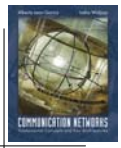
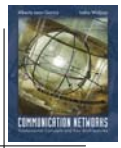
# Chapter 7 Packet-Switching Networks



Network Operation & Topology
Datagrams and Virtual Circuits
Structure of a Packet Switch
Routing in Packet Networks
Shortest Path Routing
ATM Networks

# Chapter 7 Packet-Switching Networks

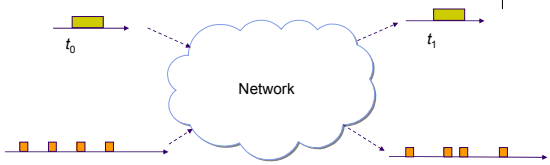


Overview: Network Services, Operation and Topology

## Network Layer

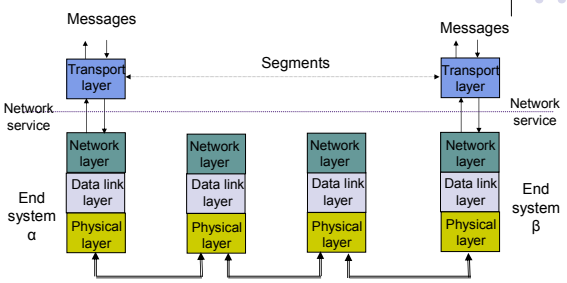
- Network Layer: the most complex layer
  - Requires the coordinated actions of multiple, geographically distributed network elements (switches & routers)
  - Must be able to deal with very large scales
    - Billions of users (people & communicating devices)
  - Biggest Challenges
    - Addressing: where should information be directed to?
    - Routing: what path should be used to get information there?

## Packet Switching




- Transfer of information as payload in data packets
- Packets undergo random delays & possible loss
- Different applications impose differing requirements on the transfer of information

## Network Service




- Network layer can offer a variety of services to transport layer
- Connection-oriented service or connectionless service
- Best-effort or delay/loss guarantees

## Network Service vs. Operation

Network Service
- Connectionless
  - Datagram Transfer
- Connection-Oriented
  - Reliable and possibly constant bit rate transfer

Internal Network Operation
- Connectionless
  - IP
- Connection-Oriented
  - Telephone connection
  - ATM

Various combinations are possible
- Connection-oriented service over Connectionless operation
- Connectionless service over Connection-Oriented operation
- Context & requirements determine what makes sense

## The End-to-End Argument for System Design

- An end-to-end function is best implemented at a higher level than at a lower level
  - End-to-end service requires all intermediate components to work properly
  - Higher-level better positioned to ensure correct operation
- Example: stream transfer service
  - Establishing an explicit connection for each stream across network requires all network elements (NEs) to be aware of connection; All NEs have to be involved in re-establishment of connections in case of network fault
  - In connectionless network operation, NEs do not deal with each explicit connection and hence are much simpler in design

## Network Layer Functions

Essential

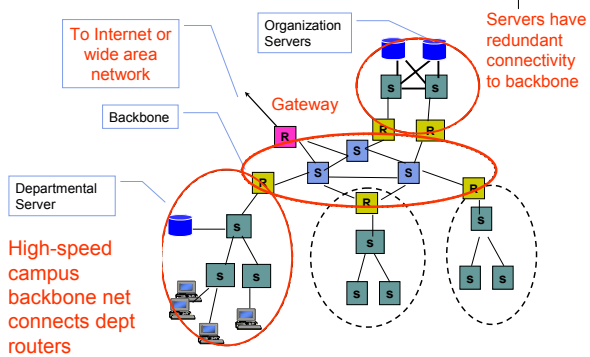
- **Routing**: mechanisms for determining the set of best paths for routing packets requires the collaboration of network elements
- **Forwarding**: transfer of packets from NE inputs to outputs
- **Priority & Scheduling**: determining order of packet transmission in each NE

Optional: congestion control, segmentation & reassembly, security

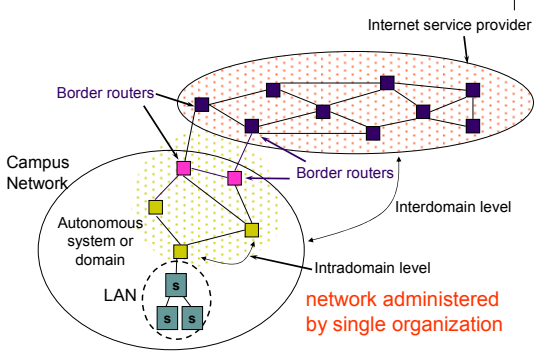
## End-to-End Packet Network

- Packet networks very different than telephone networks
- Individual packet streams are highly bursty
  - Statistical multiplexing is used to concentrate streams
- User demand can undergo dramatic change
  - Peer-to-peer applications stimulated huge growth in traffic volumes
- Internet structure highly decentralized
  - Paths traversed by packets can go through many networks controlled by different organizations
  - No single entity responsible for end-to-end service

## Example: Campus Network




## Connecting to The Internet Service Provider

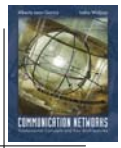



## Key Role of Routing

How to get packet from here to there?

- Decentralized nature of Internet makes routing a major challenge
  - Interior gateway protocols (IGPs) are used to determine routes within a domain
  - Exterior gateway protocols (EGPs) are used to determine routes across domains
  - Routes must be consistent & produce stable flows
- Scalability required to accommodate growth
  - Hierarchical structure of IP addresses essential to keeping size of routing tables manageable

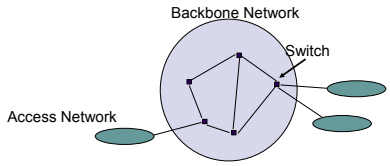
# Chapter 7
# Packet-Switching Networks



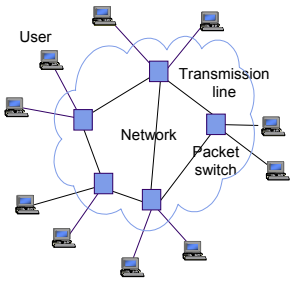
***Datagrams and Virtual Circuits***

## The Switching Function

- Dynamic interconnection of inputs to outputs
- Enables dynamic sharing of transmission resource
- Two fundamental approaches:
  - Connectionless
  - Connection-Oriented: Call setup control, Connection control




## Packet Switching Network



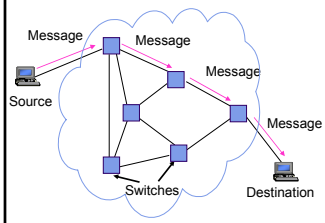

Packet switching network

- Transfers packets between users
- Transmission lines + packet switches (routers)
- Origin in message switching

Two modes of operation:

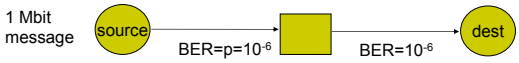
- Connectionless
- Virtual Circuit

## Message Switching




- Message switching invented for telegraphy
- Entire messages multiplexed onto shared lines, stored & forwarded
- Headers for source & destination addresses
- Routing at message switches
- Connectionless

## Long Messages vs. Packets




How many bits need to be transmitted to deliver message?

- Approach 1: send 1 Mbit message
- Probability message arrives correctly

$$P_c = (1-10^{-6})^{10^6} \approx e^{-10^6 10^{-6}} = e^{-1} \approx 1/3$$

- On average it takes about 3 transmissions/hop
- Total # bits transmitted ≈ 6 Mbits

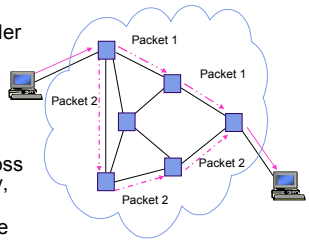
- Approach 2: send 10 100-kbit packets
- Probability packet arrives correctly

$$P'_c = (1-10^{-6})^{10^5} \approx e^{-10^5 10^{-6}} = e^{-0.1} \approx 0.9$$

- On average it takes about 1.1 transmissions/hop
- Total # bits transmitted ≈ 2.2 Mbits

## Packet Switching - Datagram

- Messages broken into smaller units (packets)
- Source & destination addresses in packet header
- Connectionless, packets routed independently (datagram)
- Packet may arrive out of order
- Pipelining of packets across network can reduce delay, increase throughput
- Lower delay that message switching, suitable for interactive traffic



## Routing Tables in Datagram Networks

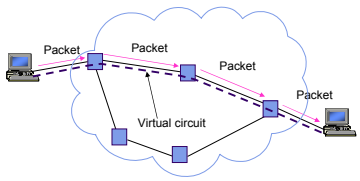
| Destination address | Output port |
|---|---|
| 0785 | 7 |
| 1345 | 12 |
| 1566 | 6 |
| 2458 | 12 |

- Route determined by table lookup
- Routing decision involves finding next hop in route to given destination
- Routing table has an entry for each destination specifying output port that leads to next hop
- Size of table becomes impractical for very large number of destinations

## Example: Internet Routing

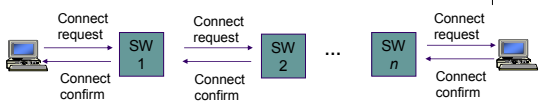
- Internet protocol uses datagram packet switching *across networks*
  - Networks are treated as data links
- Hosts have two-part IP address:
  - Network address + Host address
- Routers do table lookup on network address
  - This reduces size of routing table
- In addition, network addresses are assigned so that they can also be aggregated
  - Discussed as CIDR in Chapter 8

## Packet Switching – Virtual Circuit




- Call set-up phase sets ups pointers in fixed path along network
- All packets for a connection follow the same path
- Abbreviated header identifies connection on each link
- Packets queue for transmission
- Variable bit rates possible, negotiated during call set-up
- Delays variable, cannot be less than circuit switching

## Connection Setup



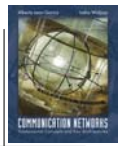

- Signaling messages propagate as route is selected
- Signaling messages identify connection and setup tables in switches
- Typically a connection is identified by a local tag, Virtual Circuit Identifier (VCI)
- Each switch only needs to know how to relate an incoming tag in one input to an outgoing tag in the corresponding output
- Once tables are setup, packets can flow along path

## Virtual Circuit Forwarding Tables

| Input VCI | Output port | Output VCI |
|---|---|---|
| 12 | 13 | 44 |
| 15 | 15 | 23 |
| 27 | 13 | 16 |
| 58 | 7 | 34 |

- Each input port of packet switch has a forwarding table
- Lookup entry for VCI of incoming packet
- Determine output port (next hop) and insert VCI for next link
- Very high speeds are possible
- Table can also include priority or other information about how packet should be treated

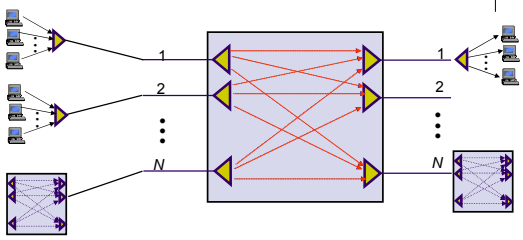
# Chapter 7
# Packet-Switching Networks



*Datagrams and Virtual Circuits*

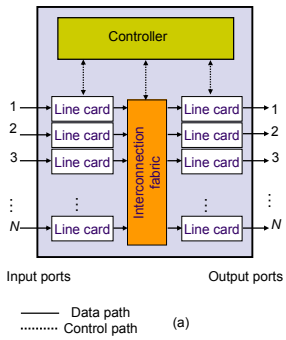
***Structure of a Packet Switch***

## Packet Switch: Intersection where Traffic Flows Meet




- Inputs contain multiplexed flows from access muxs & other packet switches
- Flows demultiplexed at input, routed and/or forwarded to output ports
- Packets buffered, prioritized, and multiplexed on output lines

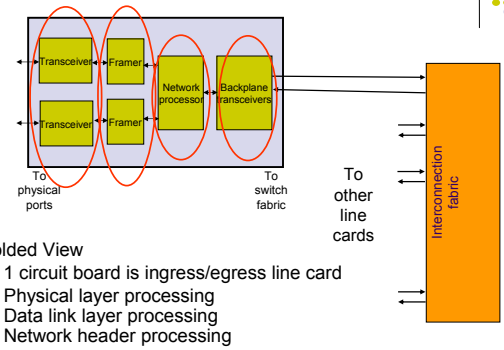
## Generic Packet Switch




"Unfolded" View of Switch

- Ingress Line Cards
  - Header processing
  - Demultiplexing
  - Routing in large switches
- Controller
  - Routing in small switches
  - Signalling & resource allocation
- Interconnection Fabric
  - Transfer packets between line cards
- Egress Line Cards
  - Scheduling & priority
  - Multiplexing

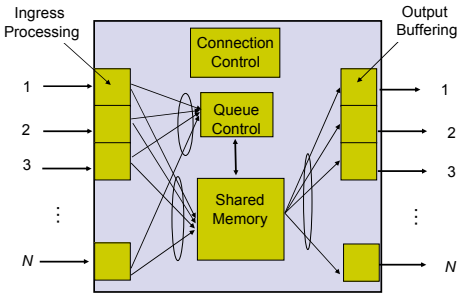
## Line Cards




Folded View

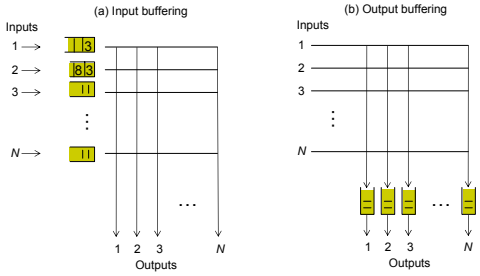
- 1 circuit board is ingress/egress line card
- Physical layer processing
- Data link layer processing
- Network header processing
- Physical layer across fabric + framing

## Shared Memory Packet Switch



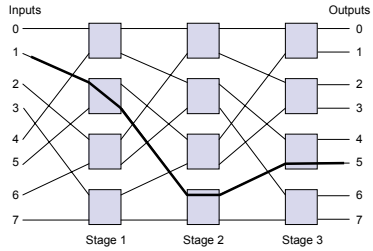

Small switches can be built by reading/writing into shared memory

## Crossbar Switches



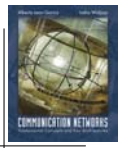

- Large switches built from crossbar & multistage space switches
- Requires centralized controller/scheduler (who sends to whom when)
- Can buffer at input, output, or both (performance vs complexity)

## Self-Routing Switches




- Self-routing switches do not require controller
- Output port number determines route
- 101 → (1) lower port, (2) upper port, (3) lower port

# Chapter 7
# Packet-Switching Networks

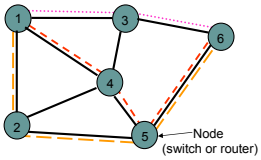



***Routing in Packet Networks***

## Desirable Routing Features

- Rapid and accurate delivery of packets
- Adaptability to network topology changes
- Adaptability to traffic load conditions
- Capability to discover network connectivity
- Capability to avoid loops
- Scalability (to larger networks)
- Low overhead (messaging) cost

## Choosing a Route in Packet Networks




- Three possible (loopfree) routes from 1 to 6:
  - 1-3-6, 1-4-5-6, 1-2-5-6
- Which one is "best"?
  - Use some routing metric: Min delay? Min # hops? 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

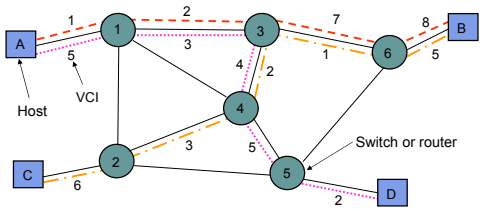
## Centralized vs Distributed Routing

- Centralized Routing
  - All routes determined by a central node
  - All state information sent to central node
  - Problems adapting to frequent topology changes
  - Does not scale
- Distributed Routing
  - Routes determined by routers using distributed algorithm
  - State information exchanged by routers
  - Adapts to topology and other changes
  - Better scalability

## Static vs Dynamic Routing

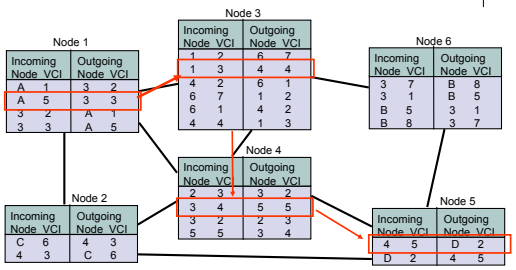
- Static Routing
  - Set up manually, do not change; requires administration
  - Works when traffic predictable & network is simple
  - Used to override some routes set by dynamic algorithm
  - Used to provide default router
- Dynamic Routing
  - Adapt to changes in network conditions
  - Automated
  - Calculates routes based on received updated network state information

## Routing in Virtual-Circuit Packet Networks



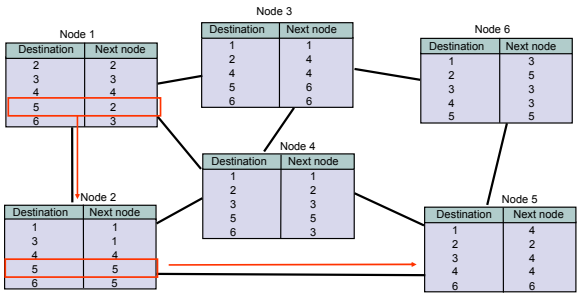

- Route determined during connection setup
- Tables in switches implement forwarding that realizes selected route

## Routing Tables in VC Packet Networks

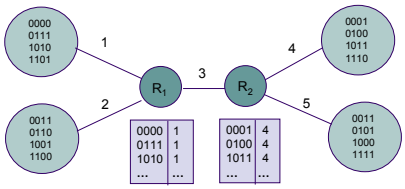



- Example: VCI from A to D
  - From A & VCI 5 → 3 & VCI 3 → 4 & VCI 4
  - → 5 & VCI 5 → D & VCI 2

## Routing Tables in Datagram Packet Networks

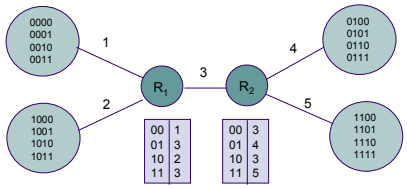



## Non-Hierarchical Addresses and Routing




- 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

## Flat vs Hierarchical Routing

- Flat Routing
  - All routers are peers
  - Does not scale
- Hierarchical Routing
  - Partitioning: Domains, autonomous systems, areas...
  - Some routers part of routing backbone
  - Some routers only communicate within an area
  - Efficient because it matches typical traffic flow patterns
  - Scales

## Specialized Routing

- Flooding
  - Useful in starting up network
  - Useful in propagating information to all nodes
- Deflection Routing
  - Fixed, preset routing procedure
  - No route synthesis

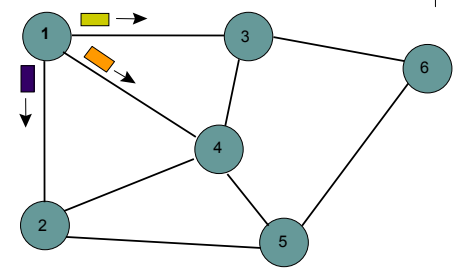
## Flooding

Send a packet to all nodes in a network

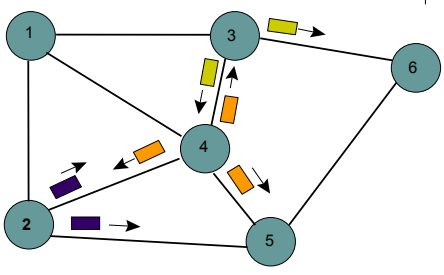
- No routing tables available
- Need to broadcast packet to all nodes (e.g. to propagate link state information)

Approach

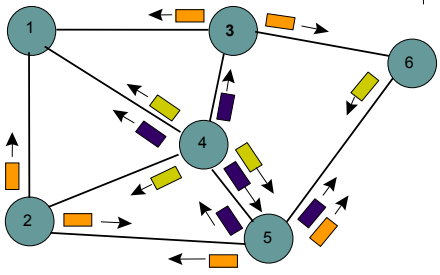
- Send packet on all ports except one from where it arrived
- Exponential growth in packet transmissions




Flooding is initiated from Node 1: Hop 1 transmissions




Flooding is initiated from Node 1: Hop 2 transmissions



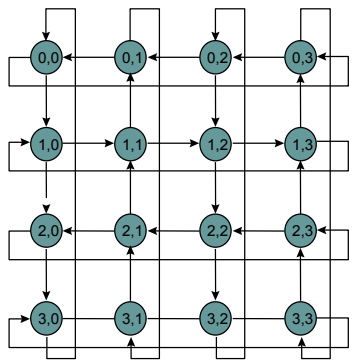

Flooding is initiated from Node 1: Hop 3 transmissions

## 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

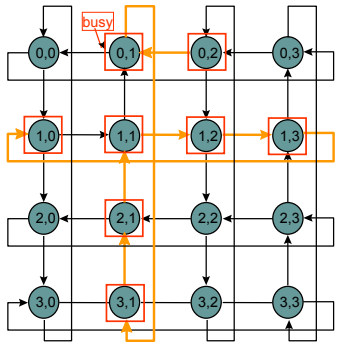
## Deflection Routing

- 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)




# Chapter 7
# Packet-Switching Networks

***Shortest Path Routing***


## Shortest Path Routing

- Many possible paths connect any given source 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 formulated 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
- Bellman-Ford (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 Approach: *Do you know the way to San Jose?*


## Distance Vector Concept

*Local Signpost*

- Direction
- Distance

*Routing Table*

For each destination list:

- Next Node
- Distance

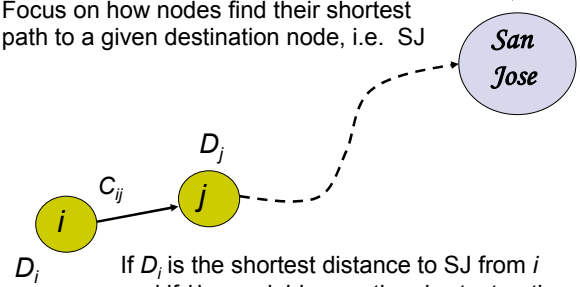
| dest | next | dist |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

*Table Synthesis*

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

## Shortest Path to SJ

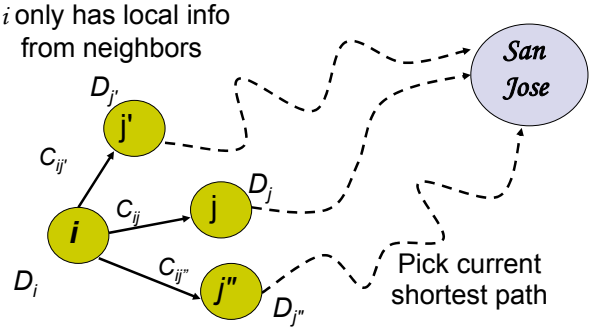
Focus on how nodes find their shortest path to a given destination node, i.e. SJ




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

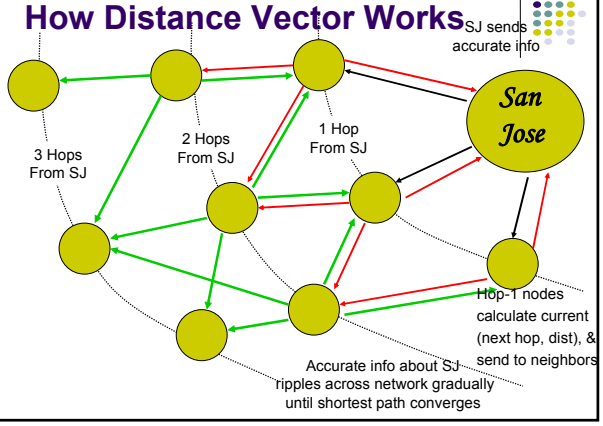
## Search for the shortest paths

*i* only has local info from neighbors




Pick current shortest path

# How 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

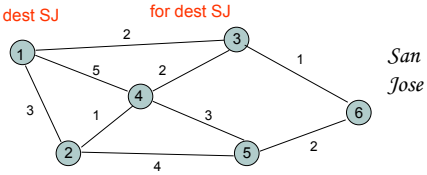
# Bellman-Ford Algorithm

- *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: $D_j(d)= \infty$, 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 $(n_j, D_j(d))$ by new $(n_j^*, D_j^*(d))$ if new next node or distance found
  - Go to send step

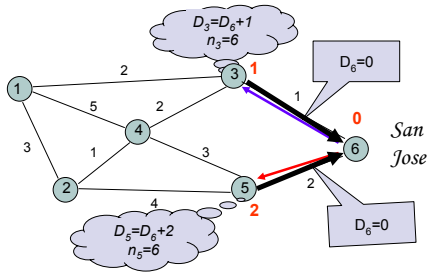
| Iteration | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 |
|---|---|---|---|---|---|
| Initial | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |

Table entry
@ node 1
for dest SJ

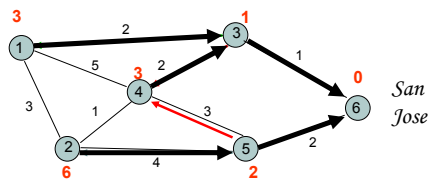
Table entry
@ node 3
for dest SJ



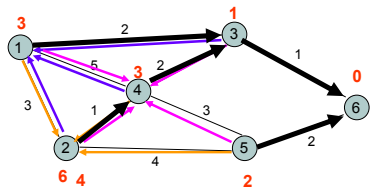

| Iteration | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 |
|---|---|---|---|---|---|
| Initial | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) |
| 1 | (-1, ∞) | (-1, ∞) | (6,1) | (-1, ∞) | (6,2) |
| 2 | | | | | |
| 3 | | | | | |



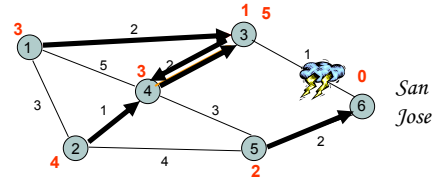

| Iteration | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 |
|---|---|---|---|---|---|
| Initial | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) |
| 1 | (-1, ∞) | (-1, ∞) | (6, 1) | (-1, ∞) | (6,2) |
| 2 | (3,3) | (5,6) | (6, 1) | (3,3) | (6,2) |
| 3 | | | | | |



| 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) |

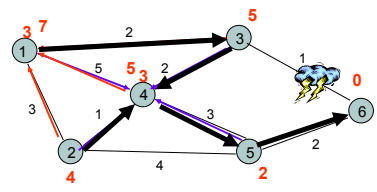



| Iteration | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 |
|---|---|---|---|---|---|
| Initial | (3,3) | (4,4) | (6, 1) | (3,3) | (6,2) |
| 1 | (3,3) | (4,4) | (4, 5) | (3,3) | (6,2) |
| 2 | | | | | |
| 3 | | | | | |



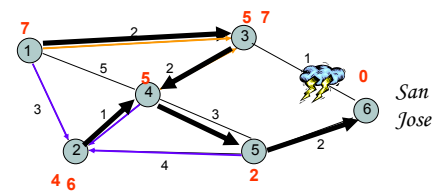

*Network disconnected; Loop created between nodes 3 and 4*

| 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 | | | | | |



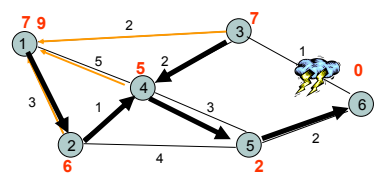

Node 4 could have chosen 2 as next node because of tie

| 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) |



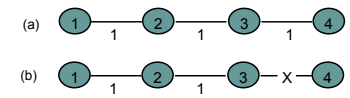

Node 2 could have chosen 5 as next node because of tie

| 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)

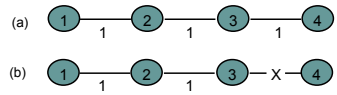
| Update | Node 1 | Node 2 | Node 3 |
|---|---|---|---|
| Before break | (2,3) | (3,2) | (4, 1) |
| After break | (2,3) | (3,2) | (2,3) |
| 1 | (2,3) | (3,4) | (2,3) |
| 2 | (2,5) | (3,4) | (2,5) |
| 3 | (2,5) | (3,6) | (2,5) |
| 4 | (2,7) | (3,6) | (2,7) |
| 5 | (2,7) | (3,8) | (2,7) |
| … | … | … |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

## 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 Approach

- 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

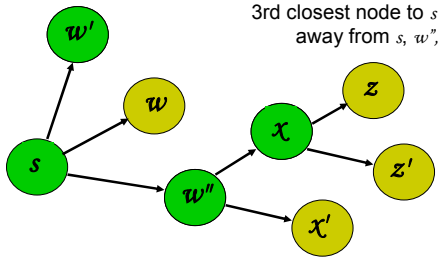
## Dijkstra Algorithm: Finding shortest paths in order

Find shortest paths from source s to all other destinations

Closest node to $s$ is 1 hop away

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

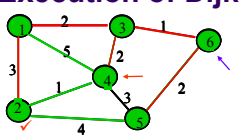
3rd closest node to $s$ is 1 hop away from $s$, $w$", or $x$

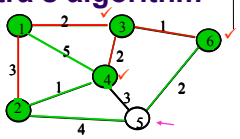



## Dijkstra's algorithm

- *N*: set of nodes for which shortest path already found
- Initialization: (*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 ($D_j$, $D_i$+$C_{ij}$) ⟸ *Minimum distance from s to j through node **i** in N*
  - Go to Step A

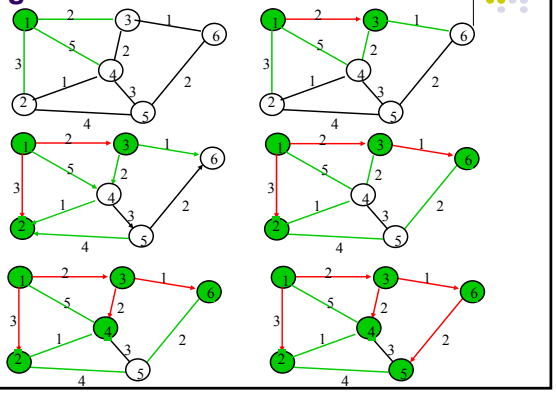
## Execution of Dijkstra's algorithm




| 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



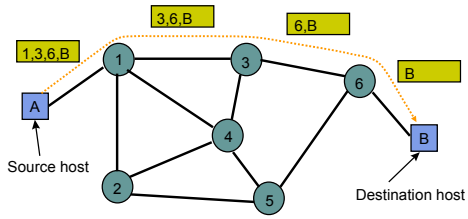
## Reaction to Failure

- 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 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

## Another Approach: Source Routing

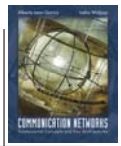
- Source host selects path that is to be followed by a packet
  - Strict: sequence of nodes in path inserted into header
  - Loose: subsequence of nodes in path specified
- Intermediate switches read next-hop address and remove address
- Source host needs link state information or access to a route server
- 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




# Chapter 7
# Packet-Switching Networks

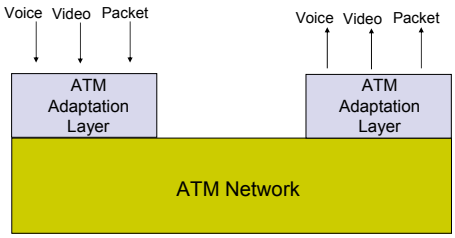
***Overview of ATM Networks***



## Asynchronous Transfer 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



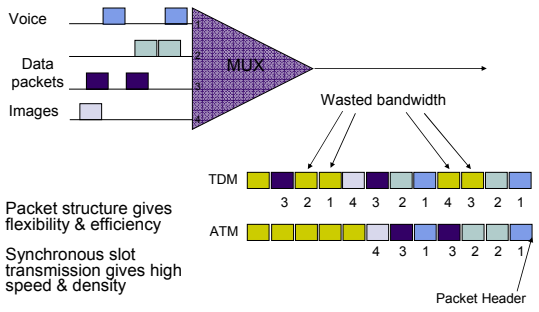

- End-to-end information transport using cells
- 53-byte cell provide low delay and fine multiplexing granularity
- Support for many services through ATM Adaptation Layer

# 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

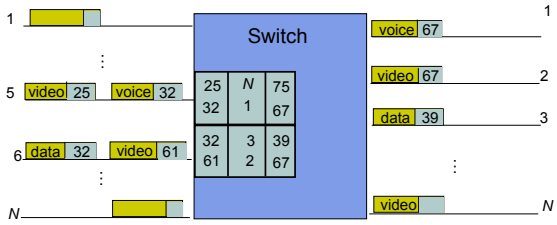
# ATM: Attributes of TDM & Packet Switching




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

# ATM Switching

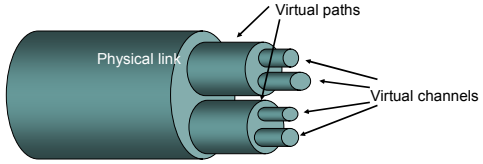
Switch carries out table translation and routing



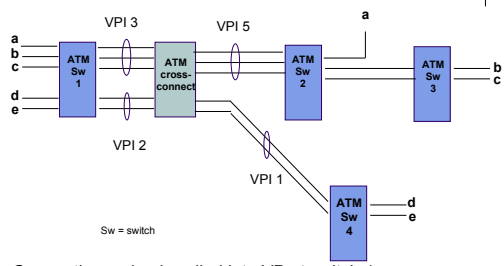

ATM switches can be implemented using shared memory, shared backplanes, or self-routing multi-stage fabrics

# 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 of virtual networks