

---

# Communications Network Design

## lecture 07

Matthew Roughan

<matthew.roughan@adelaide.edu.au>

Discipline of Applied Mathematics  
School of Mathematical Sciences  
University of Adelaide

March 26, 2009

---

Communications Network Design: lecture 07 – p.1/44

---

This lecture continues the discussion of shortest-path routing. It provides a new algorithm (Floyd-Warshall) and so details of how shortest-path routing is implemented in the Internet.

---

Communications Network Design: lecture 07 – p.1/44

---

# Routing (continued)

We continue the algorithmic viewpoint by considering an alternative to Dijkstra called the Floyd-Warshall algorithm. Also we consider routing implementation: OSPF, IS-IS, and some miscellaneous issues such as load balancing. Finally we will look into the distributed Bellman-Ford dynamic programming algorithm as implemented in RIP.

---

Communications Network Design: lecture 07 – p.2/44

---

Communications Network Design: lecture 07 – p.2/44

# Floyd-Warshall

---

Alternative to Dijkstra for **all-pairs** shortest path problem

- ▶ same input as Dijkstra (except no start node)
- ▶ add nodes in one by one, and compute shortest paths as you add in a node
  - ▷ shortest path is either the same
  - ▷ or changes to include the new node

# Floyd-Warshall

---

Let  $D_{ij}^{(k)}$  denote the shortest path length from node  $i$  to node  $j$  using intermediate nodes from 1 to  $k$  only.

**Initialise:**  $D_{ij}^{(0)} = d_{ij} \quad \forall i, j \in N$   
 $V^{(0)} = [0]$ , an  $|N| \times |N|$  zero matrix.

**Step:** for  $k = 1, 2, \dots, n$ , compute new distance estimates

$$D_{ij}^{(k)} = \min\{D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\} \quad \forall i \neq j$$

Compute the predecessor nodes

If  $D_{ij}^{(k)} < D_{ij}^{(k-1)}$  put  $V_{ij}^{(k)} = k$ ;

otherwise,  $V_{ij}^{(k)} = V_{ij}^{(k-1)}$

# Floyd-Warshall

---

- ▶ The initialisation step gives the shortest path lengths subject to no intermediate nodes
- ▶ For a given  $k$ ,  $D_{ij}^{(k-1)}$  gives the shortest path from  $i$  to  $j$  using only nodes 1 through  $k-1$  as possible intermediate nodes.
- ▶ On allowing node  $k$  as an intermediate node, either  $k$  IS on the shortest path, or it isn't.
  - ▷ **it isn't:** keep the same distance, and path
    - ★  $D_{ij}^{(k)} = D_{ij}^{(k-1)}$  and  $V_{ij}^{(k)} = V_{ij}^{(k-1)}$
  - ▷ **it is:** the new path must be made of two shortest paths, joined by node  $k$ , i.e.  $i-k$  and  $k-j$ 
    - ★  $D_{ij}^{(k)} = D_{ik}^{(k-1)} + D_{kj}^{(k-1)}$
    - ★  $V_{ij}^{(k)}$  shows where the join occurred

---

Communications Network Design: lecture 07 – p.5/44

---

Communications Network Design: lecture 07 – p.5/44

# Floyd-Warshall

---

- ▶ The 0's in  $V^{(n)}$  determine the adjacencies (links) in the final network.
  - ▷  $V_{ij}^{(n)}$  indicates that we never found a shorter path than  $d_{ij}$  along the direct path.
  - ▷ hence  $i$  and  $j$  are adjacent in the SPF tree
- ▶ The other terms in  $V^{(n)}$  show the predecessor nodes for each end-to-end path.
  - ▷ construct paths, by concatenating predecessor nodes

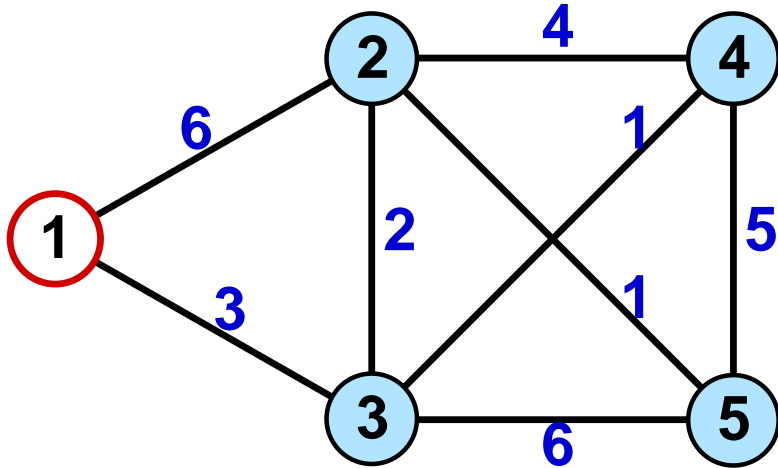
---

Communications Network Design: lecture 07 – p.6/44

---

Communications Network Design: lecture 07 – p.6/44

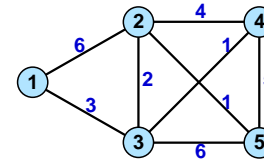
# Floyd-Warshall example



# Floyd-Warshall example

Initially, we put direct links into the matrix D

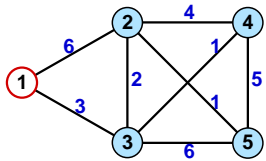
$$D_{ij}^{(0)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 6 & 3 & \infty & \infty \\ 2 & & 0 & 2 & 4 & 1 \\ 3 & & & 0 & 1 & 6 \\ 4 & & & & 0 & 5 \\ 5 & & & & & 0 \end{array}$$

$$V^{(0)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & & 0 & 0 & 0 & 0 \\ 3 & & & 0 & 0 & 0 \\ 4 & & & & 0 & 0 \\ 5 & & & & & 0 \end{array}$$


# Floyd-Warshall example

$k = 1$ : include node 1 on existing direct paths (so any path already containing node 1 e.g. top line and first column of  $D$ , can be ignored). Here, nothing changes.

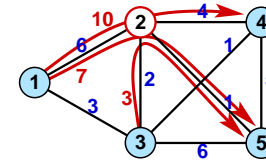
$$D_{ij}^{(1)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 6 & 3 & \infty & \infty \\ 2 & & 0 & 2 & 4 & 1 \\ 3 & & & 0 & 1 & 6 \\ 4 & & & & 0 & 5 \\ 5 & & & & & 0 \end{array}$$

$$V^{(1)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & & 0 & 0 & 0 & 0 \\ 3 & & & 0 & 0 & 0 \\ 4 & & & & 0 & 0 \\ 5 & & & & & 0 \end{array}$$


# Floyd-Warshall example

$k = 2$ : try including node 2 on existing paths (so any path already containing node 2 e.g. line 2 and second column of  $D$ , can be ignored).

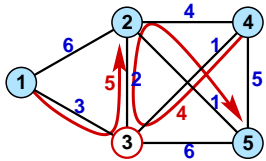
$$D_{ij}^{(2)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 6 & 3 & 10 & 7 \\ 2 & & 0 & 2 & 4 & 1 \\ 3 & & & 0 & 1 & 3 \\ 4 & & & & 0 & 5 \\ 5 & & & & & 0 \end{array}$$

$$V^{(2)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 0 & 2 & 2 \\ 2 & & 0 & 0 & 0 & 0 \\ 3 & & & 0 & 0 & 2 \\ 4 & & & & 0 & 0 \\ 5 & & & & & 0 \end{array}$$


# Floyd-Warshall example

$k = 3$ : try including node 3 on existing paths (so any path already containing node 3 e.g. line 3 and third column of  $D$ , can be ignored).

$$D_{ij}^{(3)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 5 & 3 & 4 & 6 \\ 2 & & 0 & 2 & 3 & 1 \\ 3 & & & 0 & 1 & 3 \\ 4 & & & & 0 & 4 \\ 5 & & & & & 0 \end{array}$$

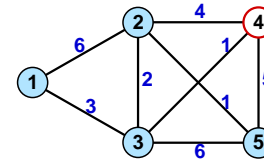
$$V^{(3)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 0 & 3 & 3 \\ 2 & & 0 & 0 & 3 & 0 \\ 3 & & & 0 & 0 & 2 \\ 4 & & & & 0 & 3 \\ 5 & & & & & 0 \end{array}$$


E.G. The old path joining 4-5 was a direct link with distance  $D_{45}^{(2)} = 5$ . But when we are allowed to include node 3, we get an alternative  $D_{43}^{(2)} + D_{35}^{(2)} = 4$ , which is better, so we set  $D_{45}^{(3)} = 4$ , and  $V_{45}^{(3)} = 3$ .

# Floyd-Warshall example

$k = 4$ : try including node 4 on existing paths:  
No changes.

$$D_{ij}^{(4)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 5 & 3 & 4 & 6 \\ 2 & & 0 & 2 & 3 & 1 \\ 3 & & & 0 & 1 & 3 \\ 4 & & & & 0 & 4 \\ 5 & & & & & 0 \end{array}$$

$$V^{(4)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 0 & 3 & 3 \\ 2 & & 0 & 0 & 3 & 0 \\ 3 & & & 0 & 0 & 2 \\ 4 & & & & 0 & 3 \\ 5 & & & & & 0 \end{array}$$


# Floyd-Warshall example

$k = 5$ : try including node 5 on existing paths. The entries  $D_{ij}^{(5)}$  give the length of the shortest path from each node  $i$  to each other node  $j$ .

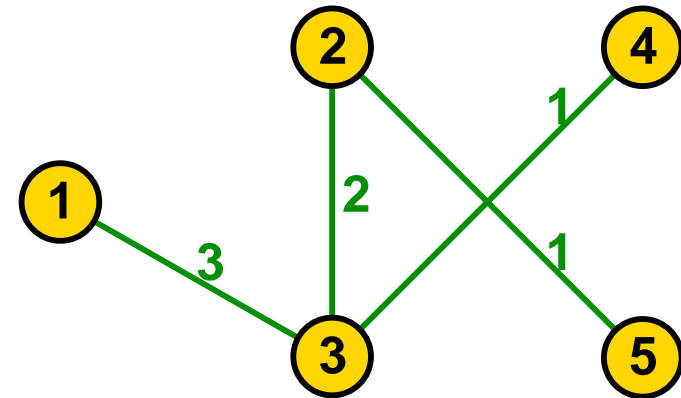
		1	2	3	4	5
1	0	5	3	4	6	
2		0	2	3	1	
3			0	1	3	
4				0	4	
5					0	

 $D_{ij}^{(5)} =$ 

		1	2	3	4	5
1	0	3	<span style="border: 1px solid black;">0</span>	3	3	
2		0	<span style="border: 1px solid black;">0</span>	3	<span style="border: 1px solid black;">0</span>	
3			0	<span style="border: 1px solid black;">0</span>	2	
4				0	3	
5					0	

Use the boxed zero entries in the final  $V$  to determine links: (1,3), (2,3), (2,5), (3,4).

# Floyd-Warshall shortest paths



Note that the solution is the same as the one we found with Dijkstra. However, that isn't guaranteed. In some cases, there may be multiple equal-length shortest paths. The path you find depends on the ordering of the nodes in the various techniques, so depending on implementation, even two different versions of Dijkstra could return a different SPF tree. We will consider this issue a little more later on.

## Floyd-Warshall complexity

---

- ▶ In calculating  $D_{ij}^{(k)}$  at each step, we need to compare two possibilities for each of  $\frac{|N|(|N|-1)}{2}$  pairs of nodes.
- ▶ the algorithm has  $|N|$  steps
- ▶ total computational complexity is  $O(|N|^3)$ .
- ▶ This of course is the same as repeating simple version of Dijkstra's algorithm  $|N|$  times (for each of  $|N|$  sources)

## Alternative algorithms

---

- ▶ Dijkstra and FW assume non-negative weights
- ▶ not a problem for network applications
- ▶ for more general applications, use Bellman-Ford
  - ▷ can be used on graphs with negative edge weights
  - ▷ as long as the graph contains no negative cycle reachable from the source node
- ▶ Johnson's algorithm solves all pairs shortest paths, may be faster than Floyd-Warshall on sparse graphs.

---

We will see a version of Bellman-Ford later on in this course.



# Routing implementation

---

- ▶ must obtain **consistent** results between routers
  - ▷ to avoid route loops, or dead-ends
- ▶ must **adapt** to changing network
  - ▷ route around link or node failures
- ▶ must use a **distributed** algorithm
  - ▷ an algorithm which enables a common objective of two or more peer processes to be performed jointly by the combination of processing and exchanging information.
  - ▷ The distributed algorithm is broken down into a set of local algorithms, one of which is performed by each peer process.
  - ▷ Each local process carries out various operations on the available data, and at various points in the algorithm, it sends/receives data to/from other peer processes.

---

Communications Network Design: lecture 07 – p.17/44

---

Communications Network Design: lecture 07 – p.17/44

# SPF implementation

---

Implementation is performed by a routing protocol

- ▶ routing protocol performs SPF calculation
- ▶ first needs to find out the topology, and weights
- ▶ each router **floods** its available topology information to all other routers
  - ▷ takes the form of LSAs
    - \* Link State Announcements
    - \* a router sends LSA describing its links to adjacent routers
      - ♦ LSA includes link weight
    - \* neighbours forward (non-duplicate) LSAs to their neighbours
  - ▷ hence this is called a **link-state** routing protocol

---

Communications Network Design: lecture 07 – p.18/44

---

Communications Network Design: lecture 07 – p.18/44

## SPF implementation

---

- ▶ once a router has seen all LSA
  - ▷ it knows the complete topology
  - ▷ it can perform Dijkstra to compute shortest paths to all other routers
- ▶ note that each router only needs to perform Dijkstra once
  - ▷ it only needs to know paths from itself, to the other routers.
  - ▷ hence  $O(|N|^2)$  for simple implementation
  - ▷  $O(|N|^3)$  workload is distributed over  $|N|$  routers

---

Communications Network Design: lecture 07 – p.19/44

---

Communications Network Design: lecture 07 – p.19/44

## SPF routing implementations

---

- ▶ common implementations
  - ▷ OSPF [1]
    - ★ Open Shortest Path First
    - ★ several RFCs needed to see all possibilities
  - ▷ IS-IS [2]
    - ★ Intermediate System-Intermediate System
    - ★ several RFCs needed to see all possibilities
- ▶ some amusement: RFC 4041, "Requirements for Morality Sections in Routing Area Drafts"  
<ftp://ftp.rfc-editor.org/in-notes/rfc4041.txt>
  - It has often been the case that morality has not been given proper consideration in the design and specification of protocols produced within the Routing Area. This has led to a decline in the moral values within the Internet and attempts to retrofit a suitable moral code to implemented and deployed protocols has been shown to be sub-optimal...

---

Communications Network Design: lecture 07 – p.20/44

---

Communications Network Design: lecture 07 – p.20/44

# OSPF

---

- ▶ **soft state**
  - ▷ periodically refresh LSA information
  - ▷ also exchange `hello` messages (between neighbouring routers) to test link states
  - ▷ in case a failure happens, and isn't detected
- ▶ **not routed**
  - ▷ LSAs are just sent in IP packets
    - ★ like everything else
  - ▷ transmitted over IP (protocol 89)
    - ★ not over TCP, so not reliable transport
  - ▷ but you can't route, until you have routes
  - ▷ hence forwarding of LSAs is limited to **adjacent routers**

---

Communications Network Design: lecture 07 – p.21/44

---

Communications Network Design: lecture 07 – p.21/44

# Scaling of OSPF

---

- ▶ as noted earlier, if  $|N|$  is too large, computing SPF takes too long, and we run into problems
- ▶ how can you build large ( $|N| \sim 1000$ ) networks
- ▶ use (2 level) hierachy
  - ▷ in subnetworks compute shortest paths
  - ▷ compute the shortest paths between subnets
  - ▷ combine the two
- ▶ not as simple as it sounds
  - ▷ **example OSPF areas**
  - ▷ **area 0** is the backbone (1st level)
  - ▷ **other areas** are the subnetworks (2nd level)

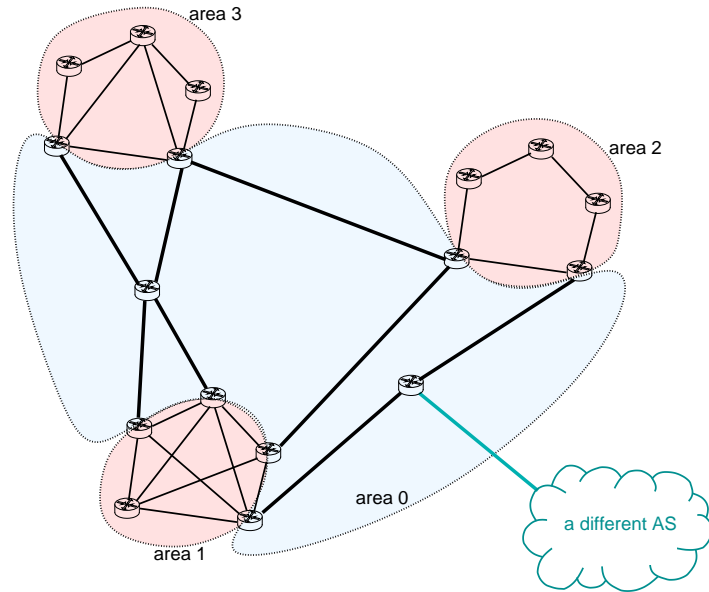
---

Communications Network Design: lecture 07 – p.22/44

---

Communications Network Design: lecture 07 – p.22/44

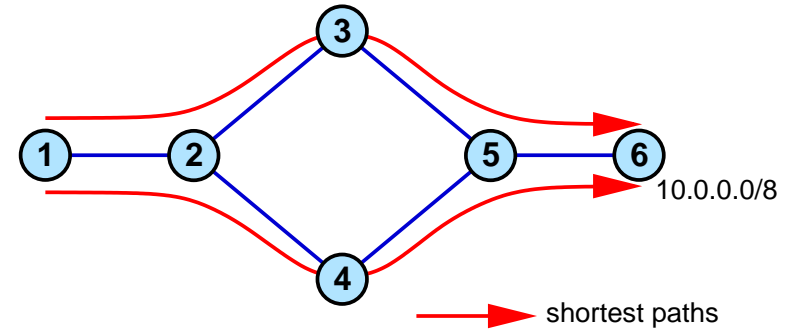
# Scaling of OSPF



Communications Network Design: lecture 07 – p.23/44

# Load balancing

- ▶ in some cases there will be two (or more) equal distance paths from source to destination
- ▶ Dijkstra and FW only give you one path
- ▶ solution is non-unique
- ▶ more efficient to share load over both paths

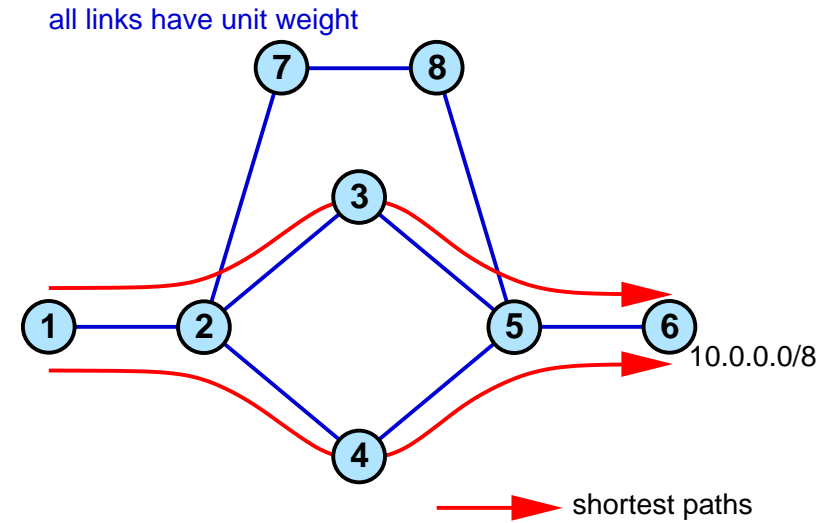


Communications Network Design: lecture 07 – p.24/44

## Dijkstra and load balancing

- ▶ for all destination nodes in graph, you have a shortest path
- ▶ start at a particular destination
- ▶ recursively descend through neighbours at the right distance back
- ▶ algorithm exponential in number of paths, but this is hopefully small

## Dijkstra and load balancing ex.



# Load balancing implementation

- ▶ method one
  - ▷ split traffic up by addresses
  - ▷ instead of a simple forwarding table
    - \* e.g. at router 2, the next hop router to prefix 10.0.0.0/8 is router 3
  - ▷ have two forwarding table entries
  - ▷ e.g. forwarding table (at router 2)

destination	next hop router
10.0.0.0/9	3
10.1.0.0/9	4

- ▶ traffic between different prefixes may be uneven

# Load balancing implementation

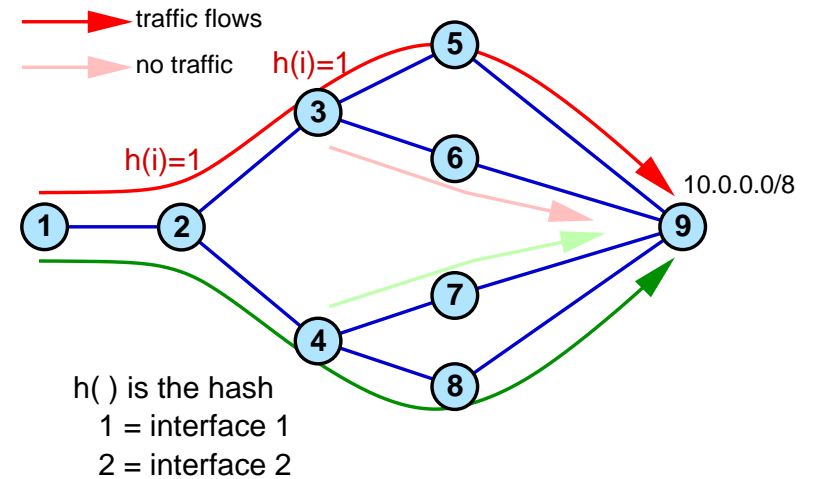
- ▶ method two
    - ▷ need multiple paths in forwarding table
- | destination | next hop router |
|-------------|-----------------|
| 10.0.0.0/8  | 3 or 4          |
- ▶ allocate traffic between two next hops randomly as it arrives
  - ▶ method is simpler to administrate
  - ▶ better balance of traffic
  - ▶ may reorder packets
- ▶ method two(b)
    - ▷ randomize first packet of a flow
    - ▷ subsequent packets of flow follow same route

# Load balancing implementation

- ▶ method three
  - ▷ allocate traffic randomly between two paths
  - ▷ but randomization is based on a **hash** of the IP source and destination address
  - ▷ effect is random allocation
    - \* but with all packets between same source and destination using the same path
    - \* so no reordering within a TCP connection
  - ▷ hash needs to be randomized at each node, otherwise multiple splits don't work
    - \* different seeds for randomization at each router

# Load balancing implementation

- ▶ method 3 without random seeds in hashes



# Link weights

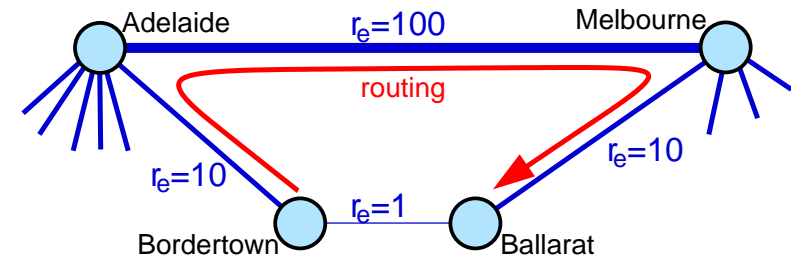
What should be the link weights  $\alpha_e$ ?

- ▶ real, physical distance?
- ▶ delay of packets along link?
- ▶ hop count (e.g.  $\alpha_e = 1$ )?
- ▶ some arbitrary number?

Cisco default

- ▶ inverse capacity weights  $\alpha_e = A/r_e$
- ▶ the higher capacity links are nominally "shorter"
- ▶ encourages traffic to use higher capacity links
- ▶ it can lead to weird routing

# Link weights



- ▶ may think don't need link between Ballarat and Bordertown, because it has no traffic
- ▶ but its just because routing is taking a longer path
  - ▷ direct path:  $D = w_e = 100/r_e = 100$
  - ▷ indirect path:  $D = 10 + 10 + 1 = 21$
- ▶ inverse capacity is often the wrong choice



# Link weights

---

- ▶ correct choice depends on objectives
- ▶ common cases occur when minimizing delays:
  - ▷ if propagation delay is dominant
    - \* minimize physical path distance
    - \* weight = link distance, e.g.  $\alpha_e = d_e$
  - ▷ if processing and transmission time dominate
    - \* minimize the hop count, e.g.  $\alpha_e = 1$
  - ▷ if queueing causes most delays, need to minimize loads on links
    - \* early ARPANET had load-sensitive routing
    - \* measured packet delays along links (to get  $\alpha_e$ )
    - \* sent packet along shortest (delay) path
- ▶ can also write link weight choice as an optimization problem (called **traffic engineering**)

Communications Network Design: lecture 07 – p.33/44

Communications Network Design: lecture 07 – p.33/44

# Incremental Dijkstra

---

As noted above, Dijkstra doesn't scale as well as we might like.

- ▶ network of 1000 nodes need some kind of hierarchy
- ▶ alternatively, note that most of the time the network doesn't change
  - ▷ when it does change, it is usually only a local change in a few links
  - ▷ perhaps we don't have to recompute everything from scratch?
- ▶ incremental Dijkstra algorithm
- ▶ latest implementations use incremental Dijkstra.

[http://www.cisco.com/en/US/about/ac123/ac114/ac173/Q3-04/sp\\_calculate.html](http://www.cisco.com/en/US/about/ac123/ac114/ac173/Q3-04/sp_calculate.html)

Communications Network Design: lecture 07 – p.34/44

Communications Network Design: lecture 07 – p.34/44

# Generalization

---

We focused here on IP routing

- ▶ but routing is needed in most communications networks

Shortest paths used in many areas - not just communications networks

- ▶ there are many other types of networks
  - ▷ often want shortest paths on these
  - ▷ e.g. for finding close linkages in social networks
- ▶ not always obvious what's a network
  - ▷ Dijkstra used in image processing
  - ▷ pixels form a grid, which is a network
- ▶ Dijkstra is often a component of another algorithm

---

Communications Network Design: lecture 07 – p.35/44

# Link state vs Distance Vector

---

- ▶ We saw OSPF was a link-state routing protocol
  - ▷ floods topology (link states), and computes SPF
  - ▷ solves shortest path problem
- ▶ alternative is called distance-vector protocol
  - ▷ examples: RIP, IGRP, ...
  - ▷ originally also aimed to solve shortest paths
    - ★ but nodes don't need to know complete topology
- ▶ hybrids exist, e.g. EIGRP

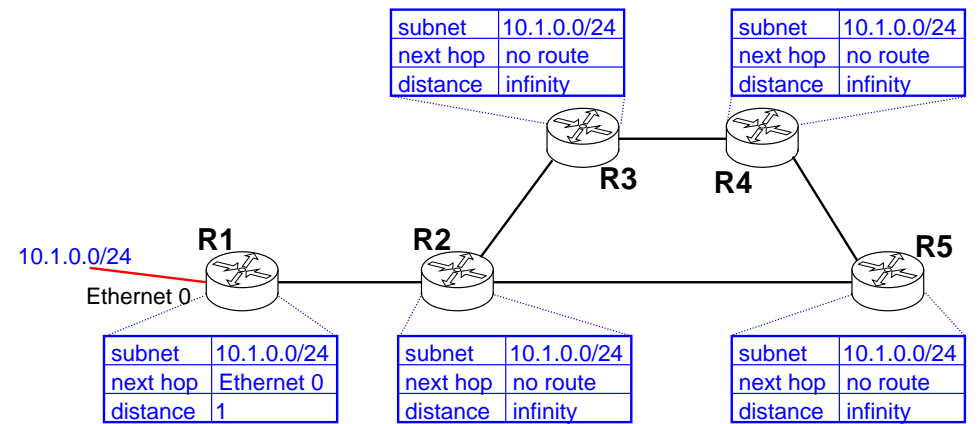
---

Communications Network Design: lecture 07 – p.36/44

# Distance Vector

- ▶ Make a list of destinations you can reach and the distance to these destinations.
  - ▷ Store in routing table
- ▶ Share this list with your neighbours
- ▶ Add to routing table new information gained from adjacent routers about the destinations they can reach
  - ▷ remember to increment their distance
  - ▷ keep the source as the next hop
- ▶ If two paths to the same destination exists, keep the shortest distance path.
- ▶ Repeat periodically (in RIP every 30 seconds).

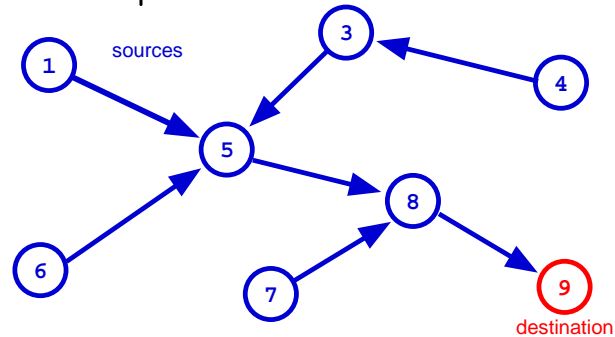
# Distance Vector example



# Sink trees

Results of algorithm must be a sink tree

- ▶ "sink" is destination
- ▶ get a tree leading to the destination
- ▶ must be a tree: shortest path can only be composed of shortest paths



Communications Network Design: lecture 07 – p.39/44

Communications Network Design: lecture 07 – p.39/44

# Distance Vector

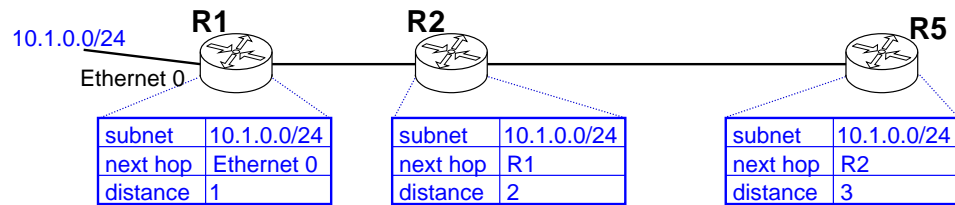
- ▶ also called Distributed Bellman-Ford
- ▶ proved converges for shortest path routing
  - ▷ ordering and timing of updates doesn't matter
- ▶ chief advantages
  - ▷ history (RIP invented way back in ARPANET)
  - ▷ simplicity
    - \* example of Cisco RIP configuration

```
router rip
network 10.1.0.0
```
- ▶ problems
  - ▷ convergence time (minutes)
  - ▷ scaling (of RIP)
  - ▷ count to infinity

Communications Network Design: lecture 07 – p.40/44

Communications Network Design: lecture 07 – p.40/44

# Count to infinity



Communications Network Design: lecture 07 – p.41/44

Communications Network Design: lecture 07 – p.41/44

# RIP

## Routing Information Protocol (RIP)

- ▶ RIP was first developed in early ARPANET
  - ▷ RIPv1, defined in RFC 1058 [3] (1988)
  - ▷ RIPv2, defined in RFC 1723 [4] (1994)
    - \* introduced classless routing (CIDR)
  - ▷ RIPng, defined in RFC 2080 (IPv6)
  - ▷ MDS authentication RFC 2082.
- ▶ implementation
  - ▷ uses UDP over IP, on port 520 to carry its data
    - \* see RFCs for packet formats
  - ▷ router transmits full updates every 30 seconds
    - \* by default

Communications Network Design: lecture 07 – p.42/44

Communications Network Design: lecture 07 – p.42/44

# RIP

---

- ▶ count-to-infinity mitigated using
  - ▷ split horizon with poison reverse
  - ▷ triggered updates
- ▶ count-to-infinity stopped
  - ▷ maximum distance = 15
  - ▷ infinity = 16
- ▶ problems
  - ▷ convergence is slow
  - ▷ count to 16 can still be slow
  - ▷ generates lots of traffic
  - ▷ maximum length path is 16

## References

---

- [1] J. Moy, "OSPF Version 2." IETF, Request for Comments: 2328, 1998.
- [2] D. Oran, "OSI IS-IS Intra-domain Routing Protocol." IETF, Request for Comments: 1142, 1990.
- [3] C. Hedrick, "Routing Information Protocol." IETF, Request for Comments: 1058, 1988.
- [4] G. Malkin, "RIP Version 2." IETF, Request for Comments: 1723, 1994.