Network Layer (3)

Networked Systems Architecture 3
Lecture 11

UNIVERSITY
of
GLASGOW

Lecture Outline

- Routing concepts
- Unicast routing
  - Intradomain: distance vector and link state
  - Interdomain: path vector protocols and BGP

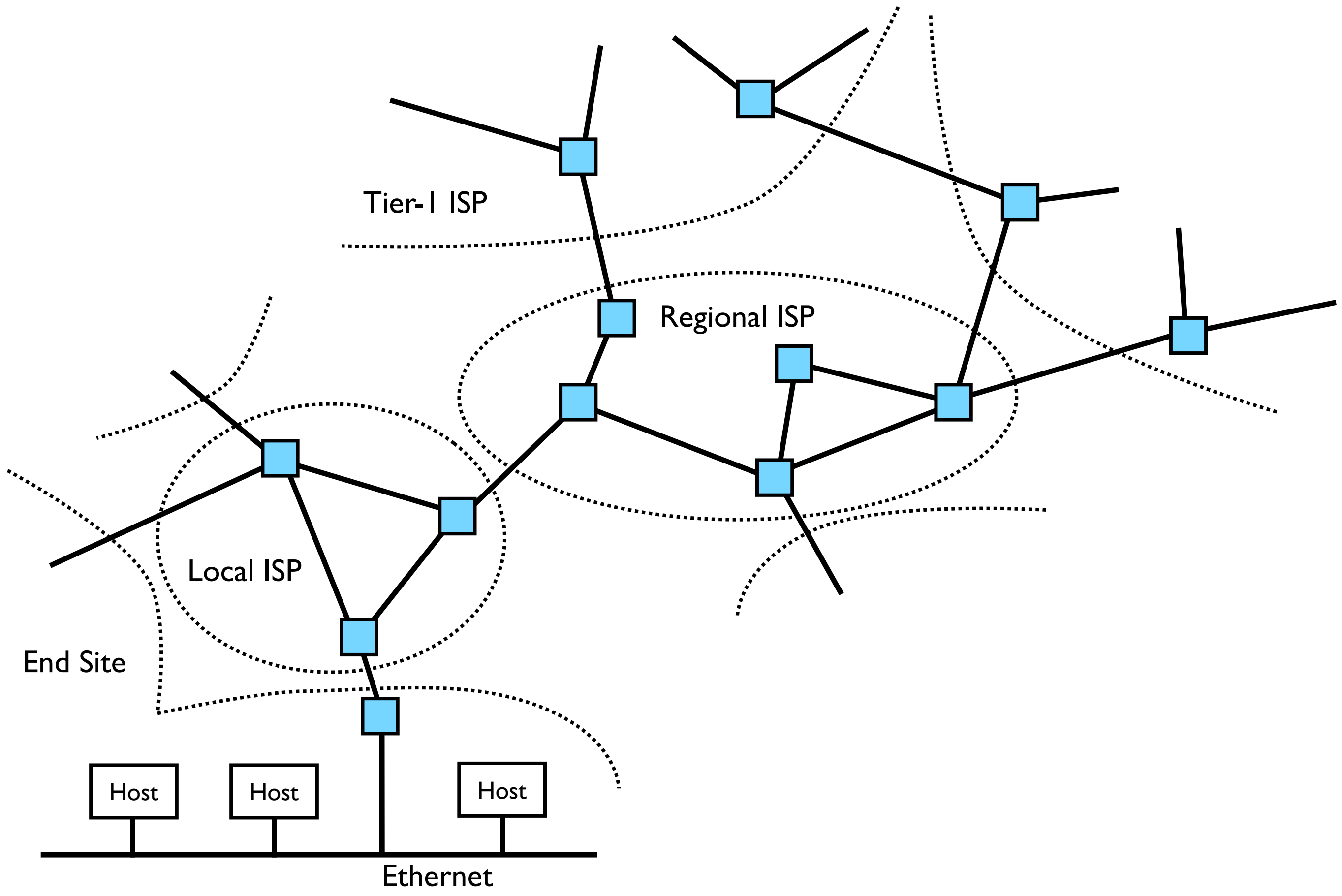
Routing

- Network layer responsible for *routing* data from source to destination across multiple hops
    - Nodes learn (a subset of) the *network topology* and run a *routing algorithm* to decide where to forward packets destined for other hosts
        - End hosts usually have a simple view of the topology ("my local network" and "everything else") and a simple routing algorithm ("if it's not on my local network, send it to the default gateway")
        - Gateway devices ("routers") exchange topology information, decide best route to destination based on knowledge of the entire network topology

Unicast Routing

- Routing algorithms to deliver packets from a source to a single destination
- Choice of algorithm affected by usage scenario
  - Intradomain routing
  - Interdomain routing
  - Politics and economics

Intradomain Unicast Routing



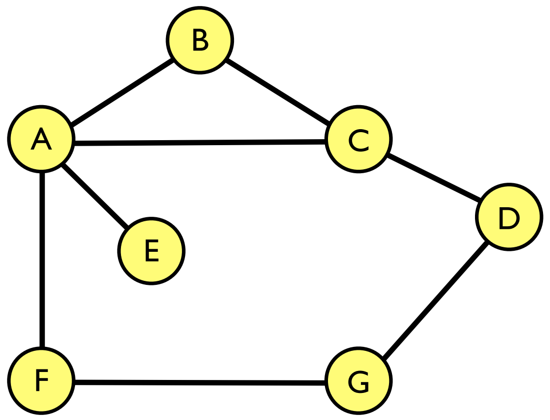
Intradomain Unicast Routing

- Routing within an AS
  - Single trust domain
    - No policy restrictions on who can determine network topology
    - No policy restrictions on which links can be used
  - Desire efficient routing → shortest path
    - Make best use of the network you have available
  - Two approaches
    - Distance vector – the Routing Information Protocol (RIP)
    - Link state – Open Shortest Path First routing (OSPF)

Distance Vector Routing

- Each node maintains a vector containing the distance to every other node in the network
  - Periodically exchanged with neighbours, so eventually each node knows the distance to all other nodes
    - The routing table “converges” on a steady state
  - Links which are down or unknown have distance = ∞
- Forward packets along route with least distance to destination

Distance Vector: Example



Time: 0

Nodes only know their immediate neighbours

Routing Table at Node A

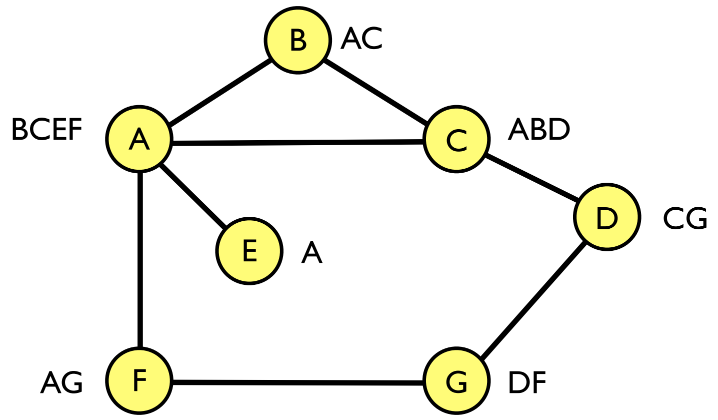
Destination	Cost	Next Hop
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

Distance to Reach Node

Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Distance Vector: Example



Time: 1

Nodes also know neighbours of their neighbours – routing data has spread one hop

Routing Table at Node A

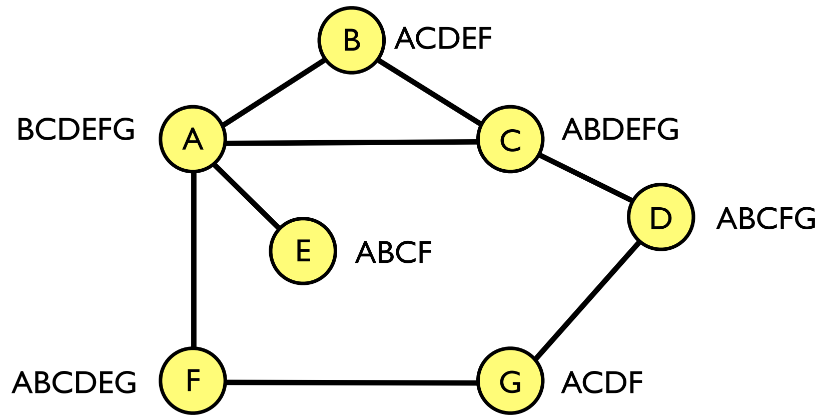
Destination	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Distance to Reach Node

Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	∞
C	1	1	0	1	2	2	2
D	2	2	1	0	∞	2	1
E	1	2	2	∞	0	2	∞
F	1	2	2	2	2	0	1
G	2	∞	2	1	∞	1	0

Distance Vector: Example



Time: 2

Routing data has spread two hops

Routing Table at Node A

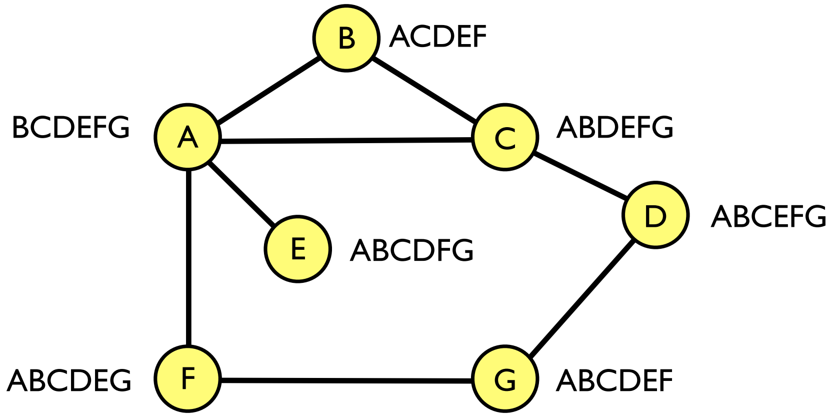
Destination	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Distance to Reach Node

Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Distance Vector: Example



Time: 3

Routing table is complete – nodes continue to exchange distance metrics in case the topology changes

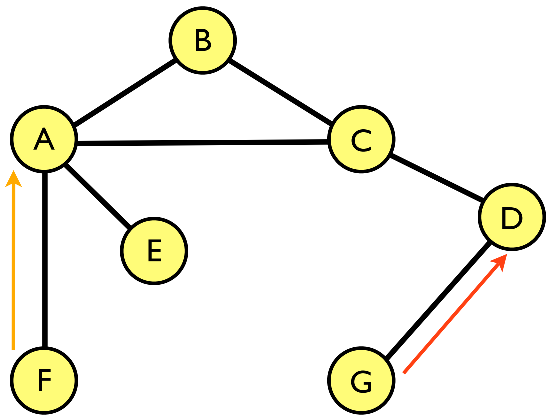
Routing Table at Node A

Destination	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Distance to Reach Node / Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Distance Vector: Example



Time: 4

Link between F and G fails
F and G notice, set the link distance to ∞, and pass an update to A and D

Routing Table at Node A

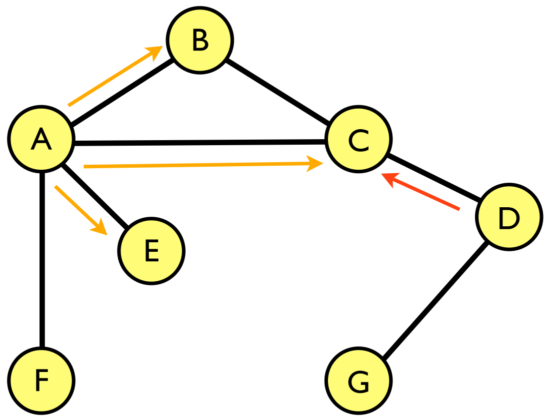
Destination	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Distance to Reach Node

Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	∞
G	2	3	2	1	3	∞	0

Distance Vector: Example



Time: 5

A sets its distance to G to ∞
D sets its distance to F to ∞
Both pass on news of the link failure

Routing Table at Node A

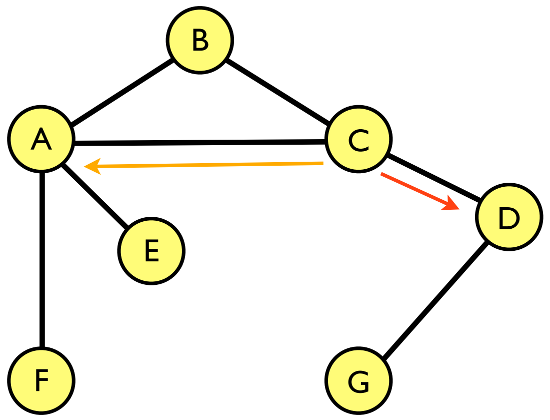
Destination	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Distance to Reach Node

Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	2	1	1	∞
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	∞	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	∞
G	2	3	2	1	3	∞	0

Distance Vector: Example



Time: 6

C knows it can reach F and G in 2 hops via alternate paths, so advertises shorter routes; network begins to converge

Routing Table at Node A

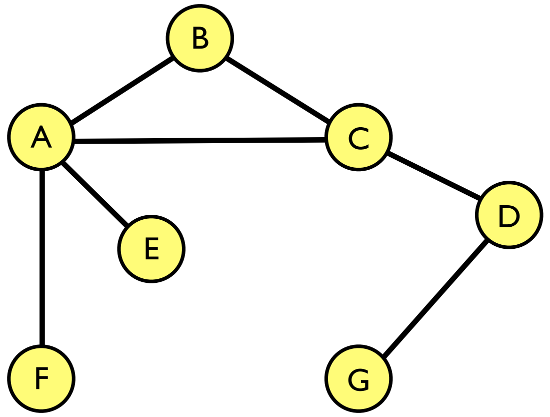
Destination	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Distance to Reach Node

Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	2	1	1	3
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	3	1
E	1	2	2	3	0	2	∞
F	1	2	2	2	2	0	∞
G	2	3	2	1	3	∞	0

Distance Vector: Example



Time: 7

Eventually, the network is stable in a new topology

Routing Table at Node A

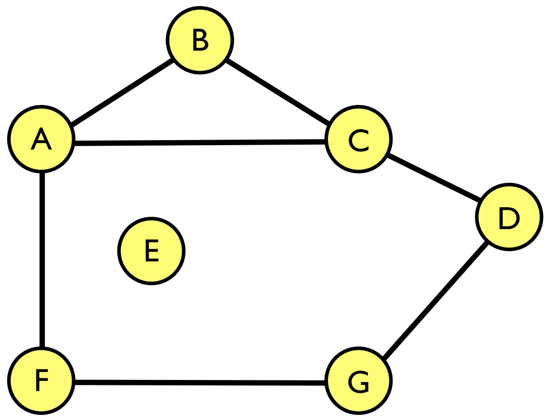
Destination	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Distance to Reach Node

Information Stored at Node

	A	B	C	D	E	F	G
A	0	1	1	2	1	1	3
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	3	1
E	1	2	2	3	0	2	4
F	1	2	2	2	2	0	4
G	2	3	2	1	3	4	0

Count to Infinity Problem



What if A-E link fails?

A advertises distance ∞ to E at the same time as C advertises a distance 2 to E (the old route via A).

B receives both, concludes that E can be reached in 3 hops via C, and advertises this to A. C sets its distance to E to ∞ and advertises this.

A receives the advertisement from B, decides it can reach E in 4 hops via B, and advertises this to C.

C receives the advertisement from A, decides it can reach E in 5 hops via A…

Loops, eventually counting up to infinity...

Solution 1: How big is infinity?

- Simple solution: `#define ∞ 16`
- Bounds time it takes to count to infinity, and hence duration of the disruption
- Provided the network is never more than 16 hops across!

Solution 2: Split Horizon

- When sending a routing update, do not send route learned from a neighbour back to that neighbour
    - Prevents loops involved two nodes, doesn’t prevent three node loops (like the previous example)
    - No general solution exists – distance vector routing always suffers slow convergence due to the count to infinity problem

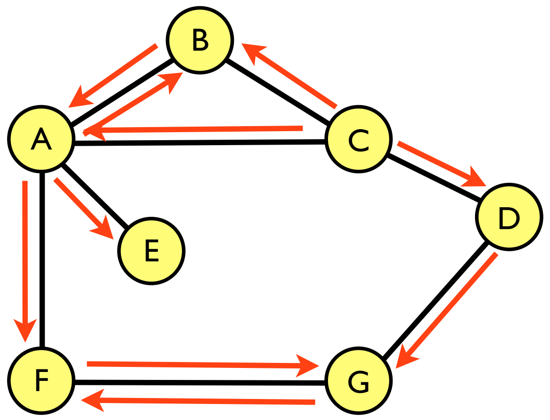
Link State Routing

- Nodes know the links to their neighbours, and the cost of using those links
    - The *link state* information
- Reliably flood this information, giving all nodes complete map of the network
- Each node then directly calculates shortest path to every other node, uses this as routing table

Link State Information

- Link state information updates are flooded on start-up, and when the topology changes
- Each update contains:
  - Name of node that sent the update
  - List of directly connected neighbours of that node, with the cost of the link to each
  - A sequence number

Flooding Link State Updates



Node C sends an update to each of its neighbours

Each receiver compares the sequence number with that of the last update from C, if greater it forwards the update on all links except the link on which it was received.

Each receiver compares the sequence number with that of the last update from C, if greater it forwards the update on all links except the link on which it was received.

Eventually, the entire network has received the update

Calculate Shortest Paths

- Flooding link state data from all nodes ensures all nodes know the entire topology
- Each node uses *Dijkstra's shortest-path algorithm* to calculate optimal route to every other node
  - Forward packets based on shortest path
  - Recalculate shortest paths on every routing update

Shortest Path Algorithm

Definitions:

N	set of all nodes in the graph
$l(i, j)$	weight of link from *i* to *j* (∞ if no link, 0 if *i* = *j*)
s	source node from which we're calculating shortest paths

Dijkstra's Algorithm for an undirected connected graph:

```
M = {s}                              The set of nodes that have been checked
foreach n in N - {s}:                The distance to directly connected neighbouring nodes
    C(n) = l(s, n)

while (N ≠ M):
    c = ∞
    foreach n in (N-M)
        if C(w) < c then w = n       Find node w such that C(w) is the minimum for all nodes in (N-M)

    M += {w}                         Add one node at a time, starting with the closest
    foreach n in (N-M):
        if C(n) > C(w) + l(w, n) then C(n) = C(w) + l(w, n)      Best route to n is via w
```

Result:

$C(x)$ cost of the shortest path from *s* to *x*

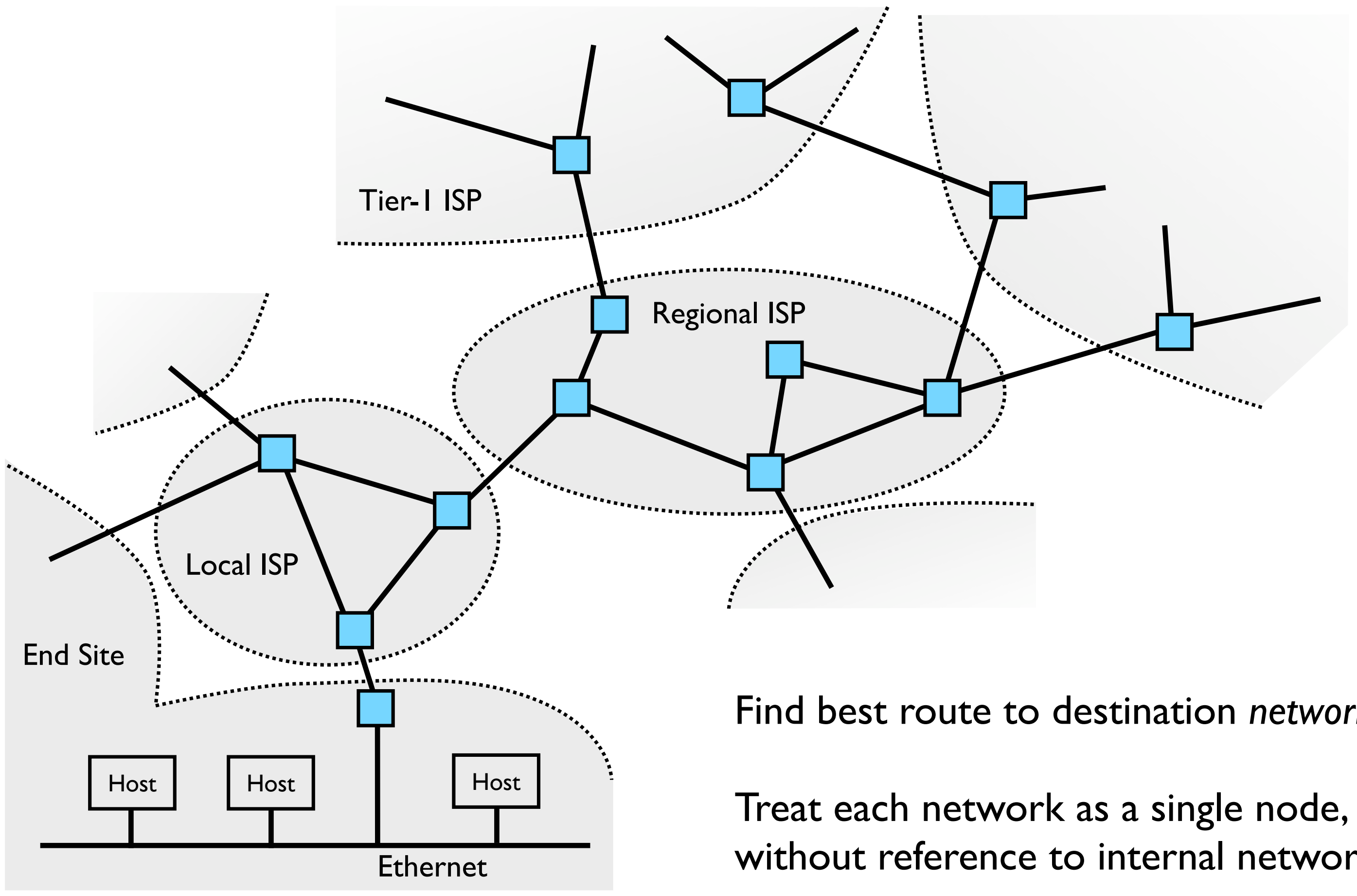
Distance Vector *vs.* Link State

- Distance vector routing:
  - Simple to implement
  - Doesn't require routers to store much information
  - Suffers from slow convergence

- Link State routing:
  - More complex
  - Requires each router to store a complete network map
  - Much faster convergence

Slow convergence times make distance vector routing unsuitable for large networks

Interdomain Unicast Routing



Find best route to destination *network*

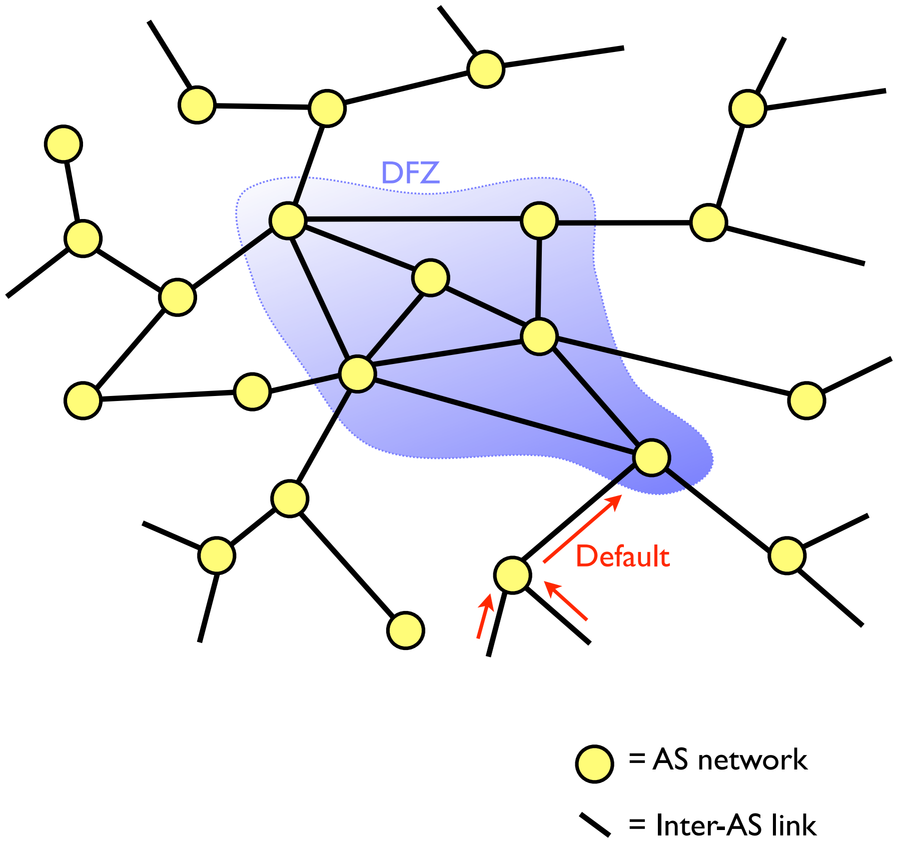
Treat each network as a single node, and route without reference to internal network topology

Interdomain Unicast Routing

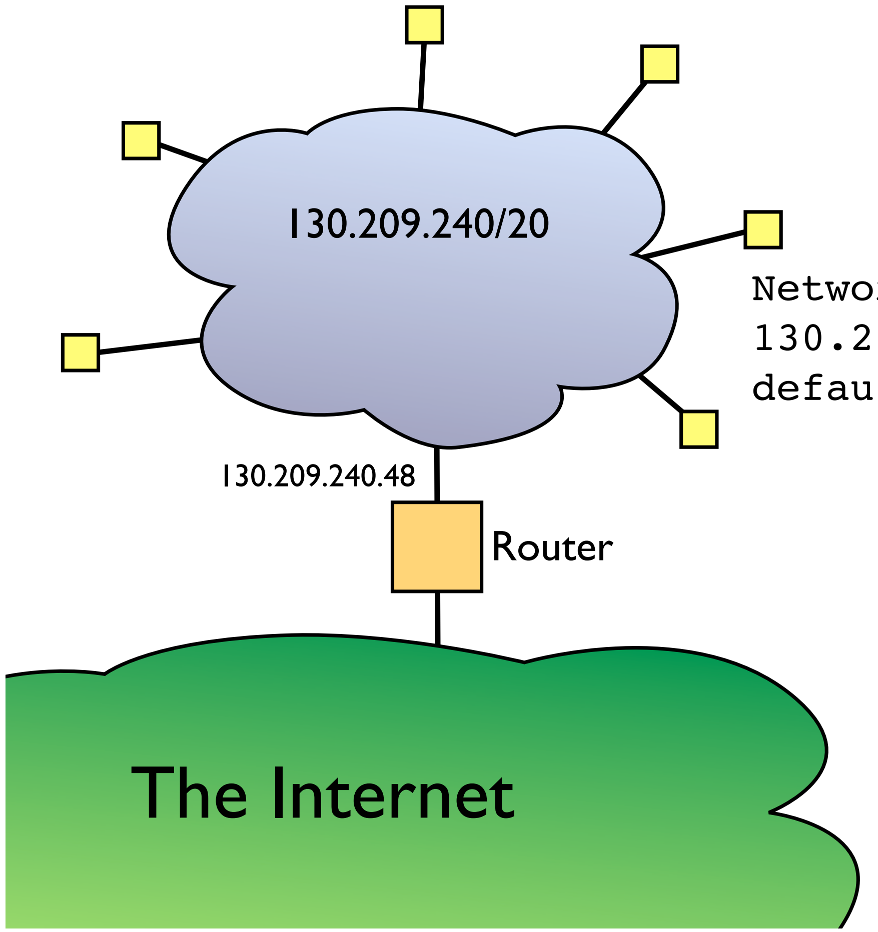
- Routing between ASes
  - You don’t trust your neighbours
    - Policy restrictions on who can determine your topology
    - Policy or economic restrictions on which route data can follow
  - Prefer control over routing, even if that means data doesn’t necessarily follow the best (shortest) path
    - Because the shortest path might pass through a competitor’s network, or a country you politically disagree with, or over an expensive link

Default Routes and the DFZ

- The AS-level topology:
  - Well connected core networks
  - Sparsely connected edges, getting service from the core networks
- Edge networks can use a *default route* to the core
- Core networks need a full routing table
  - The *default free zone* (DFZ)



Example: Local DCS Network



```
Network:      Netmask:      Gateway:
130.209.240.0 255.255.240.0 eth0
default       0.0.0.0       130.209.240.48
```

Routing in the DFZ

- Core networks are well-connected, must know about every other network
  - The *default free zone* where there is no default route
  - Wish to route based on policy, not shortest path
    - Use AS *x* in preference to AS *y*
    - Use AS *x* only to reach addresses in this range
    - Use the path that crosses the fewest number of ASes
    - Avoid ASes located in that country
  - Requires complete AS-level topology information

Border Gateway Protocol

- The Internet uses the Border Gateway Protocol (BGP v4) for interdomain routing
  - Each AS advertises the complete path it would use to reach every AS for which it wishes to provide routing
    - AS level path, not network layer path
    - Likely a subset of the ASes for which it could provide routing, due to policy
  - Apply local policy rules to learned topology, calculate feasible paths, use for routing
    - E.g. remove possible paths that conflict with local policy, adjust path length to match actual price of transit, run shortest path algorithm on result

Border Gateway Protocol

- Note:
  - BGP doesn't always find a route, even if one exists, since it might be prohibited by policy
  - Routes used are often not shortest AS path
  - Mapping business goals and policies to BGP policies is a poorly documented process, with much deep magic

Questions?