ultimately maintained by IANA

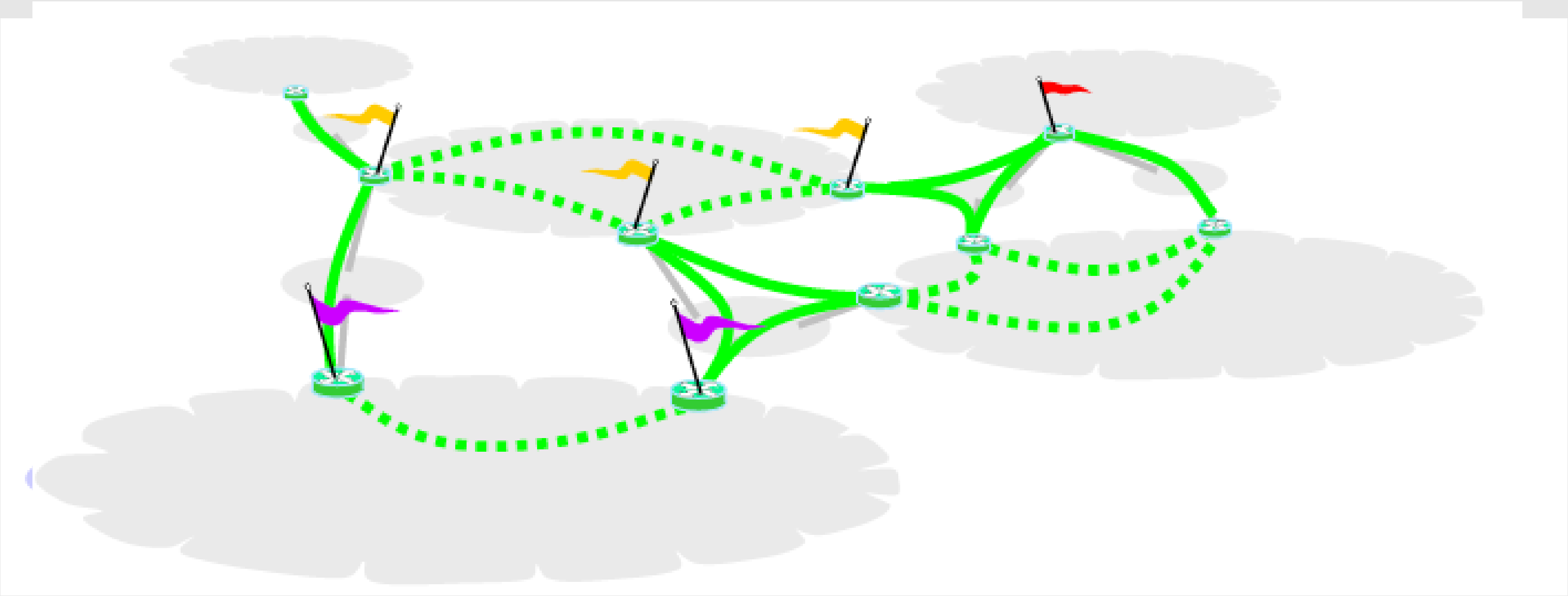


Interdomain / Intradomain Routing



- Routing protocols for intradomain routing are called interior gateway protocols (IGP – Interior Gateway Protocol)
  - Objective: shortest path
- Routing protocols for interdomain routing are called exterior gateway protocols (EGP - Exterior Gateway Protocol)
  - Objective: satisfy policy of the AS (delay, or dollars)

Two kinds of BGP



- e-BGP: pkt runs between two gateways of different ASes
- i-BGP: pkt runs within two gateways of same AS

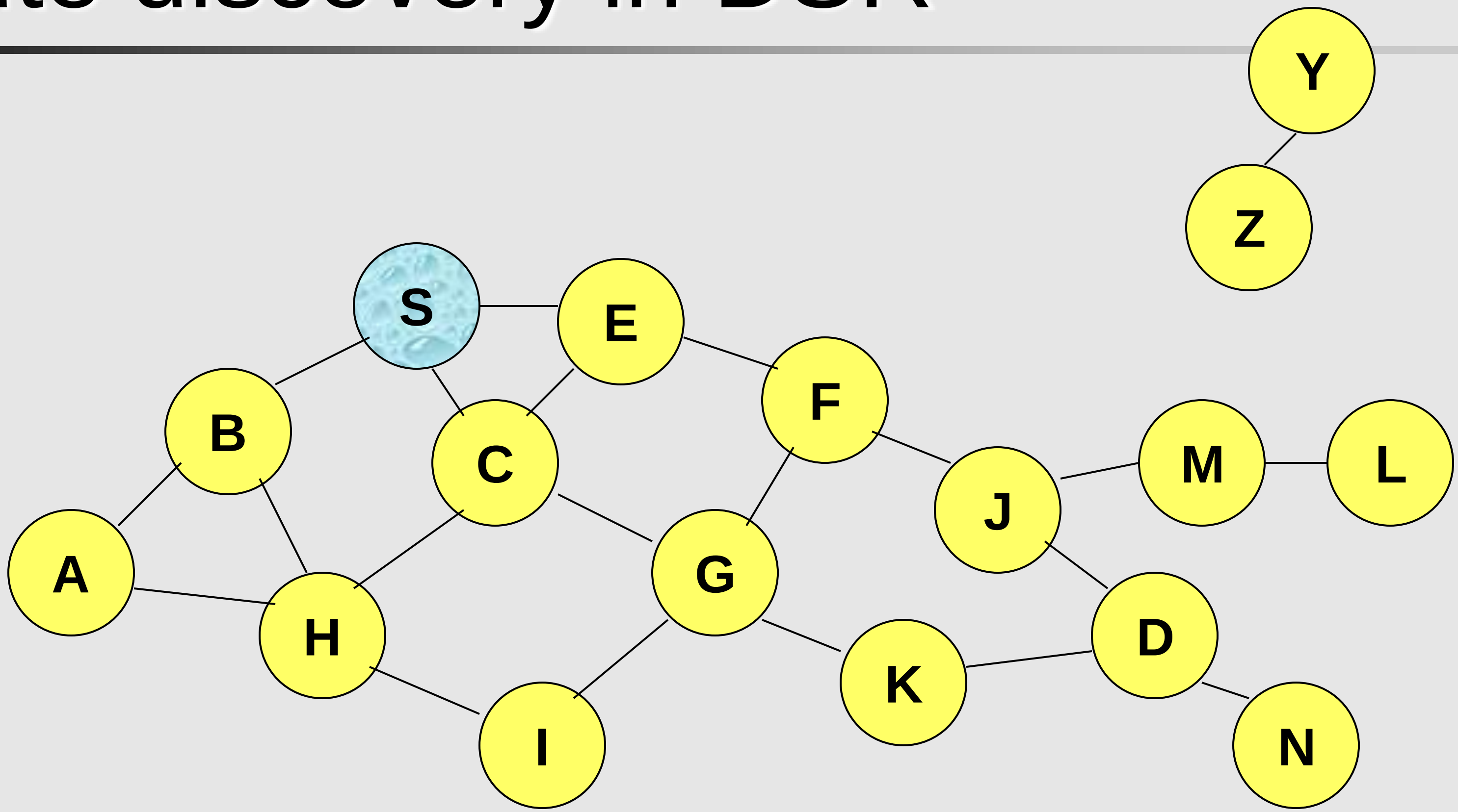
On-demand Routing

- No pre-determined paths – make forwarding decision at the time packet arrives
  - Need to constitute consistent and correct path decisions
- May save some information about network
  - Neighbors
- May utilize additional information to determine “desirability”
  - Geographic routing

Example – Dynamic Source Routing (DSR) operation

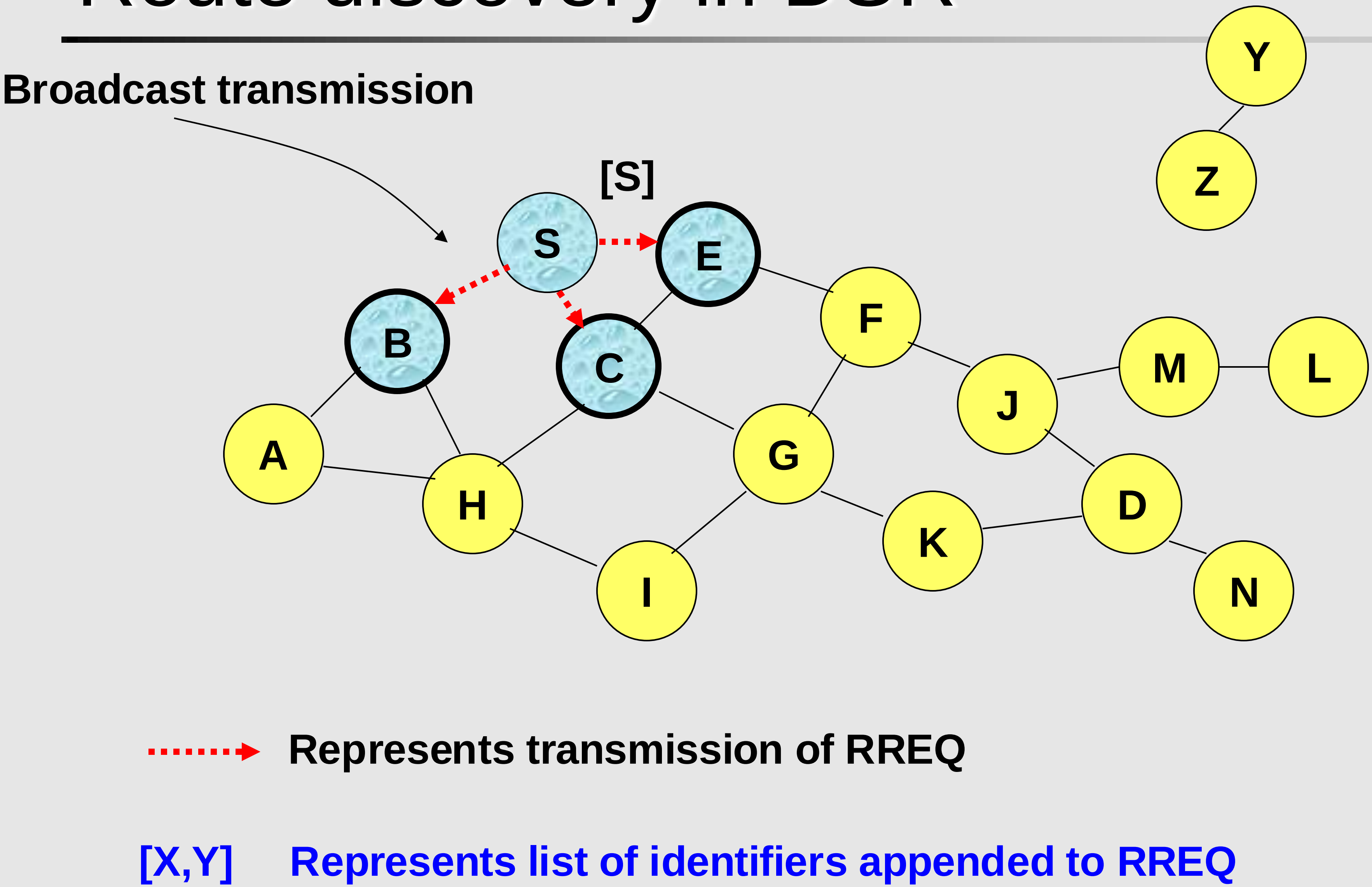
- When S wants to send a packet to D, but does not know a route to D, S initiates a route discovery
- Source node S floods Route Request (RREQ)
- Each node appends own identifier when forwarding RREQ

Route discovery in DSR

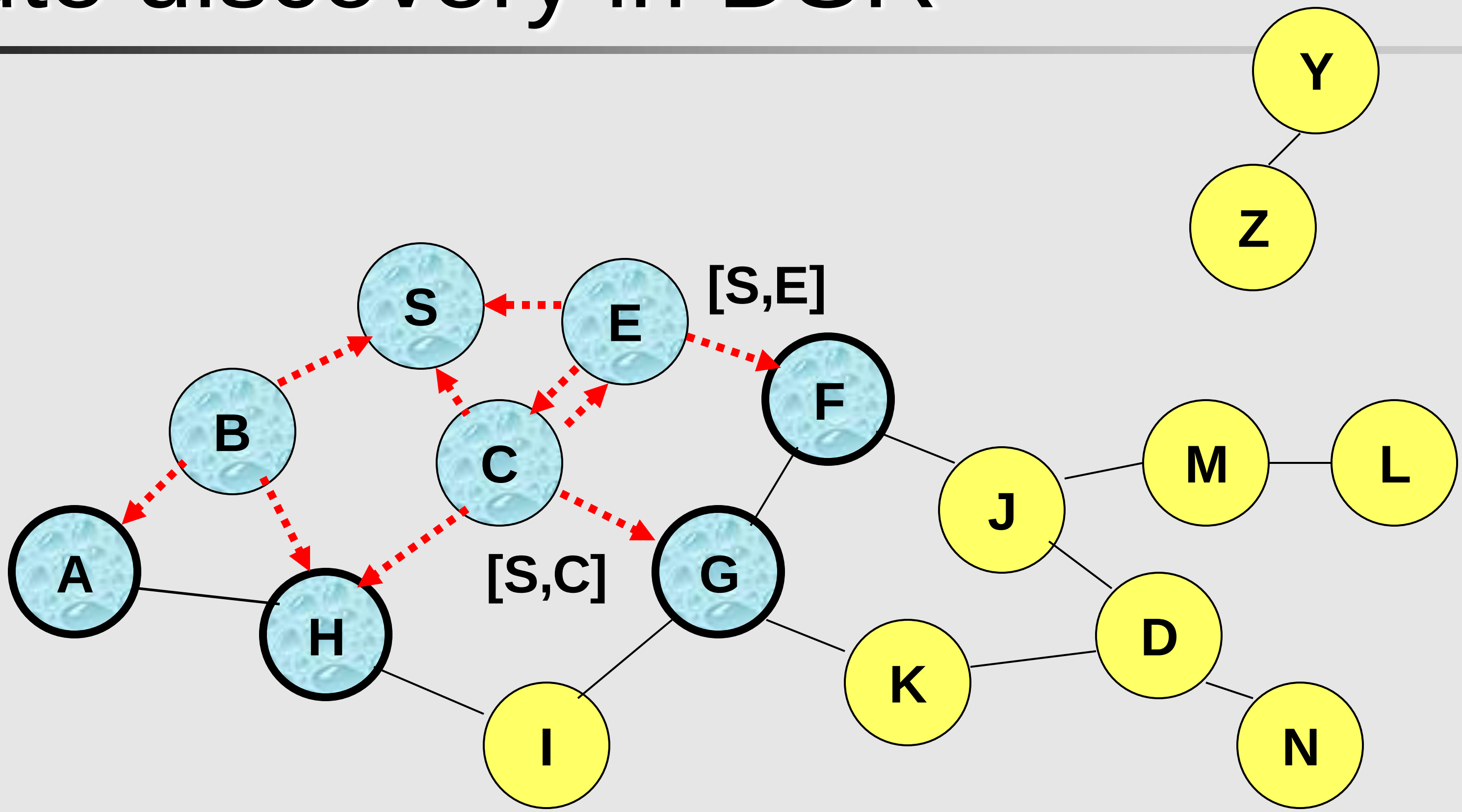


Represents a node that has received RREQ for D from S

Route discovery in DSR

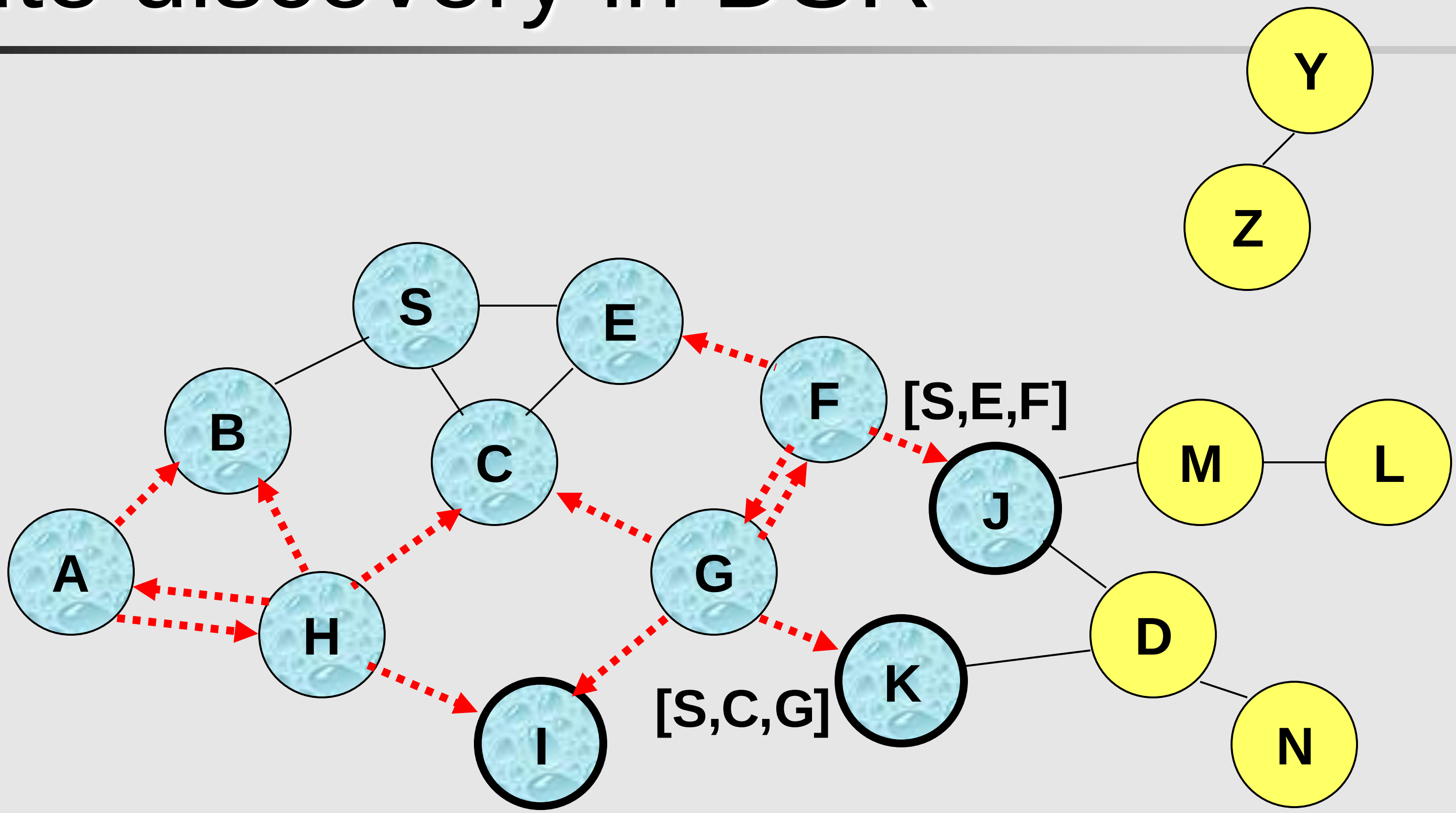


Route discovery in DSR



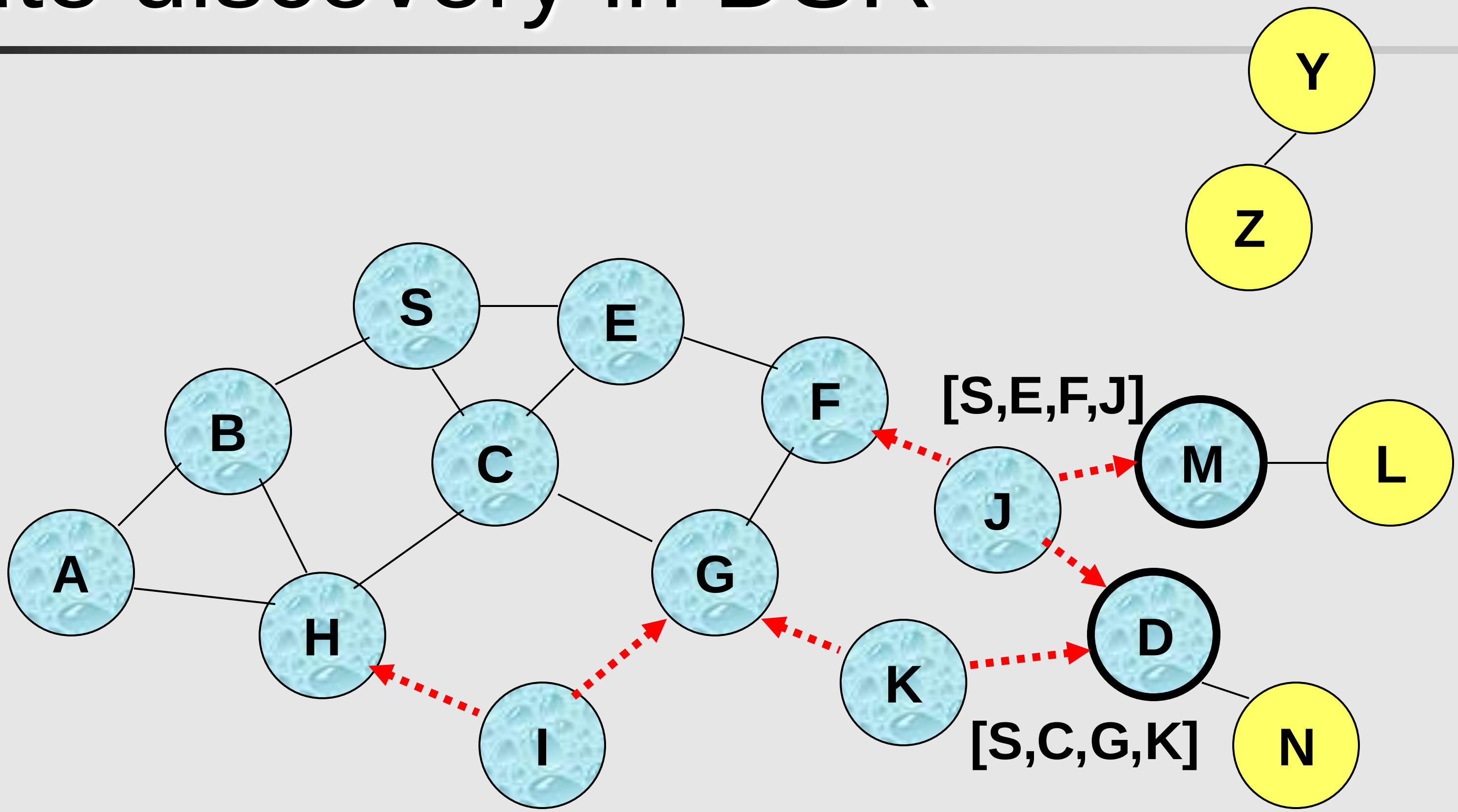
- **Node H receives packet RREQ from two neighbors (B & C): potential for collision**

Route discovery in DSR



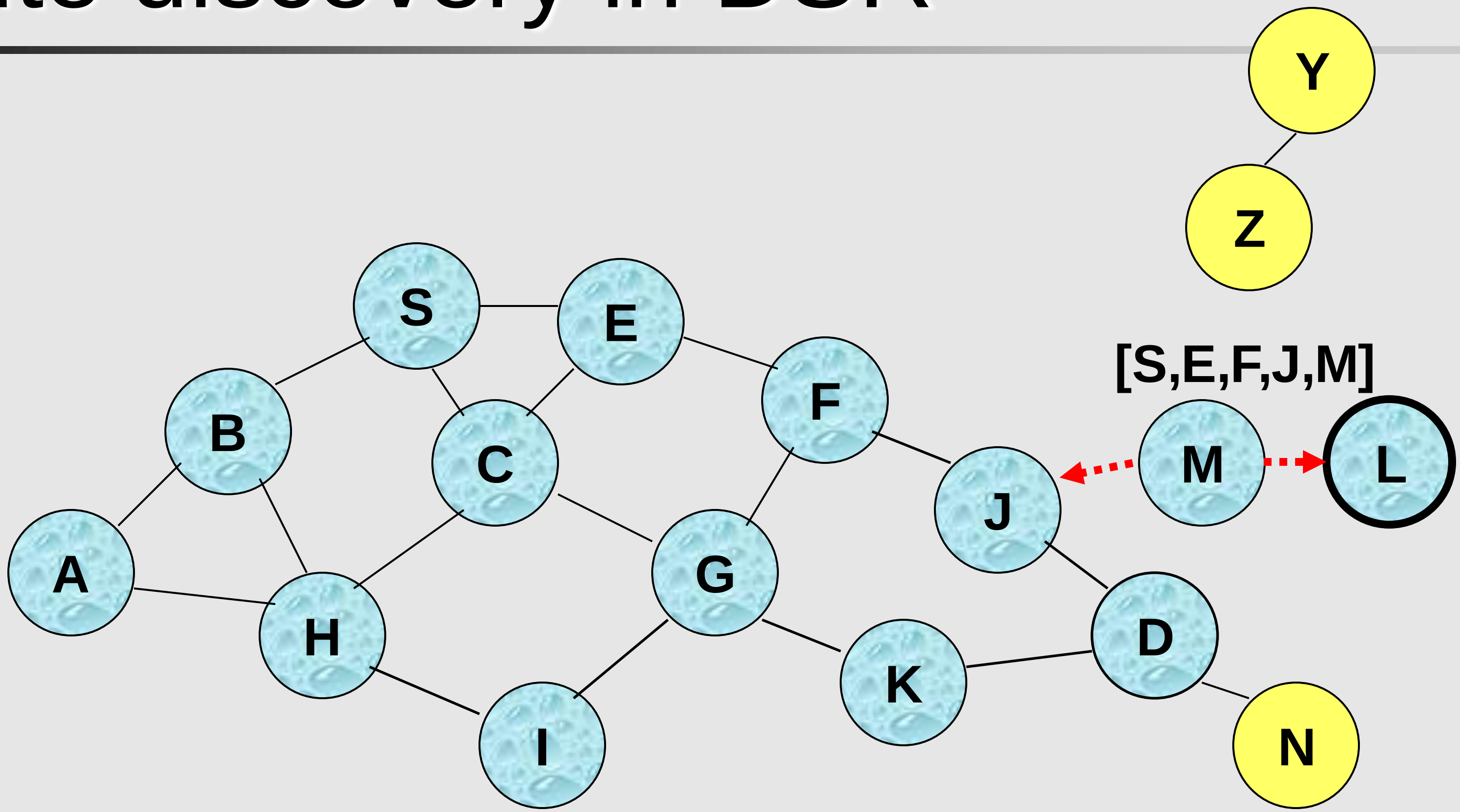
- Node C receives RREQ from G and H, but does not forward it again, because node C has already forwarded RREQ once

Route discovery in DSR



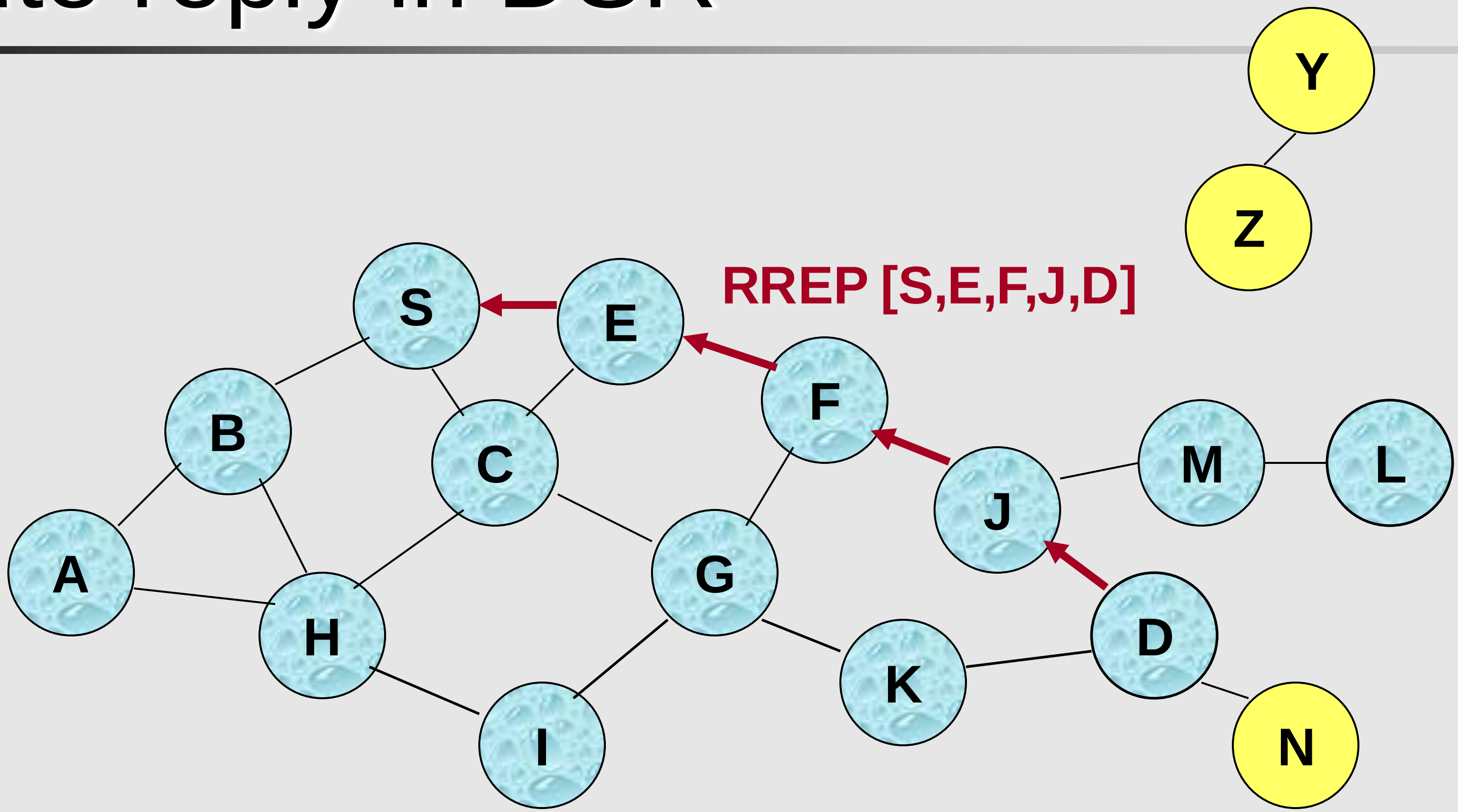
- **Nodes J and K both broadcast RREQ to node D**
- **Since nodes J and K are hidden from each other, their transmissions may collide**

Route discovery in DSR



- **Node D does not forward RREQ, because node D is the intended target of the route discovery**

Route reply in DSR



Represents RREP control message

Route reply in DSR

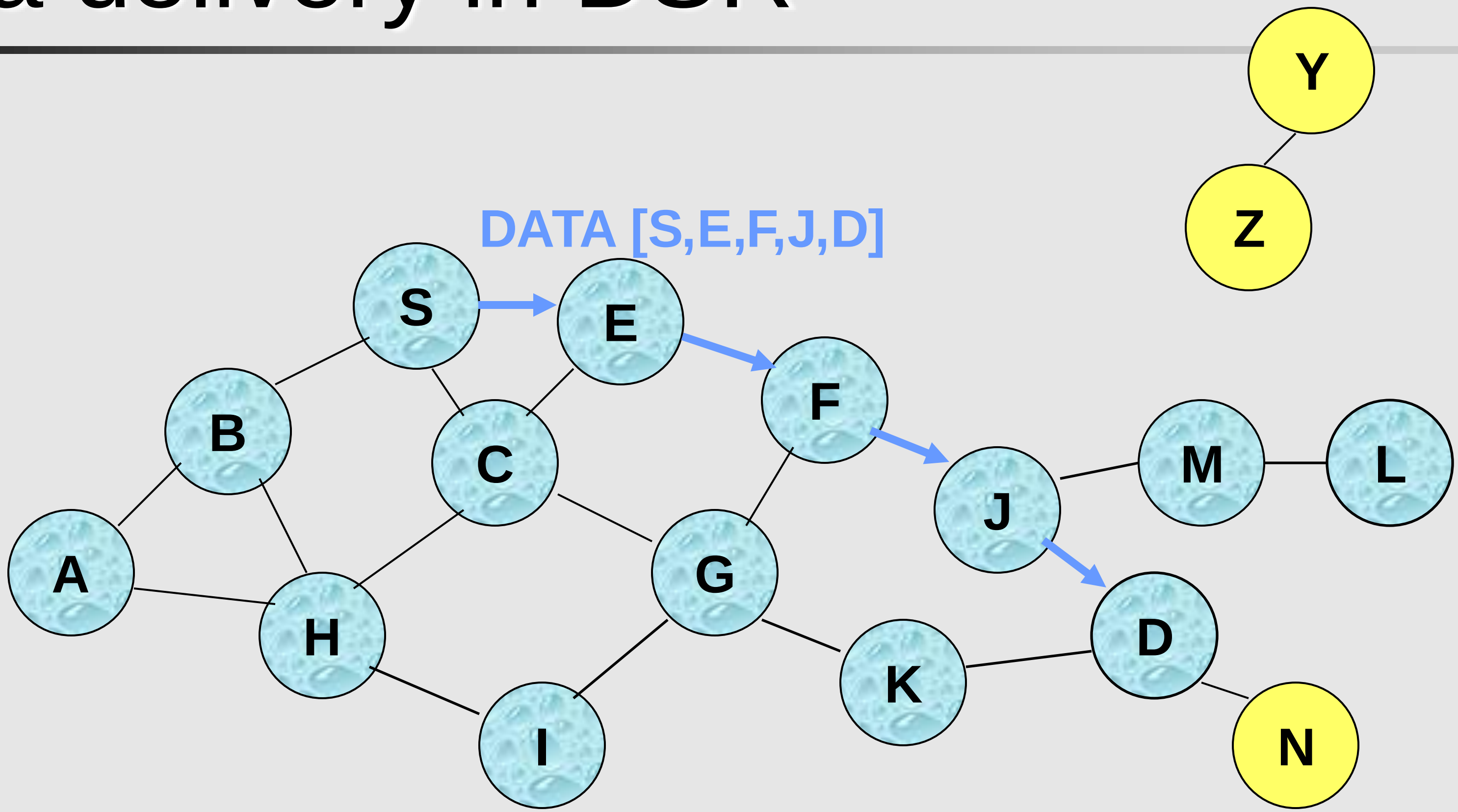
- Reverse route assumes bi-directional links
  - To ensure this, node only forwards RREQ if its link is bi-directional

- If allow unidirectional (asymmetric) links, then may need a route discovery for S from node D
  - Unless node D already knows a route to node S
  - If discover route from D to S, Route Reply is piggybacked on Route Request from D.

Data delivery in DSR

- Node S on receiving RREP, caches the route included in the RREP
- When node S sends a data packet to D, the entire route is included in the packet header
  - hence the name source routing
- Intermediate nodes use the source route included in a packet to determine to whom a packet should be forwarded

Data delivery in DSR



Packet header size grows with route length

Source-Based Routing (SBR)

- Routing table only used at the source node of the flow
- Entire route to each destination is stored in the packet
- Each hop removes the next hop from header and forwards the packets directly
  (without looking up entry in routing table)

- Advantage?
- Disadvantage?

Multi-Path Routing (MPR)

- Multiple routes to the same destination

- Routing table: [dest. nextHop1 nextHop2 …]

Destination	Next hop 1	Next hop 2	Next hop 3
10.1.1.5	10.1.1.1 (0.5)	10.1.1.2 (0.2)	10.1.1.3 (0.3)

- Advantages?

Multi-Path Routing (MPR)

- Multiple routes to the same destination

$$T_r = \max\{\frac{d_1}{R_1}, \frac{d_2}{R_2}, \ldots, \frac{d_n}{R_n}\}$$

$$d_i = \frac{D \cdot R_i}{\sum_{i \in P_s} R_i} \ (1 \leq i \leq n).$$

the assigned traffic for each route should be proportional to its achievable rate

- Advantages?

ns-3 Routing

- Magic Command
  - Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
  - Dijkstra Shortest Path Algorithm
  - Link State Routing (OSPF)

- Not support multi-path routing
- Support source-based routing
  - nix-vector routing
- traceroute

Summary

- Network layer allows separate physical networks to cooperate
- Context may or may not be utilized to minimize forwarding effort
- Dual concerns of forwarding and routing
- Various fundamental approaches to routing
  - Real strategies can be designed from combination of these approaches