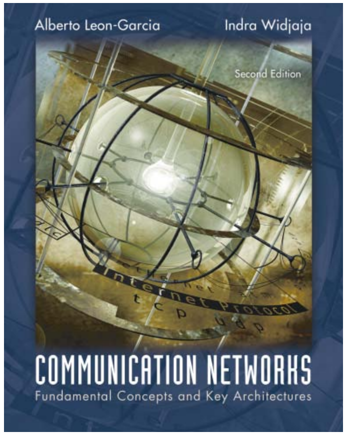
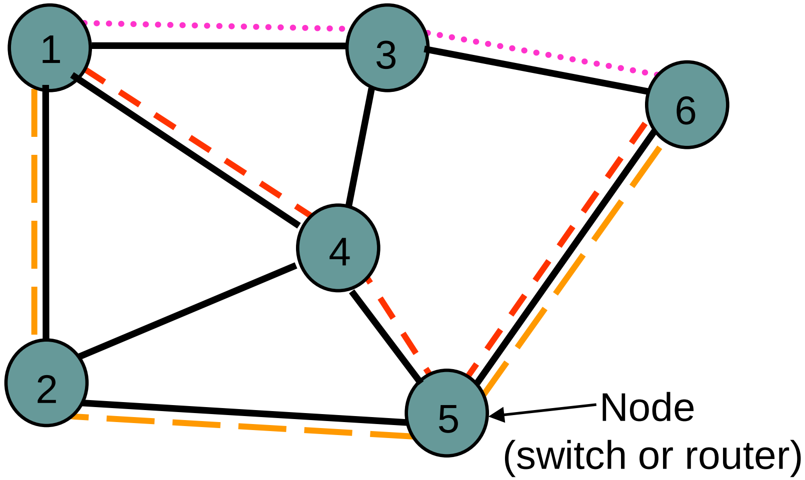
Chapter 7
Packet-Switching Networks



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?

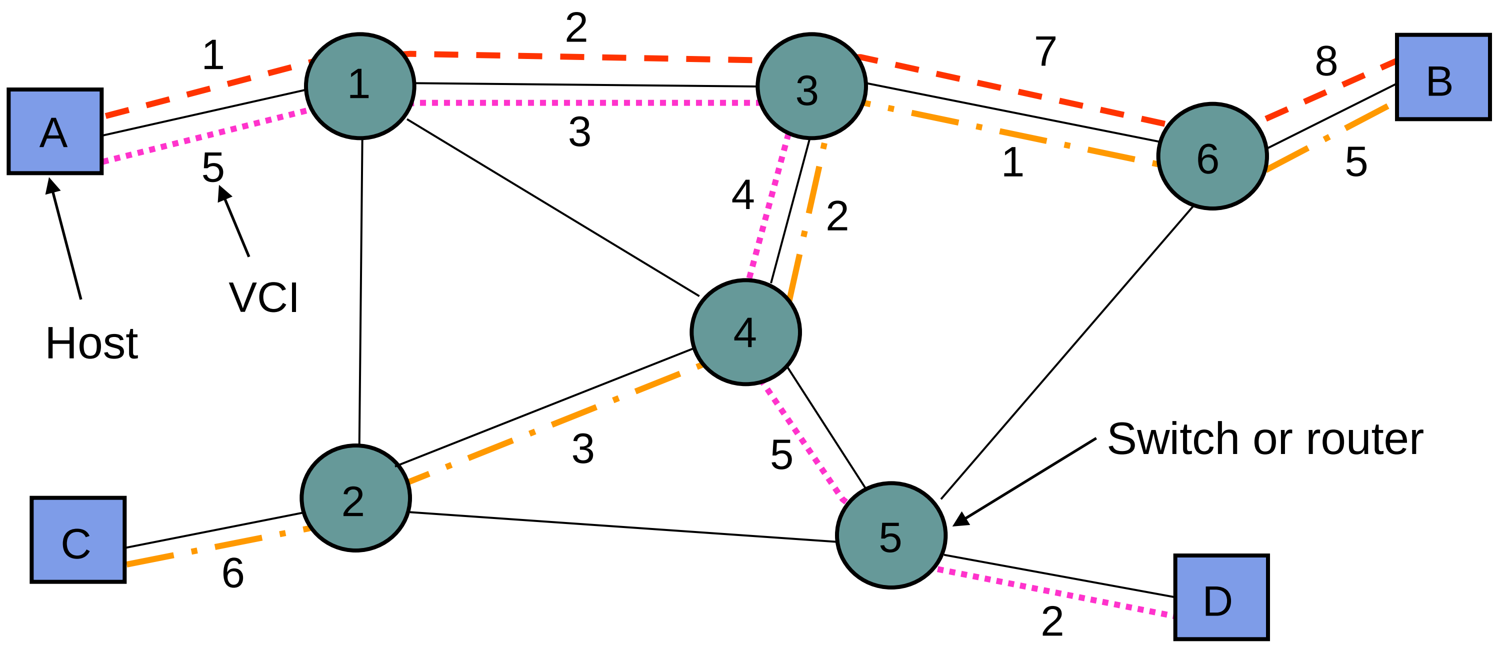
Creating the Routing Tables

- 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 A & VCI 5 → 3 & VCI 3 → 4 & VCI 4
  - → 5 & VCI 5 → 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 2

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

Node 3

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

Node 4

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

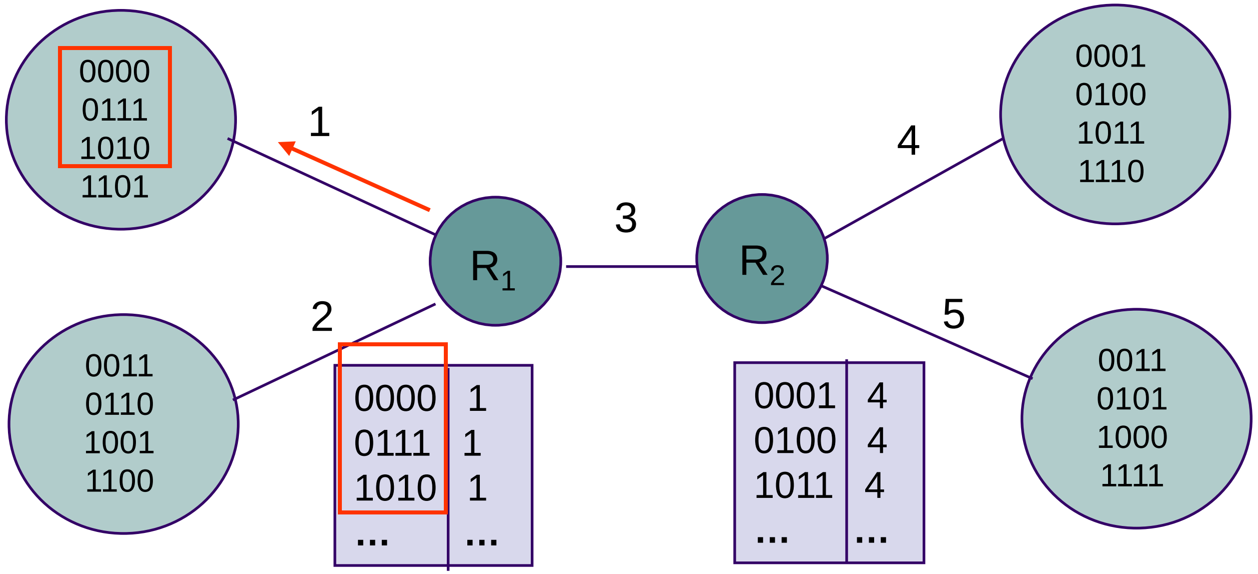
Node 5

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

Node 6

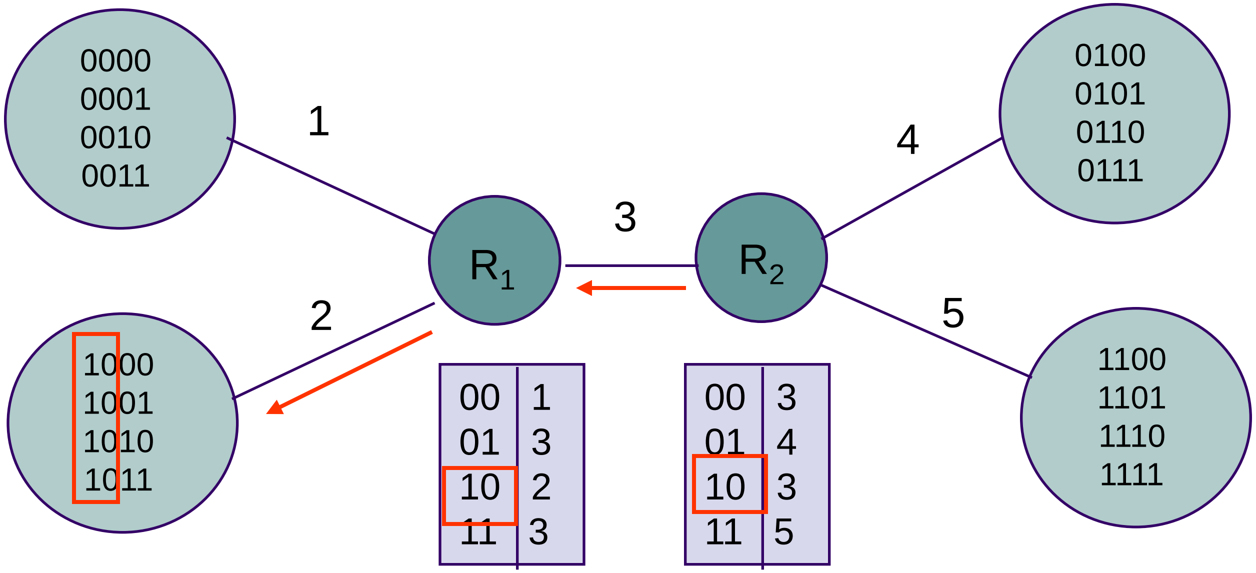
Destination	Next node
1	3
2	5
3	3
4	3
5	5

Non-Hierarchical Addresses and Routing



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

Hierarchical Addresses and Routing



- Prefix indicates network where host is attached
- Routing tables require 4 entries each

Specialized Routing

- Flooding
  - Useful in starting up network
  - Useful in propagating information to all nodes

- Deflection Routing
  - Fixed, preset routing procedure
  - No route synthesis

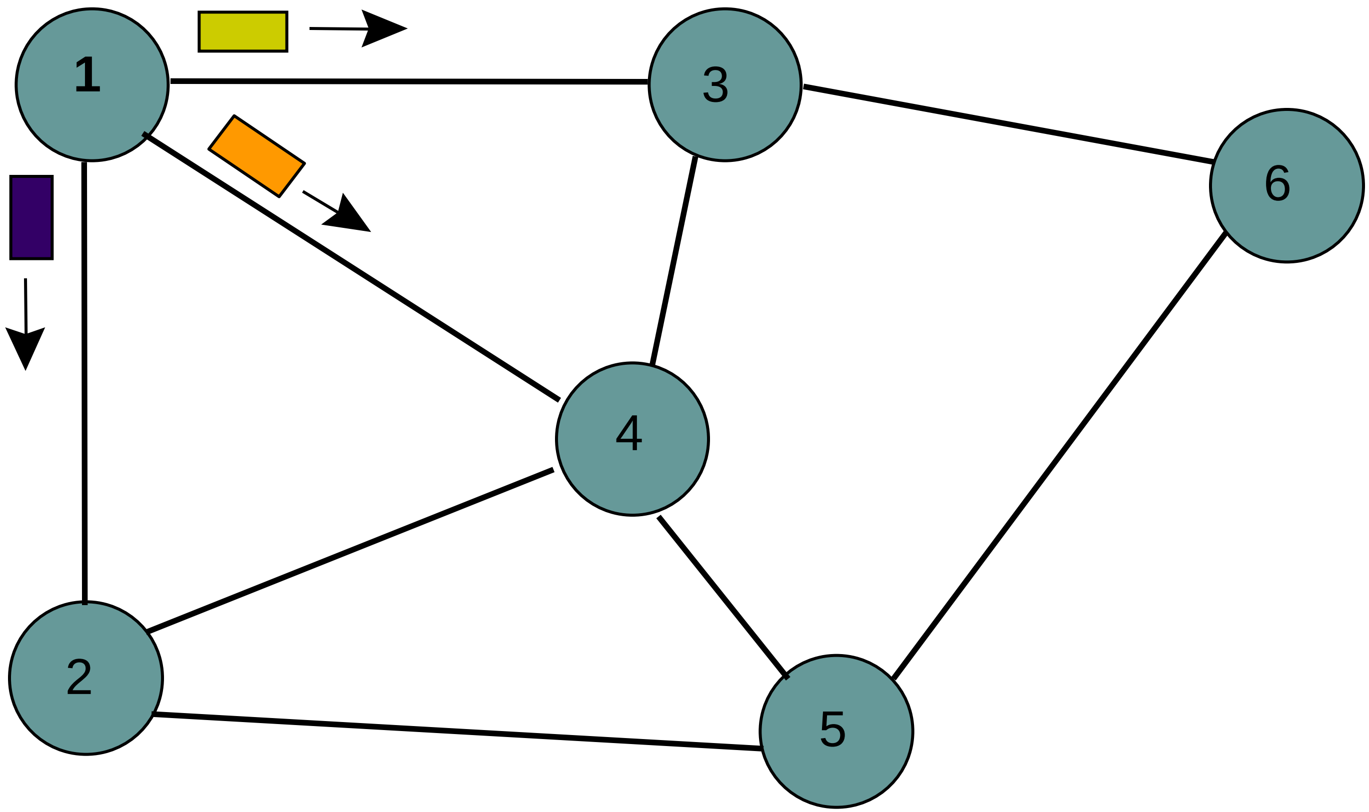
Flooding

Send a packet to all nodes in a network

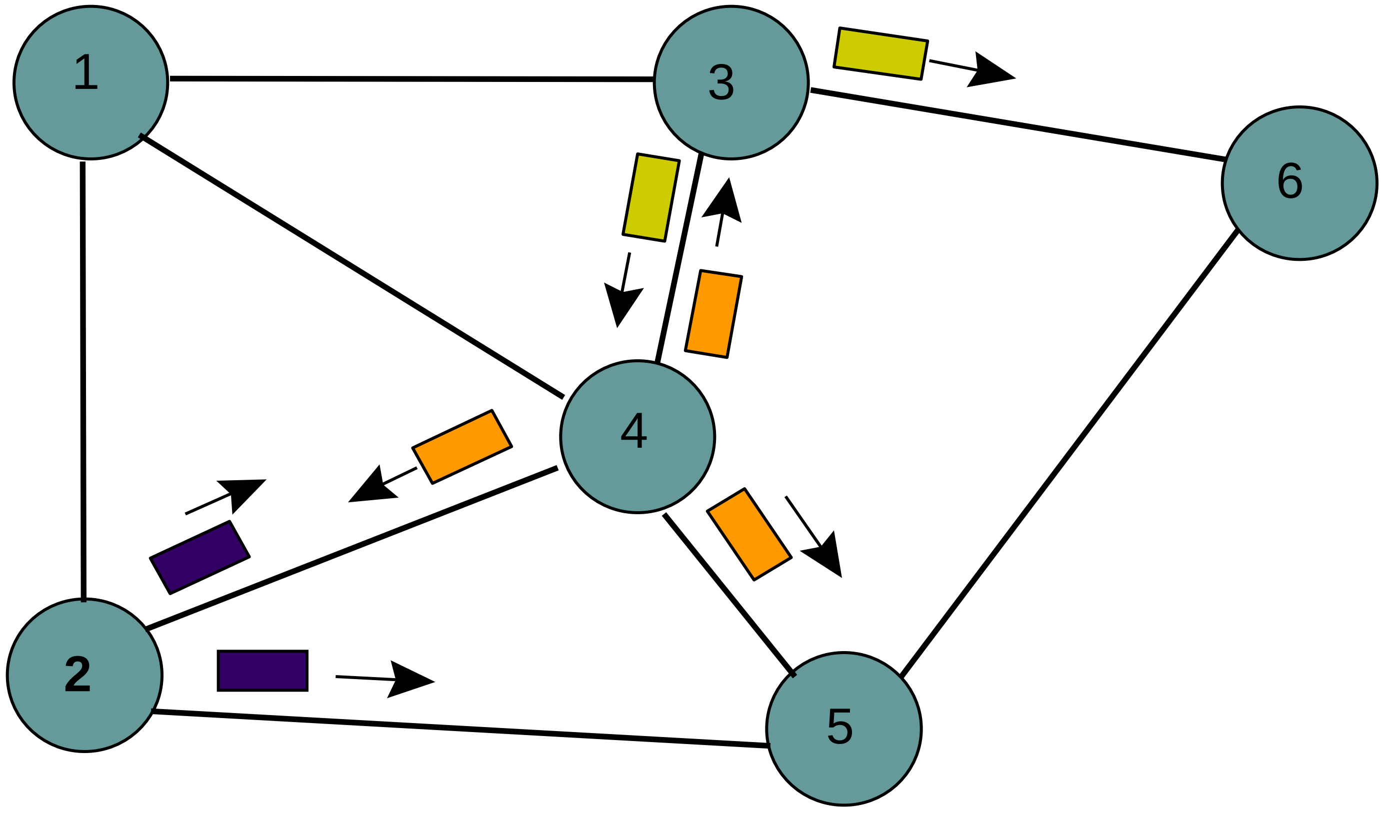
- No routing tables available
- Need to broadcast packet to all nodes (e.g. to propagate link state information)

Approach

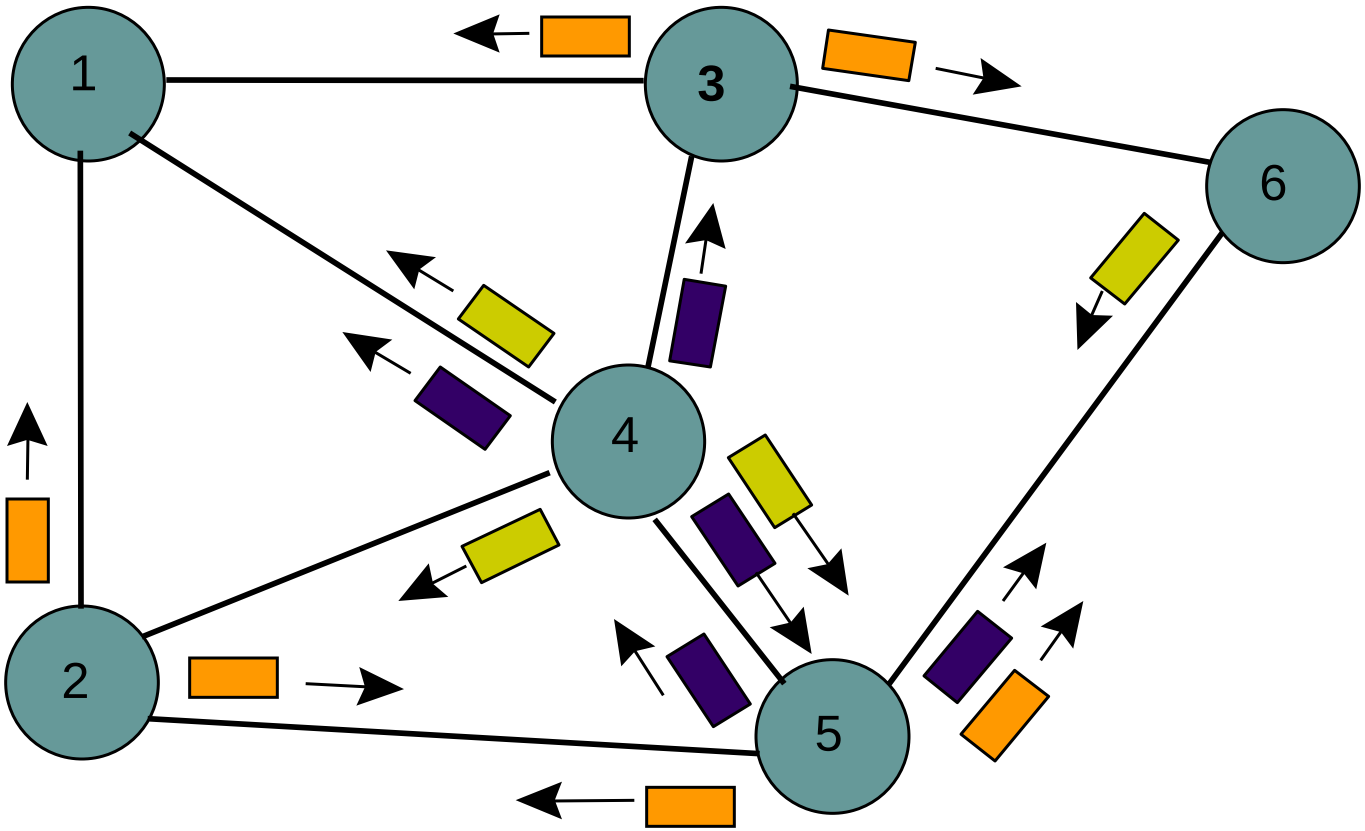
- Send packet on all ports except one where it arrived
- Exponential growth in packet transmissions



Flooding is initiated from Node 1: Hop 1 transmissions



Flooding is initiated from Node 1: Hop 2 transmissions



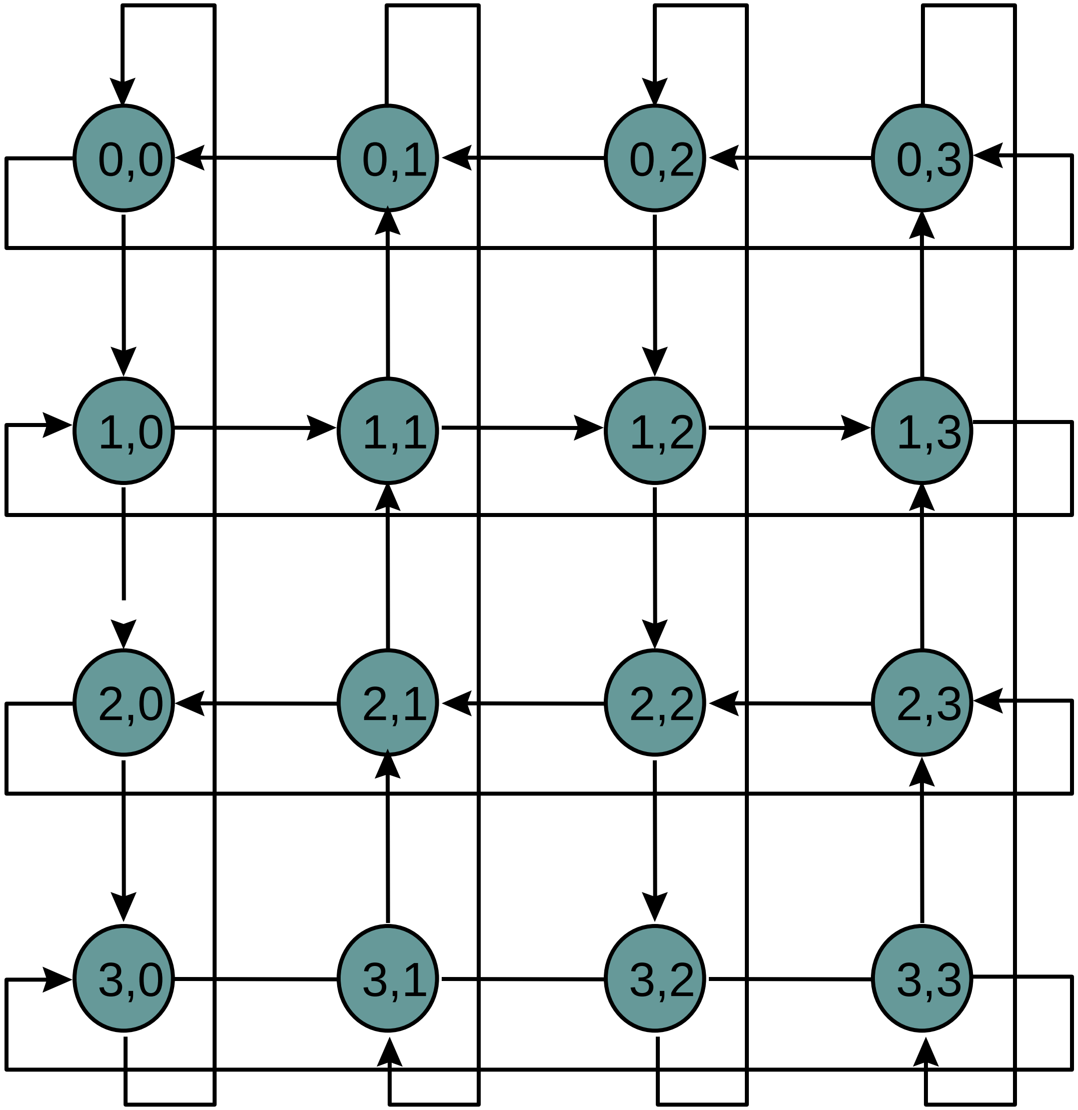
Flooding is initiated from Node 1: Hop 3 transmissions

Limited Flooding

- Time-to-Live (TTL) 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; a switch/router records source address and sequence number and discards repeats

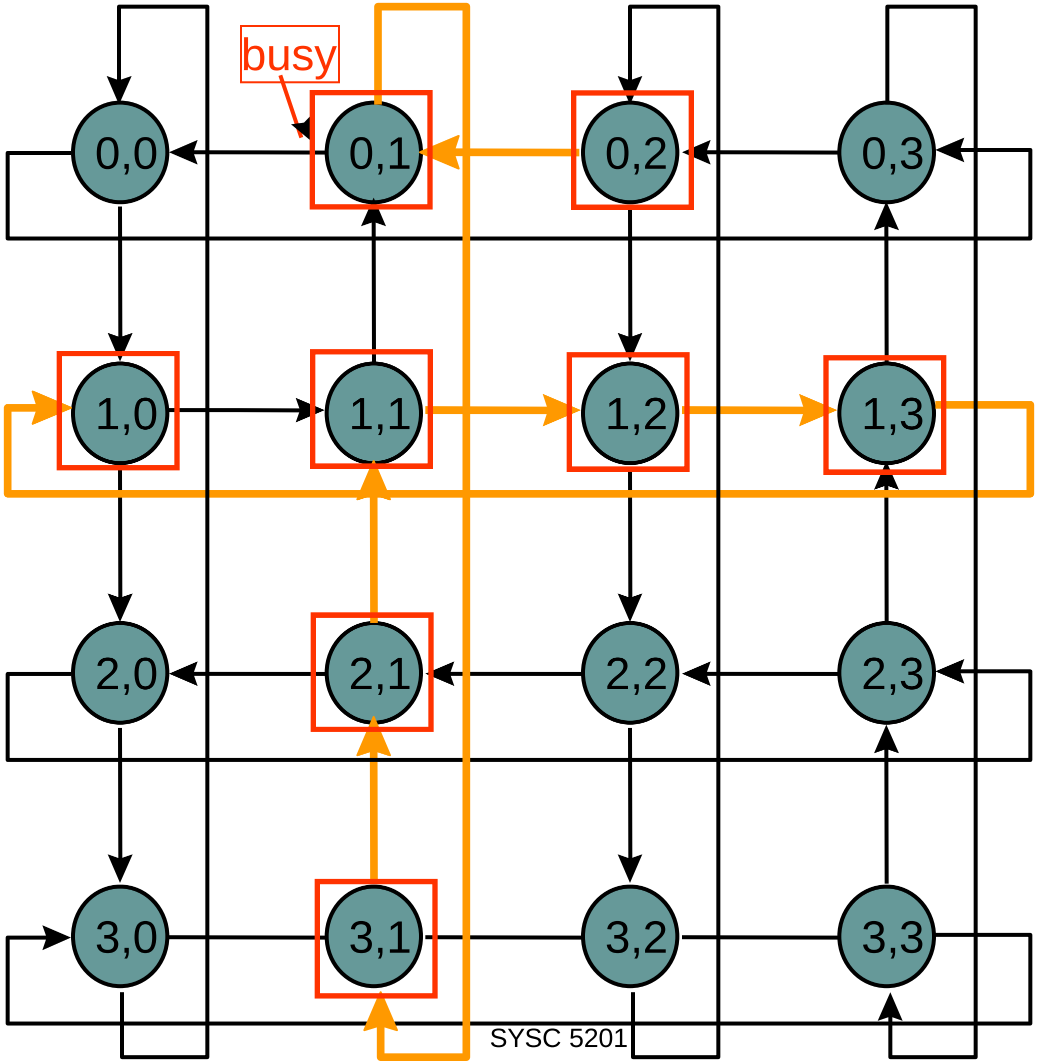
Deflection Routing

- 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

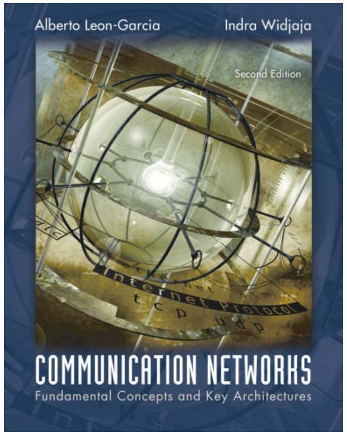


Tunnel from last column to first column or vice versa

Example: Node (0,2)→(1,0)



Chapter 7
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?

San Jose 294

San Jose 392

San Jose 596

San Jose 250

Distance Vector

Local Signpost

- Direction
- Distance

Routing Table

For each destination list:

- Next Node
- Distance

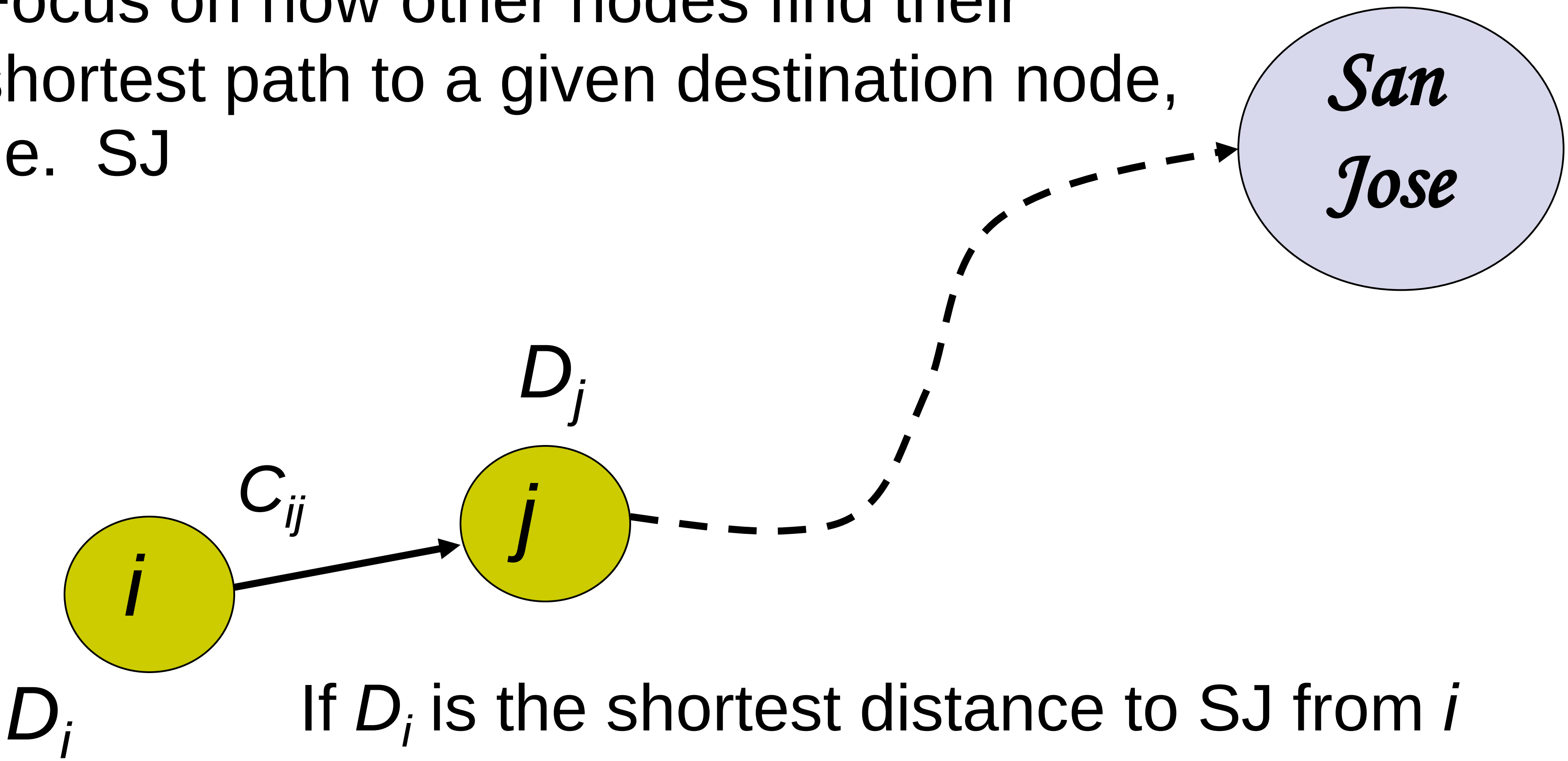
dest	next	dist

Table Synthesis

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

Shortest Path to SJ

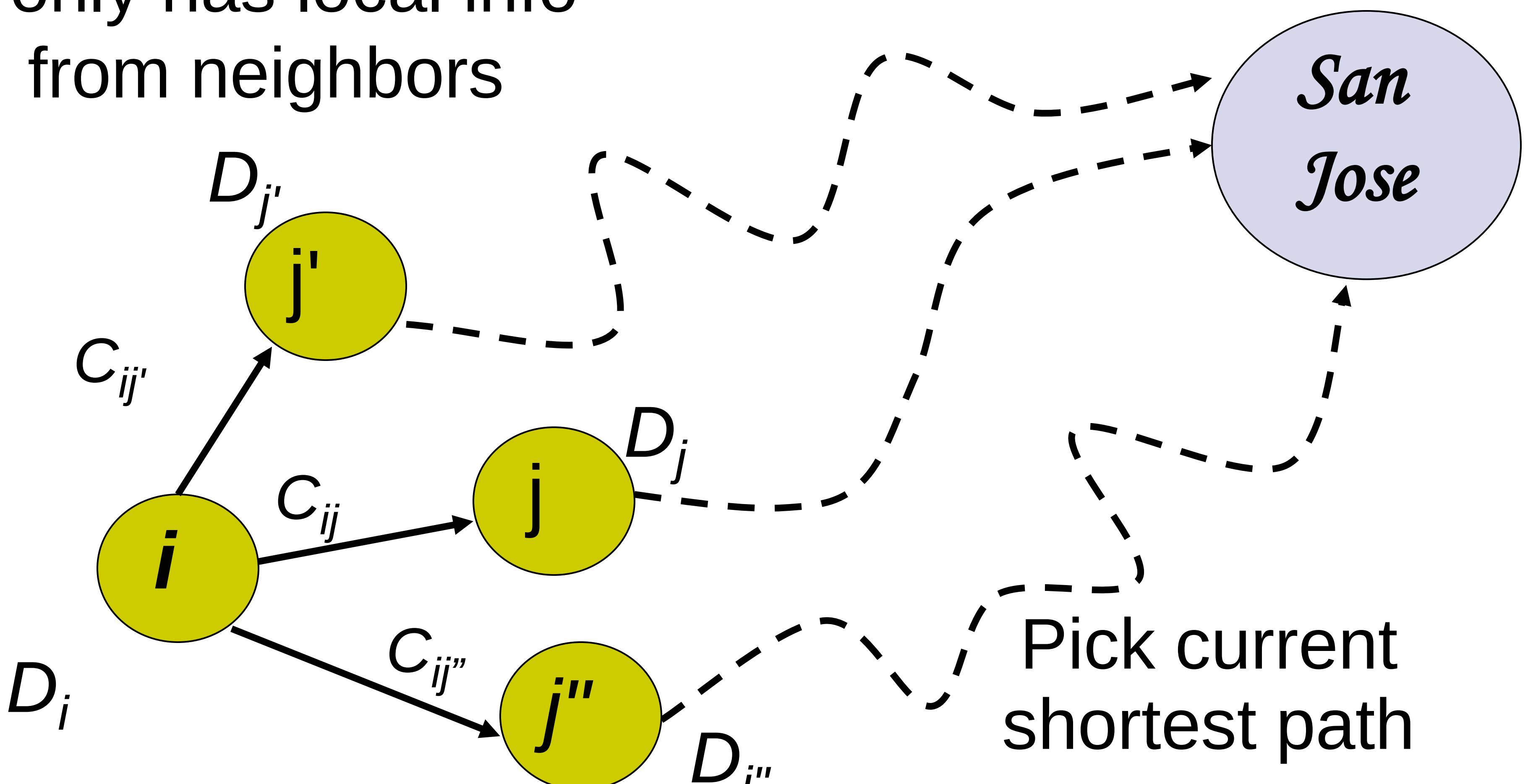
Focus on how other nodes find their shortest path to a given destination node, i.e. SJ



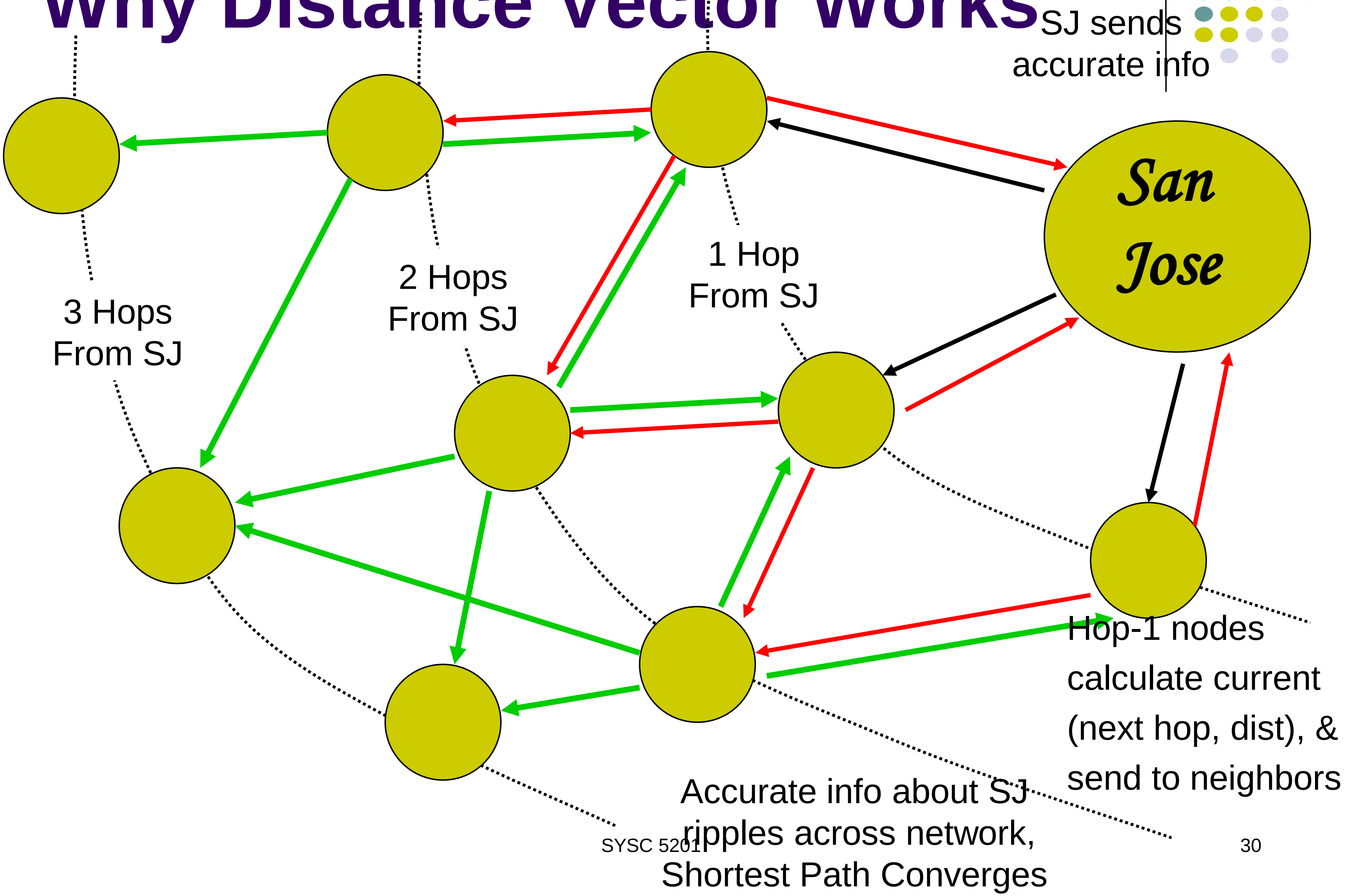
If D_i is the shortest distance to SJ from i and if j is a neighbor on the shortest path, then $D_i = C_{ij} + D_j$

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 i, find the next hop that gives the minimum distance to *d*</u>,
    - $D_i = min_j \{C_{ij} + D_j(d)\}$

    Replace old $(n_j, D_j(d))$ by new $(n_j^*, D_j^*(d))$ if new next node or distance found
  - Go to send step

Bellman-Ford Algorithm

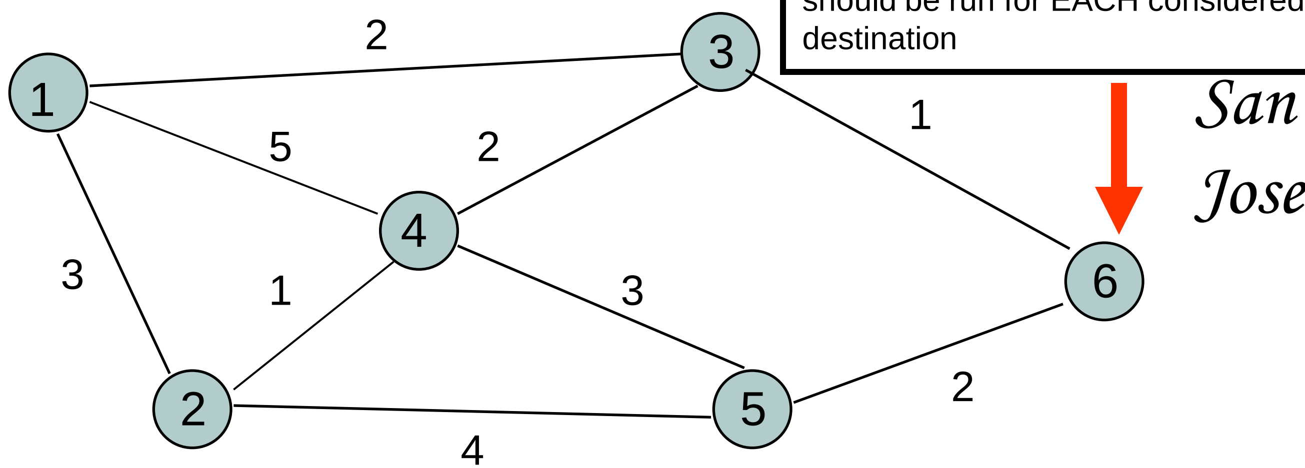
- *Now consider parallel computations* *for all destinations* d
- *Initialization*
  - Each node has 1 row for each destination *d*
  - Distance of node *d* to itself is zero: $D_d(d)=0$
  - Distance of other node *j* to *d* is infinite: $D_j(d)= \propto$, for $j \neq d$
  - Next node n_j = -1 since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

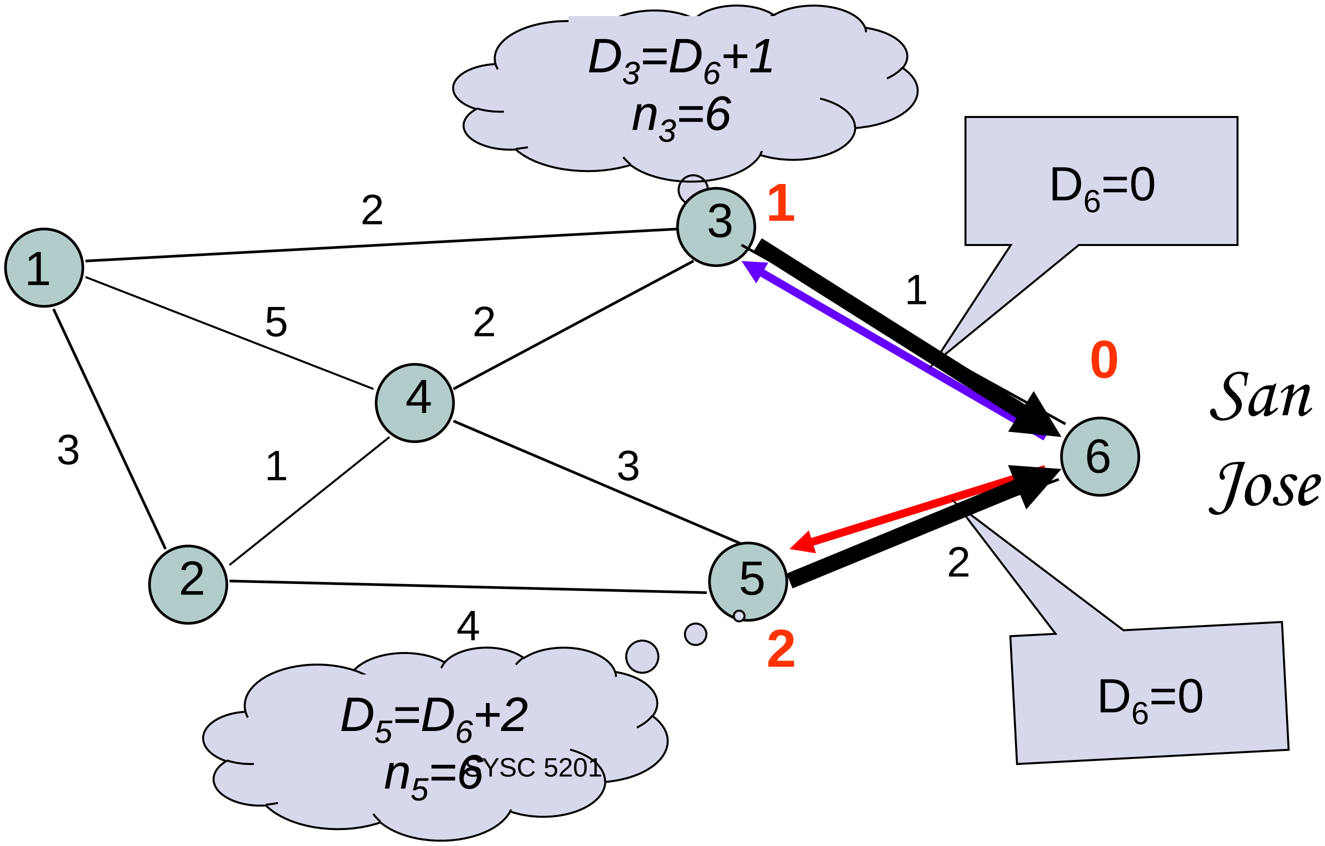
Table entry @ node 1 for dest SJ

Table entry @ node 3 for dest SJ

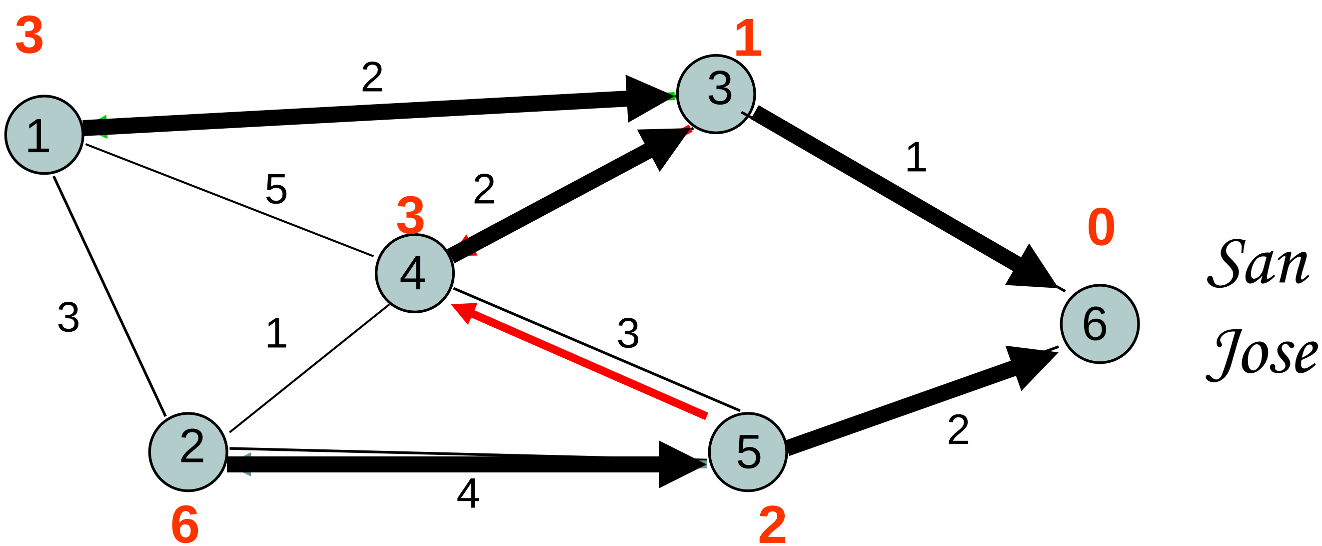
Please note that in this example we determine the optimal path to **destination node 6** from each other node. In general the same algorithm should be run for EACH considered as destination



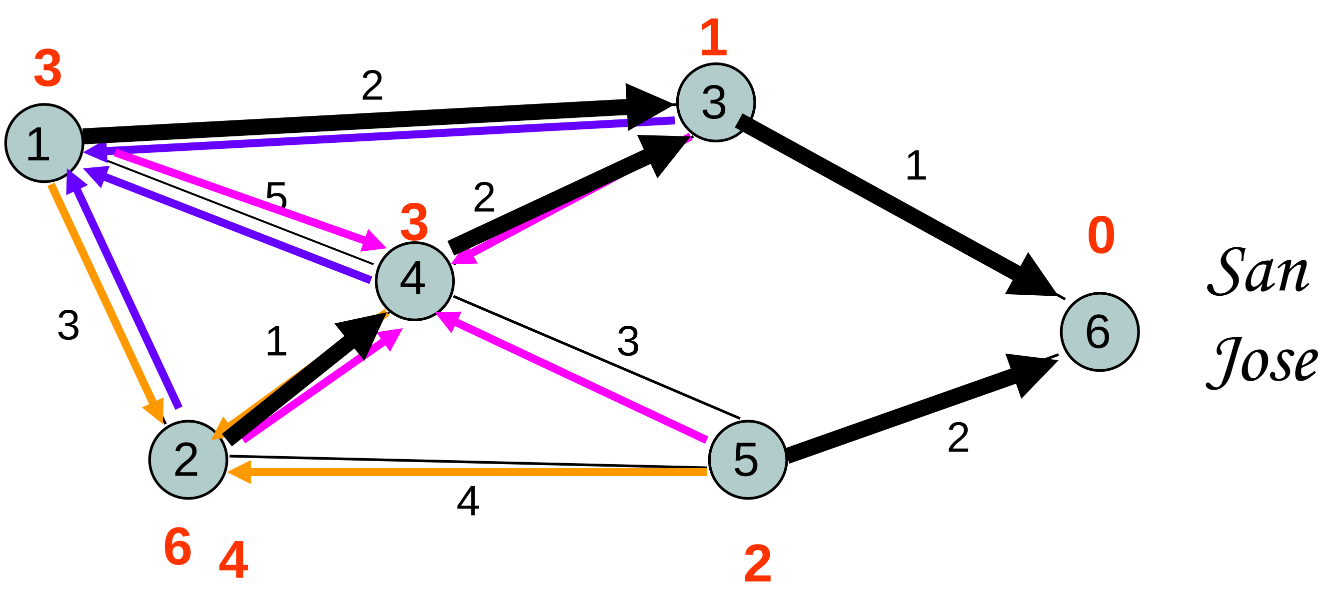
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6,1)	(-1, ∞)	(6,2)
2					
3					



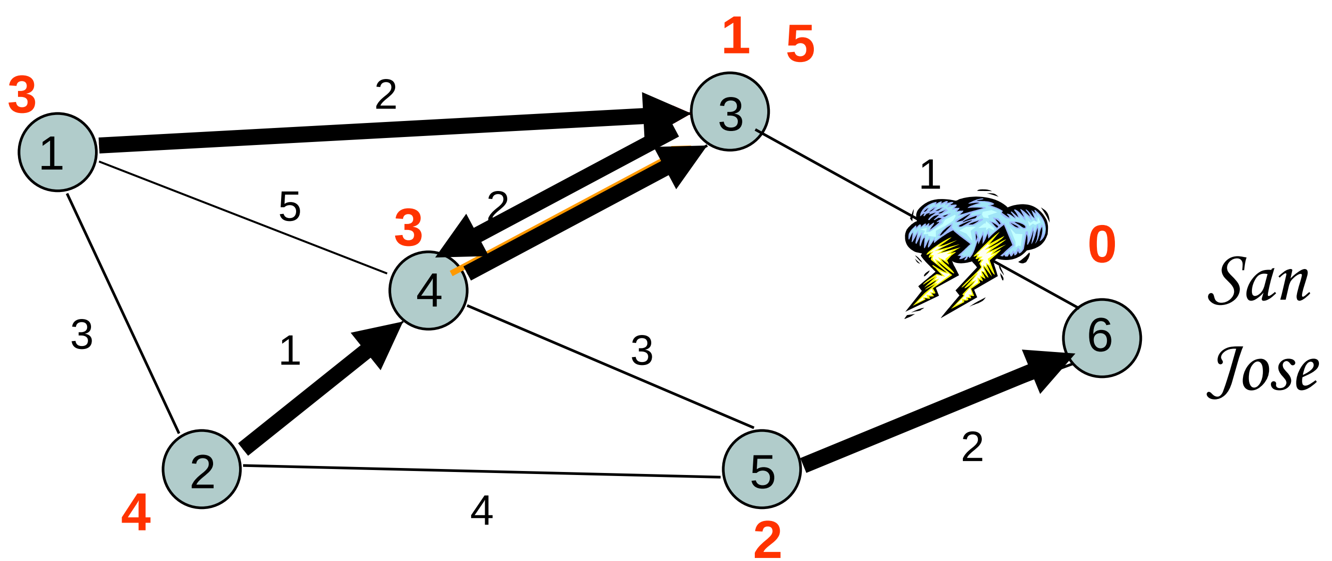
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6,2)
2	(3,3)	(5,6)	(6, 1)	(3,3)	(6,2)
3					



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)

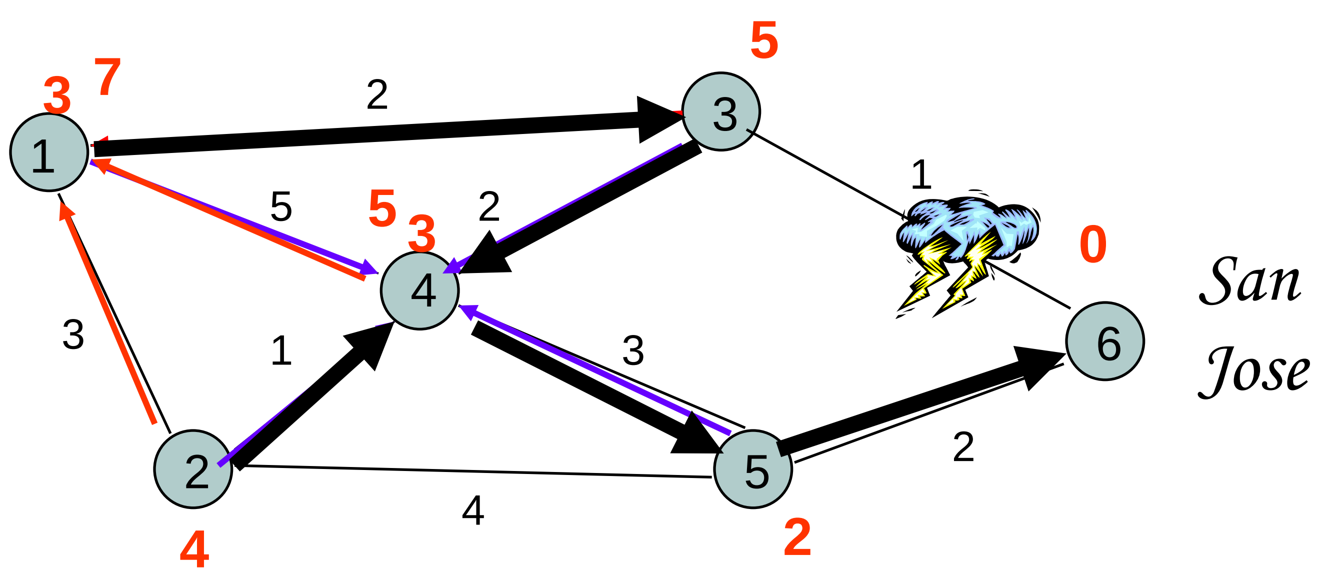


Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2					
3					



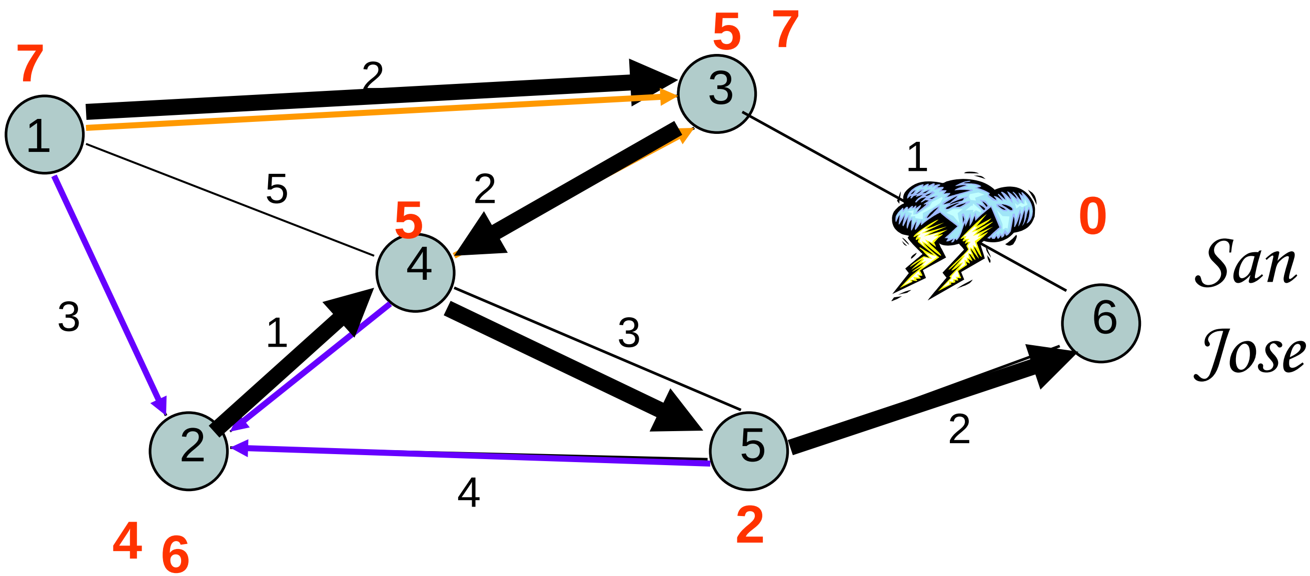
Network disconnected; Loop created between nodes 3 and 4

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					



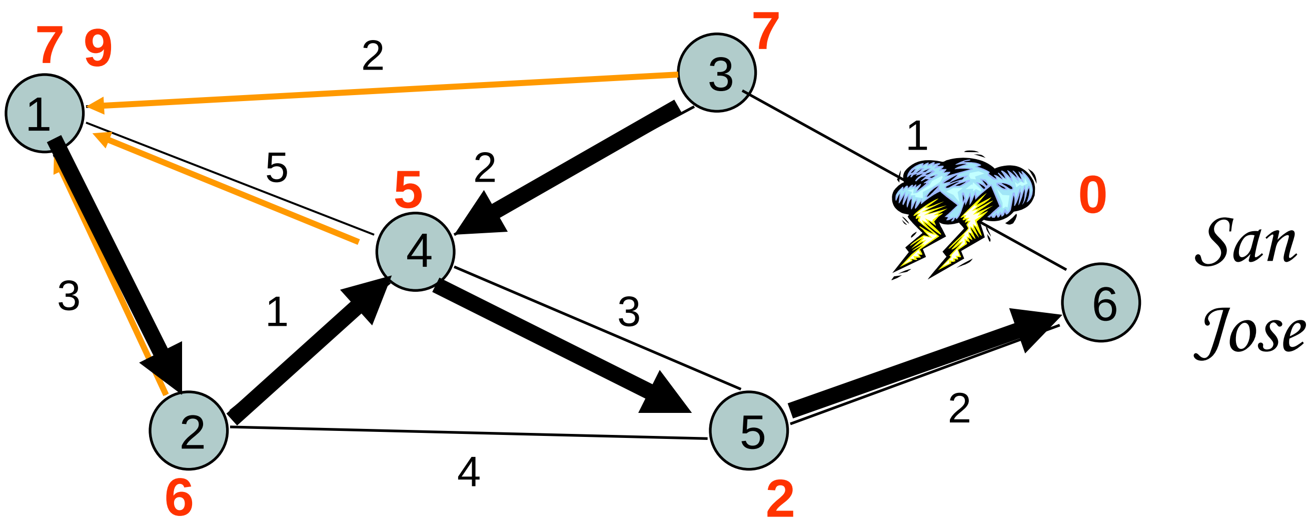
Node 4 could have chosen 2 as next node because of tie

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)



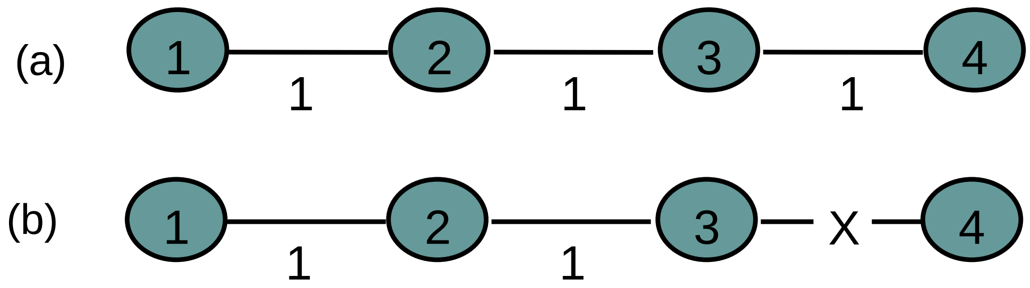
Node 2 could have chosen 5 as next node because of tie

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)

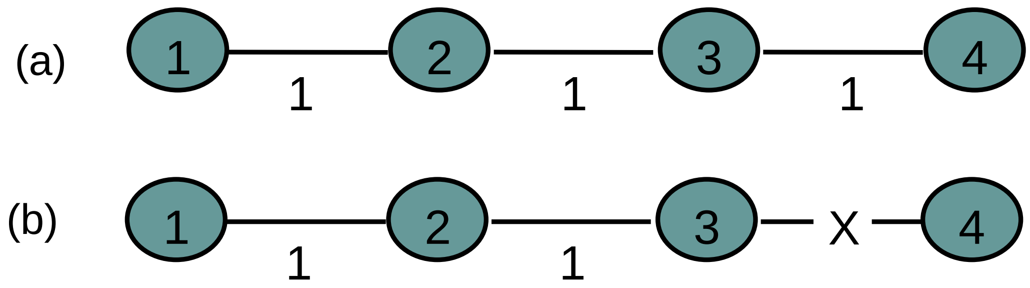
Update	Node 1	Node 2	Node 3
Before break	(2,3)	(3,2)	(4, 1)
After break	(2,3)	(3,2)	(2,3)
1	(2,3)	(3,4)	(2,3)
2	(2,5)	(3,4)	(2,5)
3	(2,5)	(3,6)	(2,5)
4	(2,7)	(3,6)	(2,7)
5	(2,7)	(3,8)	(2,7)
…	…	…	…

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 link state information

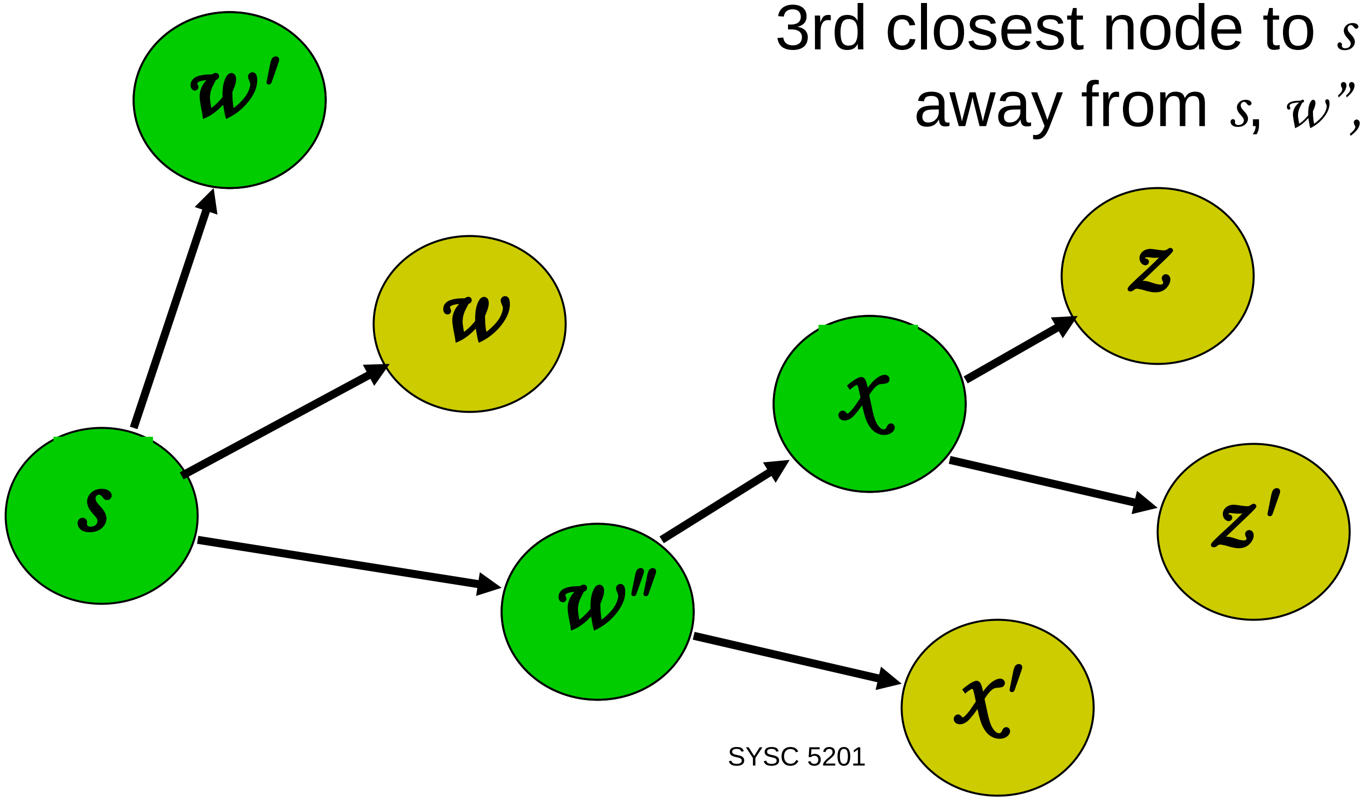
Dijkstra Algorithm: Finding shortest paths in order

Find shortest paths from source s to all other destinations

Closest node to s is 1 hop away

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

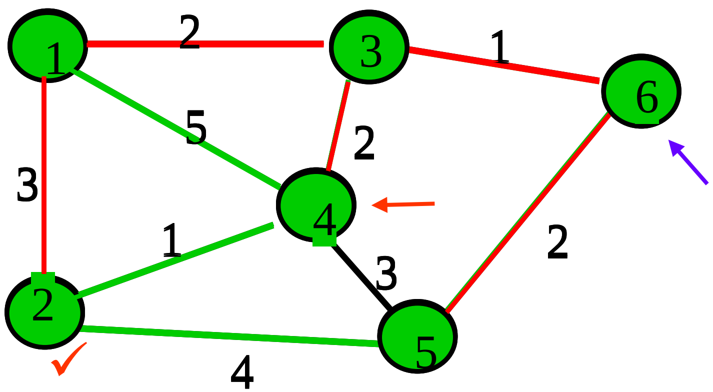
3rd closest node to s is 1 hop away from s, w'', or x

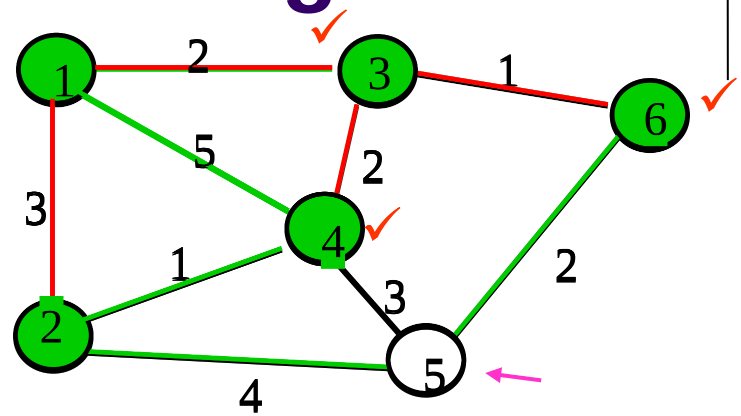


Dijkstra's algorithm

- *N*: set of nodes for which shortest path already found
- Initialization: (S*tart 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
  - $D_i = \min D_j$ for $j \notin N$
  - Add *i* to *N*
  - If *N* contains all the nodes, stop
- Step B: (*update minimum costs)*
  - For each node $j \notin N$
  - $D_j = \min (D_j, D_i + C_{ij})$ ← *Minimum distance from s to j through node **i** in N*
  - Go to Step A

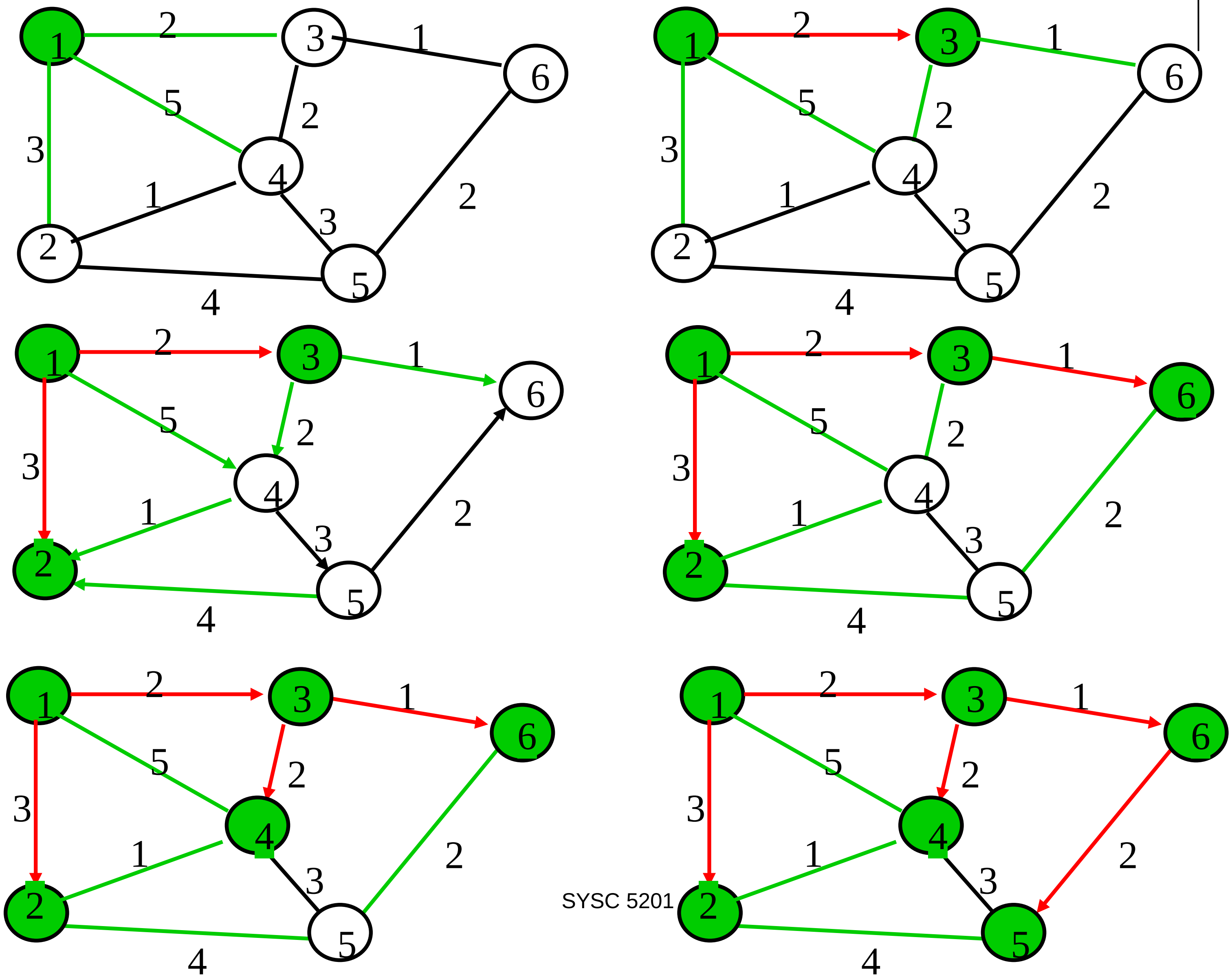
Execution of Dijkstra's algorithm





Iteration	**N**	D_2	D_3	D_4	D_5	D_6
Initial	{1}	3	2 ✓	5	∝	∝
1	{1,3}	3 ✓	2	4	∝	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 quickly (tens of seconds to minutes)
- 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

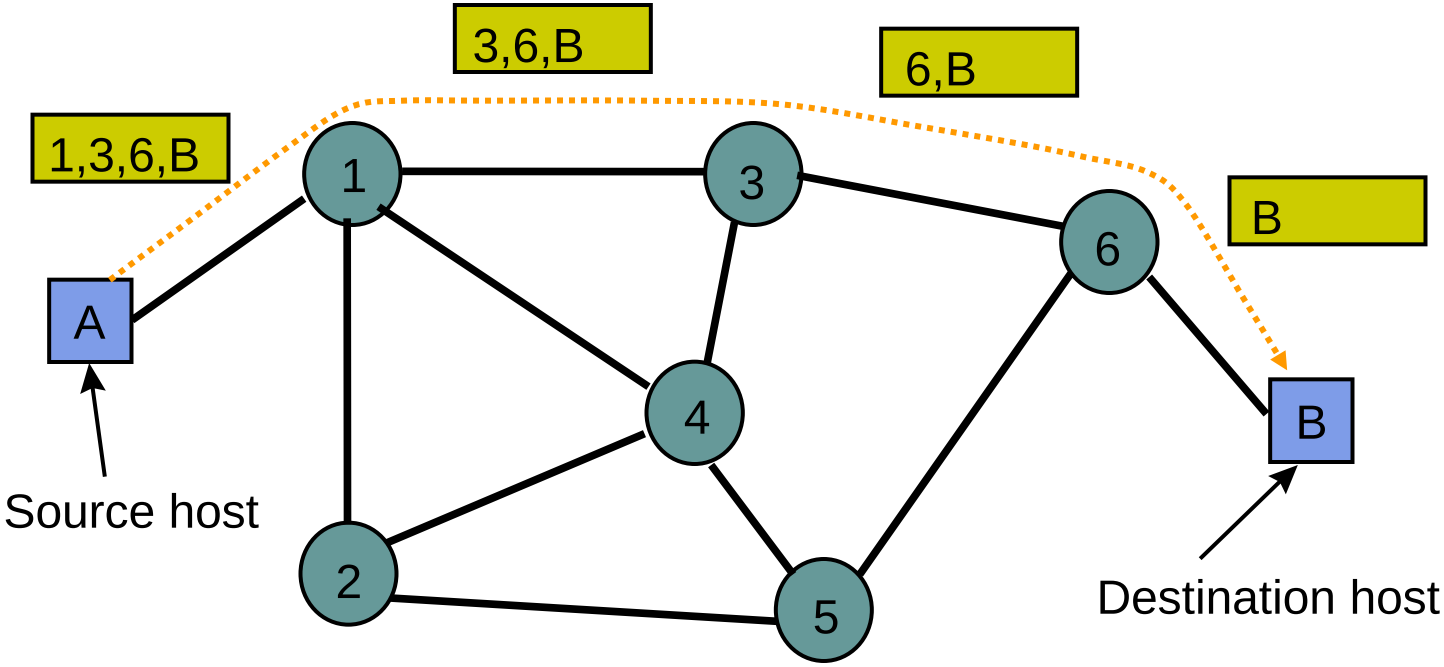
Source Routing

- Source host selects path that is to be followed by a packet
  - Strict: sequence of nodes in path inserted into header
- 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
  - Freedom comes with responsibility!
  - In practice, not supported by ISPs for customers.
  - Used for maintenance, e.g., traceroute, ICMP

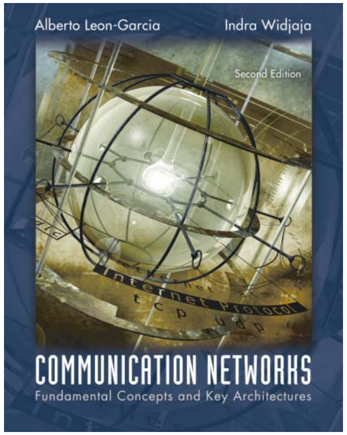
Pros: No Need for intermediate routers to maintain routing tables.

Cons: Burden at the source.

Example



Chapter 7
Packet-Switching Networks

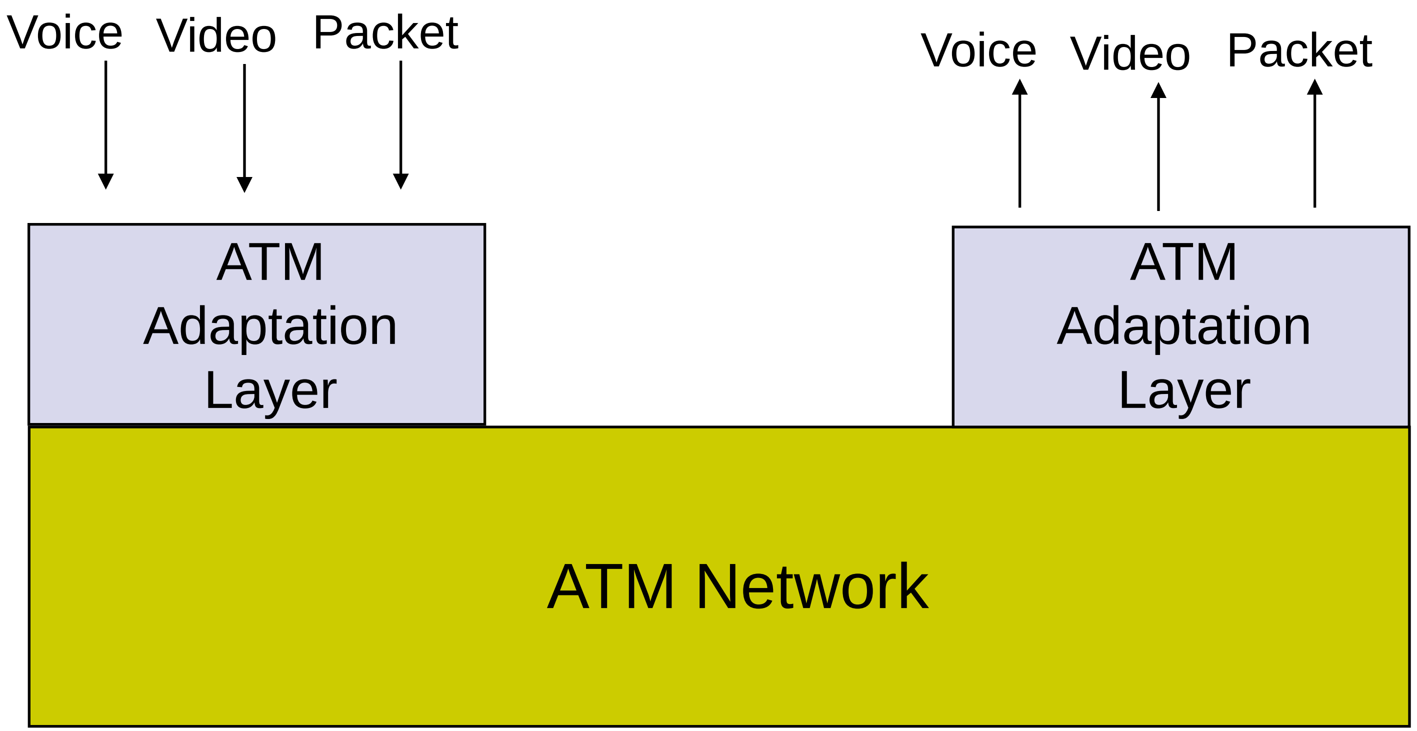


ATM Networks

Asynchronous Tranfer Mode (ATM)

- Packet multiplexing and switching
  - Fixed-length packets: “cells”
  - Connection-oriented
  - Rich Quality of Service support
- Conceived as end-to-end
  - Supporting wide range of services
    - Real time voice and video
    - Circuit emulation for digital transport
    - Data traffic with bandwidth guarantees
- Detailed discussion in Chapter 9

ATM Networking



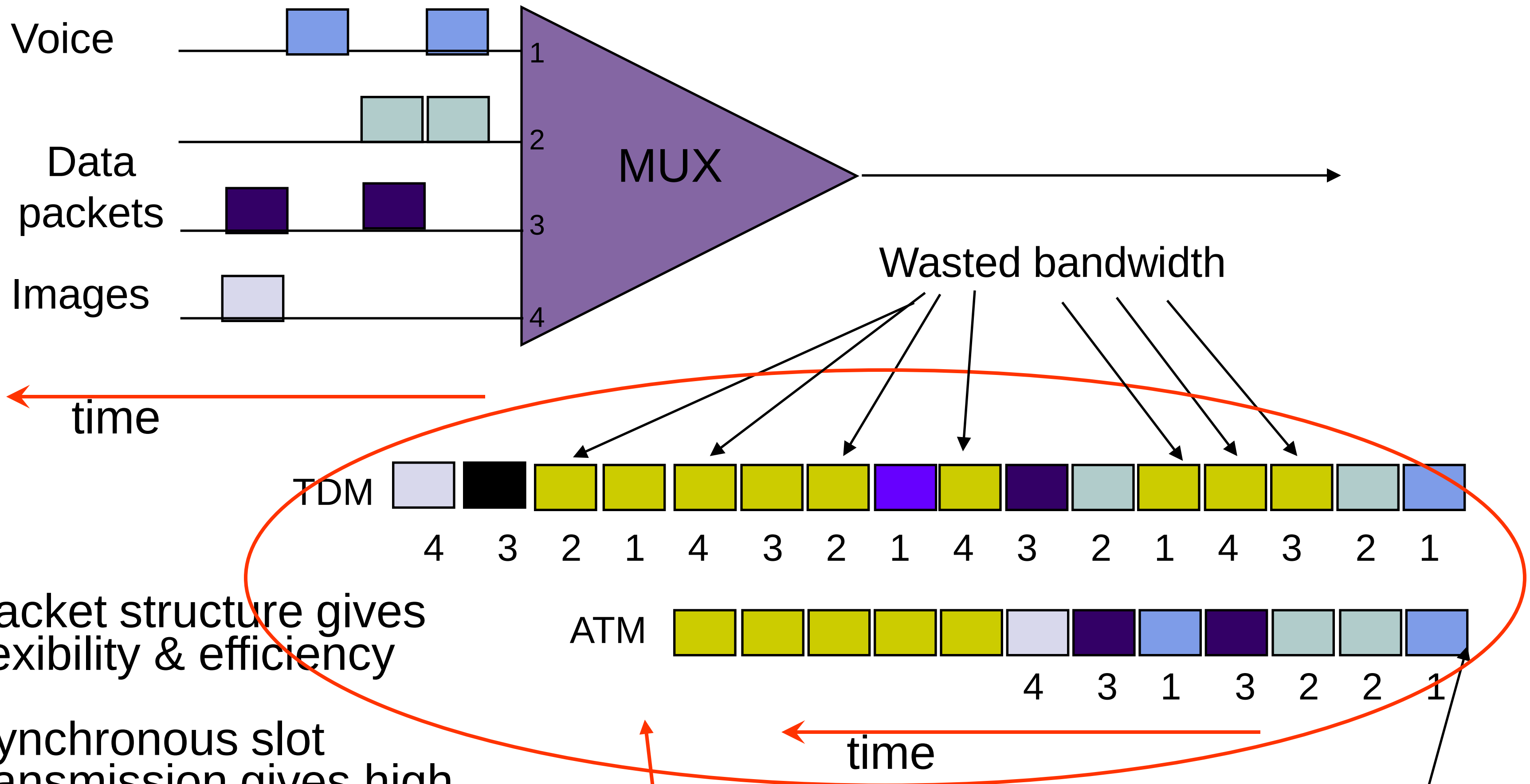
- End-to-end information transport using cells
- 53-byte cell (48bytes payload, 5bytes header), provide low delay and fine multiplexing granularity
- Support for many services through ATM Adaptation Layer

TDM vs. Packet Multiplexing

	Variable bit rate	Delay	Burst traffic	Processing
TDM	Multirate only	Low, fixed ✓	Inefficient	Minimal, very high speed
Packet	Easily handled ✓	Variable	Efficient ✓	Header & packet processing required *

In mid-1980s, packet processing mainly in software and hence slow. By late 1990s, very high speed packet processing possible. This is why ATM was promoted.

ATM: Attributes of TDM & Packet Switching

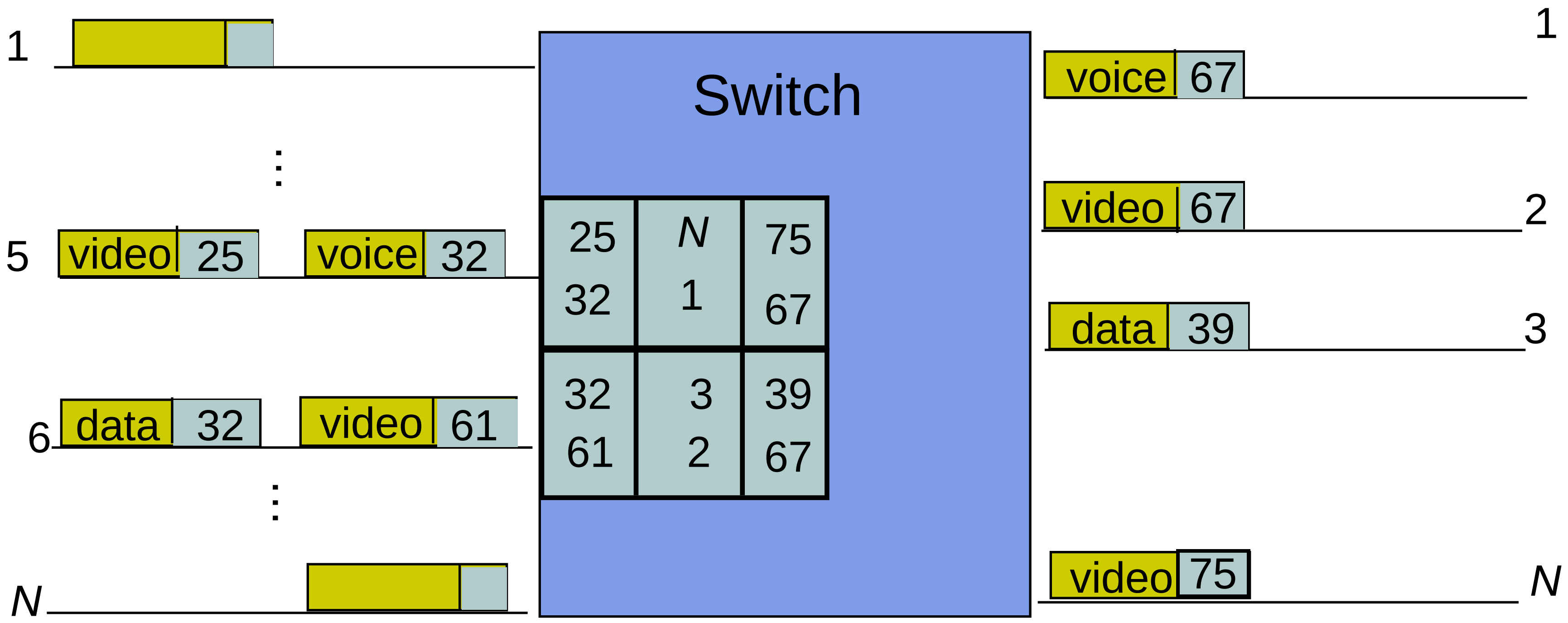


- Packet structure gives flexibility & efficiency
- Synchronous slot transmission gives high speed & density

Figure in book is inaccurate, no timing information is given.

ATM Switching

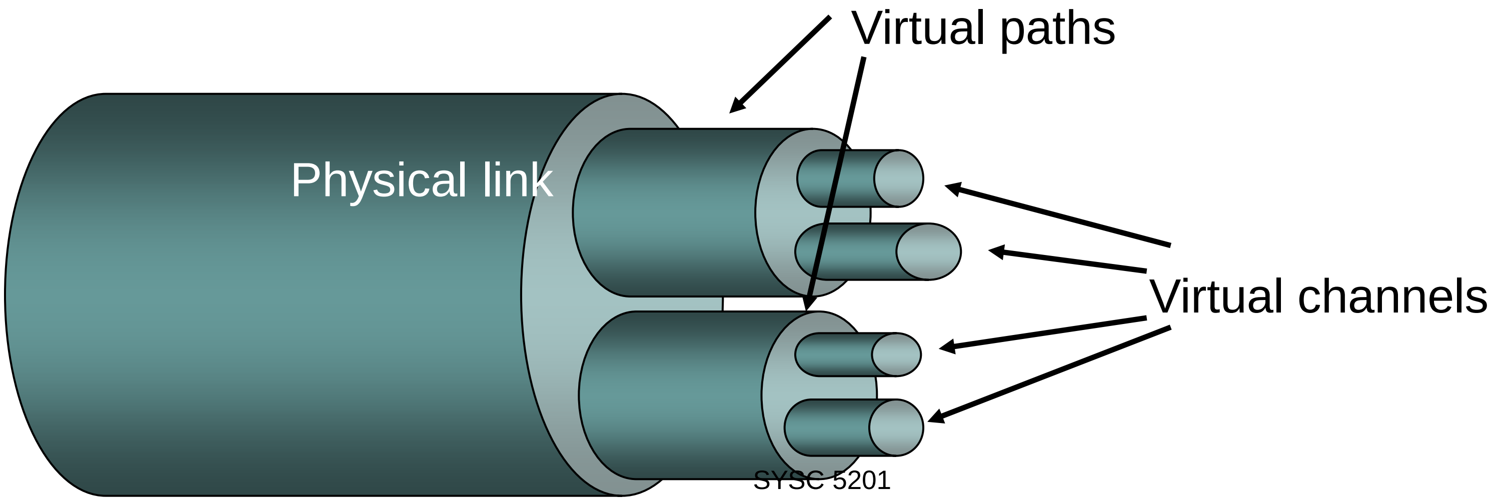
Switch carries out table translation and routing



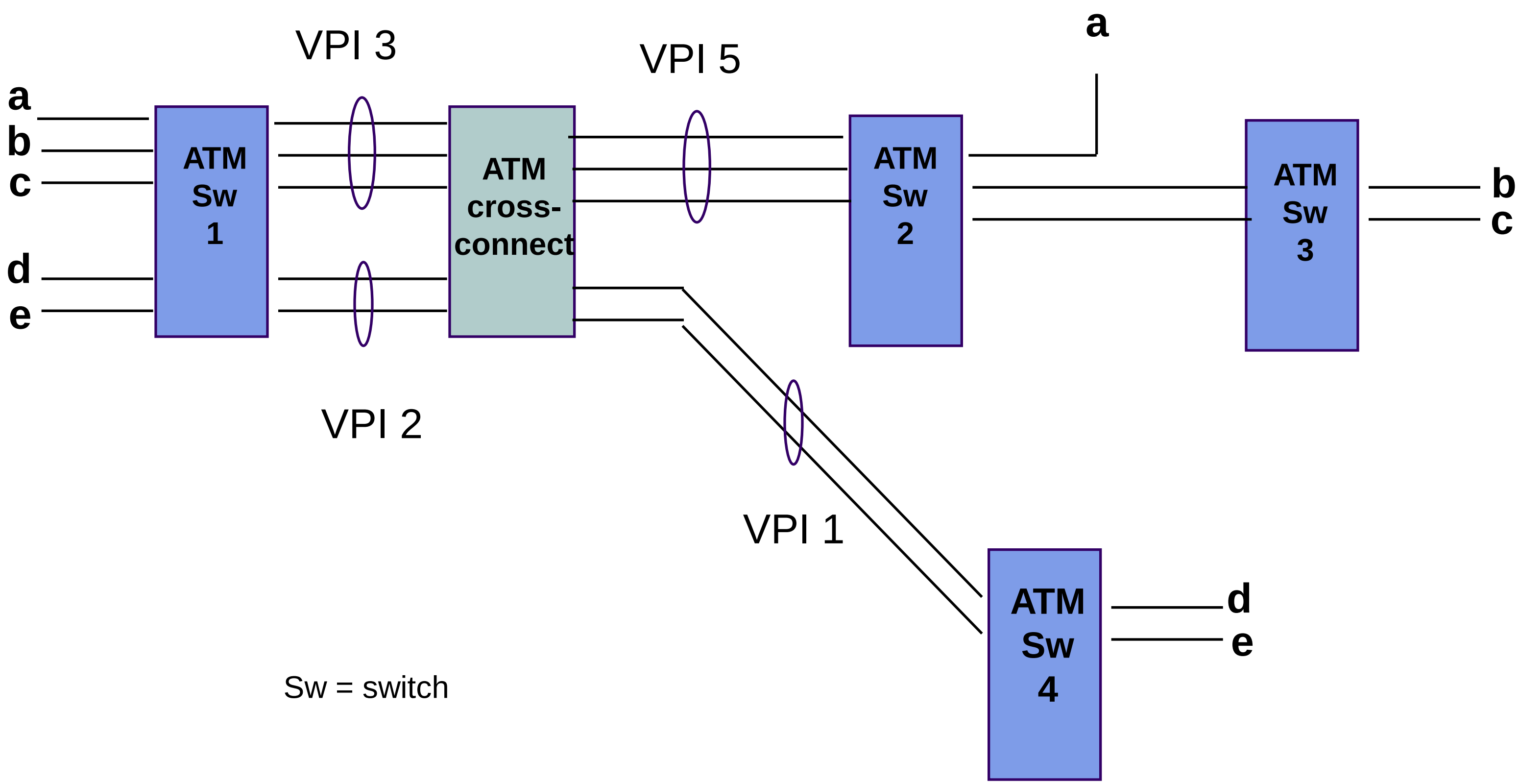
ATM switches can be implemented using shared memory, shared backplanes, or self-routing multi-stage fabrics

ATM Virtual Connections

- Virtual connections setup across network
- Connections identified by locally-defined tags
- ATM Header contains virtual connection information:
  - 8-bit Virtual Path Identifier
  - 16-bit Virtual Channel Identifier
- Powerful traffic grooming capabilities
  - Multiple VCs can be bundled within a VP
  - Similar to tributaries with SONET, except variable bit rates possible



VPI/VCI switching & multiplexing



- Connections a,b,c bundled into VP at switch 1
  - Crossconnect switches VP without looking at VCIs
  - VP unbundled at switch 2; VC switching thereafter
- VPI/VCI structure allows creation virtual networks

MPLS & ATM

- ATM initially touted as more scalable than packet switching
- ATM envisioned speeds of 150-600 Mbps
- Advances in optical transmission proved ATM to be the less scalable: @ 10 Gbps
  - Segmentation & reassembly of messages & streams into 48-byte cell payloads difficult & inefficient
  - Header must be processed every 53 bytes vs. 500 bytes on average for packets
  - Delay due to 1250 byte packet at 10 Gbps = 1 μsec; delay due to 53 byte cell @ 150 Mbps ≈ 3 μsec
- MPLS (Chapter 10) uses tags to transfer packets across virtual circuits in Internet