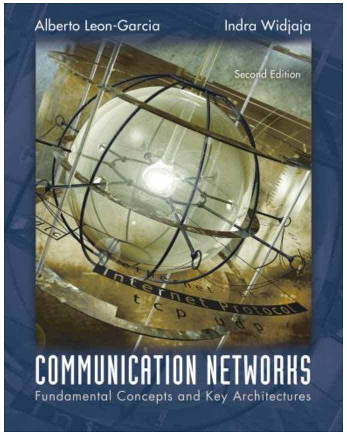
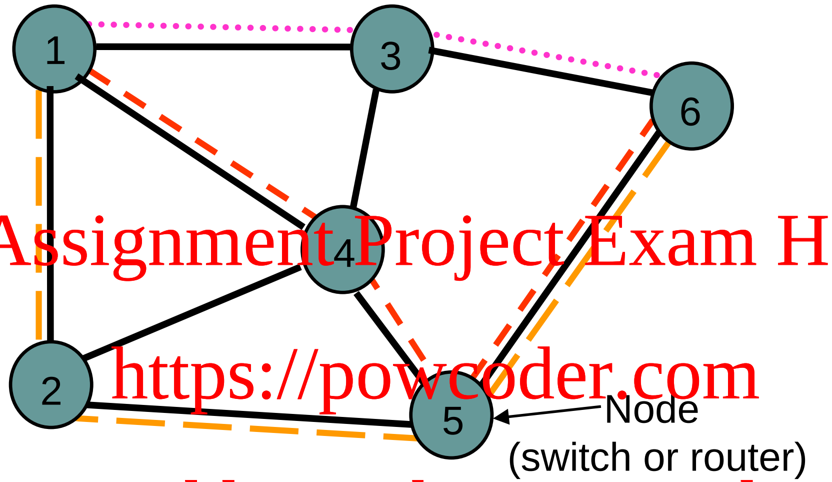
Packet-Switching Networks

Routing in Packet Networks



Routing in Packet Networks



- Three possible (loopfree) routes from 1 to 6:
  - 1-3-6, 1-4-5-6, 1-2-5-6
- Which is "best"?
  - Min delay? Min hop? Max bandwidth? Min cost? Max reliability?

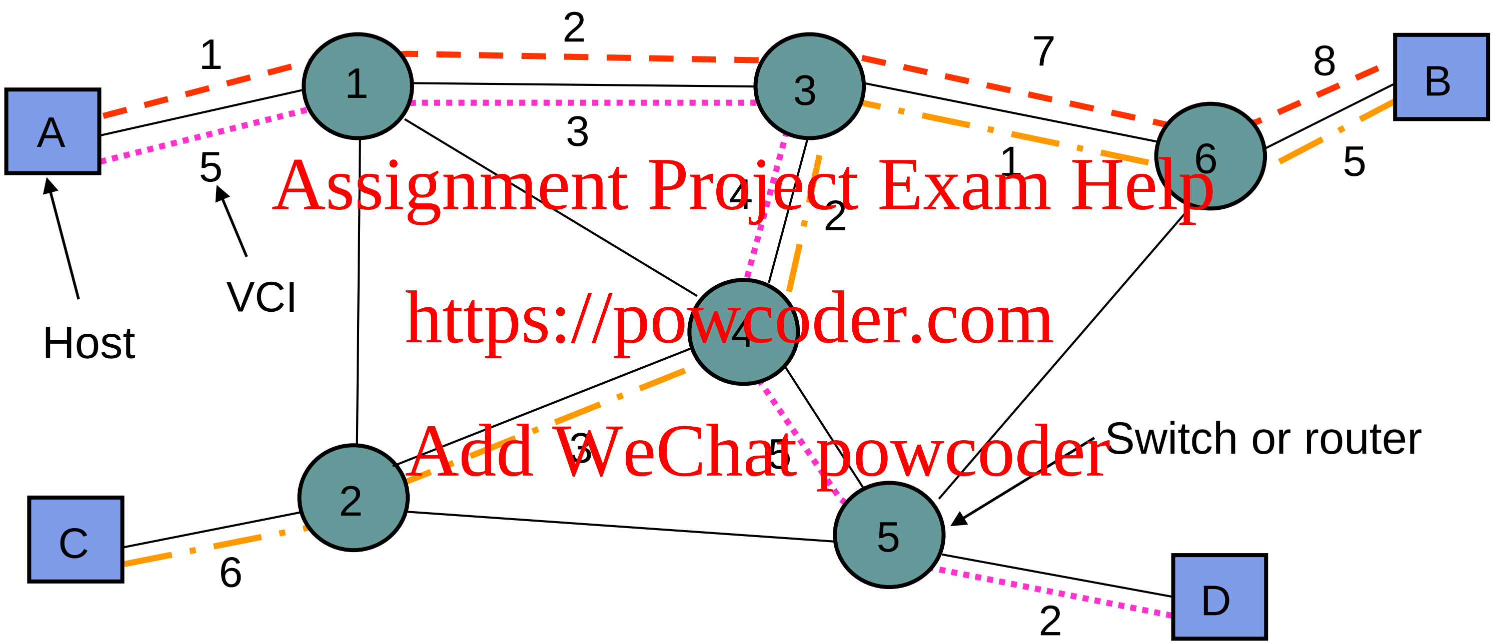
Creating the Routing Tables

- Need information on state of links
  - Link up/down; congested; delay or other metrics
- Need to distribute link state information using a routing protocol
  - What information is exchanged? How often?
  - Exchange with neighbors; Broadcast or flood
- Need to compute routes based on information
  - Single metric; multiple metrics
  - Single route; alternate routes

Routing Algorithm Requirements

- Responsiveness to changes
  - Topology or bandwidth changes, congestion
  - Rapid convergence of routers to consistent set of routes
  - Freedom from persistent loops
- Optimality
  - Resource utilization, path length
- Robustness
  - Continues working under high load, congestion, faults, equipment failures, incorrect implementations
- Simplicity
  - Efficient software implementation, reasonable processing load

Routing in Virtual-Circuit Packet Networks



- Route determined during connection setup
- Tables in switches implement forwarding that realizes selected route

Routing Tables in VC Packet Networks

Node 1

Incoming Node	Incoming VCI	Outgoing Node	Outgoing VCI
A	1	3	2
A	5	3	3
3	2	A	1
3	3	A	5

Node 3

Incoming Node	Incoming VCI	Outgoing Node	Outgoing VCI
1	2	6	7
1	3	4	4
4	2	6	1
6	7	1	2
6	1	4	2
4	4	1	3

Node 6

Incoming Node	Incoming VCI	Outgoing Node	Outgoing VCI
3	7	B	8
3	1	B	5
B	5	3	1
B	8	3	7

Node 4

Incoming Node	Incoming VCI	Outgoing Node	Outgoing VCI
2	3	3	2
3	4	5	5
3	2	2	3
5	5	3	4

Node 2

Incoming Node	Incoming VCI	Outgoing Node	Outgoing VCI
C	6	4	3
4	3	C	6

Node 5

Incoming Node	Incoming VCI	Outgoing Node	Outgoing VCI
4	5	D	2
D	2	4	5

- Example: VCI from A to D
  - From $\boxed{\text{A \& VCI 5}} \rightarrow \boxed{\text{3 \& VCI 3}} \rightarrow \boxed{\text{4 \& VCI 4}}$
  - $\rightarrow \boxed{\text{5 \& VCI 5}} \rightarrow \boxed{\text{D \& VCI 2}}$

Routing Tables in Datagram Packet Networks

Node 1

Destination	Next node
2	2
3	3
4	4
5	2
6	3

Node 3

Destination	Next node
1	1
2	4
4	4
5	6
6	6

Node 6

Destination	Next node
1	3
2	5
3	3
4	3
5	5

Node 4

Destination	Next node
1	1
2	2
3	3
5	5
6	3

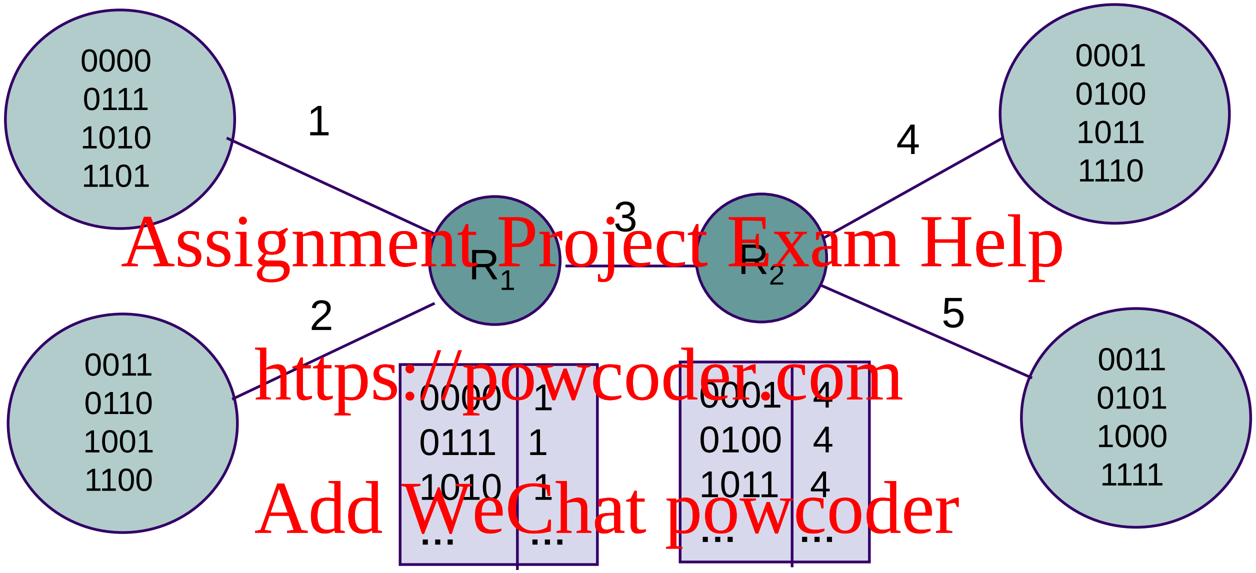
Node 2

Destination	Next node
1	1
3	1
4	4
5	5
6	5

Node 5

Destination	Next node
1	4
2	2
3	4
4	4
6	6

Non-Hierarchical Addresses and Routing



- No relationship between addresses & routing proximity
- Routing tables require 16 entries each

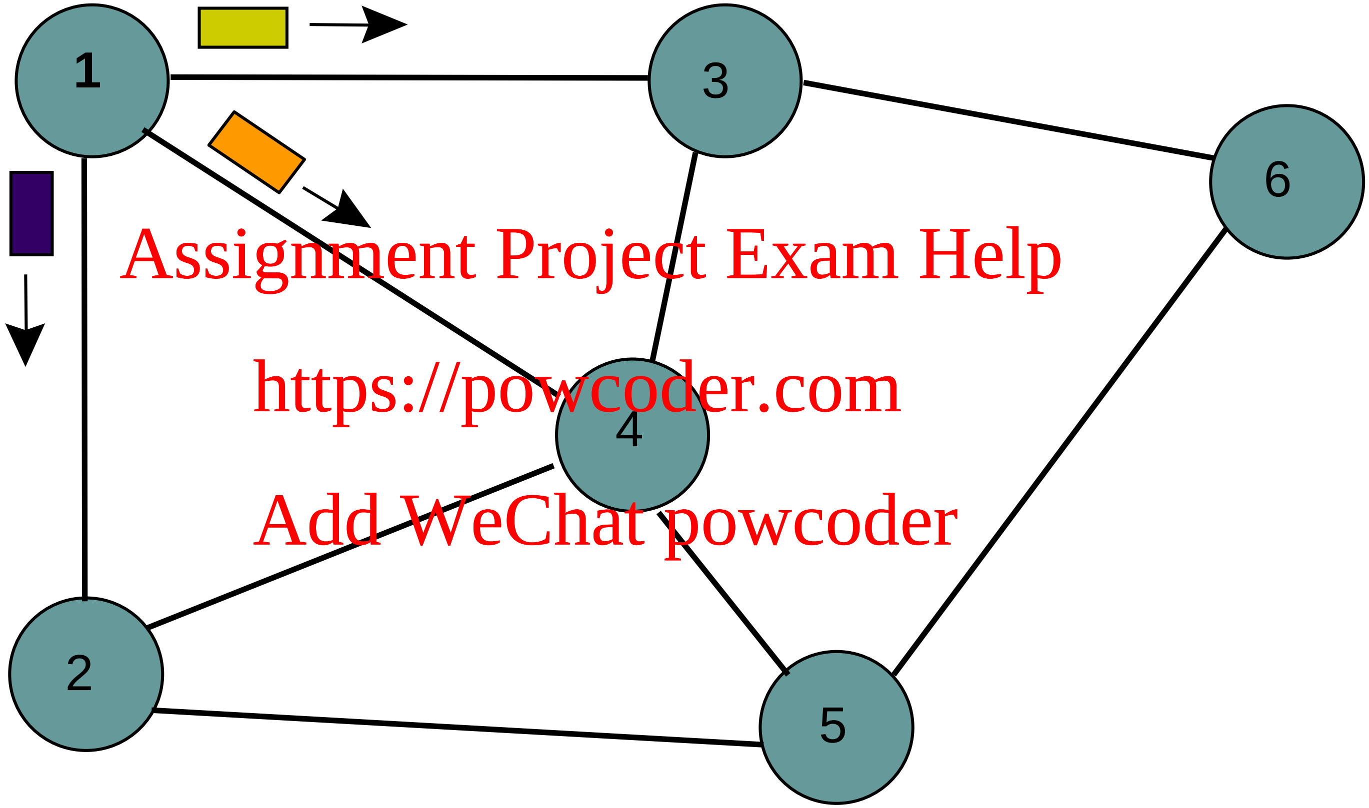
Flooding

Send a packet to all nodes in a network

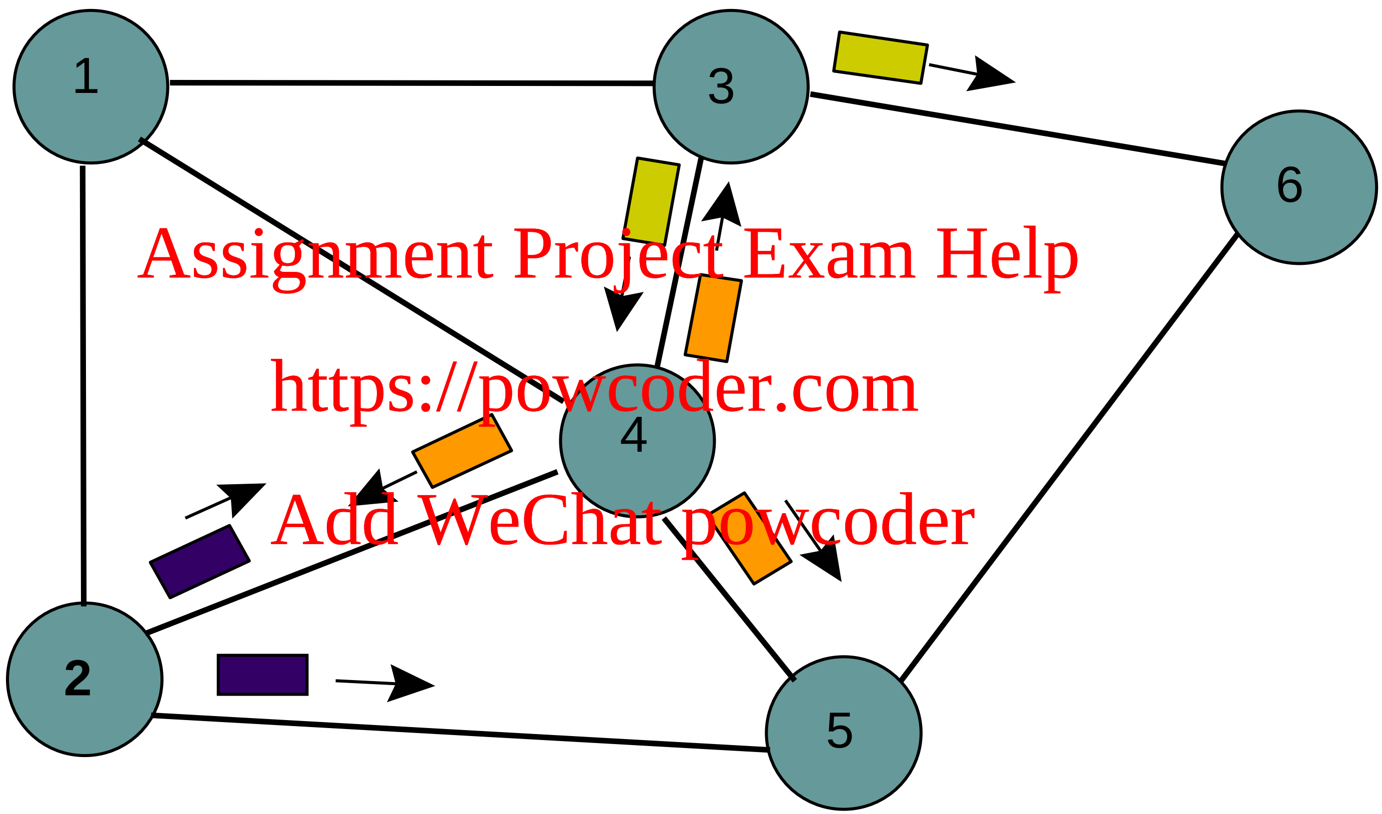
- No routing tables available
- Need to broadcast packet to all nodes (e.g. to propagate link state information)

Approach

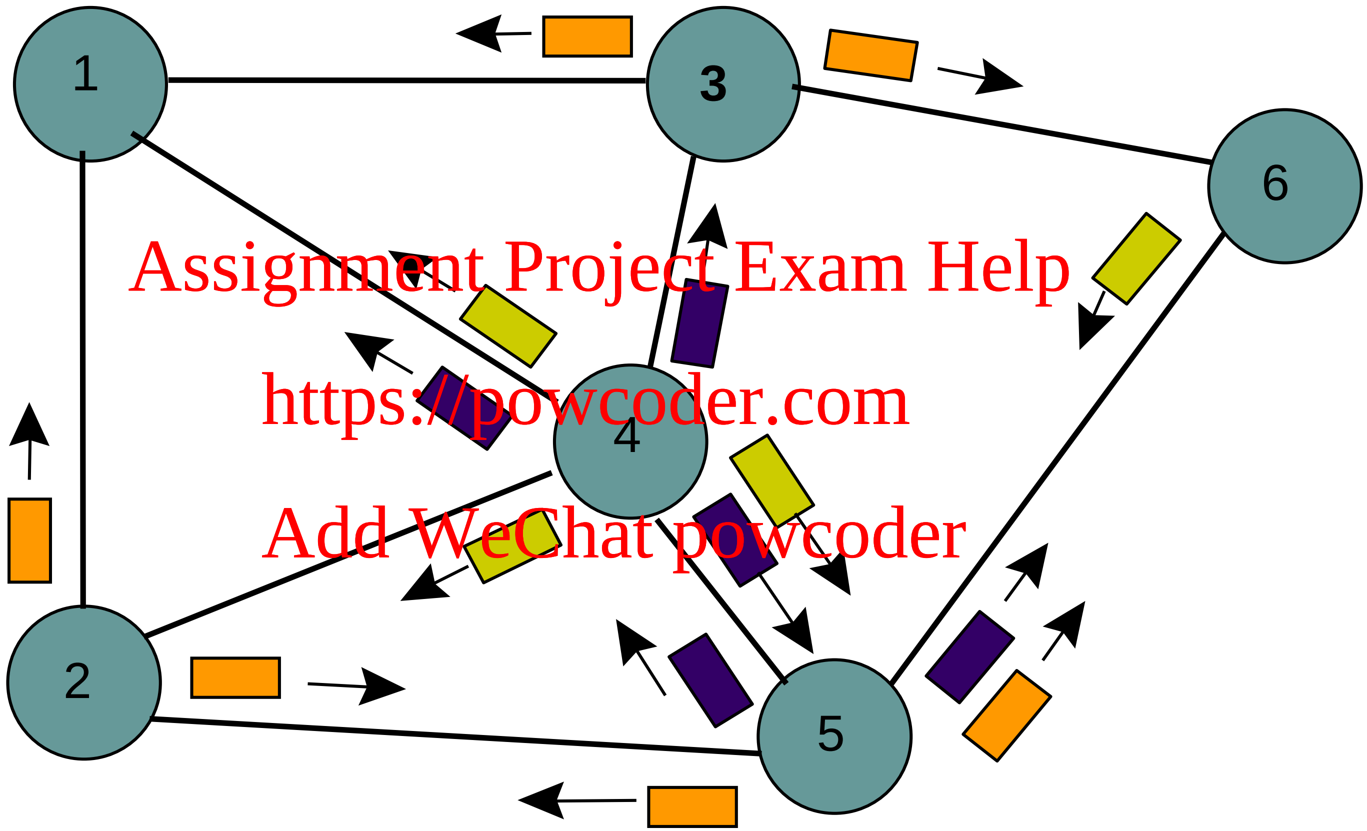
- Send packet on all ports except one where it arrived
- Exponential growth in packet transmissions



Flooding is initiated from Node 1: Hop 1 transmissions



Flooding is initiated from Node 1: Hop 2 transmissions



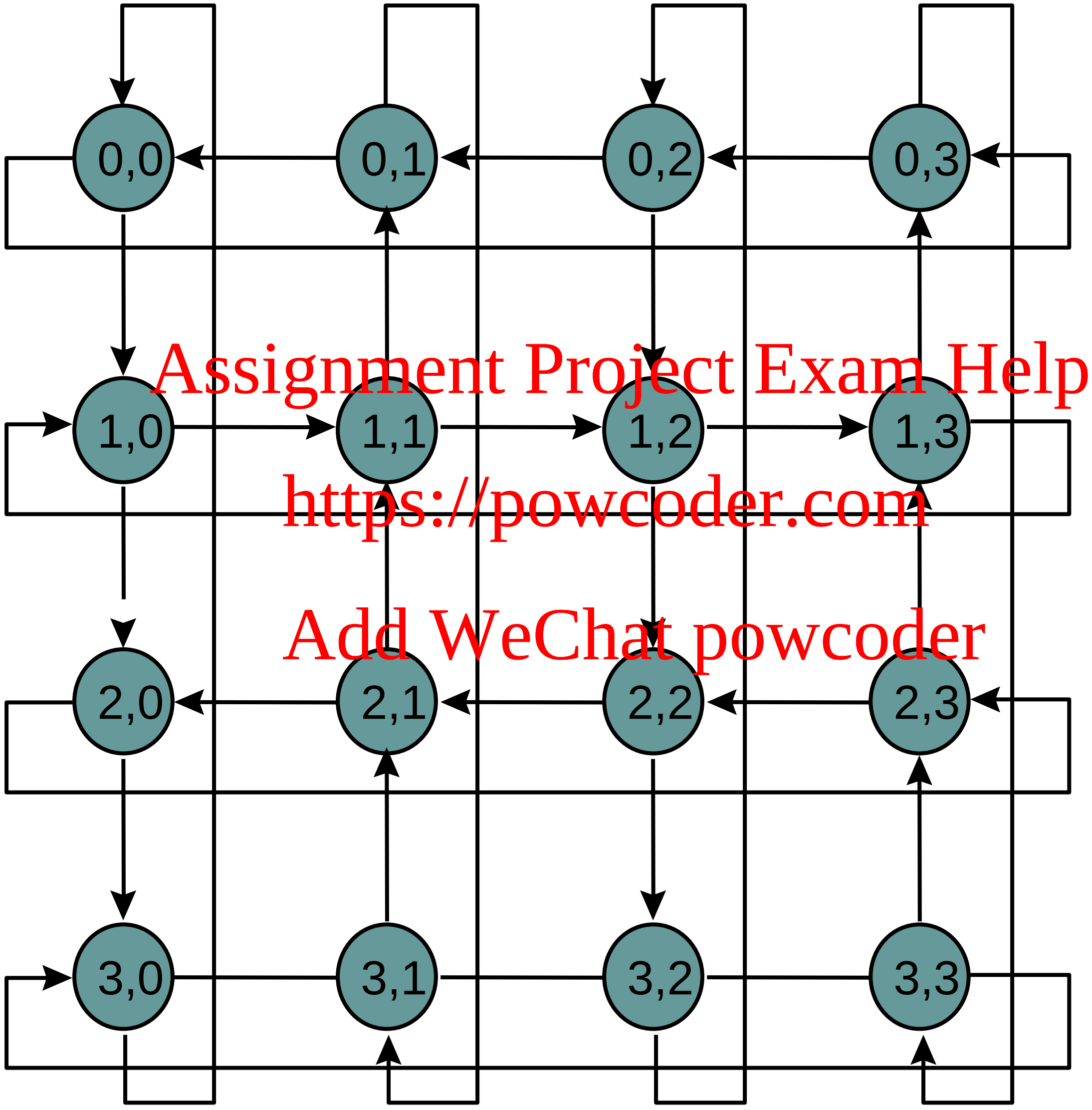
Flooding is initiated from Node 1: Hop 3 transmissions

Limited Flooding

- Time-to-Live field in each packet limits number of hops to certain diameter
- Each switch adds its ID before flooding; discards repeats
- Source puts sequence number in each packet; switches records source address and sequence number and discards repeats

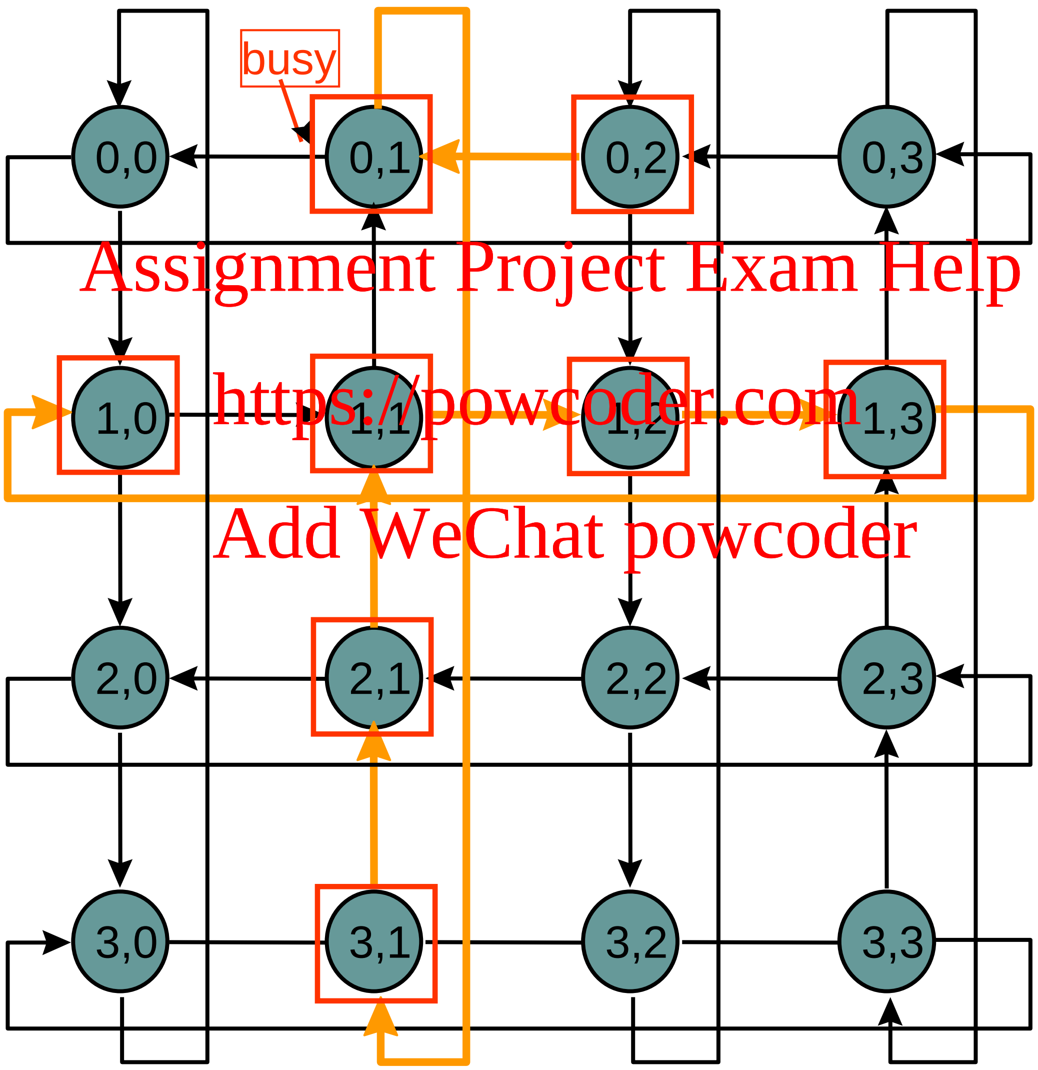
Deflection Routing

- Network nodes forward packets to preferred port
- If preferred port busy, deflect packet to another port
- Works well with regular topologies
  - Manhattan street network
  - Rectangular array of nodes
  - Nodes designated (i,j)
  - Rows alternate as one-way streets
  - Columns alternate as one-way avenues
- Bufferless operation is possible
  - Proposed for optical packet networks
  - All-optical buffering currently not viable



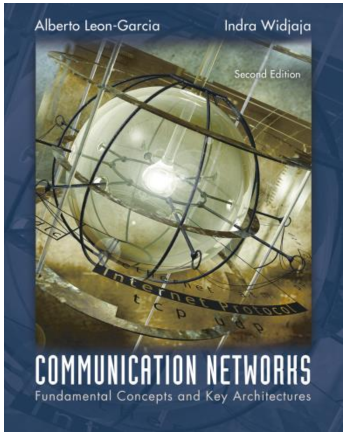
Tunnel from last column to first column or vice versa

Example: Node (0,2)→(1,0)



Packet-Switching Networks

Shortest Path Routing



Shortest Paths & Routing

- Many possible paths connect any given source and to any given destination
- Routing involves the selection of the path to be used to accomplish a given transfer
- Typically it is possible to attach a cost or distance to a link connecting two nodes
- Routing can then be posed as a shortest path problem

Routing Metrics

Means for measuring desirability of a path

- Path Length = sum of costs or distances
- Possible metrics
  - Hop count: rough measure of resources used
  - Reliability: link availability; BER
  - Delay: sum of delays along path; complex & dynamic
  - Bandwidth: “available capacity” in a path
  - Load: Link & router utilization along path
  - Cost: $$$

Shortest Path Approaches

Distance Vector Protocols

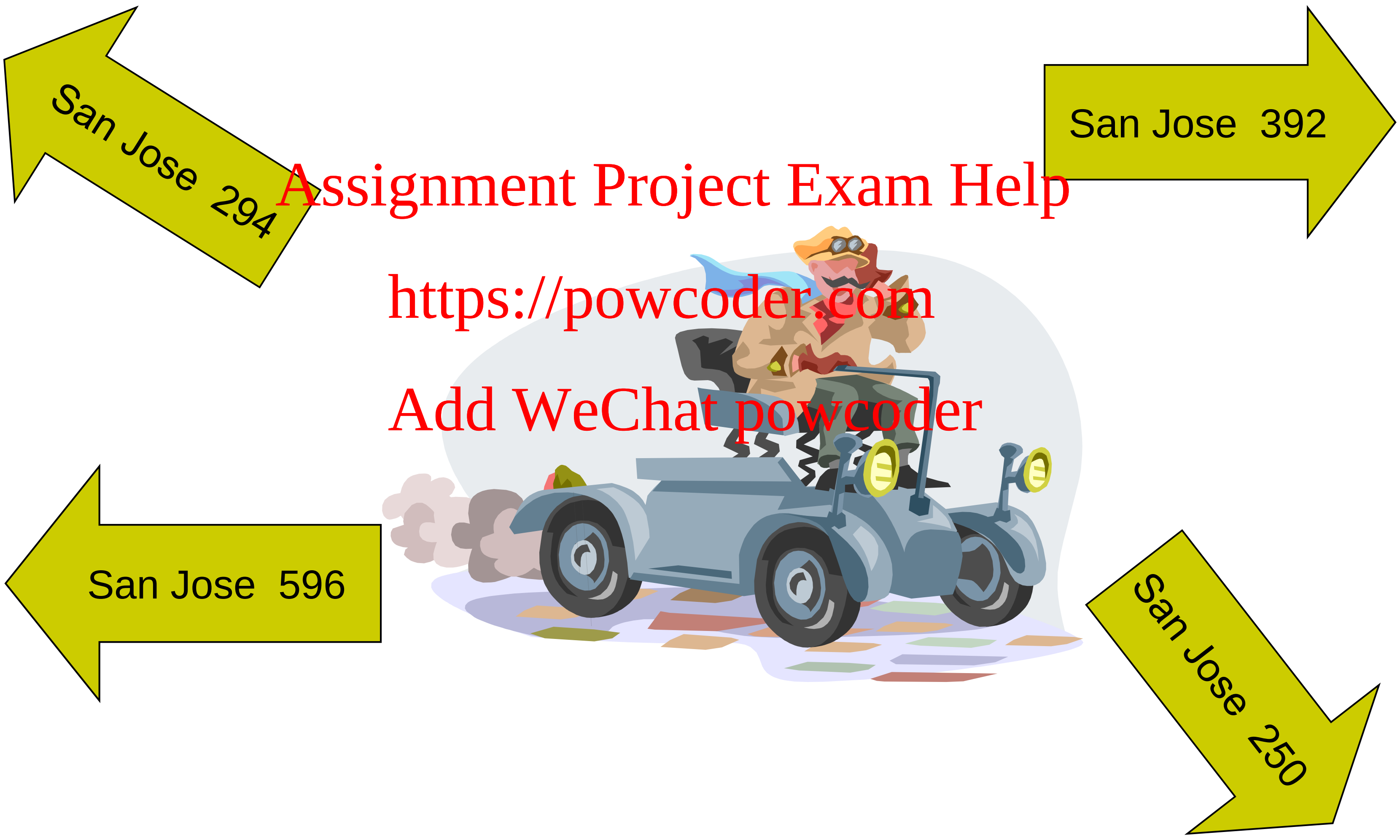
- Neighbors exchange list of distances to destinations
- Best next-hop determined for each destination
- Ford-Fulkerson (distributed) shortest path algorithm

Link State Protocols

- Link state information flooded to all routers
- Routers have complete topology information
- Shortest path (& hence next hop) calculated
- Dijkstra (centralized) shortest path algorithm

Distance Vector

Do you know the way to San Jose?



Distance Vector

Local Signpost

- Direction
- Distance

Routing Table

For each destination list:

- Next Node
- Distance

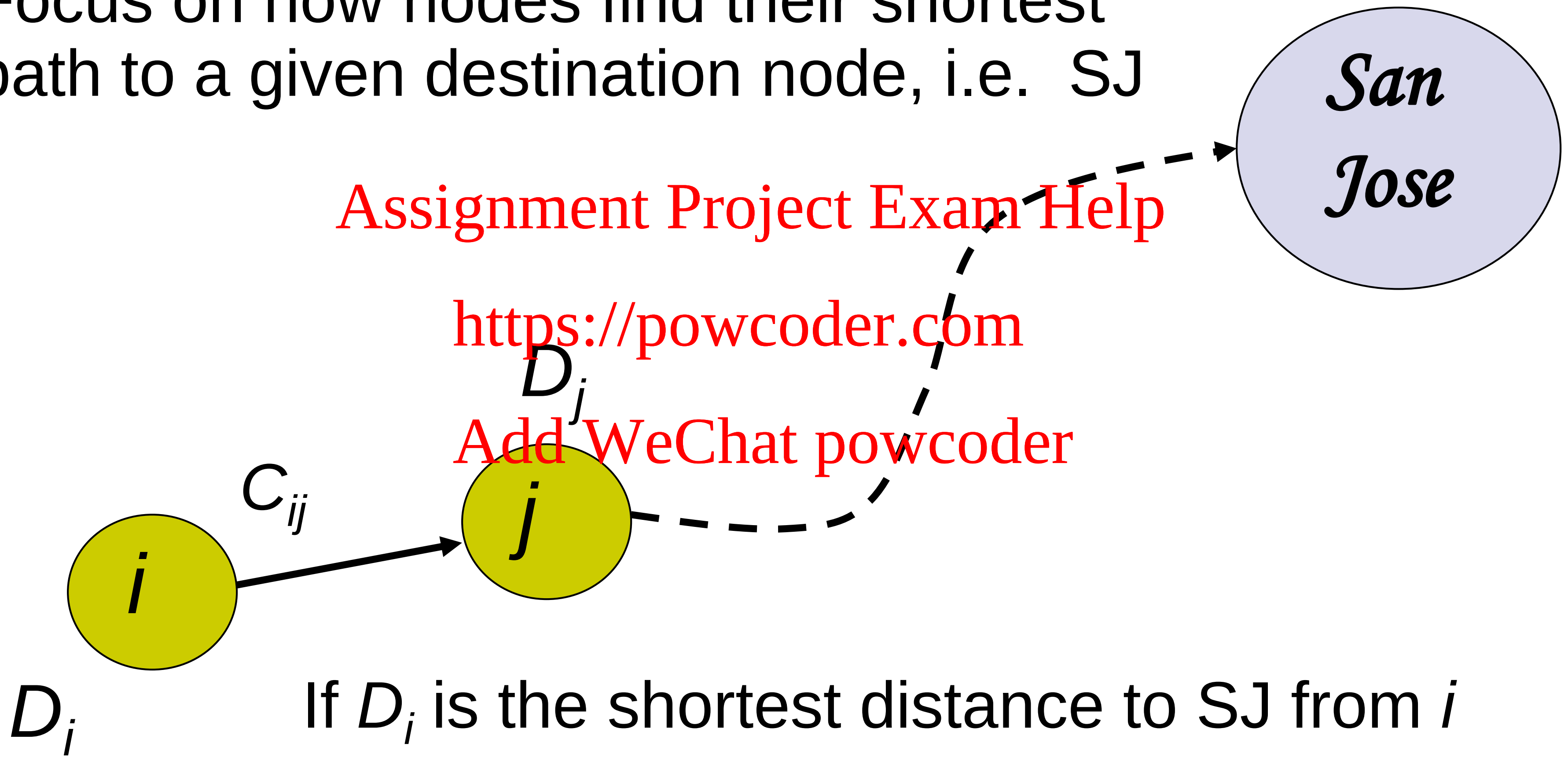
dest	next	dist

Table Synthesis

- Neighbors exchange table entries
- Determine current best next hop
- Inform neighbors
  - Periodically
  - After changes

Shortest Path to SJ

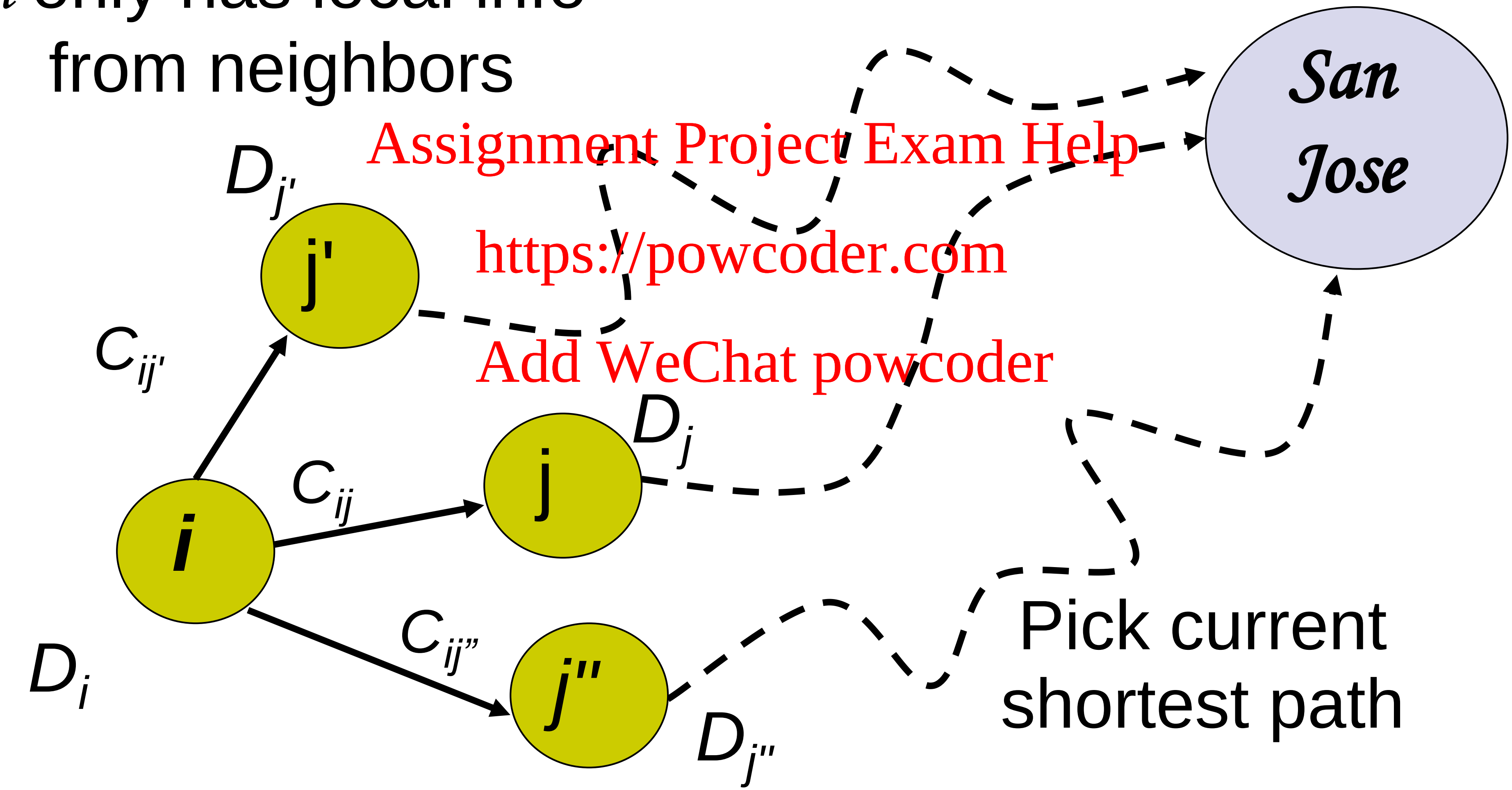
Focus on how nodes find their shortest path to a given destination node, i.e. SJ



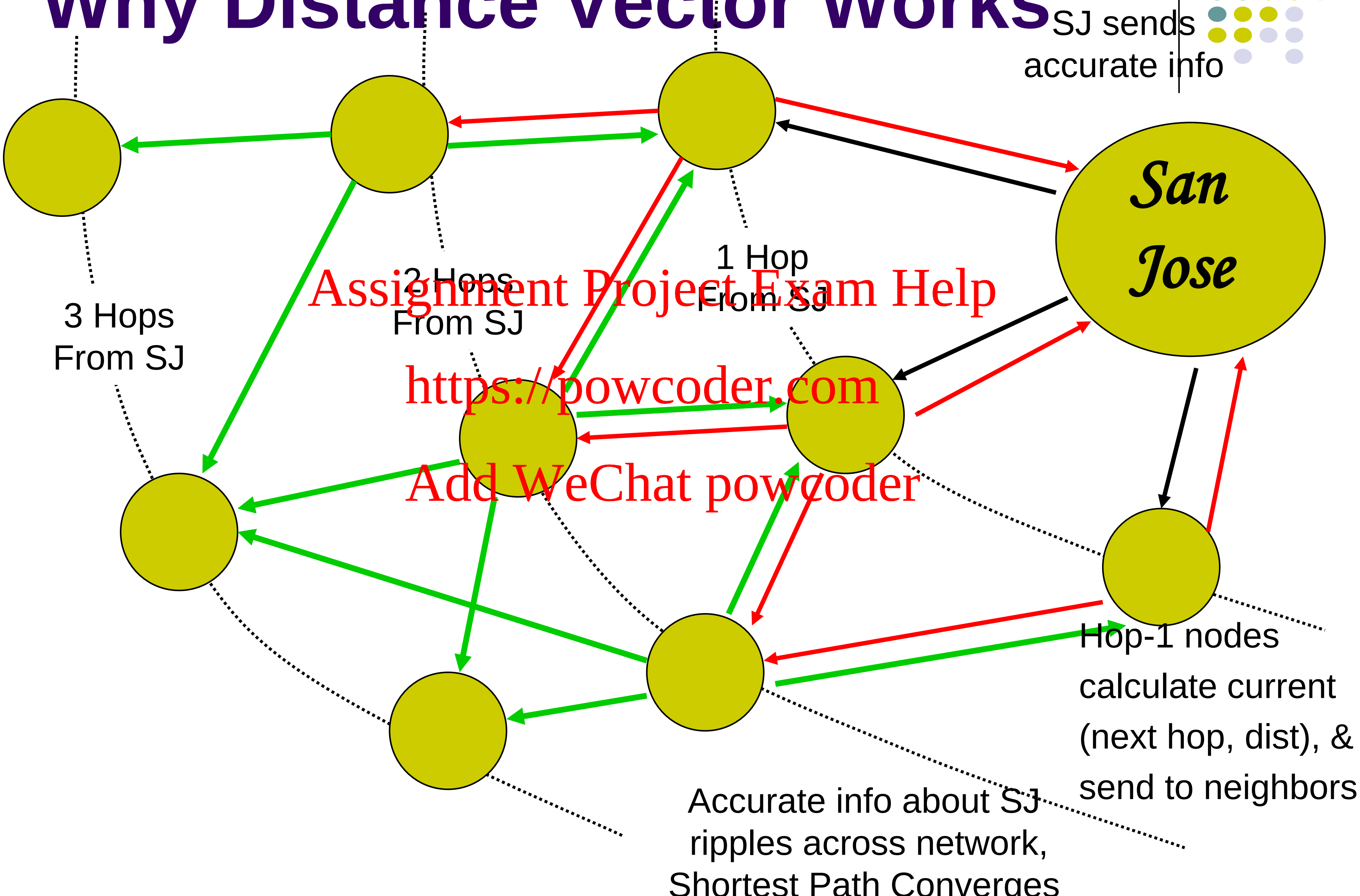
If D_i is the shortest distance to SJ from i and if j is a neighbor on the shortest path, then $D_i = C_{ij} + D_j$

But we don't know the shortest paths

i only has local info from neighbors



Why Distance Vector Works



Bellman-Ford Algorithm

- *Consider computations <u>for one destination</u> d*
- *Initialization*
  - Each node table has 1 row for destination *d*
  - Distance of node *d* to itself is zero: $D_d=0$
  - Distance of other node *j* to *d* is infinite: $D_j=\infty$, for $j \neq d$
  - Next hop node $n_j = -1$ to indicate not yet defined for $j \neq d$
- *Send Step*
  - Send new distance vector to immediate neighbors across local link
- *Receive Step*
  - <u>At node *j*, find the next hop that gives the minimum distance to *d*</u>,
    - $Min_j \{ C_{ij} + D_j \}$
      - Replace old $(n_j, D_j(d))$ by new $(n_j^*, D_j^*(d))$ if new next node or distance
  - Go to send step

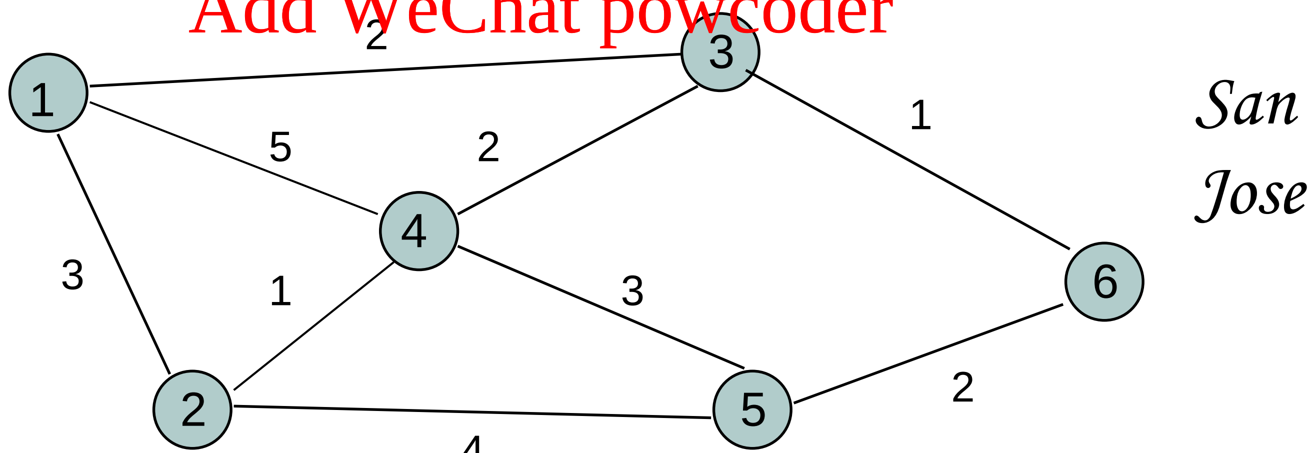
Bellman-Ford Algorithm

- *Now consider parallel computations* <u>*for all destinations*</u> *d*
- *Initialization*
  - Each node has 1 row for each destination *d*
  - Distance of node *d* to itself is zero: $D_d(d)=0$
  - Distance of other node *j* to *d* is infinite: $D_j(d)= \propto$, for $j \neq d$
  - Next node n_j = -1 since not yet defined
- *Send Step*
  - Send new distance vector to immediate neighbors across local link
- *Receive Step*
  - <u>For each destination *d*, find the next hop that gives the minimum distance to *d*</u>,
    - $Min_j \{ C_{ij} + D_j(d) \}$
    - Replace old $(n_j, D_i(d))$ by new $(n_j^*, D_j^*(d))$ if new next node or distance found
  - Go to send step

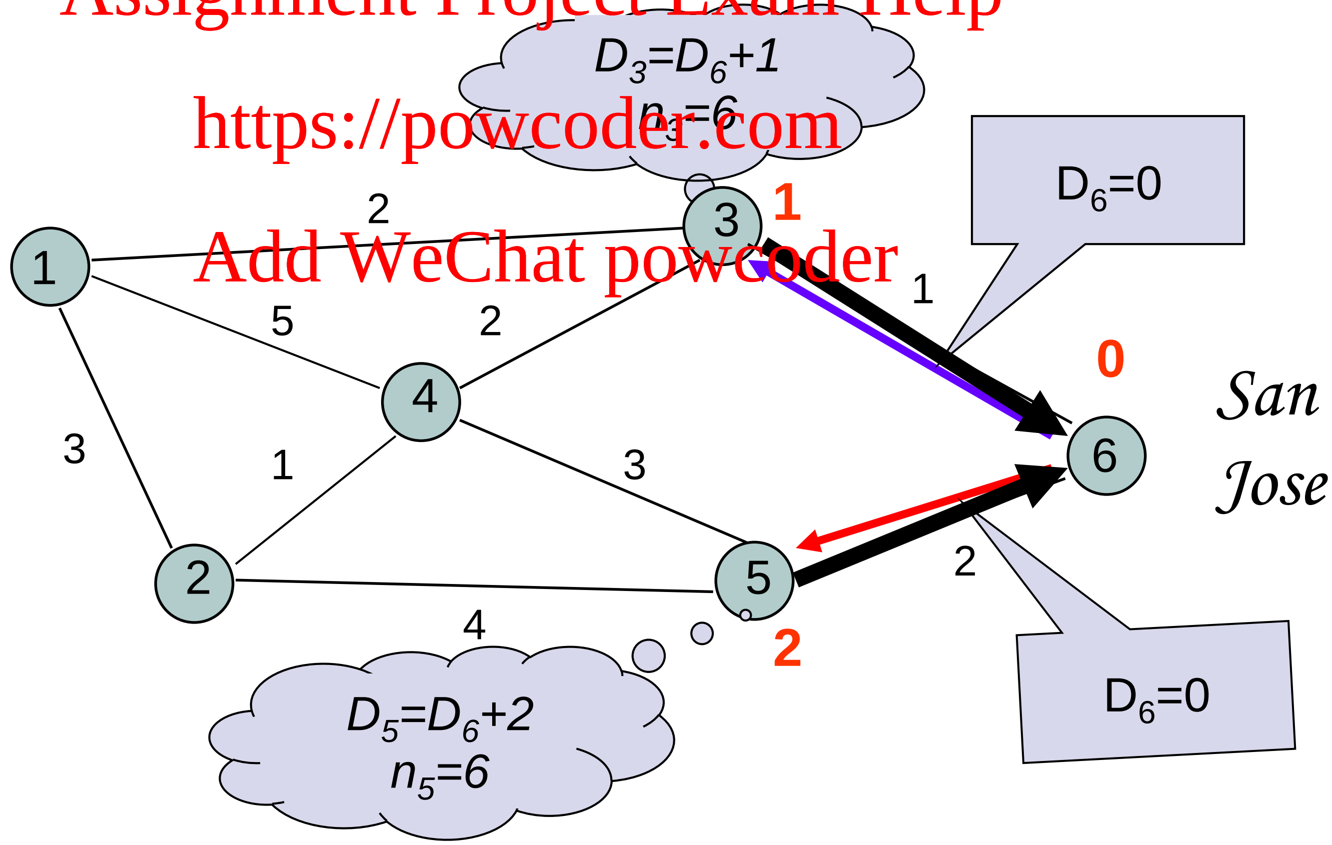
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1					
2					
3					

Table entry
@ node 1
for dest SJ

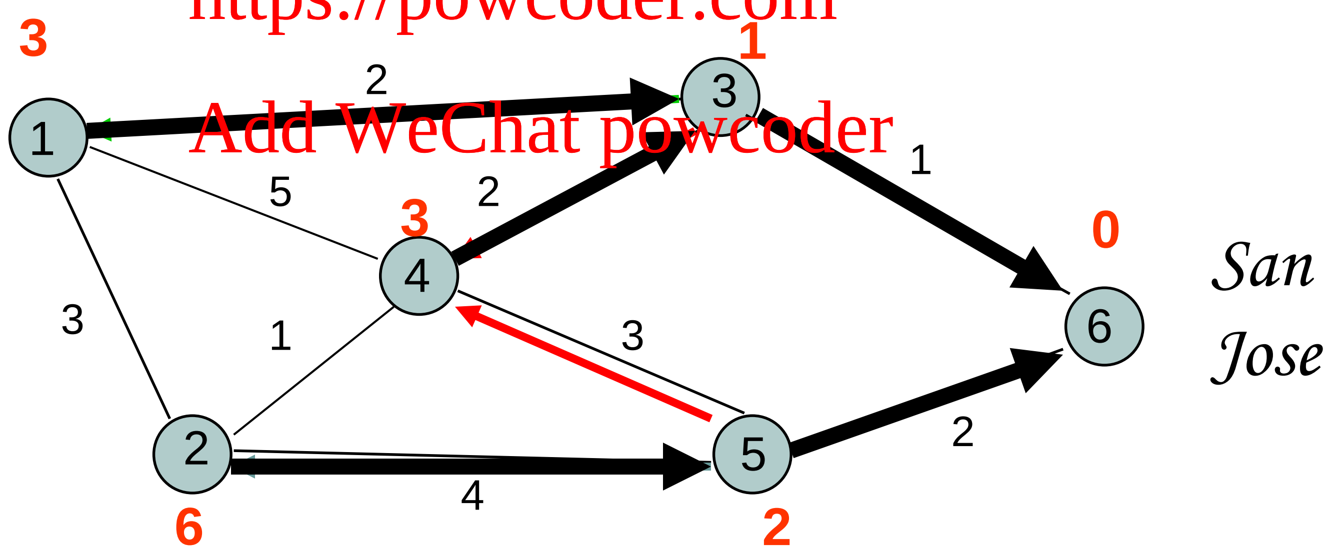
Table entry
@ node 3
for dest SJ



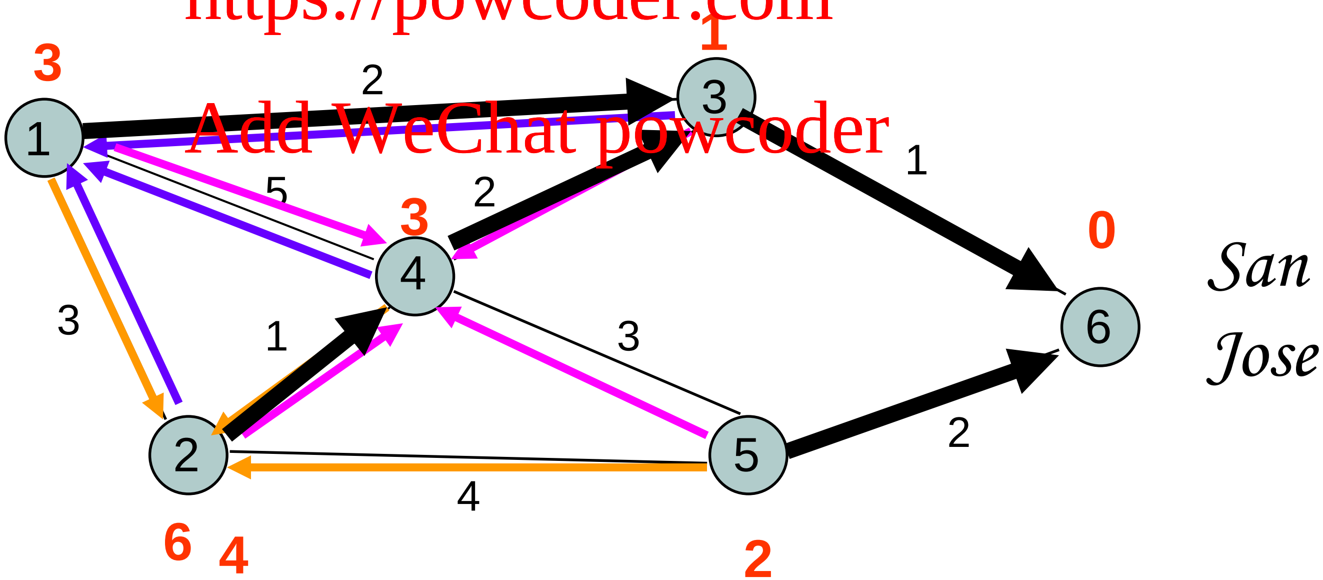
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6,1)	(-1, ∞)	(6,2)
2					
3					



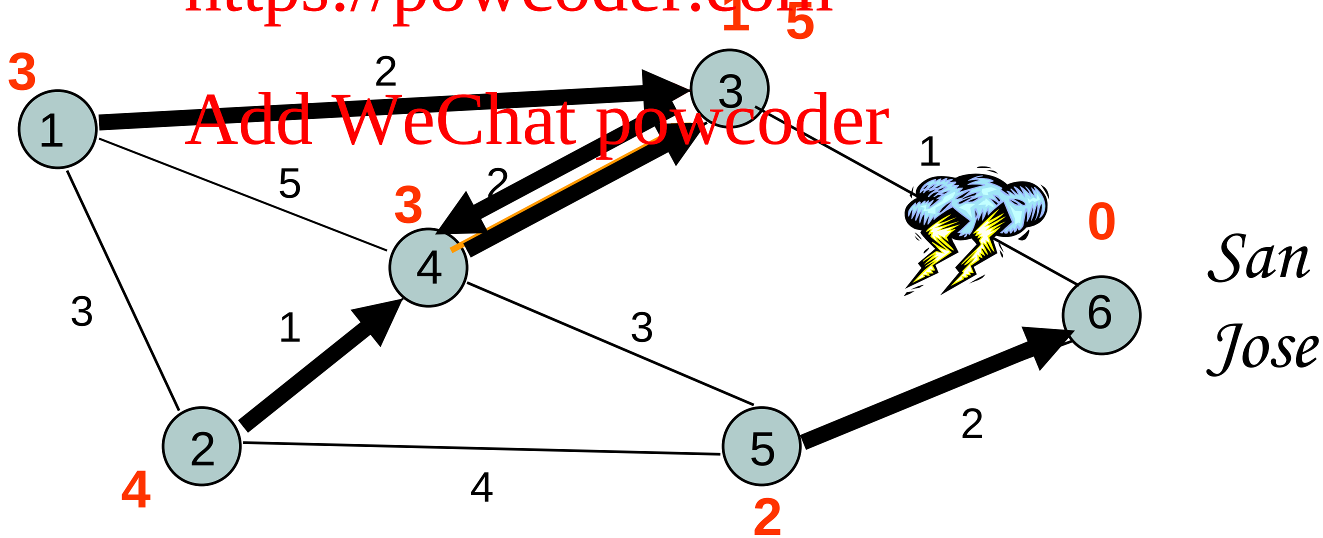
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6,2)
2	(3,3)	(5,6)	(6, 1)	(3,3)	(6,2)
3					



Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6,2)
2	(3,3)	(5,6)	(6, 1)	(3,3)	(6,2)
3	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)

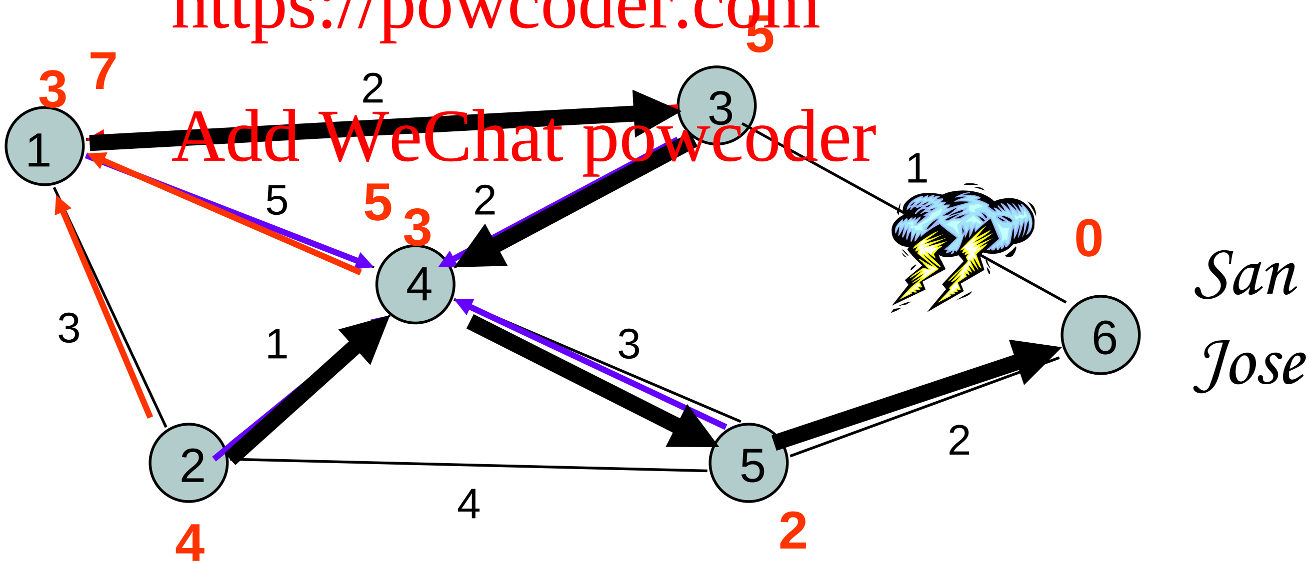


Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2					
3					



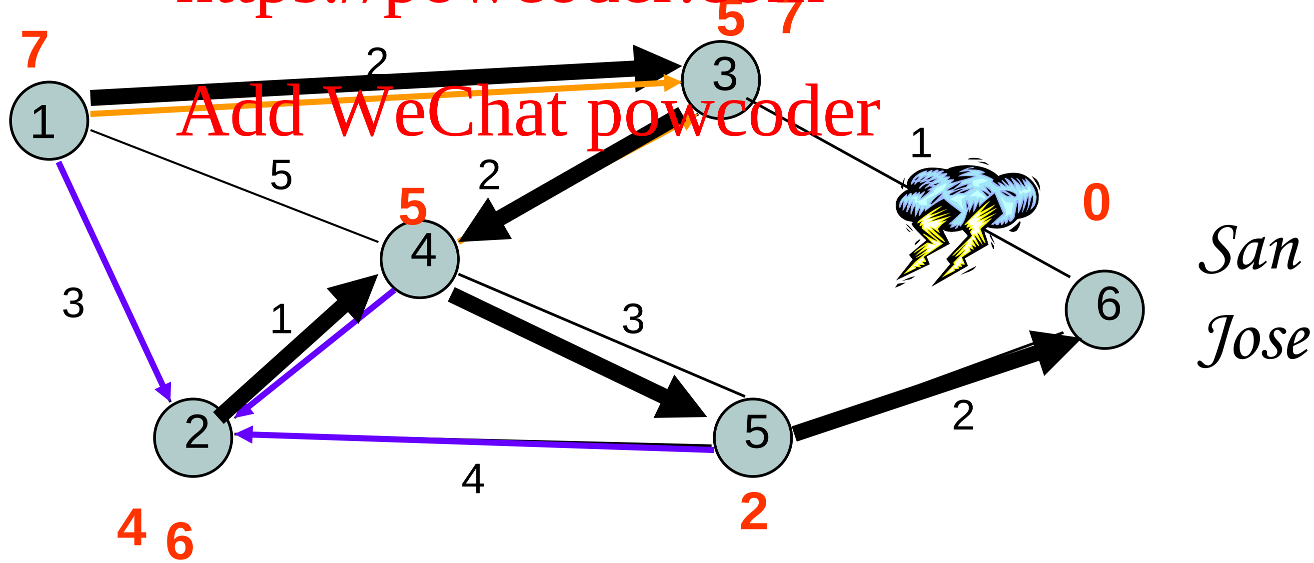
Network disconnected; Loop created between nodes 3 and 4

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4, 5)	(5,5)	(6,2)
3					



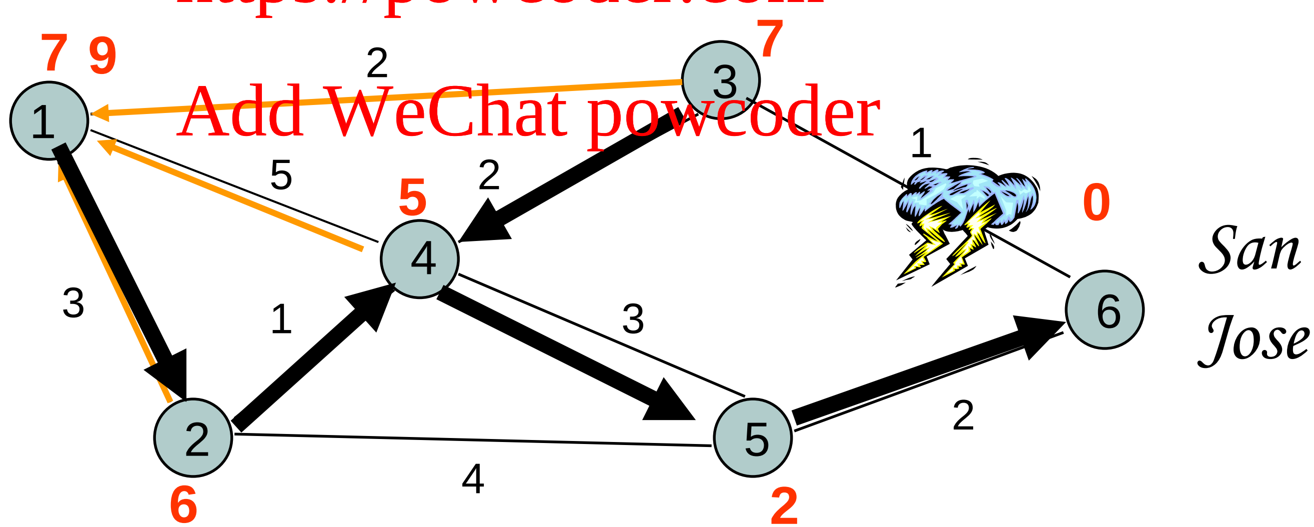
Node 4 could have chosen 2 as next node because of tie

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4, 5)	(5,5)	(6,2)
3	(3,7)	(4,6)	(4, 7)	(5,5)	(6,2)



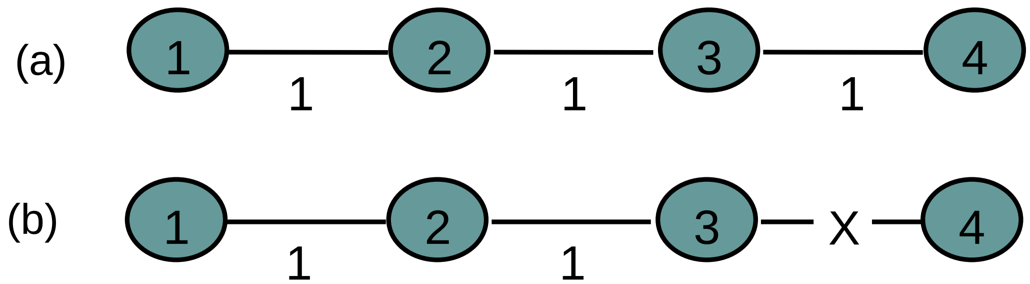
Node 2 could have chosen 5 as next node because of tie

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4, 5)	(2,5)	(6,2)
3	(3,7)	(4,6)	(4, 7)	(5,5)	(6,2)
4	(2,9)	(4,6)	(4, 7)	(5,5)	(6,2)



Node 1 could have chose 3 as next node because of tie

Counting to Infinity Problem



Nodes believe best path is through each other

(Destination is node 4)

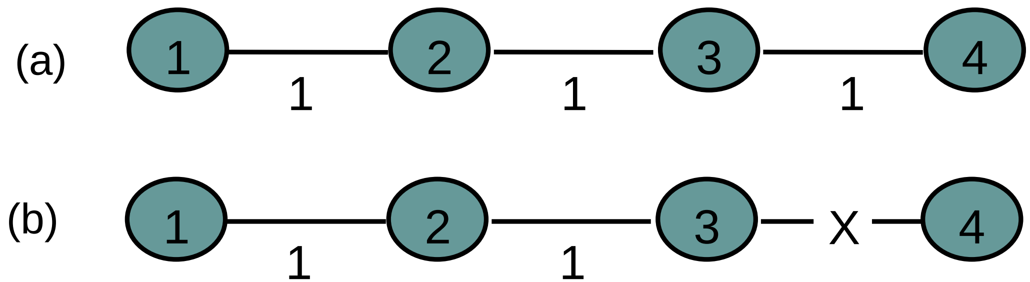
Update	**Node 1**	**Node 2**	**Node 3**
Before break	(2,3)	(3,2)	(4, 1)
After break	(2,3)	(3,2)	(2,3)
1	(2,3)	(3,4)	(2,3)
2	(2,5)	(3,4)	(2,5)
3	(2,5)	(3,6)	(2,5)
4	(2,7)	(3,6)	(2,7)
5	(2,7)	(3,8)	(2,7)
…	…	…	…

Problem: Bad News Travels Slowly

Remedies

- Split Horizon
  - Do not report route to a destination to the neighbor from which route was learned
- Poisoned Reverse
  - Report route to a destination to the neighbor from which route was learned, but with infinite distance
  - Breaks erroneous direct loops immediately
  - Does not work on some indirect loops

Split Horizon with Poison Reverse



Nodes believe best path is through each other

Update	Node 1	Node 2	Node 3	
Before break	(2, 3)	(3, 2)	(4, 1)	
After break	(2, 3)	(3, 2)	(-1, ∞)	Node 2 advertizes its route to 4 to node 3 as having distance infinity; node 3 finds there is no route to 4
1	(2, 3)	(-1, ∞)	(-1, ∞)	Node 1 advertizes its route to 4 to node 2 as having distance infinity; node 2 finds there is no route to 4
2	(-1, ∞)	(-1, ∞)	(-1, ∞)	Node 1 finds there is no route to 4

Link-State Algorithm

- Basic idea: two step procedure
  - Each source node gets a map of all nodes and link metrics (link state) of the entire network
  - Find the shortest path on the map from the source node to all destination nodes
- Broadcast of link-state information
  - Every node *i* in the network broadcasts to every other node in the network:
    - ID's of its neighbors: $\mathcal{N}_i$=set of neighbors of i
    - Distances to its neighbors: $\{C_{ij} \mid j \in N_i\}$
  - Flooding is a popular method of broadcasting packets

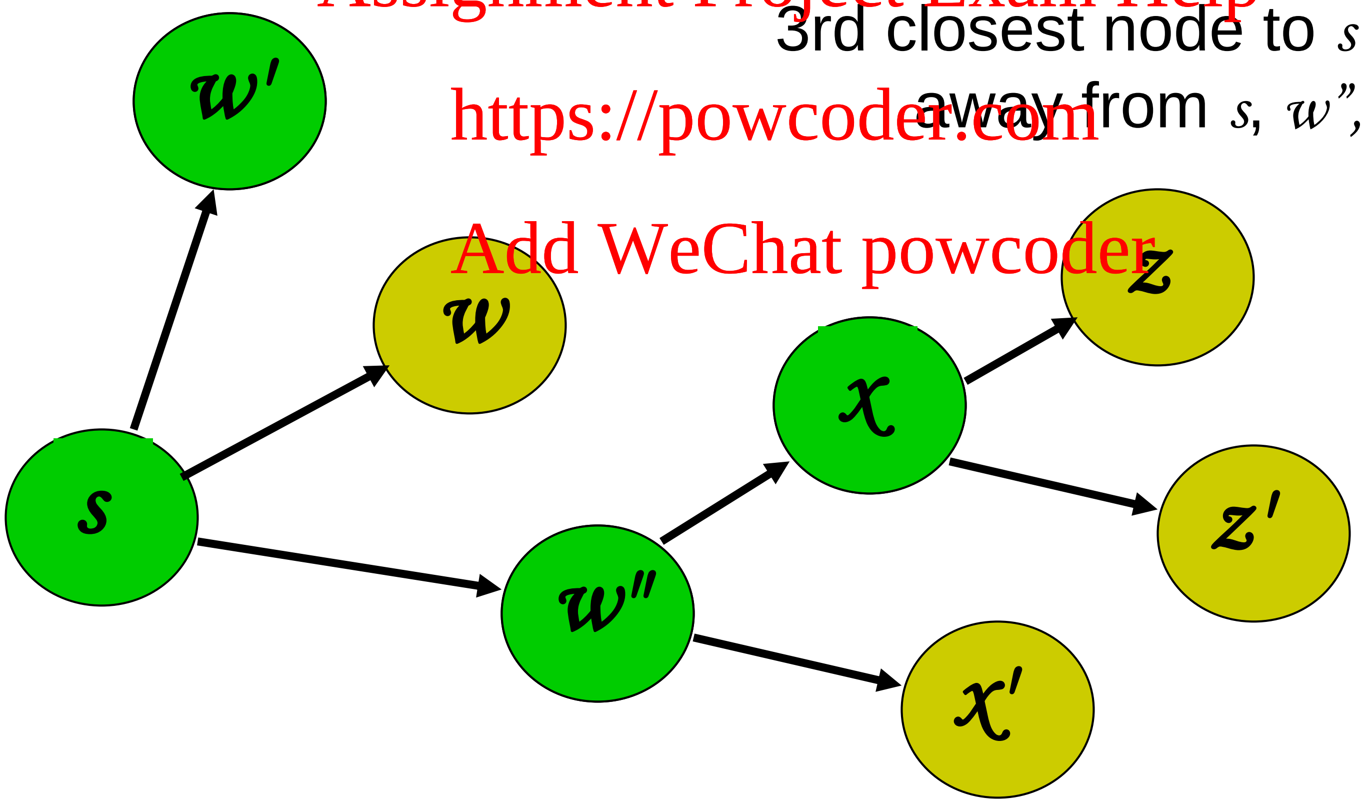
Dijkstra Algorithm: Finding shortest paths in order

Find shortest paths from source s to all other destinations

Closest node to s is 1 hop away

2^{nd} closest node to s is 1 hop away from s or w''

3rd closest node to s is 1 hop away from s, w'', or x

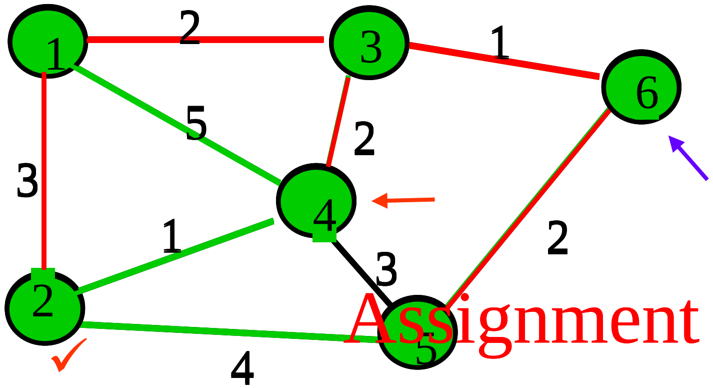


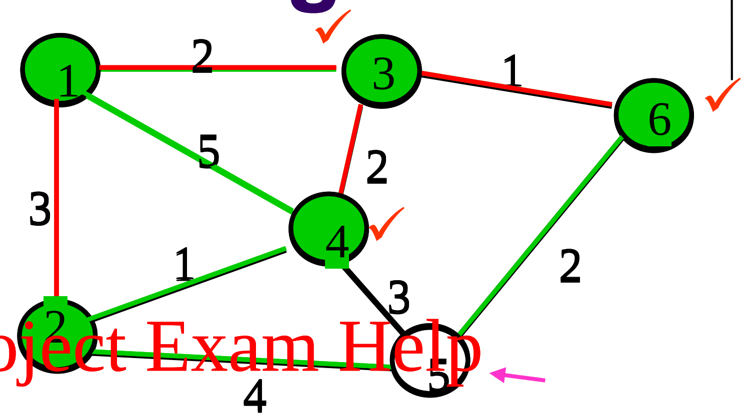
Dijkstra’s algorithm

- N: set of nodes for which shortest path already found
- Initialization: (*Start with source node s)*
  - $N = \{s\}$, $D_s = 0$, “*s* is distance zero from itself”
  - $D_j = C_{sj}$ for all $j \neq s$, distances of directly-connected neighbors
- Step A: (*Find next closest node i*)
  - Find $i \notin N$ such that
  - $D_i = \min Dj$ for $j \notin N$
  - Add *i* to *N*
  - If *N* contains all the nodes, stop
- Step B: (*update minimum costs)*
  - For each node $j \notin N$
  - $D_j = \min (D_j, D_i + C_{ij})$
  - Go to Step A

Minimum distance from s to *j through node **i** in N*

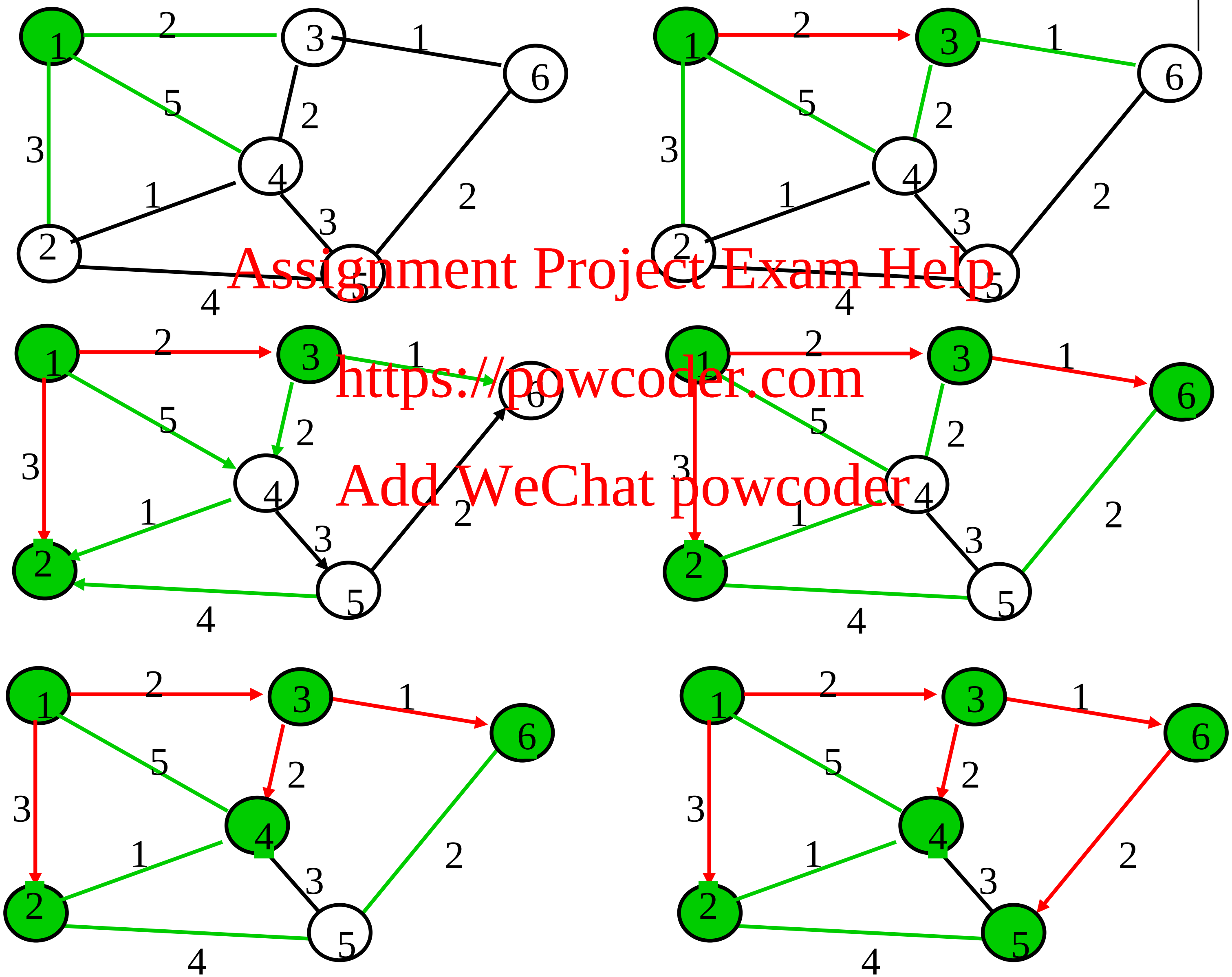
Execution of Dijkstra's algorithm





Iteration	**N**	D_2	D_3	D_4	D_5	D_6
Initial	{1}	3	2	5	$\propto$	$\propto$
1	{1,3}	3	2	4	$\propto$	3
2	{1,2,3}	3	2	4	7	3
3	{1,2,3,6}	3	2	4	5	3
4	{1,2,3,4,6}	3	2	4	5	3
5	{1,2,3,4,5,6}	3	2	4	5	3

Shortest Paths in Dijkstra's Algorithm



Reaction to Failure

- If a link fails,
  - Router sets link distance to infinity & floods the network with an update packet
  - All routers immediately update their link database & recalculate their shortest paths
  - Recovery very quick
- But watch out for old update messages
  - Add time stamp or sequence # to each update message
  - Check whether each received update message is new
  - If new, add it to database and broadcast
  - If older, send update message on arriving link

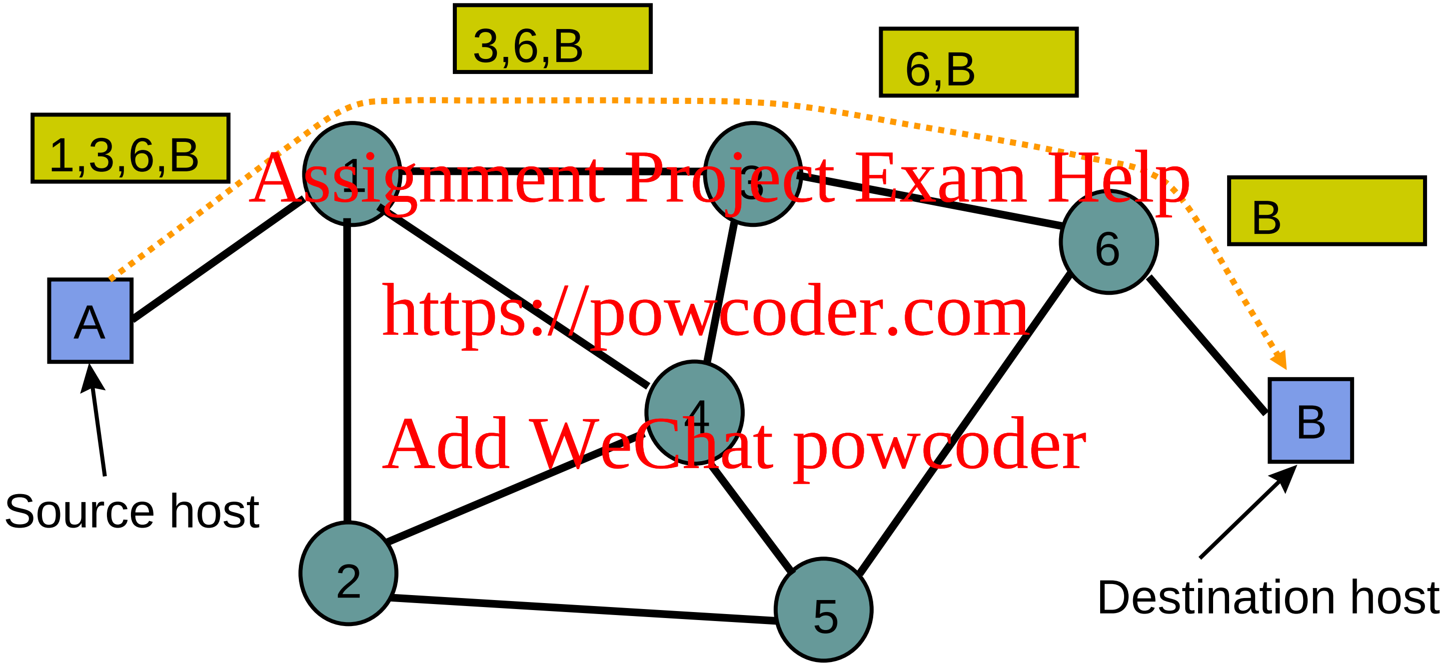
Why is Link State Better?

- Fast, loopless convergence
- Support for precise metrics, and multiple metrics if necessary (throughput, delay, cost, reliability)
- Support for multiple paths to a destination
  - algorithm can be modified to find best two paths

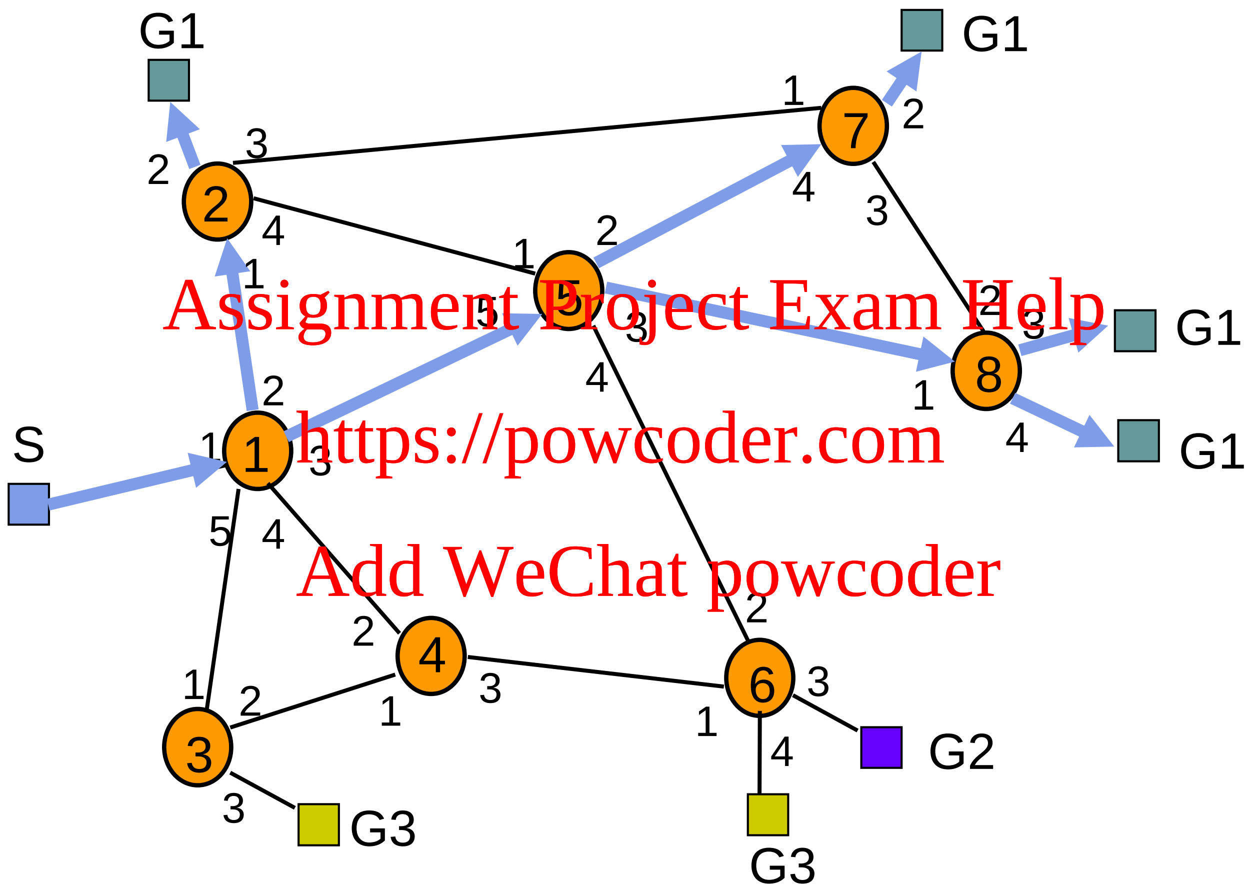
Source Routing

- Source host selects path that is to be followed by a packet
  - Strict: sequence of nodes in path inserted into header
  - Loose: subsequence of nodes in path specified
- Intermediate switches read next-hop address and remove address
- Source host needs link state information or access to a route server
- Source routing allows the host to control the paths that its information traverses in the network
- Potentially the means for customers to select what service providers they use

Example



Multicasting



- Source S sends packets to multicast group G1

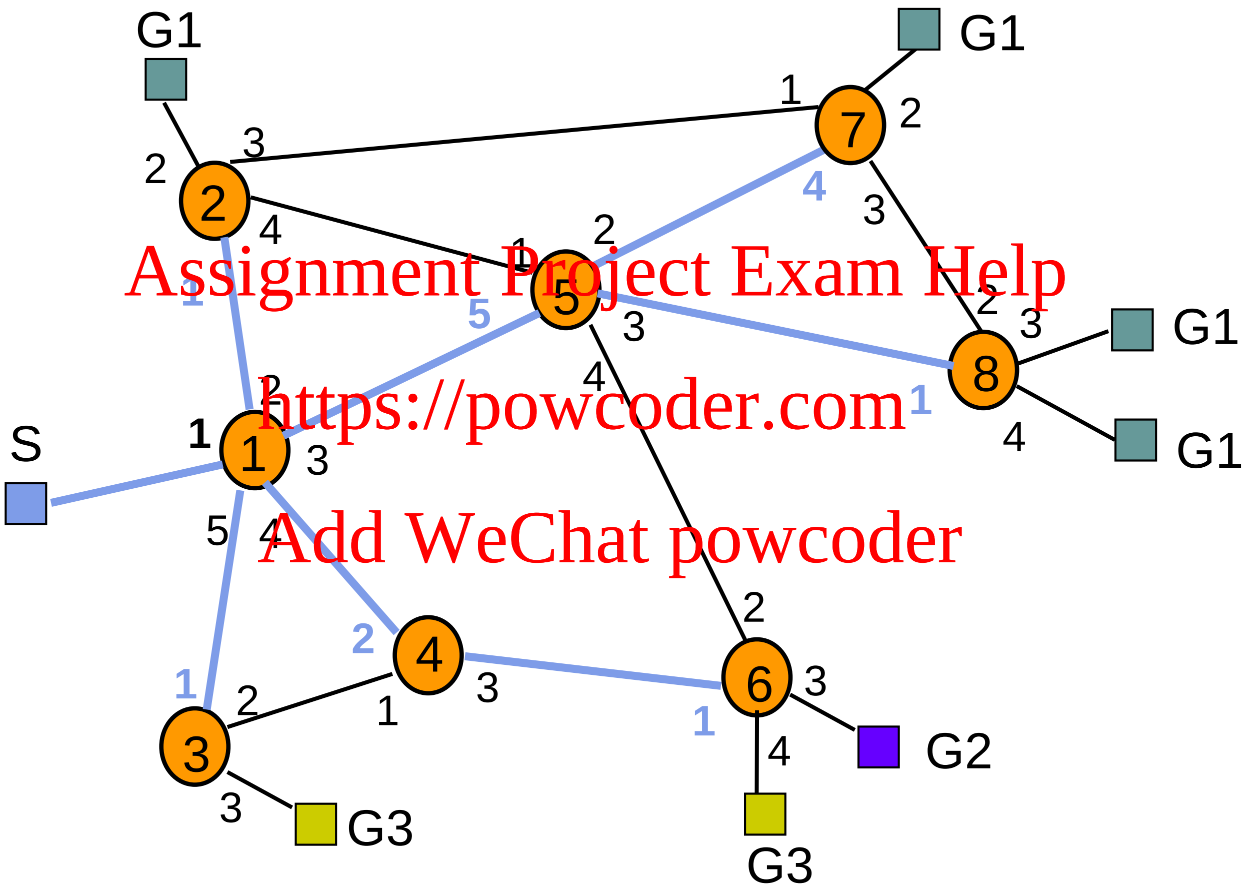
Multicast Routing

- Multicast routing useful when a source wants to transmits its packets to several destinations simultaneously
- Relying on unicast routing by transmitting each copy of packet separately works, but can be very inefficient if number of destination is large
- Typical applications is multi-party conferencing over the Internet
- Example: Multicast Backbone (MBONE) uses *reverse path multicasting*

Reverse-Path Broadcasting (RPB)

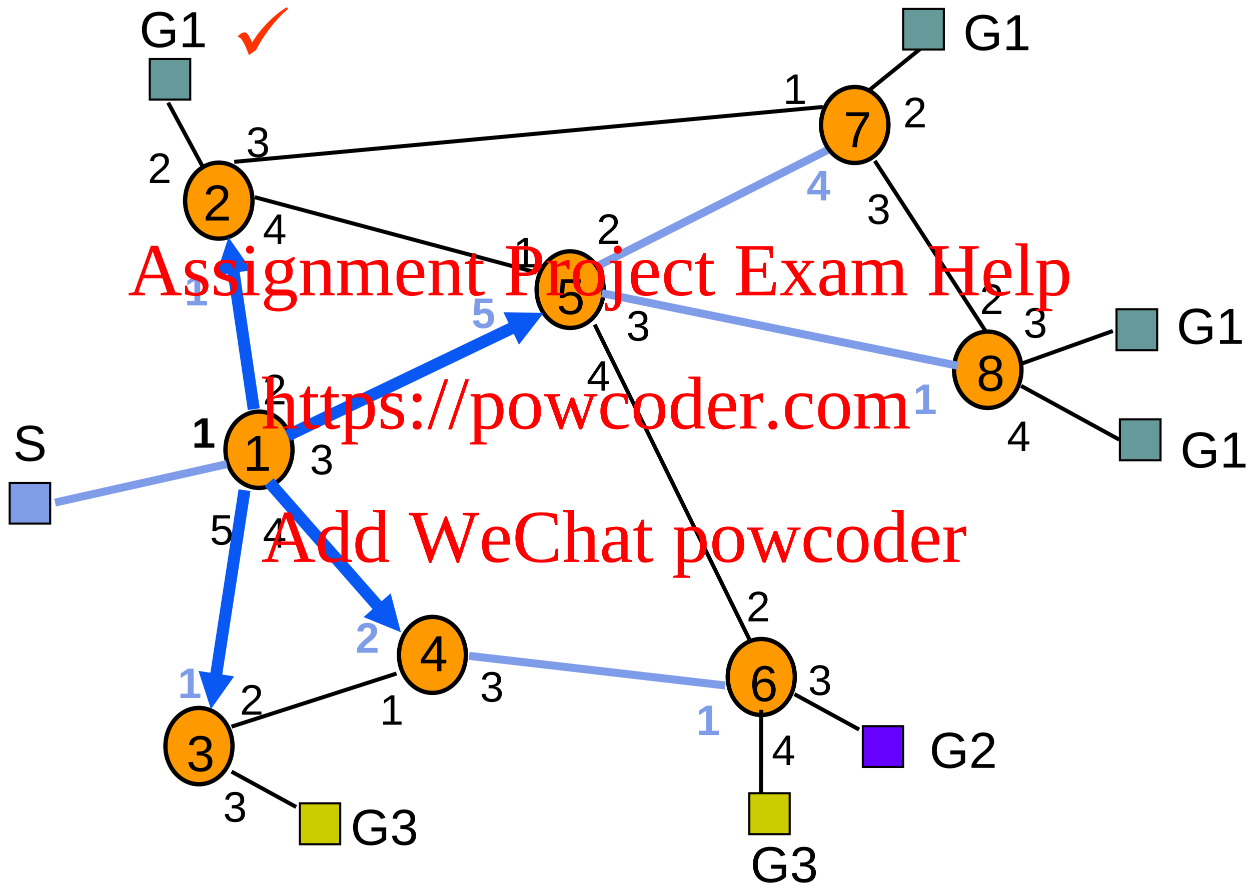
- Fact: Set of shortest paths to the source node S forms a tree that spans the network
  - Approach: Follow paths in *reverse* direction
- Assume each router knows current shortest path to S
  - Upon receipt of a multicast packet, router records the packet’s source address and the port it arrives on
  - If shortest path to source is through same port (“parent port”), router forwards the packet to all other ports
  - Else, drops the packet
- Loops are suppressed; each packet forwarded a router exactly once
- Implicitly assume shortest path to source S is same as shortest path from source
  - If paths asymmetric, need to use link state info to compute shortest paths from S

Example: Shortest Paths from S



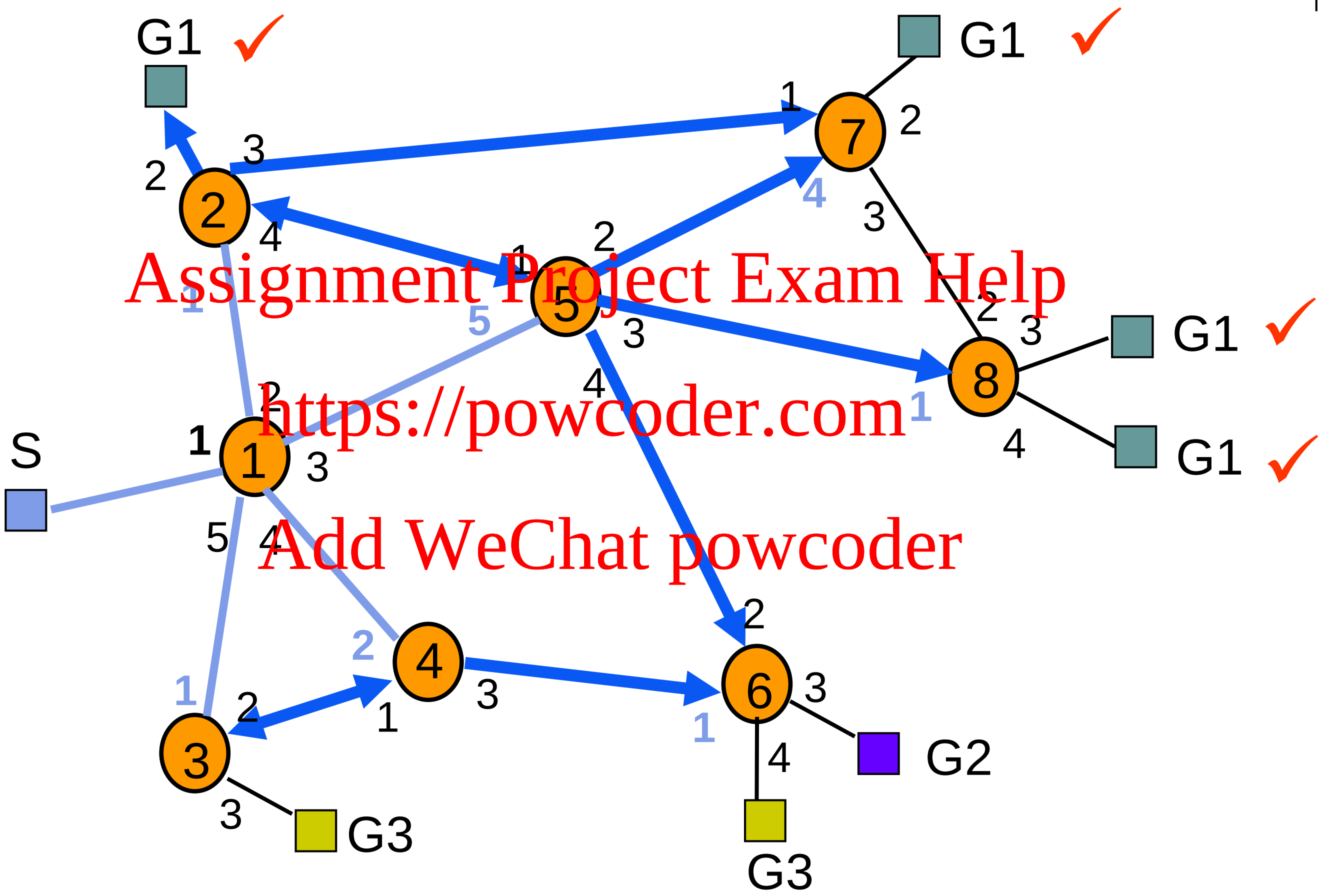
- Spanning tree of shortest paths to node S and parent ports are shown in blue

Example: S sends a packet



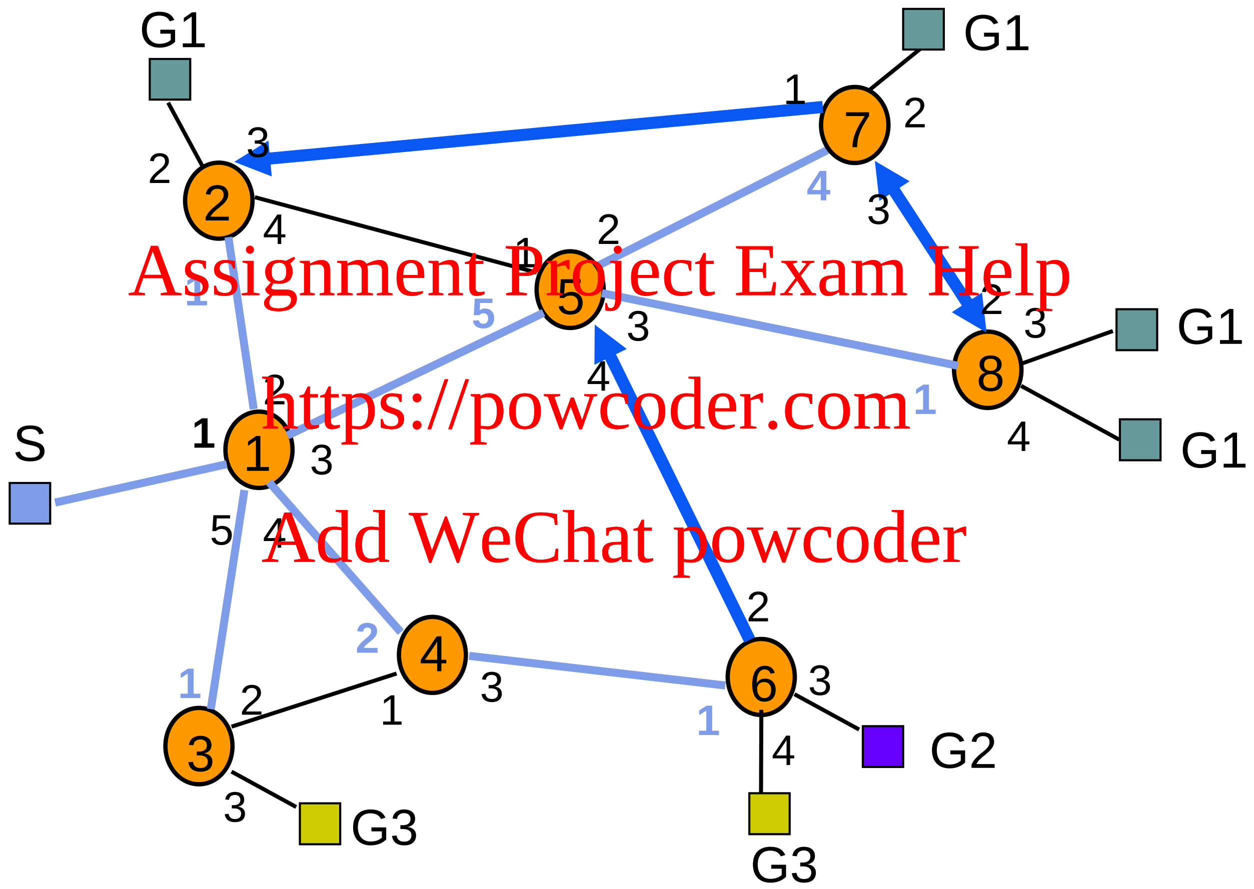
- S sends a packet to node 1
- Node 1 forwards to all ports, except parent port

Example: Hop 1 nodes broadcast



- Nodes 2, 3, 4, and 5 broadcast, except on parent ports
- All nodes, not only G1, receive packets

Example: Broadcast continues



- *Truncated RPB (TRPB)*: Leaf routers do not broadcast if none of its attached hosts belong to packet’s multicast group

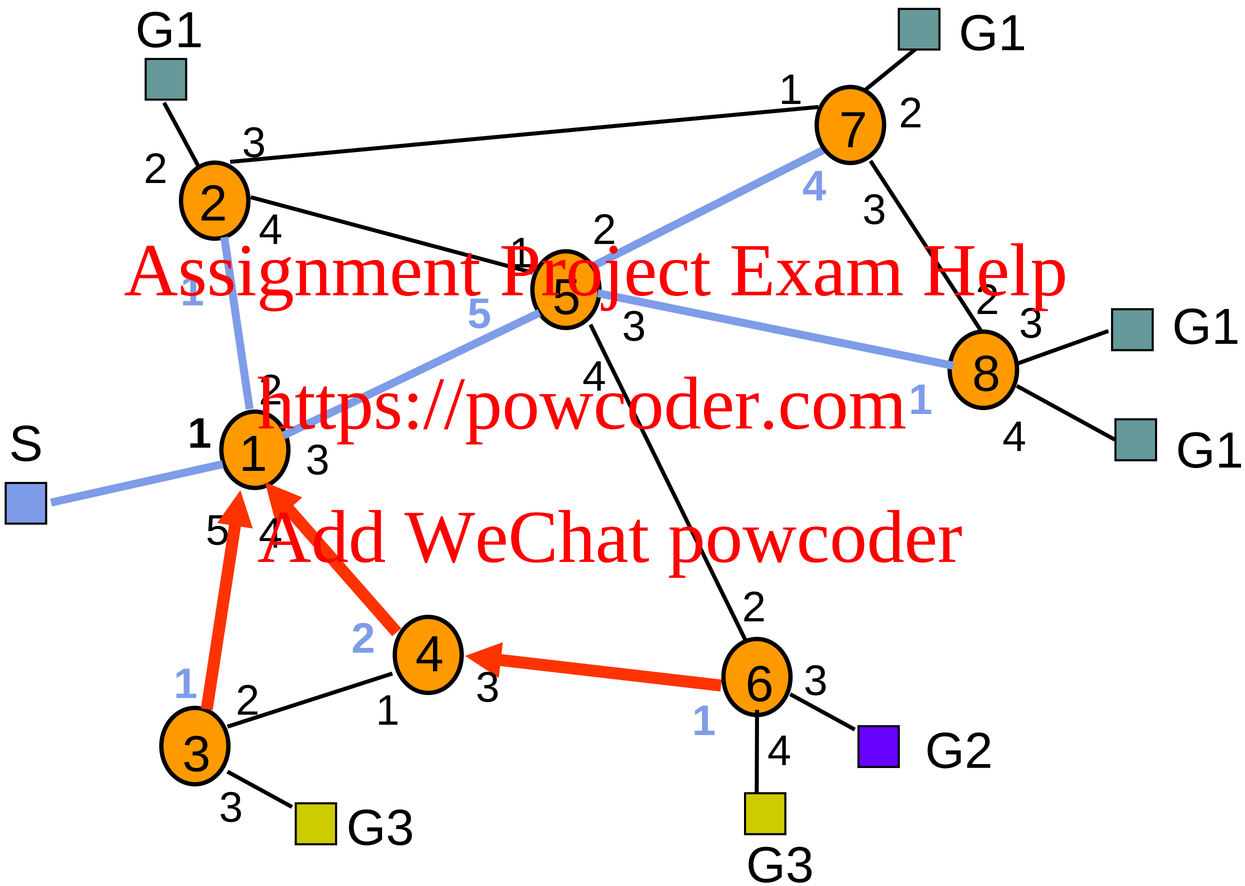
Internet Group Management Protocol (IGMP)

- *Internet Group Management Protocol:*
  - Host can join a multicast group by sending an IGMP message to its router
- Each multicast router periodically sends an IGMP query message to check whether there are hosts belonging to multicast groups
  - Hosts respond with list of multicast groups they belong to
  - Hosts randomize response time; cancel response if other hosts reply with same membership
- Routers determine which multicast groups are associated with a certain port
- Routers only forward packets on ports that have hosts belonging to the multicast group

Reverse-Path Multicasting (RPM)

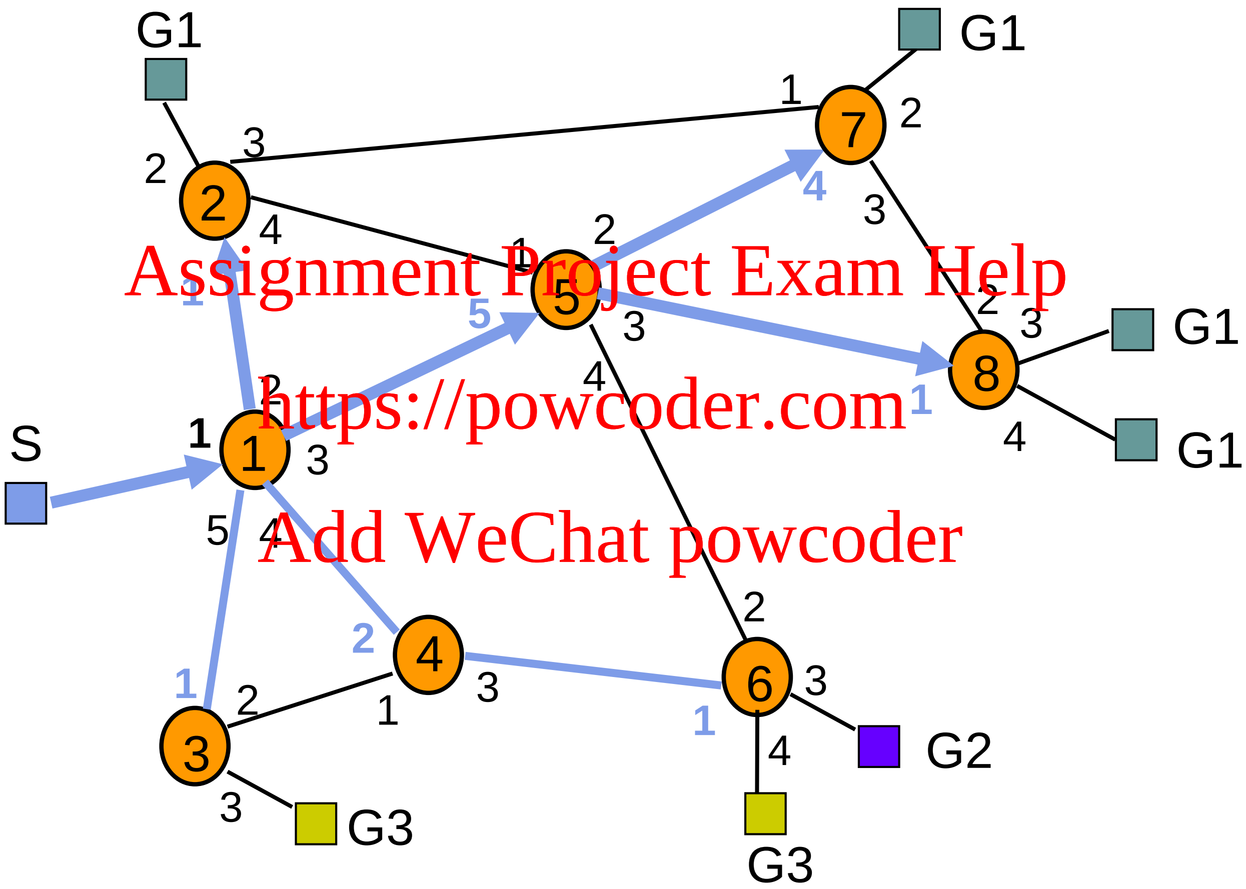
- *Reverse Path Multicasting (RPM)* relies on IGMP to identify multicast group membership
- The first packet to a given (source, group), i.e. (S,G) is transmitted to all leaf routers using TRPB
- Each leaf router that has no hosts that belong to this group on any of its ports, sends a ***prune*** message to its upstream router to stop sending packets to (S, G)
- Upstream routers that receive prune messages from all their downstream routers, send prune messages upstream
- Prune entries in each router have finite lifetime
- If a host requests to join a group, routers can cancel previous pruning with a ***graft*** message

Example: Pruning for G1



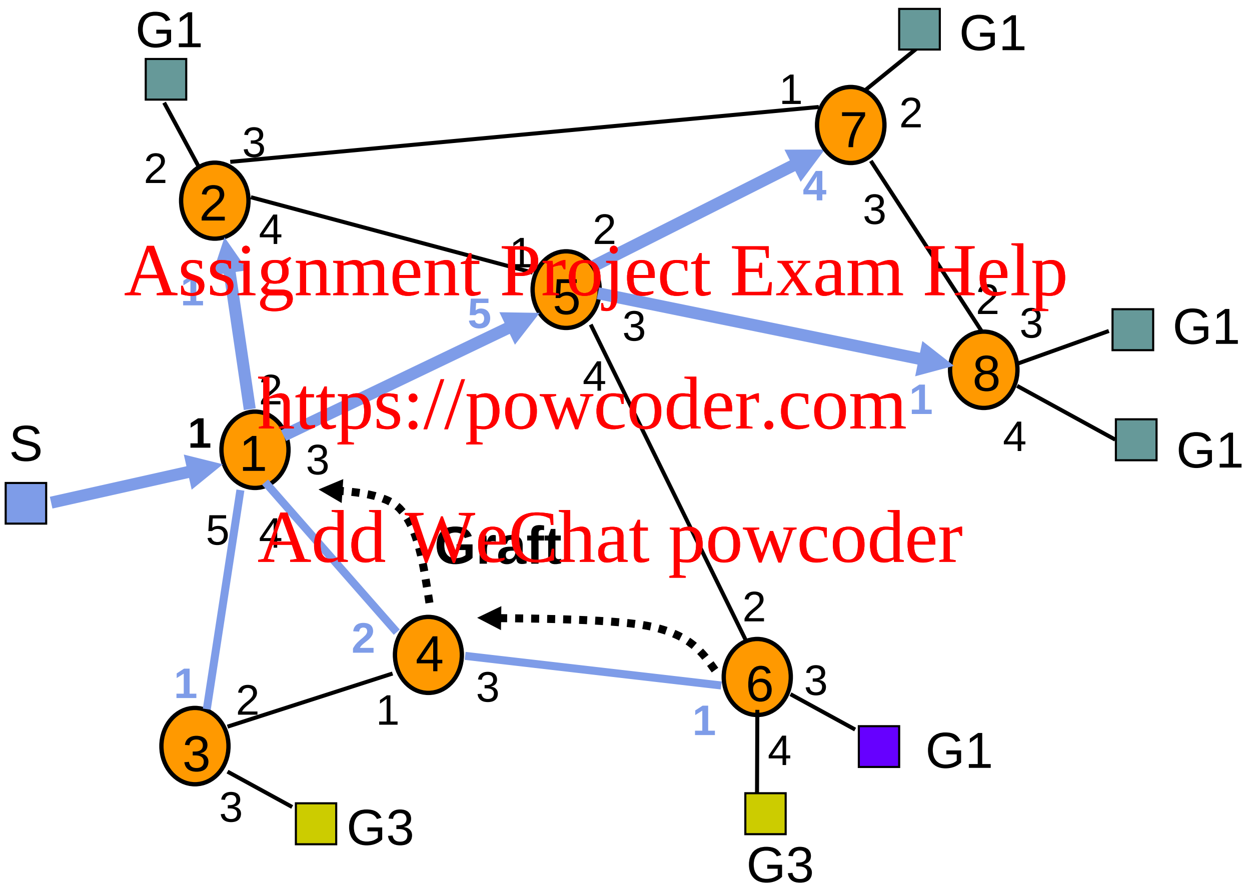
- Routers 3, 4, and 6 send prune messages upstream

Example: RPM Multicast Tree



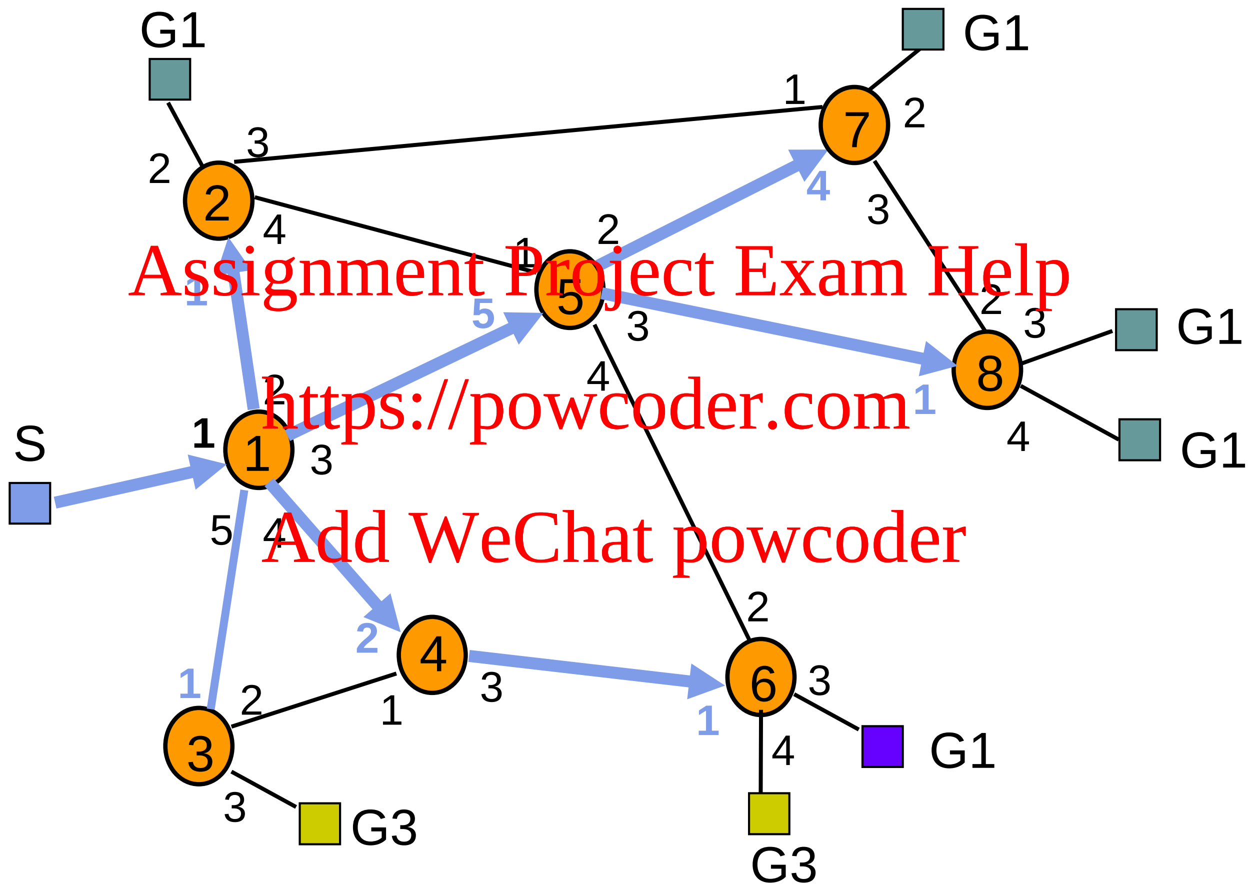
- RPM multicast tree after pruning

Example: Graft from Router 6



- Graft message flows upstream to router 1

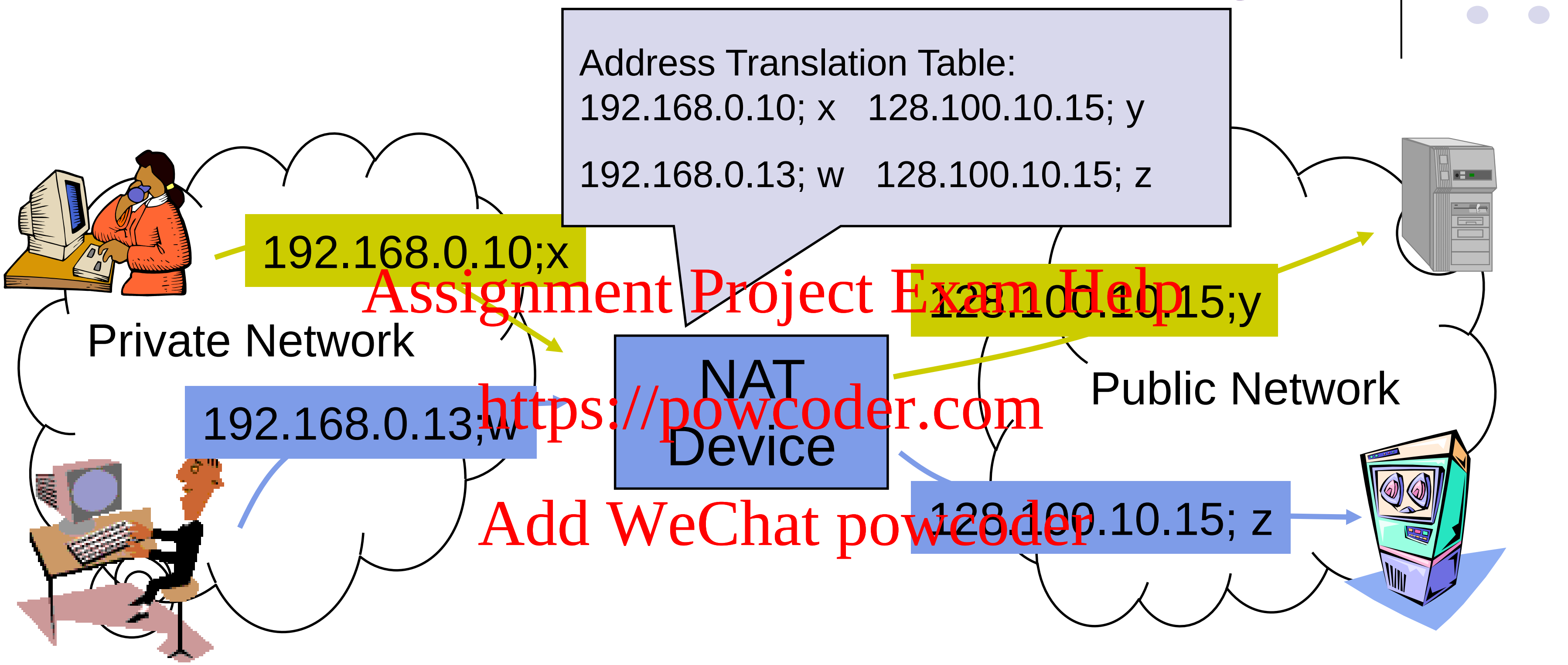
Example: RPM Tree after Graft



Network Address Translation (NAT)

- Class A, B, and C addresses have been set aside for use within private internets
  - Packets with private ("unregistered") addresses are discarded by routers in the global Internet
- NAT (RFC 1631): method for mapping packets from hosts in private internets into packets that can traverse the Internet
  - A device (computer, router, firewall) acts as an agent between a private network and a public network
  - A number of hosts can share a limited number of registered IP addresses
    - Static/Dynamic NAT: map unregistered addresses to registered addresses
    - Overloading: maps multiple unregistered addresses into a single registered address (e.g. Home LAN)

NAT Operation (Overloading)



- Hosts inside private networks generate packets with private IP address & TCP/UDP port #s
- NAT maps each private IP address & port # into shared global IP address & available port #
- Translation table allows packets to be routed unambiguously