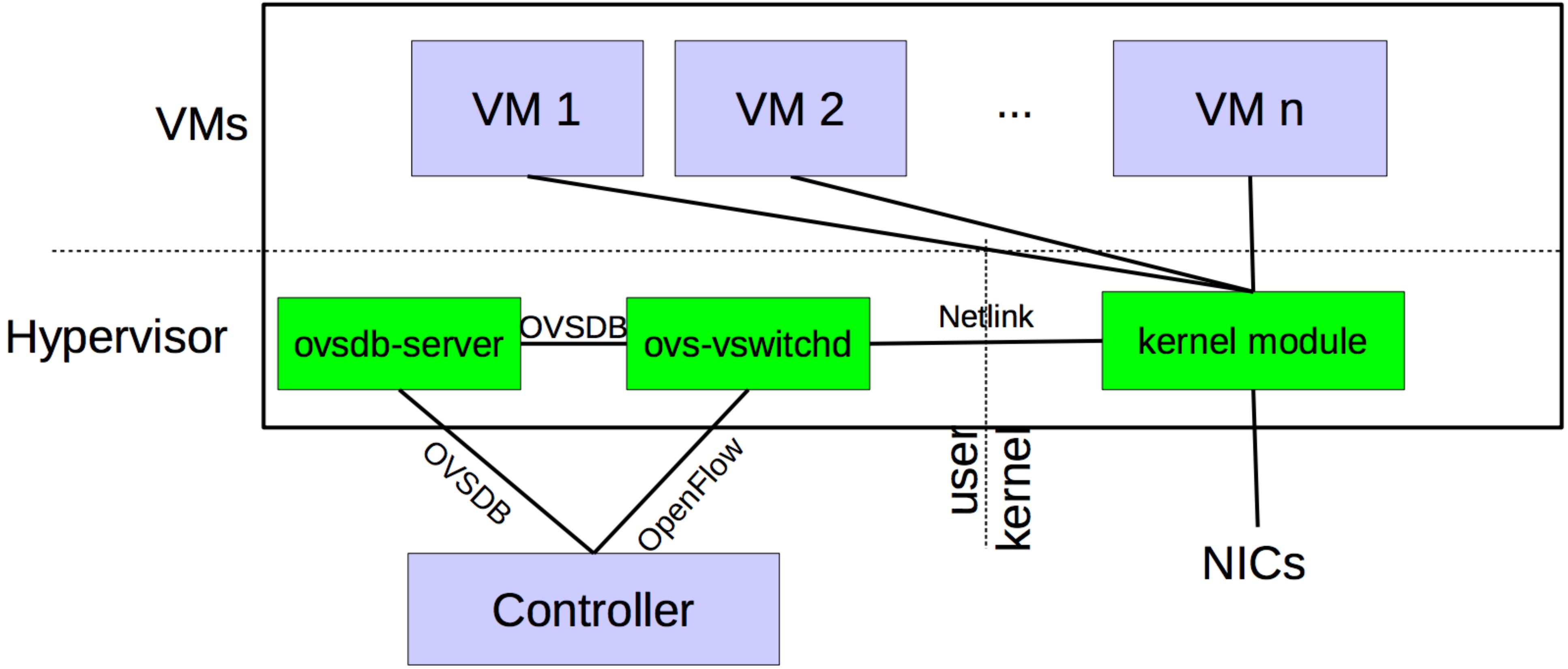
Network

RUTGERS
UNIVERSITY | NEW BRUNSWICK

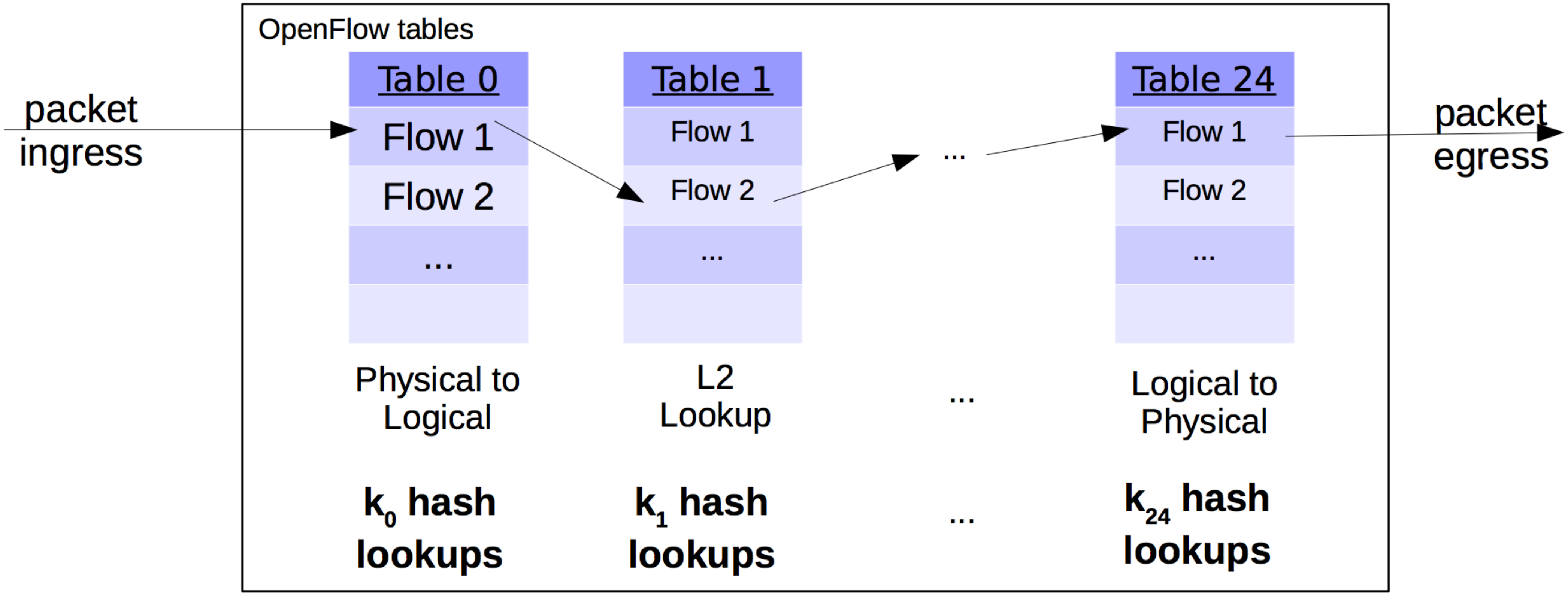
OpenVSwitch: Requirements

- Support large and complex policies
- Support updates in such policies, e.g., VM migration, new customers, …
- Don't take up too much resources (CPU must do useful work, not just policy processing)
- Process packets with high performance
    - High throughput and low delay

OVS design


VMs
VM 1
VM 2
...
VM n
Hypervisor
ovsdb-server
OVSDB
ovs-vswitchd
Netlink
kernel module
OVSDB
OpenFlow
user
kernel
NICs
Controller

First design: put OF tables in the kernel



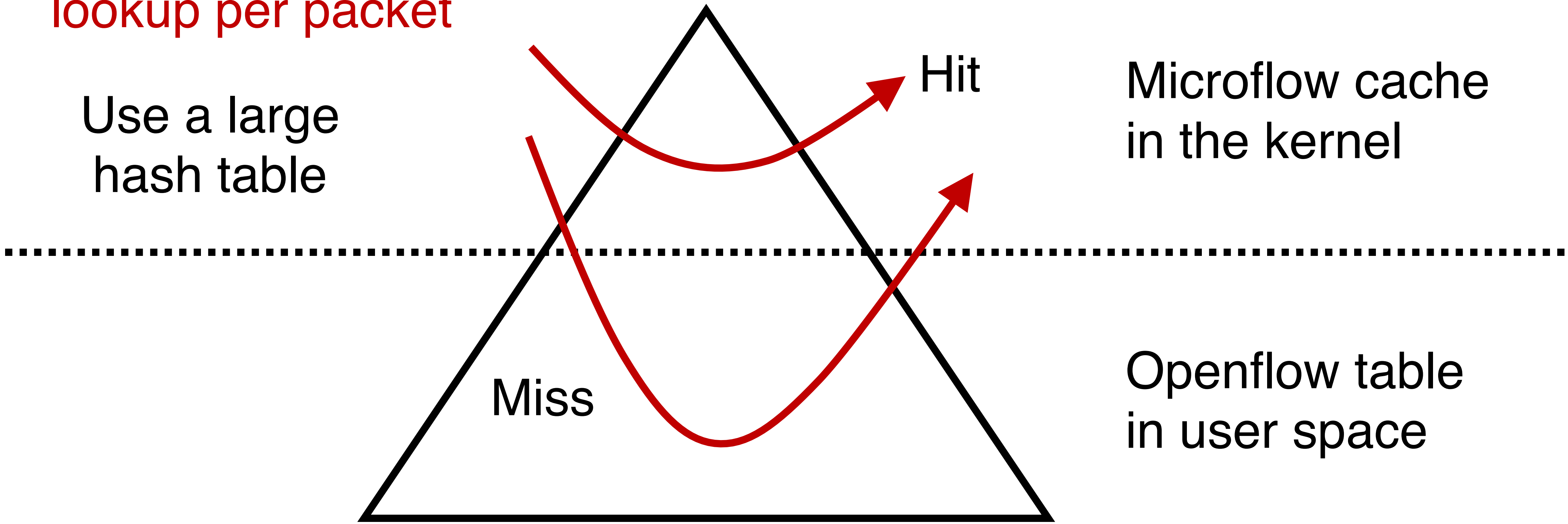
Large policies: Low performance with 100+ lookups per packet

Merging policies is problematic: cross-product explosion

Complex logic in kernel: rules with wildcards require complex algoriths

Idea 1: Microflow cache

- Microflow: complete set of packet headers with action
  - Example: srcIP, dstIP, IP TTL, srcMAC, dstMAC
- Same insight as tuple space search; attempt to do one memory lookup per packet



Problems with micro-flows

- Too many micro-flows: e.g., each TCP port
- Many micro-flows may be short lived
    - Poor cache-hit rate for memory lookup

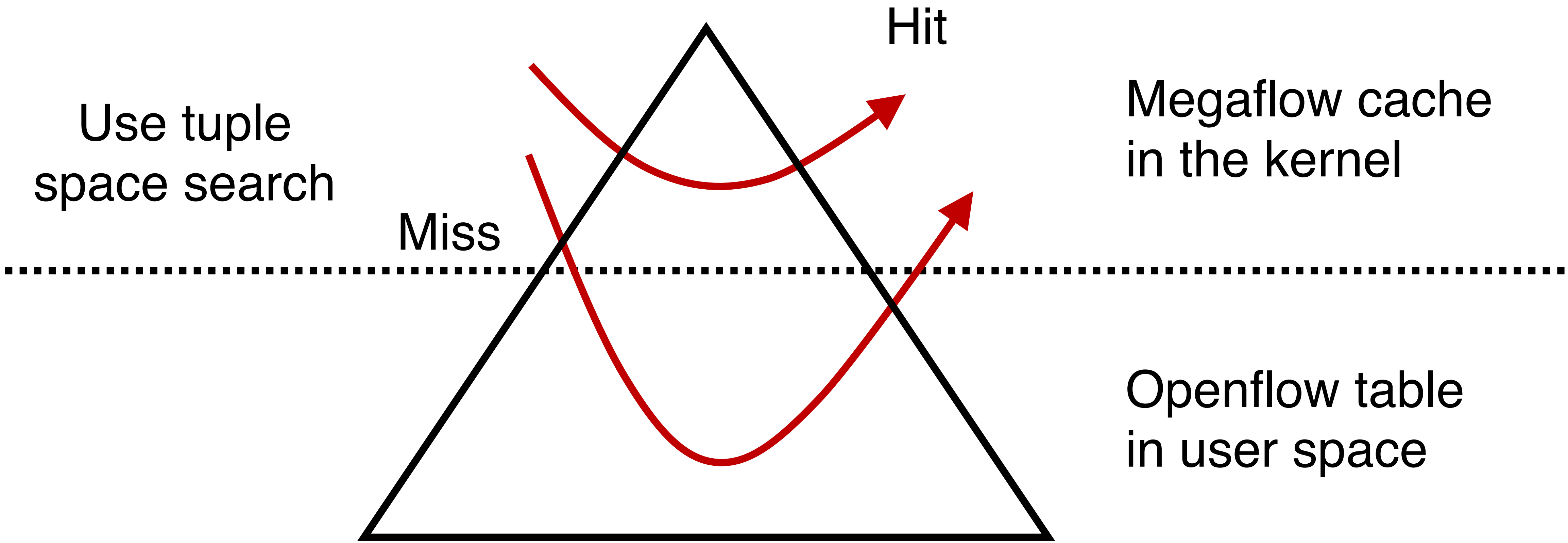
- Can we cache the outcome of rule lookup directly?

- Naive approach: Cross-product explosion!
    - Example: Table 1 on source IP, table 2 on destination IP

- Recurring theme: avoid up-front (proactive) costs

Idea 2: Mega-flow cache

- Build the cache of rules lazily using just the fields accessed
  - Ex: contain just src/dst IP combinations that appeared in packets



Outlook: fast packet processing

- Get rid of needless software if you can
- Specialization to app can bring significant benefits
  - IDS (hyperscan), caching in switches & load balancers
  - Algorithms can be as important as the frameworks
- Software changes
  - Application-kernel interface: application must be modified
  - Device drivers must often be modified
- Multitenancy: think about implications to weakening fault isolation
- Can we get isolation with efficiency?

Going beyond one (software) box

- Safe & efficient composition of middleboxes

- Share or shard state
- Failover and migration
- Placement and routing
- Scaling and compaction

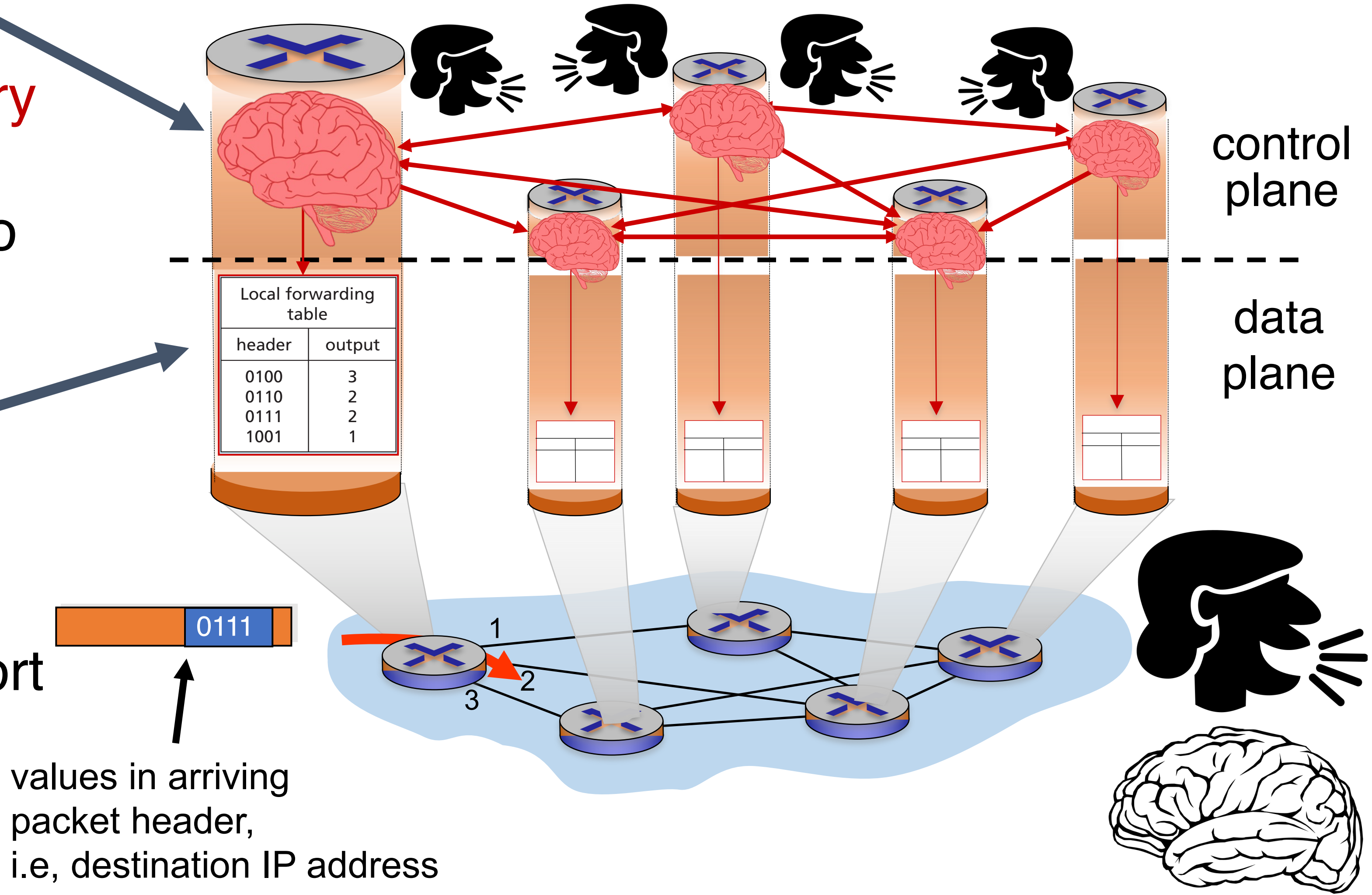
Distributed Control Planes

Acknowledgment: Jennifer Rexford

Per-router control plane

Distributed control plane:
Components in every router interact with other components to produce a routing outcome.

Data plane
per-packet processing, moving packet from input port to output port



Routing protocol

Q1. What info exchanged?

Q2. What computation?

Routing protocols enable FT computation

- What does the protocol compute?
  - Spanning tree, shortest path, local policy, arbitrary end-to-end paths

- What algorithm does the protocol run?
  - Information exchange + computation
  - Spanning-tree construction, distance vector, link-state routing, path-vector routing, source routing, end-to-end signaling

- How do routers learn end-host locations?
  - Learning/flooding, injecting into the routing protocol, dissemination using a different protocol, and directory server

Goals of Routing Protocols #1

- Determine good paths from source to destination

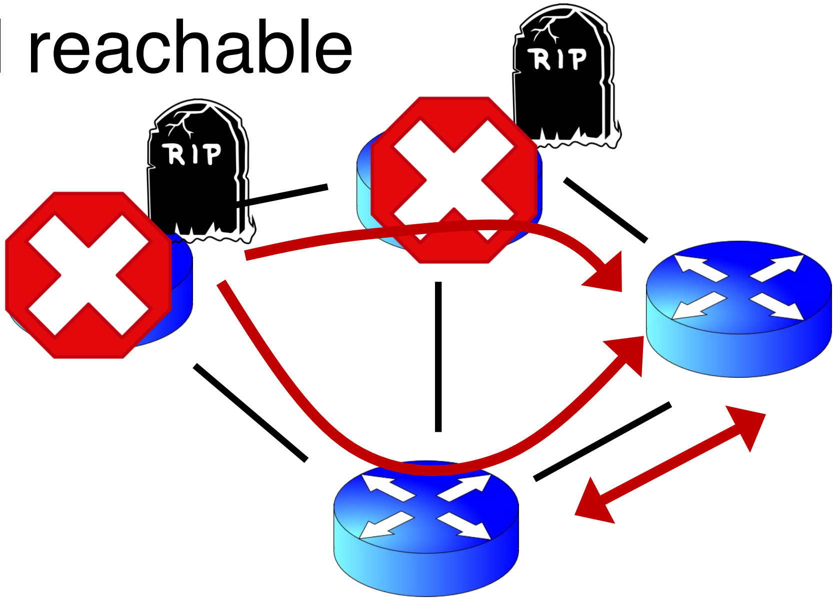
- “Good” = least cost
  - Least propagation delay
  - Least cost per unit bandwidth (e.g., $ per Gbit/s)
  - Least congested (workload-driven)

- “Good” = policy compliant

- “Path” = a sequence of router ports (links)

Goals of Routing Protocols #2

- Make networks resilient to failures
- Routers & links can fail without taking down the entire network
- Entire subsets can be unreachable; rest still reachable
- Hence, the protocol must be distributed



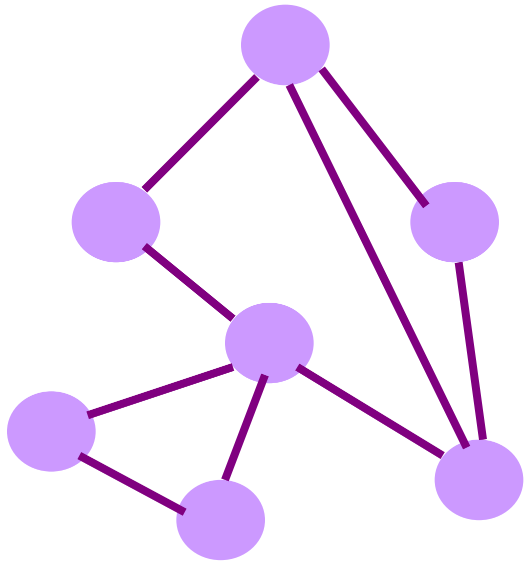
What does the protocol compute?

(the outcome, not the computation)

Different ways to represent paths

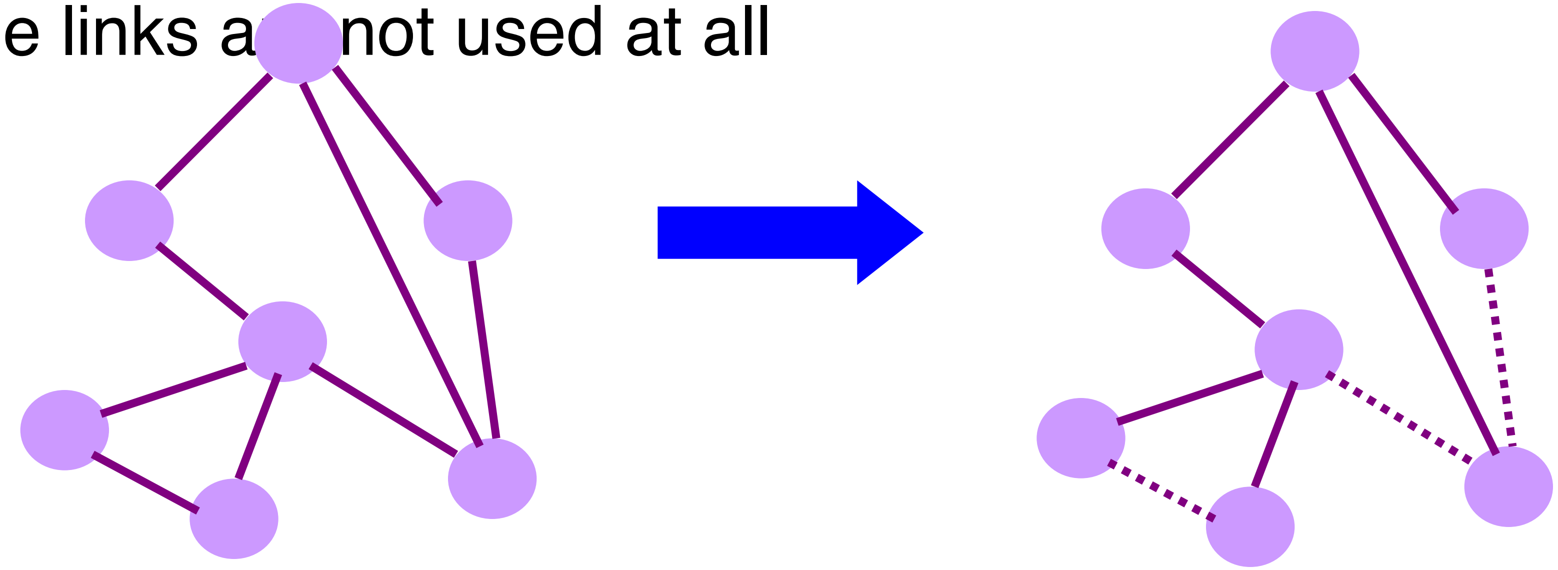
- Trade-offs
  - State required to represent the paths
  - Efficiency of the resulting paths
  - Ability to support multiple paths
  - Complexity of computing the paths
  - Which nodes are “in charge”

- Applied in different settings
  - LAN, intra-domain, inter-domain



Spanning tree (Ethernet)

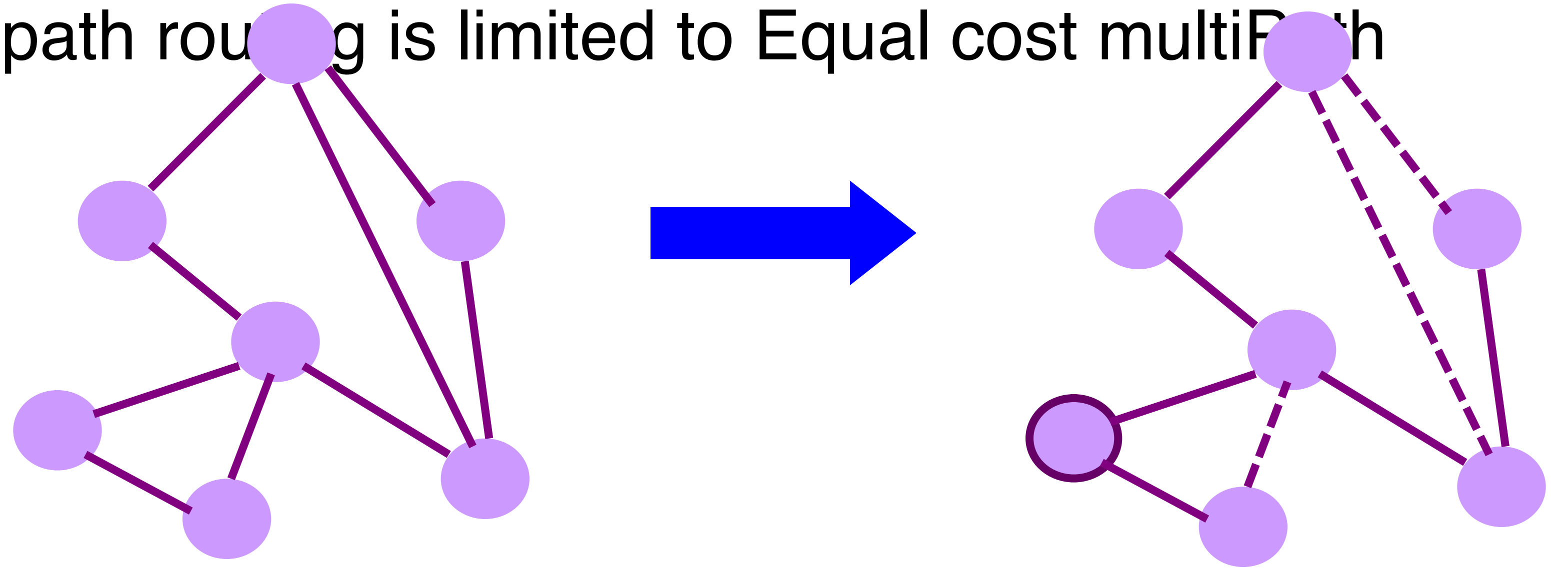
- One tree that reaches every node
  - Single path between each pair of nodes
  - No loops, so can support broadcast easily
- Disadvantages
  - Paths are sometimes long
  - Some links a… not used at all



Shortest paths (OSPF/IS-IS)

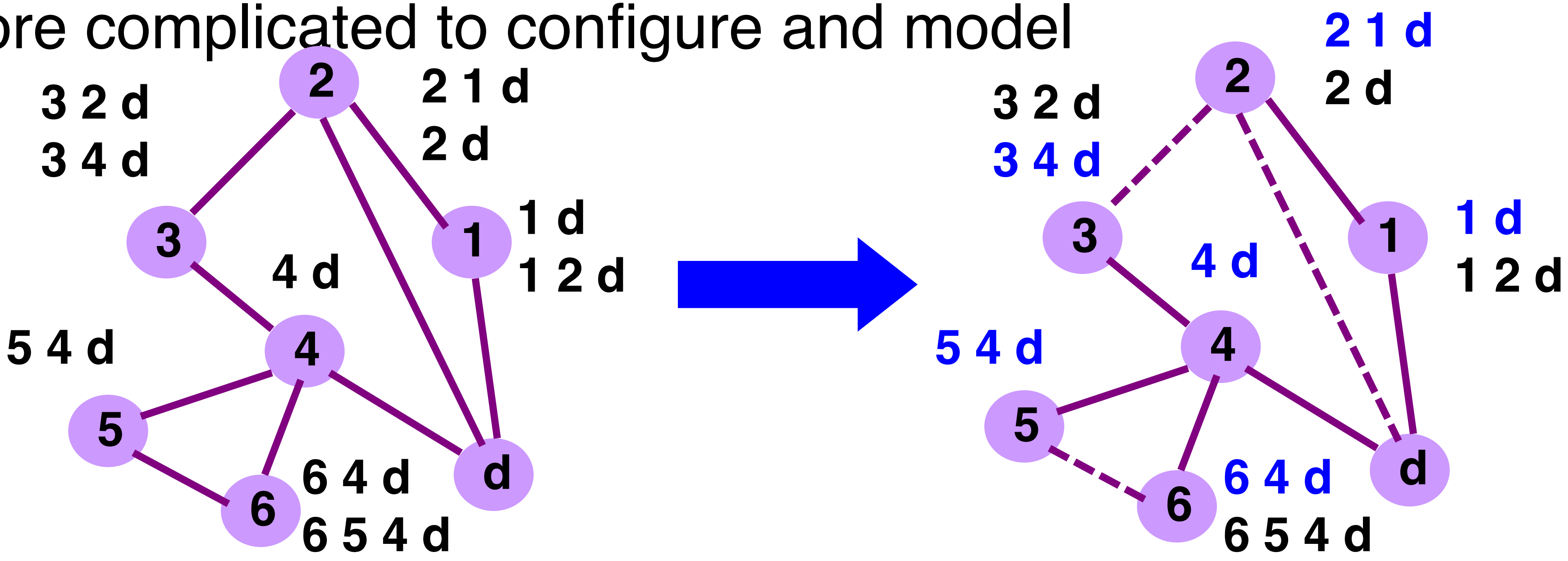
Set by network administrator

- Shortest path(s) between each pair of nodes
  - Separate shortest-path tree rooted at each node
  - Minimum hop count or minimum sum of edge weights
- Disadvantages
  - All nodes need to agree on the link metrics
  - Multipath routing is limited to Equal cost multiPath



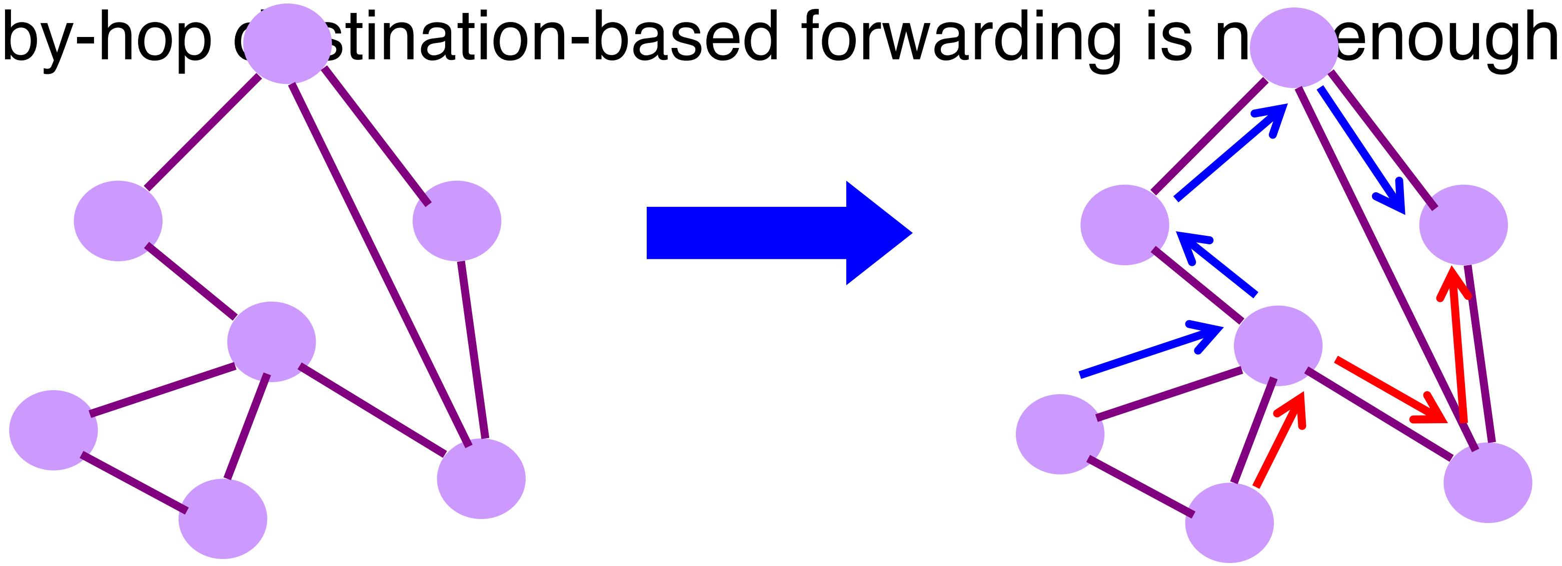
Local policy at each hop (BGP)

- Locally best path
  - Local policy: each node picks the path it likes best
  - … among the paths chosen by its neighbors
- Disadvantages
  - More complicated to configure and model



End-to-end path selection (IP src route)

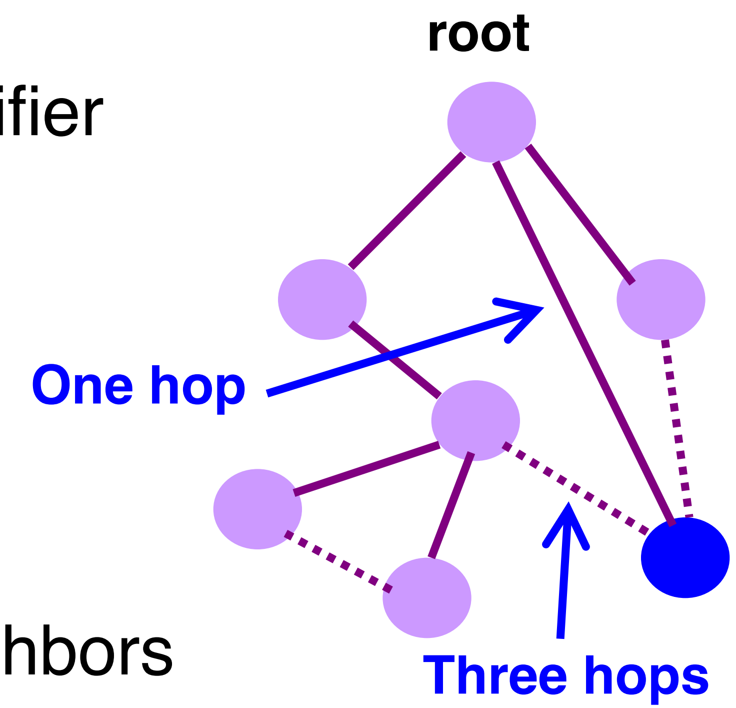
- End-to-end path selection
  - Each node picks its own end to end paths
  - … independent of what other paths other nodes use
- Disadvantages
  - More state and complexity in the nodes
  - Hop-by-hop destination-based forwarding is not enough



How to compute paths?

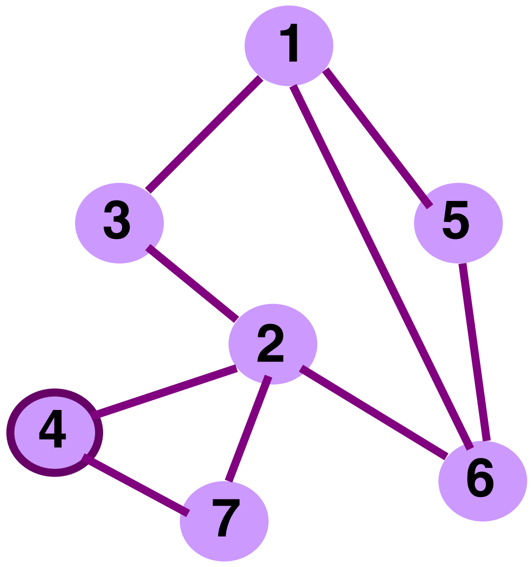
Spanning tree algorithm (Ethernet)

- Elect a root
  - The switch with the smallest identifier
  - And form a tree from there
- Algorithm
  - Repeatedly talk to neighbors
    - "I think node Y is the root"
    - "My distance from Y is d"
  - Update information based on neighbors
    - Smaller id as the root
    - Smaller distance d+1
  - Don't use interfaces not in the path



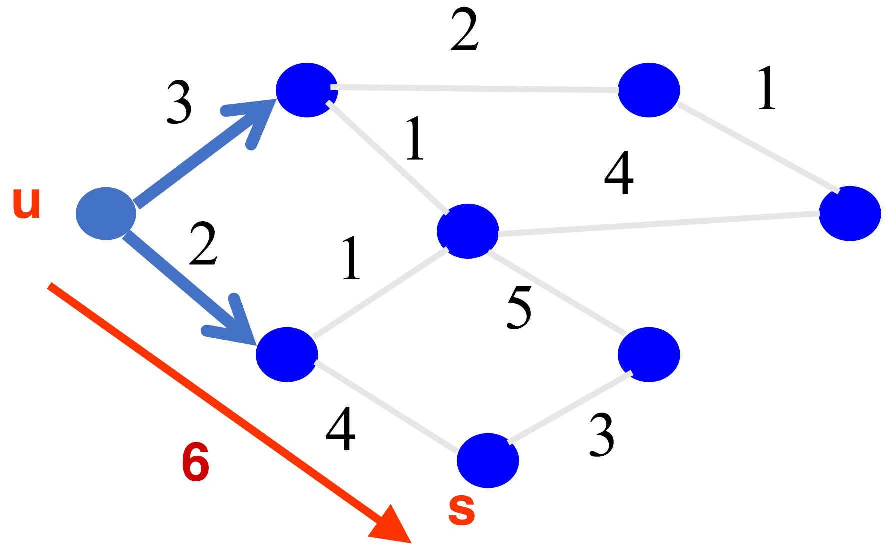
Spanning tree example: switch #4

- Switch #4 thinks it is the root
  - Sends (4, 0) message to 2 and 7
- Switch #4 hears from #2
  - Receives (2, 0) message from 2
  - … and thinks that #2 is the root
  - And realizes it is just one hop away
- Switch #4 hears from #7
  - Receives (2, 1) from 7
  - And realizes this is a longer path
  - So, prefers its own one-hop path
  - And removes 4-7 link from the tree



Shortest-path problem

- Compute: *path costs* to all nodes
  - From a given source u to all other nodes
  - Cost of the path through each outgoing link
  - Next hop along the least-cost path to s

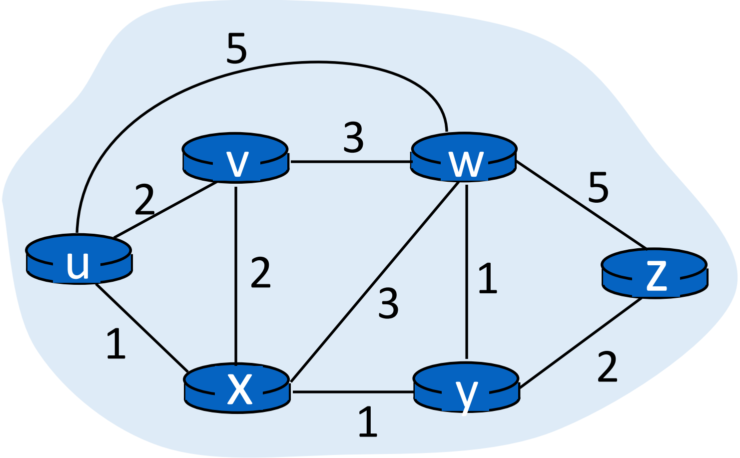


The graph abstraction

- Routing algorithms work over an abstract representation of a network: the graph abstraction

Ex: Rutgers campus

u: Computer Science
v: School of Engineering
…



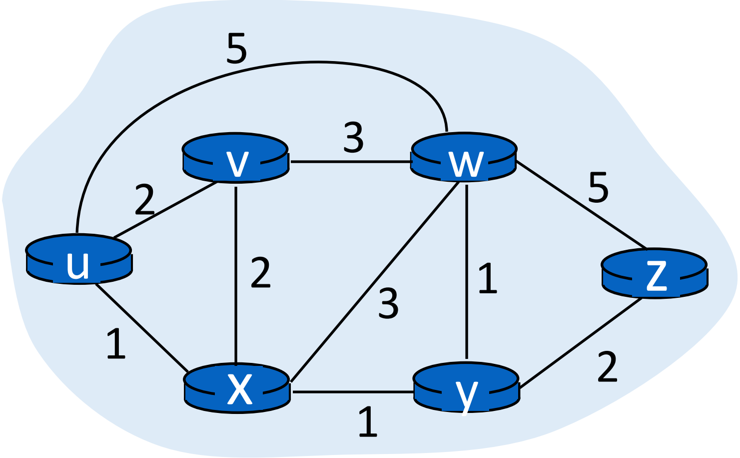
- Each router is a node in a graph
- Each link is an edge in the graph
- Edges have weights (also called link metrics). Set by netadmin

The graph abstraction

- Routing algorithms work over an abstract representation of a network: the graph abstraction

Ex: Rutgers campus

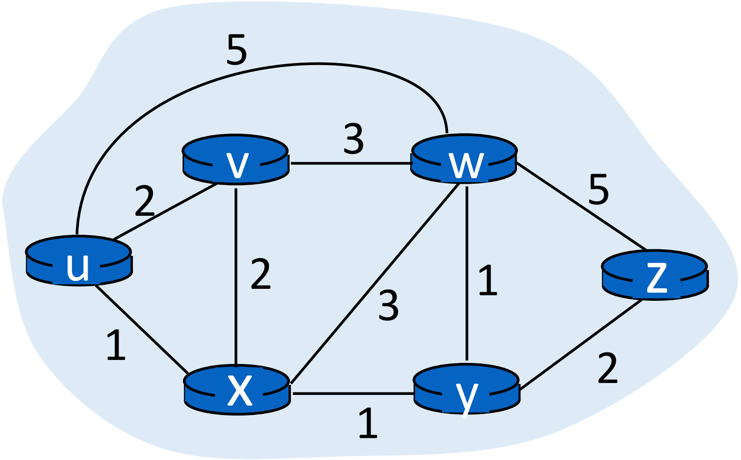
u: Computer Science
v: School of Engineering
…



- G = (N, E)
- N = {u, v, w, x, y, z}
- E = { (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

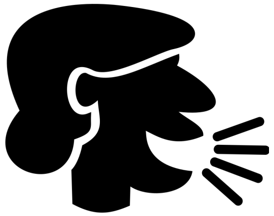
The graph abstraction

- Cost of an edge: c(x, y)
  - Examples: c(u, v) = 2, c(u, w) = 5
- Cost of a path = sum of edge costs
  - c(path x → w → y → z) = 3 + 1 + 2 = 6

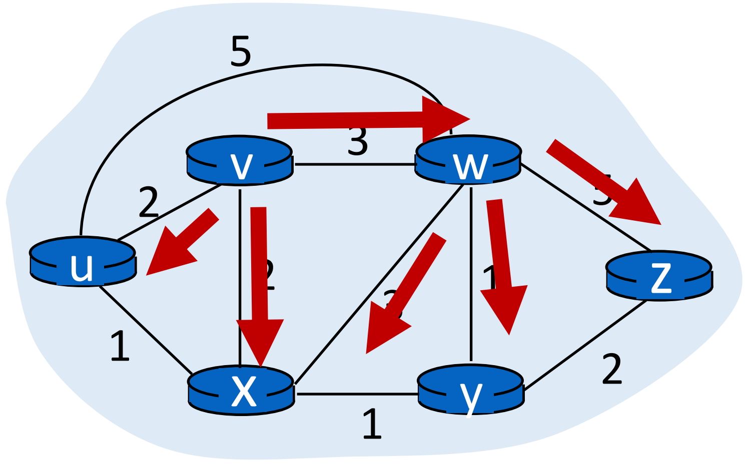


- Outcome of routing: each node should determine the least cost path to every other node
- Q1: What information should nodes exchange with each other to enable this computation?
- Q2: What algorithm should each node run to compute the least cost path to every node?

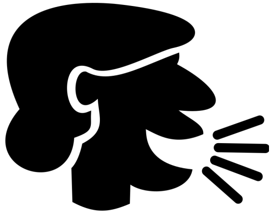
Q1: Information exchange



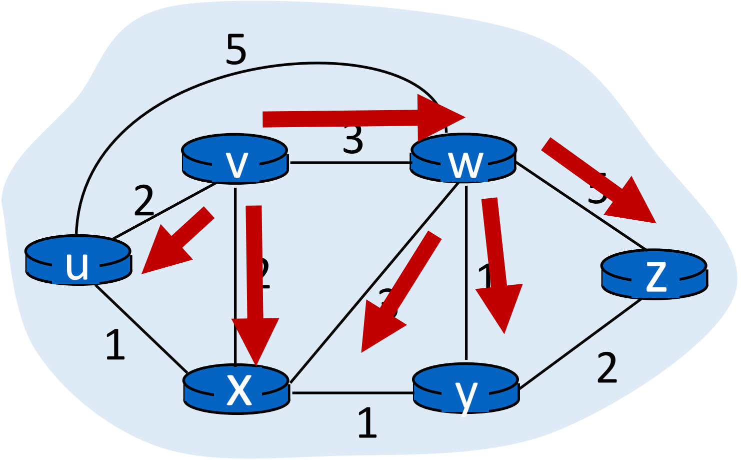
- Link state flooding: the process by which neighborhood information of each network router is transmitted to all other routers
- Each router sends a link state advertisement (LSA) to each of its neighbors
- LSA contains the router ID, the IP prefix owned by the router, the router's neighbors, and link cost to those neighbors
- Upon receiving an LSA, a router forwards it to each of its neighbors: flooding



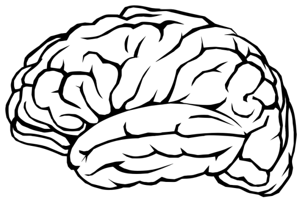
Q1: Information exchange



- Eventually, the entire network receives LSAs originated by each router
- LSAs put into a link state database
- LSAs occur periodically and whenever the graph changes
  - Example: if a link fails
  - Example: if a new link or router is added
- The routing algorithm running at each router can use the entire network's graph to compute least cost paths



Q2: The algorithm



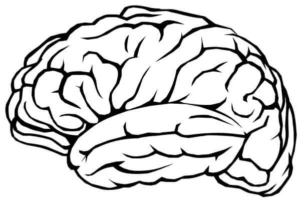
Dijkstra’s algorithm

- Given a network graph, the algorithm computes the least cost paths from one node (source) to all other nodes
- This can then be used to compute the forwarding table at that node
- Iterative algorithm: maintain estimates of least costs to reach every other node. After k iterations, each node definitively knows the least cost path to k destinations

Notation:

- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current estimate of cost of path from source to destination v
- p(v): (predecessor node) the last node before v on the path from source to v
- N': set of nodes whose least cost path is definitively known

Dijsktra’s Algorithm



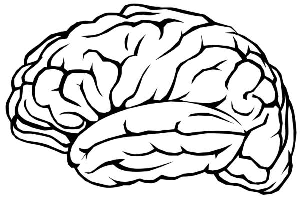
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
   find w not in N' such that D(w) is a minimum
   add w to N'
   update D(v) for all v adjacent to w and not in N' :
       D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
    shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Initial estimates of distances are just the link costs of neighbors.

Least cost node among all estimates. This cost cannot decrease further.

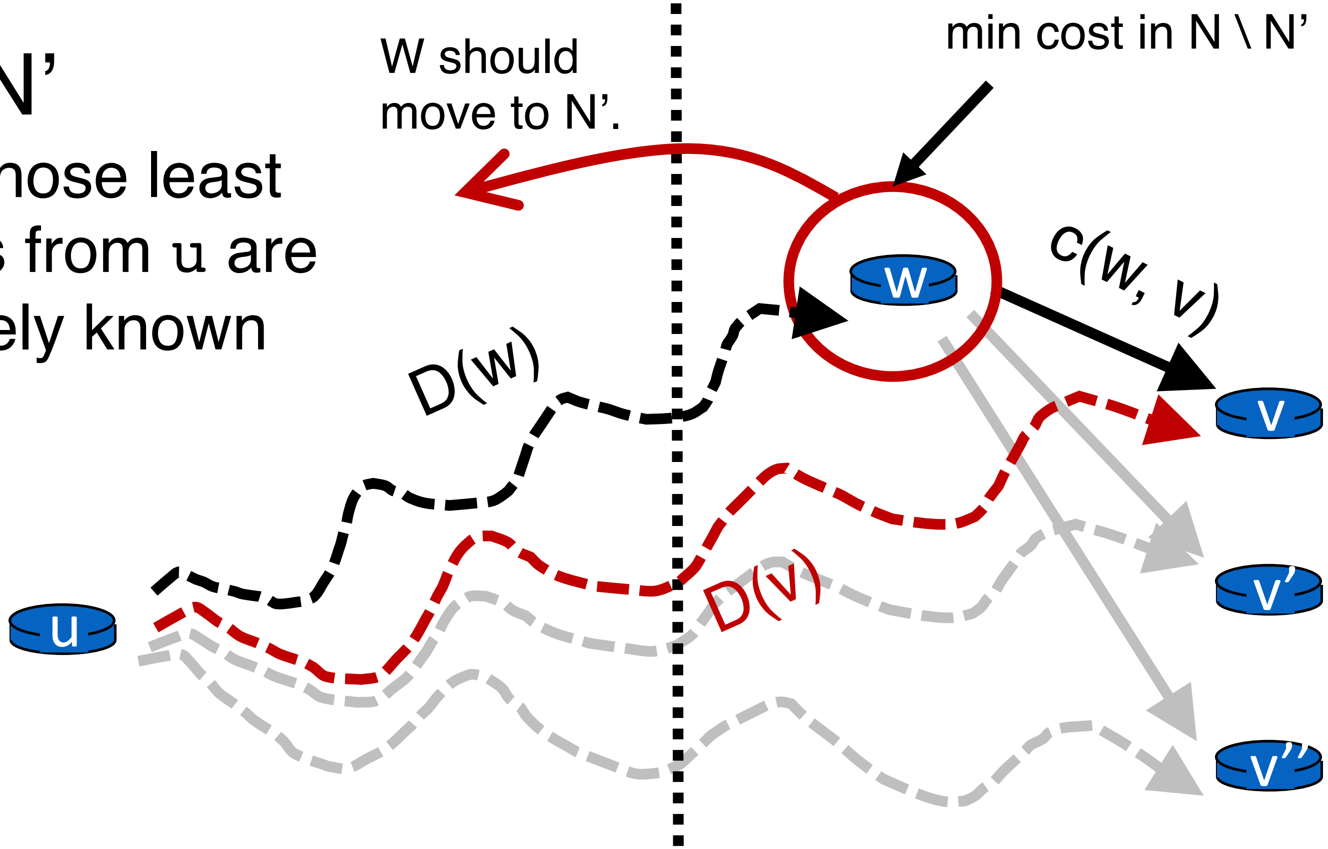
Relaxation

Visualization



N'

nodes whose least cost paths from u are definitively known



N \ N'

Nodes with estimated least path costs, not definitively known to be smallest possible

Relaxation: for each v in N \ N', is the cost of the path via w smaller than known least cost path to v?

If so, update D(v)

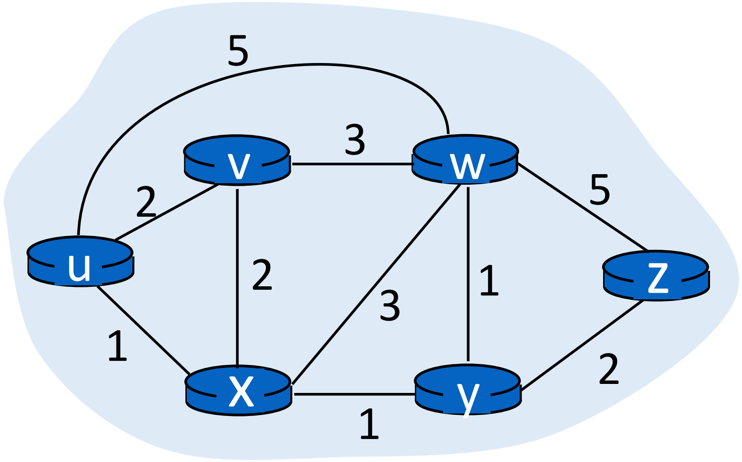
Predecessor of v is w.

Cost of path via w: D(w) + c(w,v)

Cost of known best path: D(v)

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	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



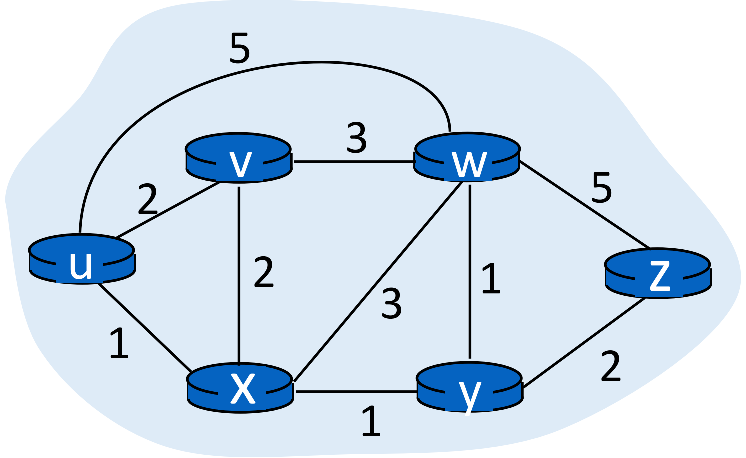
Constructing the forwarding table

- To find the router port to use for a given destination (router), find the predecessor of the node iteratively until reaching an immediate neighbor of the source `u`

- The port connecting `u` to this neighbor is the output port for this destination

Constructing the forwarding table

- Suppose we want forwarding entry for z.



D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
2,u	3,y	1,u	2,x	4,y

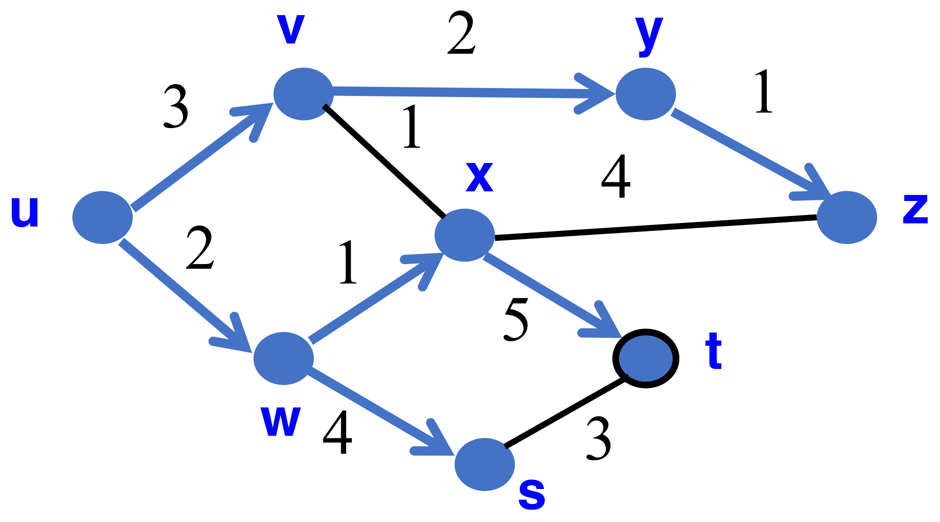
z: p(z) = y
y: p(y) = x
x: p(x) = u
x is an immediate neighbor of u

Forwarding table at u:

destination	link
z	(u,x)

Link-state: Shortest-path tree

- Shortest-path tree from u
- Forwarding table at u



	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Counter-intuitive: Operators may set the link metric to achieve certain shortest-path trees with the protocol

Path-vector routing (BGP)

- Key idea: advertise the entire path
- Distance vector: send *distance metric* per dest d
- Path vector: send the *entire path* for each dest d