EECS 489
Computer Networks

Routing Fundamentals

Agenda

- Finishing up from last lecture
- Routing fundamentals

Router-assisted congestion control

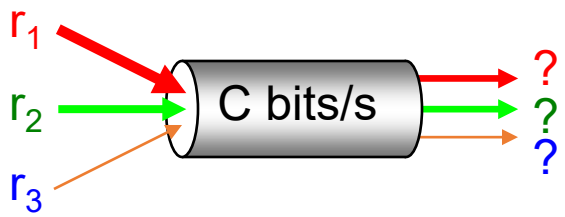
- Three tasks for congestion control
  - Isolation/fairness
  - Adjustment
  - Detecting congestion

Fairness: General approach

- Routers classify packets into “flows”
  - Let’s assume flows are TCP connections
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion
  - When line becomes free, take packet from next flow in a fair order
- What does “fair” mean exactly?

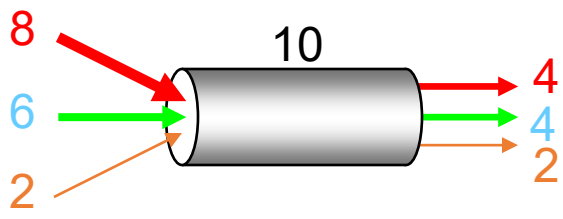
Max-Min fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where f is the unique value such that $\text{Sum}(a_i) = C$



Example

- C = 10; r_1 = 8, r_2 = 6, r_3 = 2; N = 3
- C/3 = 3.33 →
  - r_3 needs only 2
    - Can service all of r_3
  - Remove r_3 from the accounting: C = C – r_3 = 8; N = 2
- C/2 = 4 →
  - Can't service all of r_1 or r_2
  - So hold them to the remaining fair share: f = 4



f = 4:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Max-Min fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where f is the unique value such that $\text{Sum}(a_i) = C$
- If you don't get full demand, no one gets more than you
- This is what round-robin service gives if all packets are the same size

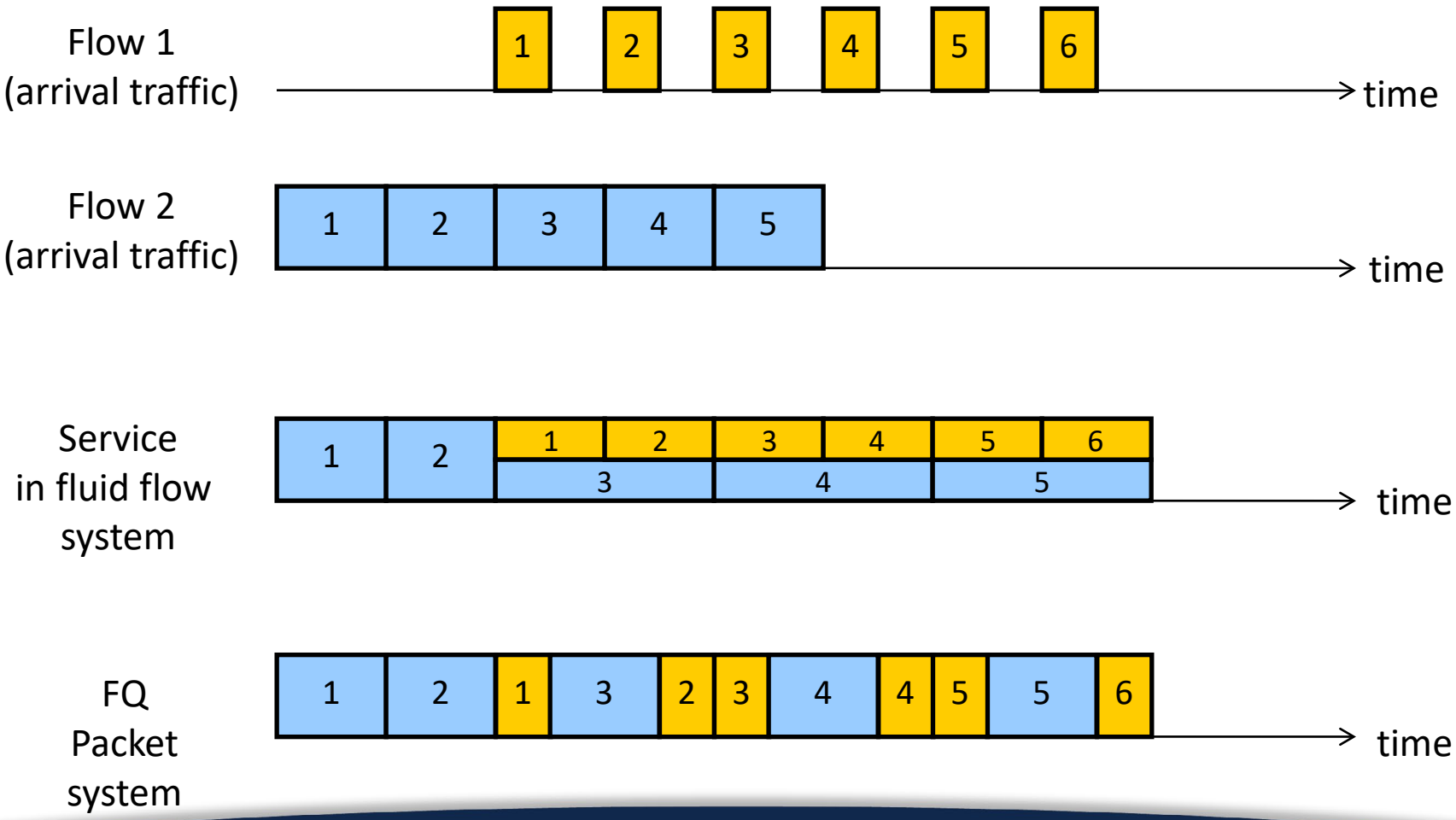
How do we deal with packets of different sizes?

- Mental model: Bit-by-bit round robin ("fluid flow")
- Can you do this in practice?
  - No, packets cannot be preempted
- But we can approximate it
  - This is what "fair queuing" routers do

Fair Queuing (FQ)

- For each packet, compute the time at which the last bit of a packet would have left the router if flows are served bit-by-bit
- Then serve packets in the increasing order of their deadlines

Example


Flow 1
(arrival traffic)
1
2
3
4
5
6
time
Flow 2
(arrival traffic)
1
2
3
4
5
time
Service
in fluid flow
system
1
2
1
2
3
4
5
6
3
4
5
time
FQ
Packet
system
1
2
1
3
2
3
4
4
5
5
6
time

Fair Queuing (FQ)

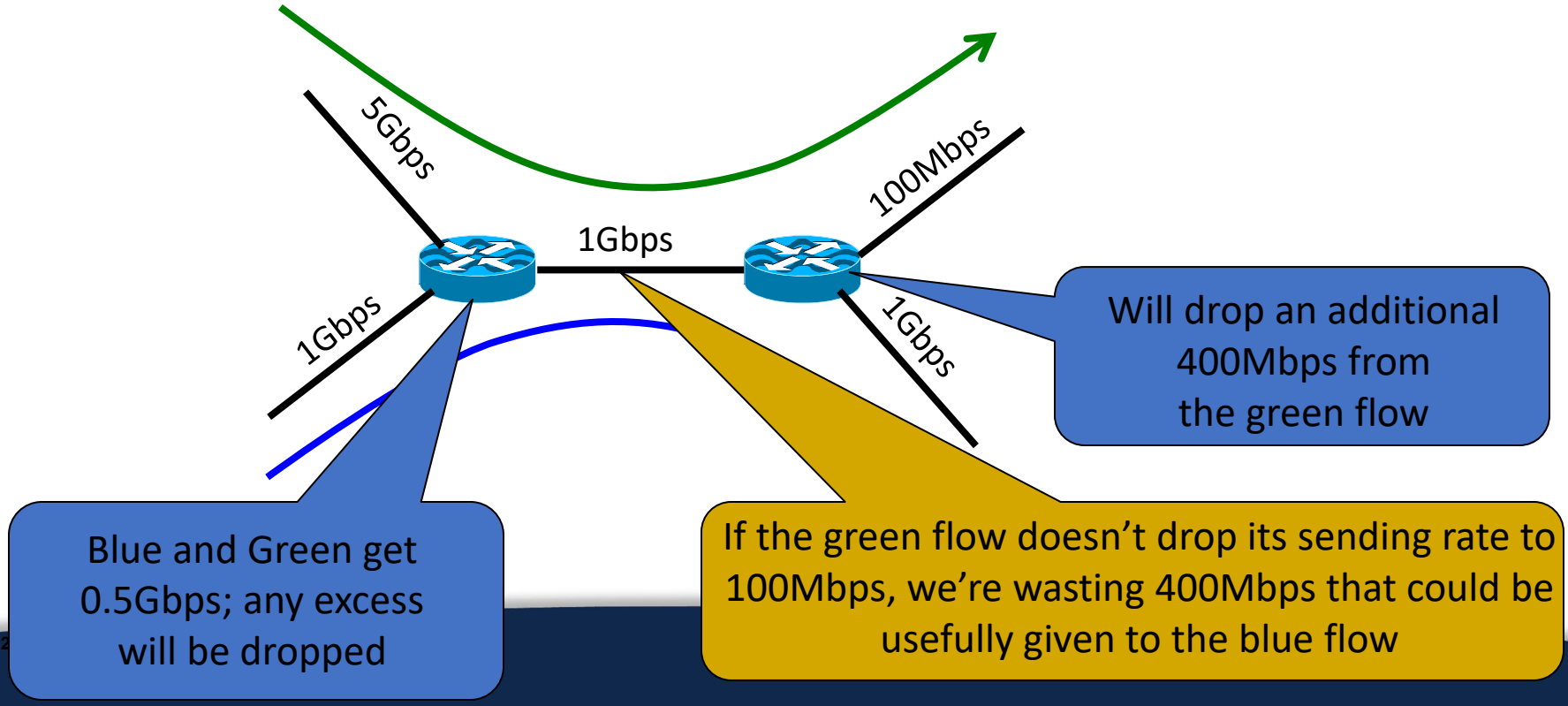
- Implementation of round-robin generalized to the case where not all packets are equal sized
- Weighted fair queuing (WFQ): assign different flows different shares
- Today, some form of WFQ implemented in almost all routers
  - Not the case in the 1980-90s, when CC was being developed
  - Mostly used to isolate traffic at larger granularities (e.g., per-prefix)

FQ vs. FIFO

- FQ advantages:
  - Isolation: cheating flows don't benefit
  - Bandwidth share does not depend on RTT
  - Flows can pick any rate adjustment scheme they want
- Disadvantages:
  - More complex than FIFO: per flow queue/state, additional per-packet book-keeping

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion



FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion
  - Robust to cheating, variations in RTT, details of delay, reordering, retransmission, etc.
- But congestion (and packet drops) still occurs
- We still want end-hosts to discover/adapt to their fair share!
- What would the end-to-end argument say w.r.t. congestion control?

Fairness is a controversial goal

- What if you have 8 flows, and I have 4?
  - Why should you get twice the bandwidth?
- What if your flow goes over 4 congested hops, and mine only goes over 1?
  - Why shouldn't you be penalized for using more scarce bandwidth?
- What is a flow anyway?
  - TCP connection
  - Source-Destination pair?
  - Source?

Router-Assisted Congestion Control

- CC has three different tasks:
  - Isolation/fairness
  - Rate adjustment
  - Detecting congestion

Why not let routers tell what rate end hosts should use?

- Packets carry “rate field”
- Routers insert “fair share” f in packet header
- End-hosts set sending rate (or window size) to f
  - Hopefully (still need some policing of end hosts!)
- This is the basic idea behind the “Rate Control Protocol” (RCP) from Dukkipati et al. ’07
  - Flows react faster

Router-Assisted Congestion Control

- CC has three different tasks:
  - Isolation/fairness
  - Rate adjustment
  - Detecting congestion

Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Many options for when routers set the bit
  - Tradeoff between (link) utilization and (packet) delay
- Congestion semantics can be exactly like that of drop
  - i.e., end-host reacts as though it saw a drop

ECN

- Advantages:
  - Don’t confuse corruption with congestion; recovery w/ rate adjustment
  - Can serve as an early indicator of congestion to avoid delays
  - Easy (easier) to incrementally deploy
    - Today: defined in RFC 3168 using ToS/DSCP bits in the IP header
    - Common in datacenters

Routing

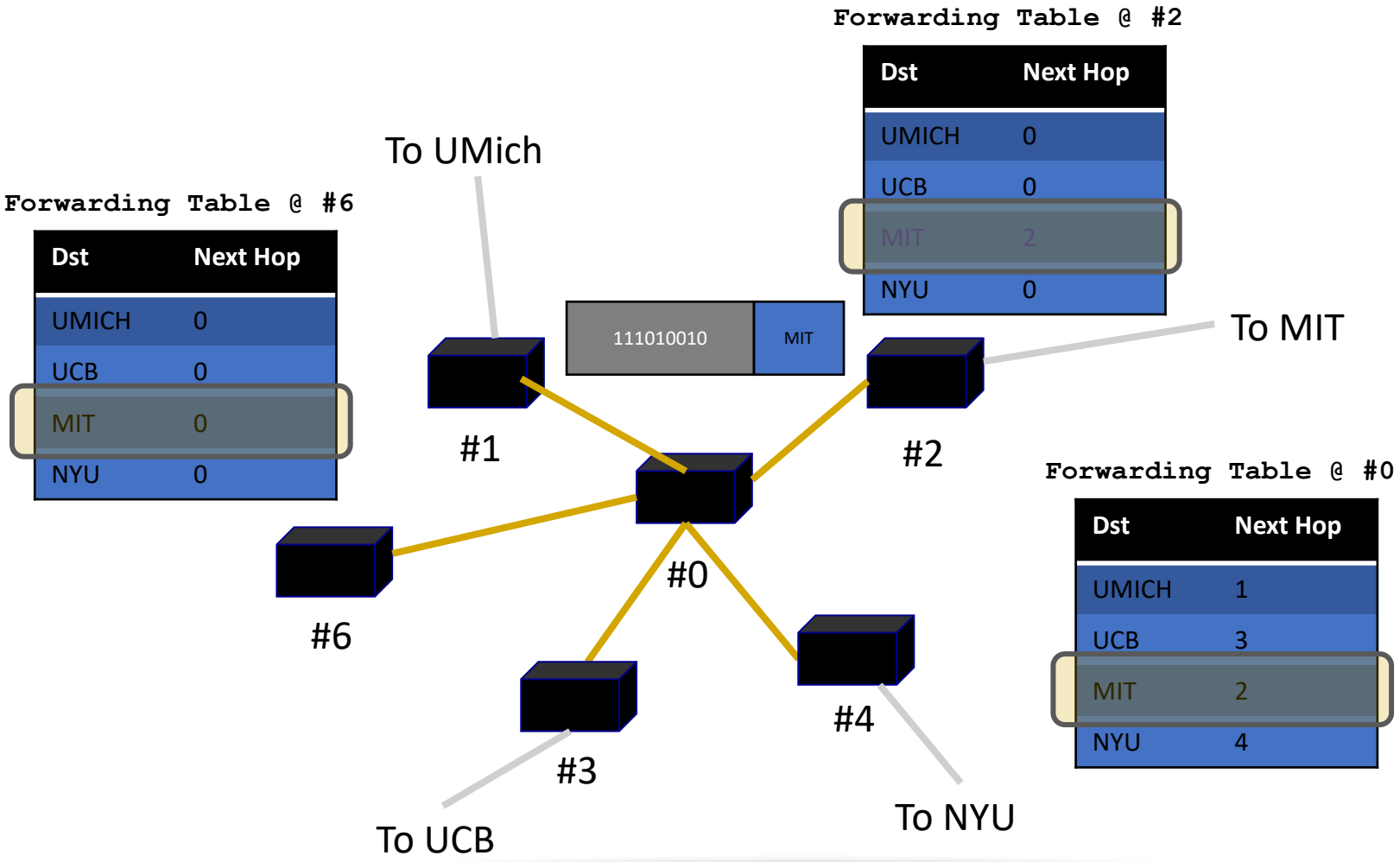
Goal of routing

- Find a path to a given destination
- How do we know that the state contained in forwarding tables meets our goal?
  - This is what "validity" of routing state tells us
  - [This is non-standard terminology]

Local vs. global view of state

- *Local* routing state is the forwarding table in a single router
  - By itself, the state in a single router cannot be evaluated
  - It must be evaluated in terms of the global context

Example:
Local vs. global view of state



Local vs. global view of state

- *Local* routing state is the forwarding table in a single router
  - By itself, the state in a single router cannot be evaluated
  - It must be evaluated in terms of the global context
- *Global* state refers to the collection of forwarding tables in each of the routers
  - Global state determines which paths packets take
  - (Will discuss later where this routing state comes from)

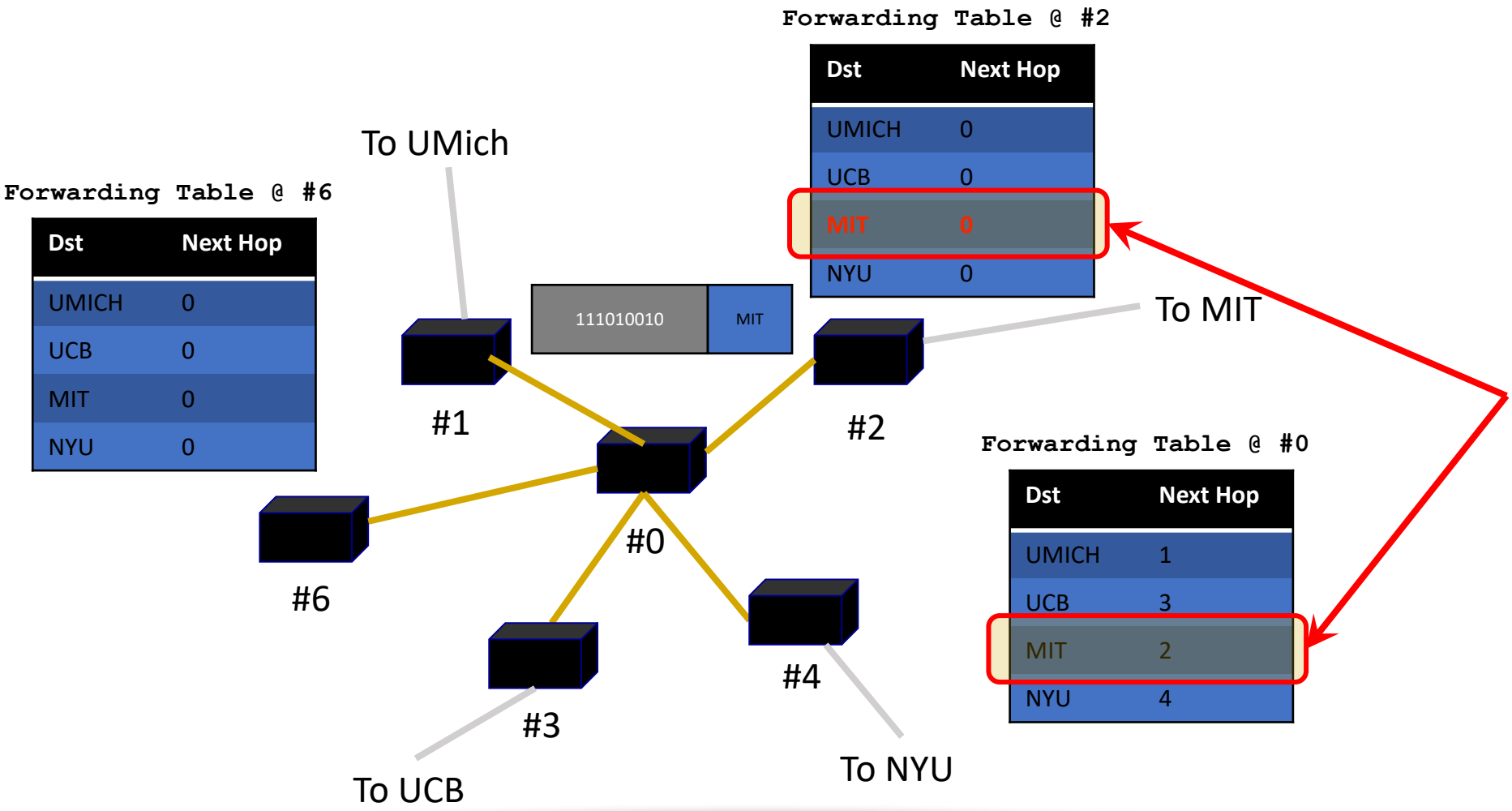
“Valid” routing state

- Global state is “valid” if it produces forwarding decisions that always deliver packets to their destinations
- Goal of routing protocols: compute valid state
  - How can we tell if routing state is valid?
- Need a succinct correctness condition for routing

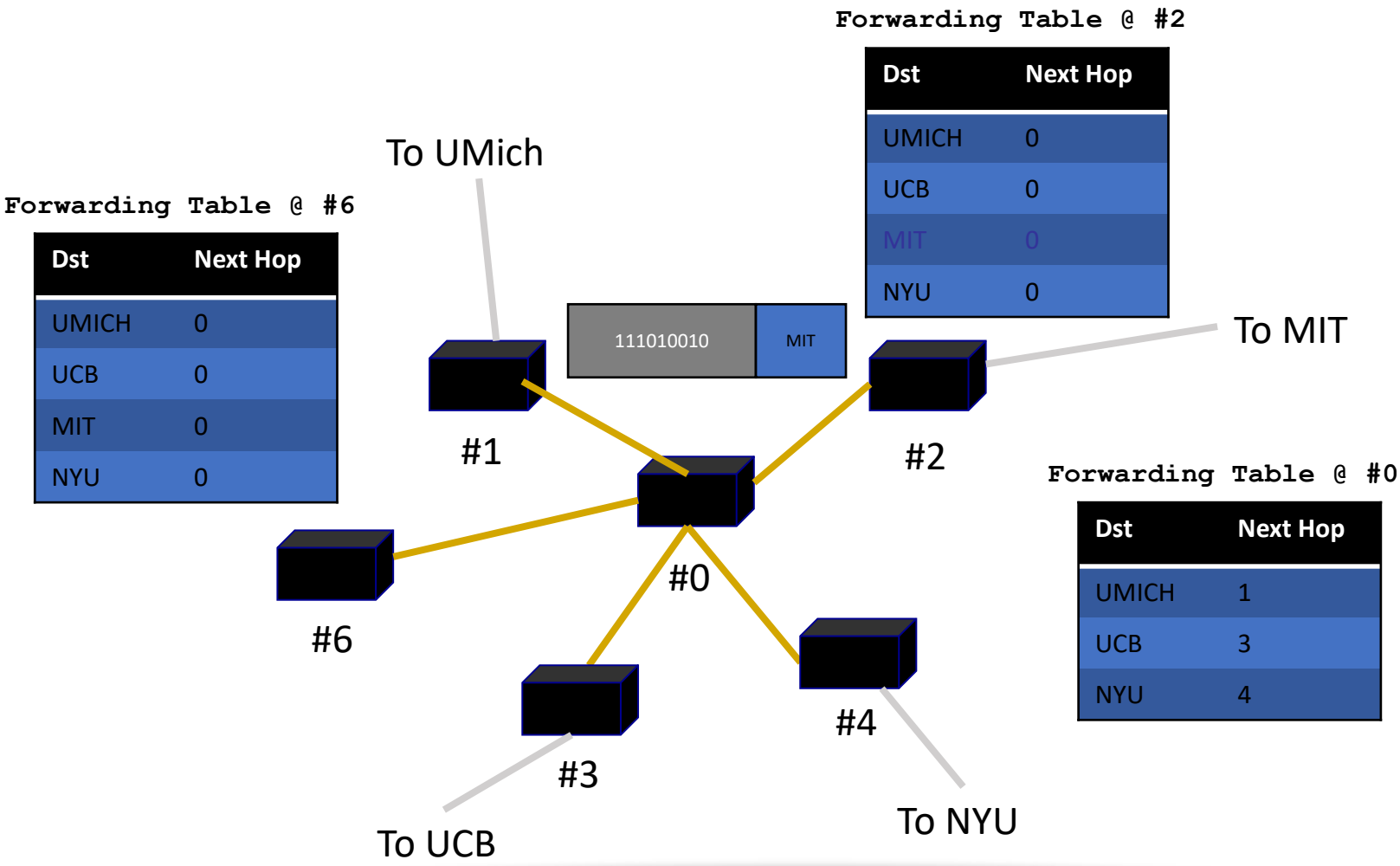
Necessary and sufficient condition

- Global routing state is valid *if and only if*:
  - There are no dead ends (other than destination)
  - There are no loops
- A dead end is when there is no outgoing link (next-hop)
  - A packet arrives, but the forwarding decision does not yield any outgoing link
- A loop is when a packet cycles around the same set of nodes forever

Loop!


Forwarding Table @ #2
Dst
Next Hop
UMICH
0
UCB
0
MIT
0
NYU
0
To UMich
Forwarding Table @ #6
Dst
Next Hop
UMICH
0
UCB
0
MIT
0
NYU
0
111010010
MIT
To MIT
#1
#2
#0
#6
#3
#4
Forwarding Table @ #0
Dst
Next Hop
UMICH
1
UCB
3
MIT
2
NYU
4
To UCB
To NYU

Dead end to MIT @ #0



Necessary and sufficient condition

- Global routing state is valid *if and only if*:
  - There are no dead ends (other than destination)
  - There are no loops

Necessary (“only if”)

- If you run into a dead end before hitting destination,
  - you’ll never reach the destination
- If you run into a loop,
  - you’ll never reach destination

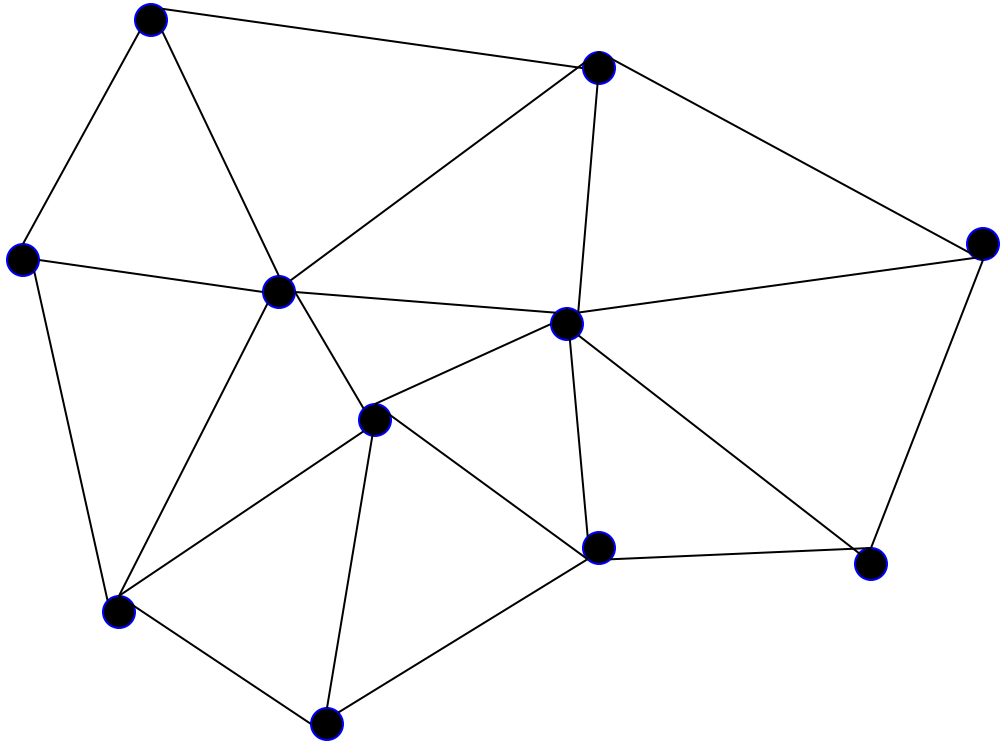
Sufficient ("if")

- Assume there are no dead ends and no loops
- Packet must keep wandering, but without repeating
  - If ever enter same switch from same link, will loop
- Only a finite number of possible links for it to visit
  - It cannot keep wandering forever without looping
  - Must eventually hit destination

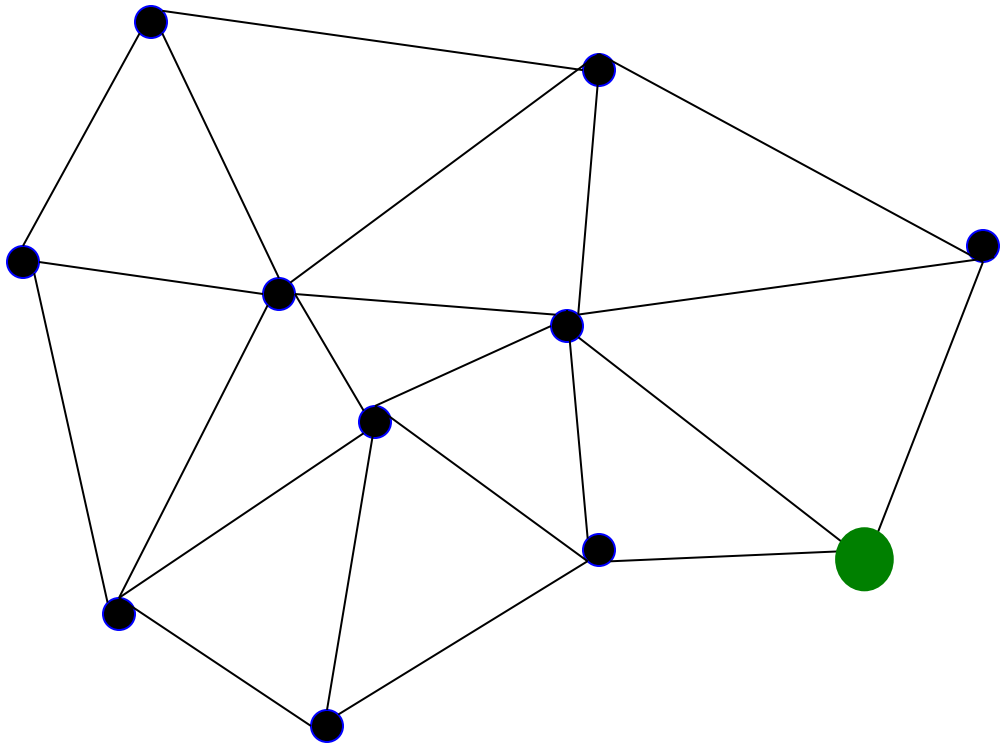
Checking validity of routing state

- Focus only on a single destination
  - Ignore all other routing state
- Mark outgoing link ("next hop") with arrow
  - There is only one at each node
- Eliminate all links with no arrows
- Look at what's left

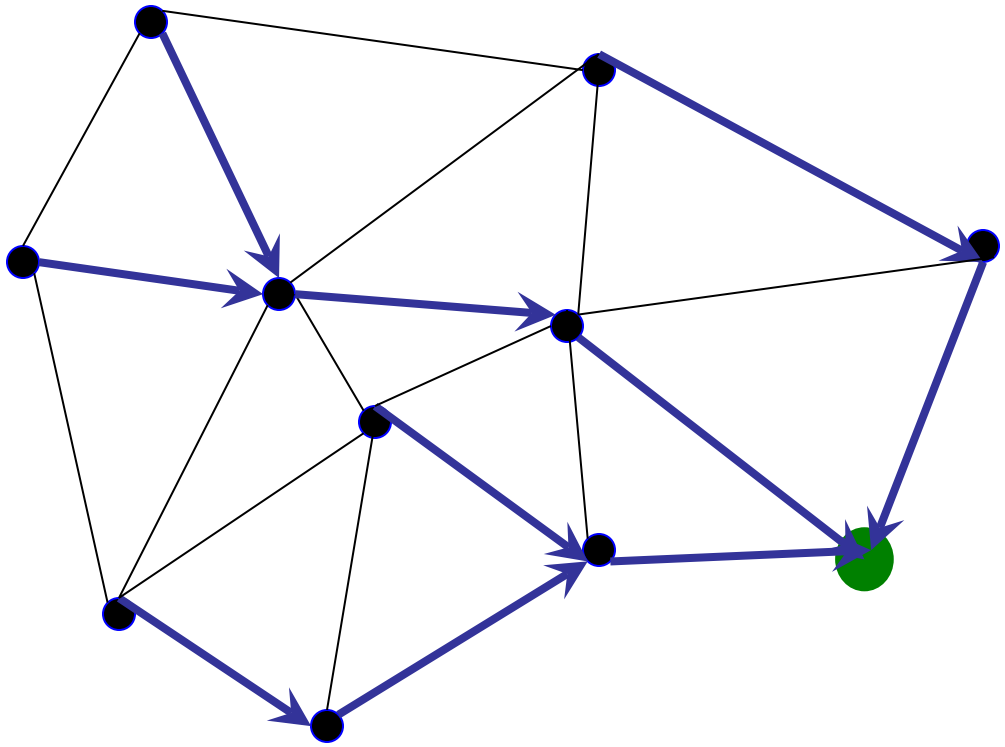
Example 1



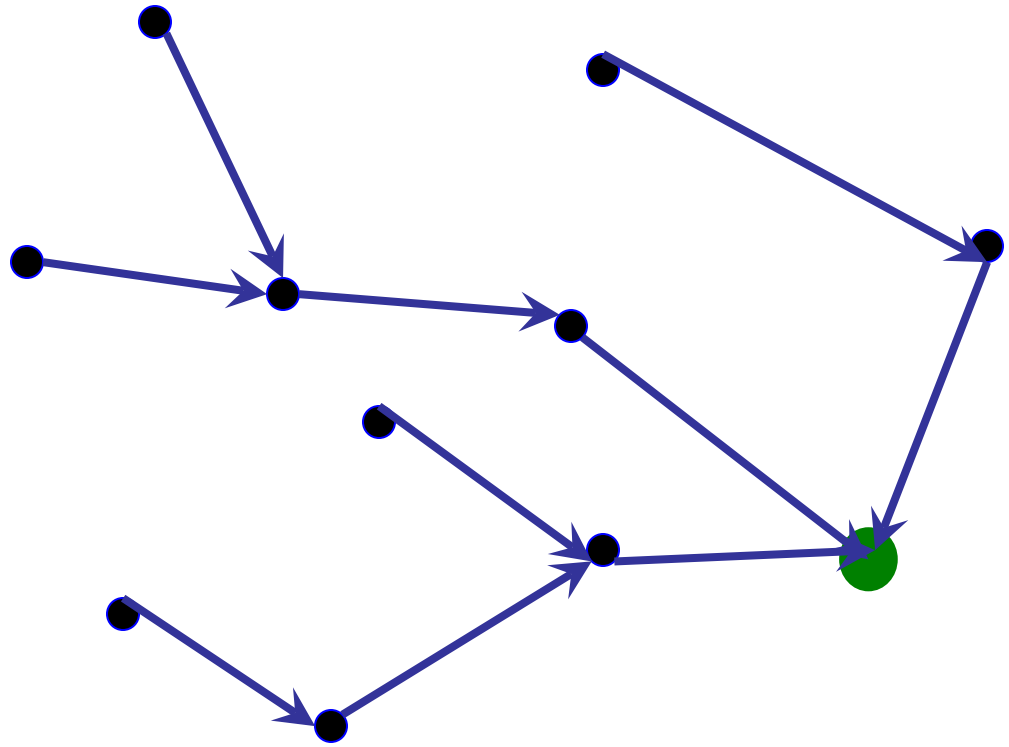
Pick destination



Put arrows on outgoing links (to green dot)

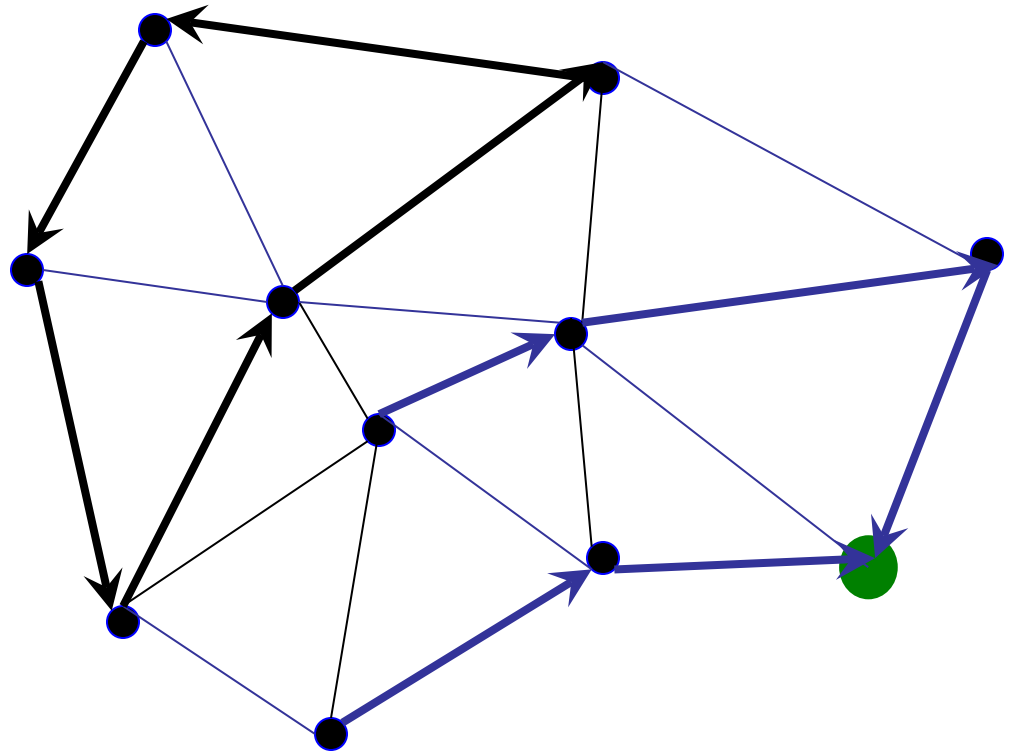


Remove unused links



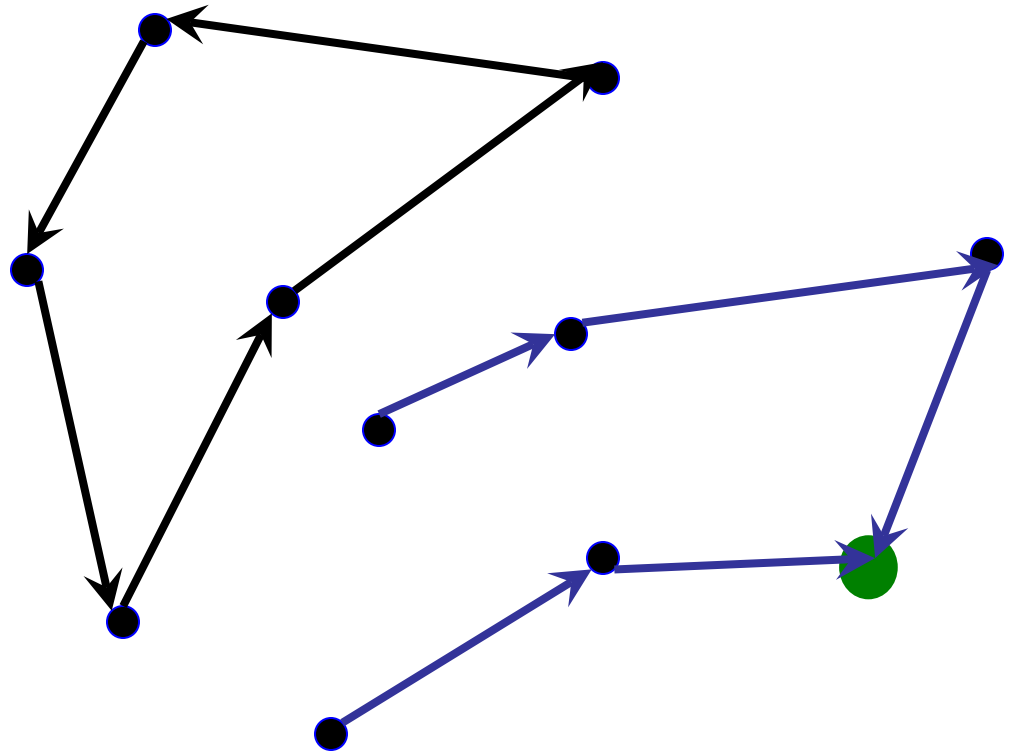
Leaves spanning tree: Valid

Example 2



Is this valid?

Not valid: Contains loop!



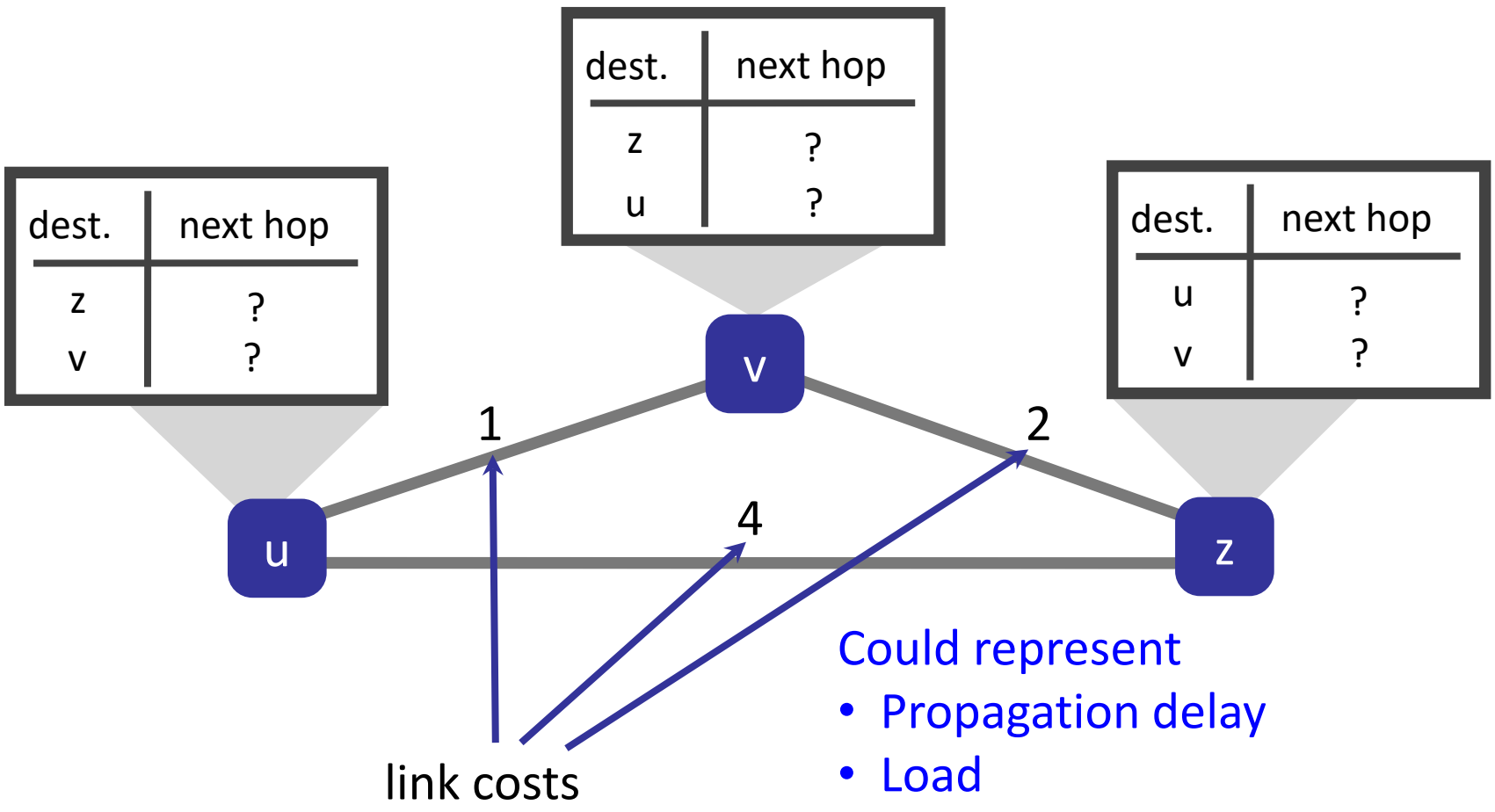
Routing validity

- Very easy to check validity of routing state for a particular destination
- Dead ends are nodes without outgoing arrow
- Loops are obvious too
  - Disconnected from rest of graph

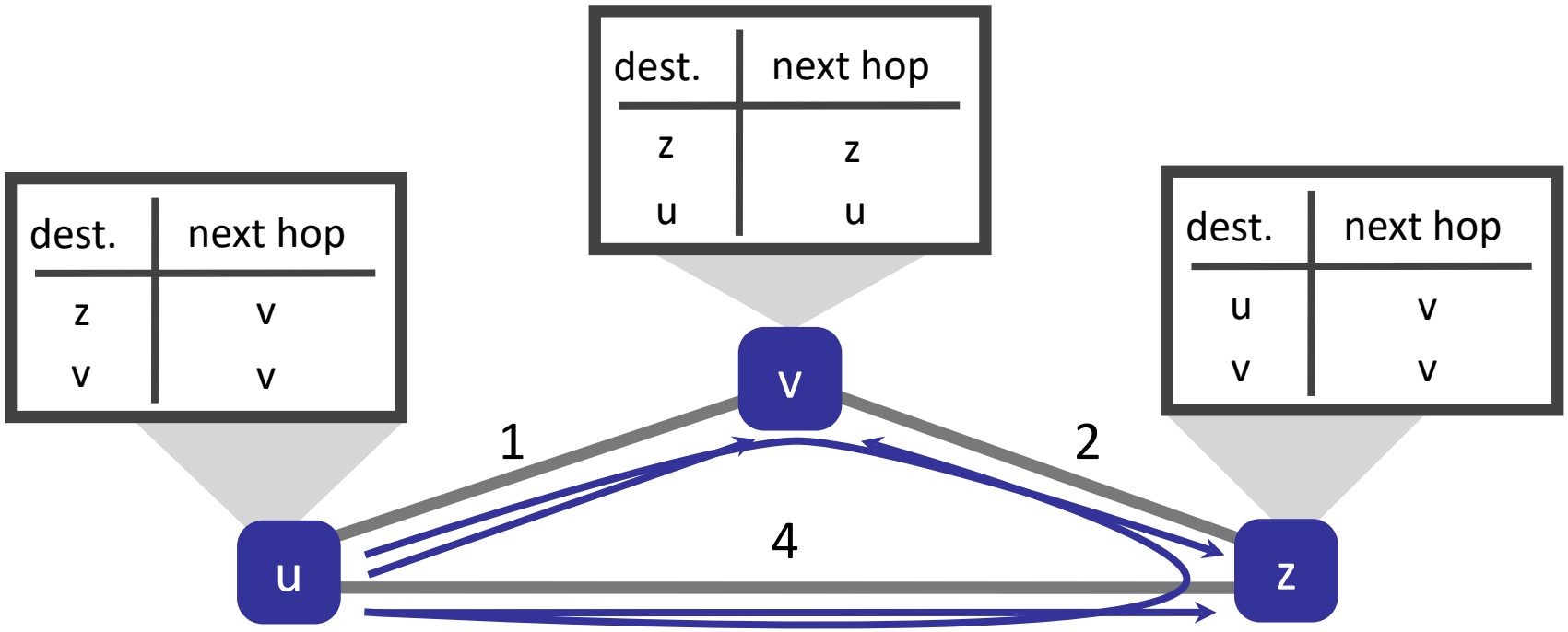
Goal of routing

- v1: Find a path to a given destination
- v2: Find a *least-cost path* to a given destination

Example


dest. | next hop
z | ?
u | ?
dest. | next hop
z | ?
v | ?
dest. | next hop
u | ?
v | ?
v
u
z
1
2
4
link costs
Could represent
• Propagation delay
• Load
• Cost

Example



least-cost path from u to z: u v z

least cost path from u to v: u v

Least-cost path routing

- Given: router graph & link costs
- Goal: find least-cost path
  - From each source router to each destination router

Least-cost routes

- Least-cost routes provide an easy way to avoid loops
  - No reasonable cost metric is minimized by traversing a loop
- Least-cost paths form a spanning tree for each destination rooted at that destination

EECS 281: Dijkstra's algorithm

- Network topology, link costs known to all nodes
  - All nodes have same info
- Computes least-cost paths from one node ("src") to all other nodes
  - After k iterations, know least-cost path to k destinations

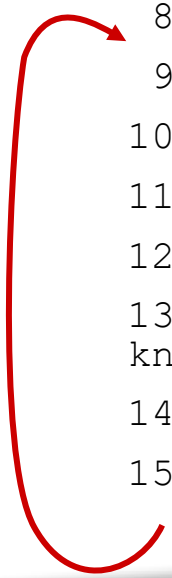
- **Notations**
  - c(x,y): link cost from x to y;
    - ∞ if not direct neighbors
  - D(v): current value of cost of path from src to dst v
  - p(v): predecessor node along path from source to v
  - N': set of nodes whose least-cost path definitively known

Dijkstra's algorithm

```
1  Initialization:
2    N' = {u}; D(u) = 0
3    for all nodes v
4      if v adjacent to u
5        then D(v) = c(u,v)
6      else D(v) = ∞
```

Dijkstra's algorithm

```
1   Initialization:
2     N' = {u}; D(u) = 0
3     for all nodes v
4       if v adjacent to u
5         then D(v) = c(u,v)
6       else D(v) = ∞
7
8   Loop
9     find w not in N' such that D(w) is a minimum
10    add w to N'
11    update D(v) for all v adjacent to w and not in N':
12        D(v) = min( D(v), D(w) + c(w,v) )
13        /* new cost to v is either old cost to v or
known
14         least path cost to w plus cost from w to v */
15  until all nodes are in N'
```

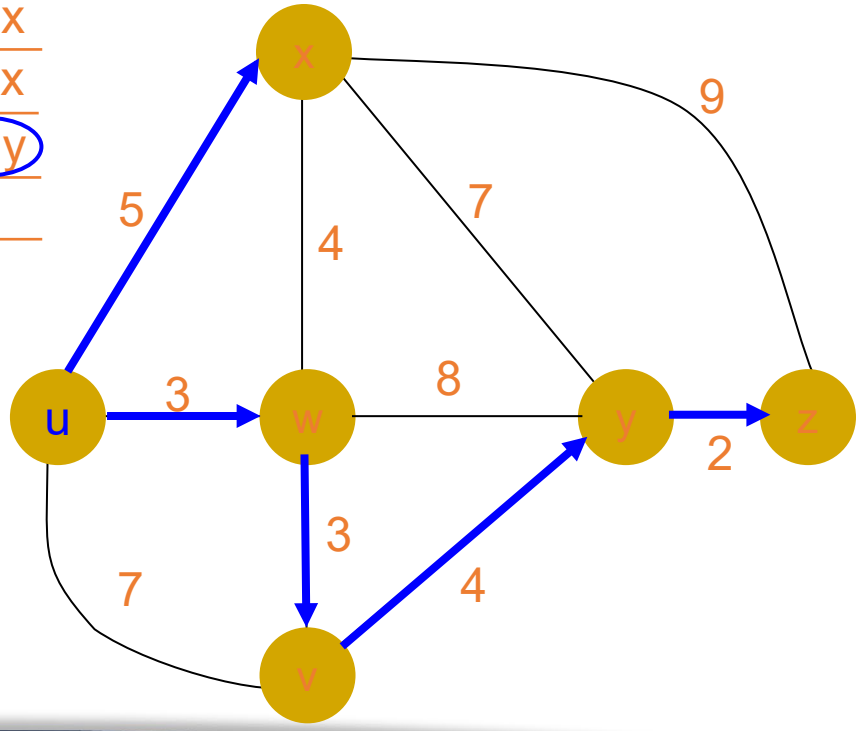


Dijkstra's algorithm: Example

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

Notes:

- Construct shortest path tree by tracing predecessor nodes
- Ties can exist (can be broken arbitrarily)

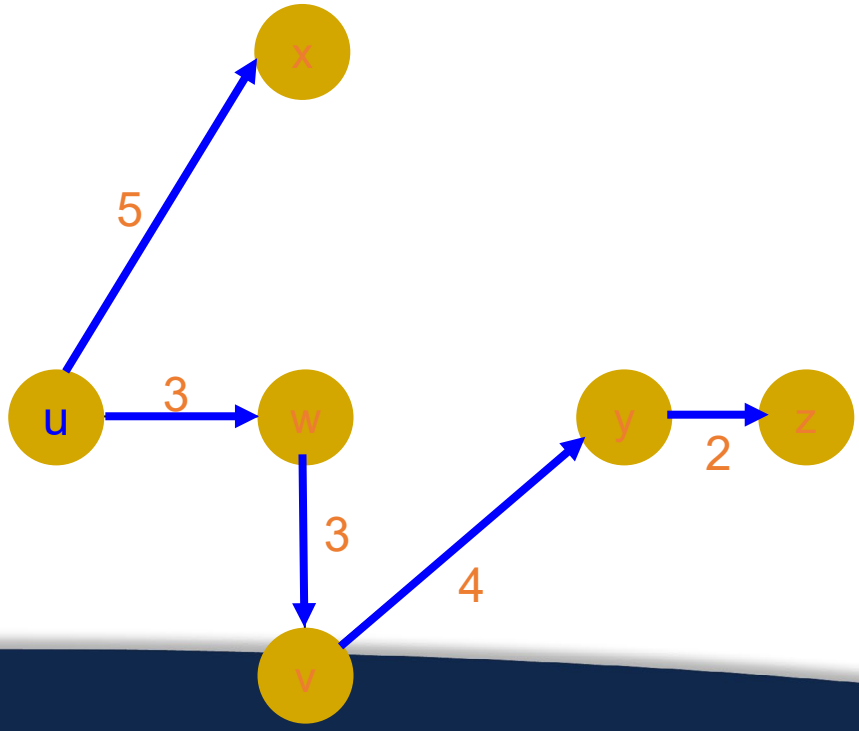


Dijkstra's algorithm: Example

Resulting forwarding table in u

destination	link
v	(u, w)
w	(u, w)
x	(u, x)
y	(u, w)
z	(u, w)

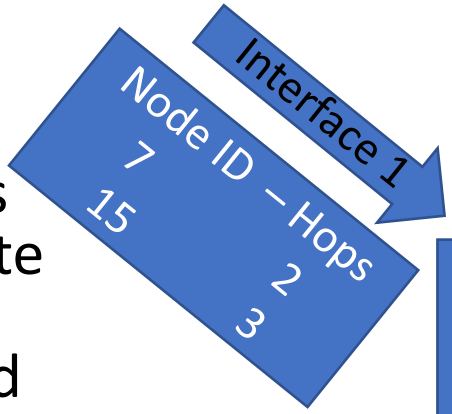
Resulting least-cost tree from u



Let's Try to learn about the network

- Send a message to your neighbors with your id
- Build your routing table based on this information
- Send to your neighbors your routing table (without the interface)
- When you receive information about nodes you don't know of, update your table with your neighbor as next hop and send you table to your neighbors

Node ID	Interface	Hops
5	1	1
8	2	1
4	3	1



Node ID	Interface	Hops
5	1	1
8	2	1
4	3	1
7	1	3
15	1	4

Let’s Try to learn about the network

- Send a message to your neighbors
- Send a message to each neighbor, informing them about each one of your neighbors
- If you receive this message from a neighbor, make copies and forward one copy to each neighbor

Summary

- Network layer control plane calculates valid routes and sets up forwarding table
  - Avoiding loops and dead ends
- Least-cost routes can be calculated using Dijkstra's algorithm

- Next lecture: Routing protocols

Bonus Quiz 11

- https://forms.gle/8e6vJUgRfadYNvQK9