Tutorial on Network Layers 2 and 3

Radia Perlman
Intel Labs
(radia@alum.mit.edu)

Why?

- Demystify this portion of networking, so people don’t drown in the alphabet soup
- Think about these things critically
- N-party protocols are “the most interesting”
- Lots of issues are common to other layers
- You can’t design layer n without understanding layers n-1 and n+1

What can we do in 1 ½ hours?

- Understand the concepts
- Understand various approaches, and tradeoffs, and where to go to learn more
- A little of the history: without this, it's hard to really "grok" why things are the way they are

Outline

- layer 2 issues: addresses, multiplexing, bridges, spanning tree algorithm
- layer 3: addresses, neighbor discovery, connectionless vs connection-oriented
  - Routing protocols
    - Distance vector
    - Link state
    - Path vector
- Layer 2 ½ ... as if 2 vs 3 weren't confusing enough

Why this whole layer 2/3 thing?

- Myth: bridges/switches simpler devices, designed before routers
- OSI Layers
  - 1: physical

Why this whole layer 2/3 thing?

- Myth: bridges/switches simpler devices, designed before routers
- OSI Layers
  - 1: physical
  - 2: data link (nbr-nbr, e.g., Ethernet)

Why this whole layer 2/3 thing?

- Myth: bridges/switches simpler devices, designed before routers
- OSI Layers
  - 1: physical
  - 2: data link (nbr-nbr, e.g., Ethernet)
  - 3: network (create entire path, e.g., IP)

Why this whole layer 2/3 thing?

- Myth: bridges/switches simpler devices, designed before routers
- OSI Layers
  - 1: physical
  - 2: data link (nbr-nbr, e.g., Ethernet)
  - 3: network (create entire path, e.g., IP)
  - 4 end-to-end (e.g., TCP, UDP)

Why this whole layer 2/3 thing?

- Myth: bridges/switches simpler devices, designed before routers
- OSI Layers
  - 1: physical
  - 2: data link (nbr-nbr, e.g., Ethernet)
  - 3: network (create entire path, e.g., IP)
  - 4 end-to-end (e.g., TCP, UDP)
  - 5 and above: boring

Definitions

- Repeater: layer 1 relay

Definitions

- Repeater: layer 1 relay
- Bridge: layer 2 relay

Definitions

- Repeater: layer 1 relay
- Bridge: layer 2 relay
- Router: layer 3 relay

Definitions

- Repeater: layer 1 relay
- Bridge: layer 2 relay
- Router: layer 3 relay
- OK: What is layer 2 vs layer 3?

Definitions

- Repeater: layer 1 relay
- Bridge: layer 2 relay
- Router: layer 3 relay
- OK: What is layer 2 vs layer 3?
  - The "right" definition: layer 2 is neighbor-neighbor. "Relays" should only be in layer 3!

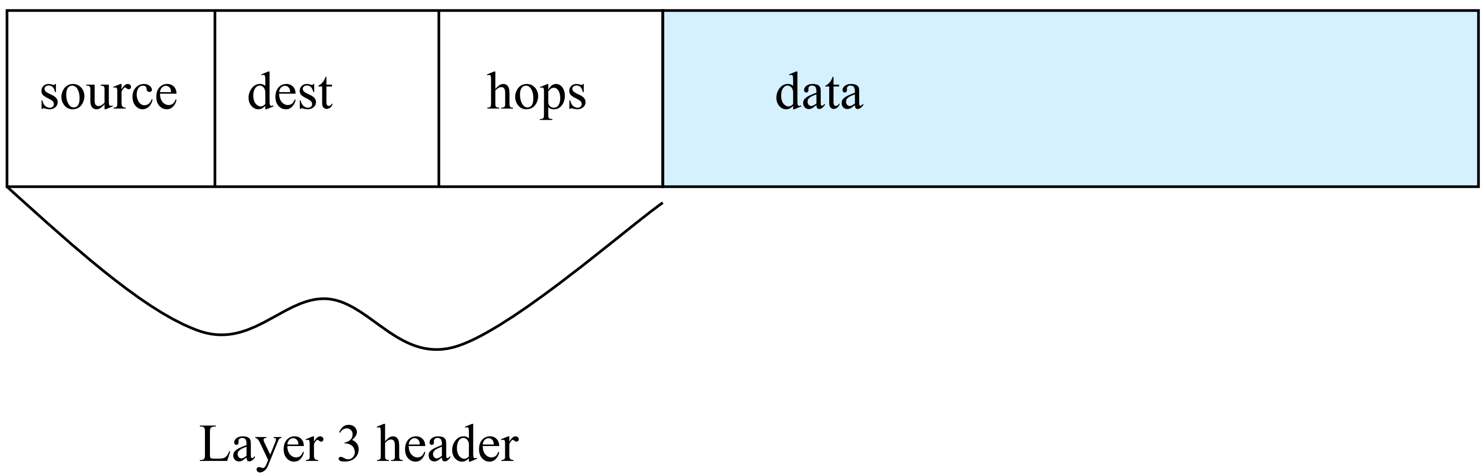
Definitions

- Repeater: layer 1 relay
- Bridge: layer 2 relay
- Router: layer 3 relay
- OK: What is layer 2 vs layer 3?
- True definition of a layer n protocol: *Anything designed by a committee whose charter is to design a layer n protocol*

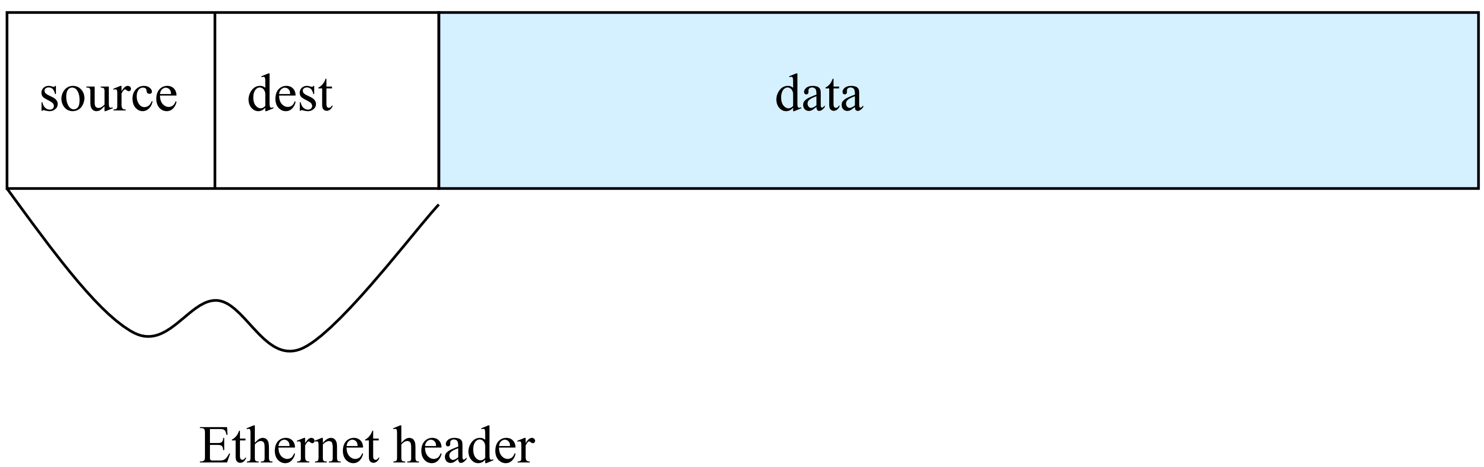
Layer 3 (e.g., IPv4, IPv6, DECnet, Appletalk, IPX, etc.)

- Put source, destination, hop count on packet
- Addresses are assigned so that a bunch of addresses can be summarized with a prefix
- Just like postal addresses:
  - Country
  - State
  - City

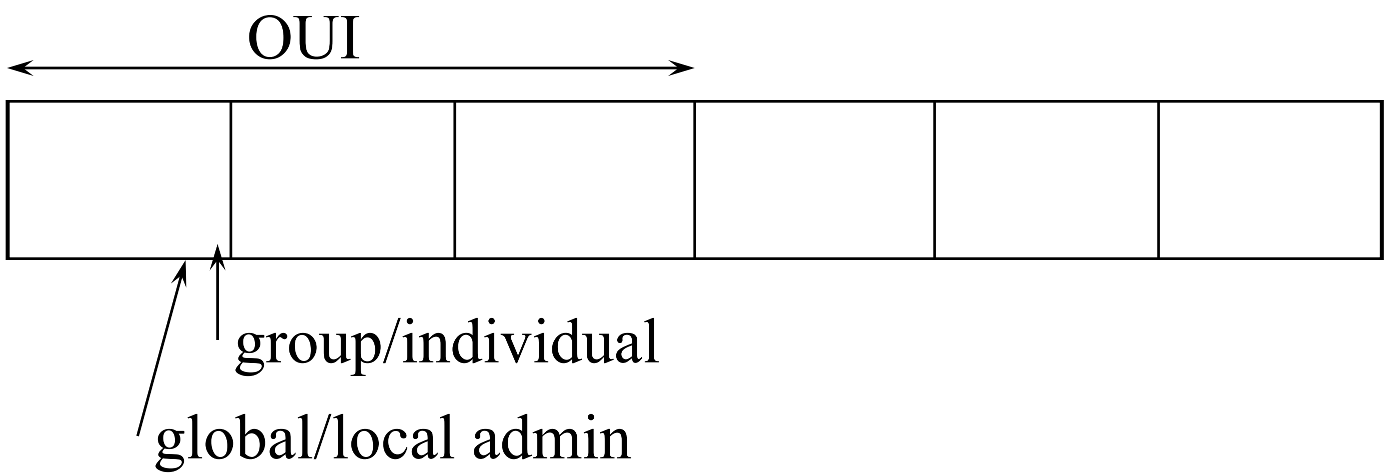
Layer 3 packet



Ethernet packet



Ethernet (802) addresses



- Assigned in blocks of 2^{24}
- Given 23-bit constant (OUI) plus g/i bit
- all 1's intended to mean "broadcast"

Ethernet addresses are "flat"

- Which means that Ethernet addresses have nothing to do with where a device is
- It looks like there is structure there, but the whole point of the OUI is to assign a fixed address at time of manufacture

Ethernet “religion” was autoconfiguration

- Assigning addresses are the manufacturer’s problem
- Then the customer just plugs things together and they work

It’s easy to confuse “Ethernet” with “network”

- Both are multiaccess clouds
- But Ethernet does not scale. It can’t replace IP as the Internet Protocol
  - Flat addresses
  - No hop count
  - Missing additional protocols (such as neighbor discovery)
  - Perhaps missing features (such as fragmentation, error messages, congestion feedback)

Original Ethernet Design

- CSMA/CD
  - CS: “carrier sense” listen before talking so you don’t interrupt
  - MA: “multiple access” shared medium
  - CD: “collision detect” listen even while talking in case someone else started talking at “the same time”
  - Do exponential random backoff if collision

CSMA/CD pretty much dead

- So what is Ethernet today?

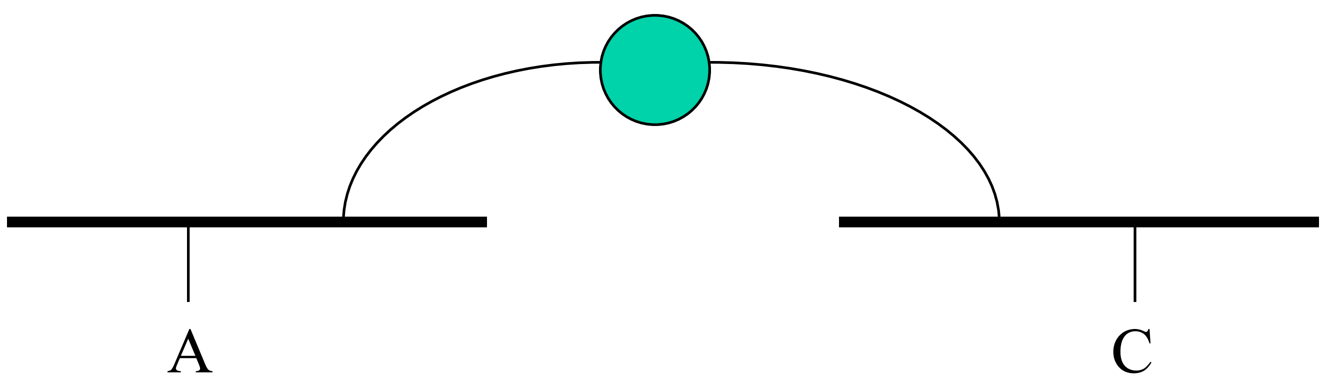
So where did bridges come from?

So where did bridges come from?

- Early 1980's…Ethernet new and highly hyped
- People thought it was "the new way to do networking"
- People built applications directly on Ethernet (leaving out layer 3)

Problem Statement

Need something that will sit between two Ethernets, and let a station on one Ethernet talk to another



Basic idea

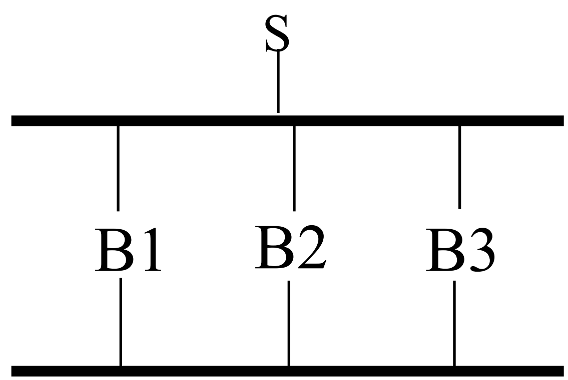
- Listen promiscuously
- Learn location of source address based on source address in packet and port from which packet received
- Forward based on learned location of destination

What’s different between this and a repeater?

- no collisions
- with learning, can use more aggregate bandwidth than on any one link
- no artifacts of LAN technology (# of stations in ring, distance of CSMA/CD)

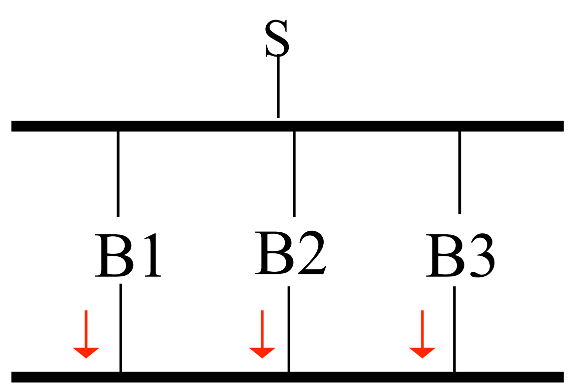
But loops are a disaster

- No hop count
- Exponential proliferation



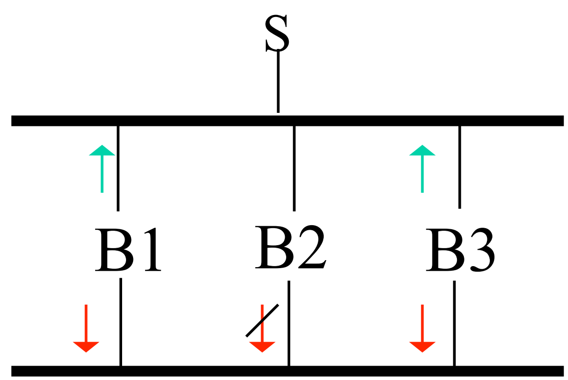
But loops are a disaster

- No hop count
- Exponential proliferation



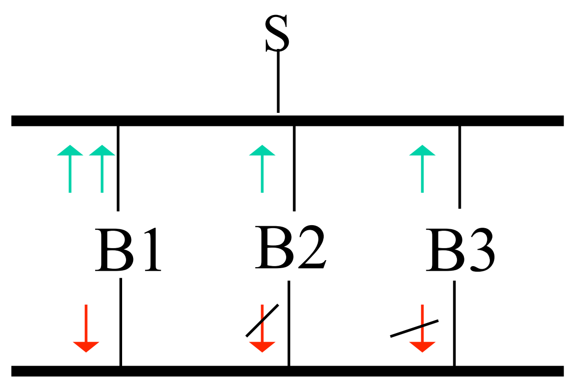
But loops are a disaster

- No hop count
- Exponential proliferation



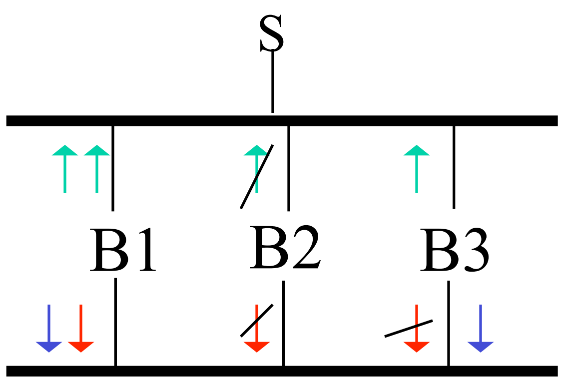
But loops are a disaster

- No hop count
- Exponential proliferation



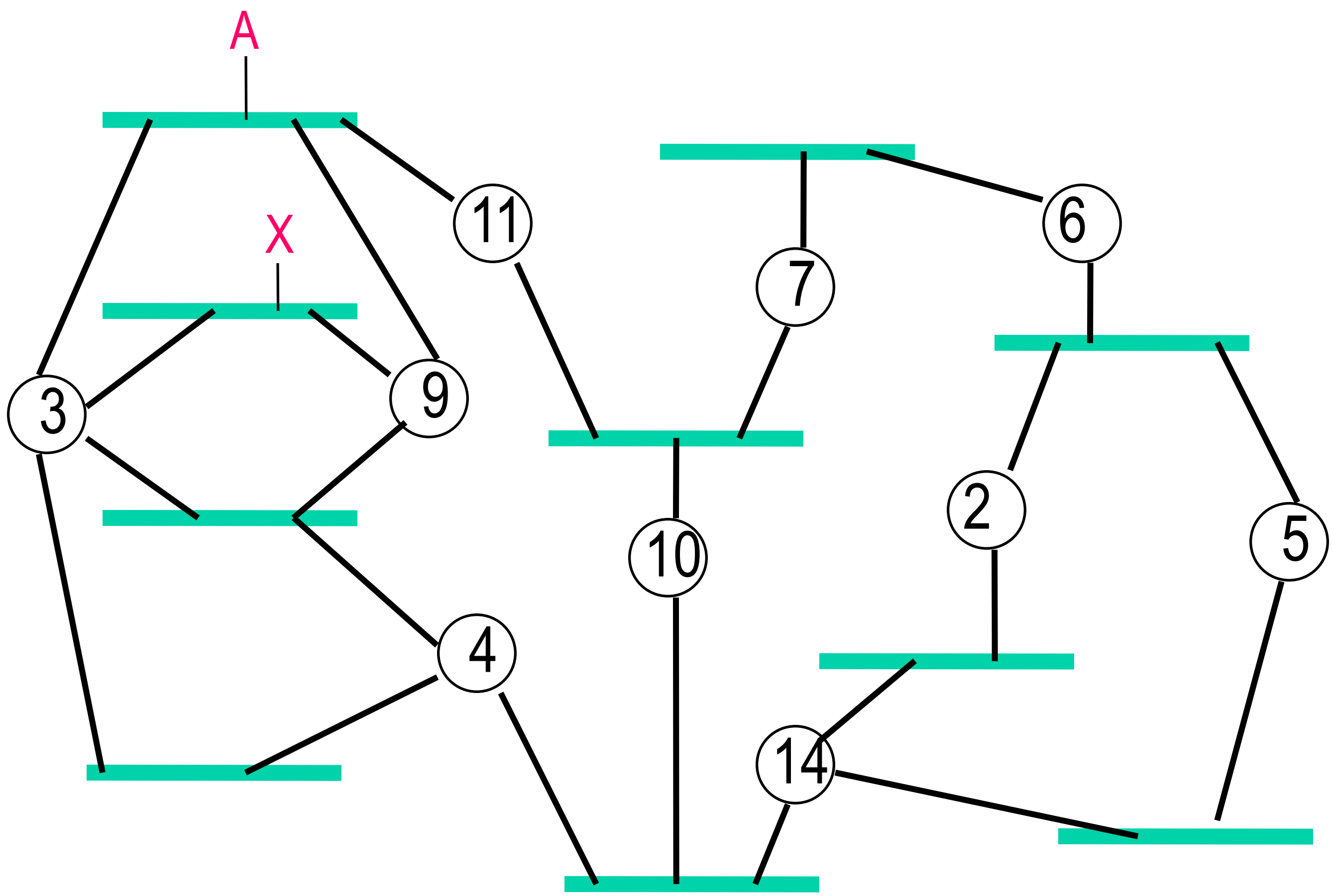
But loops are a disaster

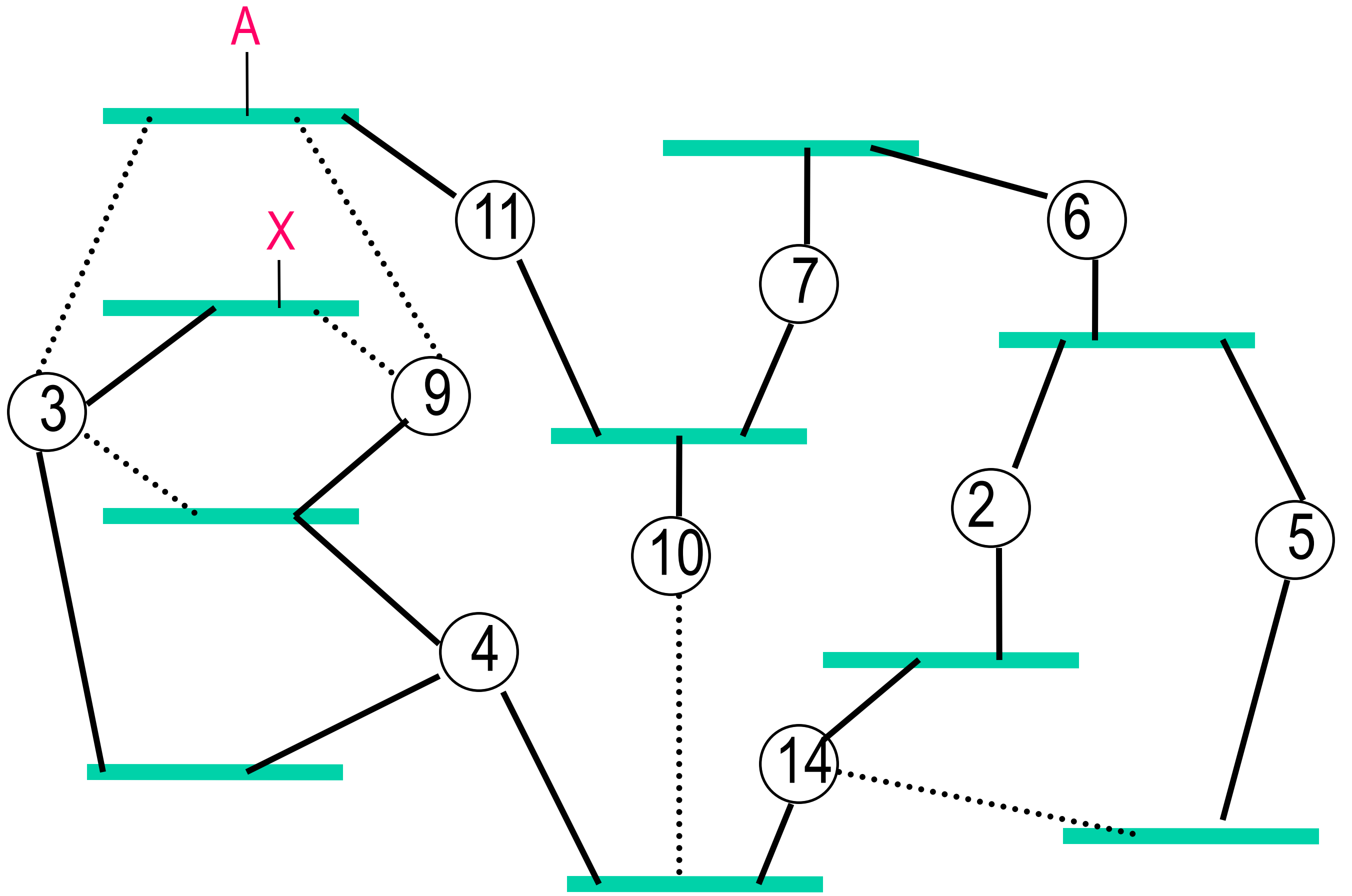
- No hop count
- Exponential proliferation



What to do about loops?

- Just say “don’t do that”
- Or, spanning tree algorithm
  - Bridges gossip amongst themselves
  - Compute loop-free subset
  - Forward data on the spanning tree
  - Other links are backups


A
X
11
7
6
3
9
10
2
5
4
14


A
X
11
7
6
3
9
10
2
5
4
14

Algorhyme

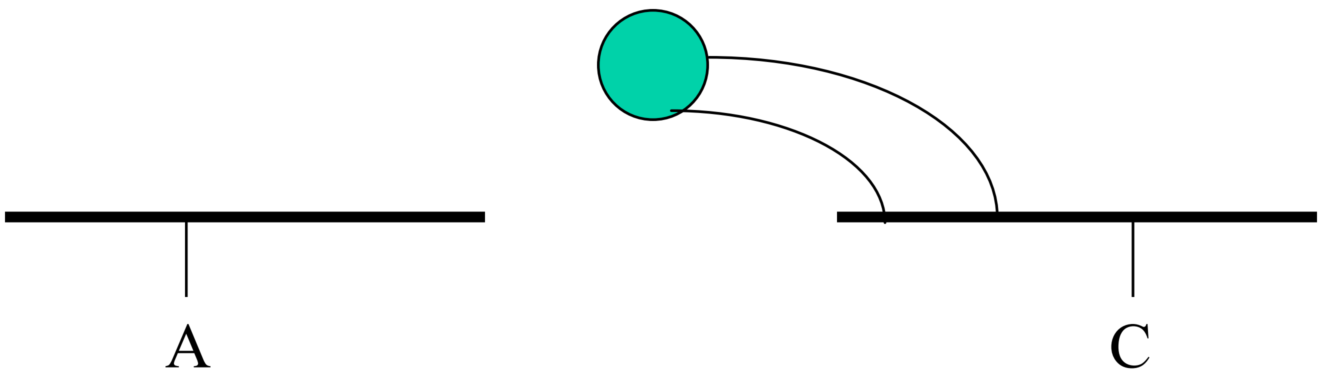
I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.
A tree which must be sure to span
So packets can reach every LAN.
First the Root must be selected
By ID it is elected.
Least cost paths from Root are traced
In the tree these paths are placed.
A mesh is made by folks like me.
Then bridges find a spanning tree.

Radia Perlman

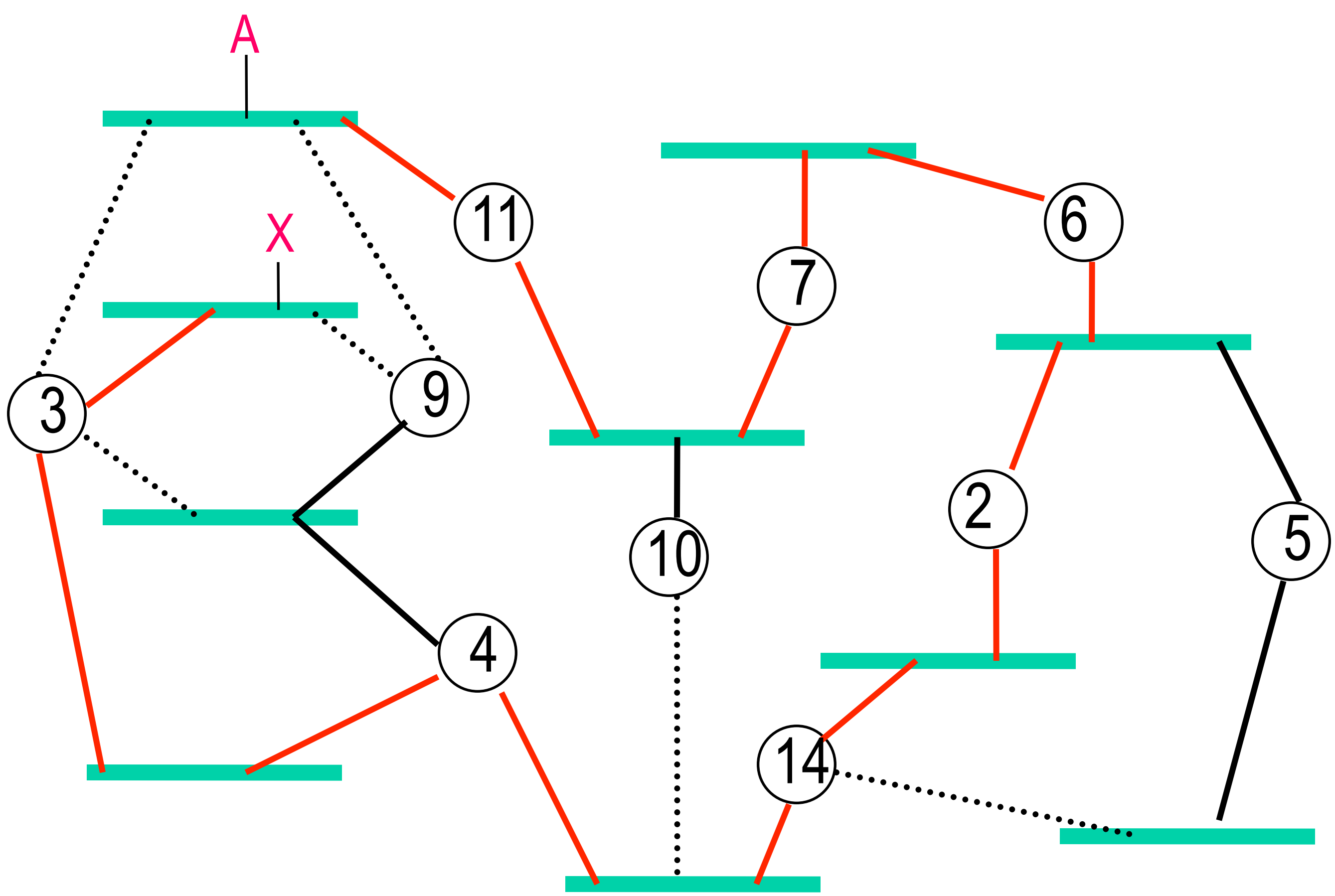
Bother with spanning tree?

- Maybe just tell customers “don’t do loops”
- First bridge sold...

First Bridge Sold



Suboptimal routes



So Bridges were a kludge, digging out of a bad decision

- Why are they so popular?
  - plug and play
  - simplicity
  - high performance
- Will they go away?
  - because of idiosyncracy of IP, need it for lower layer.

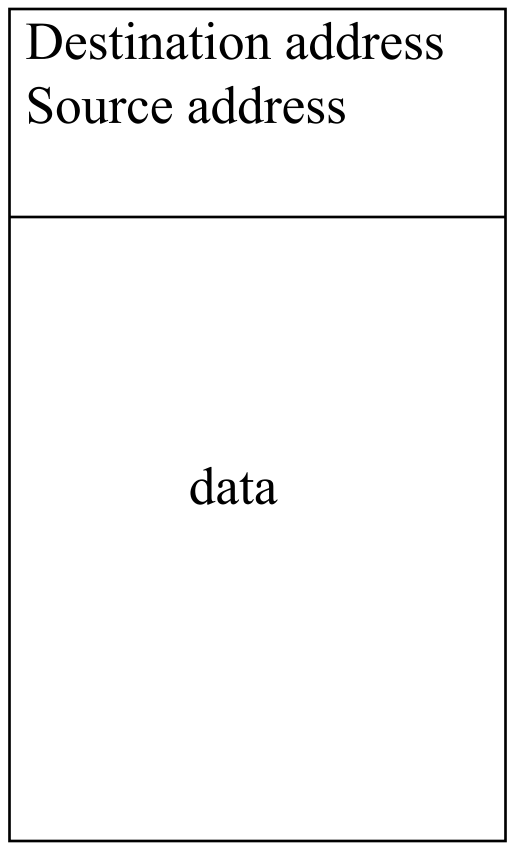
Note some things about bridges

- Certainly don’t get optimal source/ destination paths
- Temporary loops are a disaster
  - No hop count
  - Exponential proliferation
- But they are wonderfully plug-and-play

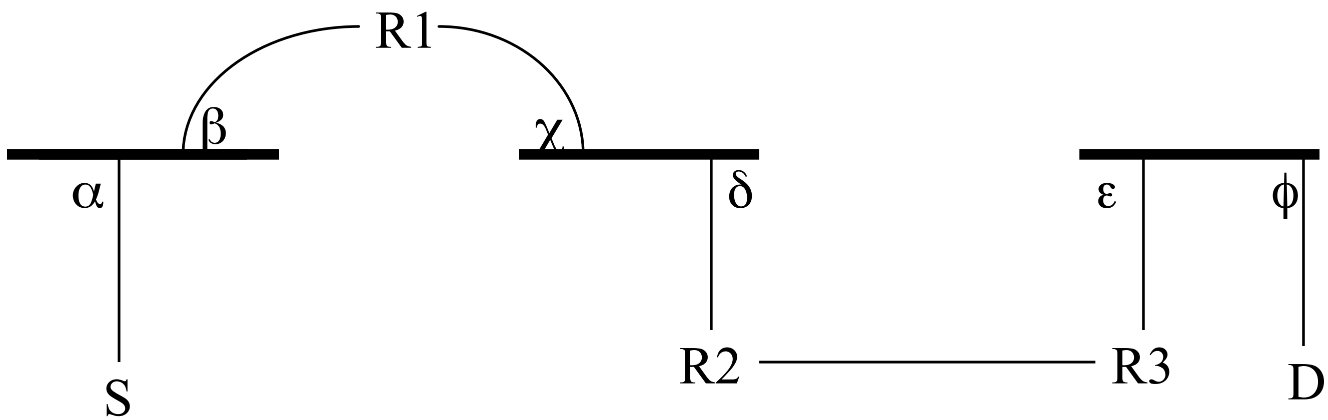
Switches

- Ethernet used to be bus
- Easier to wire, more robust if star (one huge multiport repeater with pt-to-pt links
- If store and forward rather than repeater, and with learning, more aggregate bandwidth
- Can cascade devices…do spanning tree
- We're reinvented the bridge!

Basic idea of a packet

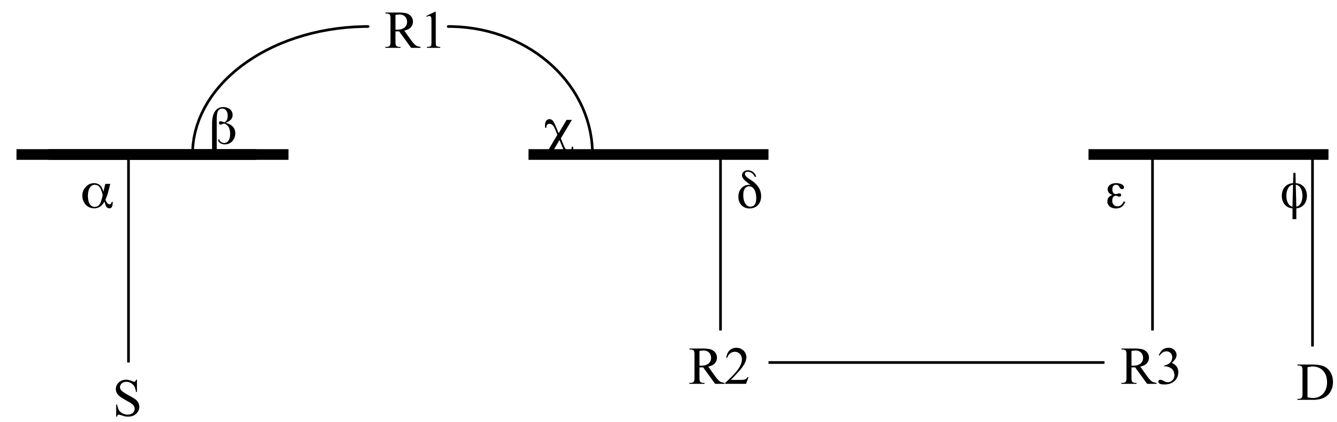


Hdrs inside hdrs



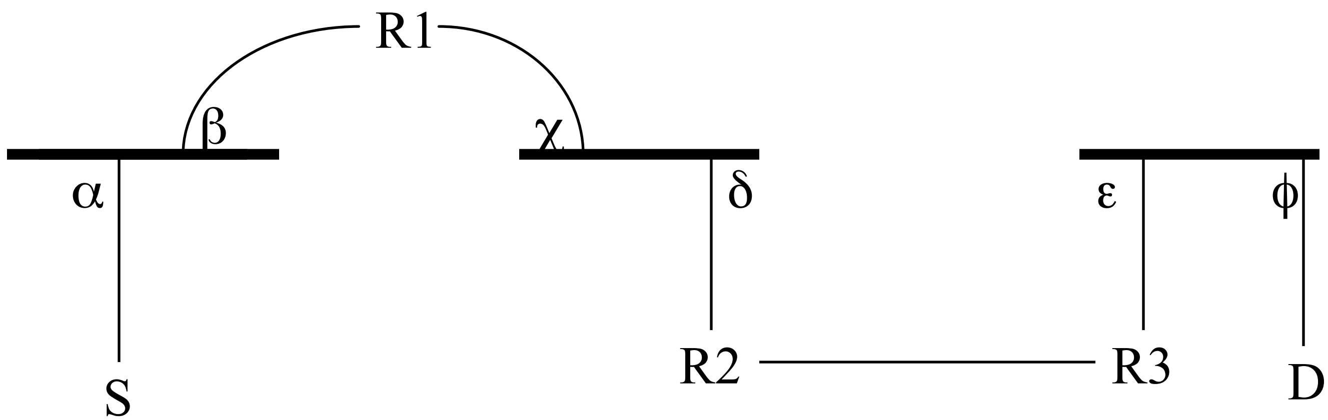
As transmitted by S? (L2 hdr, L3 hdr)
As transmitted by R1?
As received by D?

Hdrs inside hdrs



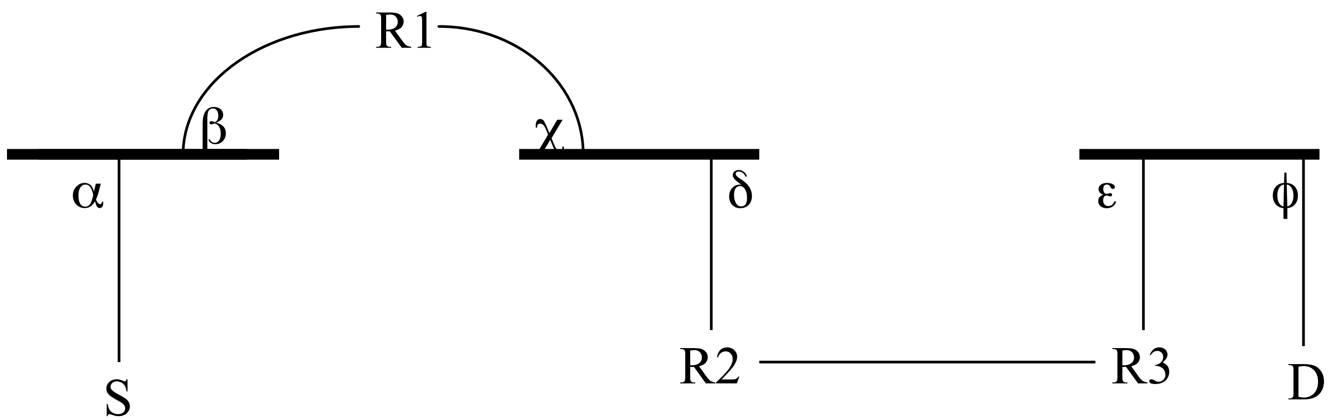
S:	Dest=β Source=α	Dest=D Source=S	
	Layer 2 hdr	Layer 3 hdr	

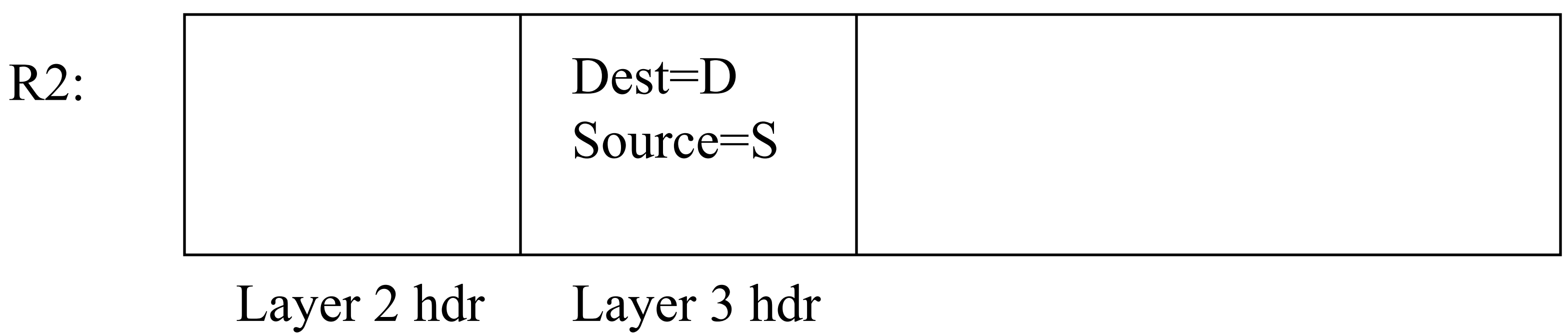
Hdrs inside hdrs



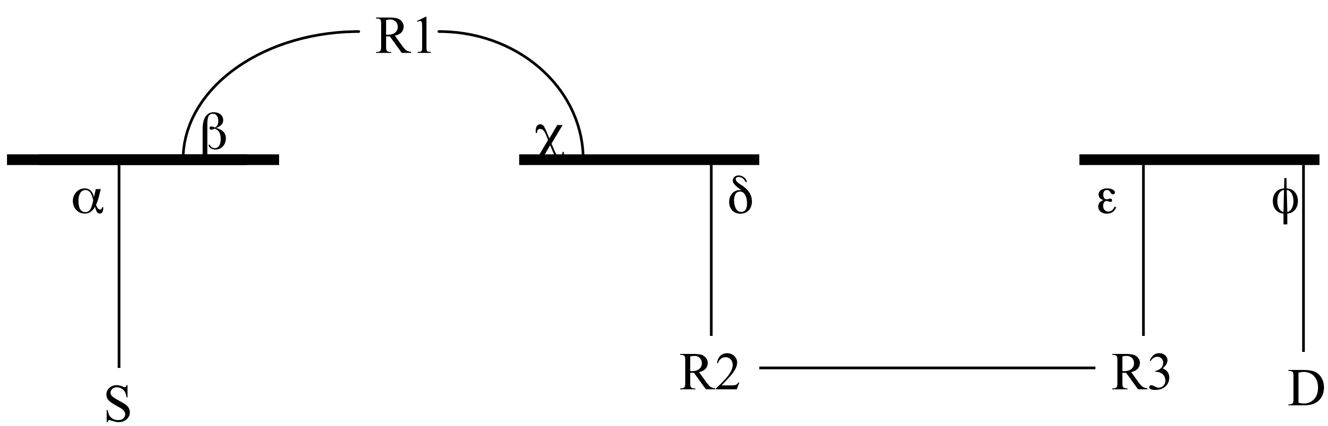
R1:	Dest=δ Source=χ	Dest=D Source=S	
	Layer 2 hdr	Layer 3 hdr	

Hdrs inside hdrs





Hdrs inside hdrs



R3:

Layer 2 hdr	Layer 3 hdr	
Dest=φ Source=ε	Dest=D Source=S	

What designing “layer 3” meant

- Layer 3 addresses
- Layer 3 packet format (IP, DECnet)
  – Source, destination, hop count, …
- A routing algorithm
  – Exchange information with your neighbors
  – Collectively compute routes with all rtrs
  – Compute a forwarding table

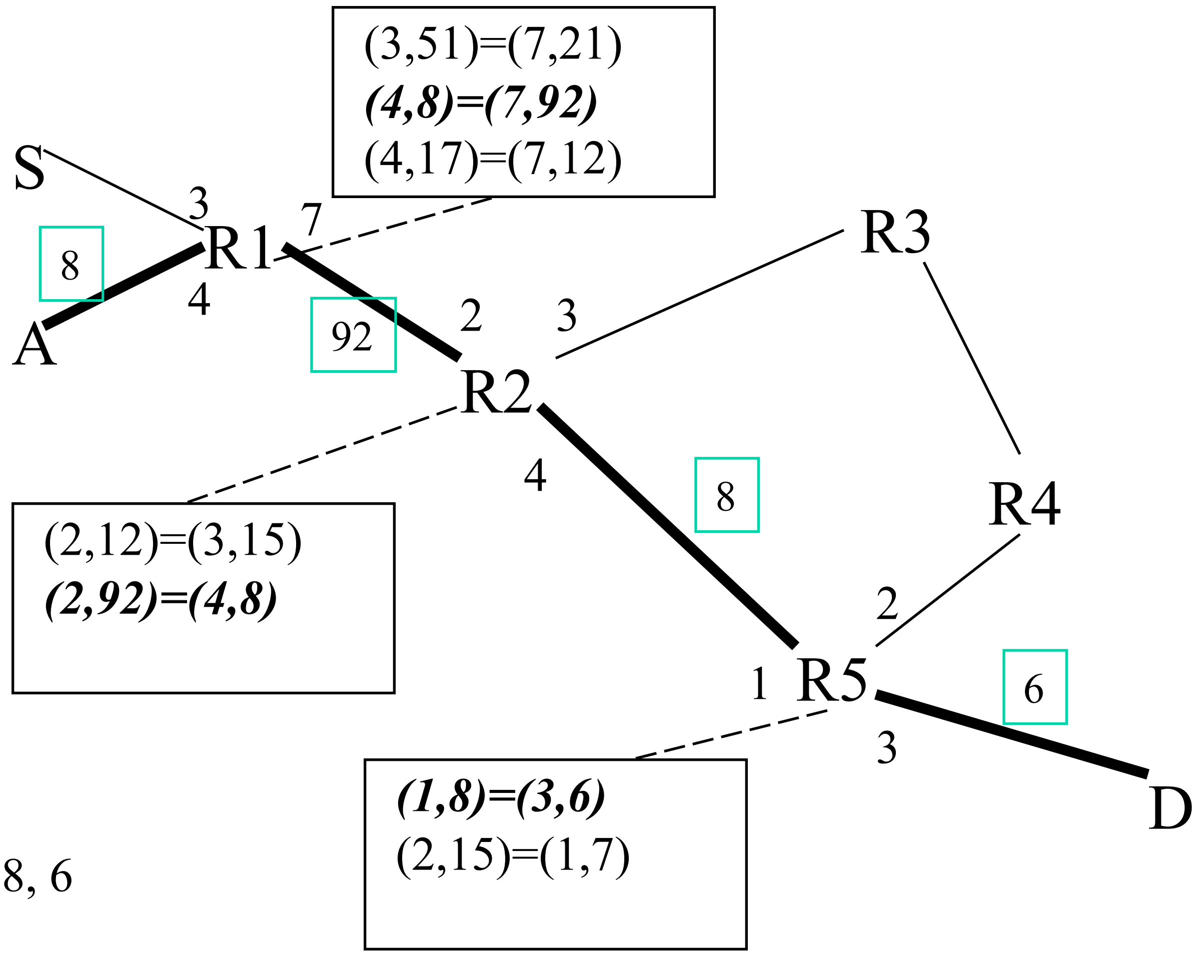
Network Layer

- connectionless fans designed IPv4, IPv6, CLNP, IPX, AppleTalk, DECnet
- Connection-oriented reliable fans designed X.25
- Connection-oriented datagram fans designed ATM, MPLS

Pieces of network layer

- interface to network: addressing, packet formats, fragmentation and reassembly, error reports
- routing protocols
- autoconfiguring addresses/nbr discovery/ finding routers

Connection-oriented Nets



VC=8, 92, 8, 6

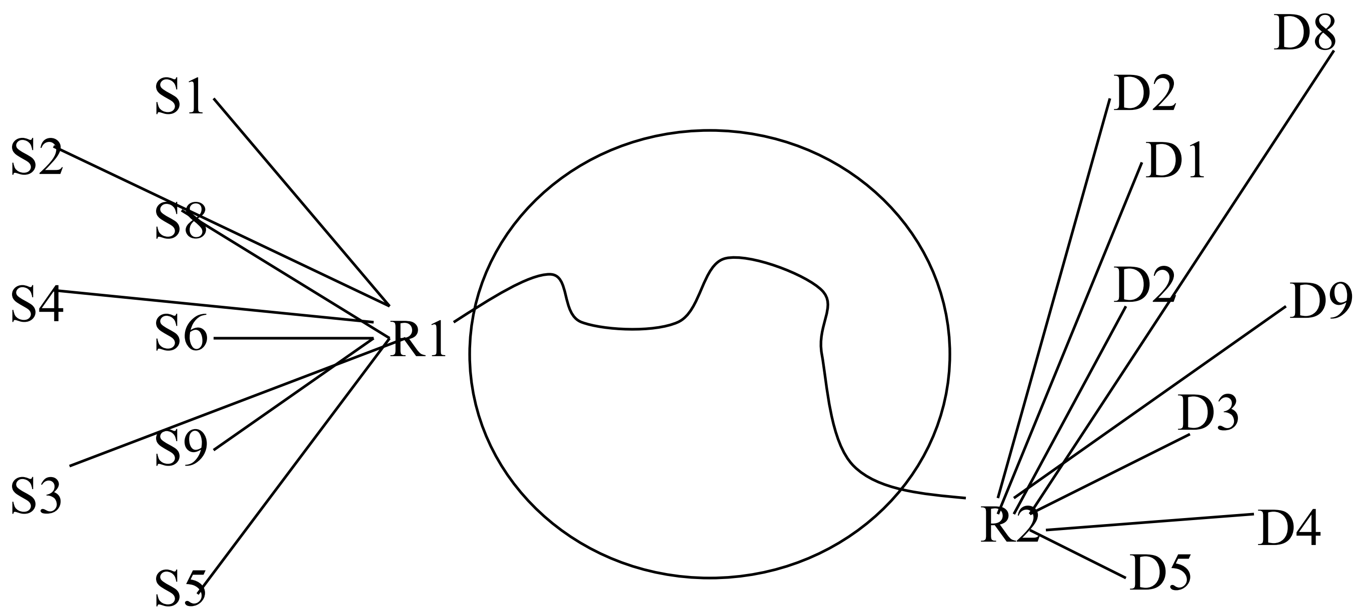
Connection-oriented networks

- X.25: also have sequence number and ack number in packets (like TCP), and layer 3 guarantees delivery
- ATM: datagram, but fixed size packets (48 bytes data, 5 bytes header)

MPLS (multiprotocol label switching)

- Connectionless, like MPLS, but arbitrary sized packets
- Add 32-bit hdr on top of IP pkt
  - 20 bit "label"
  - Hop count (hooray!)

Hierarchical connections (stacks of MPLS labels)



Routers in backbone only need to know about one flow: R1-R2

MPLS

- Originally for faster forwarding than parsing IP header
- later “traffic engineering”
- classify pkts based on more than destination address

Connectionless Network Layers

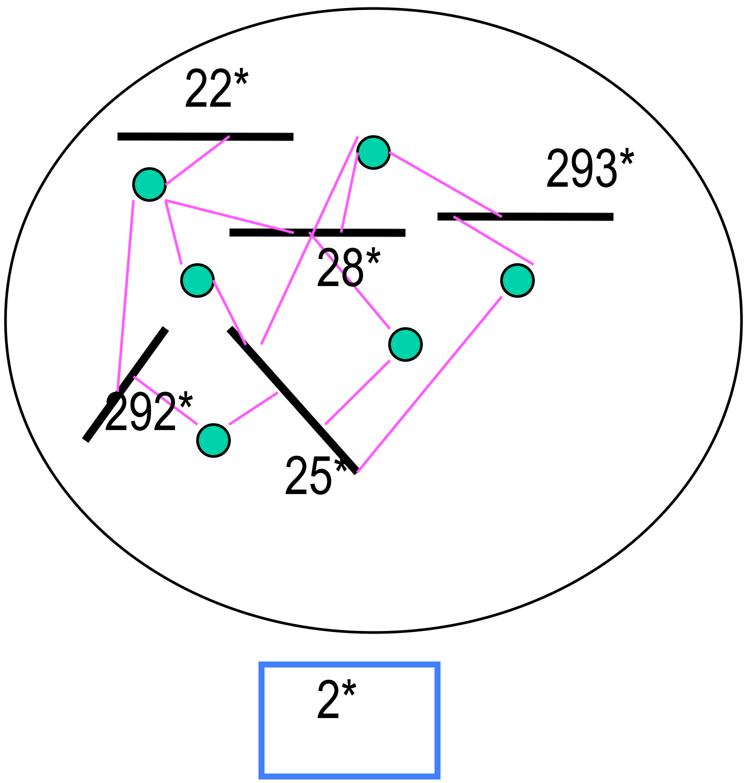
- Destination, source, hop count
- Maybe other stuff
  - fragmentation
  - options (e.g., source routing)
  - error reports
  - special service requests (priority, custom routes)
  - congestion indication
- Real diff: size of addresses

Addresses

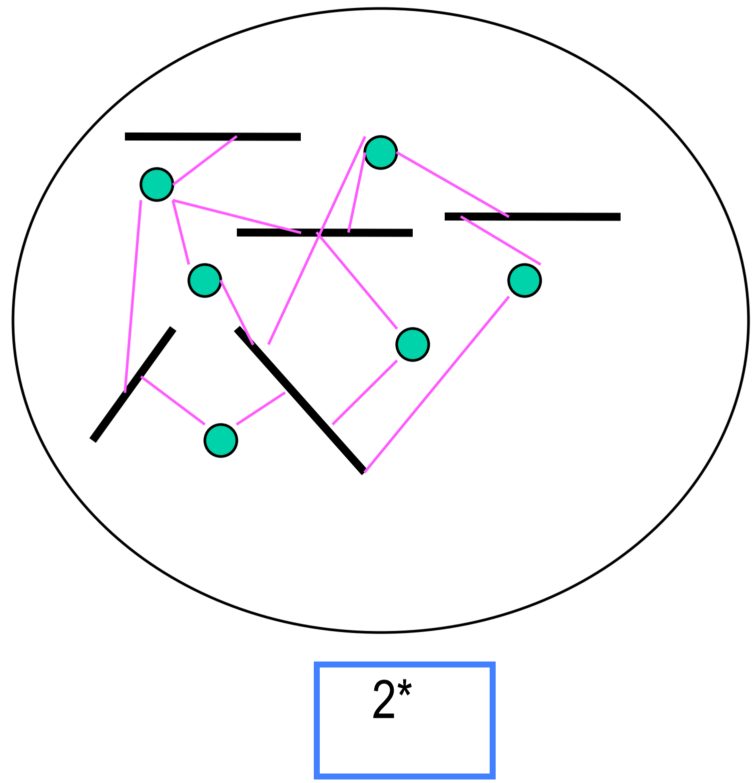
- 802 address “flat”, though assigned with OUI/rest. No topological significance
- layer 3 addresses: locator/node : topologically hierarchical address
- interesting difference:
  - IPv4, IPv6, IPX, AppleTalk: locator specific to a link
  - CLNP, DECnet: locator “area”, whole campus

Hierarchy

One prefix per link

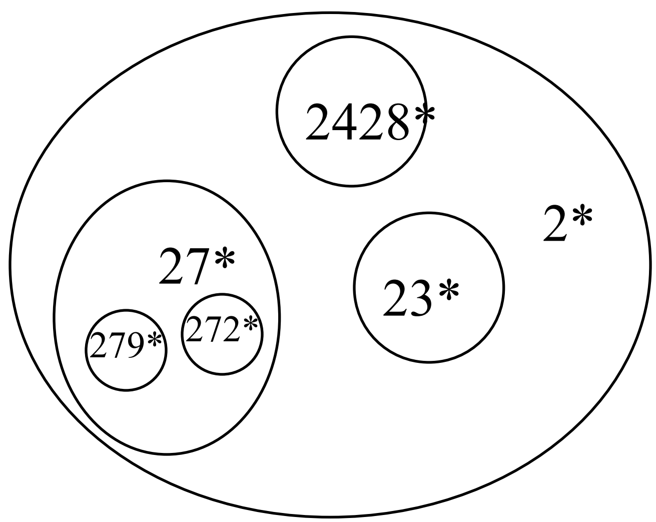


One prefix per campus

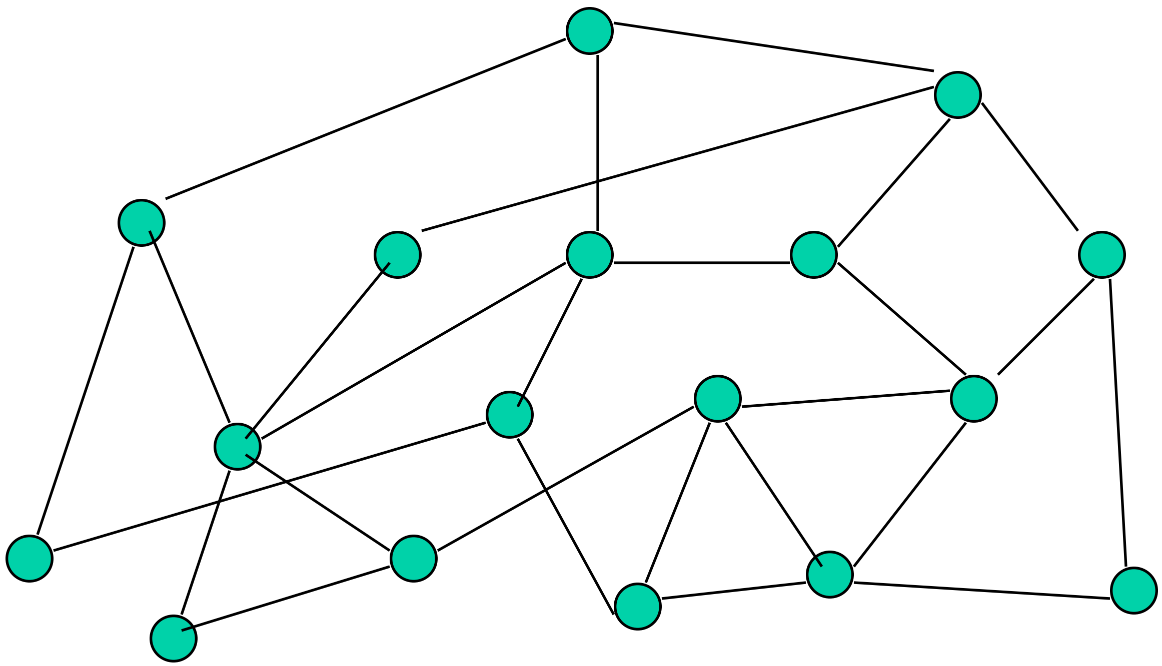


Hierarchy within Locator

- Assume addresses assigned so that within a circle everything shares a prefix
- Can summarize lots of circles with a shorter prefix



New topic: Routing Algorithms

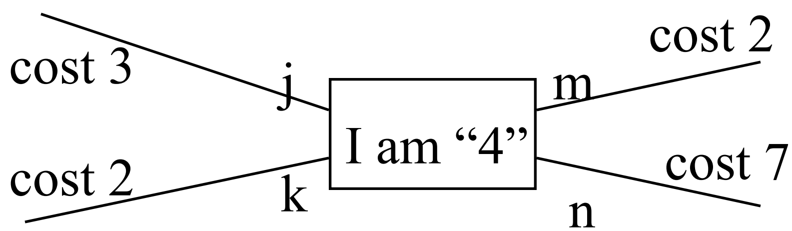


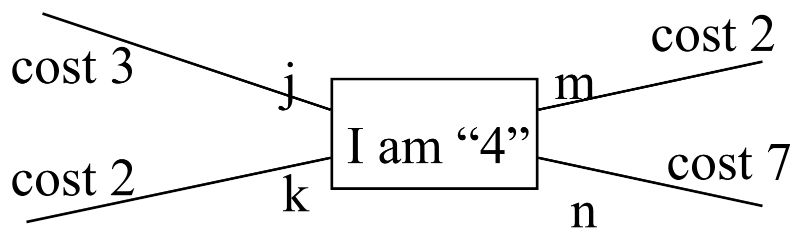
Distributed Routing Protocols

- Rtrs exchange control info
- Use it to calculate forwarding table
- Two basic types
  - distance vector
  - link state

Distance Vector

- Know
  - your own ID
  - how many cables hanging off your box
  - cost, for each cable, of getting to nbr





distance vector rcv’d from cable j

cost 3

12	3	15	3	12	5	3	18	0	7	15

distance vector rcv’d from cable k

cost 2

5	8	3	2	10	7	4	20	5	0	15

distance vector rcv’d from cable m

cost 2

0	5	3	2	19	9	5	22	2	4	7

distance vector rcv’d from cable n

cost 7

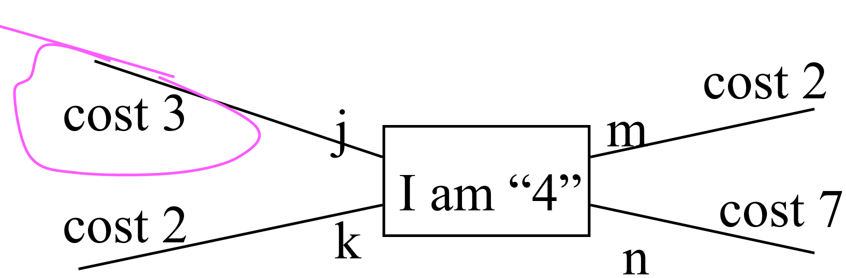
6	2	0	7	8	5	8	12	11	3	2

your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

your own calculated forwarding table

m	j	m	0	k	j	k/j	n	j	?	?



distance vector rcv'd from cable j

cost 3	12	3	15	3	12	5	3	18	0	7	15

distance vector rcv'd from cable k

cost 2	5	8	3	2	10	7	4	20	5	0	15

distance vector rcv'd from cable m

cost 2	0	5	3	2	19	9	5	22	2	4	7

distance vector rcv'd from cable n

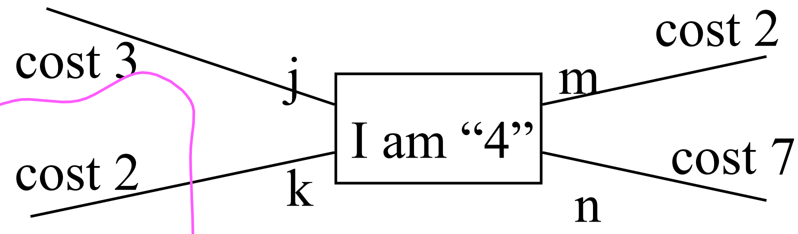
cost 7	6	2	0	7	8	5	8	12	11	3	2

your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

your own calculated forwarding table

m	j	m	0	k	j	k/j	n	j	?	?



distance vector rcv’d from cable j

cost 3	12	3	15	3	12	5	3	18	0	7	15

distance vector rcv’d from cable k

cost 2	5	8	3	2	10	7	4	20	5	0	15

distance vector rcv’d from cable m

cost 2	0	5	3	2	19	9	5	22	2	4	7

distance vector rcv’d from cable n

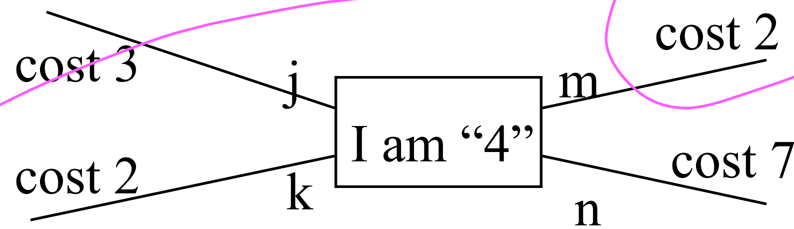
cost 7	6	2	0	7	8	5	8	12	11	3	2

your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

your own calculated forwarding table

m	j	m	0	k	j	k/j	n	j	?	?



distance vector rcv’d from cable j

cost 3	12	3	15	3	12	5	3	18	0	7	15

distance vector rcv’d from cable k

cost 2	5	8	3	2	10	7	4	20	5	0	15

distance vector rcv’d from cable m

cost 2	0	5	3	2	19	9	5	22	2	4	7

distance vector rcv’d from cable n

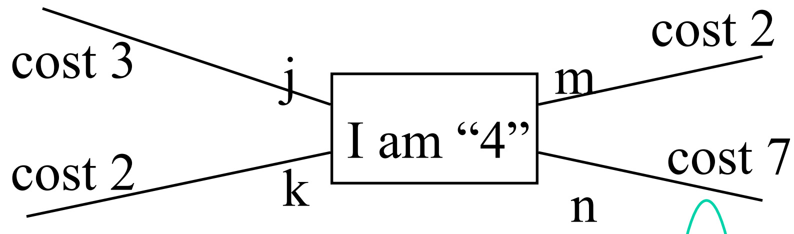
cost 7	6	2	0	7	8	5	8	12	11	3	2

your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

your own calculated forwarding table

m	j	m	0	k	j	k/j	n	j	?	?



distance vector rcv'd from cable j

cost 3

12	3	15	3	12	5	3	18	0	7	15

distance vector rcv'd from cable k

cost 2

5	8	3	2	10	7	4	20	5	0	15

distance vector rcv'd from cable m

cost 2

0	5	3	2	19	9	5	22	2	4	7

distance vector rcv'd from cable n

cost 7

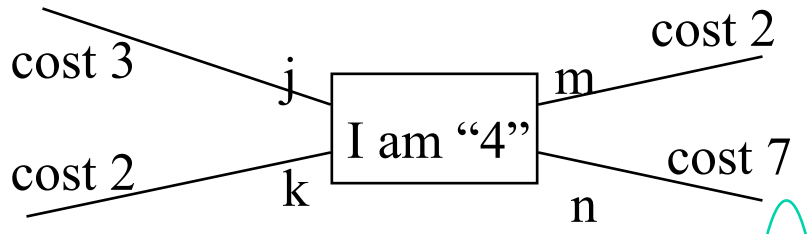
6	2	0	7	8	5	8	12	11	3	2

your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

your own calculated forwarding table

m	j	m	0	k	j	k/j	n	j	?	?



distance vector rcv’d from cable j

cost 3	12	3	15	3	12	5	3	18	0	7	15

distance vector rcv’d from cable k

cost 2	5	8	3	2	10	7	4	20	5	0	15

distance vector rcv’d from cable m

cost 2	0	5	3	2	19	9	5	22	2	4	7

distance vector rcv’d from cable n

cost 7	6	2	0	7	8	5	8	12	11	3	2

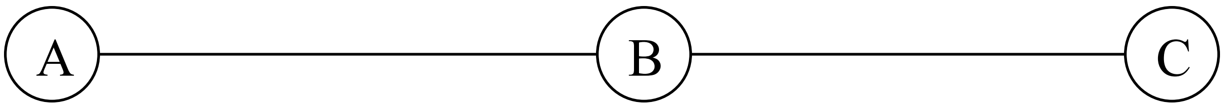
your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

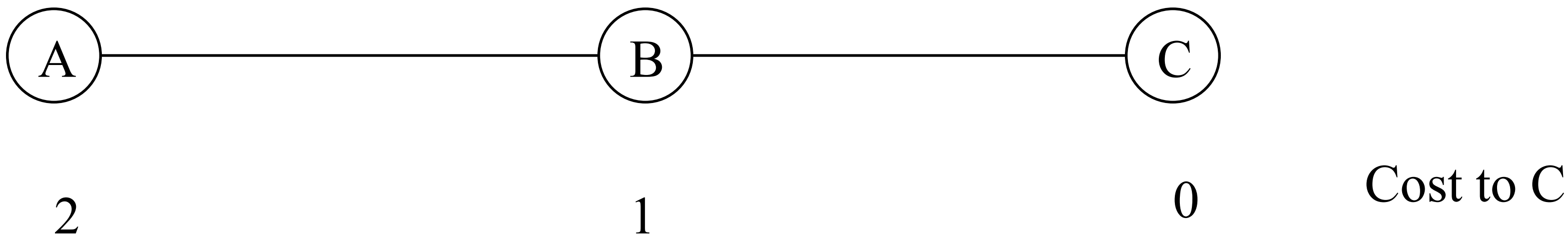
your own calculated forwarding table

m	j	m	0	k	j	k/j	n	j	?	?

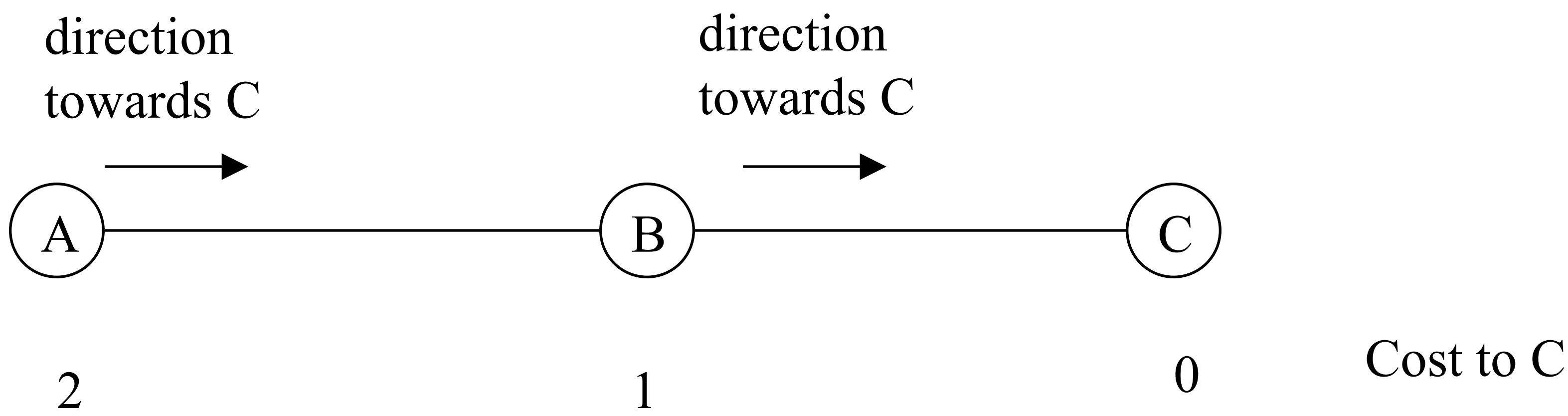
Looping Problem



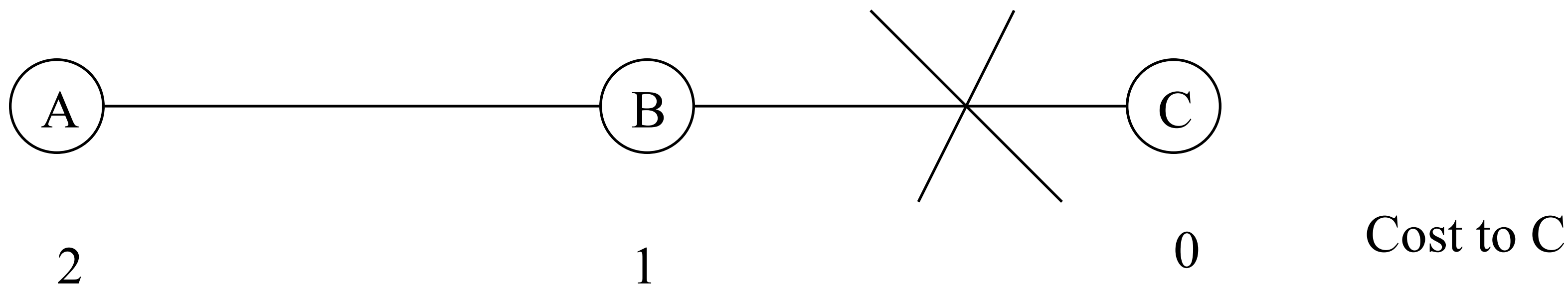
Looping Problem



Looping Problem

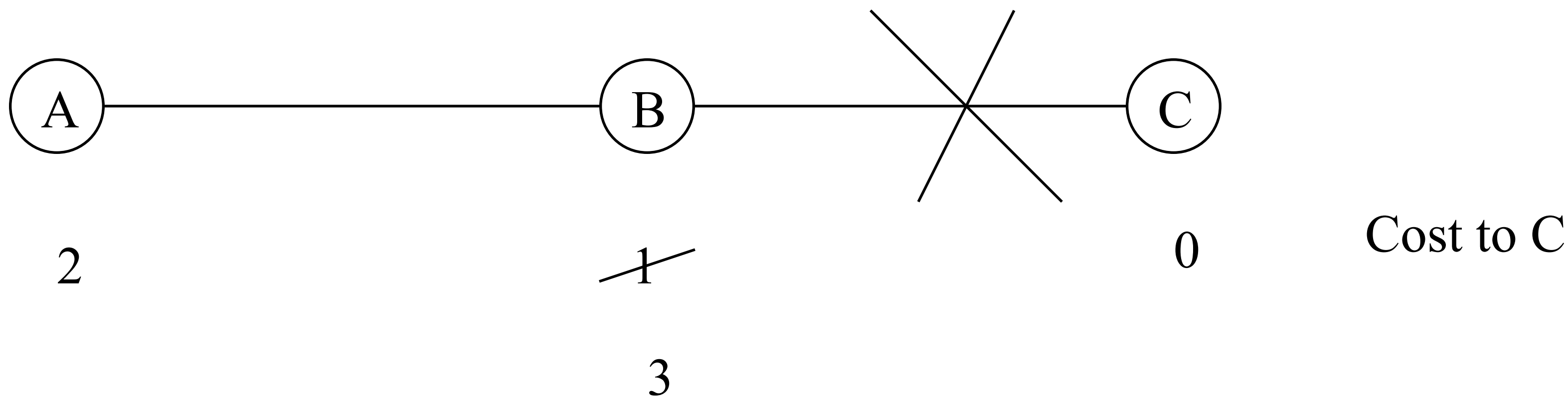


Looping Problem

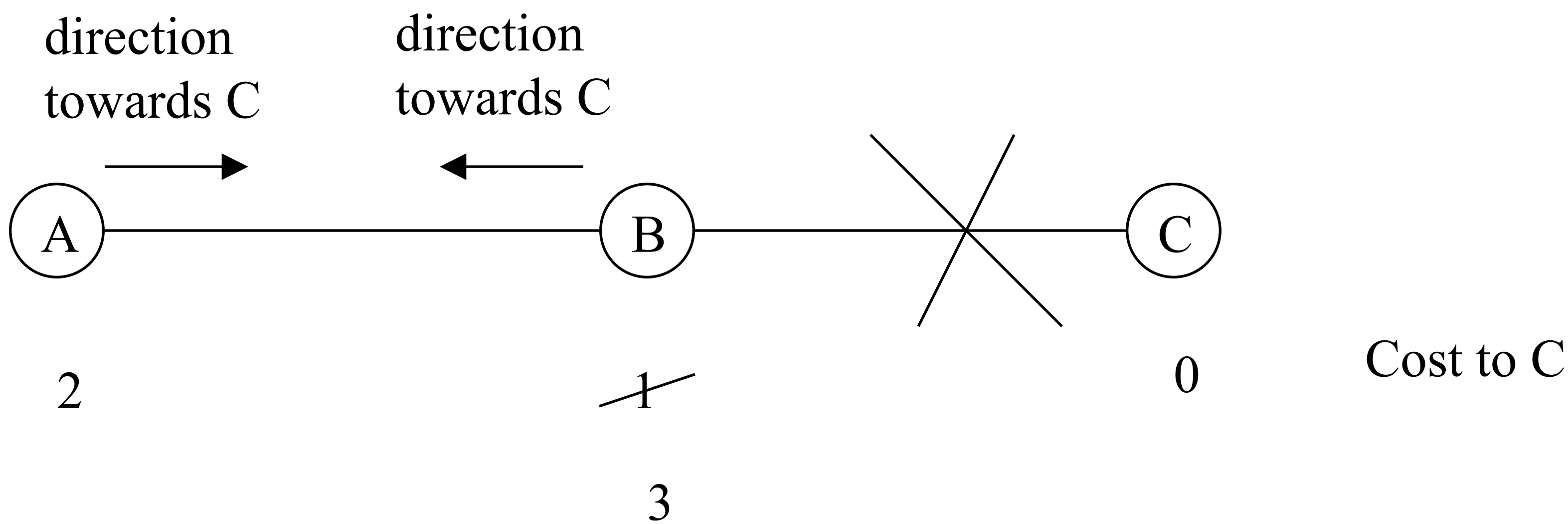


What is B's cost to C now?

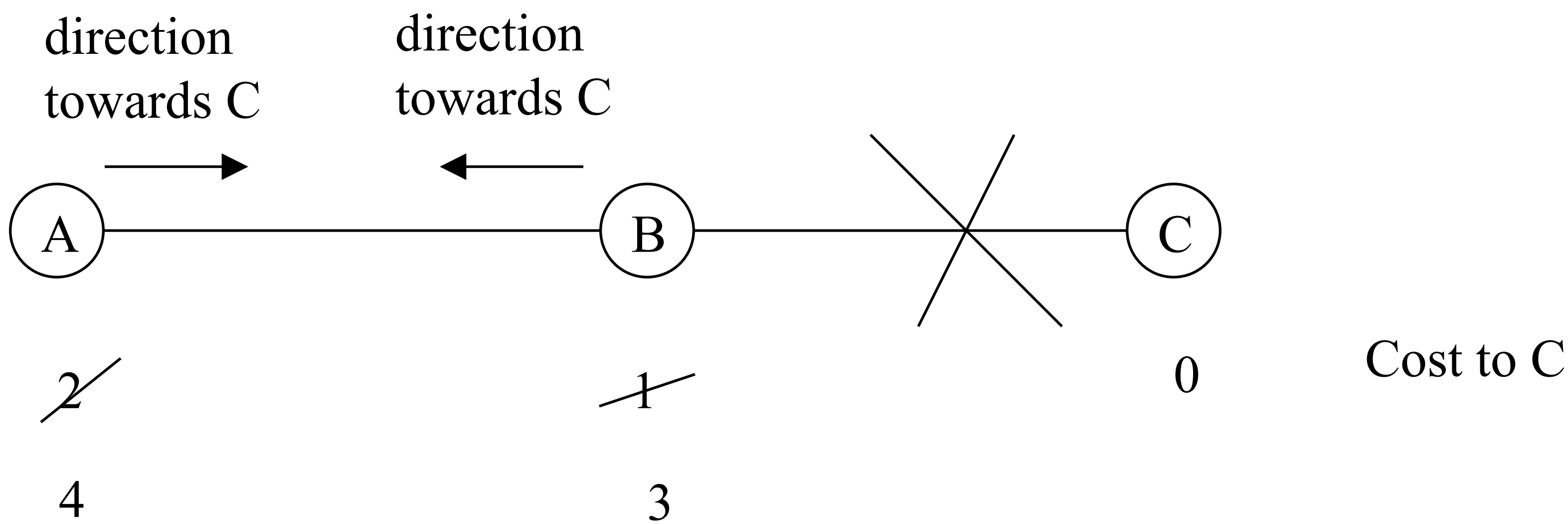
Looping Problem



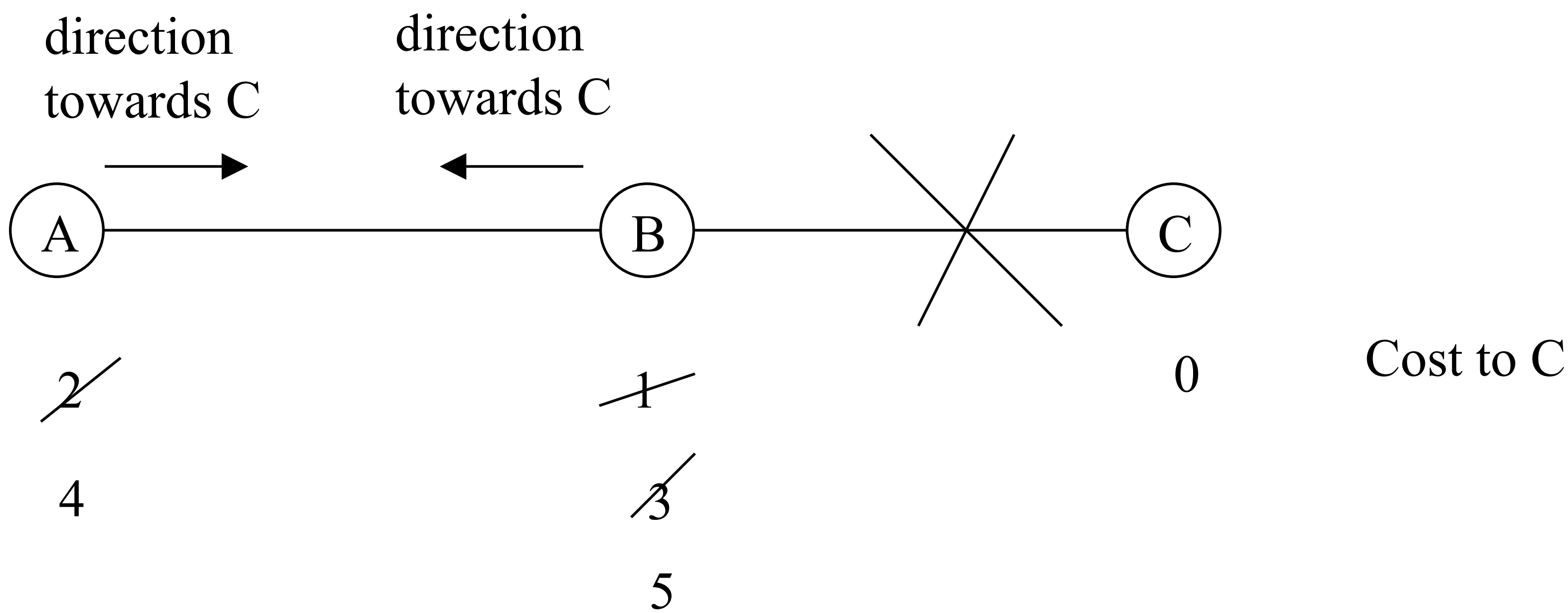
Looping Problem



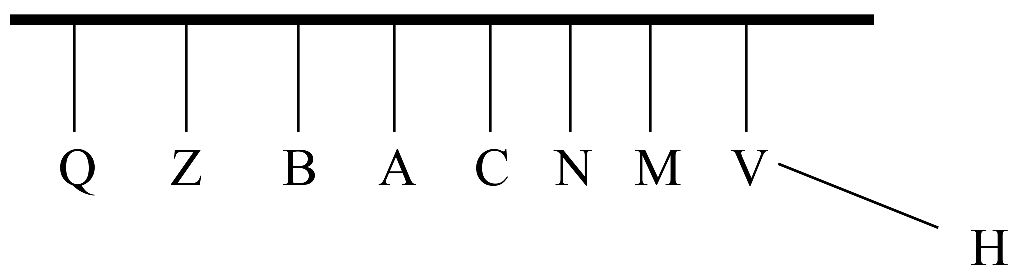
Looping Problem



Looping Problem

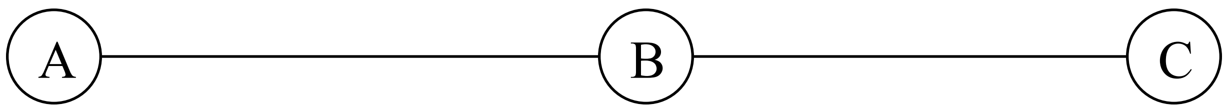


Looping Problem worse with high connectivity

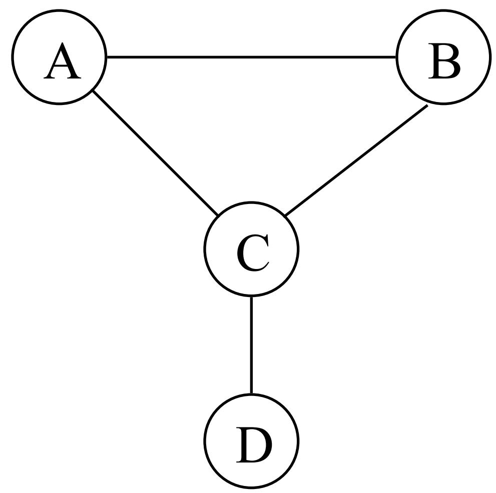


Split Horizon: one of several optimizations

Don't tell neighbor N you can reach D if you'd forward to D through N

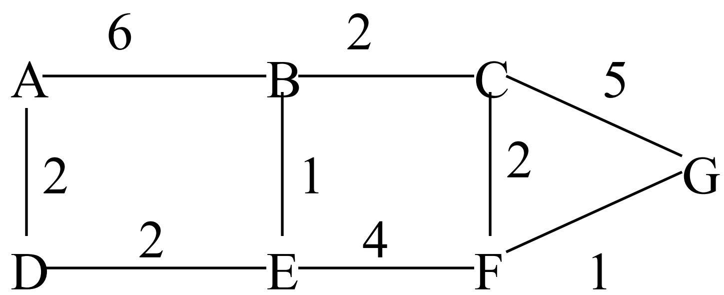


Split Horizon: …but it won’t work with loops of more than 2 nodes



Link State Routing

- meet nbrs
- Construct Link State Packet (LSP)
  - who you are
  - list of (nbr, cost) pairs
- Broadcast LSPs to all rtrs (“a miracle occurs”)
- Store latest LSP from each rtr
- Compute Routes (breadth first, i.e., “shortest path” first—well known and efficient algorithm)



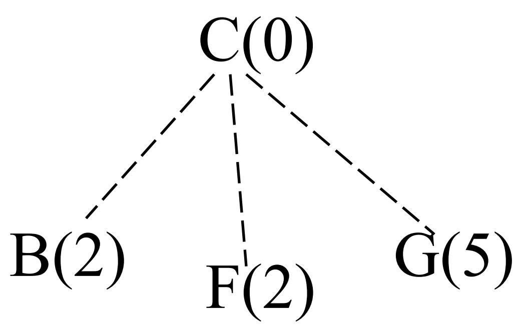
A	B	C	D	E	F	G
B/6	A/6	B/2	A/2	B/1	C/2	C/5
D/2	C/2	F/2	E/2	D/2	E/4	F/1
	E/1	G/5		F/4	G/1	

Computing Routes

- Edsgar Dijkstra's algorithm:
  - calculate tree of shortest paths from self to each
  - also calculate cost from self to each
  - Algorithm:
    - step 0: put (SELF, 0) on tree
    - step 1: look at LSP of node (N,c) just put on tree. If for any nbr K, this is best path so far to K, put (K, c +dist(N,K)) on tree, child of N, with dotted line
    - step 2: make dotted line with smallest cost solid, go to step 1

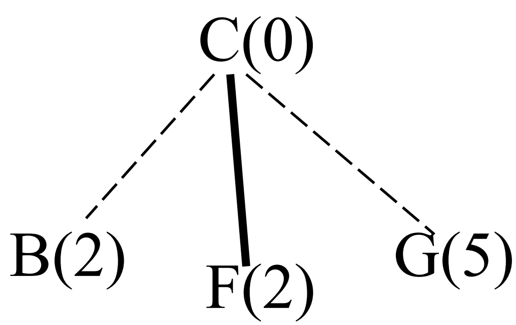
Look at LSP of new tree node

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	



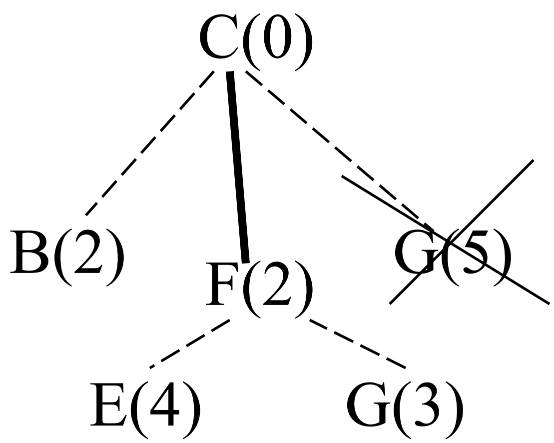
Make shortest TENT solid

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	



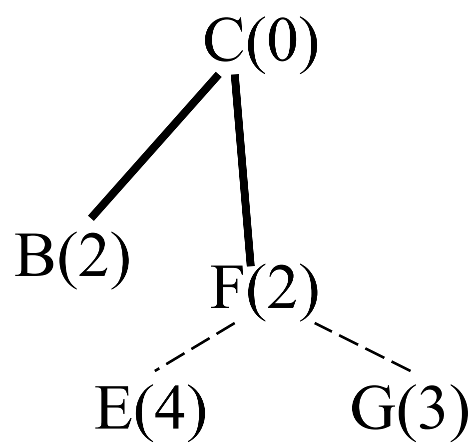
Look at LSP of newest tree node

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	



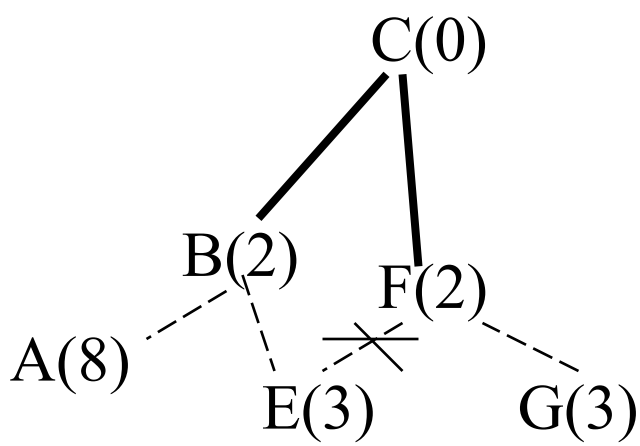
Make shortest TENT solid

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	



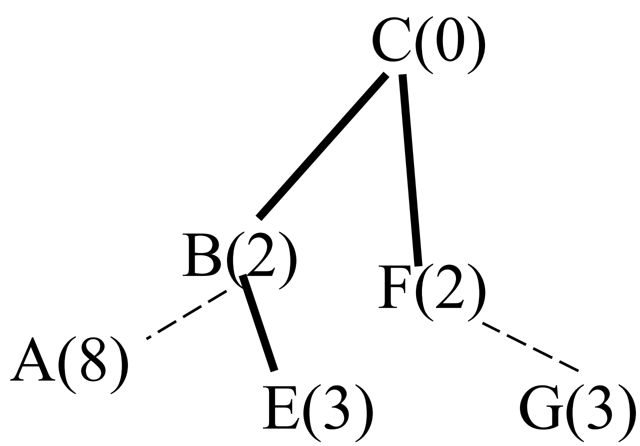
Look at LSP of newest tree node

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	



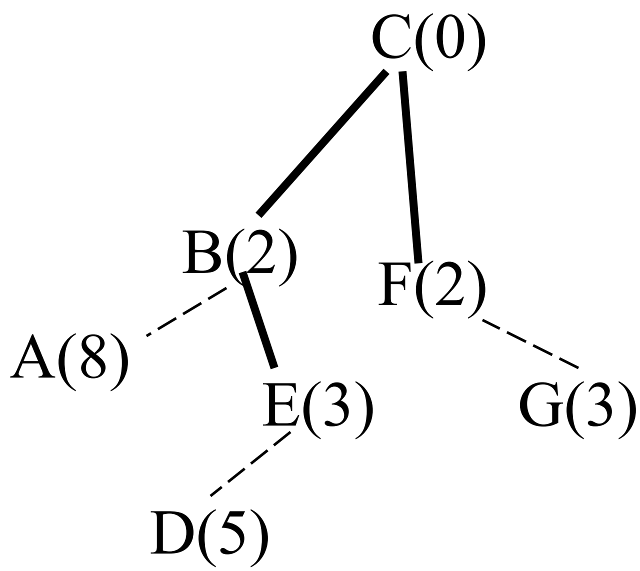
Make shortest TENT solid

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	



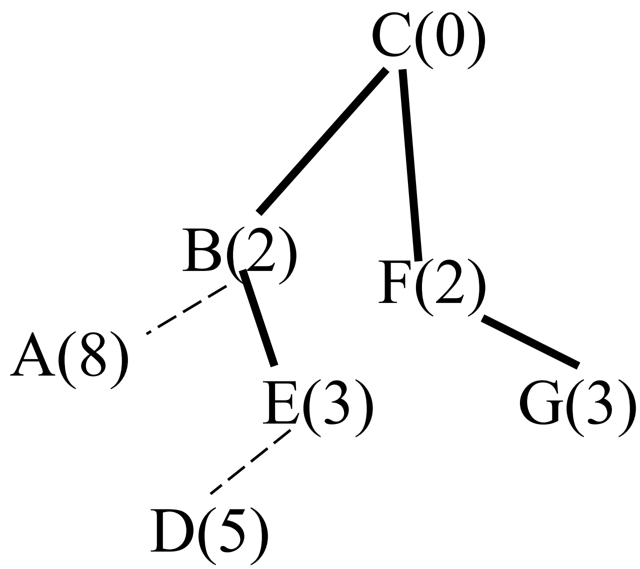
Look at LSP of newest tree node

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	



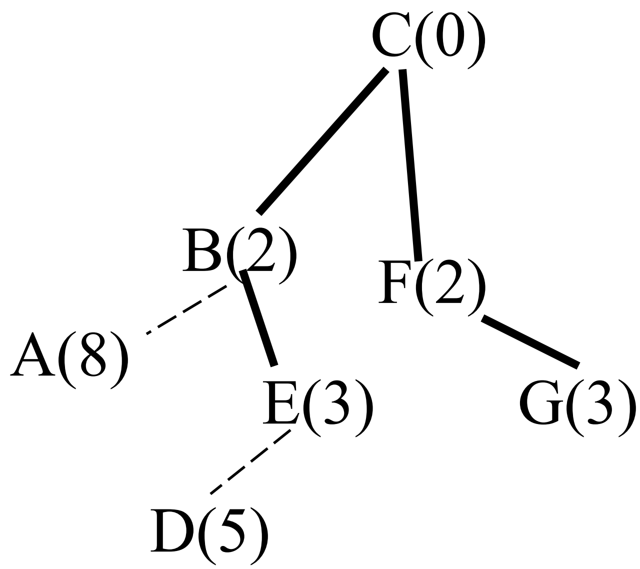
Make shortest TENT solid

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	



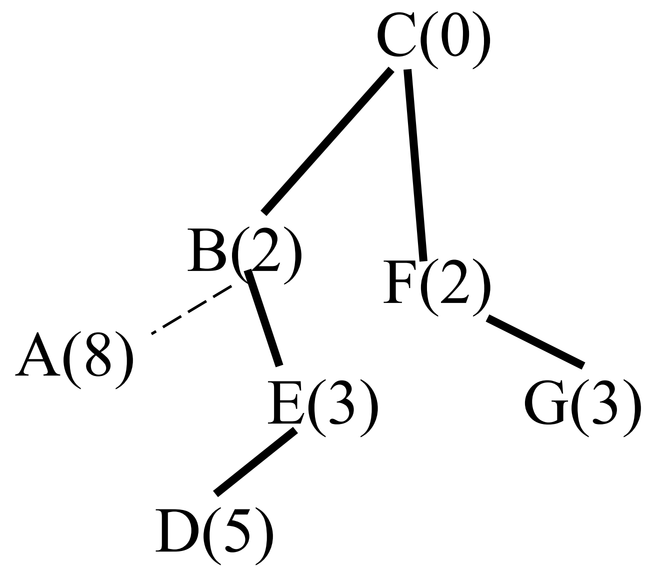
Look at newest tree node's LSP

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	



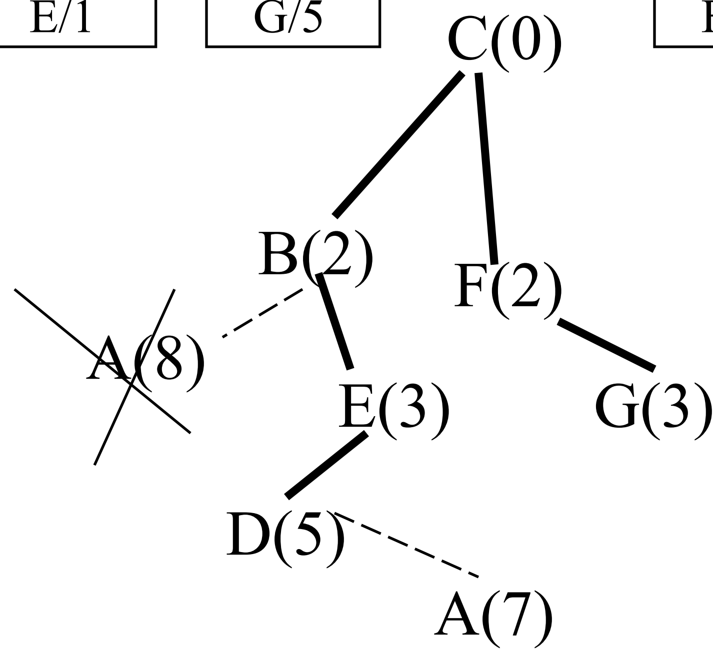
Make shortest TENT solid

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	



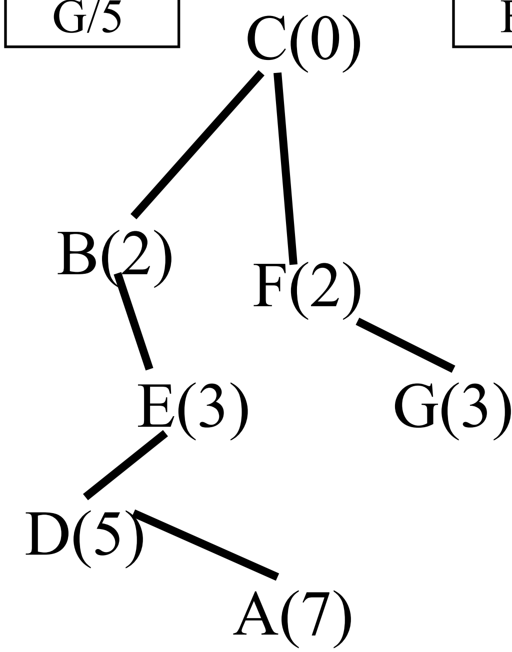
Look at newest node's LSP

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	



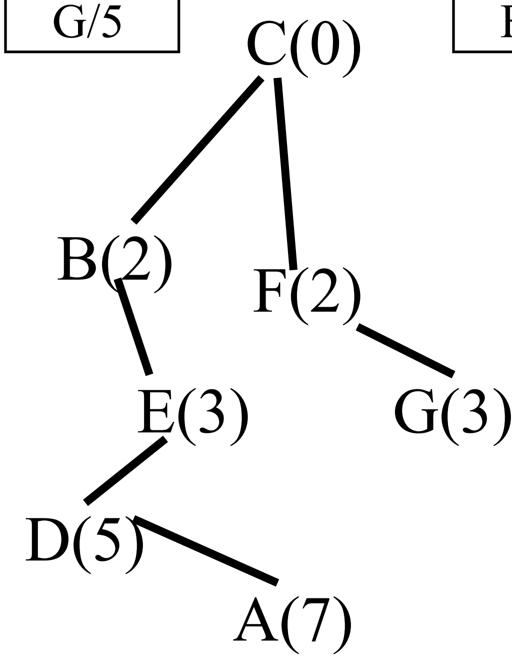
Make shortest TENT solid

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	



We’re done!

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	

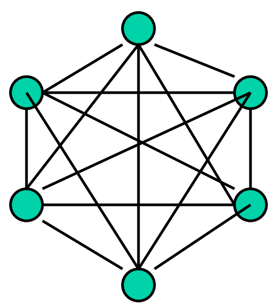


Another interesting detail of link state

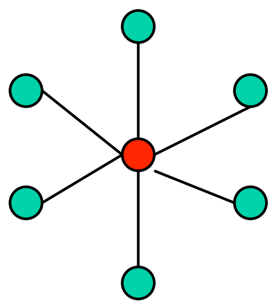
- Pseudonodes
- Since routing algorithm is proportional to the number of links
- If an Ethernet with 100s of nodes were considered fully connected, the link state database would be too large

Pseudonodes

Instead of:



Use pseudonode



Designated Routers

- Elect a router to be the master of the link
- It names the pseudonode
  - In IS-IS, a node's ID is 7 bytes: 6 bytes of system ID (usually the MAC address of one of its ports), plus an extra byte. E.G., R1 is DR, names link R1.25
- All routers (including R1) claim a link to R1.25
- R1 (pretending to be the pseudonode), claims connectivity to each of the routers on the link

Distance vector vs link state

- Memory: distance vector wins (but memory is cheap)
- Computation: debatable
- Simplicity of coding: simple distance vector wins. Complex new-fangled distance vector, no
- Convergence speed: link state
- Functionality: link state; custom routes, mapping the net, troubleshooting, sabotage-proof routing

Specific Routing Protocols

- Interdomain vs Intradomain
- Intradomain:
  - link state (OSPF, IS-IS)
  - distance vector (RIP)
- Interdomain
  - BGP

BGP (Border Gateway Protocol)

- "Policies", not just minimize path
- "Path vector": given reported paths to D from each nbr, and configured preferences, choose your path to D
  - don't ever route through domain X, or not to D, or only as last resort
- Other policies: don't tell nbr about D, or lie to nbr about D making path look worse

Interesting use of BGP

- Lifeguard: Locating Internet Failures WEffectively and Generating Usable Alternative Routes Dynamically
- Work at University of Washington:
  - Ethan Katz-Bassett, David Choffnes, Colin Scott, Arvind Krishnamurthy, Tom Anderson
- If want others to avoid ASx, claim ASx is already in the path!

Path vector/Distance vector

- Distance vector
  - Each router reports to its neighbors {(D,cost)}
  - Each router chooses best path based on min (reported cost to D+link cost to nbr)
- Path vector
  - Each rtr R reports {(D,list of AS's in R's chosen path to D)…}
  - Each rtr chooses best path based on configured policies

BGP Configuration

- path preference rules
- which nbr to tell about which destinations
- how to "edit" the path when telling nbr N about prefix P (add fake hops to discourage N from using you to get to P)

So, world is confusing, what with layer 2 and layer 3

So, world is confusing, what with layer 2 and layer 3

- So let’s invent layer 2 ½!

What's wrong with bridges?

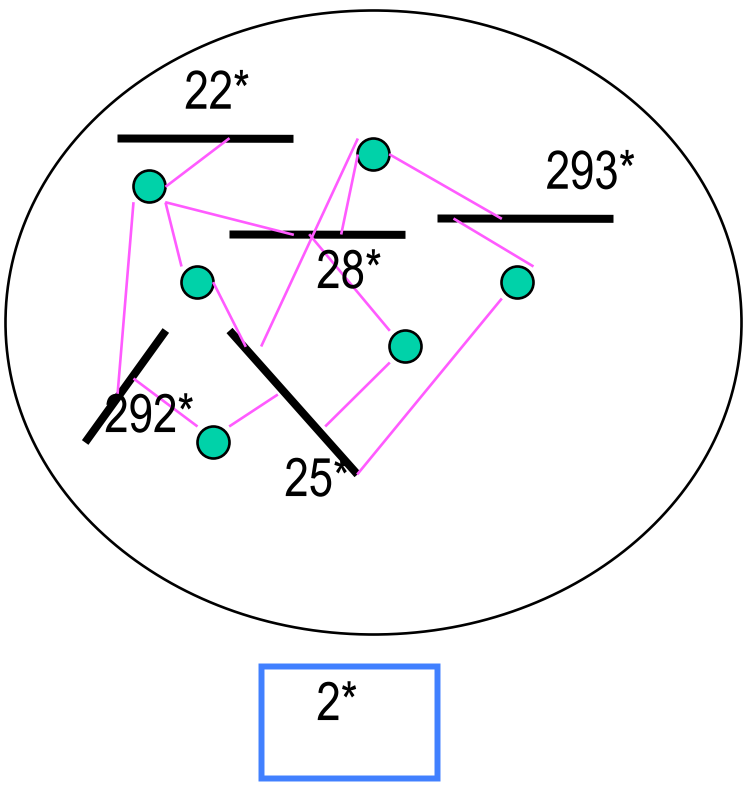
- Suboptimal routing
- Traffic concentration
- Temporary loops real dangerous (no hop count, exponential proliferation)
- Fragile
  - If lose packets (congestion?), turn ***<u>on</u>*** port

Why not replace bridges with IP routers?

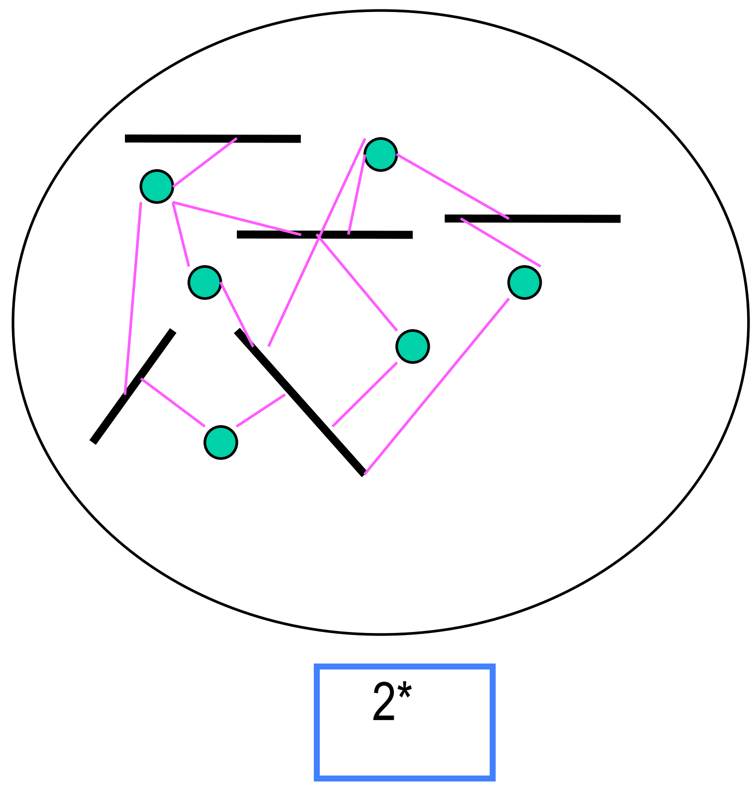
- Subtle reason: IP needs address per link.
- Layer 3 doesn’t have to work that way
  - CLNP / DECnet
    - Bottom level of routing is a whole cloud with the same prefix
    - Routing is to endnodes inside the cloud
    - Enabled by “ES-IS” protocol, where endnodes periodically announce themselves to the routers
    - Also in ES-IS: routers announce themselves to endnodes…

Hierarchy

One prefix per link



One prefix per campus



A bit of history

- 1992…Internet could have adopted CLNP
- Easier to move to a new layer 3 back then
  - Internet smaller
  - Not so mission critical
  - IP hadn't yet (out of necessity) invented DHCP, NAT, so CLNP gave understandable advantages
- CLNP still has advantages over IPv6 (e.g., large multilink level 1 clouds)

TRILL working group in IETF

- TRILL= TRansparent Interconnection of Lots of Links
- Use layer 3 routing, and encapsulate with a civilized header
- But still look like a bridge from the outside

Goal

- Design so that change can be incremental
- With TRILL, replace any subset of bridges with RBridges
  - still looks to IP like one giant Ethernet
  - the more bridges you replace with RBridges, better bandwidth utilization, more stability

Run link state protocol

- So all the RBridges know how to reach all the other RBridges
- But don't know anything about endnodes

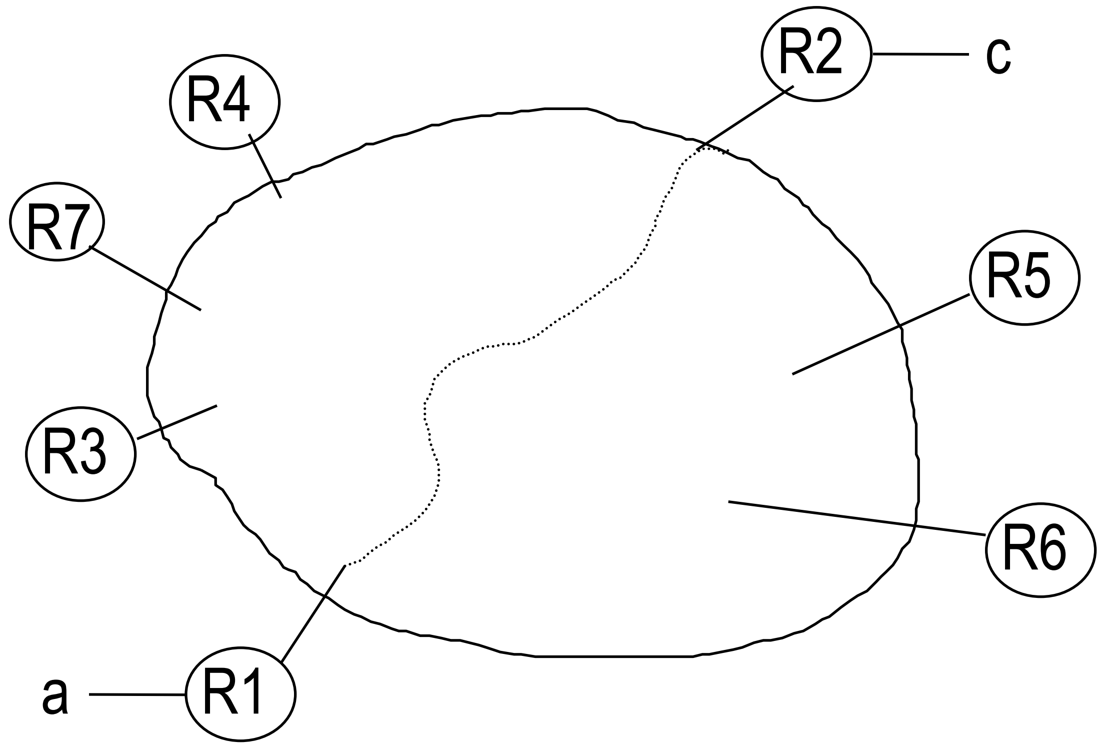
Why link state?

- Since all switches know the complete topology, easy to compute lots of trees deterministically (we’ll get to that later)
- Easy to piggyback “nickname allocation protocol” (we’ll get to that later)

Routing inside campus

- First RB encapsulates to last RB
  - So header is “safe” (has hop count)
  - Inner RBridges only need to know how to reach destination RBridge
- Still need tree for unknown/multicast
  - But don’t need spanning tree protocol – compute tree(s) deterministically from the link state database

Rbridging



Details

- What the encapsulated packet looks like
- How R1 knows that R2 is the correct "last RBridge"

Encapsulated Frame

(Ethernet) outer header	TRILL header	original frame
dest (nexthop) srce (Xmitter) Ethertype=TRILL	first RBridge last RBridge TTL	

TRILL header specifies RBridges with 2-byte nicknames

2-byte Nicknames

- Saves hdr room, faster fwd'ing
- Dynamically acquired
- Choose unused #, announce in LSP
- If collision, IDs and priorities break tie
- Loser chooses another nickname
- Configured nicknames higher priority

How does R1 know that R2 is the correct “last RBridge”?

- If R1 doesn’t, R1 sends packet through a tree
- When R2 decapsulates, it remembers (ingress RBridge, source MAC)

How does R1 know that R2 is the correct “last RBridge”?

- Original design
  - R1 responsible for learning its attached endnodes, and advertising to the other RBs

How does R1 know that R2 is the correct “last RBridge”?

- Original design
  - R1 responsible for learning its attached endnodes, and advertising to the other RBs
- People more familiar with layer 2 wanted
  - R2 (decapsulating RB) sees “first RB=R1, source MAC =a” and learns that “a” is attached to R1

Compromise

- Mandatory for last RB to learn from data
- Optional for an RB to advertise its endnodes
- Optional to learn from advertisements
- Advertised info “more definitive” than seeing source address in data packets
- Reasoning
  – In some cases learning local endnodes is more definitive

What if R1 doesn't know that R2 is the correct "last RBridge"?

- If R1 doesn't, R1 sends packet through a tree
- When R2 decapsulates, it remembers (ingress RBridge, source MAC)

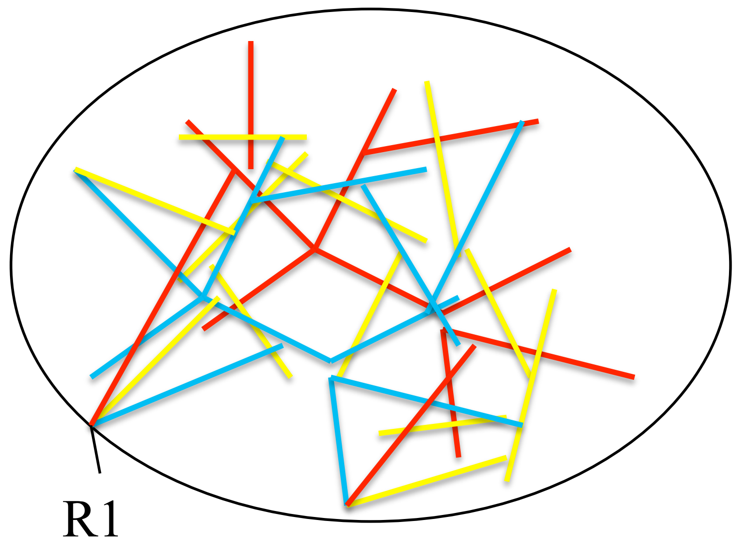
Trees

- Calculate based on link state database (not by running the spanning tree protocol)
- Original design: One tree
- WG wanted to multipath multidestination frames
- So TRILL calculates some # of trees, and ingress RB selects which tree

Use of “first” and “last” RBridge in TRILL header

- For Unicast, obvious
  - Route towards “last” RBridge
  - Learn location of source from “first” RBridge
- For Multicast/unknown destination
  - Use of “first”
    - to learn location of source endnode
    - to do “RPF check” on multicast
  - Use of “last”
    - To allow first RB to specify a tree
    - Campus calculates some number of trees

Multiple trees



R1 specifies which tree
(yellow, red, or blue)

RPF check

- RPF=reverse path forwarding
- For safety on multidestination frames…do sanity check: Could this frame have arrived on this tree, on this port, from this ingress RB?

Filtering of Multidestination Frames

- Sometimes a frame need not be "spanning", i.e., it need not be delivered everywhere
- Filtering is optional
- Two things that can help limit the spread (no RBs along a branch of the tree need to see the pkt)
  - VLAN
  - IP multicast group

Summary Tree distribution

- Each RB, say RB1, calculates which ports are in tree X (for each of the several trees)
- For tree X, for each port in tree X
  - RPF info: Which ingress RBs on that port
  - Filtering info: Which VLANs, which IP multicast addresses, that this port leads to
- If RPF=true, and that branch leads to receivers, transmit on that port

Some of the Future Work

- Taming various types of broadcast traffic (ARP, NETBIOS)
  - Cache ARP replies and negative responses
  - Query a directory which stores (IP, MAC, switch nickname)
    - Could be ingress switch, hypervisor, or end node
- Pseudonode nickname
- OAM
- Increasing the number of VLANs

Algorhyme v2

I hope that we shall one day see
A graph more lovely than a tree.

A graph to boost efficiency
While still configuration-free.

A network where RBridges can
Route packets to their target LAN.

The paths they find, to our elation,
Are least cost paths to destination.

With packet hop counts we now see,
The network need not be loop-free.

RBridges work transparently.
Without a common spanning tree.

Ray Perlner

Wrap-up

- folklore of protocol design
- things too obvious to say, but everyone gets them wrong

Forward Compatibility

- Reserved fields
  - spare bits
  - ignore them on receipt, set them to zero. Can maybe be used for something in the future
- TLV encoding
  - type, length, value
  - so can skip new TLVs
  - maybe have range of T's to ignore if unknown, others to drop packet

Forward Compability

- Make fields large enough
  - IP address, packet identifier, TCP sequence #
- Version number
  - what is “new version” vs “new protocol”?
    - same lower layer multiplex info
  - therefore, must always be in same place!
  - drop if version # bigger

Fancy version # variants

- Might be security threat to trick two Vn nodes into talk V(n-1)
- So maybe have "highest version I support" in addition to "version of this packet"
- Or just a bit "I can support higher" (we did this for IKEv2)
- Maybe have "minor version #", for compatible changes. Old node ignores it

Version

- Nobody seems to do this right
- IP, IKEv1, SSL unspecified what to do if version # different. Most implementations ignore it.
- SSL v3 moved version field!
  - v2 sets it to 0.2. v3 sets (different field) to 3.0.
  - v2 node will ignore version number field, and happily parse the rest of the packet

Avoid “flag days”

- Want to be able to migrate a running network
- ARPANET routing: ran both routing algorithms (but they had to compute the same forwarding table)
  - initially forward based on old, compute both
  - one by one: forward based on new
  - one-by-one: delete old

Parameters

- Minimize these:
  - someone has to document it
  - customer has to read documentation and understand it
- How to avoid
  - architectural constants if possible
  - automatically configure if possible

Settable Parameters

- Make sure they can’t be set incompatibly across nodes, across layers, etc. (e.g., hello time and dead timer)
- Make sure they can be set at nodes one at a time and the net can stay running

Parameter tricks

- IS-IS
  - pairwise parameters reported in "hellos"
  - area-wide parameters reported in LSPs
- Bridges
  - Use Root's values, sent in spanning tree msgs

Summary

- If things aren’t simple, they won’t work
- Good engineering requires understanding tradeoffs and previous approaches.
- It’s never a “waste of time” to answer “why is something that way”
- Don’t believe everything you hear
- Know the problem you’re solving before you try to solve it!