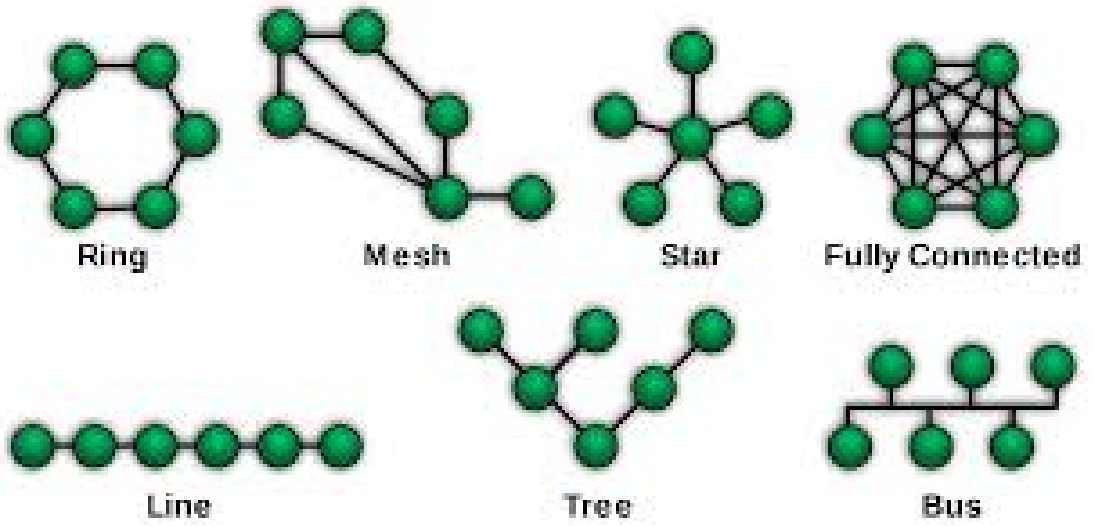
Routing in a single domain

IERG5090 – Jan 16, 2017

Motivation

- How are nodes connected:
  - By links and intermediate nodes, such as switches, routers etc
- Why are intermediate nodes needed?
- What are the advantages for different network topologies?
- What do intermediate nodes do to provide the connection?

Example network topologies



Outline

- We take a systems point of view
- Consider different types of scenarios, and discuss the suitable routing mechanism
- Give more details to those widely used in the internet:
  - Bridge spanning tree algorithm
  - Distance vector algorithm
  - Link state algorithm

Different type of networks

- Connectivity-challenged networks
  - Battlefield or post-disaster emergency networks
- Plug-and-play networks
  - Home, campus, SME, sensor networks
- Regular case
  - A regular ISP, e.g. Large campus, city level ISPs, cross university, hospital, government networks
- Performance critical networks
  - Data center, Google's server network, special purpose teleconference network, bank's or corporate network with sensitive data

Should we use the same kind of routing mechanism for all these scenarios?

Connectivity challenged networks

- Try brute-force methods to get connected and communicate
  - **Flooding**: send each message to all nodes in vicinity; each node tries to forward to other nodes (except up-streaming node); use scope (TTL) control.

  Broadcast LANs (e.g. Ethernet) reply partly on flooding
  - **On-demand routing**: in wireless settings, it is considered too costly to periodically exchange topological information among nodes; hence build routes (forwarding table) on demand

  Wireless Ad hoc networks, e.g. AODV protocol

- In these networks, there is no background process to set up routes

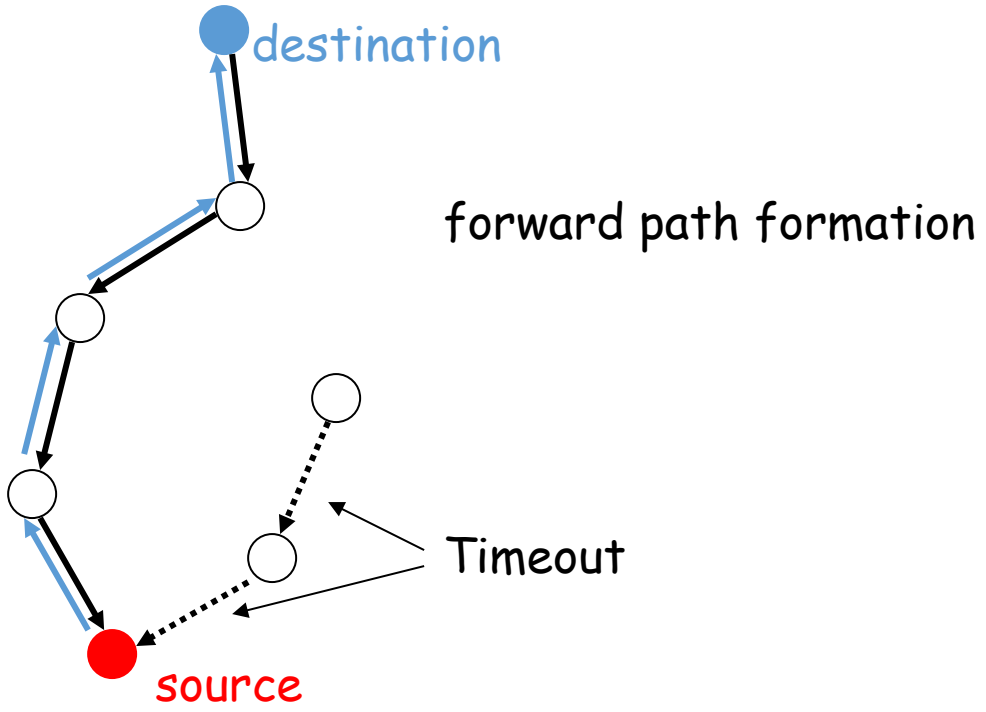
Flooding

- Advantage?
  - Simple
  - All paths explored
- Disadvantage?
  - “Disturbs” all nodes
  - Inefficient, uses all paths
  - Uncertain reliability
    - If sending multiple copies for each packet, overhead is high
    - Absolute reliability is hard (ack requires more flooding!

- Maybe okay for scenarios reliability and efficiency is not that important, and some amount of communication is desperately needed.

On-demand routing

- Path discovery
  - Source floods RReq, and get RRep back
- Routing table management
  - Routing table stores soft state, timed out when no activity
  - Reverse path not used will be timeout
  - Inactive forward path will be timed out with a different timer
- Path maintenance
  - If source moves, re-initiate path discovery
  - If intermediate or dest moves, re-send RRep
- Local connectivity management
  - If no data, periodically broadcast hellos (TTL=1)



Ad hoc wireless networks:
Very hot research for a few years, but few applications in practice

Plug-and-play networks

- Primary requirement
  - Self configuration, plug-and-play, as much as possible
  - Should be like a phone: plug it into the wall, it works
- Switch connected LANs
  - No need to configure IP addresses (until you need to be connected to rest of Internet)
  - Spanning tree based forwarding, self configured
  - “Switch” is used to be called a “Bridge”, used to connect LANs (nowadays, mostly Ethernets)
- The difference is:
  - The topology is stable
  - Nodes are not moving around
- The switched LAN technology is widely deployed, but
  - It is not part of the network layer of internet
  - It is done in the data link layer

What is a bridge (now called switch)

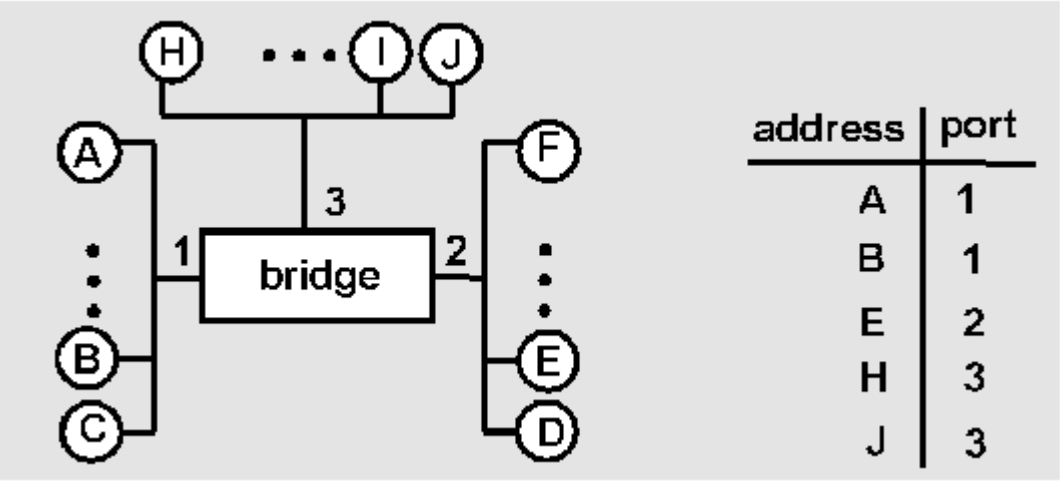
- Data Link level store-and-forward device that connects two or more LAN (Ethernet segments)
- Bridge isolates collision domains since it buffers packets
- Can connect LANs of different types

Note: Think of LAN as Ethernet, or something similar; all nodes hear each other.

- bridges *learn* which hosts can be reached through which interfaces: maintain filtering tables
  - when packet received, bridge "learns" location of sender: incoming LAN segment
  - records sender location in filtering table
- filtering table entry:
  - (Node LAN Address, Bridge Interface, Time Stamp)
  - stale entries in Filtering Table dropped (TTL can be 60 minutes)

Bridge filtering table example

Suppose C sends packet to D and D replies back with packet to C



C sends packet, bridge has no info about D, so floods to both LANs

- bridge notes that C is on port 1
- packet ignored on upper LAN
- frame received by D

D generates reply to C, sends

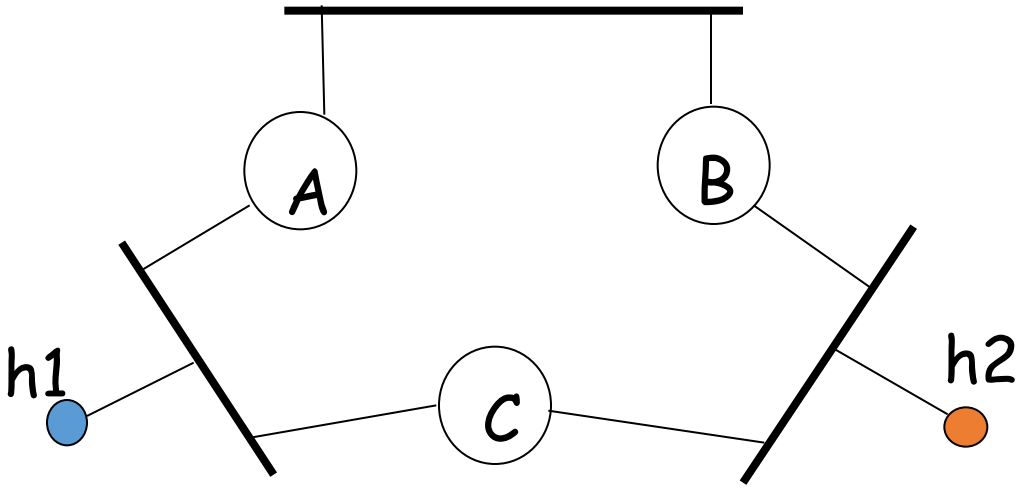
- bridge sees packet from D
- bridge notes that D is on interface 2
- bridge knows C on interface 1, so *selectively* forwards packet out via interface 1

Forwarding and filtering procedure

```
if destination is on LAN on which packet was received
    then drop the packet
    else { lookup filtering table
            if entry found for destination
             then forward the packet on interface indicated;
             else flood;    /* forward on all but the interface
                               on which the packet arrived*/
        }
```

One assumption

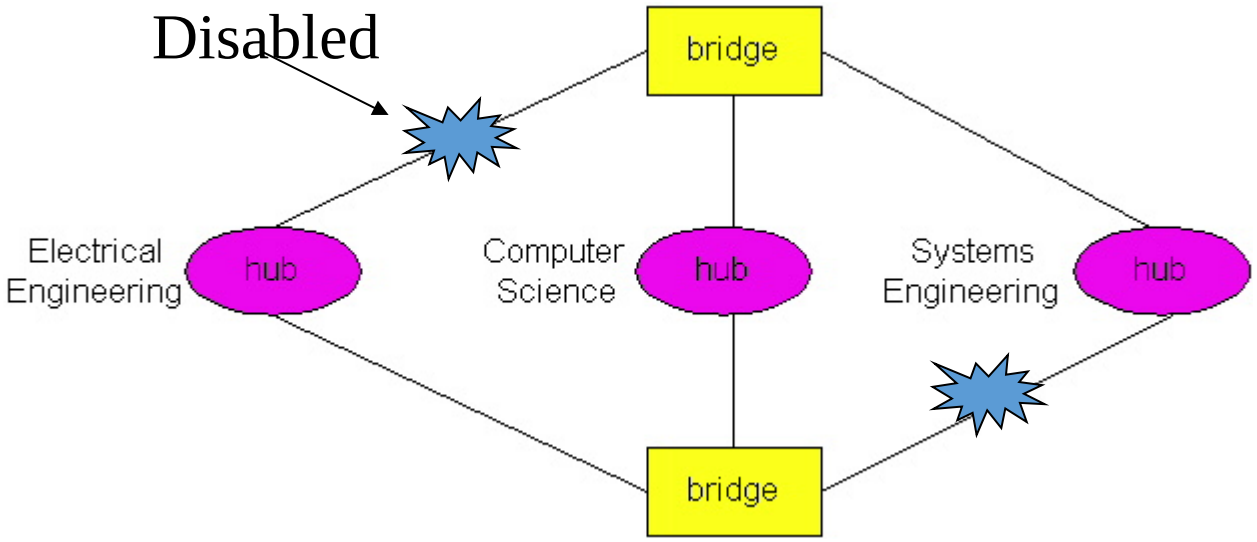
- The learning algorithm still works even when multiple bridges are connecting many LANs, provided the bridges and LANs do not form a loop!



- - When h1 sends a packet to h2, C thinks h1 is to its left
- - When B forwards h1's packet, C thinks h1 is to its right

Bridge spanning tree

- for increased reliability, desirable to have redundant, alternate paths from source to destination
- with multiple simultaneous paths, cycles result - bridges may multiply and forward packets forever
- solution: organize bridges in a spanning tree by disabling subset of interfaces



Spanning tree algorithm

- A protocol to decide which interfaces to disable, to ensure
  - complete connectivity
  - no loops
- Each bridge sends a configuration msg to a port unless a *better* one heard on that LAN
- Configuration message contains
  - Root id
  - Cost = #hops from Root
  - Transmitting bridge's id

Ranking of configuration msgs:

- If C1's root id is lower than C2's
- If root ids equal, then C1's cost is lower
- If root ids and costs same, then C1's transmitter's id is lower than C2's

Then C1 ranks higher than C2

A bridge's own configuration msg:

- Root id is its own id or the lowest heard so far
- Cost is the number of hops from root id (0 if self)

Example

These are configuration msgs heard on each port:

	Root	Cost	Transmitter id
Port 1	12	9	51
Port 2	12	8	47
Port 3	81	0	81
Port 4	15	3	27

The bridge's own id is 15

So its configuration msg is (12,9,15)

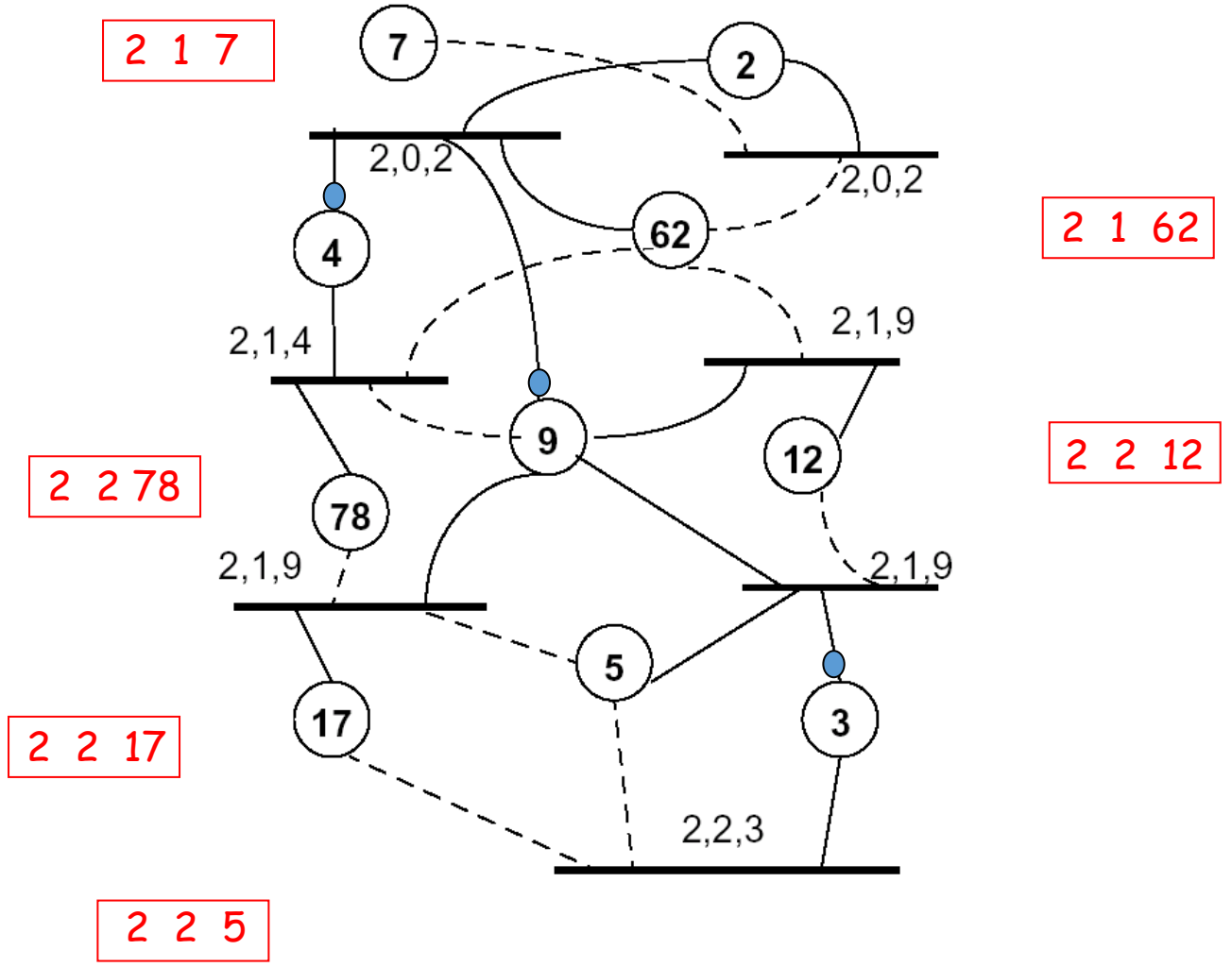
This is transmitted on port 1, 3, and 4

The bridge that wins for each LAN is the designated bridge for that LAN

Prove result is a tree:

- In steady state:
  - A single root is elected
  - Each bridge knows its shortest distance to the root
  - only one bridge transmits conf msgs on each LAN (the *designated* bridge) – this is the bridge that will forward packets from that LAN towards the root
  - Every bridge determines its port that gives its best path from itself to the root – that port is called a root port
- Only the root ports, and the ports on which "self" has been selected as designated bridge are enabled. All other ports are disabled.
- Result: Each designated bridge is a "parent" and the other bridges attached to the same LAN are its children.
- Since each bridge has only one root port, hence a tree

Example of a bigger spanning tree, and a poem



I think that I shall never see
A graph more lovely than a tree.

A tree whose crucial property
Is loop-free connectivity.

A tree which must be sure to span
So packets can reach every LAN.

First the Root must be selected
By ID it is elected.

Least cost paths from Root are traced
In the tree these paths are placed.

A mesh is made by folks like me.
Then bridges find a spanning tree.

Radia Perlman

inventor of the spanning tree alg

Host configuration

- Can it be completely plug-and-play?

What does a host need in order to operate?

- IP address (plus network mask)
- Local DNS server
- Router addresses
- Domain name (specially if web server)
  - name to address mapping in DNS

Anything else?

- Many other things, e.g. application settings, security features…
- Most can operate based on default settings

Auto-configuration

Basic idea: ask the network

- RARP: given one's MAC address, return IP (reverse of ARP, which returns MAC given IP)
- Dynamic Host Configuration Protocol (DHCP)
  - DHCP does not use fixed mapping, instead "leases" IP addresses
  - Configure local name server, router list as well
  - Specially useful for mobile hosts

  -- Plug-and-play, zeroconfig (IETF WG)

- DHCP server returns its own address for future use by client
- Given leased IP address, client may test it using ARP
- Upon expiry, the client can "renew" the lease

Recap of “Plug-and-play”

- Plug-and-play is important
- We reviewed auto-configuration for switched LAN, and end hosts:
  - The spanning tree algorithm
  - DHCP (browse RFC1531 for details)
- Read [Perlman 11.2] for review

The regular ISP

- Spanning tree is not good enough
  - Why?
- The basic routing problem
  - Discover the topology of links and nodes
  - Compute a forwarding table for each (intermediate) node
- Internet's approach – distributed routing protocol
  - Compute shortest path for each destination, and use that result for forwarding table
  - Why shortest path is reasonable?
  - Is it the only choice?

Other requirements:

- Robust
  - Self-stabilize
  - No loops, black holes
  - Fault tolerant, but avoid oscillation
- Scalable
  - Manageable forwarding table size, even for large nets
- Efficient
  - Minimize number/frequency of control messages
- Performance
  - Shortest paths
  - Load balanced

Many approaches

- Centralized vs distributed
  - Centralized simpler, but requires more admin, is harder to scale, and less robust
- Source-based vs router-based
  - Source-based means routing done by the source; each packet carries its path in the header
- Single vs multiple path
  - Multipath can help deal with congestion
- State-dependent vs state-independent
  - Compute routes based on current network load (e.g. delay)
- Periodic versus On-demand
  - On-demand proposed for wireless networks

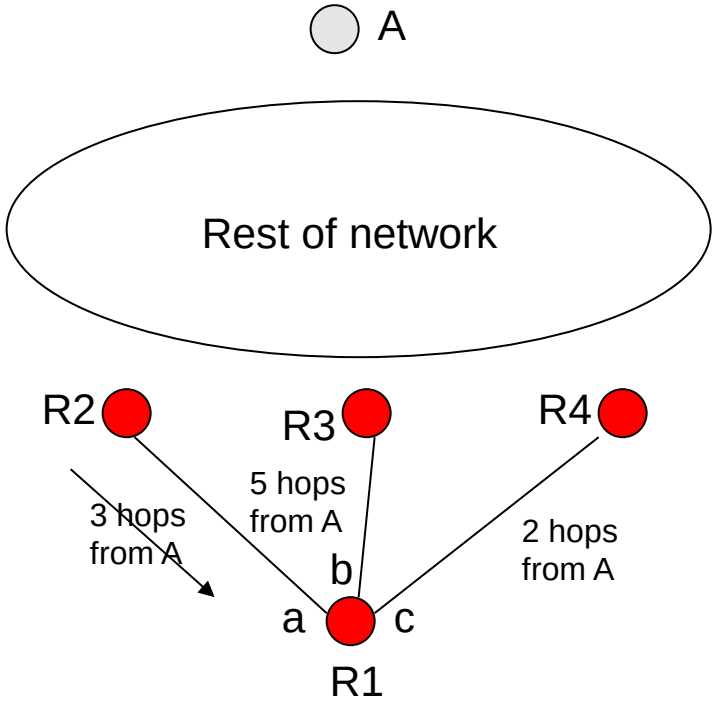
- Internet picked distributed routing, and router-based, single-path, state-independent, and periodic

- Two distributed algorithms:
  - Distance Vector
  - Link State

- Congestion control is not the responsibility of routing

Distance vector routing, by example

- Each router listens for neighbors' routing messages
- In this example, R1 has 3 neighbors. For destination A, the path advertised by R4 is the shortest
- R1 put that information into its forwarding table: for A, forward via interface c, router R4, which costs 3 hops.
- All routers do the same, for all destinations.

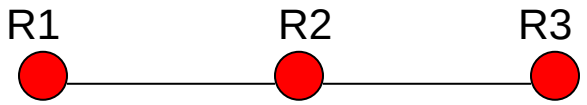


Forwarding table entry

A	c	R4	3

The “count-to-infinity” problem

- The advantage with distance vector is simplicity and memory efficiency
- One major disadvantage is slow convergence, and the count to infinity problem.
- If the link between R1 and R2 goes down, R2 and R3 keep thinking R1 reachable via each other



Count to infinity example

- R3->R2: I can reach R1 in 2 hops
- R2->R3: I can reach R1 in 3 hops
- R3->R2: I can reach R1 in 4 hops
- …
- R2->R3: I can reach R1 in n hops

- When n = infinity, then R2 and R3 know R1 is not reachable

Solutions for "count-to-infinity"

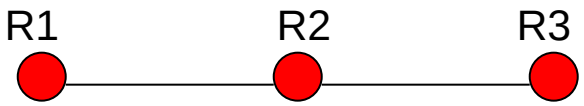
Split horizon

- R3's best route to R1 is via R2, so it tells R2 its distance to R1 is infinity
- Doesn't always work: e.g. R4 and R5 keep thinking D is reachable

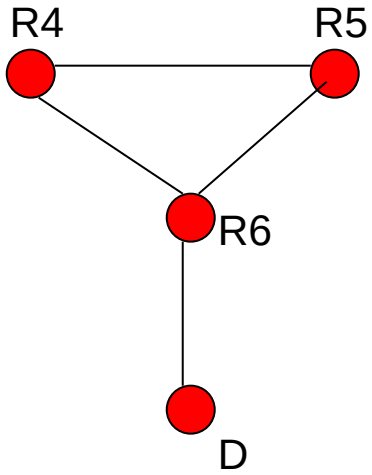
Triggered update

- Normally neighbors exchange updates every n (=30) seconds
- When a link is down, send updates that triggers immediate updates
- Count to infinity faster

These two implemented in RIP



Count to infinity example



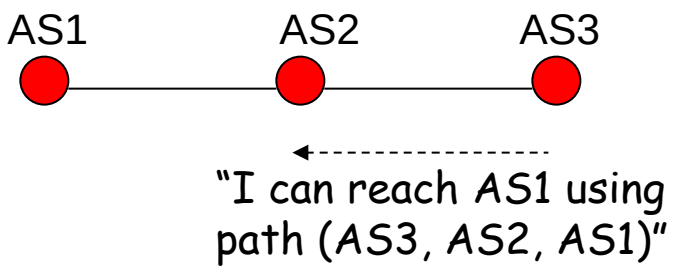
Split horizon fails here

Another solution to "count-to-infinity"

Path vector

- Include path information as well as distance in distance vector
- Can exclude path with loops, or going through self

This is adopted in BGP



AS is a network, rather than a node

The basic steps of Link State Routing

Each router is responsible for

1. "meeting" its neighbor (end nodes) and learning their Ids
2. constructing a link state packet (LSP), containing
   - Id
   - Cost

   for each neighbor (end node)
3. transmitting the LSP to all other routers
   - Each router stores the most recently generated LSP from all other routers
4. computing the shortest path for each destination and fill forwarding table
5. Periodically, or when link state changes, go back to step 2

- We will discuss the colored parts

Disseminating LSPs

LSP = Link State Packet

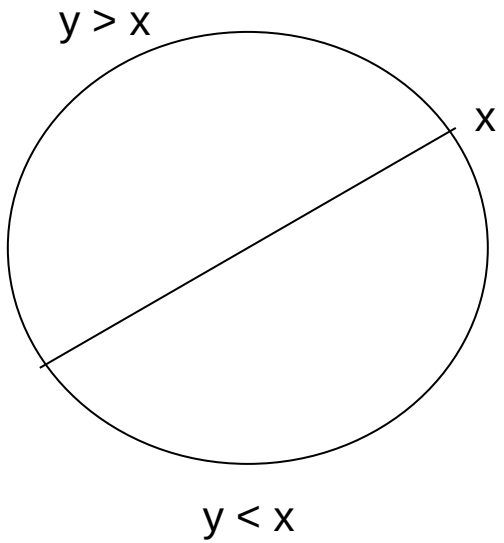
Basic requirement:

- for correct route computation, each LSP must reach all routers

- Idea 1: Use current routing tables - Send my LSPs to each router one-by-one as normal data
  - Routing tables derived from LSPs: chicken-and-egg
- Idea 2: Flooding - Each router sends/forwards an LSP to all neighbors except "upstream" neighbor
  - How to stop generating infinite number of off-springs?
- Idea 3: Intelligent flooding - If the LSP received is the same as locally stored version, don't forward
  - But most recently received may not be most recently generated, i.e. we need to do "in order, intelligent flooding"

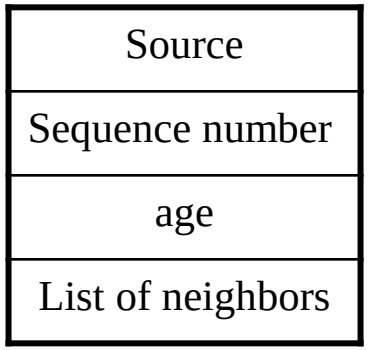
Disseminating LSPs (cont)

- Idea 4: Use timestamp in LSPs
  - An error in using a future timestamp can cause problems
  - Need synchronized clocks to work
- Idea 5: Use Sequence number in LSPs
  - Sequence number wraps around. Given any x, half sequence numbers greater than x, half smaller than x.
  - Sequence number alone not enough
    - Network may be partitioned (sequence number get out of synch)
    - Router may crash, and starts with sequence number 0 after recovery, old sequence number not meaningful



Disseminating LSPs (cont)

- Idea 6: Sequence number plus age
  - (Each LSP times out after maximum age)
  - When LSP is generated, age = MaxAge
  - When LSP received, it overwrites current LSP if seq number of new one is larger; and it is further propagated;
  - Age is decremented as LSP sits in memory
    - LSP with zero age is not propagated
    - So a received LSP always has nonzero age
  - If stored LSP has zero age, a received LSP is always accepted regardless of seq number



Idea 6 was implemented in the original ARPAnet

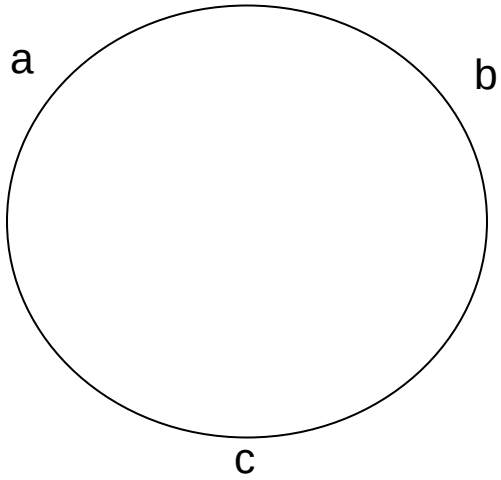
- Age is 3-bit field, unit is 8 seconds
- Max-age = 56 seconds
- New LSP generated every 60 seconds
- A rebooted router must wait 90 seconds before sending out its new LSP – this is to let its old LSP age to zero

The ARPAnet “incident”

- One day, ARPAnet stopped working!
- Difficult to remotely diagnose when routing is not working
- Rebooting a router, still broken!
- Each router’s queue was full of LSP packets, all from same source S, with 3 sequence numbers:

  a < b < c < a

  with approx the same timestamp



- It is in a state it cannot get out of!
  - The LSPs keep getting forwarded w/o adjustment to age field
- They did not want to stop all routers
  - Made a patch to ignore all LSPs from faulty source
  - Fixed routers one by one
- True story!

The lesson

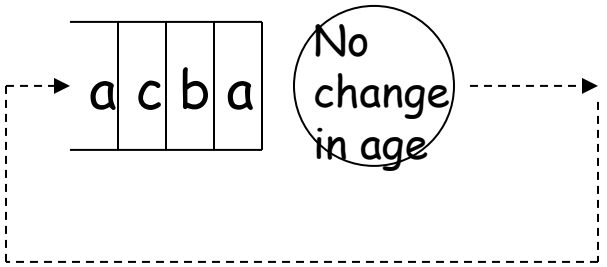
- Designing distributed algorithms is tricky
- It needs to be self-stabilizing
  - No matter how corrupted the databases become, after faulty or malicious component is removed, system should return to normalcy in reasonable amount of time.
- Need to be efficient
  - New LSP every 60 seconds is too frequent
- Need to be responsive as well
  - Should not make a rebooted router to wait new 90 seconds before sending out its new LSP

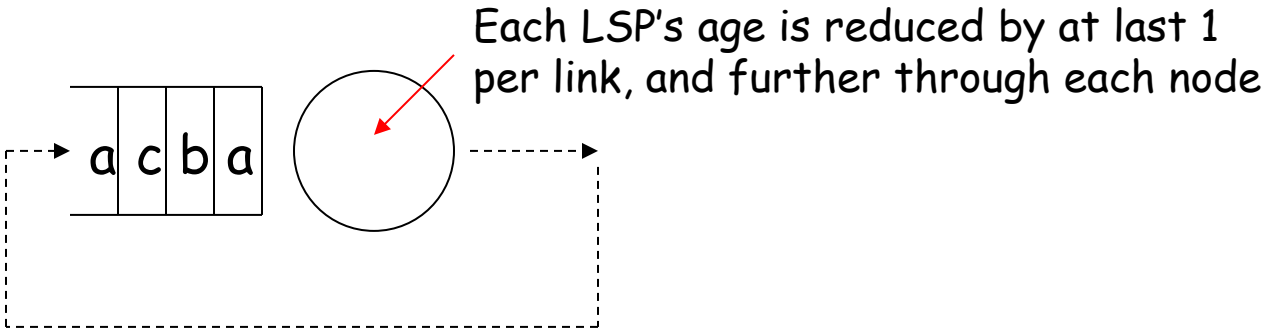
Back to drawing board

The new LSP flooding algorithm

- Stop the sequence number from wrapping
  - When seq number from a router S reaches maximum value, no new LSP from S is accepted till age of old one reaches 0
  - Not responsive at largest seq number, but this happens rarely since sequence number space large (32 bits)
  - Since sequence number no longer wraps, no need for rebooted node to wait 90 seconds
- Max age is about 1 hour (improves efficiency)
  - Age is decremented (by at least 1) as LSP is forwarded
- LSPs are acknowledged
  - LSP is persistently forwarded until acked
  - Ack assume received
- Self-stabilizing property proved

Consider the failure case before:





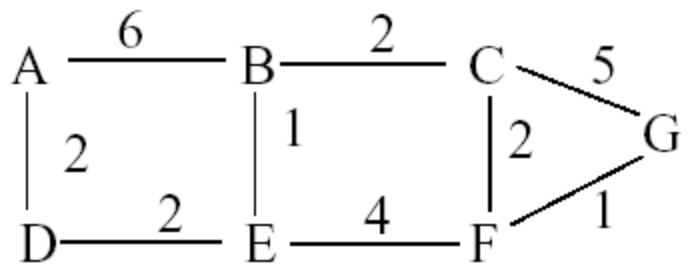
Coordinating LSPs and ACKs from multiple neighbors

- The sending of LSPs and Acks is not based on single queue
- Each LSP has k flags, k = number of router's interfaces
- When LSP generated, a "send" flag set for each interface
- When LSP received, store it, an "ack" flag set for interface received from, and "send" flag for rest of the interfaces
- After sending LSP, "send" stays till receiving ack, "send" reset to "ok";
- After sending ack, "ack" reset to "ok"

Src	nbr1	nbr2	nbr3	nbr4	nbr5
A	ok	ack	ok	send	ack
B	ok	ok	ok	ok	ok
C	ack	send	send	send	send
D	ok	send	ok	ok	ok
E	send	ok	ack	ok	ok

- Each flag has three possible states: "send", "ack", "ok"
- This table visited periodically (retransmit timer)
- LSPs are aged, and deleted when (age=0) and (all flags = "ok")

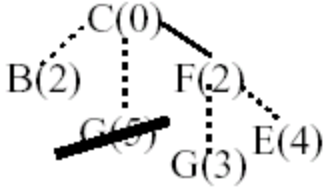
Computing the shortest path: Dijkstra's algorithm explained by example

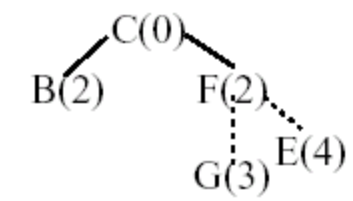


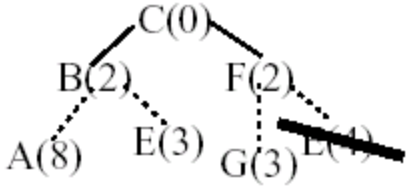
LSPs:

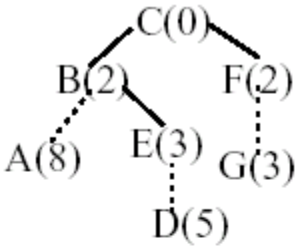
A	B	C	D	E	F	G
B/6	A/6	B/2	A/2	B/1	C/2	C/5
D/2	C/2	F/2	E/2	D/2	E/4	F/1
	E/1	G/5		F/4	G/1	

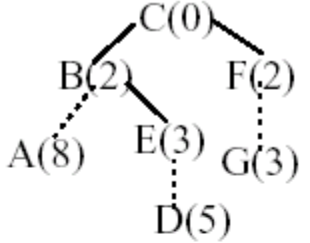
Compute routes at C:

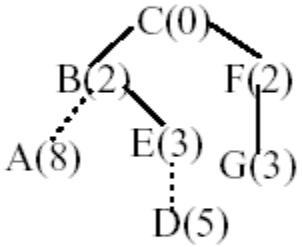


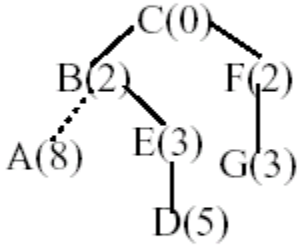


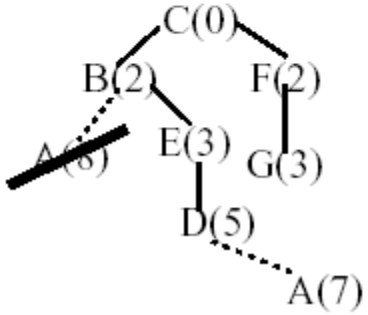












Pseudo code of Dijkstra's algorithm

```
dist[s] ←0                          (distance to source vertex is zero)
for all v ∈ V–{s}
   do dist[v] ←∞                    (set all other distances to infinity)
S←∅                                 (S, the set of visited vertices is initially empty)
Q←V                                 (Q, the queue initially contains all vertices)
while Q ≠∅                          (while the queue is not empty)
do  u ← mindistance(Q,dist)         (select the element of Q with the min. distance)
   S←S∪{u}                          (add u to list of visited vertices)
   for all v ∈ neighbors[u]
      do if dist[v] > dist[u] + w(u, v)   (if new shortest path found)
         then d[v] ←d[u] + w(u, v)        (set new value of shortest path)
return dist
```

Recap of Link State routing algorithm

- When a link state is changed (link down, or up), the information is quickly put into a LSP and disseminated to all routers
- Each router has (same) up-to-date "map" of the network
- Each router will compute new shortest paths using Dijkstra's algorithm
- This results in fast reaction to network changes

- In internet, there are two different standards for Link State Routing:
  - IS-IS protocol (IS = Intermediate System; ES = End System)
  - OSPF protocol

- IS-IS was first developed as part of the OSI protocols; it was brought to IETF for standardization

Comparison of DV and LS routing

- Memory (assume each router has k neighbors)
  - DV: in worse case, each DV is O(n) => O(k*n)
  - LS: each router keeps n LSPs, each LSP is O(k), so also O(k*n)
  - But due to address aggregation each DV << O(n)
- Bandwidth used by control messages
  - For LS, each LSP will always traverse each link once
  - For DV, some changes may not affect distances (hence not propagated much); other changes (e.g. count-to-infinity situation) traverse each link more than once
  - Hard to compare
- Computation
  - #links ~ O(n*k), #nodes = n
  - Dijkstra's algorithm takes O(links*log(nodes))=O(n*k*logn)
  - Comparing k distance vectors (each O(n)) takes O(n*k)
  - Further savings if each DV << O(n)
  - But the algorithm may require several passes

It is widely agreed

- DV has an edge in memory, computation and simplicity
- LS converges much faster, which is probably more important IF you can afford the resources

Traffic engineering

- What kind of performance problems ISPs often face?
- What can ISPs do?

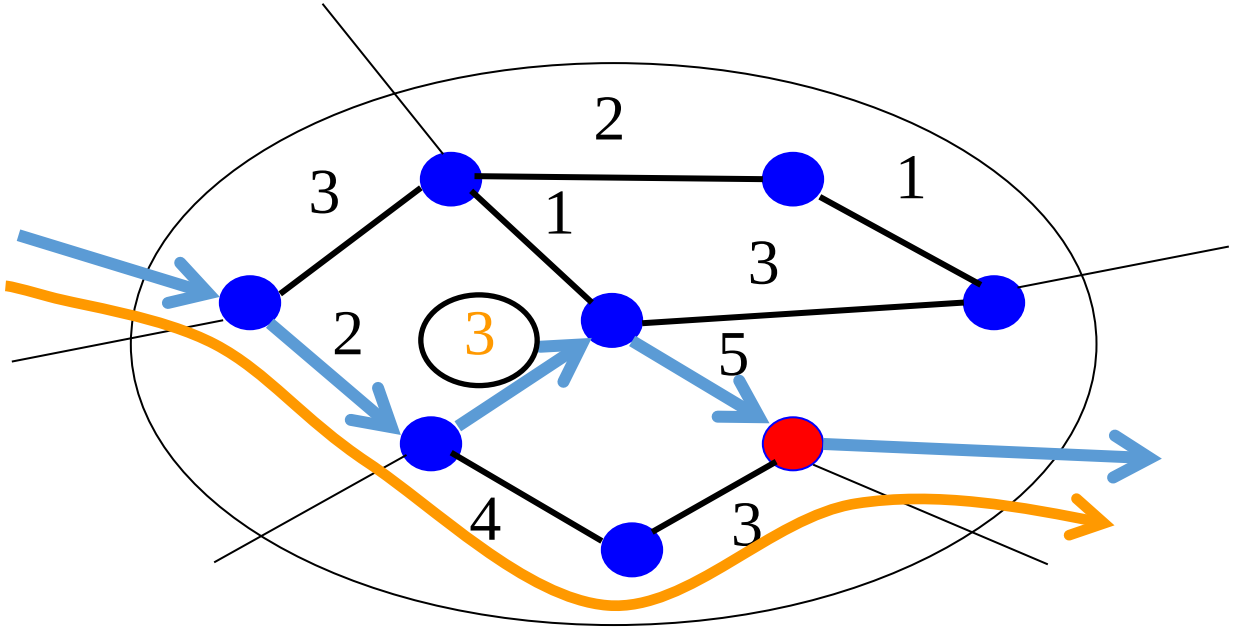
- Load balancing
- Provide some performance guarantee for some applications

Since routing decisions are made by the routing algorithms, the ISPs can only try to change the link weights to influence the decisions

- This is called “traffic engineering”

Traffic engineering example

- Try to balance load on links by adjusting weights
- Originally, weights are “static”, representing the delay, or cost of a link
- Traffic engineering inverts the problem, by setting link weights to achieve balanced load, only performed by large ISPs
- Some challenges:
  - Computation complexity
  - Load is not static



- By decreasing the weight of indicated link, some traffic can be moved there

Traffic engineering as an optimization problem

Traffic engineering can be formulated as an optimization problem:

Input:

- Topology:
  - Connectivity – a graph G=(V,E)
  - Capacity – for each edge (i,j), a capacity c(i,j)
- Traffic matrix: offered load between nodes in the network
  - Demand = d(i,j)
- Cost function: for each link of capacity c and demand d
  - Cost = f(c,d)

Problem: setting weights to minimize total cost

Limitations:

- In practice, traffic is rarely stationary, but changing all the time
- Hence traffic engineering in practice is usually ad hoc

Later, we will discuss Software Defined Networks, and how traffic engineering can be done more directly

In Inter-domain (inter-ISP) routing, there are other techniques for traffic engineering

Summary

- We reviewed different needs for different network settings
- We explained three mostly commonly used distributed routing algorithms used in Internet:
  - Spanning tree routing
  - Distance vector routing
  - Link State routing
- We explained the challenges in designing these distributed algorithms

Reading:

Radia Perlman,

Interconnections: Bridges, Routers, Switches, and Internetworking Protocols (2nd Edition) 2nd Edition