# Computer Networks

## *Chapter 5: Network Layer*

**Prof. Xudong Wang**

**Wireless Networking and Artificial Intelligence Lab**
**UM-SJTU Joint Institute**
**Shanghai Jiao Tong University**
**Shanghai, China**
**http://wanglab.sjtu.edu.cn**

# Outline

- Network Layer Services
- Major Network Layer Functions
  - Routing
  - Forwarding
  - Prioritizing and scheduling – packet-level traffic management
  - Congestion control – flow-level traffic management
    - Open-loop: admission control, policing, traffic shaping
    - Close-loop: flow control
- Protocols in the Network Layer
  - Internet protocol (IP)
  - ATM

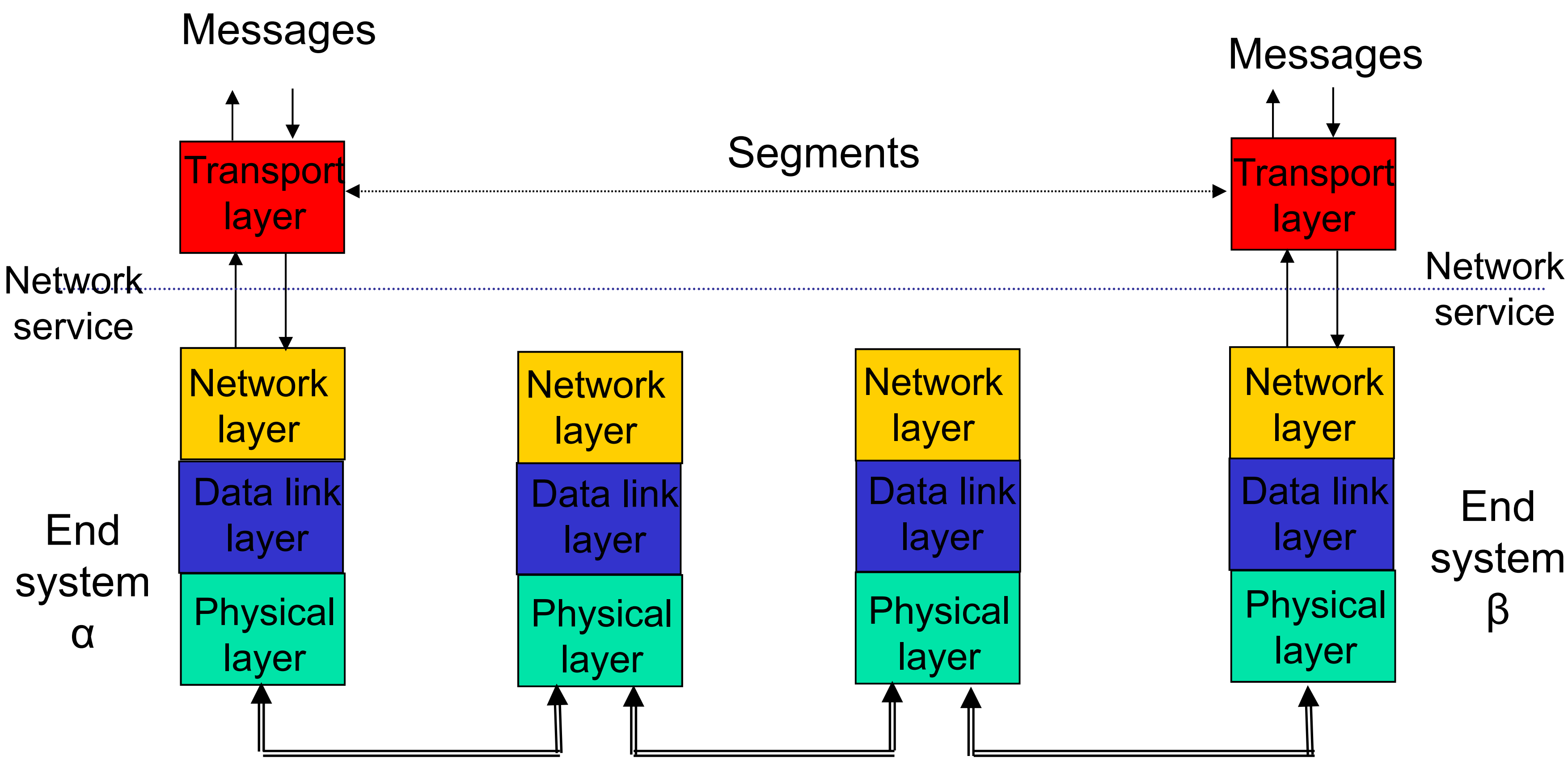
# Network Layer Services

# Network Layer Challenges

- Network Layer: the complicated 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?

# 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



# 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

# End-to-End Packet Network

- Packet networks are 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

# Network Layer Functions

- **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

# Routing in the Network Layer

# Key Roles 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

# 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

# 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 than message switching, suitable for interactive traffic

# Routing Tables in Datagram Networks

- 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

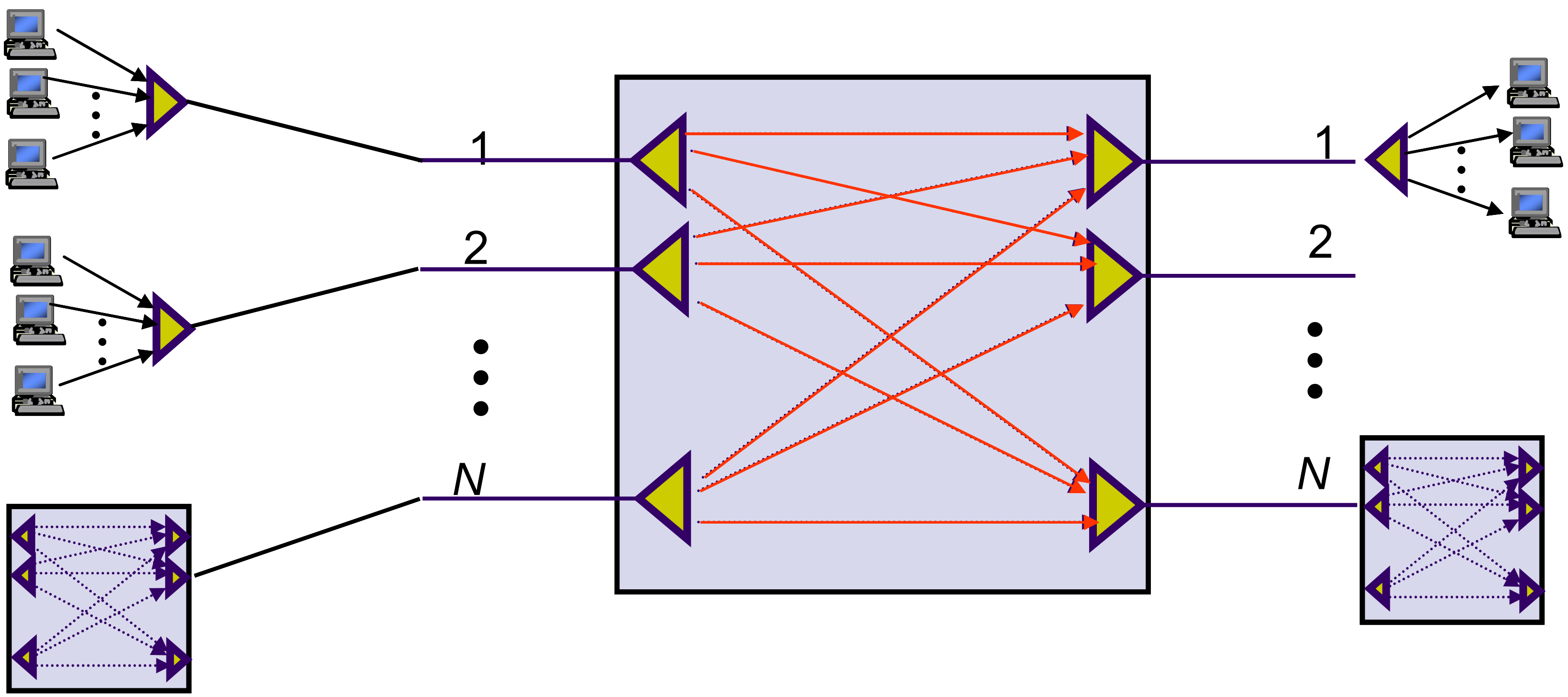
| Destination address | Output port |
|---|---|
| | |
| 0785 | 7 |
| | |
| 1345 | 12 |
| | |
| 1566 | 6 |
| | |
| 2458 | 12 |
| | |

# 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

# Packet Switch: Intersection where Traffic Flows Meet

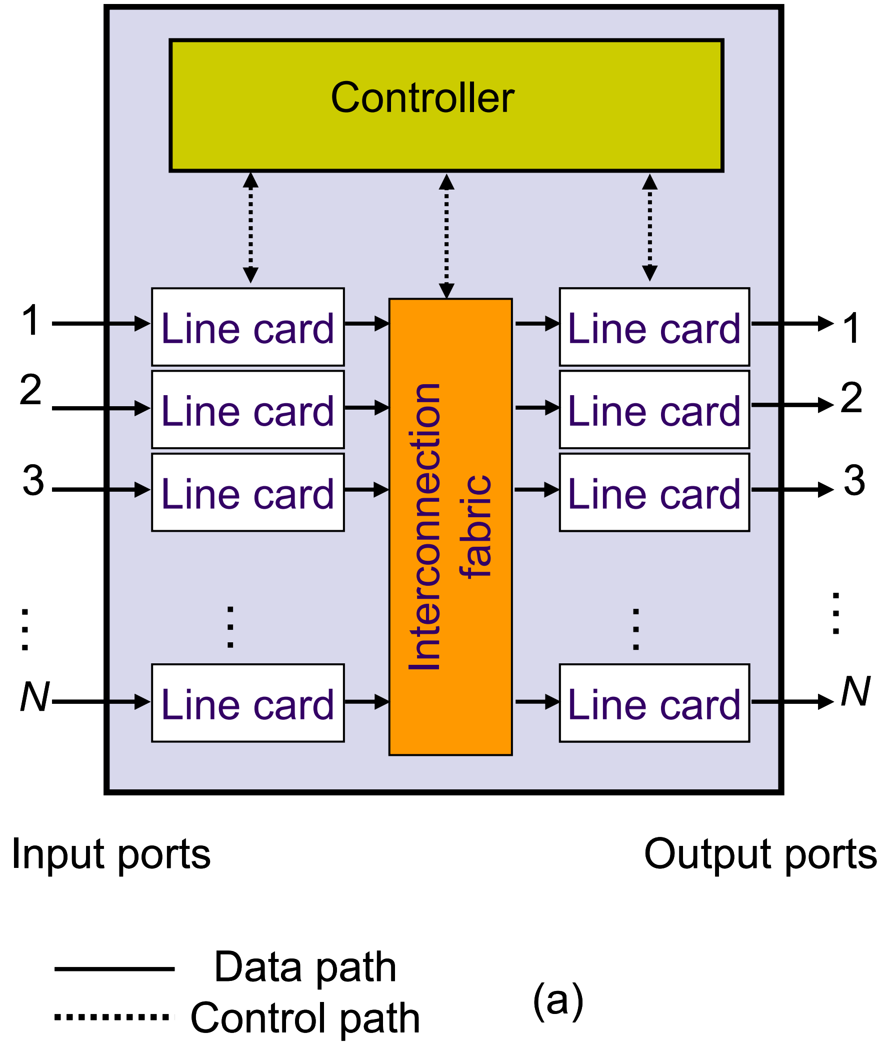
- Inputs contain multiplexed flows from access muxs & other packet switches
- Flows demultiplexed at input, routed and/or forwarded to output ports
- Packets buffered, prioritized, and multiplexed on output lines



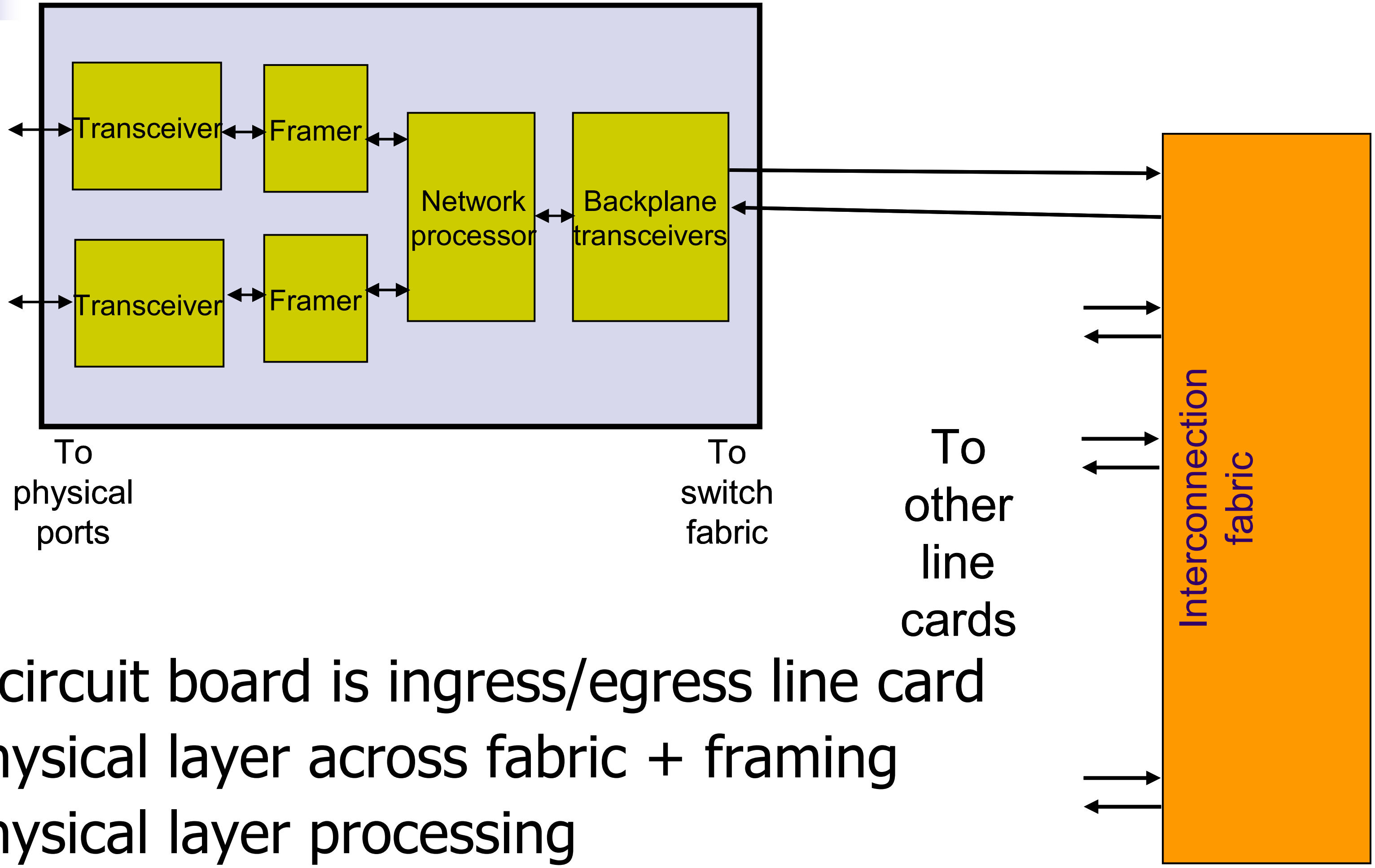
# Generic Packet Switch

"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

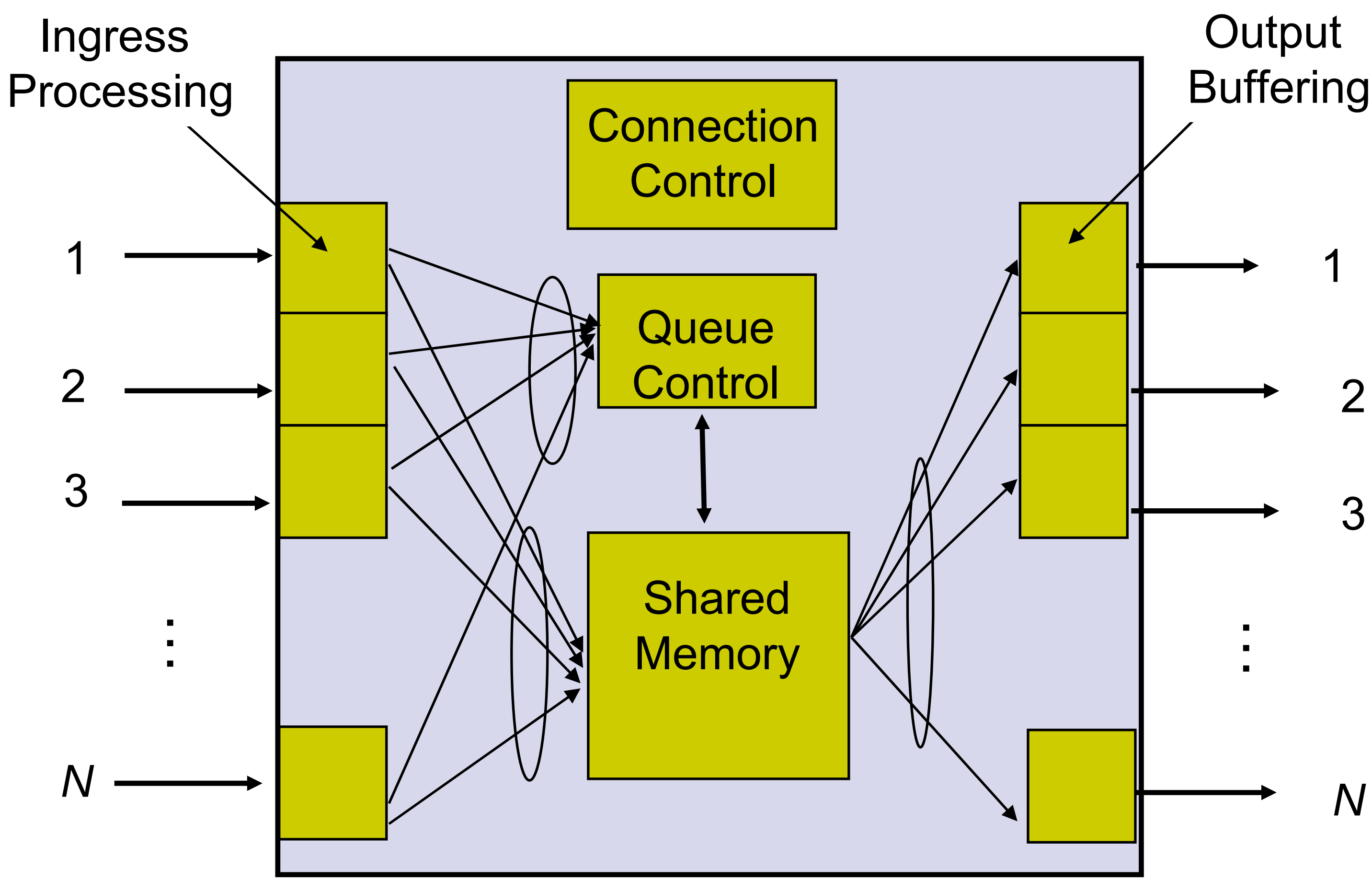


# Line Cards




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

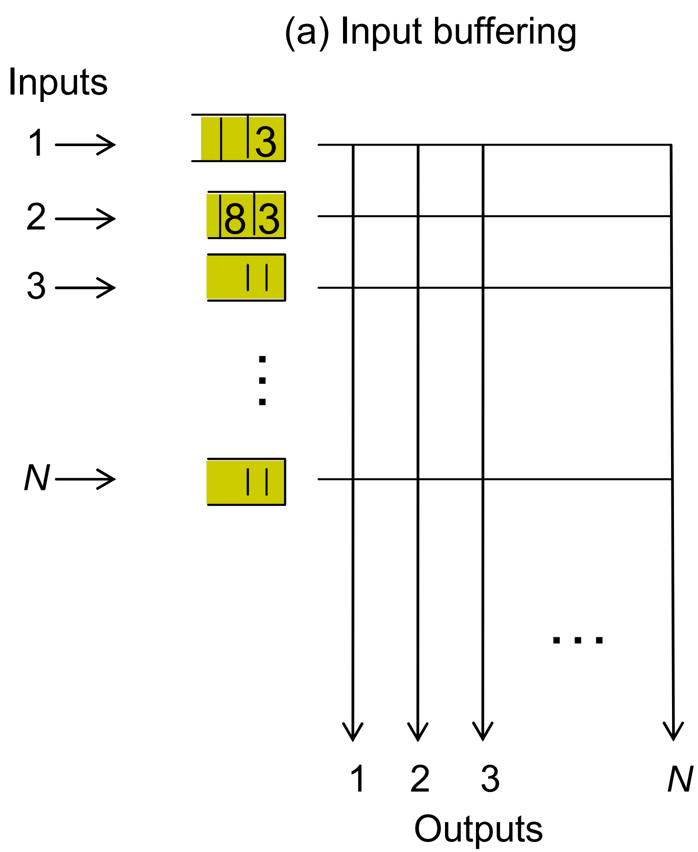
# Shared Memory Packet Switch

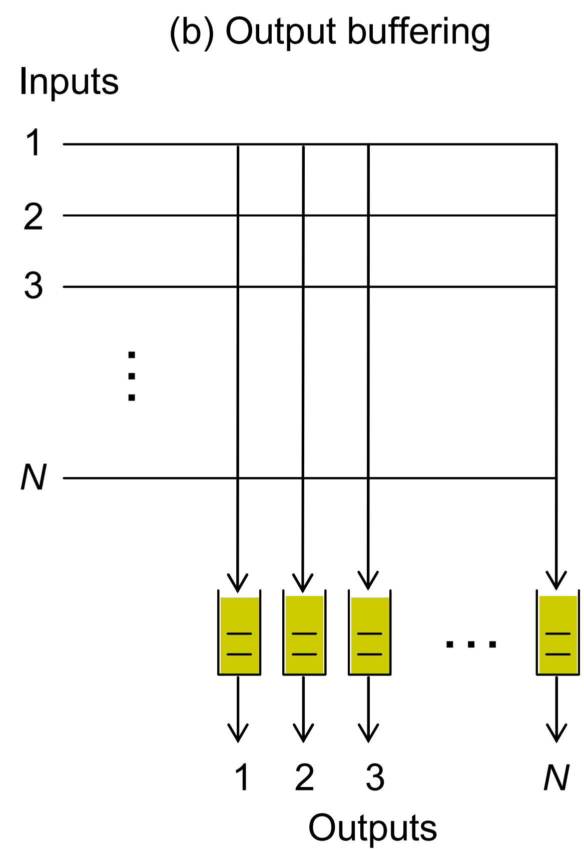



Small switches can be built by reading/writing into shared memory

# Crossbar Switches

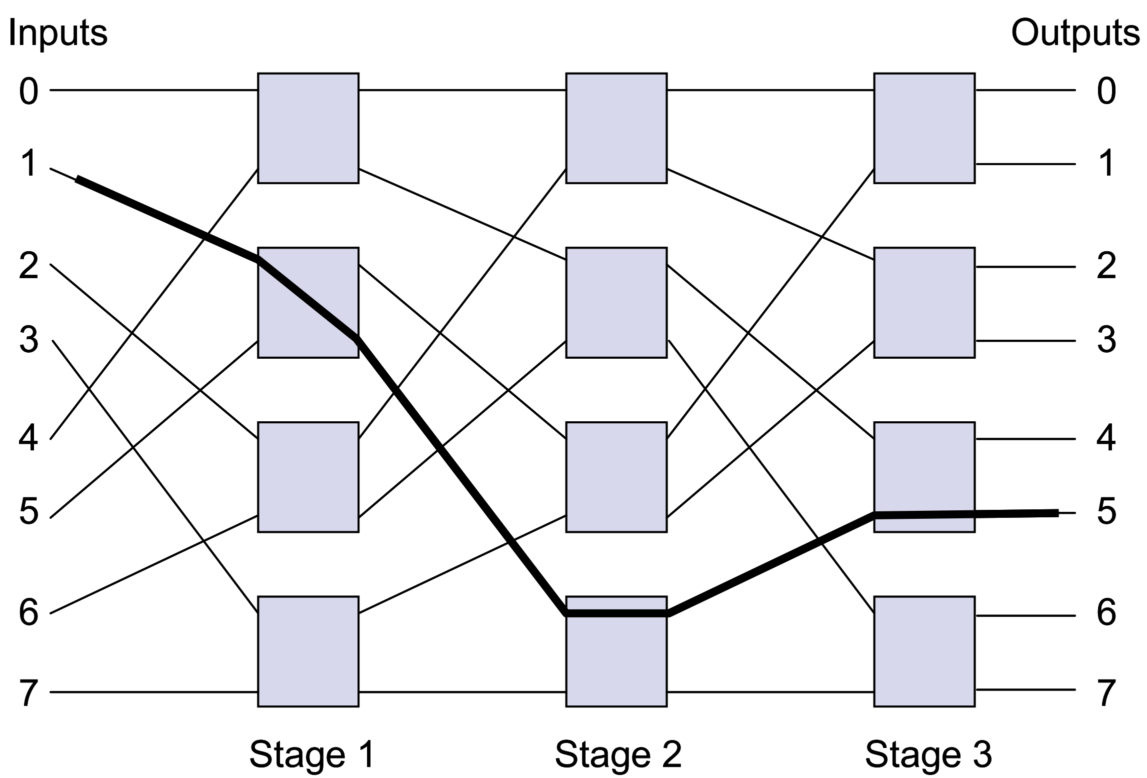
- Large switches built from crossbar & multistage space switches
- Requires centralized controller/scheduler (who sends to whom when)
- Can buffer at input, output, or both (performance vs complexity)



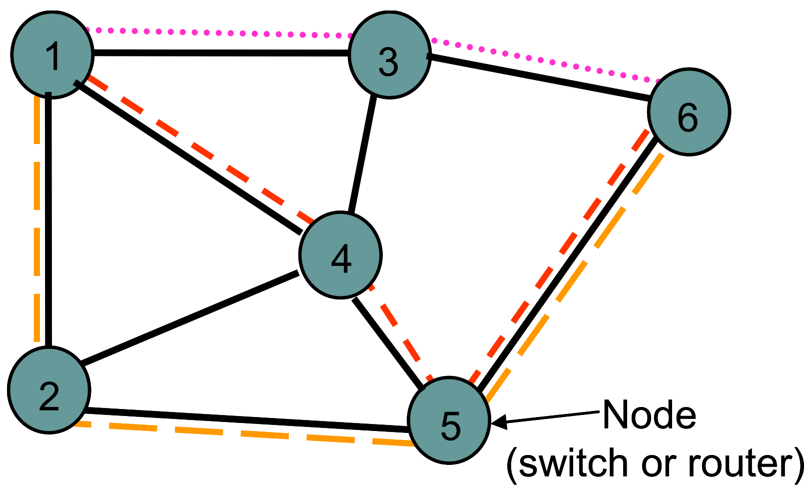
# Self-Routing Switches

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



# 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?



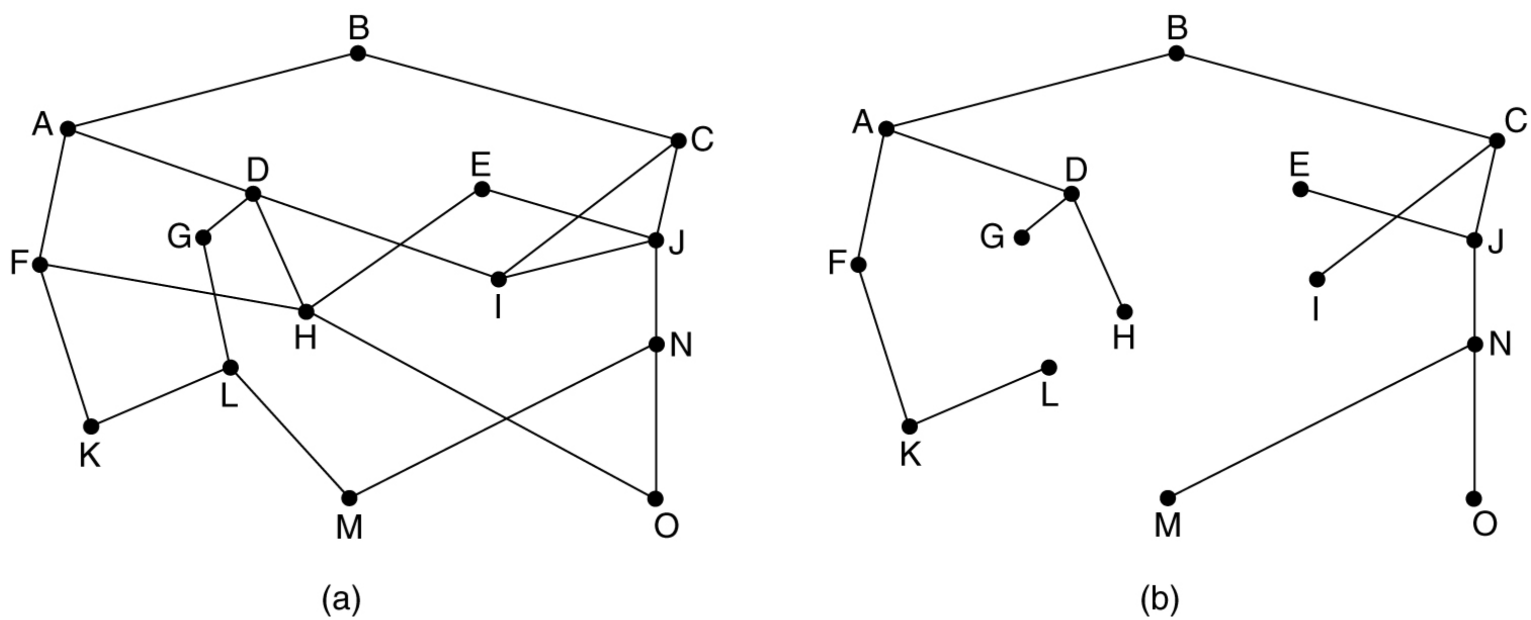
# Creating the Routing Tables

- 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

# The Optimality Principle




(a) A subnet. (b) A sink tree for router B.

***If router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route***

# Routing Algorithms

- Flooding
- Shortest Path Routing
- Distance Vector Routing
- Link State Routing
- Hierarchical Routing
- Broadcast Routing
- Multicast Routing
- Routing for Mobile Hosts
- Routing in Ad Hoc Networks

***The above is not mutually exclusive classification***

# 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 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

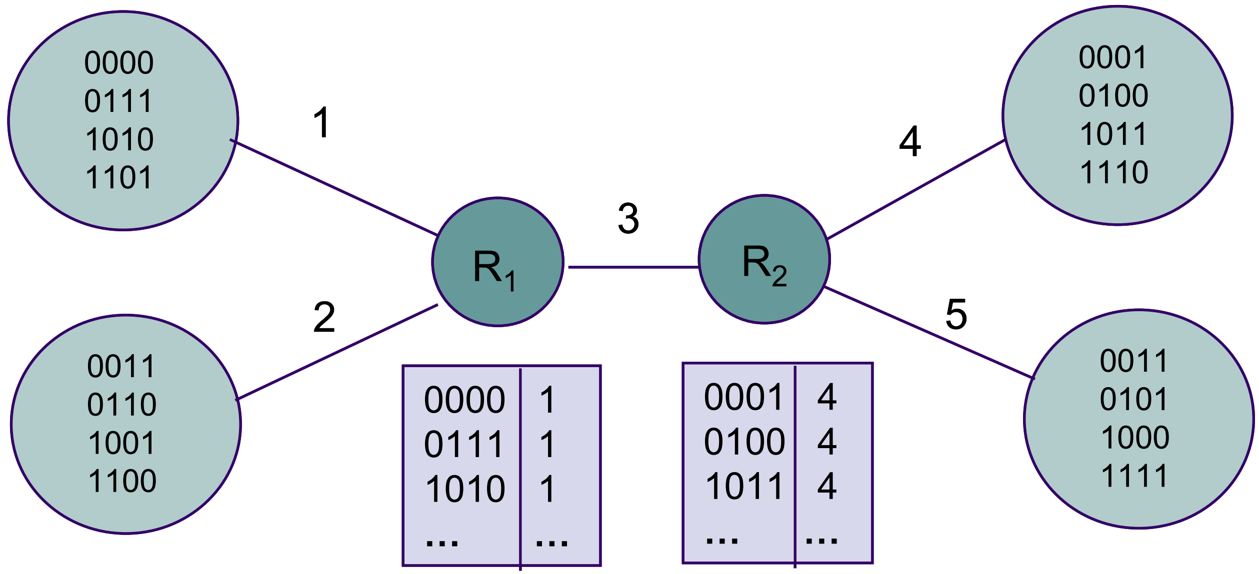
| Destination | Next node |
|---|---|
| 1 | 1 |
| 3 | 1 |
| 4 | 4 |
| 5 | 5 |
| 6 | 5 |

Node 5

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

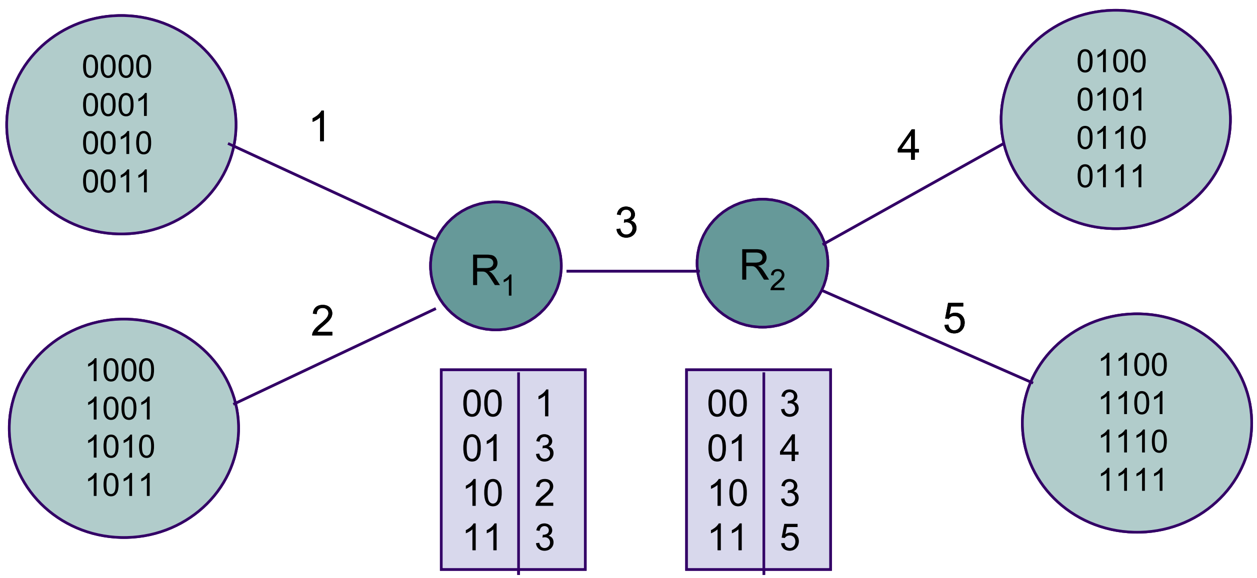
# Non-Hierarchical Addresses and Routing

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



# 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

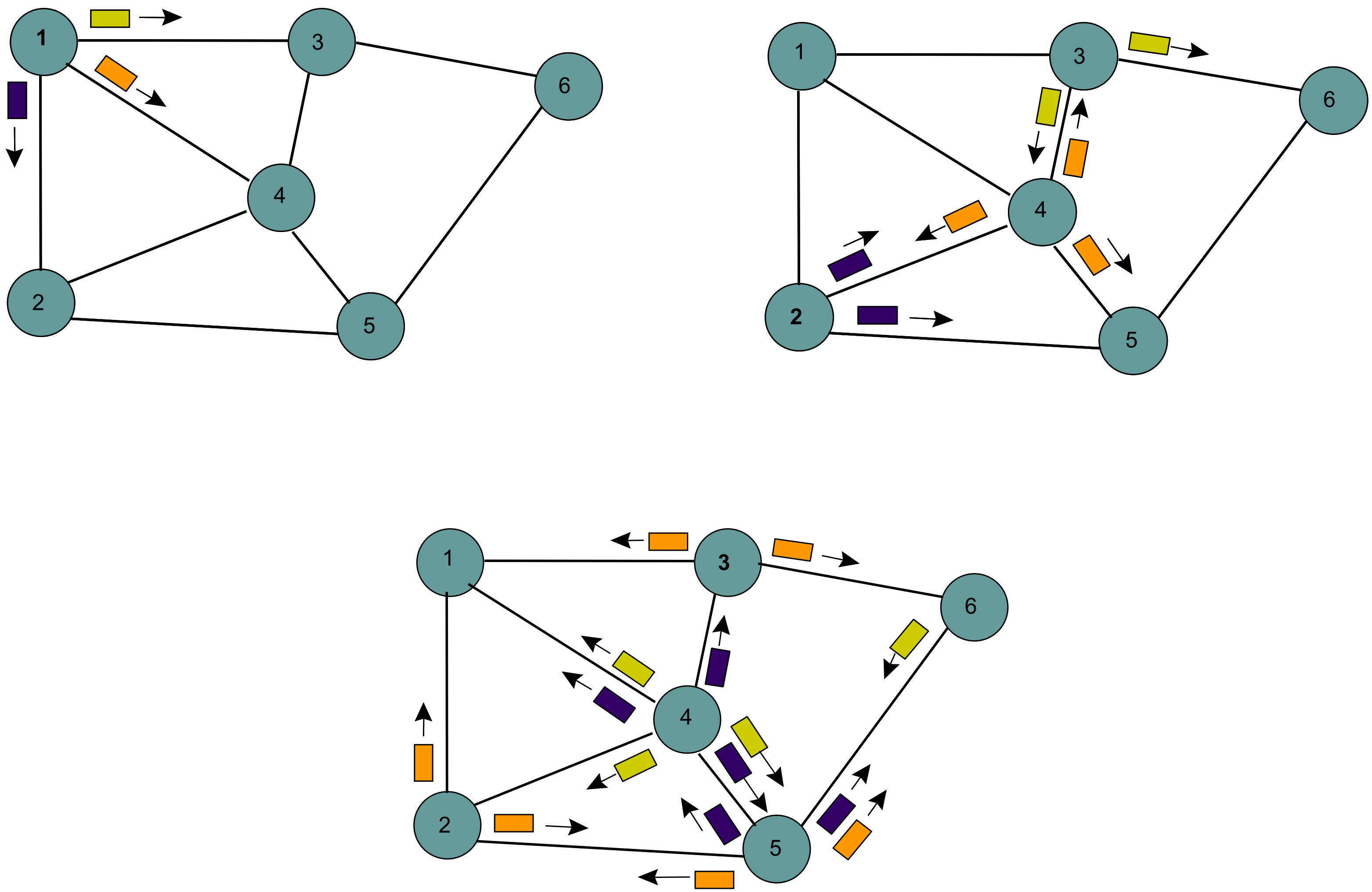
Send a packet to all nodes in a network

- 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 where it arrived
- Exponential growth in packet transmissions

# Example of Flooding



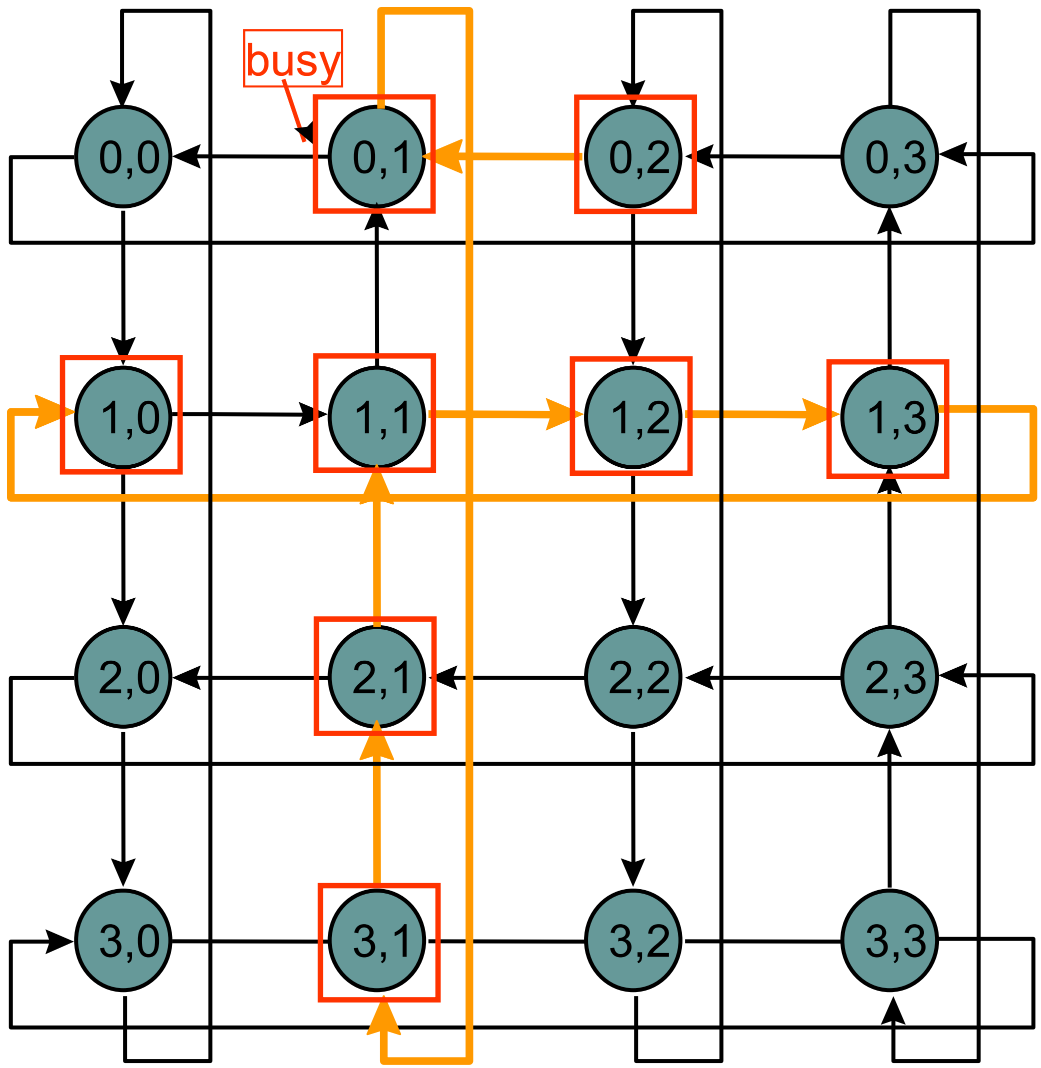
# Limited Flooding

- 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 record 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

# Example: (0,2)→(1,0)



# Paths versus 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 an optimal 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
  - Bellman-ford 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

*Local Signpost*

- Direction
- Distance

*Routing Table*

For each destination list:

- Next Node
- Distance

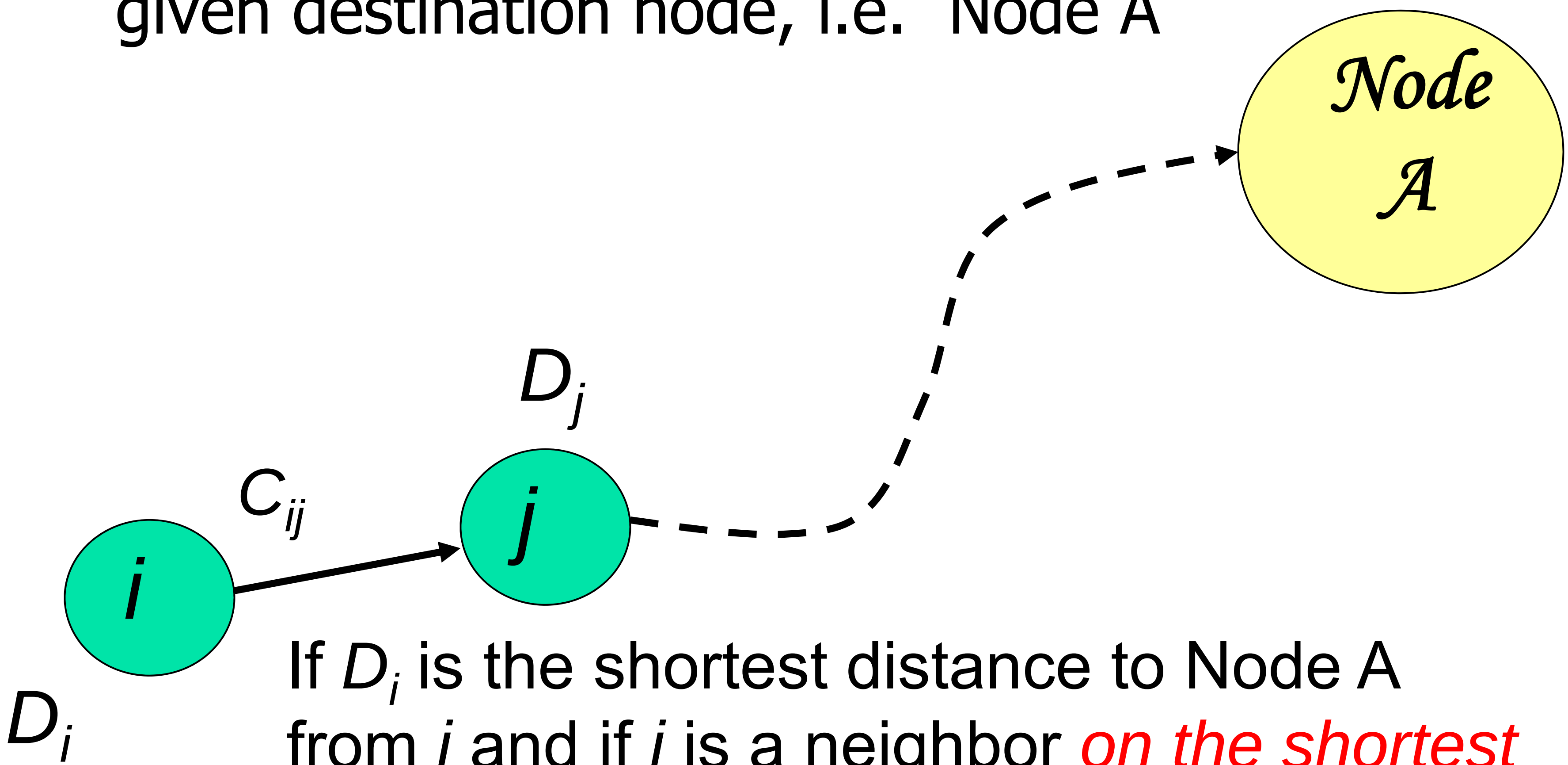
*Table Synthesis*

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

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

# Shortest Path to a Node

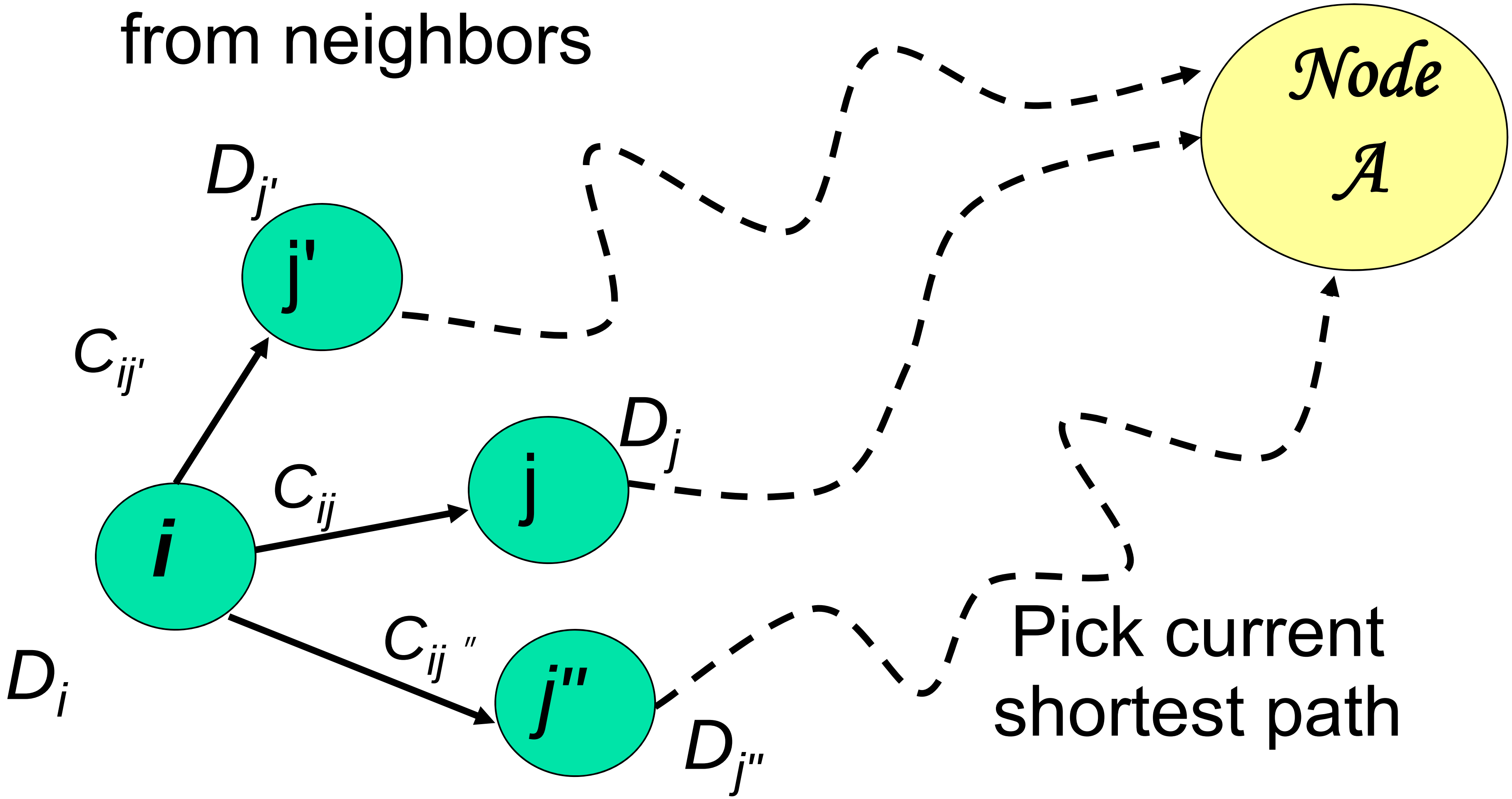
- Focus on how nodes find their shortest path to a given destination node, i.e.  Node A




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

# Why Distance Vector Work?

*Node i* only has local info from neighbors




Pick current shortest path

# 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>,
    - $Min_j \{ C_{ij} + D_j \}$
    - 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 (continued)

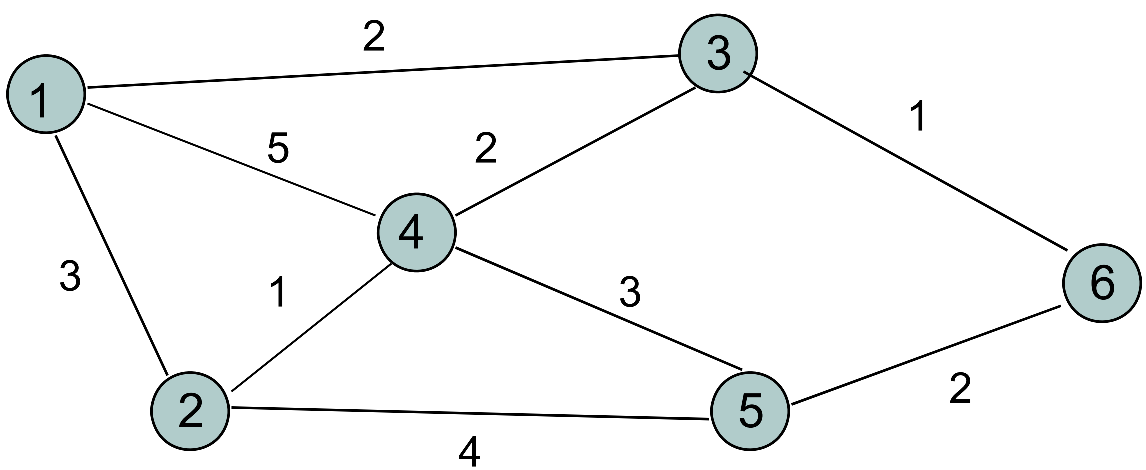
- *Now consider parallel computations for all destinations d*
- *Initialization*
  - Each node has 1 row for each destination $d$
  - Distance of node $d$ to itself is zero: $D_d(d)=0$
  - Distance of other node $j$ to $d$ is infinite: $D_j(d)= \propto$, for $j \neq d$
  - Next node $n_j(d)$= -1 since 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 $d$,
    - $Min_j \{ C_{ij}+ D_j(d) \}$
    - Replace old $(n_i(d), D_i(d))$ by new $(n_j^*, D_j^*(d))$ if new next node or distance found
  - Go to send step

# Example

| Iteration | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 |
|---|---|---|---|---|---|
| Initial | $(-1, \infty)$ | $(-1, \infty)$ | $(-1, \infty)$ | $(-1, \infty)$ | $(-1, \infty)$ |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |

Table entry @ node 1 for dest A

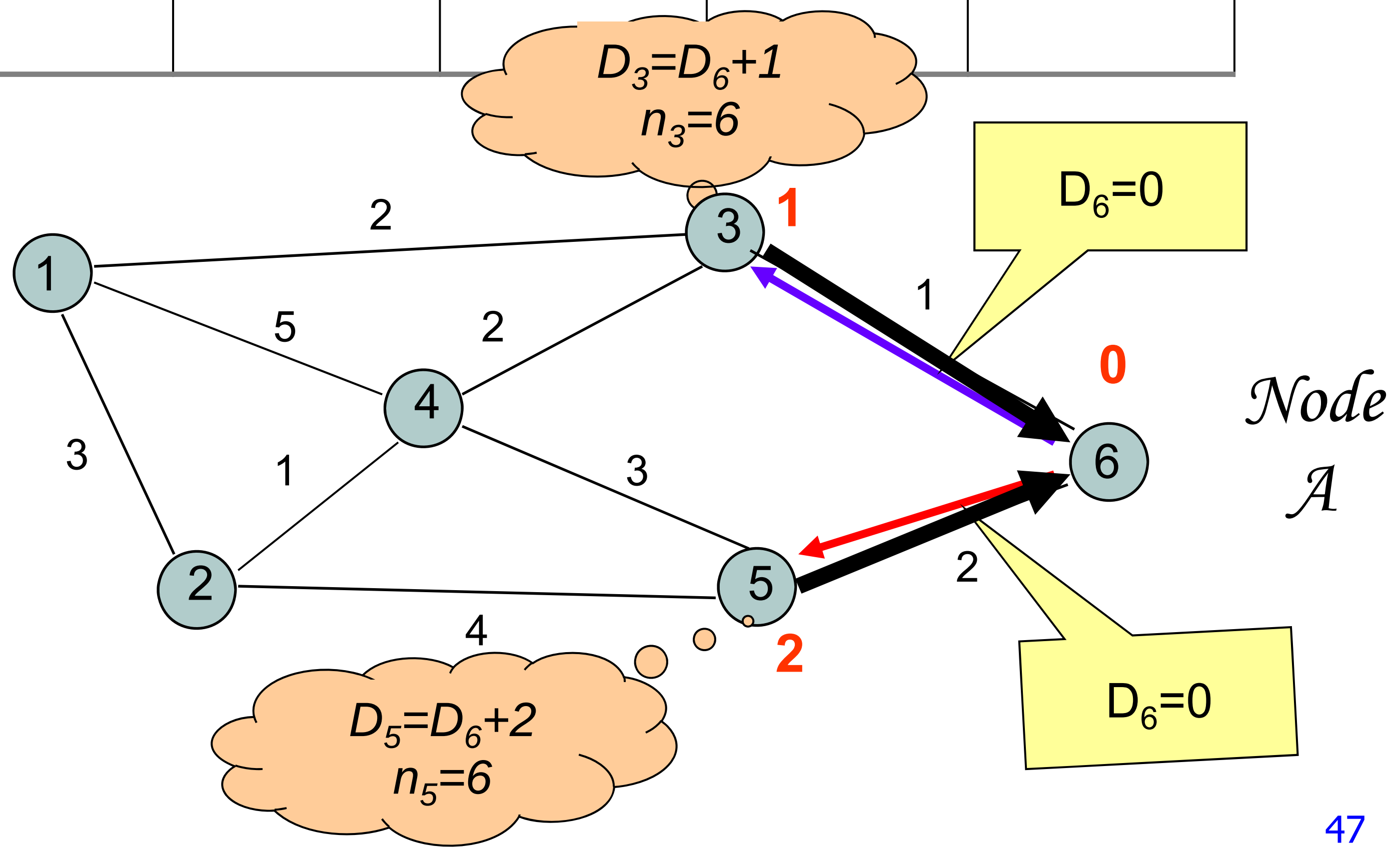
Table entry @ node 3 for dest A




*Node A*

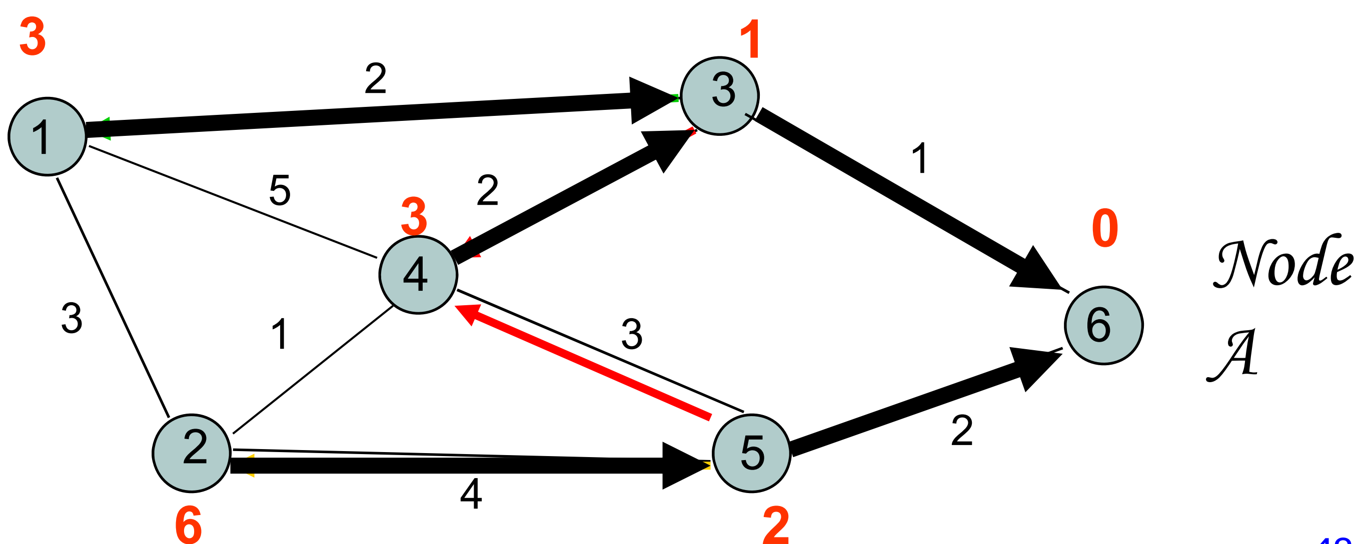
# Example (continued)

| 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 | | | | | |



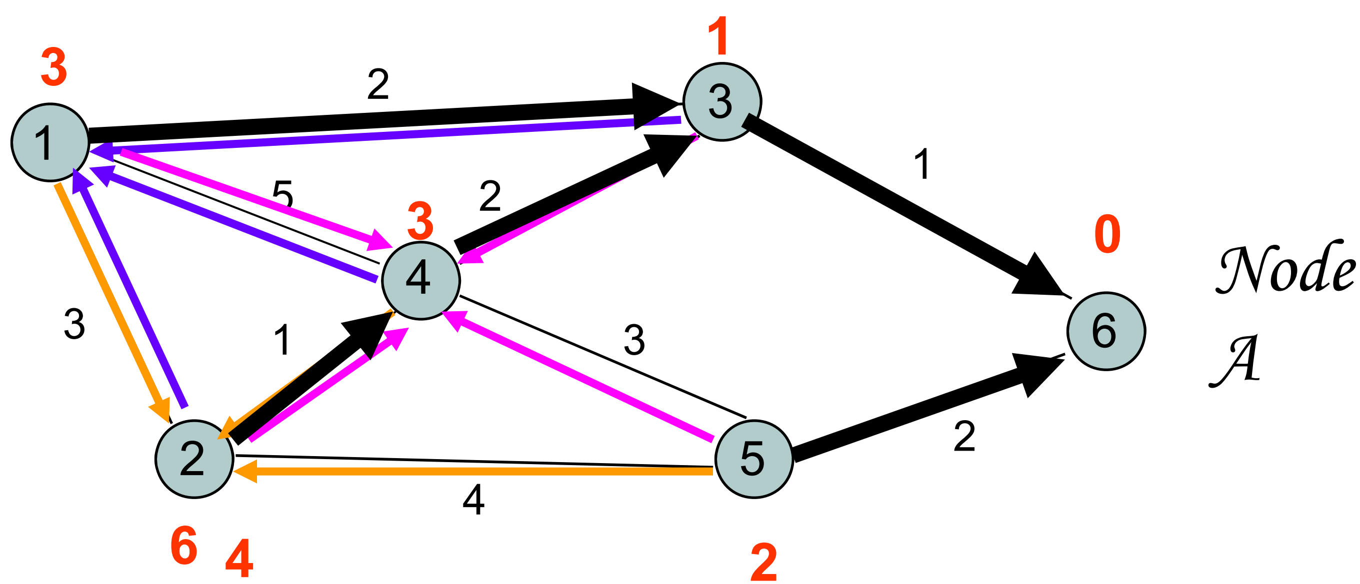
# Example (Continued)

| 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 | | | | | |



# Example (Continued)

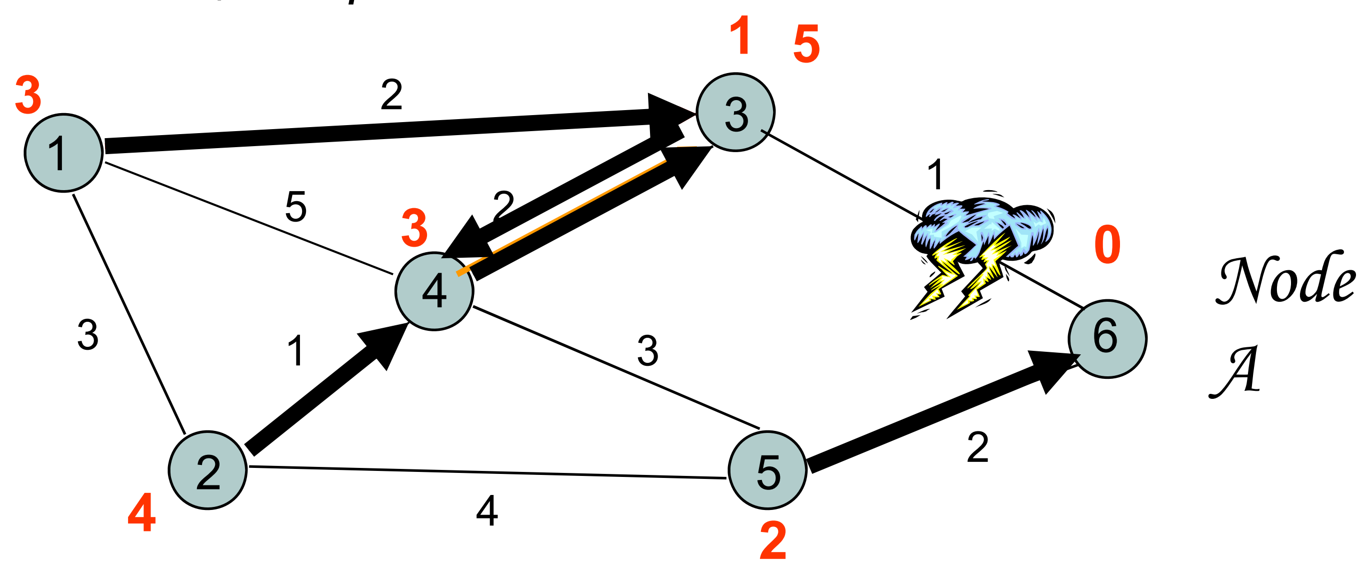
| Iteration | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 |
|---|---|---|---|---|---|
| Initial | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) | (-1, ∞) |
| 1 | (-1, ∞) | (-1, ∞) | (6, 1) | (-1, ∞) | (6,2) |
| 2 | (3,3) | (5,6) | (6, 1) | (3,3) | (6,2) |
| 3 | (3,3) | (4,4) | (6, 1) | (3,3) | (6,2) |



# Example (Continued)

| 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 | | | | | |

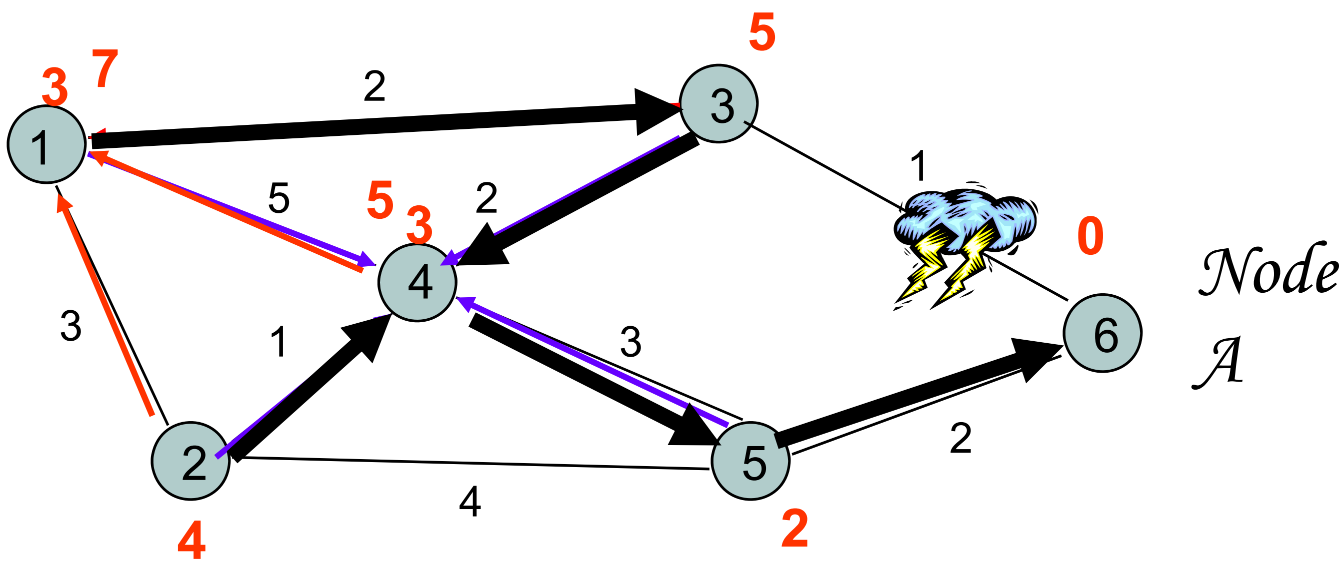
*Network disconnected; Loop created between nodes 3 and 4*



# Example (continued)

| 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 | | | | | |

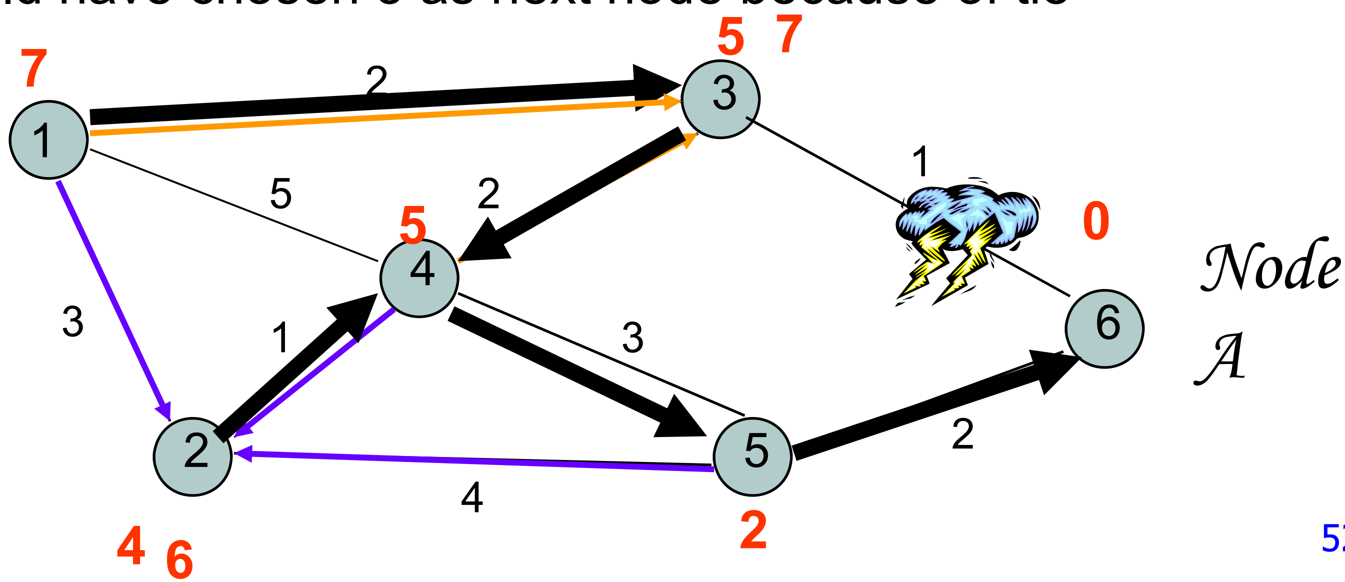
Node 4 could have chosen 2 as next node because of tie



# Example (Continued)

| 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) |

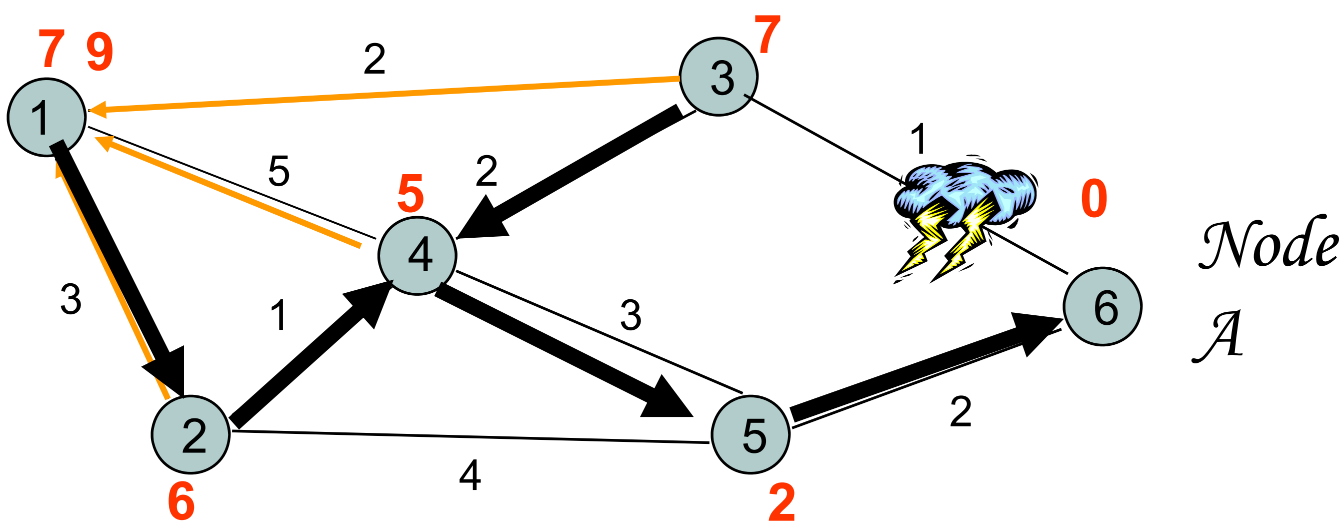
Node 2 could have chosen 5 as next node because of tie



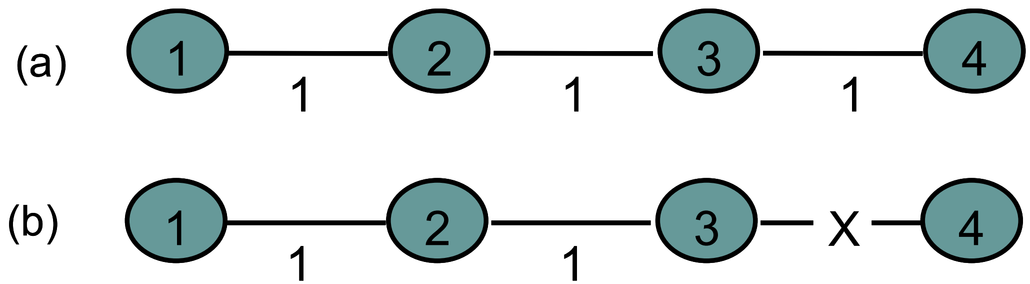
# Example (Continued)

| 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)

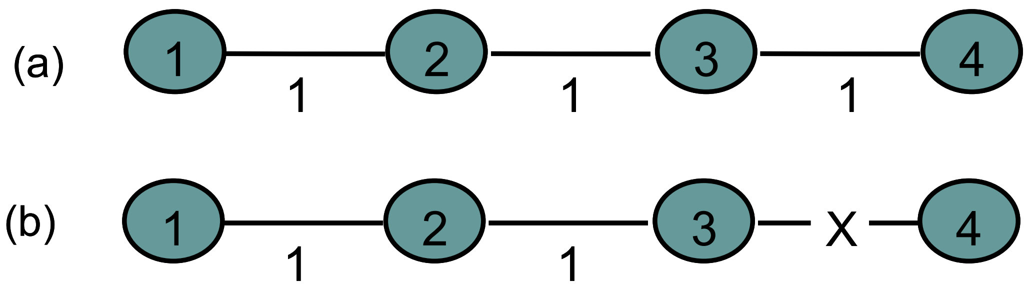
| Update | Node 1 | Node 2 | Node 3 |
|---|---|---|---|
| Before break | (2,3) | (3,2) | (4, 1) |
| After break | (2,3) | (3,2) | (2,3) |
| 1 | (2,3) | (3,4) | (2,3) |
| 2 | (2,5) | (3,4) | (2,5) |
| 3 | (2,5) | (3,6) | (2,5) |
| 4 | (2,7) | (3,6) | (2,7) |
| 5 | (2,7) | (3,8) | (2,7) |
| … | … | … |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

# Problem: Bad News Travels Slowly

Remedies

- Split Horizon
  - Do not report *route to a destination* to the neighbor from which route was learned
- Split Horizon with 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
- Another solution?
  - Report (distance, via node)?

# Split Horizon with Poisoned 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

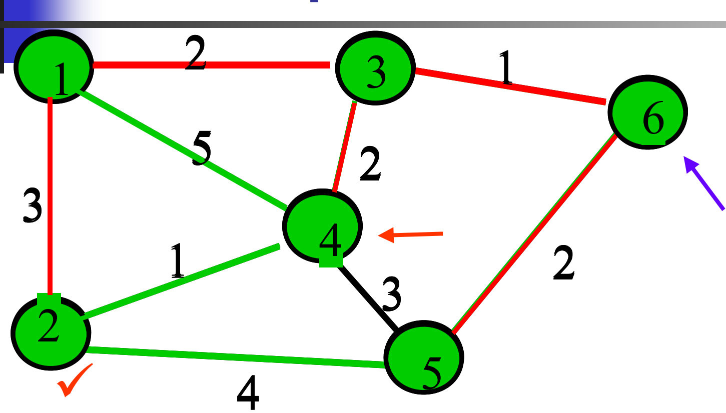
# Dijkstra's algorithm

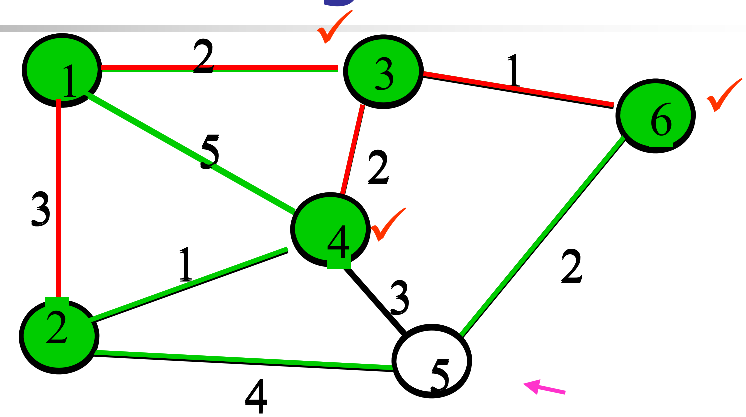
- $N$: set of nodes for which shortest path already found
- Initialization: (*Start with source node s)*
  - $N = \{s\}$, $D_s = 0$, "*s* is distance zero from itself"
  - $D_j = C_{sj}$ for all $j \neq s$, distances of directly-connected neighbors
- Step A: (*Find next closest node i*)
  - Find $i \notin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

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

  - If $N$ contains all the nodes, stop
  - Else go to Step A

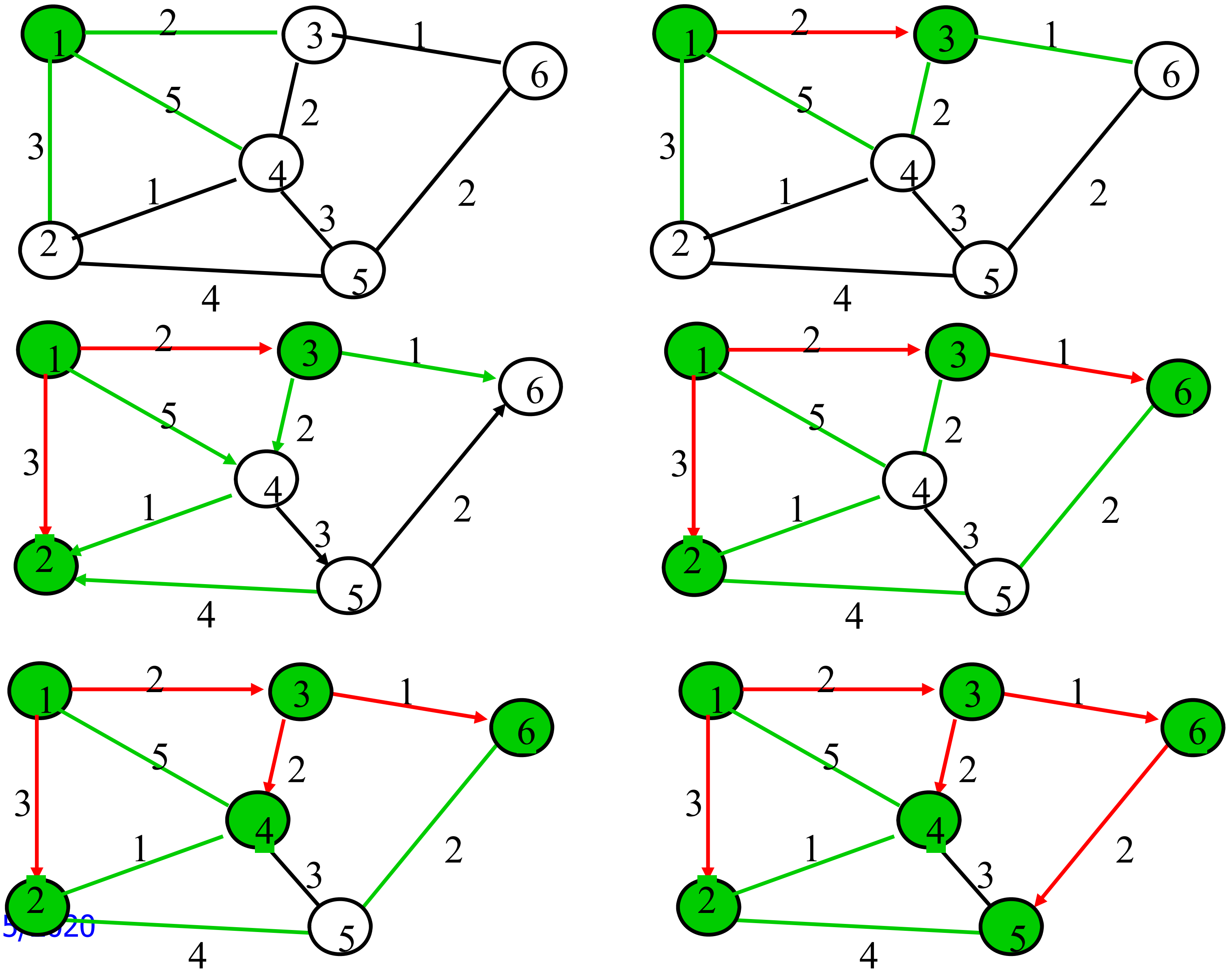
# Example of the Dijkstra's algorithm





| 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 could be very quick (but depending on network size)
- 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

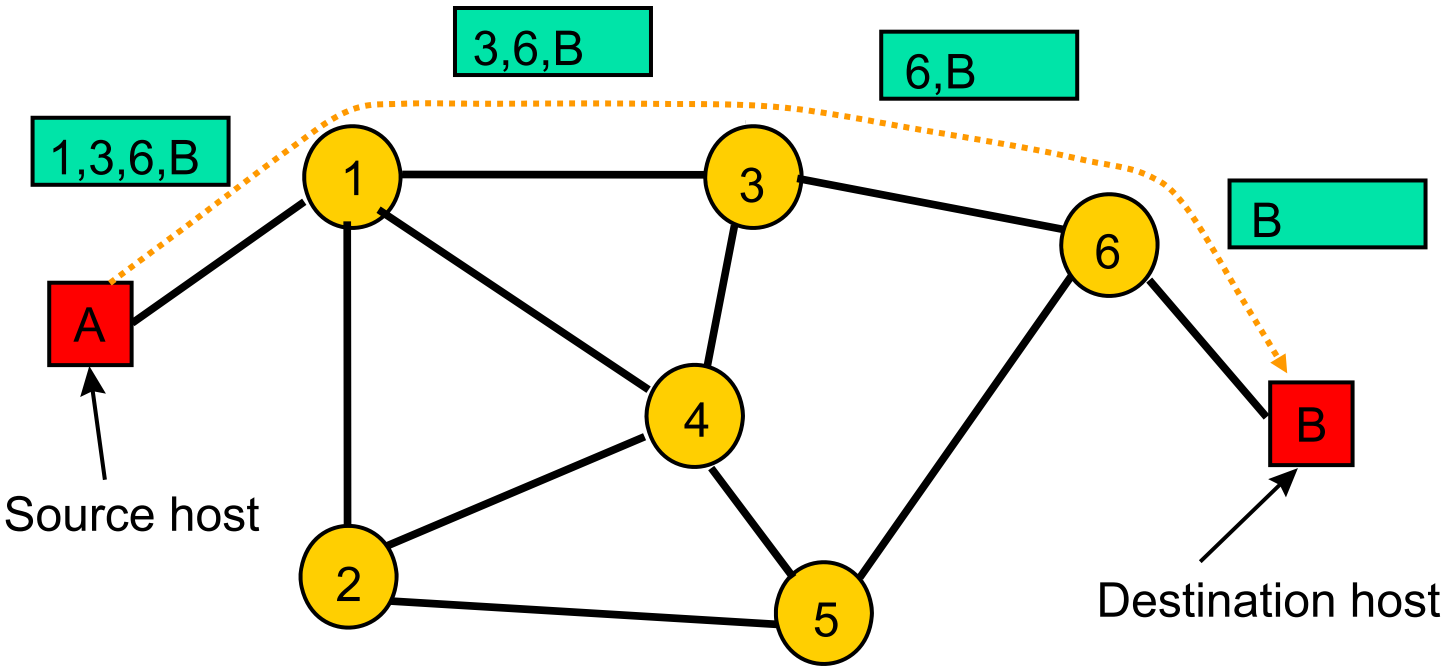
# Why is Link State Better?

- 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 that means for customers to select what service providers they use

# Example



# Prioritization and Scheduling – Packet-Level Traffic Management

# Traffic Management

Vehicular traffic management

- Traffic lights & signals control flow of traffic in city street system
- Objective is to maximize flow with tolerable delays
- Priority Services
  - Police sirens
  - Cavalcade for dignitaries
  - Bus & High-usage lanes
  - Trucks allowed only at night

Packet traffic management

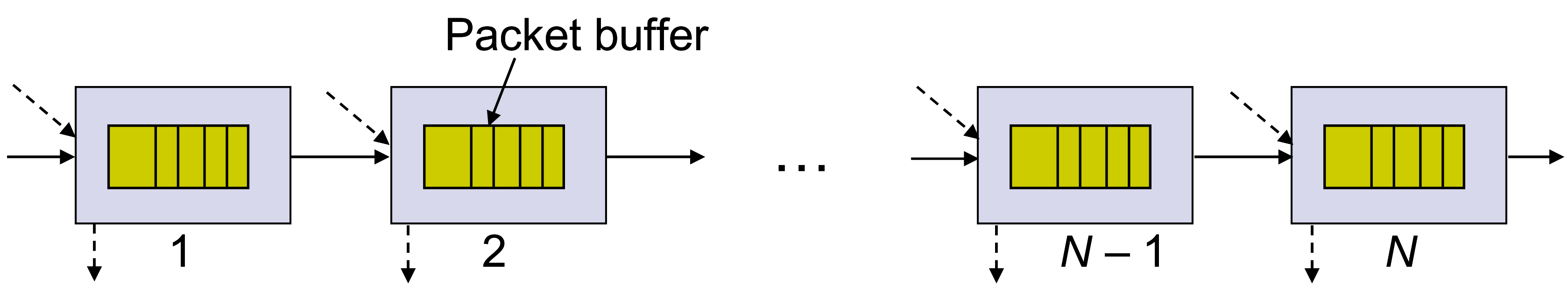
- Multiplexing & access mechanisms to control flow of packet traffic
- Objective is make efficient use of network resources & deliver QoS
- Priority
  - Fault-recovery packets
  - Real-time traffic
  - Enterprise (high-revenue) traffic
  - High bandwidth traffic

# Time Scales & Granularities

- Packet Level
  - Queueing & scheduling at multiplexing points
  - Determines relative performance offered to packets over a short time scale (microseconds)
- Flow Level
  - Management of traffic flows & resource allocation to ensure delivery of QoS (milliseconds to seconds)
  - Matching traffic flows to resources available; congestion control
- Flow-Aggregate Level
  - Routing of aggregate traffic flows across the network for efficient utilization of resources and meeting of service levels
  - "Traffic Engineering", at scale of minutes to days

# End-to-End QoS

- A packet traversing network encounters delay and possible loss at various multiplexing points
- End-to-end performance is accumulation of per-hop performance

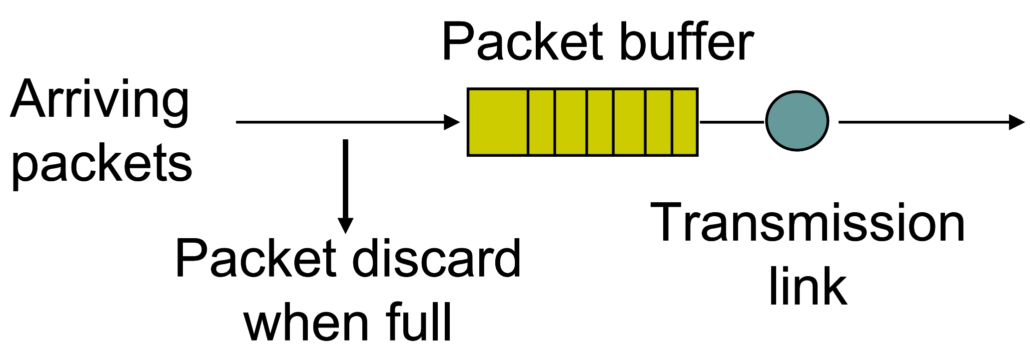


# Scheduling & QoS

- End-to-End QoS & Resource Control
  - Buffer & bandwidth control → Performance
  - Admission control to regulate traffic level
- Scheduling Concepts
  - fairness/isolation
  - priority, aggregation,
- Fair Queueing & Variations
  - WFQ, PGPS
- Guaranteed Service
  - WFQ, Rate-control
- Packet Dropping
  - aggregation, drop priorities

# FIFO Queueing

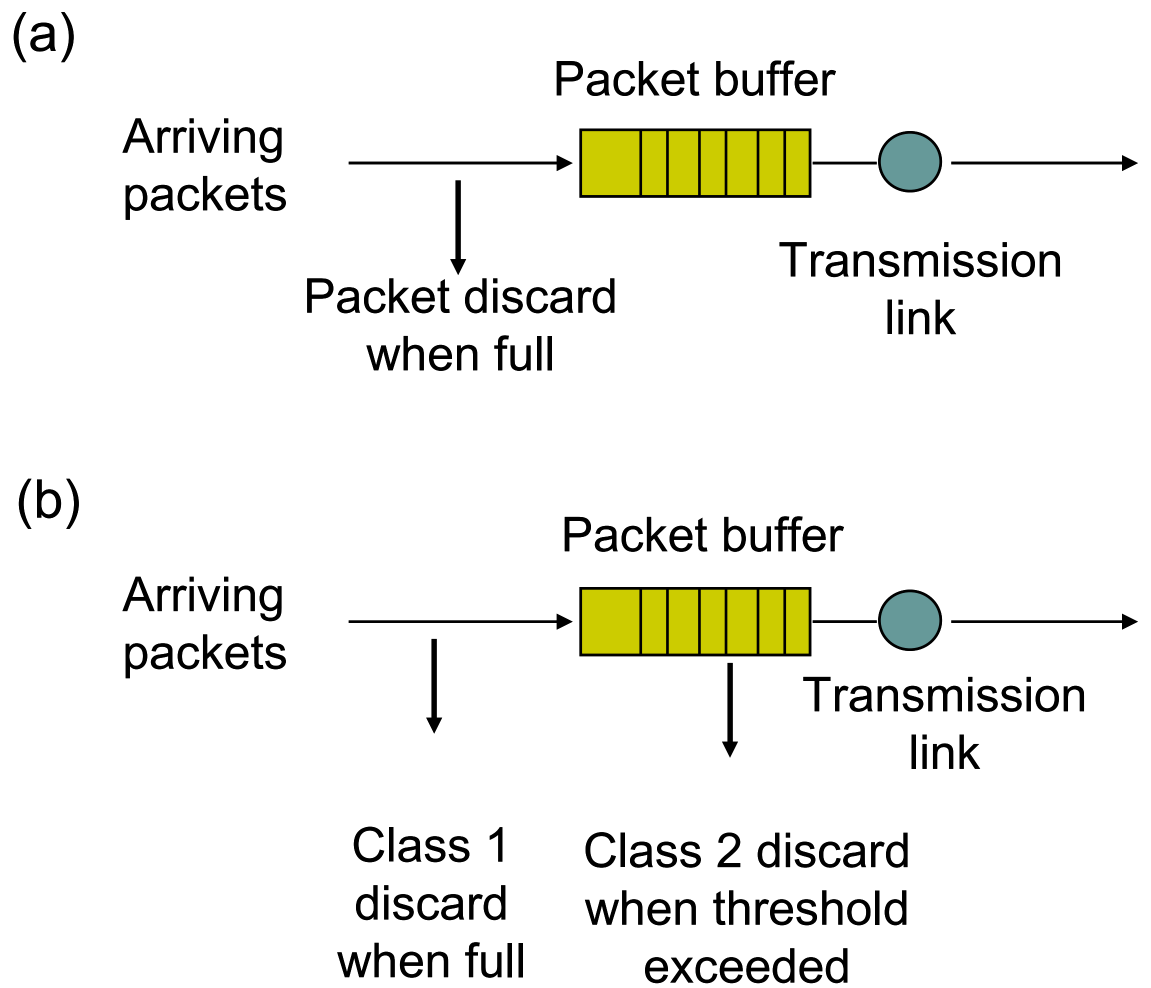
- All packet flows share the same buffer
- Transmission Discipline: First-In, First-Out
- Buffering Discipline: Discard arriving packets if buffer is full (Alternative: random discard; pushout head-of-line, i.e. oldest, packet)



# FIFO Queueing

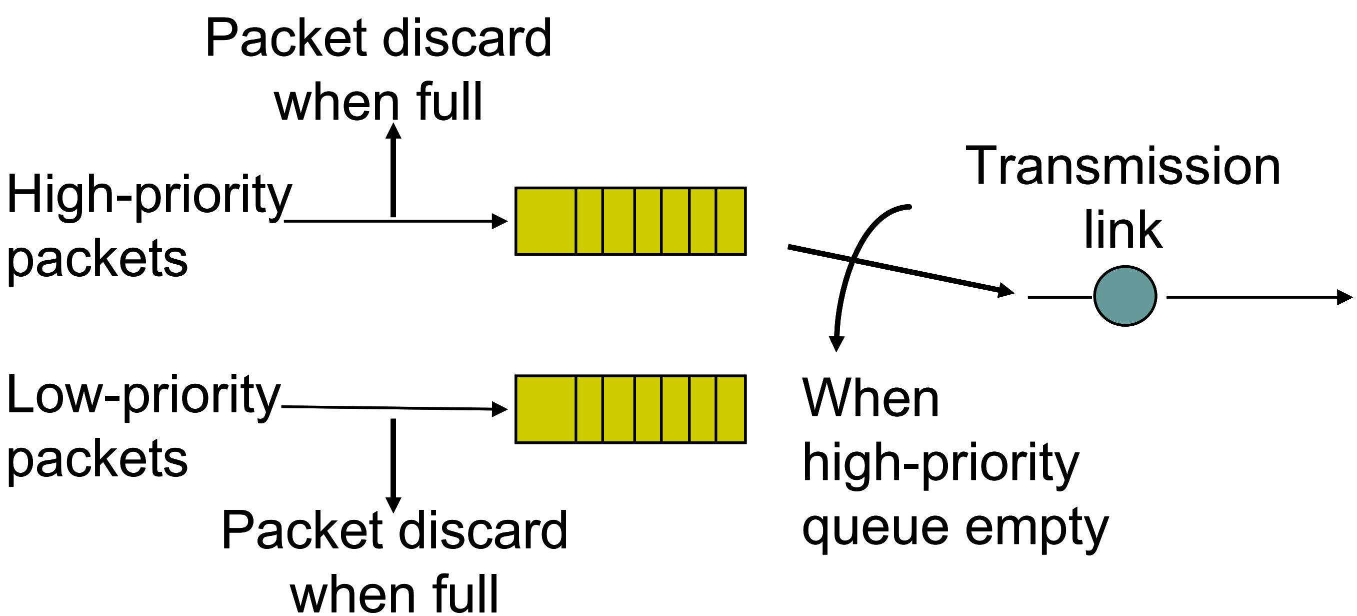
- Cannot provide differential QoS to different packet flows
  - Different packet flows interact strongly
- Statistical delay guarantees via load control
  - Restrict number of flows allowed (connection admission control)
  - Difficult to determine performance delivered
- Finite buffer determines a maximum possible delay
- Buffer size determines loss probability
  - But depends on arrival & packet length statistics
- Variation: packet enqueueing based on queue thresholds
  - some packet flows encounter blocking before others
  - higher loss, lower delay

# FIFO Queueing with Discard Priority



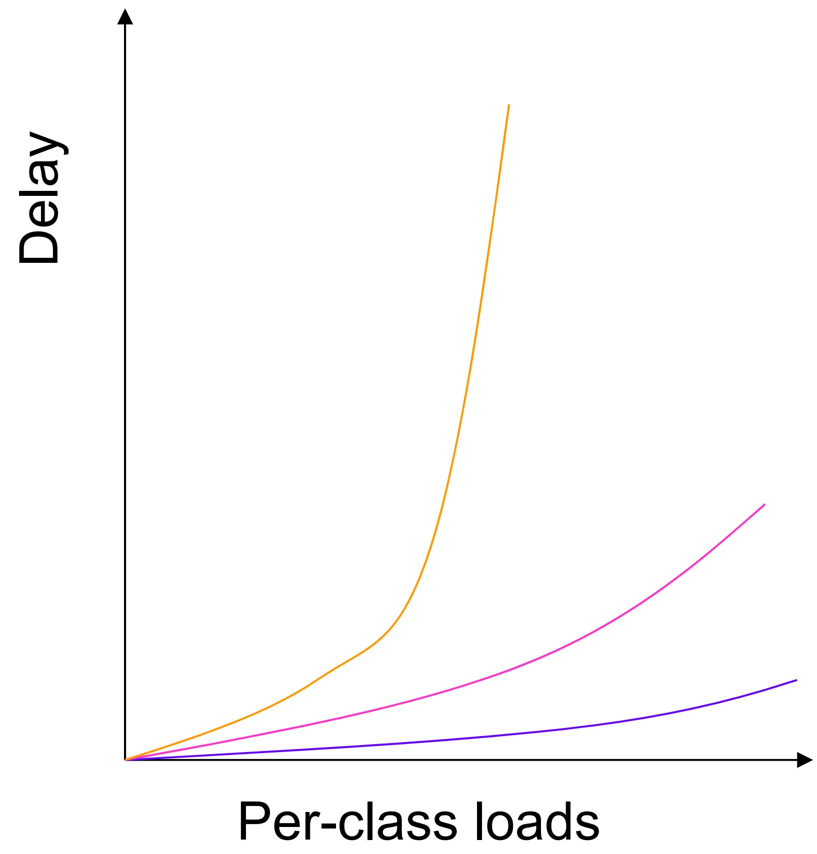
# HOL Priority Queueing

- High priority queue serviced until empty
- High priority queue has lower waiting time
- Buffers can be dimensioned for different loss probabilities
- Surge in high priority queue can cause low priority queue to saturate



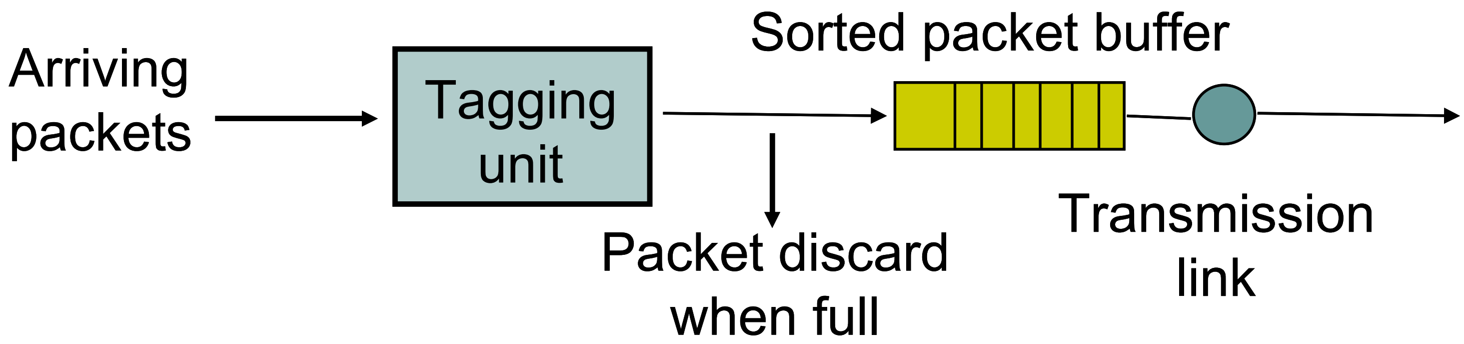
# HOL Priority Features

- Provides differential QoS
- Pre-emptive priority: lower classes invisible
- Non-preemptive priority: lower classes impact higher classes through residual service times
- High-priority classes can hog all of the bandwidth & starve lower priority classes
- Need to provide some isolation between classes



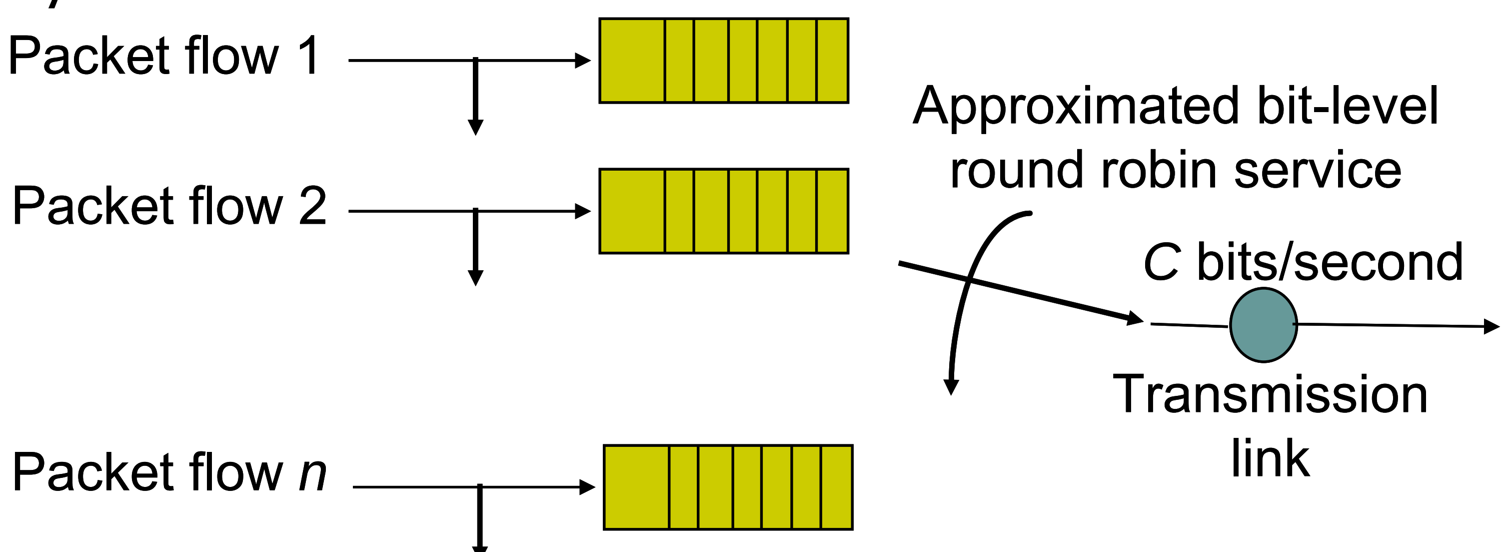
# Earliest Due Date Scheduling

- Queue in order of "due date"
  - packets requiring low delay get earlier due date
  - packets without delay get indefinite or very long due dates

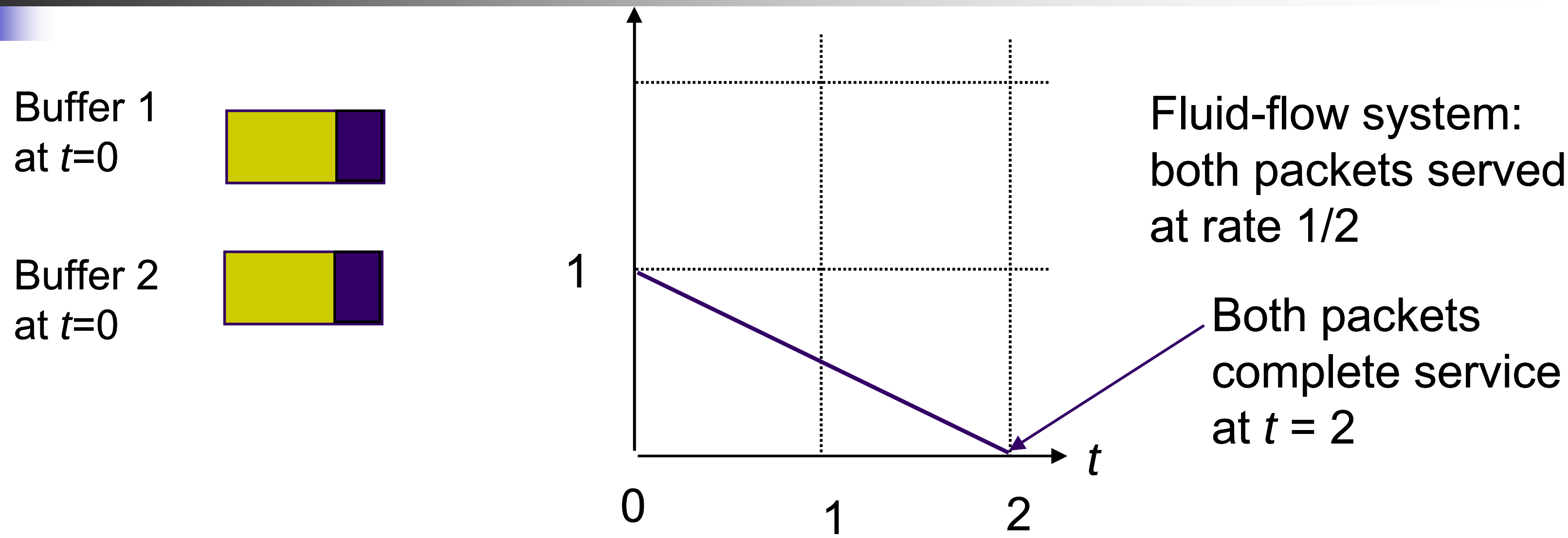


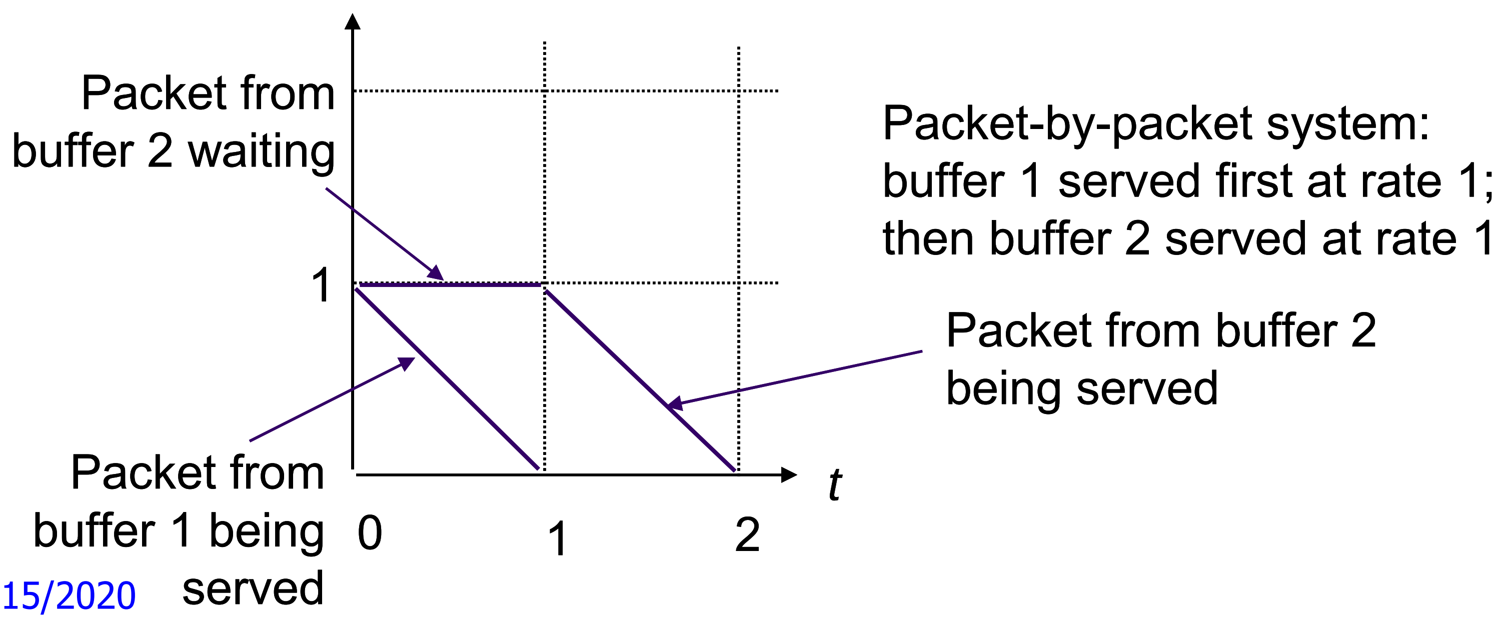
# Fair Queueing / Generalized Processor Sharing

- Each flow has its own logical queue: prevents hogging; allows differential loss probabilities
- C bits/sec allocated equally among non-empty queues
  - transmission rate = C / n(t), where n(t)=# non-empty queues
- Idealized system assumes fluid flow from queues
- Implementation requires approximation: simulate fluid system; sort packets according to completion time in ideal system

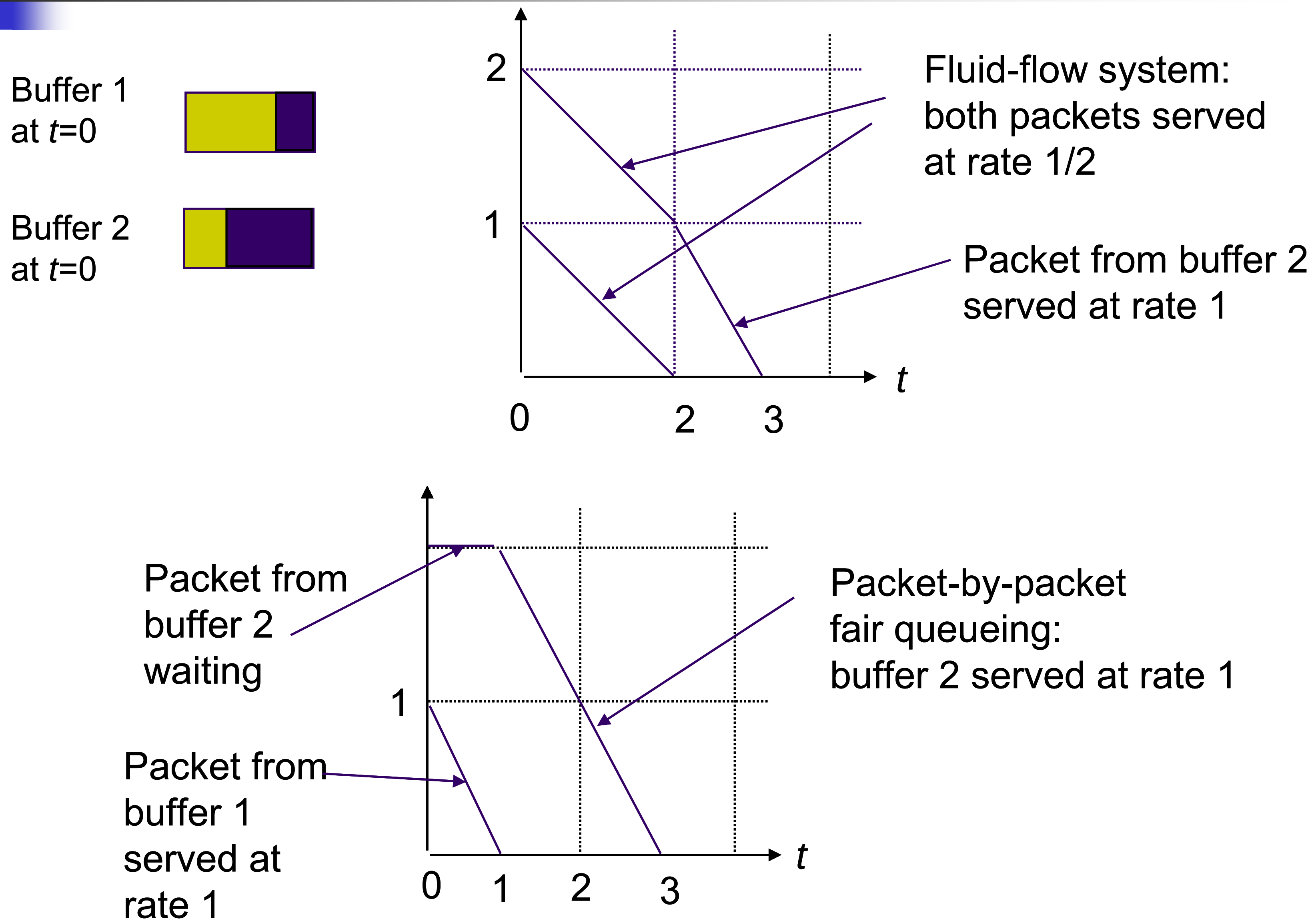


# Fluid-Flow versus Packet-by-Packet Service



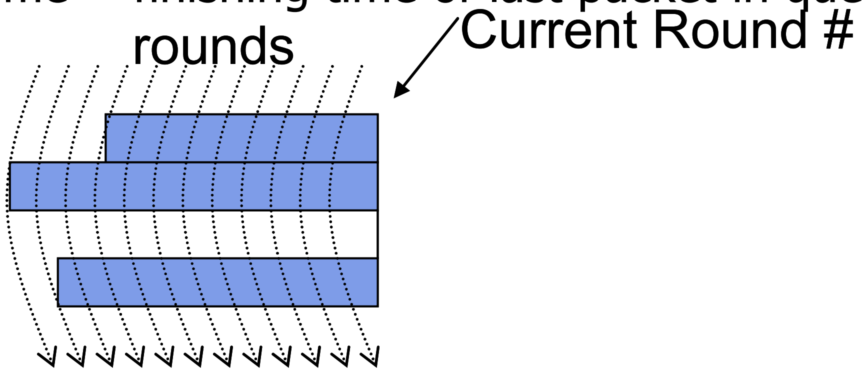



# Fluid-Flow versus Packet-by-Packet Service (continued)



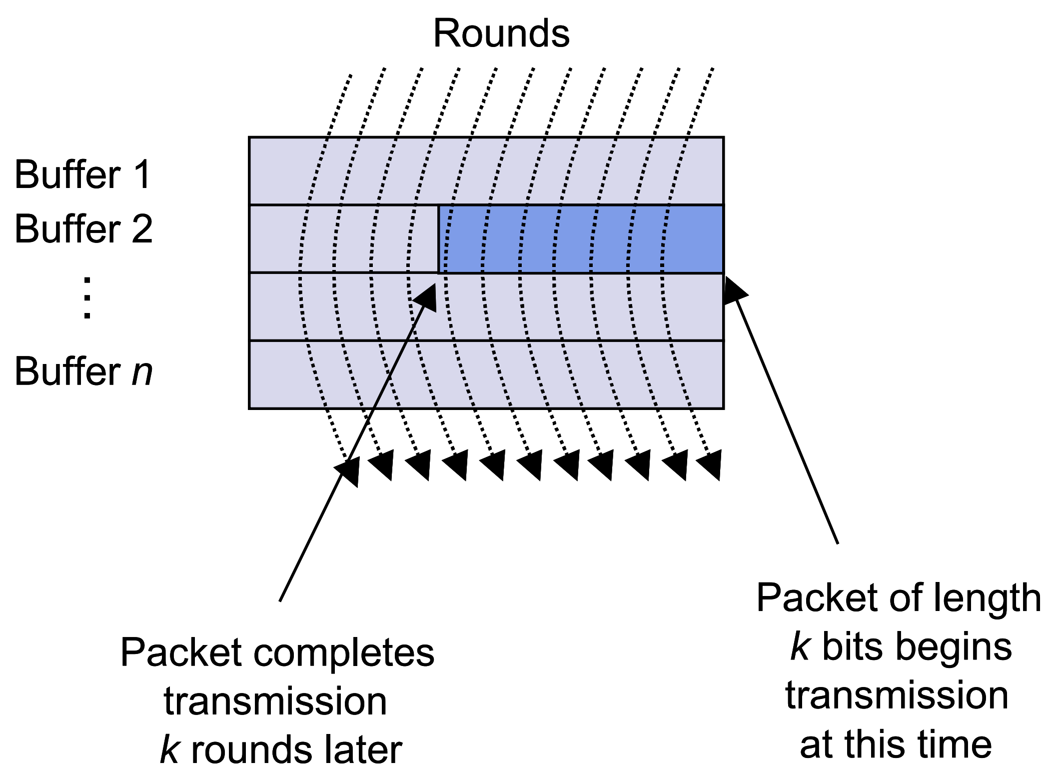
# Bit-by-Bit Fair Queueing

- Assume n flows, n queues
- 1 round = 1 cycle serving all n queues
- If each queue gets 1 bit per cycle, then 1 round = # active queues
  - Actual duration of a round is determined by active queues.
- Round number = number of cycles of service that have been completed
- If packet arrives to idle queue:
  Finishing time = round number + packet size in bits
- If packet arrives to active queue:
  Finishing time = finishing time of last packet in queue + packet size



# More Details

Number of rounds = Number of bit transmission opportunities



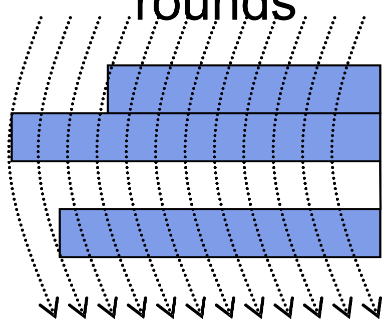

Differential Service:

If a traffic flow is to receive twice as much bandwidth as a regular flow, then its packet completion time would be half

# Computing the Finishing Time

- *F(i,k,t)* = finish time of *k*th packet that arrives at time *t* to flow *i*
- *P(i,k,t)* = size of *k*th packet that arrives at time t to flow *i*
- *R(t)* = round number at time *t*

Generalize so *R(t)* continuous, not discrete




*R(t)* grows at rate inversely proportional to *n(t)*

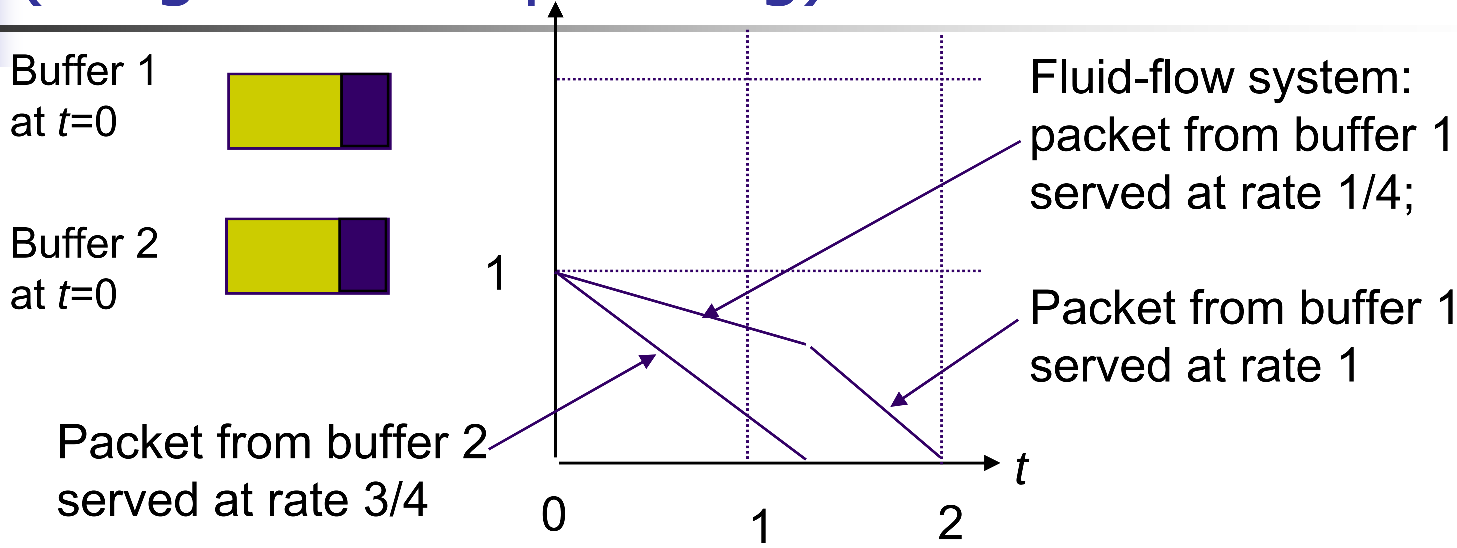
- Packet-by-Packet Fair Queueing:

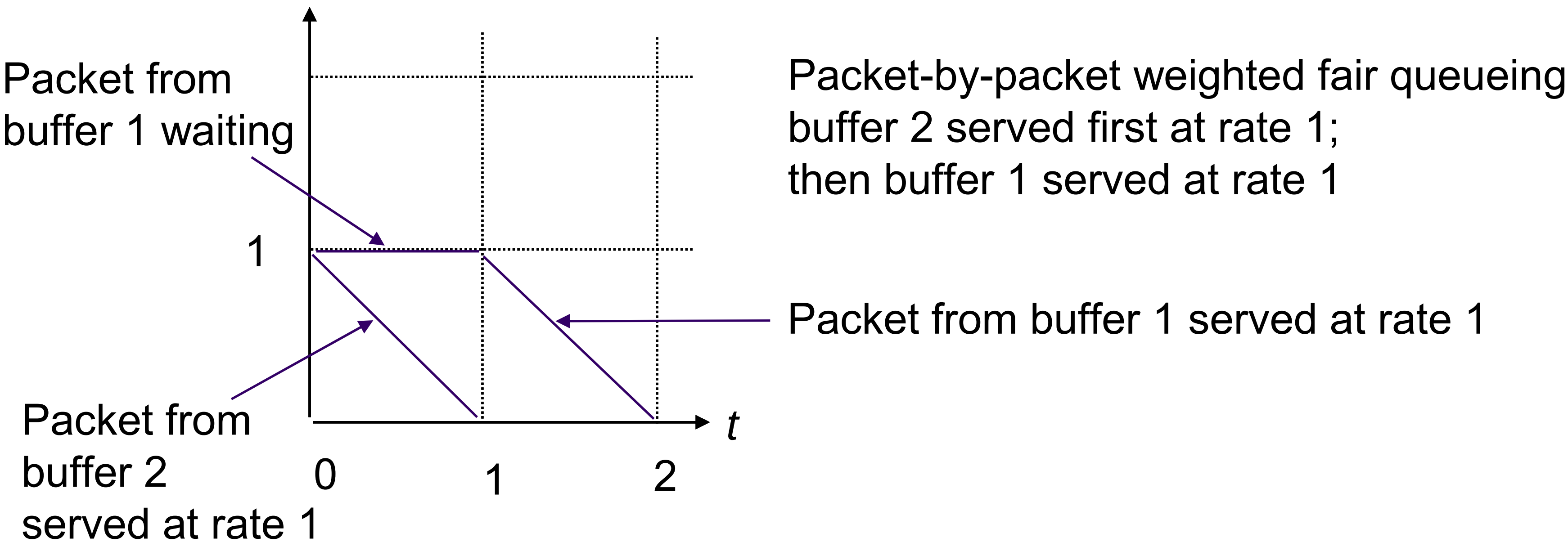
$$F(i,k,t) = \max\{F(i,k-1,t^{k-1}),\ R(t)\} + P(i,k,t)$$

- Weighted Fair Queueing:

$$F(i,k,t) = \max\{F(i,k-1,t^{k-1}), R(t)\} + P(i,k,t)/w_i$$

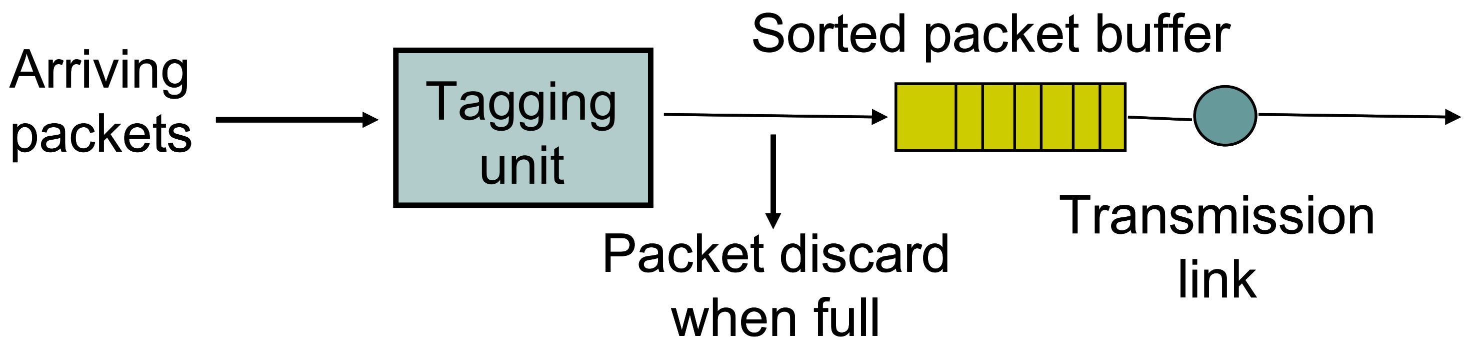
# Fluid-Flow versus Packet-by-Packet Service (weighted fair queueing)



# Packetized GPS/WFQ

- Compute packet completion time in ideal system
  - add tag to packet
  - sort packet in queue according to tag
  - serve according to HOL



# WFQ and Packet QoS

- WFQ and its many variations form the basis for providing QoS in packet networks
- Very high-speed implementations available, up to 10 Gbps and possibly higher
- WFQ must be combined with other mechanisms to provide end-to-end QoS

# Buffer Management

- Packet drop strategy: Which packet to drop when buffers full
- Fairness: protect well-behaving sources from misbehaving sources
- Aggregation:
  - Per-flow buffers protect flows from misbehaving flows
  - Full aggregation provides no protection
  - Aggregation into classes provided intermediate protection
- Drop priorities:
  - Drop packets from buffer according to priorities
  - Maximizes network utilization & application QoS
  - Examples: layered video, policing at network edge
- Controlling sources at the edge

# Early or Overloaded Drop

*Random early detection:*

- Drop packets if short-term avg of queue exceeds threshold
- Packet drop probability increases linearly with queue length
- Dropped packets provide feedback to sources to reduce rate
- Why random?
  - Higher rate gets a higher chance of packet drop
- Improves performance of cooperating sources
- Increases loss probability of misbehaving sources

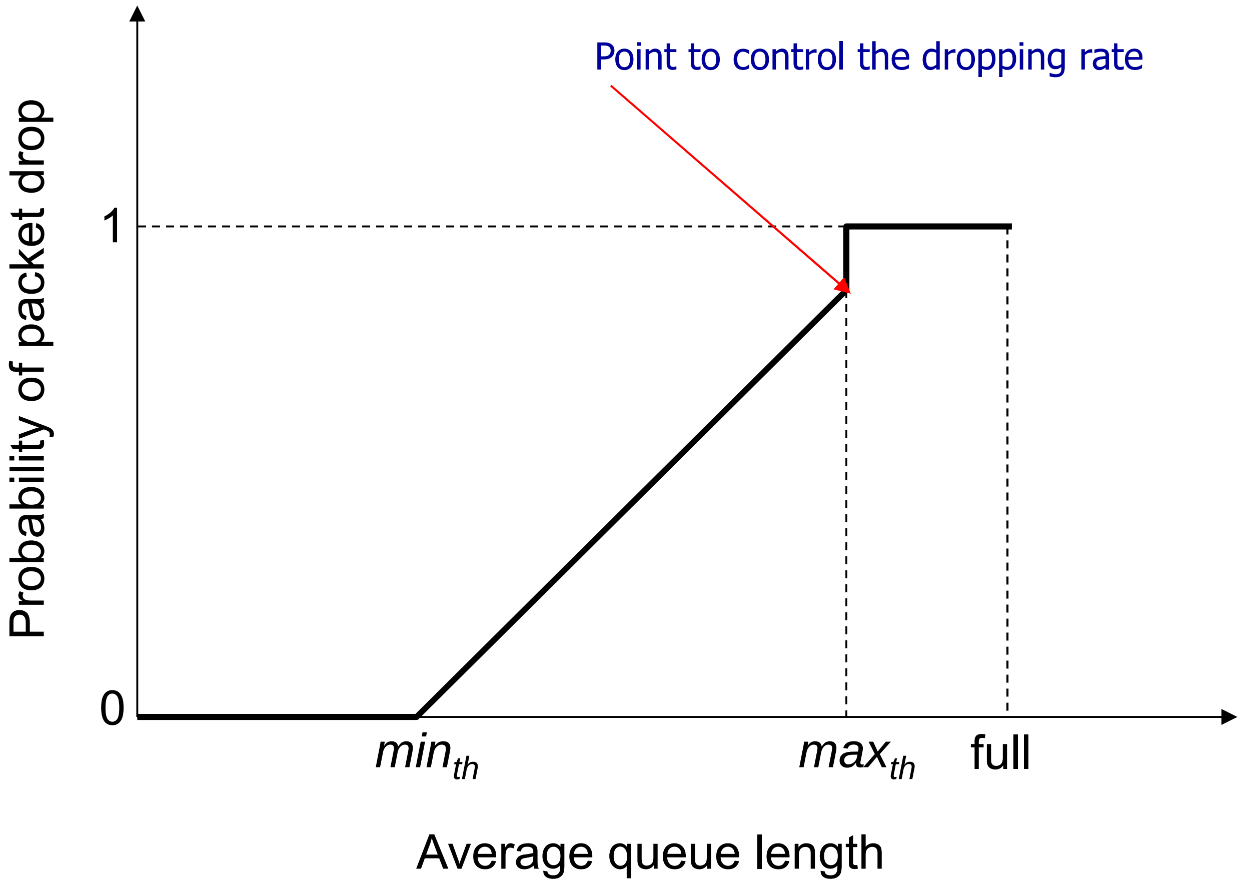
# Random Early Detection (RED)

- Packets produced by TCP will reduce input rate in response to network congestion
- Early drop: discard packets before buffers are full
- Random drop causes some sources to reduce rate before others, causing gradual reduction in aggregate input rate

Algorithm:

- Maintain running average of queue length
- If $Q_{avg}$ < minthreshold, do nothing
- If $Q_{avg}$ > maxthreshold, drop packet
- If in between, drop packet according to probability
- Flows that send more packets are more likely to have packets dropped

# Packet Drop Profile in RED

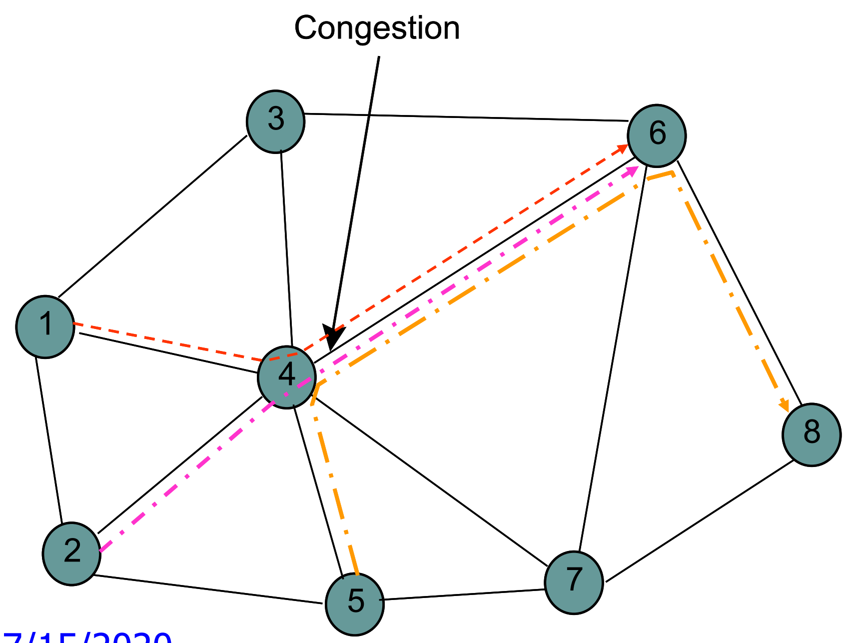


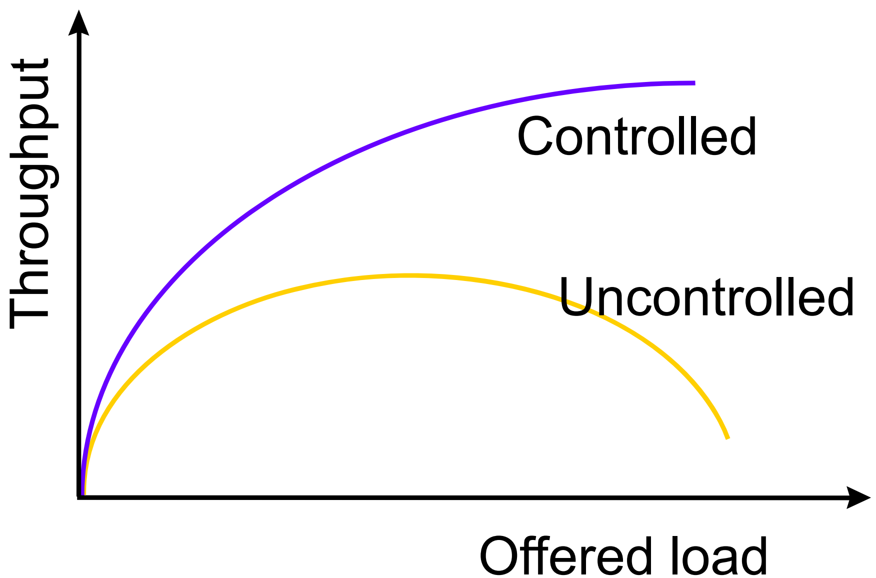
# Congestion Control –
# Flow-Level Traffic Management

# Congestion

Congestion occurs when a surge of traffic overloads network resources

- Approaches to Congestion Control:
  - Preventive Approaches: Scheduling & Reservations
  - Reactive Approaches: Detect & Throttle/Discard



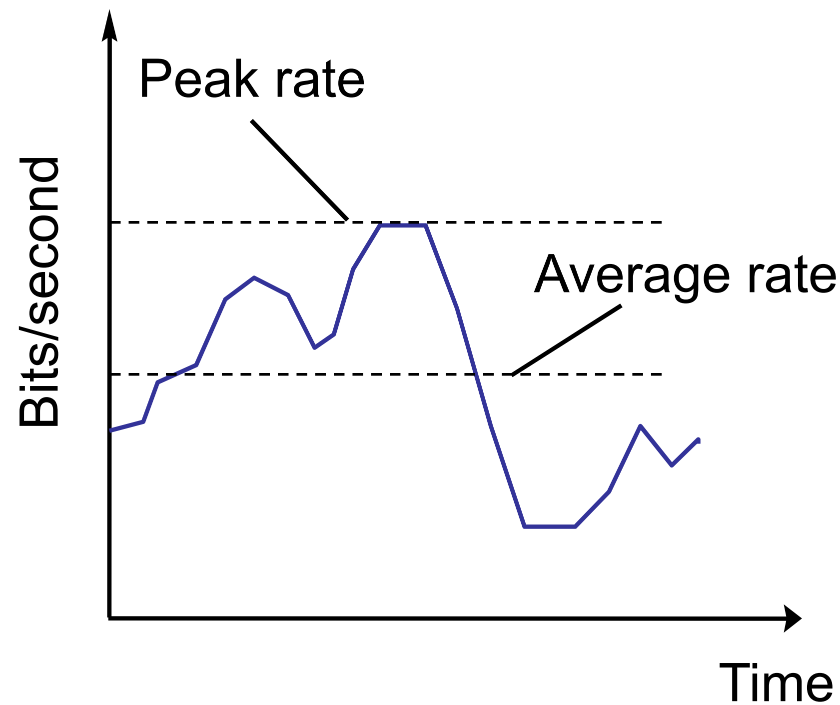



# Open-Loop Control

- Network performance is guaranteed to all traffic flows that have been admitted into the network
- Initially for connection-oriented networks
- Key Mechanisms
  - Admission Control
  - Policing
  - Traffic Shaping
  - Traffic Scheduling

# Admission Control

- Flows negotiate contract with network
- Specify requirements:
  - Peak, Avg., Min Bit rate
  - Maximum burst size
  - Delay, Loss requirement
- Network computes resources needed
  - "Effective" bandwidth
- If flow accepted, network allocates resources to ensure QoS delivered as long as source conforms to contract



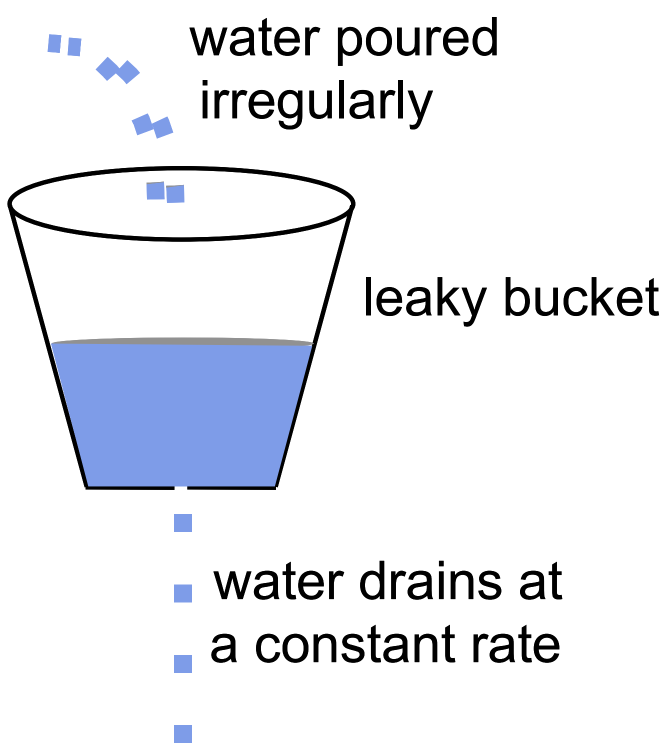

Typical bit rate demanded by a variable bit rate information source

# Policing

- Network monitors traffic flows continuously to ensure they meet their traffic contract
- When a packet violates the contract, network can discard or tag the packet, giving it lower priority
- If congestion occurs, tagged packets are discarded first
- *Leaky Bucket Algorithm* is the most commonly used policing mechanism
  - Bucket has specified leak rate for average contracted rate
  - Bucket has specified depth to accommodate variations in arrival rate
  - Arriving packet is *conforming* if it does not result in overflow

# Leaky Bucket Algorithm

Leaky Bucket algorithm can be used to police arrival rate of a packet stream




Leak rate corresponds to long-term rate

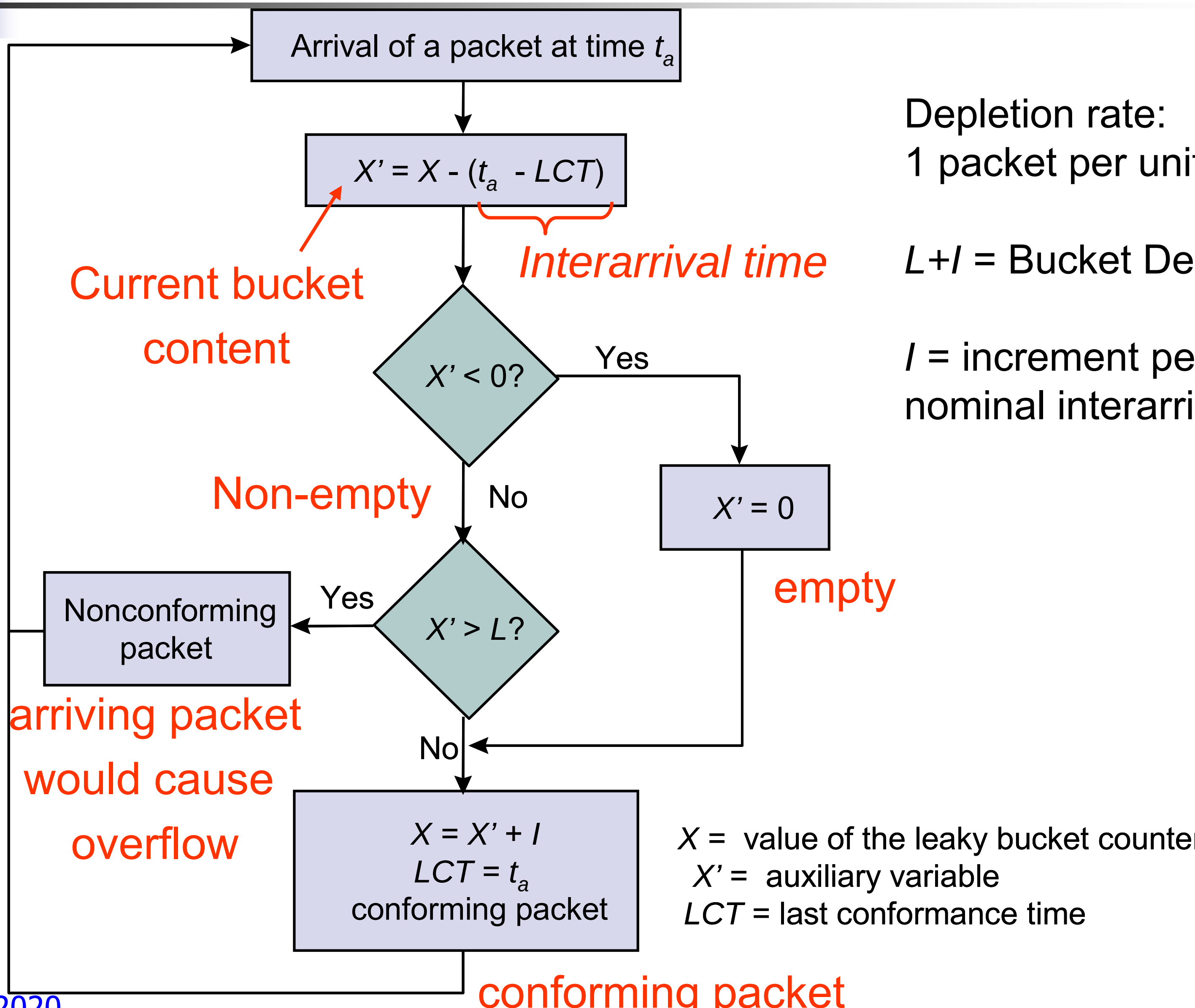
Bucket depth corresponds to maximum allowable burst arrival

1 packet per unit time
Assume constant-length packet as in ATM

Let X = bucket content at last conforming packet arrival

Let $t_a$ – last conforming packet arrival time = depletion in bucket

# Details of the Algorithm



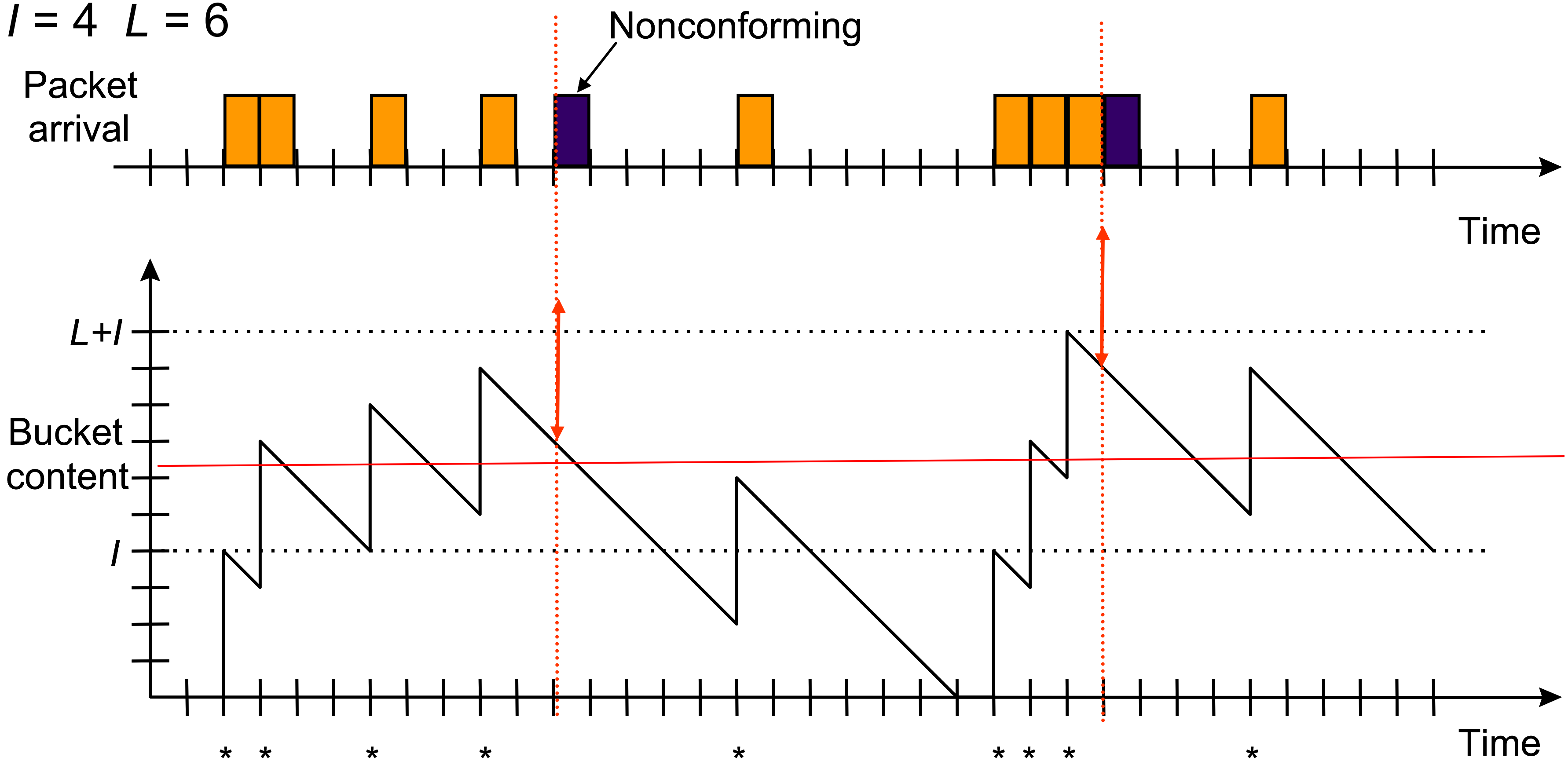

Depletion rate:
1 packet per unit time

$L+I$ = Bucket Depth

$I$ = increment per arrival, nominal interarrival time

# Leaky Bucket Example

$I = 4 \quad L = 6$



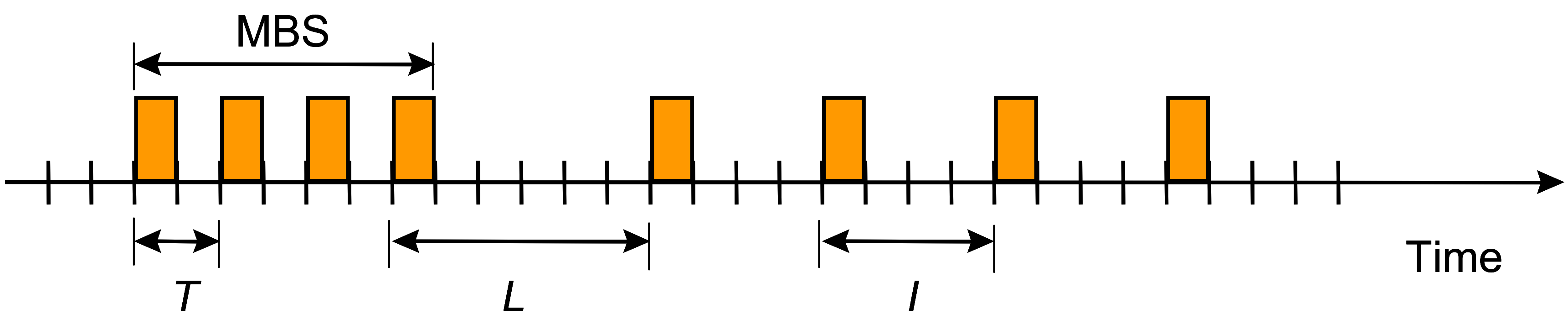

Non-conforming packets not allowed into bucket & hence not included in calculations

# Policing Parameters

*T* = 1 / peak rate
MBS = maximum burst size
*I* = nominal interarrival time = 1 / sustainable rate

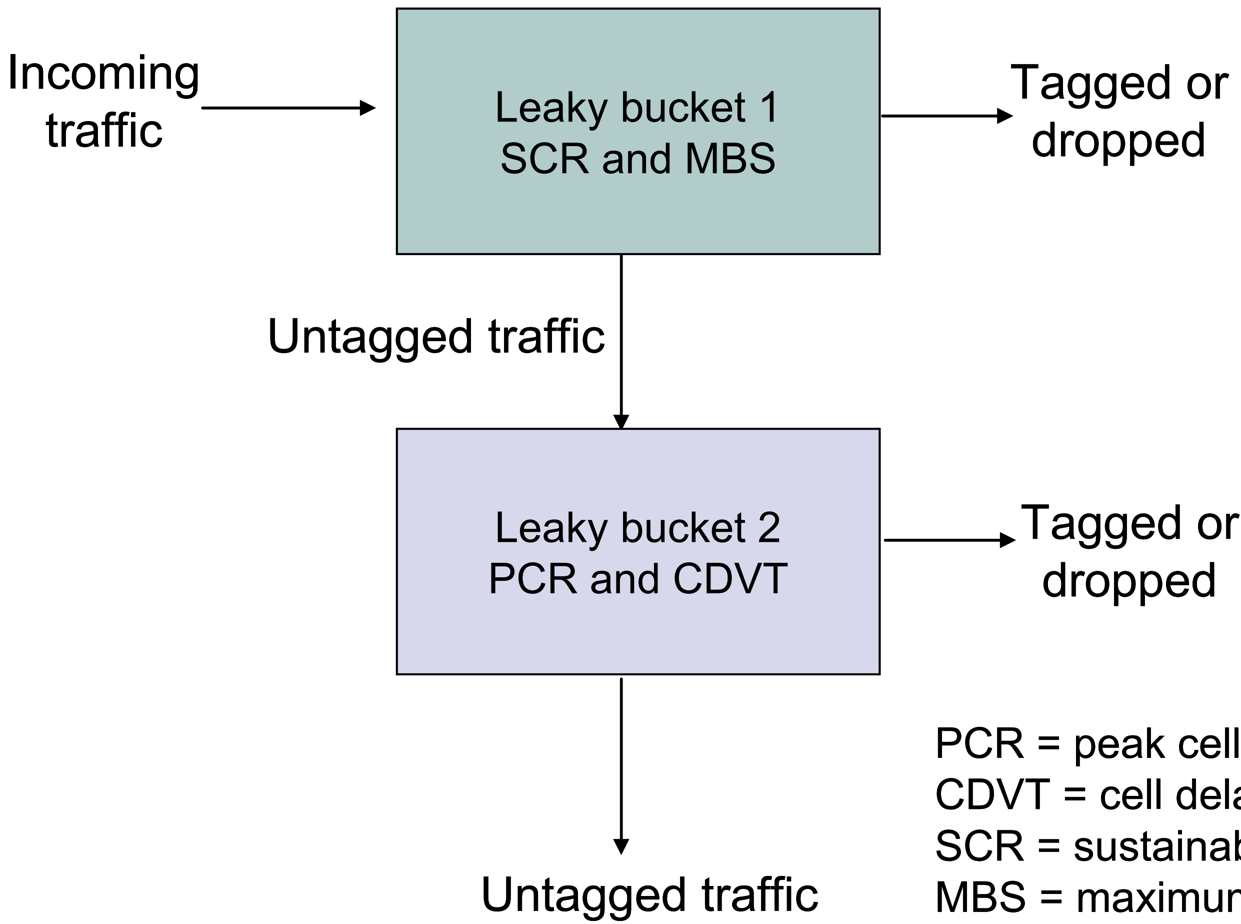
$$MBS = 1 + \left[ \frac{L}{I - T} \right]$$

After 1st packet, the content increases by I – T per packet arrival



# Dual Leaky Bucket

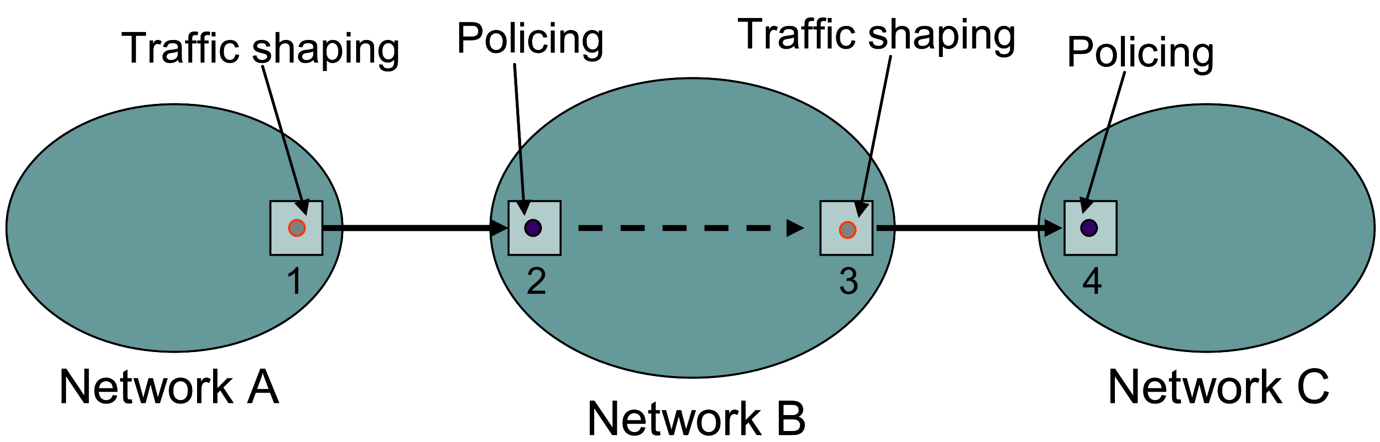
Dual leaky bucket to police PCR, SCR, and MBS:




PCR = peak cell rate
CDVT = cell delay variation tolerance
SCR = sustainable cell rate
MBS = maximum burst size

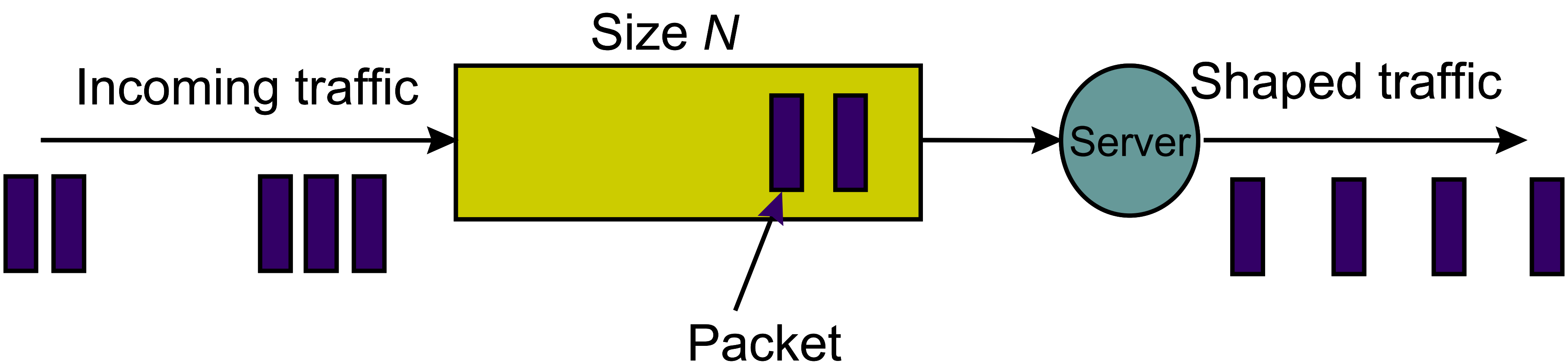
# Traffic Shaping

- Networks police the incoming traffic flow
- *Traffic shaping* is used to ensure that a packet stream conforms to specific parameters
- Networks can shape their traffic prior to passing it to another network



# Leaky Bucket Traffic Shaper

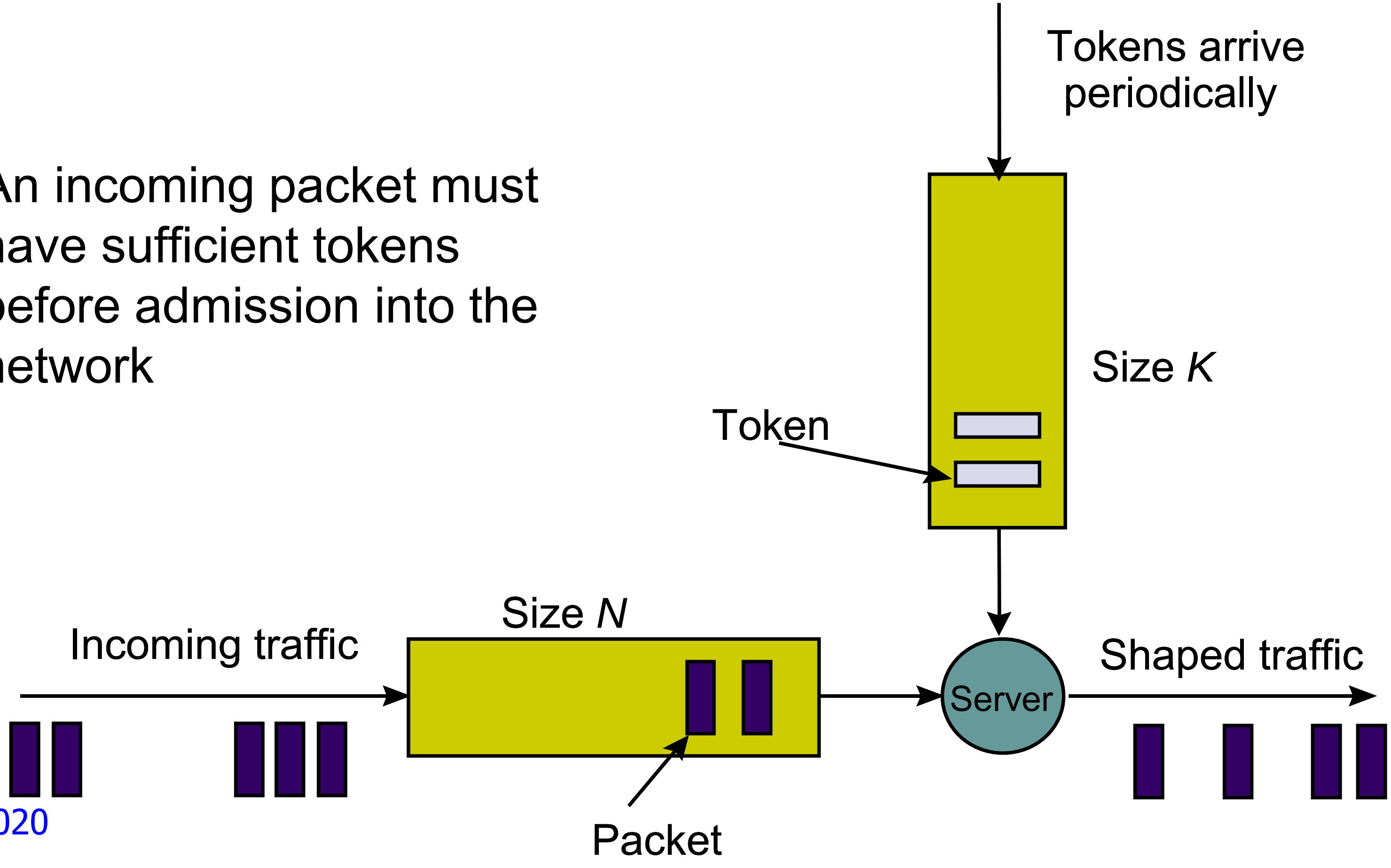
- Buffer incoming packets
- Play out periodically to conform to parameters
- Surges in arrivals are buffered & smoothed out
- Possible packet loss due to buffer overflow
- Too restrictive, since conforming traffic does not need to be completely smooth



# Token Bucket Traffic Shaper

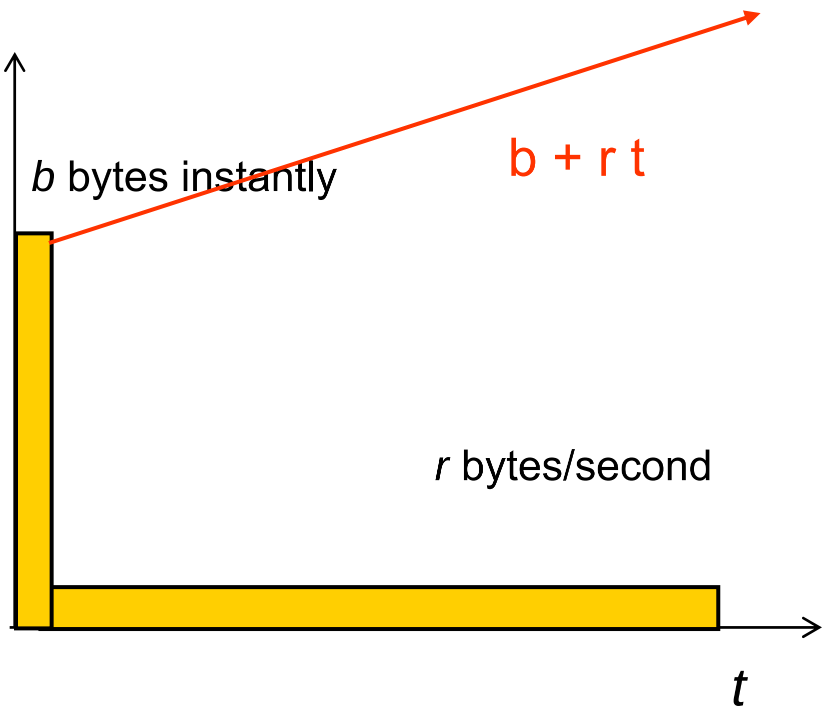
- Token rate regulates transfer of packets
- If sufficient tokens available, packets enter network without delay
- K determines how much burstiness allowed into the network

An incoming packet must have sufficient tokens before admission into the network



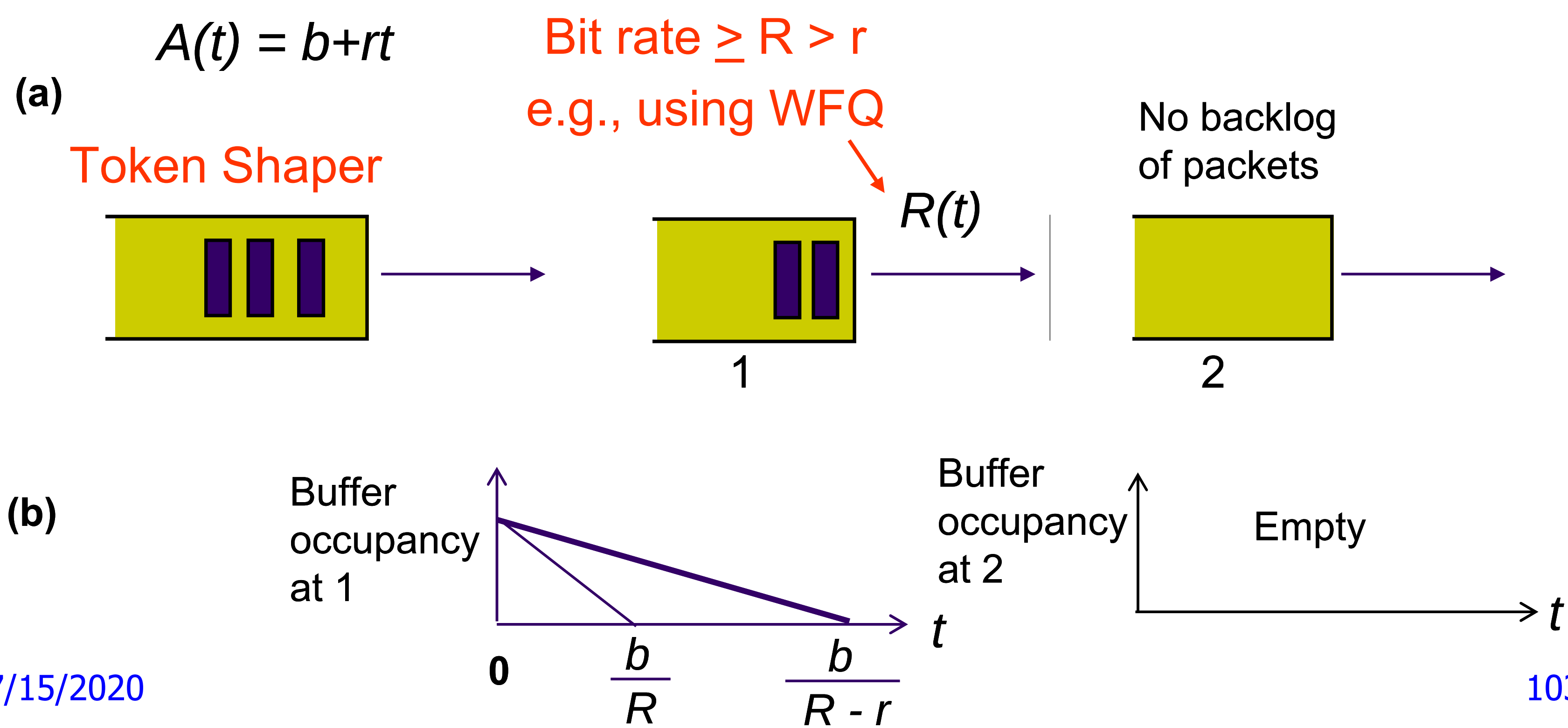
# Token Bucket Shaping Effect

The token bucket constrains the traffic from a source to be limited to *b* + *r t* bytes in an interval of length *t. r is the token rate in bytes/second.*



# Packet transfer with Delay Guarantees

- Assume fluid flow for information
- Token bucket allows burst of b bytes & then r bytes/second
  - Since R>r, buffer content @ 1 never greater than b bytes
  - Thus delay @ mux < b/R
- Rate into second mux is r < R, so bytes are never delayed



# Delay Bounds with WFQ / PGPS

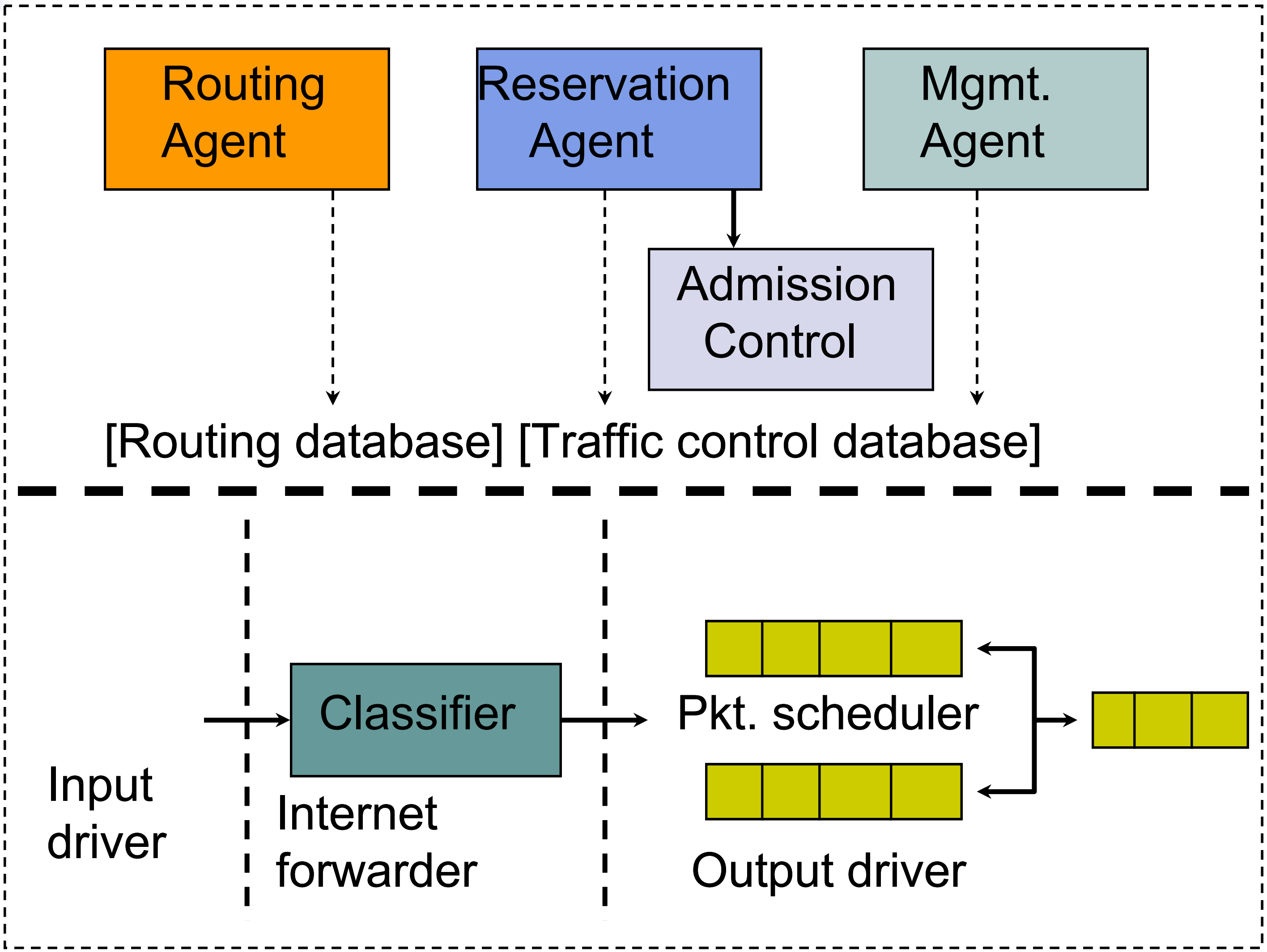
- Assume
  - traffic shaped to parameters b & r
  - schedulers give flow at least rate R>r
  - H hop path
  - m is maximum packet size for the given flow
  - M maximum packet size in the network
  - $R_j$ transmission rate in jth hop
- Maximum end-to-end delay that can be experienced by a packet from flow i is:

$$D \le \frac{b}{R} + \frac{(H-1)m}{R} + \sum_{j=1}^{H} \frac{M}{R_j}$$

# Scheduling for Guaranteed Service

- Suppose guaranteed bounds on end-to-end delay across the network are to be provided
- A call admission control procedure is required to allocate resources & set schedulers
- Traffic flows from sources must be shaped/regulated so that they do not exceed their allocated resources
- Strict delay bounds can be met

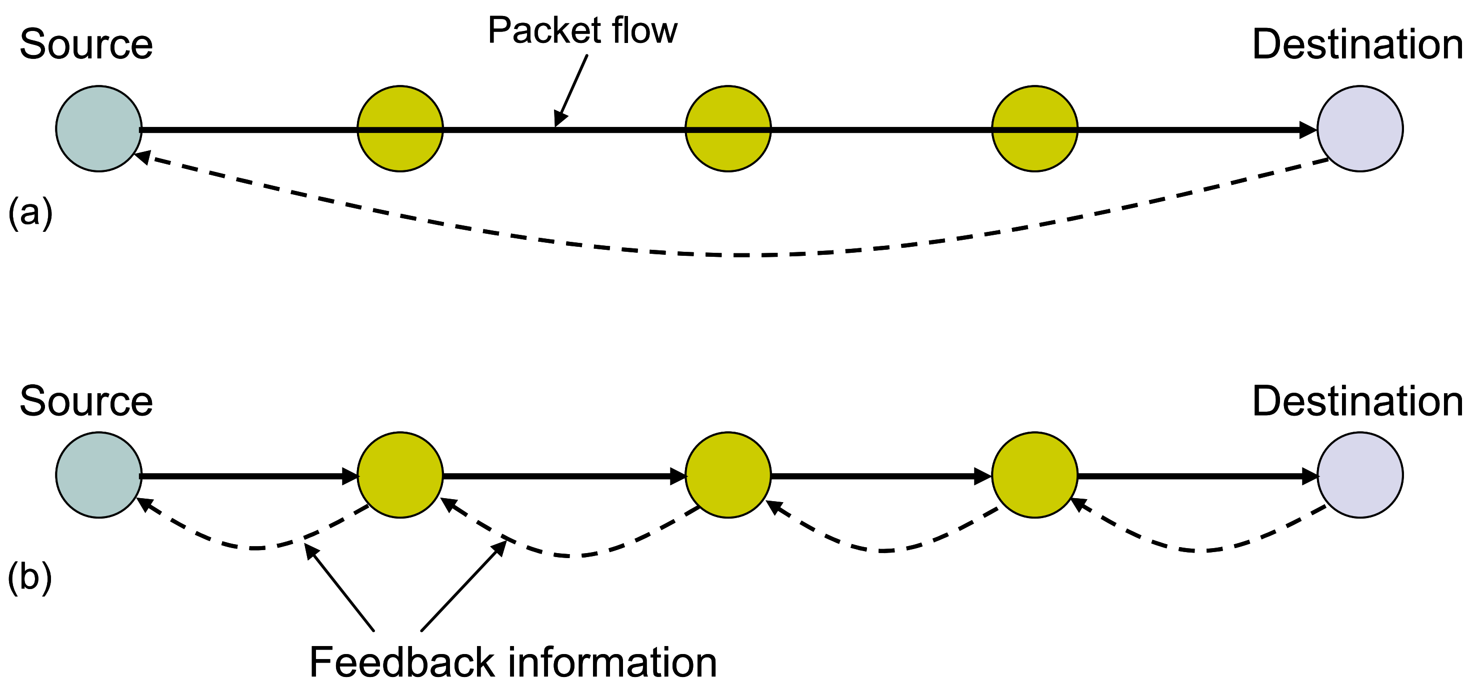
# Current View of Router Function



# Closed-Loop Flow Control

- Closed-Loop Congestion control
  - feedback information to regulate flow from sources into network
  - Based on buffer content, link utilization, etc.
  - Examples:  TCP at transport layer; congestion control at ATM level
- End-to-end vs. Hop-by-hop
  - Delay in effecting control
- Implicit vs. Explicit Feedback
  - Source deduces congestion from observed behavior
  - Routers/switches generate messages alerting to congestion

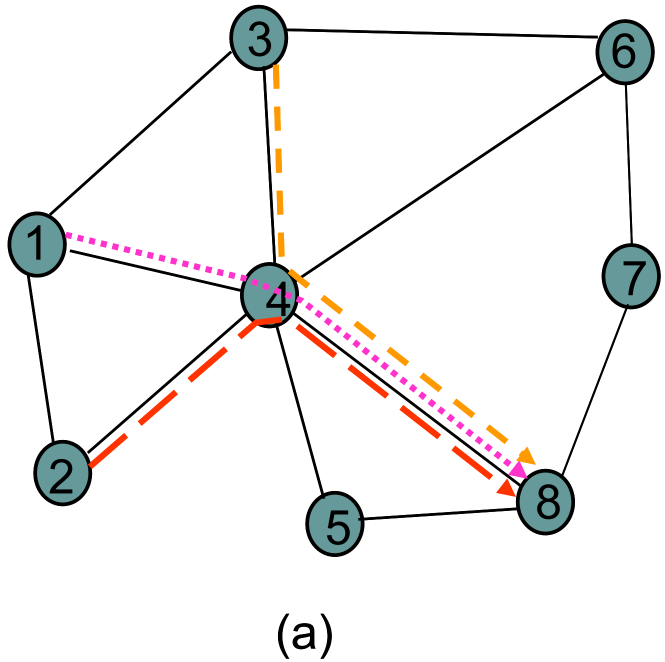
# End-to-End vs. Hop-by-Hop Congestion Control

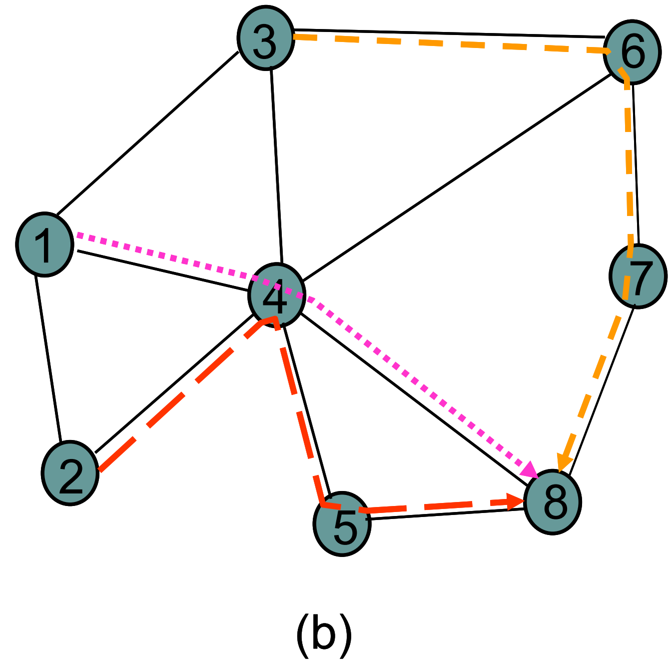


# Traffic Engineering

- Management exerted at flow aggregate level
- Distribution of flows in network to achieve efficient utilization of resources (bandwidth)
- Shortest path algorithm to route a given flow not enough
  - Does not take into account requirements of a flow, e.g. bandwidth requirement
  - Does not take account interplay between different flows
- Must take into account aggregate demand from all flows

# Example




Shortest path routing congests link 4 to 8

Better flow allocation distributes flows more uniformly