University of Glasgow | School of Computing Science

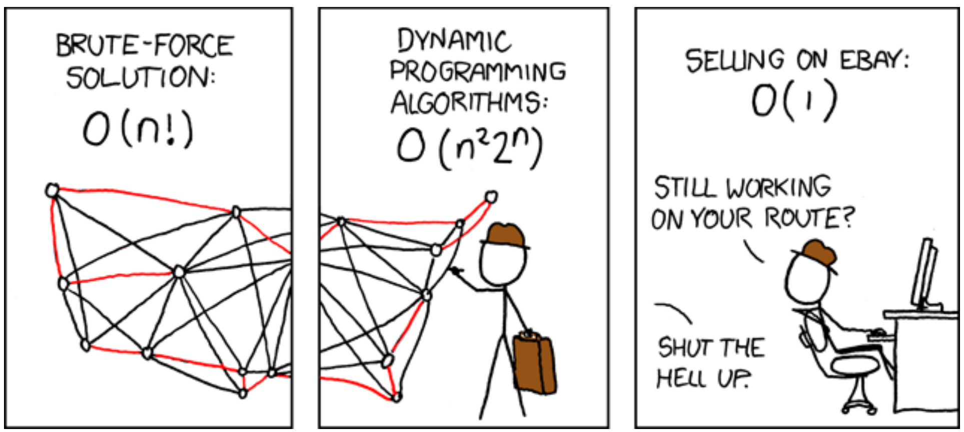
Networks & Operating Systems Essentials

Dr Angelos Marnerides

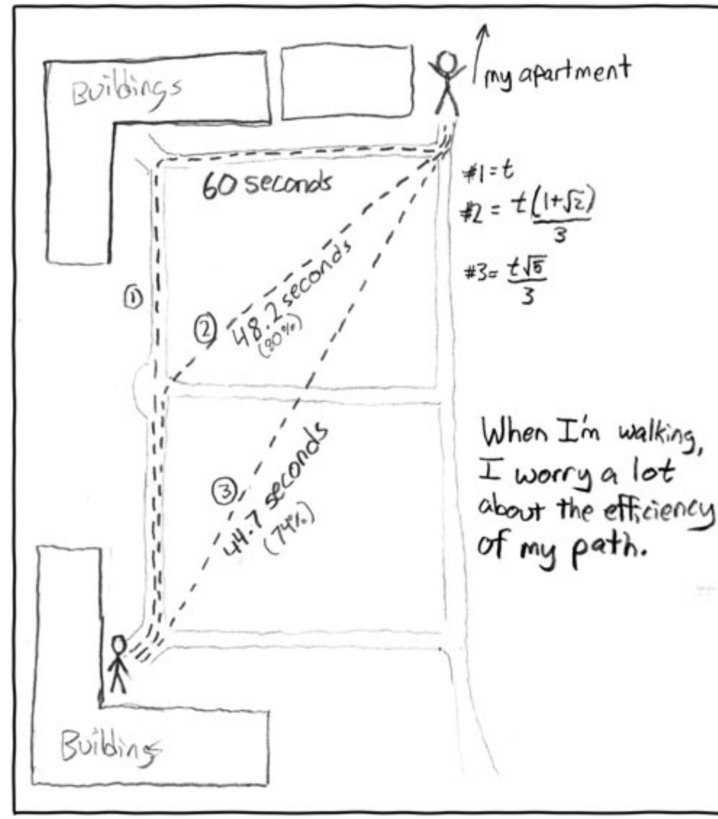
<*angelos.marnerides@glasgow.ac.uk*>

School of Computing Science, Room: S122

Today, on NOSE2…



Source: https://xkcd.com/399/



Source: https://xkcd.com/85/

Based on slides © 2017 Colin Perkins

ROUTING

Routing

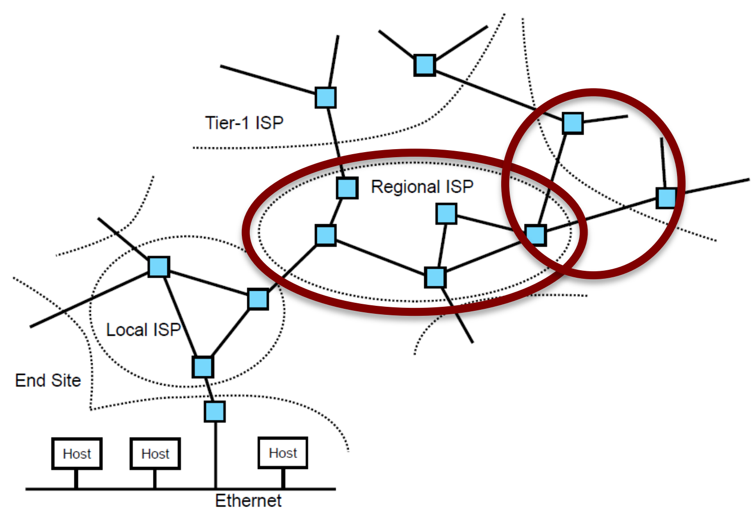
- Network layer responsible for routing data from source to destination across multiple hops
  - Nodes learn (a subset of) the network topology and run a routing algorithm to decide where to forward packets destined for other hosts
    - End hosts usually have a simple view of the topology ("my local 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
  - Intra-domain routing
  - Inter-domain routing
  - Politics and economics

- Inter-domain unicast routing
  - Each network administered separately - an autonomous system (AS)
    - Different technologies
    - Different policies
    - Mutual distrust – e.g., between AS and its peers

- Intra-domain 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)



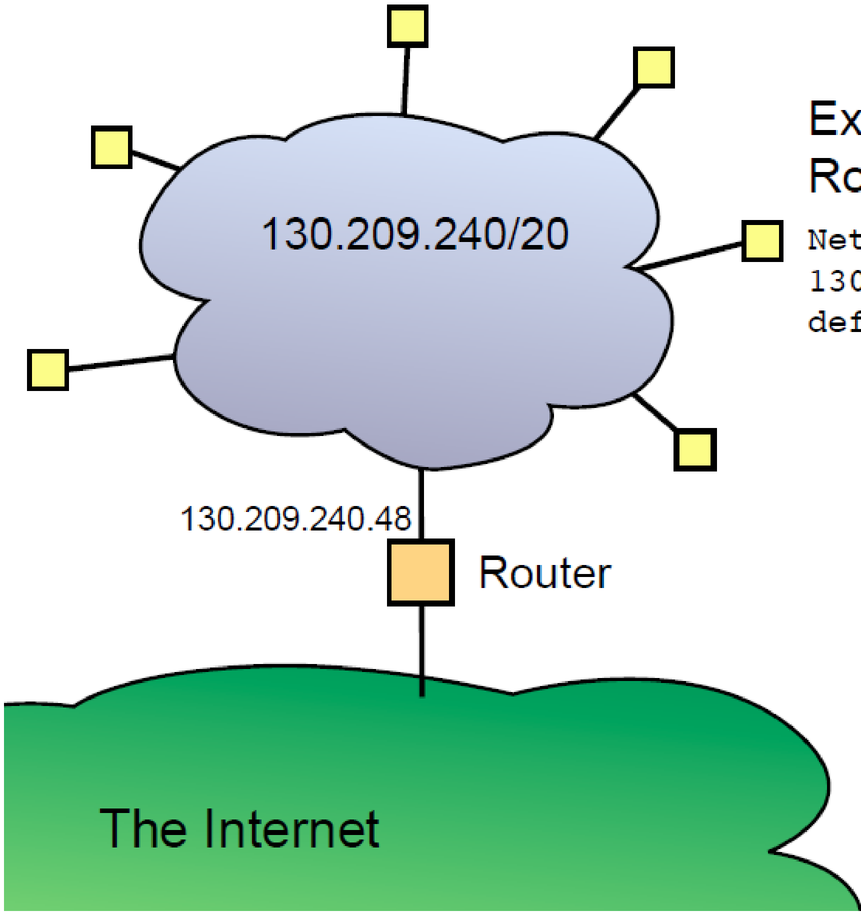
Based on slides © 2017 Colin Perkins

INTER-DOMAIN ROUTING

Inter-Domain Routing

- Goal: Find best route to *destination network*
  - Treat each network as a single node, and route without reference to internal network topology
- Network comprised of autonomous systems (ASes)
  - Each AS is an independently administered network
- Routing problem is finding best AS-level path from source AS to destination AS
  - Treat each AS as a node on the routing graph (the "AS topology graph")
  - Treat connections between ASes as edges in the graph
- 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 full routing table
  - The default free zone (DFZ) ;
  - Achievable at Regional Internet Registries (RIRs) and Internet Exchange Points (IXPs)

Routing at the edge



Example:
Routing table for hosts in Glasgow SoCS

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
  - Route based on policy, not necessarily 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

Aside: Inter-Domain Routing Policy

- Interdomain routing is between competitors
  - ASes are network operators and businesses that compete for customers
  - Implication: an AS is unlikely to trust its neighbours
- Routing must consider policy
  - Policy restrictions on who can determine your topology
  - Policy restrictions on which route data can follow
  - Prefer control over routing, even if that means data doesn’t necessarily follow the best (shortest) path
    - The shortest path might pass through a competitor’s network, or a country you politically disagree with, or over an expensive link…

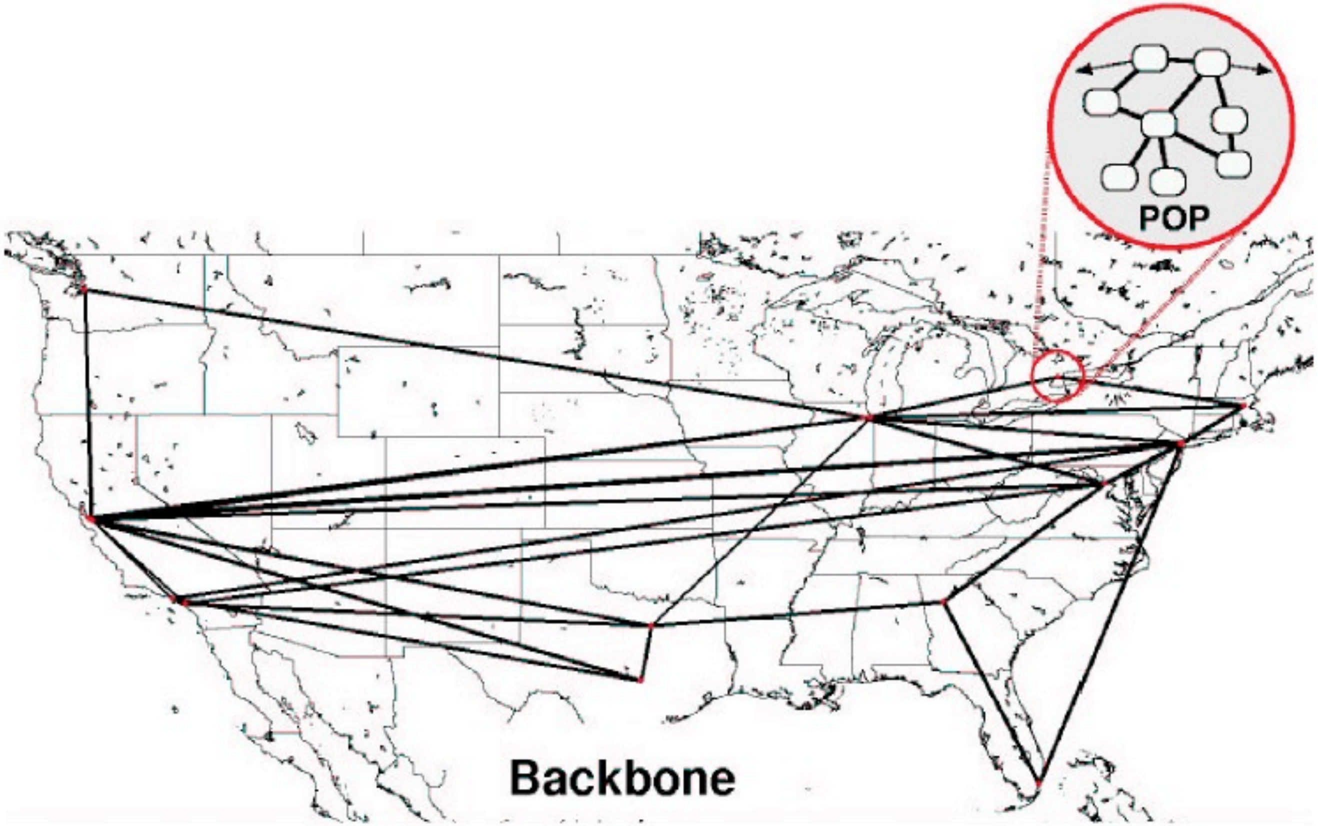
Extra: BGP

- Interdomain routing in the Internet uses the Border Gateway Protocol (BGP)
  - External BGP (eBGP) used to exchange routing information between ASes
    - Neighbouring ASes configure an eBGP session to exchange routes
    - Runs over a TCP connection between routers; exchanges knowledge of the AS graph topology
    - Used to derive "best" route to each destination; installed in routers to control forwarding
  - Internal BGP (iBGP) propagates routing information to routers within an AS
    - The intra-domain routing protocol handles routing within the AS
    - iBGP distributes information on how to reach external destinations
- Operation:
  - eBGP routers advertise lists of IP address ranges ("prefixes") and their associated AS-level paths
    - Combined to form a routing table
  - Each AS chooses what routes to advertise to its neighbours
    - Doesn't need to advertise everything it receives
    - Usual to drop some routes from the advertisement – depends on the chosen routing policy
    - Common approach: the Gao-Rexford rules:
      - Routes from customers advertised to everyone
      - Routes from peers and providers only advertised to customers
    - Ensures the AS graph is a valley-free DAG (recommended, but not required, policy)
  - BGP routers receive path vectors from neighbouring ASes giving possible routes to prefixes
    - Filtered based on the policy of each AS in the path from the source
- BGP decision process is complex and policy-driven
  - Choose what route to install for destination prefix in forwarding table based on multiple criteria – policy, shortest path, etc.
  - BGP doesn't always find a route, even if one exists, as may be prohibited by policy
  - Routes are often not the shortest AS path
  - Mapping business goals to BGP policies is a poorly documented process, with many operational secrets

Intra-Domain Routing

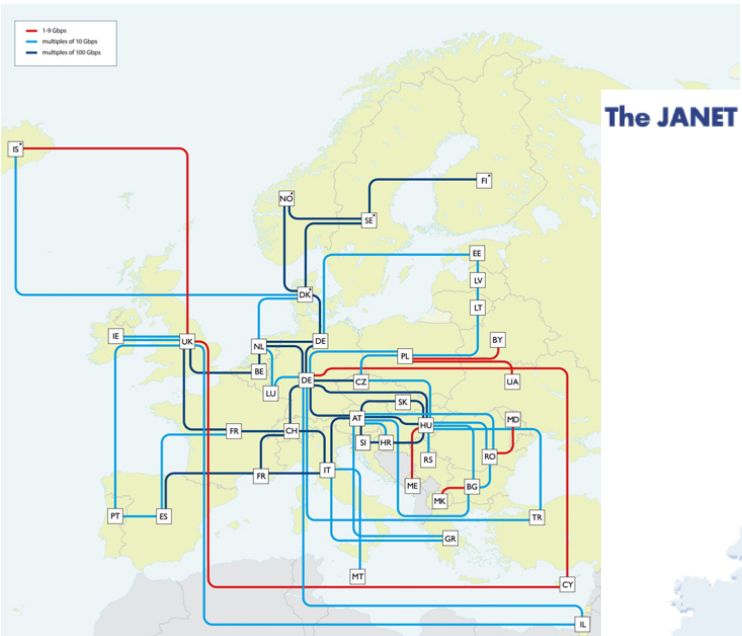
- Intra-domain routing operates within a network
  - Any network operated as a single entity – autonomous system
    - Could be local area, nationwide, or even worldwide
  - Operates a single routing protocol
    - Typically the OSPF link-state protocol exchanging routes to IP address prefixes
  - Running on IP routers within an autonomous system
    - Typically with fibre connections wide area and ethernet local area
  - Exchange routes to IP prefixes, representing regions in the network topology

Intra-Domain Routing



N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Network Topologies with Rocketfuel", Presentation at ACM SIGCOMM conference, 2002

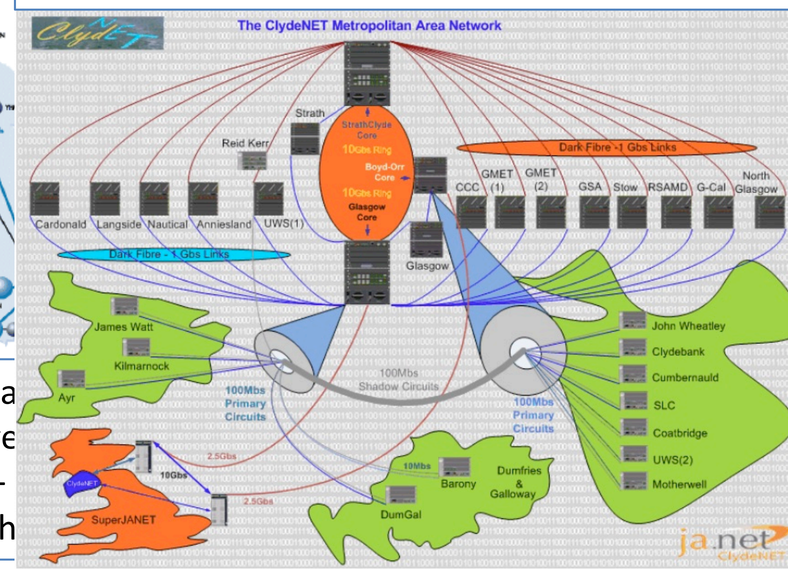
Intra-Domain Routing



The GÉANT network inter-connects
national research networks in Europ
POPs connect to networks such as JA

JANET is the UK national resea
It interconnects major unive
metropolitan area networks –
are a mix of end sites and oth

ClydeNet is the metropolitan area network for the Glasgow region – the regional JANET PoP



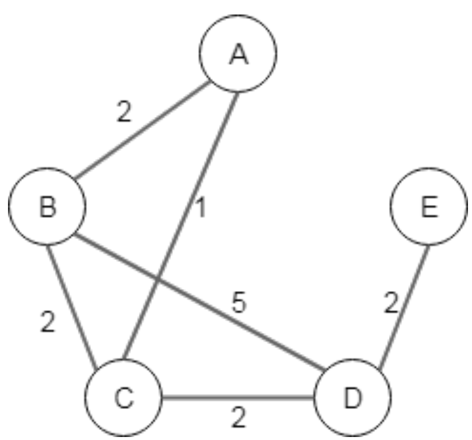
Distance Vector Protocols

- 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

- To make this easier to understand/visualise, assume that:
  - The protocol runs in rounds, somehow synchronised across all devices
    - Not necessary but makes reasoning about routing state easier
  - Each device sends its current distance metrics from its routing state/table to all its neighbours at the beginning of the round
    - Unreachable neighbours are marked as such
  - Each device waits until it has received updates from all of its reachable neighbours
  - Each device computes and stores its new routing table state, thus ending the current round

Distance Vector Protocols: Example

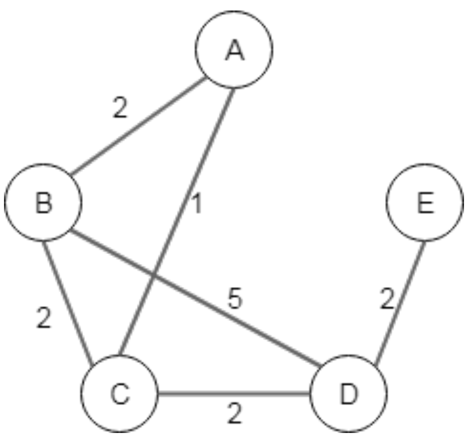
- Information stored at every node, to allow routing to other nodes:
  - Cost (length of path to X)
  - Next hop (next node on the path to X -- dash (-) signifies unknown next hop, dot (•) signifies no hop required)
  - Tables at every node encoded as a row in the table below
- This example uses names (A, B, C, …) to keep the diagram readable
  - Real implementations identify nodes by their IP address, or by IP prefixes if routing to networks
- Initially table is empty – know of no other nodes



Routing table at node	Cost / Next hop to node: A	B	C	D	E
A	0/•	∞/-	∞/-	∞/-	∞/-
B	∞/-	0/•	∞/-	∞/-	∞/-
C	∞/-	∞/-	0/•	∞/-	∞/-
D	∞/-	∞/-	∞/-	0/•	∞/-
E	∞/-	∞/-	∞/-	∞/-	0/•

Distance Vector Protocols: Example

- Time: 0
  - Nodes initialised; only know their immediate neighbours
  - Example: Routing table at A:
    - 0 hops to A
    - 1 hop to B (cost 2) and to C (cost 1), with each of these being the next hop
    - No known path to D and E, thus infinite cost and unknown next hop



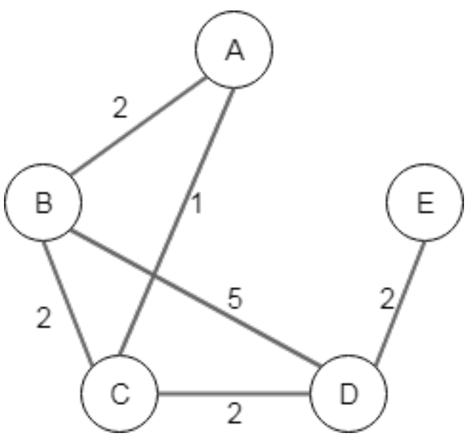
Routing table at node	Cost / Next hop to node A	B	C	D	E
A	0/•	**2/B**	**1/C**	∞/-	∞/-
B	**2/A**	0/•	**2/C**	**5/D**	∞/-
C	**1/A**	**2/B**	0/•	**2/D**	∞/-
D	∞/-	**5/B**	**2/C**	0/•	**2/E**
E	∞/-	∞/-	∞/-	**2/D**	0/•

Distance Vector Protocols: Example

- Time: 1
  - Routing data has spread one hop (Nodes also know neighbours of their neighbours)
  - Example: Updating of the routing table at A:
    - A receives routing state from B and C

B:	2	0	2	5	∞
C:	1	2	0	2	∞

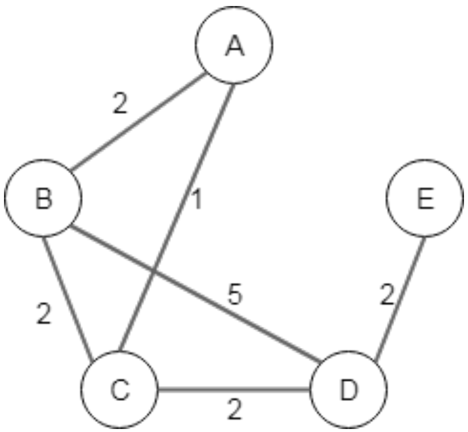
    - A then sees that it can reach D via both B (cost: 2 + 5) and C (1 + 2); C is thus selected as the next hop to D
    - B sees that it can reach D via C with a lower cost (2 + 2) than the previous one (5); also E via D (through C)



Routing table at node	Cost / Next hop to node: A	B	C	D	E
A	0/•	2/B	1/C	**3/C**	∞/-
B	2/A	0/•	2/C	**4/C**	**6/C**
C	1/A	2/B	0/•	2/D	**4/D**
D	**3/C**	**4/C**	2/C	0/•	2/E
E	∞/-	**7/D**	**4/D**	2/D	0/•

Distance Vector Protocols: Example

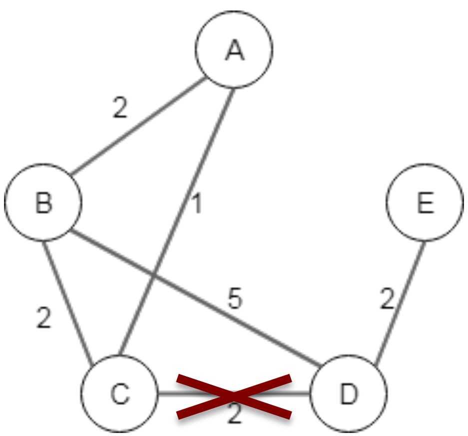
- Time: 2
  - Routing data has spread two hops (routing tables complete)
- Time: 3 onwards
  - Nodes continue to exchange distance metrics in case the topology changes



Routing table at node	Cost / Next hop to node: A	B	C	D	E
A	0/•	2/B	1/C	3/C	**5/C**
B	2/A	0/•	2/C	4/C	6/C
C	1/A	2/B	0/•	2/D	4/D
D	3/C	4/C	2/C	0/•	2/E
E	**5/D**	**6/D**	4/D	2/D	0/•

Distance Vector Protocols: Example

- Time: t
  - Assume the link between C and D fails
  - C/D notice, set the distance to each other to infinity and the next hop to unknown
  - C/D also update entries in their routing tables going through each other
    - E.g., C's route to E, D's route to A and B
  - C/D then send their updated tables to their neighbours when the time comes
  - *For simplicity assume we start from the state below (see end of slides for the actual process)*



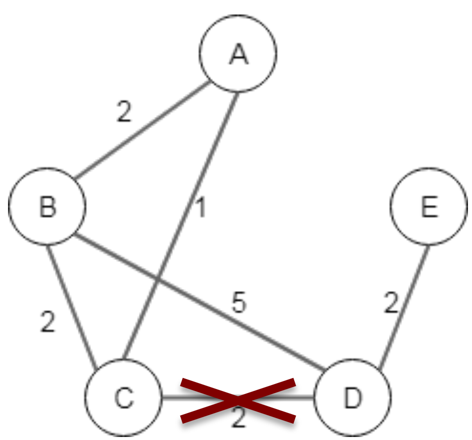
Routing table at node	Cost / Next hop to node A	B	C	D	E
A	0/•	2/B	1/C	∞/-	∞/-
B	2/A	0/•	2/C	5/D	∞/-
C	1/A	2/B	0/•	**∞/-**	**∞/-**
D	**∞/-**	**5/B**	**∞/-**	0/•	2/E
E	∞/-	∞/-	∞/-	2/E	0/•

Distance Vector Protocols: Example

- Time t+1
  - The update has propagated 1 hop; the network starts to converge again
  - Example: Updating of the routing table at D
    - D receives routing state from B and E
      - B:

B:	2	0	2	5	∞
E:	∞	∞	∞	2	0

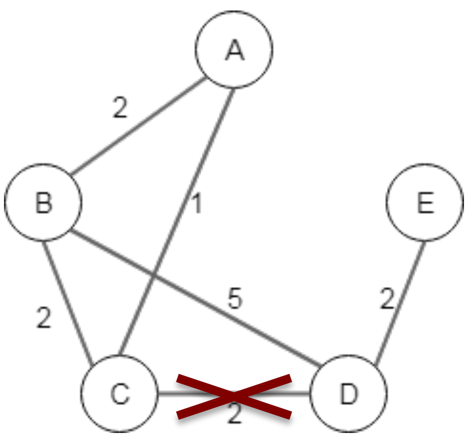
    - D then sees that it can reach A and C via B (cost: 5 to B plus 2/2 from there to A/C resp.)



Routing table at node	Cost / Next hop to node: A	B	C	D	E
A	0/•	2/B	1/C	**7/B**	∞/-
B	2/A	0/•	2/C	5/D	**7/D**
C	1/A	2/B	0/•	**7/B**	∞/-
D	**7/B**	5/B	**7/B**	0/•	2/E
E	∞/-	**7/D**	∞/-	2/E	0/•

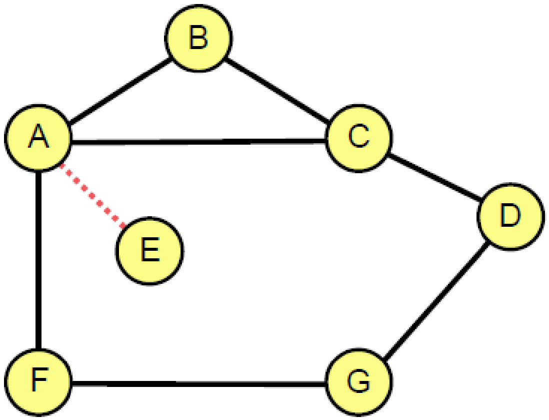
Distance Vector Protocols: Example

- Time t+2
  - The update has propagated 2 hops; the network will eventually converge
  - Example: Updating of the routing table at A
    - A receives routing state from B and C
      - B: | 2 | 0 | 2 | 5 | 7 |
      - C: | 1 | 2 | 0 | 7 | ∞ |
    - A then sees that it can reach E via B (cost: 2 to B plus 7 from there to E.)



Routing table at node	Cost / Next hop to node A	B	C	D	E
A	0/•	2/B	1/C	7/B	**9/B**
B	2/A	0/•	2/C	5/D	7/D
C	1/A	2/B	0/•	7/B	**9/B**
D	7/B	5/B	7/B	0/•	2/E
E	**9/D**	7/D	**9/D**	2/E	0/•

Distance Vector Protocols: Example



- 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…

Distance Vector Protocols: Limitations

- Count-to-infinity problem
  - 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 involving 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
    - Implies distance vector algorithm only suitable for small networks
- Distance vector routing tries to minimise state at nodes
  - As a consequence, is slow to converge
- An alternative is *link state* routing

Link State Routing

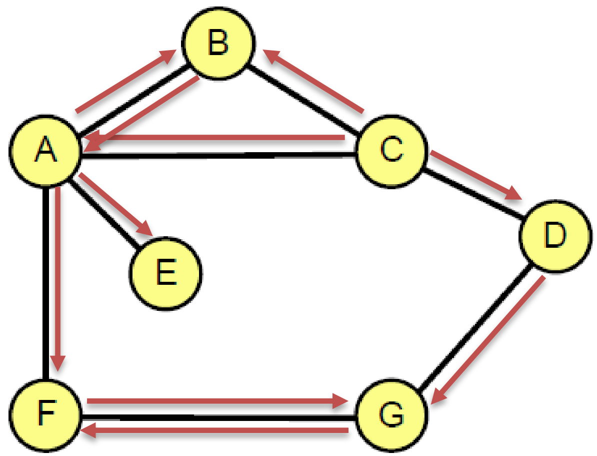
- Nodes know the links to their neighbours, and the cost of using those links – the link state information
- Reliably flood this information, giving all nodes complete map of the network
- Each node then directly calculates shortest path to every other node locally, uses this as routing table

- Link state information updates are flooded on start-up, and when the topology changes
- Each update contains:
  - The address of node that sent the update
  - List of directly connected neighbours of that node
    - With the cost of the link to each neighbour
    - With the range of addresses assigned to each link, if it connects to more than one host
  - A sequence number

Link State Routing: Forwarding & Route Updates

- Forward packets based on calculated shortest path
  - Static forwarding decision based on weights distributed by the routing protocol
  - Does not take into account network congestion
- Recalculate shortest paths on every routing update
  - Updates occur if a link fails, or a new link is added

Link State Routing: Example



- 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

Link State Routing: Shortest Path

- 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
- Optimal is assumed to be the shortest path, by weight

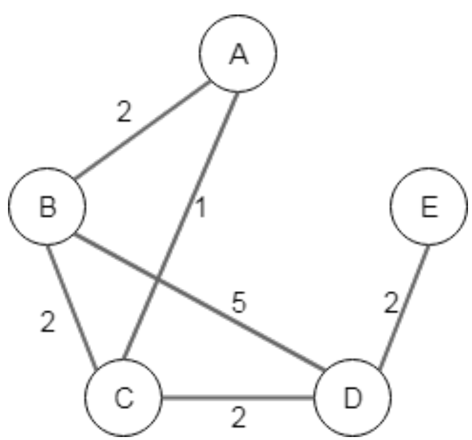
```
Input/Definitions:
      N             Set of all nodes in the graph
      l(i,j)        Weight (cost) of link from i to j (∞ if no link, 0 if i == j)
      s             Source node from which we are calculating shortest paths

Dijkstra's algorithm for an undirected connected graph:
      M = {s}                                       The set of nodes that have been checked so far.
      foreach n in N \ {s}:                         Initialise distance to all other nodes.
            C(n) = l(s, n)

      while (N \ M) ≠ {}:                           While there are nodes that haven't been checked.
            c = ∞, w = None                         Select node w in (N \ M) with minimum C(w).
            foreach n in (N \ M):
                  if C(n) < c:
                        w = n
                        c = C(w)
            M += {w}                                Add w to set of checked nodes.
            foreach n in (N \ M):                   For all nodes not checked yet…
                  if C(n) > C(w) + l(w,n):          …if the route to n through w is faster than the
                        C(n) = C(w) + l(w,n)        current one, then best route to n is through w.

Result:
      C(x): Cost of the shortest path from s to x
```

Link State vs Distance Vector

- Link State (Dijkstra's) result for s = A:
  - C{n} = {B: 2/B, C: 1/C, D: 3/C, E: 5/C}

  (you can see how this is obtained on your own at supplementary slides in the end)

- Distance Vector converged state:



Routing table at node	Cost / Next hop to node: A	B	C	D	E
A	**0/•**	**2/B**	**1/C**	**3/C**	**5/C**
B	2/A	0/•	2/C	4/C	6/C
C	1/A	2/B	0/•	2/D	4/D
D	3/C	4/C	2/C	0/•	2/E
E	5/D	6/D	4/D	2/D	0/•

Distance Vector vs Link State

Distance Vector Routing

- Simple to implement
- Routers only store the distance to each other node
  - $O(n)$
- Suffers from slow convergence
- Example: **Routing Information Protocol (RIP)**

Link State Routing

- More complex
- Requires each router to store complete network map
  - $O(n^2)$
- Much faster convergence
- Example: **Open Shortest Path First (OSPF)**

Slow convergence times make distance vector routing unsuitable for large networks

Reading material

- **Peterson & Davie "Computer Networks: A systems approach"**: Chapter 3, section 3.3
- **Kurose & Ross "Computer Networking: A top-down approach"** : Chapter 4, sections 4.5.1, 4.5.2, 4.6.1, 4.6.2
- **Tanenbaum & Wetherall "Computer Networks" 5th edition**: Chapter 5, sections 5.2.2, 5.2.3, 5.2.4, 5.2.5
- Bonaventure "Computer Networking" 1st edition: https://www.computer-networking.info/1st/html/network/network.html#routing-in-ip-networks

Coming up next...

I'd tell you a UDP joke
but you might not get it...

What is the best part about TCP jokes?

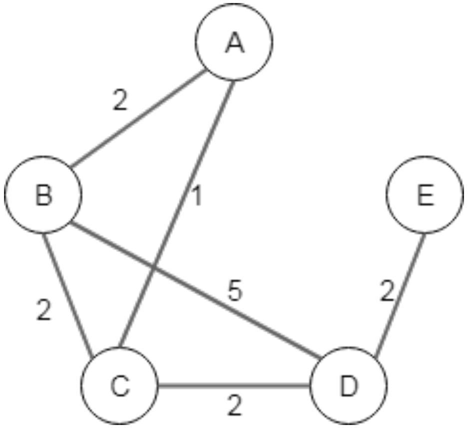
I get to keep telling them until you get them...

Examples

Link State Routing: Shortest Path

- s = A
- M = {A}
- N \ {A} = {B, C, D, E}
  - C{n} = {B: 2/B, C: 1/C, D: ∞, E: ∞}
- N \ M = {B, C, D, E}
  - w = C, c = 1
  - M = {A, C}
  - N \ M = {B, D, E}
    - C{B} = 2 *vs* 1 + 2
      ➔ no change
    - C{D} = ∞ *vs* 1 + 2
      ➔ C{D} = 3, path to D via C
    - C{E} = ∞ *vs* 1 + ∞
      ➔ no change
  - C{n} = {B: 2/B, C: 1/C, D: 3/C, E: ∞}

```
Input/Definitions:
      N           Set of all nodes in the graph
      l(i,j)      Weight (cost) of link from i to j
      s           Source node
Dijkstra's alg. for an undirected connected graph:
      M = {s}
      foreach n in N \ {s}:
            C(n) = l(s, n)
      while (N \ M) ≠ {}:
            c = ∞, w = None
            foreach n in (N \ M):
                  if C(n) < c:
                        w = n
                        c = C(w)
            M += {w}
            foreach n in (N \ M):
                  if C(n) > C(w) + l(w,n):
                        C(n) = C(w) + l(w,n)
Result:
      C(x): Cost of the shortest path from s to x
```



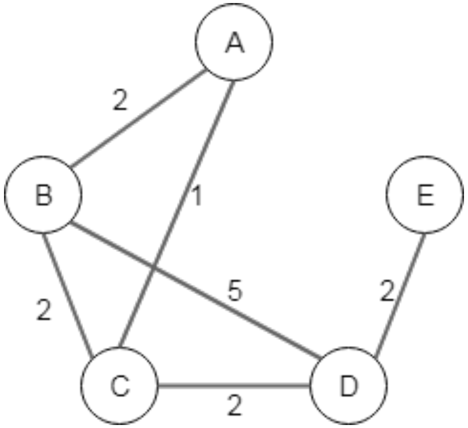
Link State Routing: Shortest Path

- $C\{n\} = \{B: 2/B, C: 1/C, D: 3/C, E: \infty\}$
- $N \setminus M = \{B, D, E\}$
  - $w = B$, $c = 2$
  - $M = \{A, B, C\}$
  - $N \setminus M = \{D, E\}$
    - $C\{D\} = 3$ *vs* $2 + 5$

      ➔ no change
    - $C\{E\} = \infty$ *vs* $2 + \infty$

      ➔ no change
  - $C\{n\} = \{B: 2/B, C: 1/C, D: 3/C, E: \infty\}$

```
Input/Definitions:
      N             Set of all nodes in the graph
      l(i,j)        Weight (cost) of link from i to j
      s             Source node
Dijkstra's alg. for an undirected connected graph:
      M = {s}
      foreach n in N \ {s}:
            C(n) = l(s, n)
      while (N \ M) ≠ {}:
            c = ∞, w = None
            foreach n in (N \ M):
                  if C(n) < c:
                        w = n
                        c = C(w)
            M += {w}
            foreach n in (N \ M):
                  if C(n) > C(w) + l(w,n):
                        C(n) = C(w) + l(w,n)
Result:
      C(x): Cost of the shortest path from s to x
```

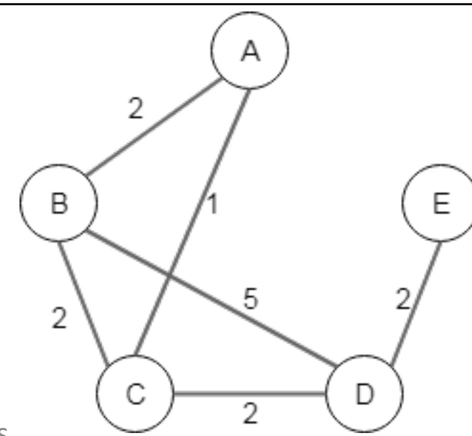


Link State Routing: Shortest Path

- C{n} = {B: 2/B, C: 1/C, D: 3/C, E: ∞}
- N \ M = {D, E}
  - w = D, c = 3
  - M = {A, B, C, D}
  - N \ M = {E}
    - C{E} = ∞ *vs* 3 + 2

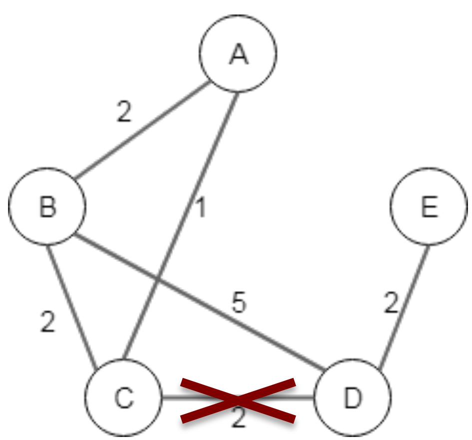
      ➔ 5, path to E via D (via C)
  - C{n} = {B: 2/B, C: 1/C, D: 3/C, E: 5/C}
- C{n} = {B: 2/B, C: 1/C, D: 3/C, E: 5/C}
- N \ M = {E}
  - w = E, c = 5
  - M = {A, B, C, D, E}
  - N \ M = {}
- **s = A:**
  - **C{n} = {B: 2/B, C: 1/C, D: 3/C, E: 5/C}**

```
Input/Definitions:
        N            Set of all nodes in the graph
        l(i,j)       Weight (cost) of link from i to j
        s            Source node
Dijkstra's alg. for an undirected connected graph:
        M = {s}
        foreach n in N \ {s}:
                C(n) = l(s, n)
        while (N \ M) ≠ {}:
                c = ∞, w = None
                foreach n in (N \ M):
                        if C(n) < c:
                                w = n
                                c = C(w)
                M += {w}
                foreach n in (N \ M):
                        if C(n) > C(w) + l(w,n):
                                C(n) = C(w) + l(w,n)
Result:
        C(x): Cost of the shortest path from s to x
```



Distance Vector Protocols: Example

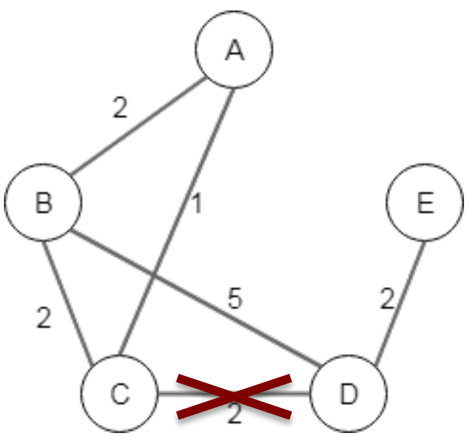
- Time: t
  - Assume the link between C and D fails
  - C/D notice, set the distance to each other to infinity and the next hop to unknown
  - C/D also update entries in their routing tables going through each other
    - E.g., C's route to E, D's route to A and B
  - C/D then send their updated tables to their neighbours when the time comes
  - *This is how this situation would actually pan out, step by step*



Routing table at node	Cost / Next hop to node A	B	C	D	E
A	0/•	2/B	1/C	3/C	5/C
B	2/A	0/•	2/C	4/C	6/C
C	1/A	2/B	0/•	∞/-	∞/-
D	∞/-	∞/-	∞/-	0/•	2/E
E	5/D	6/D	4/D	2/D	0/•

Distance Vector Protocols: Example

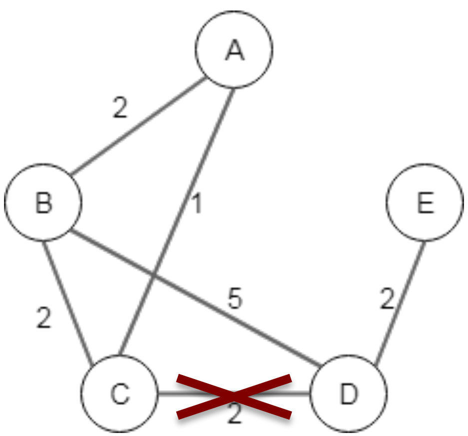
- Time: t+1
  - C/D then send their updated tables to their neighbours, receive their vectors in the meanwhile



		Cost / Next hop to node				
		A	B	C	D	E
Routing table at node	A	0/•	2/B	1/C	**6/B**	**8/B**
	B	2/A	0/•	2/C	**5/D**	**7/D**
	C	1/A	2/B	0/•	**4/A**	**6/A**
	D	**7/B**	**5/B**	**6/E**	0/•	2/E
	E	**∞/-**	**∞/-**	**∞/-**	2/D	0/•

Distance Vector Protocols: Example

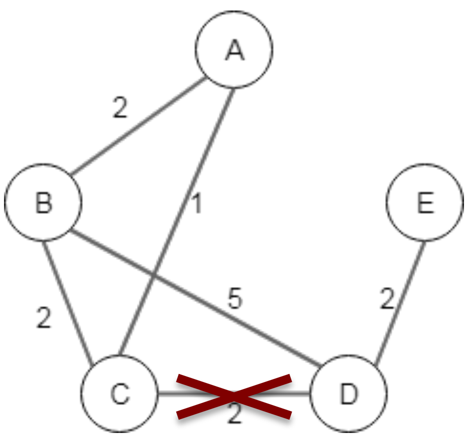
- Time: t+2



Routing table at node	Cost / Next hop to node				
	A	B	C	D	E
A	0/•	2/B	1/C	**5/C**	**7/C**
B	2/A	0/•	2/C	5/D	7/D
C	1/A	2/B	0/•	**7/A**	**9/A**
D	7/B	5/B	**7/B**	0/•	2/E
E	**9/D**	7/D	**8/D**	2/D	0/•

Distance Vector Protocols: Example

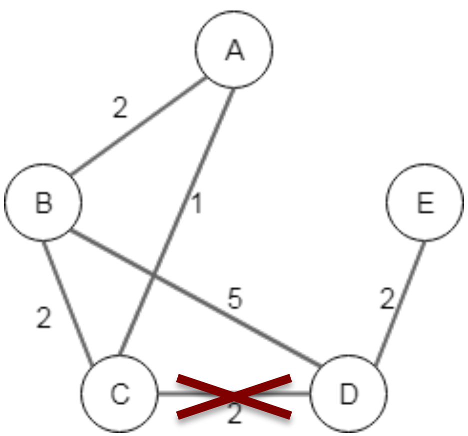
- Time: t+3



		Cost / Next hop to node				
		A	B	C	D	E
Routing table at node	A	0/•	2/B	1/C	**7/B**	**9/B**
	B	2/A	0/•	2/C	5/D	7/D
	C	1/A	2/B	0/•	**6/A**	**8/A**
	D	7/B	5/B	7/B	0/•	2/E
	E	9/D	7/D	**9/D**	2/D	0/•

Distance Vector Protocols: Example

- Time: t+4



		Cost / Next hop to node				
		A	B	C	D	E
Routing table at node	A	0/•	2/B	1/C	7/B	9/B
	B	2/A	0/•	2/C	5/D	7/D
	C	1/A	2/B	0/•	**7/B**	**9/B**
	D	7/B	5/B	7/B	0/•	2/E
	E	9/D	7/D	9/D	2/D	0/•