Tutorial on Bridges, Routers, Switches, Oh My!

Radia Perlman
(radia.perlman@sun.com)

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

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!

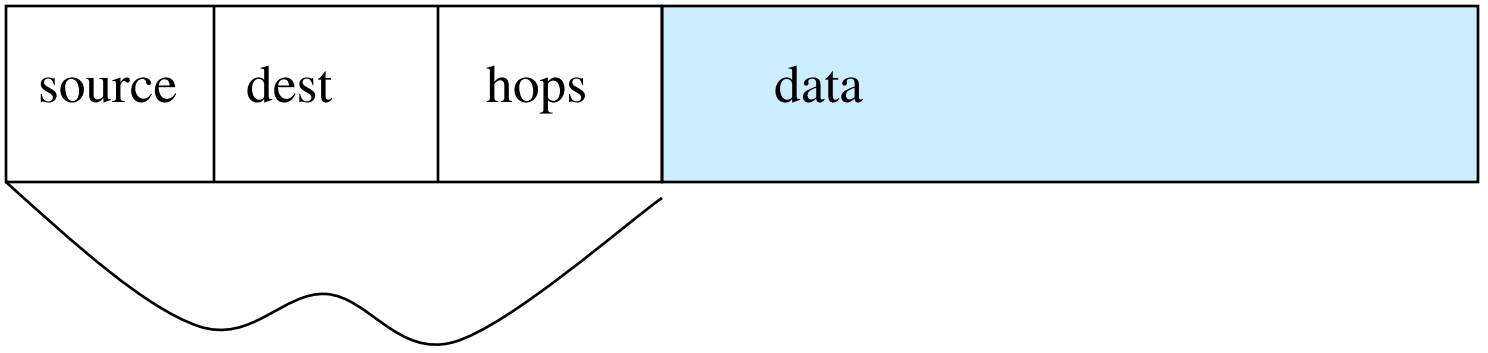
Definitions

- 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

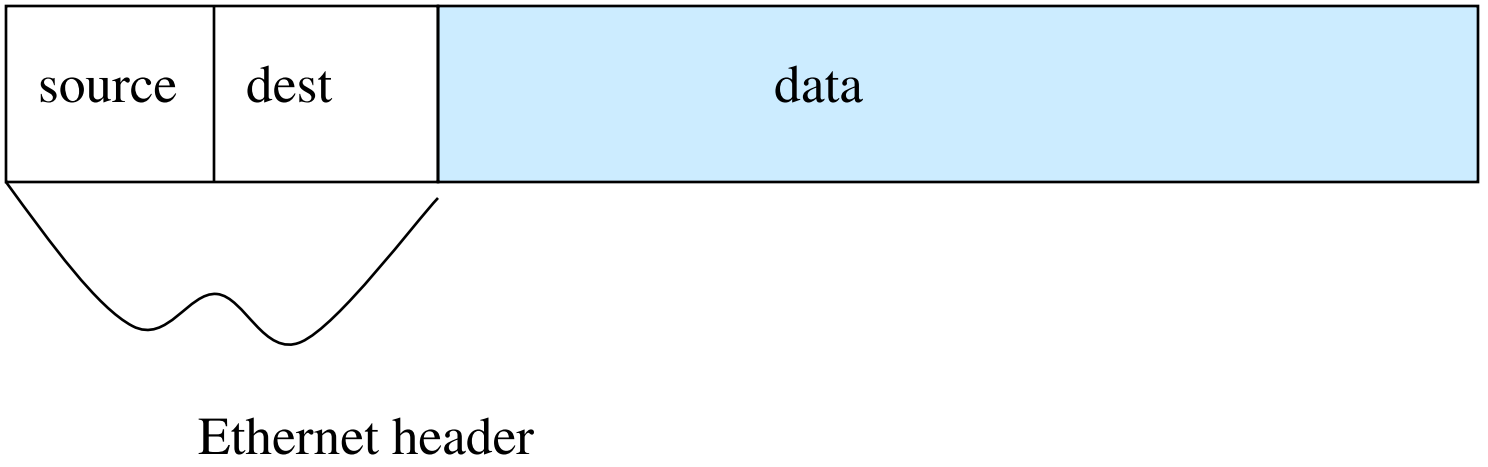
Layer 3 (e.g., IPv4, IPv6, DECnet, Appletalk, IPX, etc.)

- Put source, destination, hop count on packet
- Then along came "the EtherNET"
  - rethink routing algorithm a bit, but it's a link not a NET!
- The world got confused. Built on layer 2
- I tried to argue: "*But you might want to talk from one Ethernet to another*!"
- "*Which will win? Ethernet or DECnet*?"

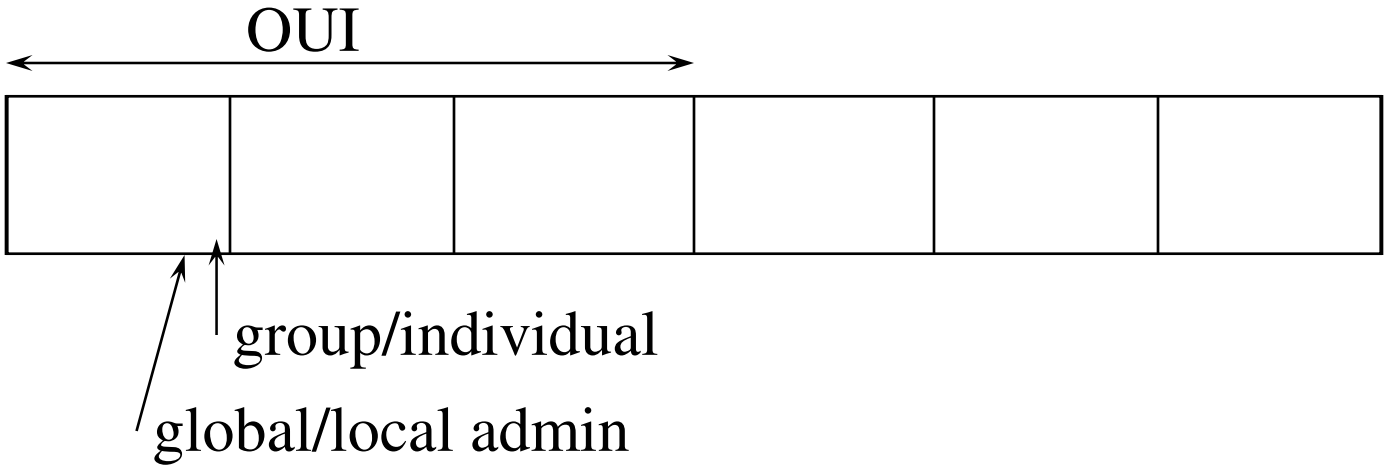
Layer 3 packet



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"

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)

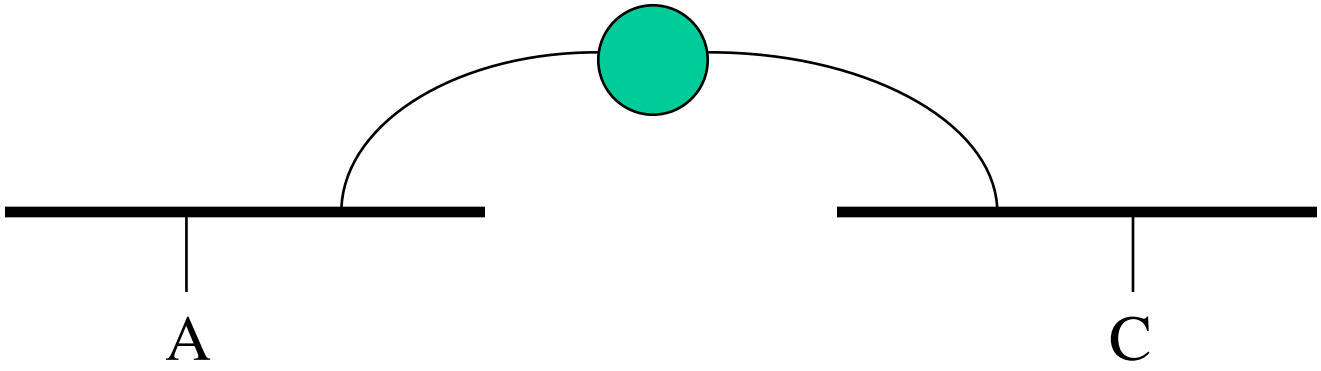
Horrible terminology

- Local area net
- Subnet
- Ethernet
- Internet

So where did bridges come from?

Problem Statement

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



Basic idea

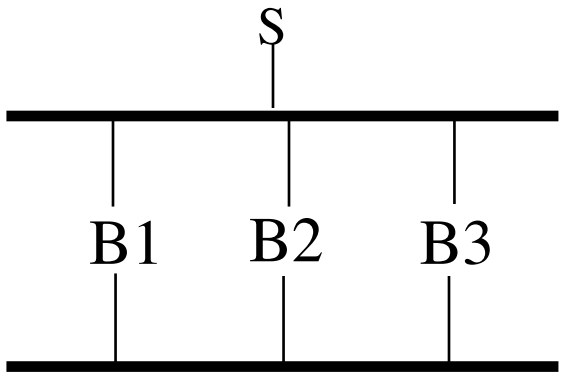
- Listen promiscuously
- Learn location of source address based on source address in packet and port from which packet received
- Forward based on learned location of destination

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)

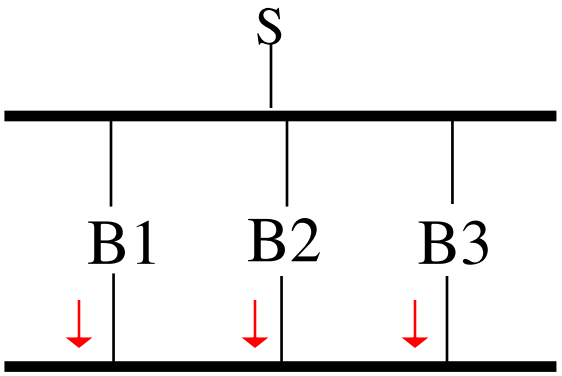
But loops are a disaster

- No hop count
- Exponential proliferation



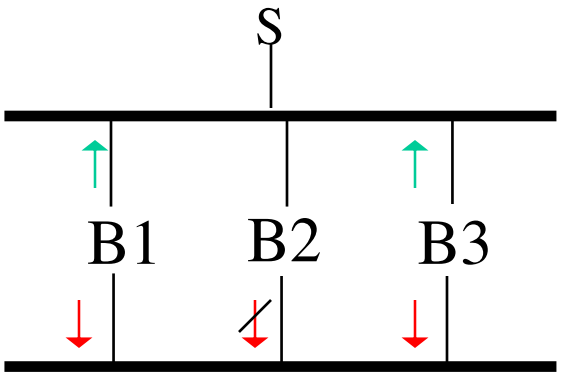
But loops are a disaster

- No hop count
- Exponential proliferation



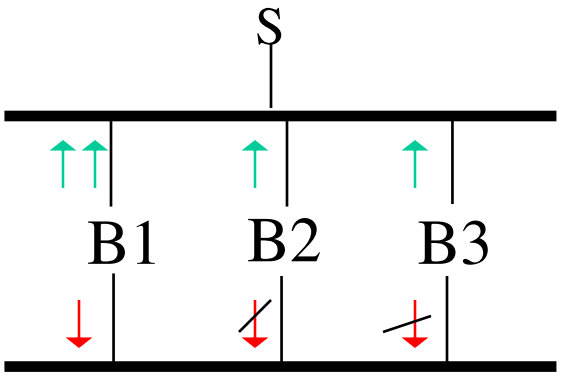
But loops are a disaster

- No hop count
- Exponential proliferation



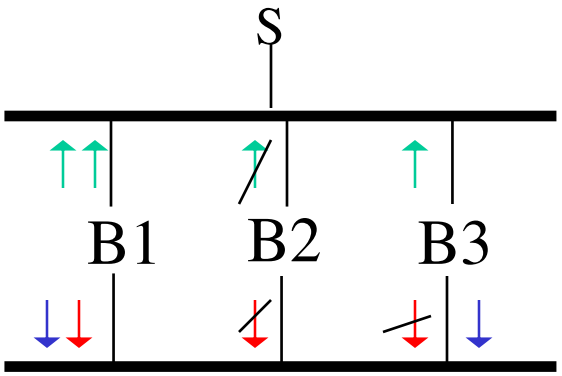
But loops are a disaster

- No hop count
- Exponential proliferation



But loops are a disaster

- No hop count
- Exponential proliferation



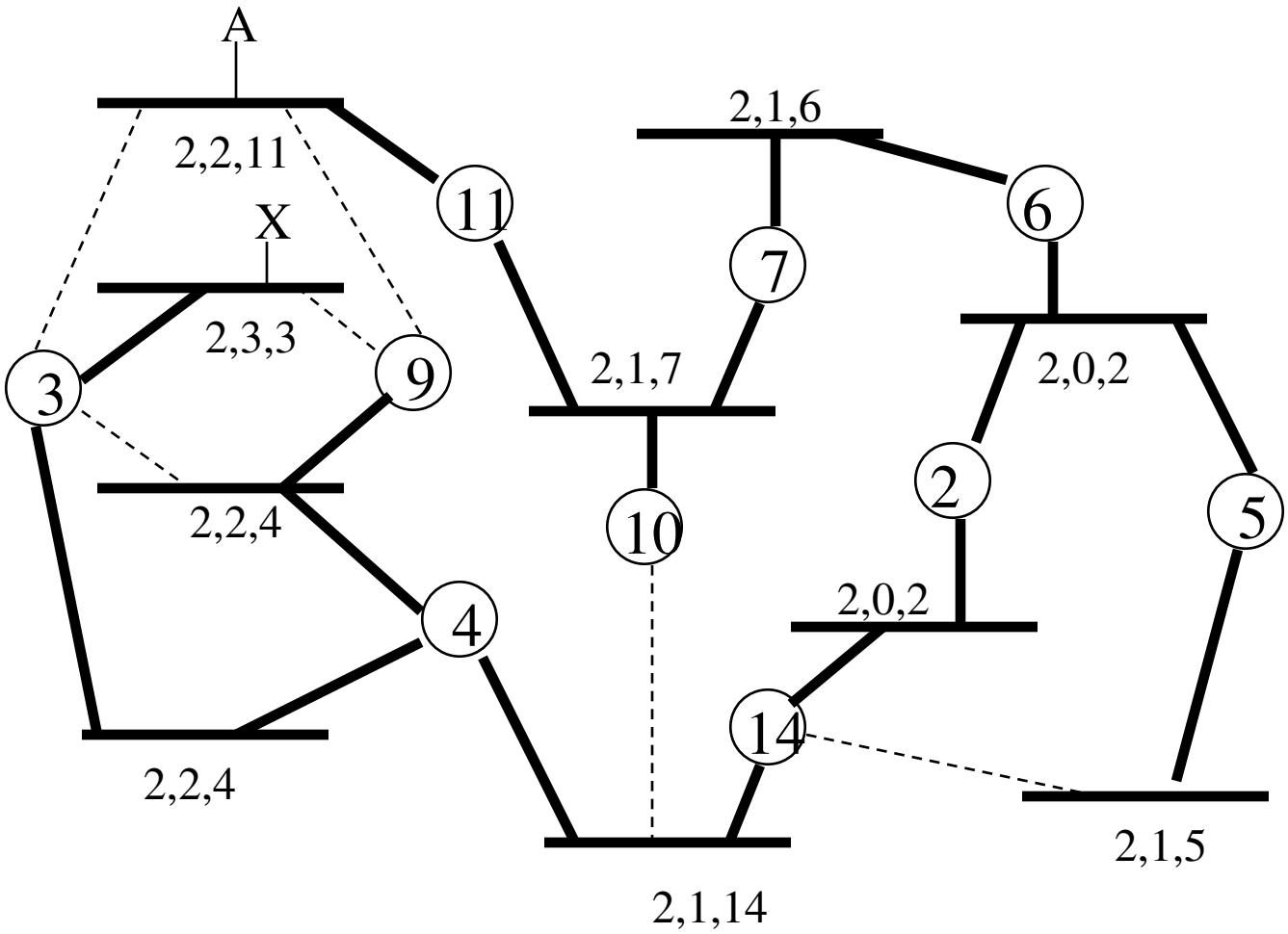
What to do about loops?

- 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

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.

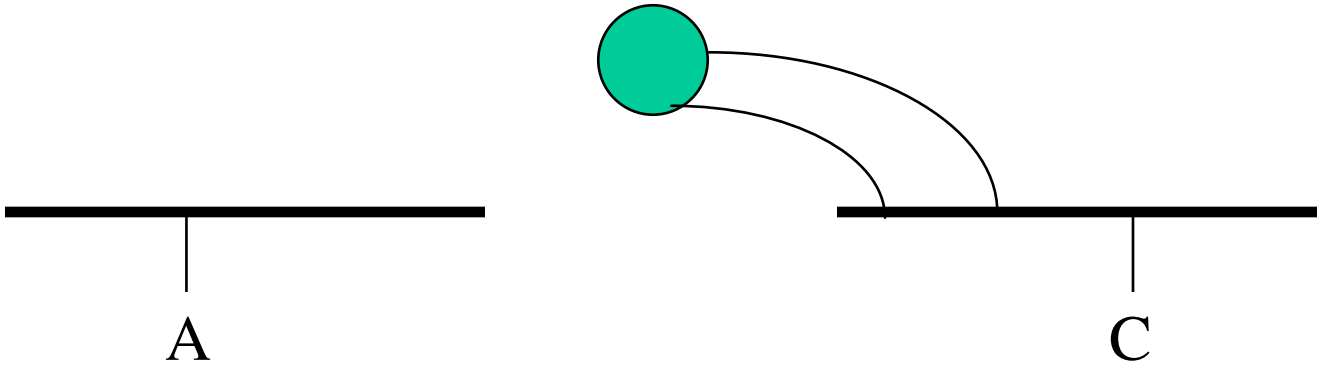
Radia Perlman


A
2,2,11
X
2,3,3
3
9
2,2,4
4
2,2,4
11
2,1,6
7
6
2,1,7
2,0,2
2
5
10
2,0,2
14
2,1,5
2,1,14

Bother with spanning tree?

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

First Bridge Sold



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

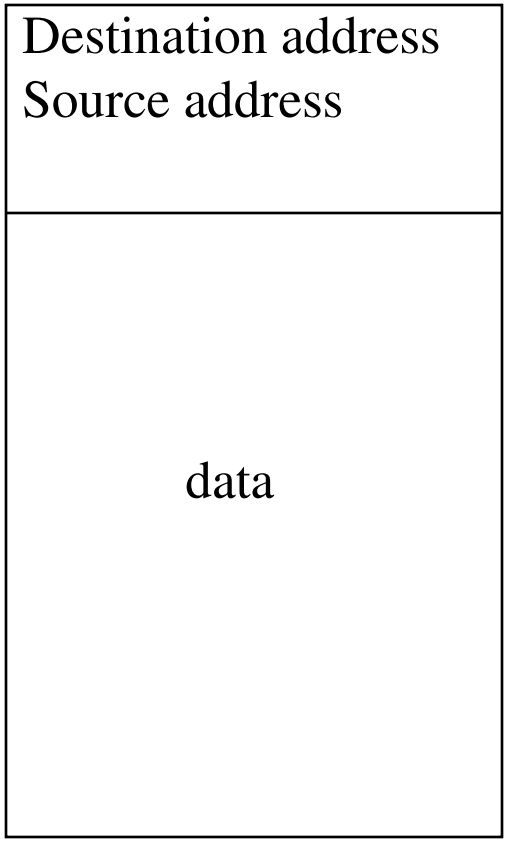
So what is Ethernet?

- CSMA/CD, right? Not any more, really...
- source, destination (and no hop count)
- limited distance, scalability (not any more, really)

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!

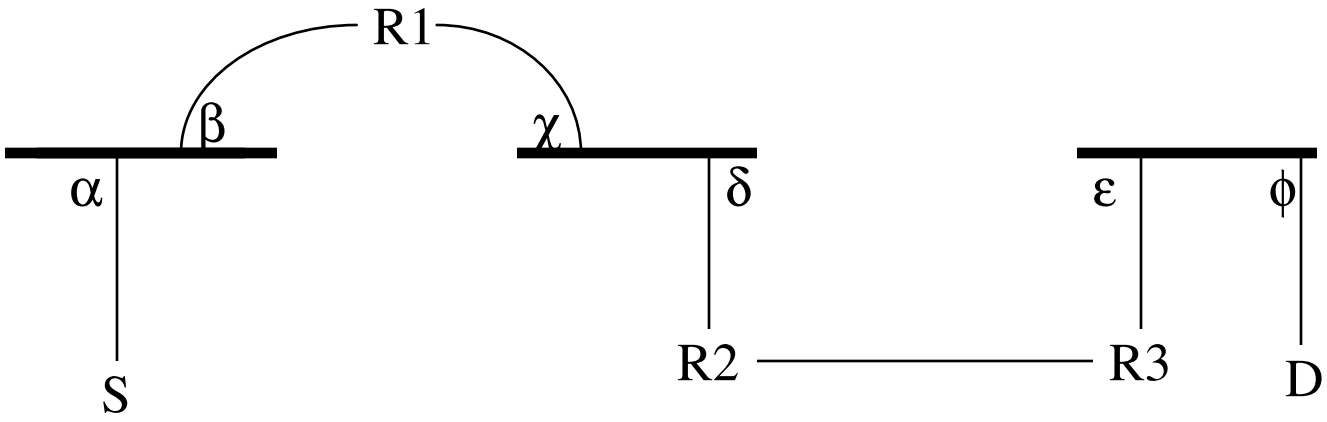
Basic idea of a packet



When I started

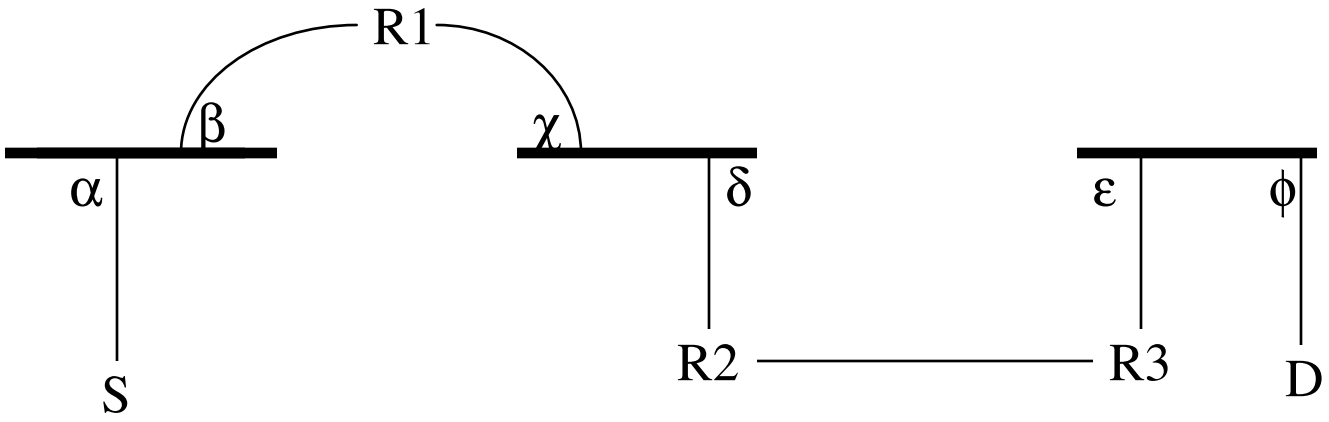
- Layer 3 had source, destination addresses
- Layer 2 was just point-to-point links (mostly)
- If layer 2 is multiaccess, then need two headers:
  - Layer 3 has ultimate source, destination
  - Layer 2 has next hop source, destination

Hdrs inside hdrs



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

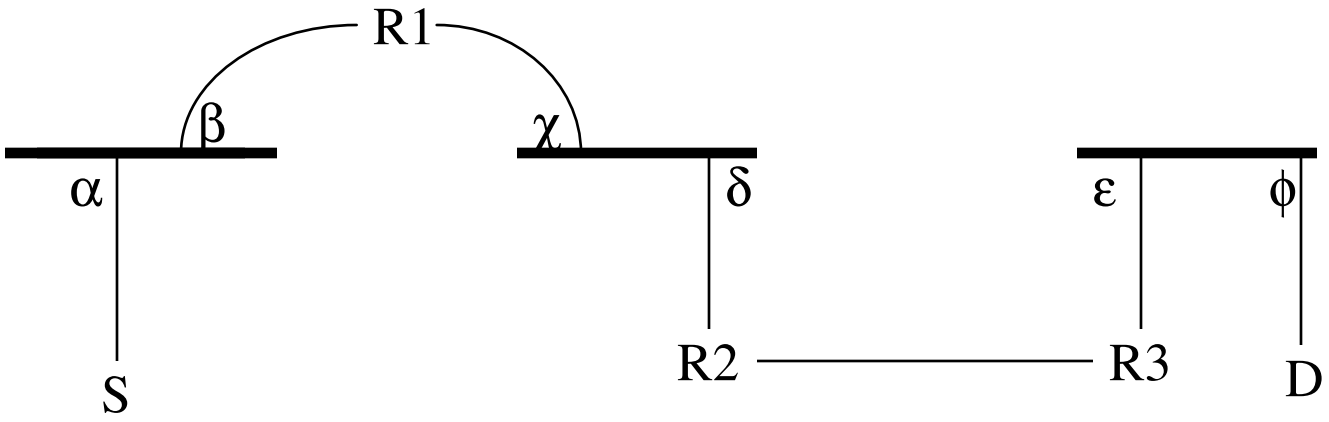
Hdrs inside hdrs



S:

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

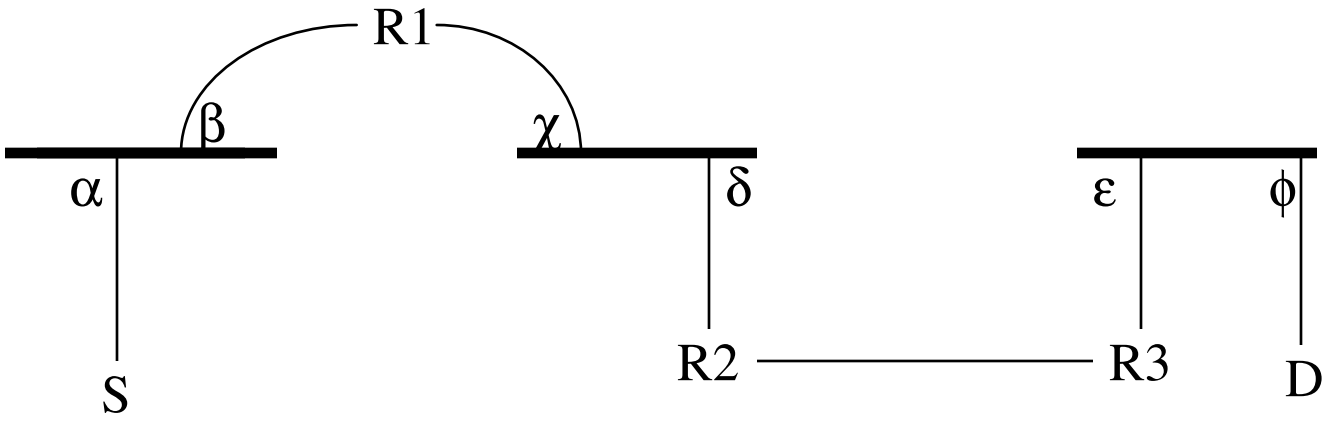
Hdrs inside hdrs

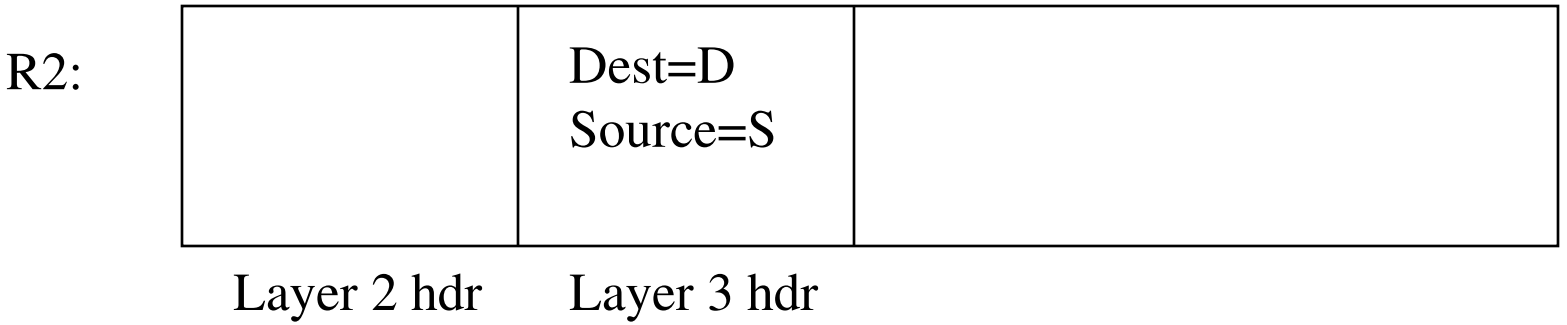


R1:

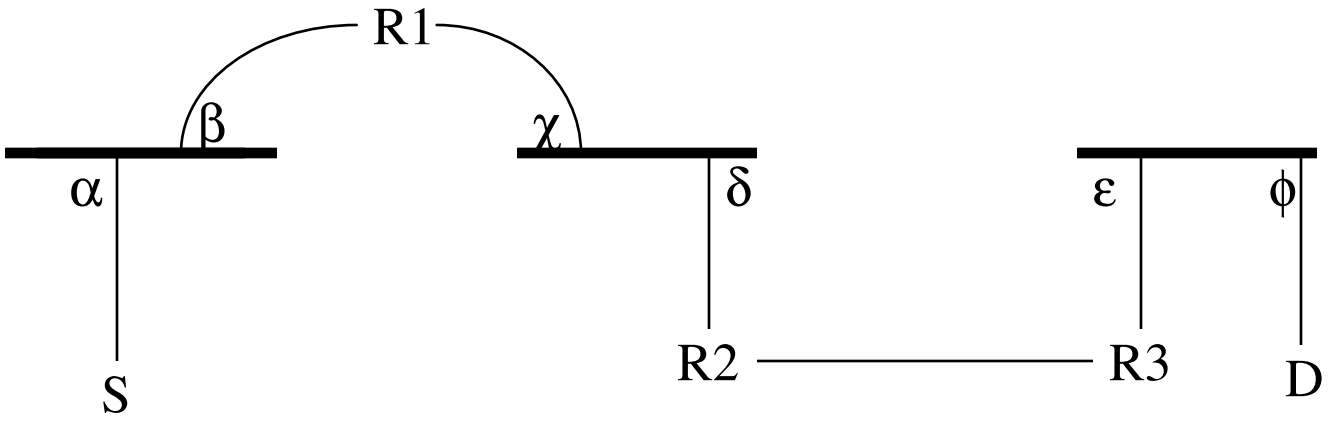
Dest=δ Source=χ	Dest=D Source=S	
Layer 2 hdr	Layer 3 hdr	

Hdrs inside hdrs





Hdrs inside hdrs



R3:

Dest=ϕ Source=ε	Dest=D Source=S	

Layer 2 hdr Layer 3 hdr

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

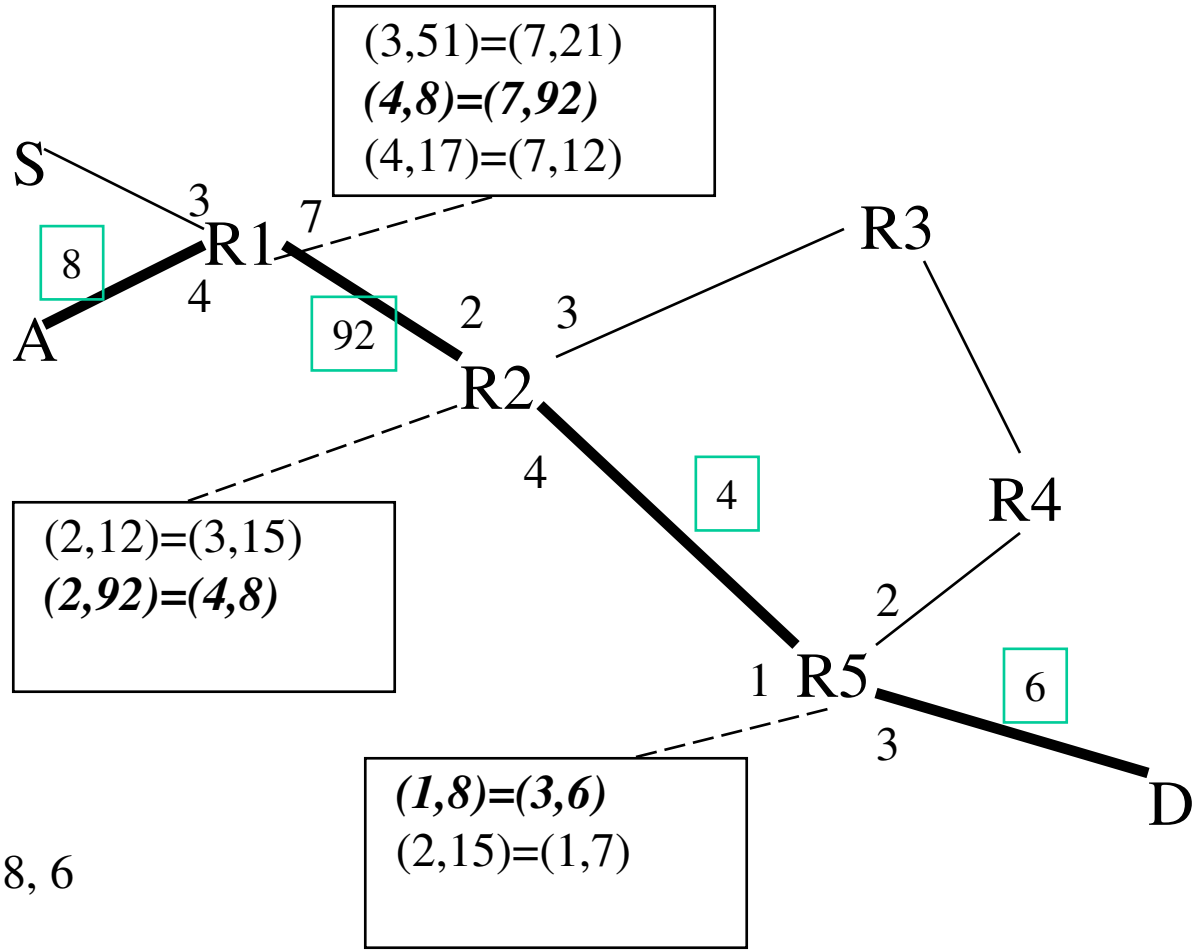
Network Layer

- 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

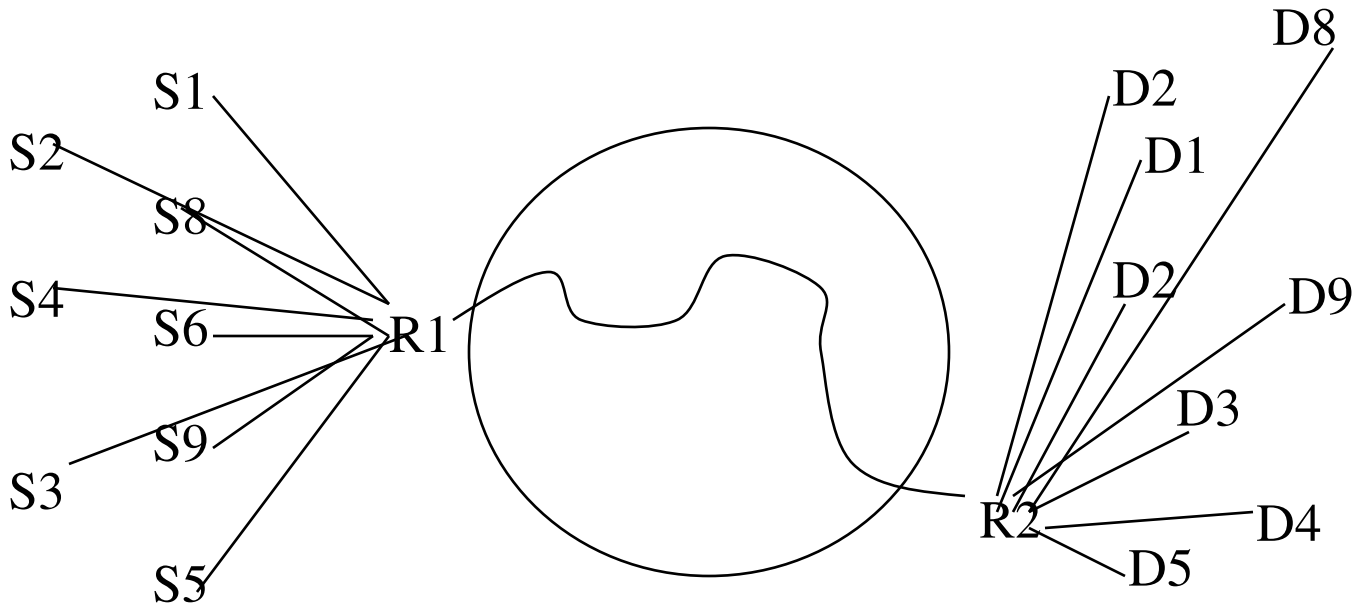
Lots of connection-oriented networks

- 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

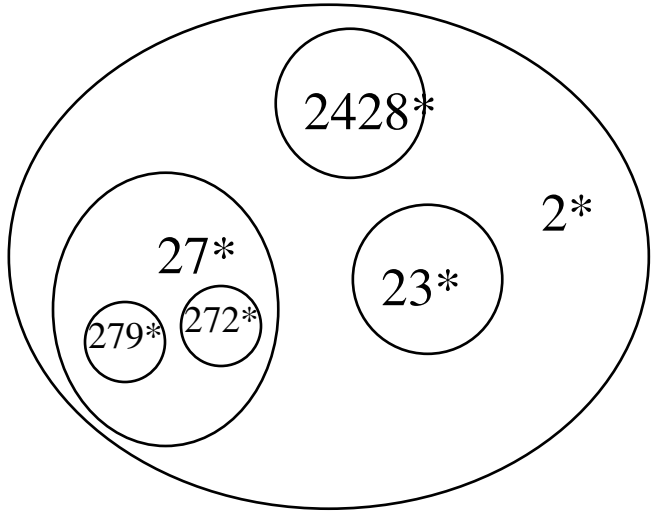
- Destination, source, hop count
- Maybe other stuff
  - fragmentation
  - options (e.g., source routing)
  - error reports
  - special service requests (priority, custom routes)
  - congestion indication
- Real diff: size of addresses

Addresses

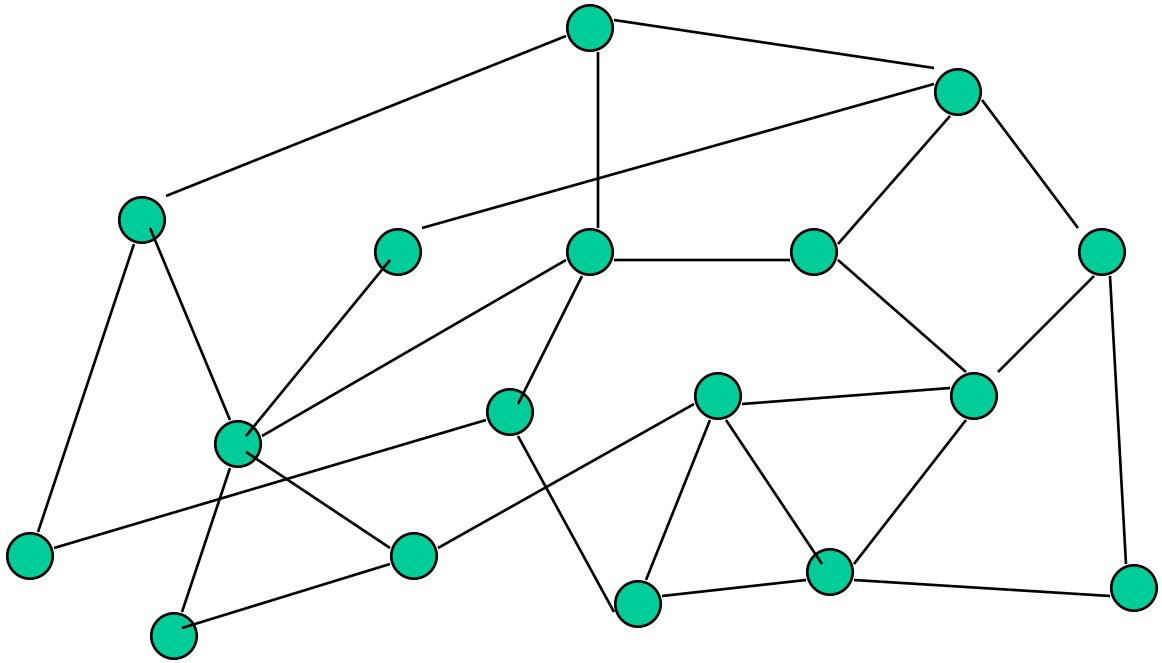
- 802 address “flat”, though assigned with OUI/rest. No topological significance
- layer 3 addresses: locator/node : topologically hierarchical address
- interesting difference:
  - IPv4, IPv6, IPX, AppleTalk: locator specific to a link
  - CLNP, DECnet: locator “area”, whole campus

Hierarchy 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

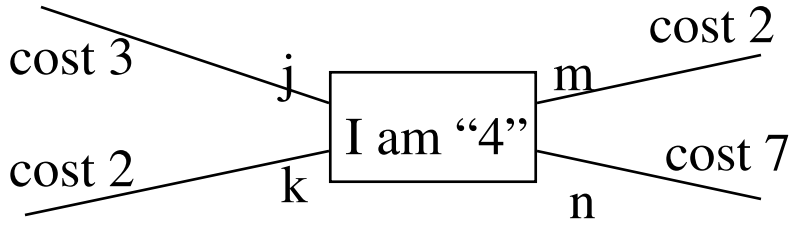


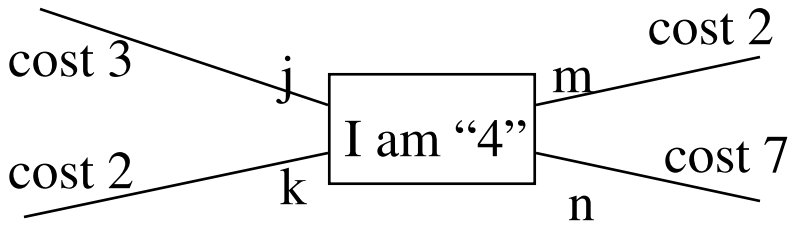
Distributed Routing Protocols

- 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

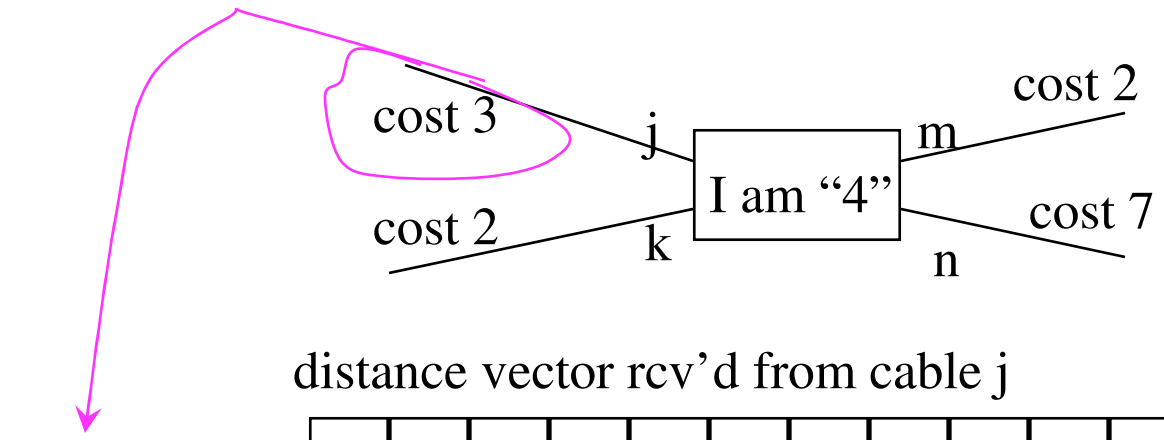
6	2	0	7	8	5	8	12	11	3	2

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

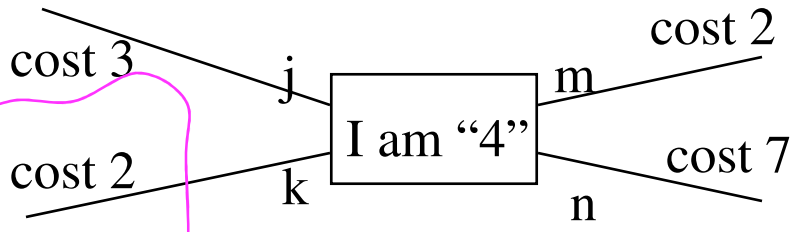
6	2	0	7	8	5	8	12	11	3	2

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

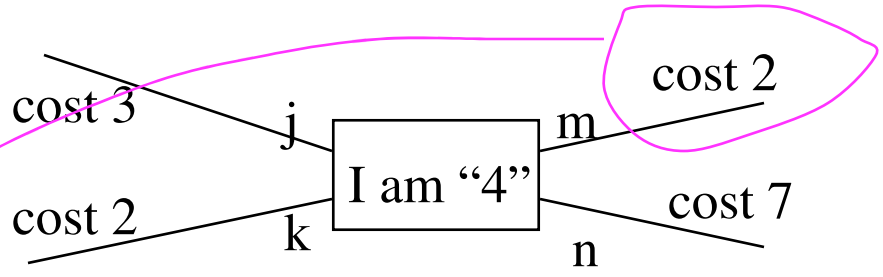
6	2	0	7	8	5	8	12	11	3	2

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

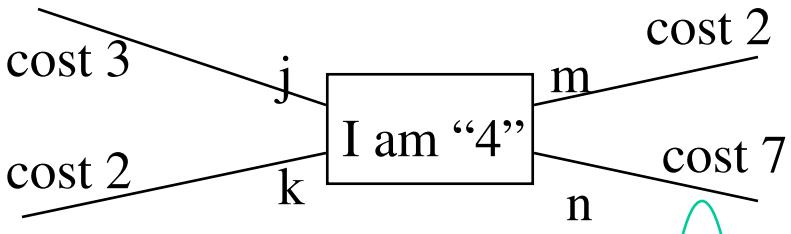
6	2	0	7	8	5	8	12	11	3	2

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

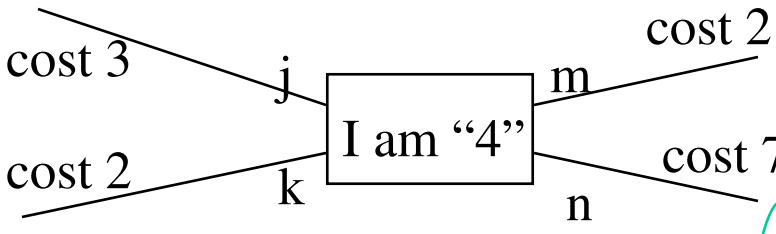
6	2	0	7	8	5	8	12	11	3	2

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

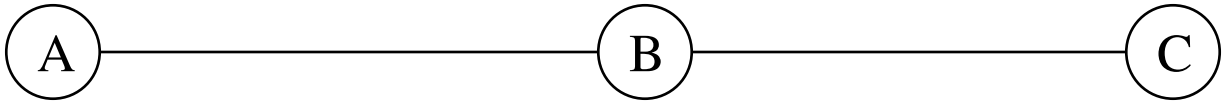
your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

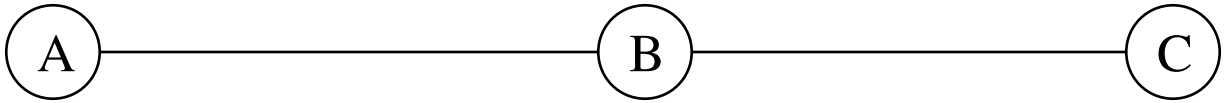
your own calculated forwarding table

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

Looping Problem

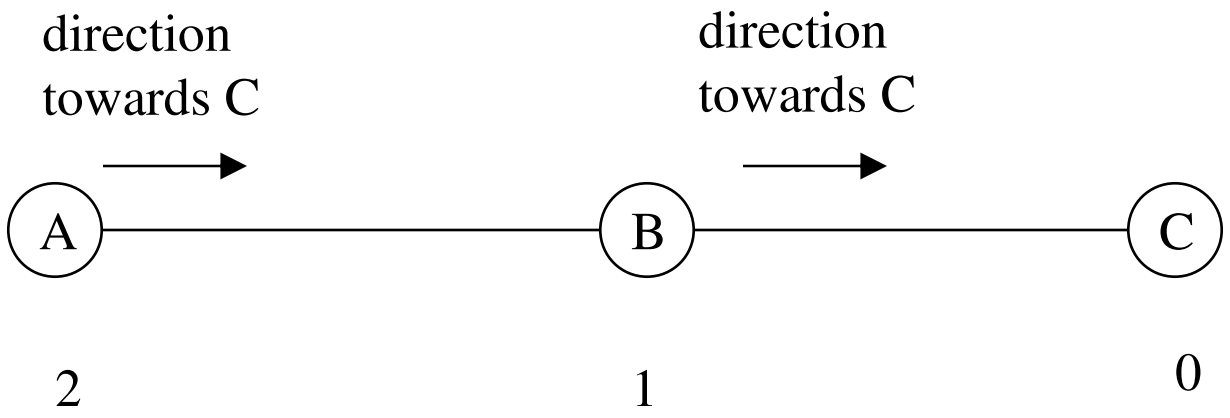


Looping Problem

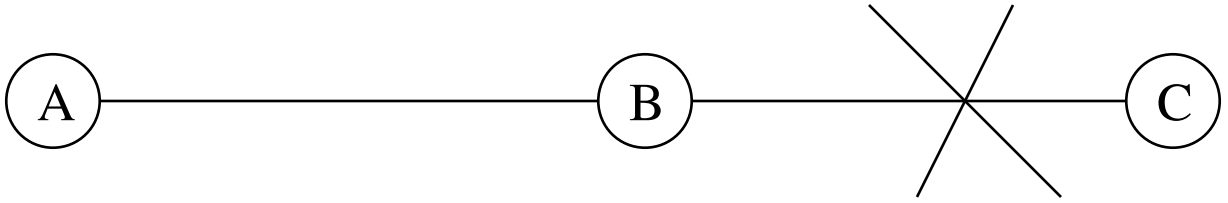


2 1 0 Cost to C

Looping Problem

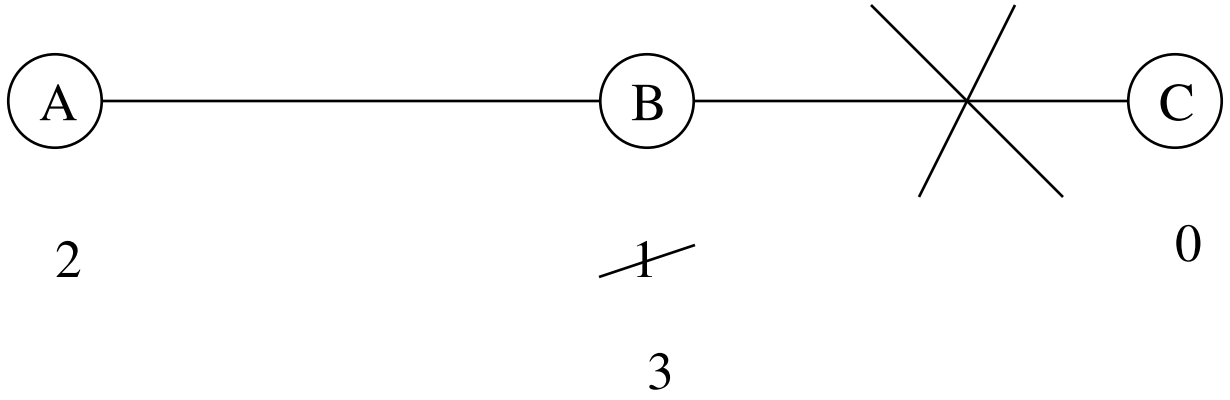


Looping Problem

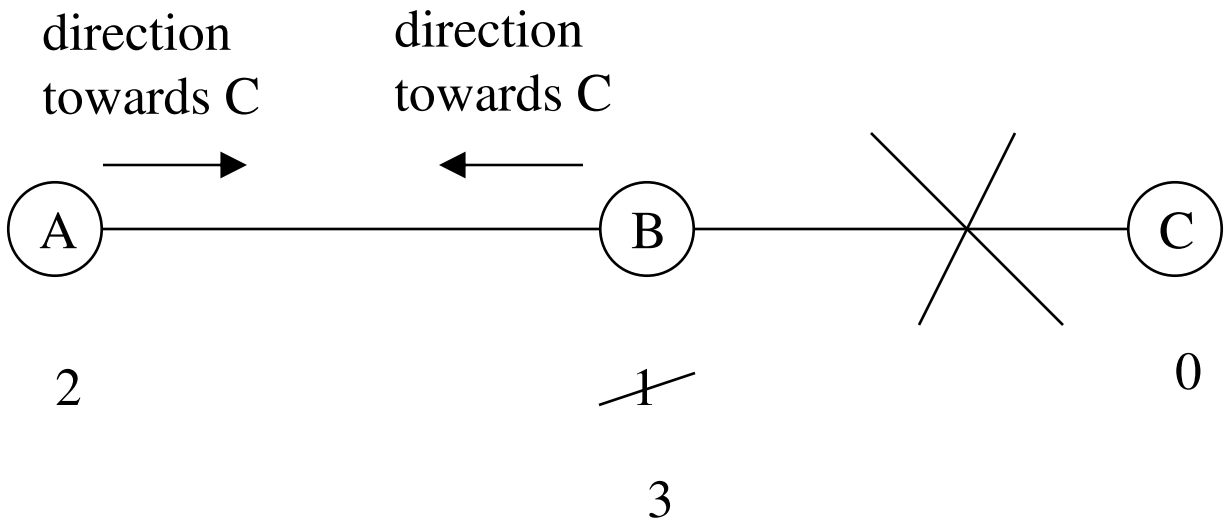


What is B's cost to C now?

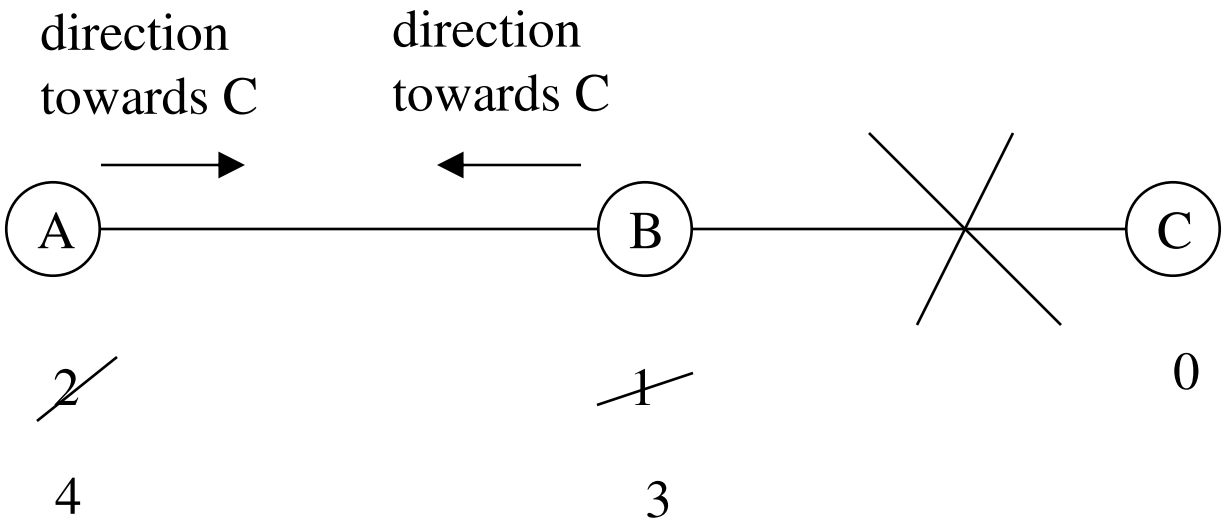
Looping Problem



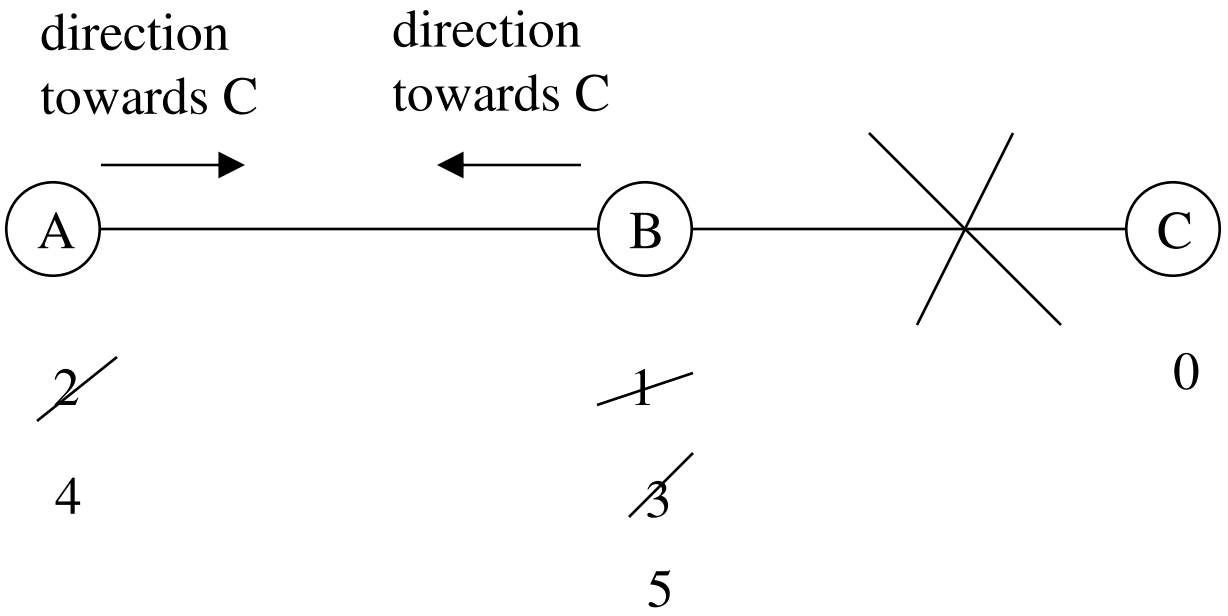
Looping Problem



Looping Problem



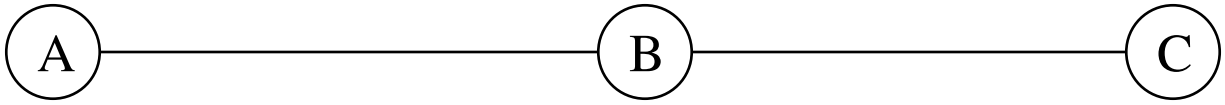
Looping Problem

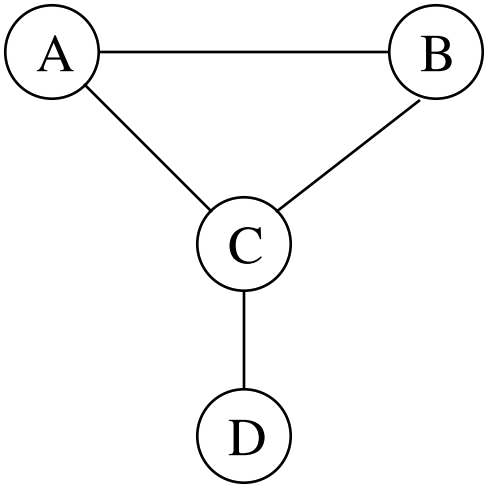


Looping Problem
worse with high connectivity

Split Horizon: one of several optimizations

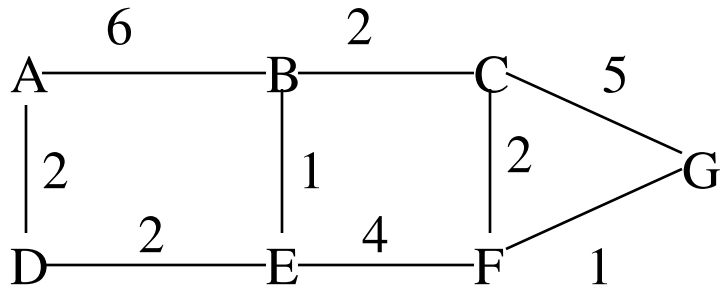
Don’t tell neighbor N you can reach D if you’d forward to D through N





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)



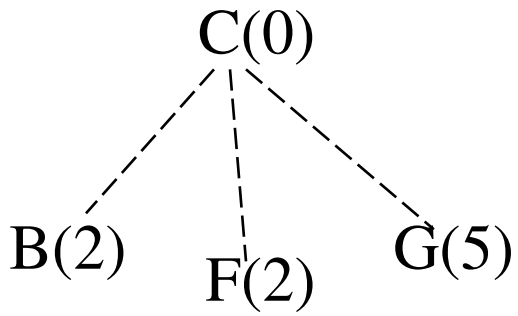
A	B	C	D	E	F	G
B/6	A/6	B/2	A/2	B/1	C/2	C/5
D/2	C/2	F/2	E/2	D/2	E/4	F/1
	E/1	G/5		F/4	G/1	

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

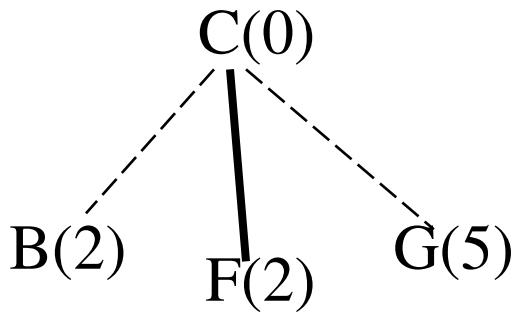
Look at LSP of new tree node

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	



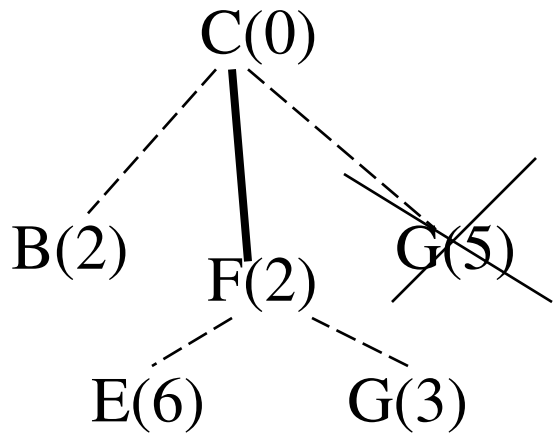
Make shortest TENT solid

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	



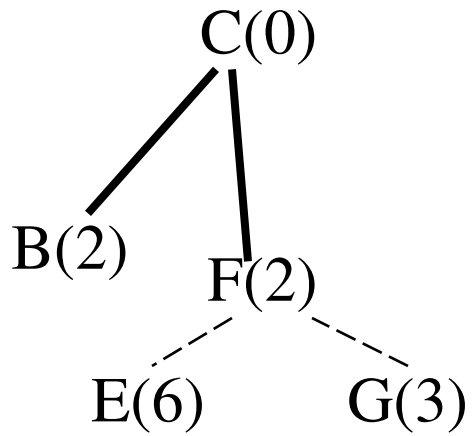
Look at LSP of newest tree node

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	



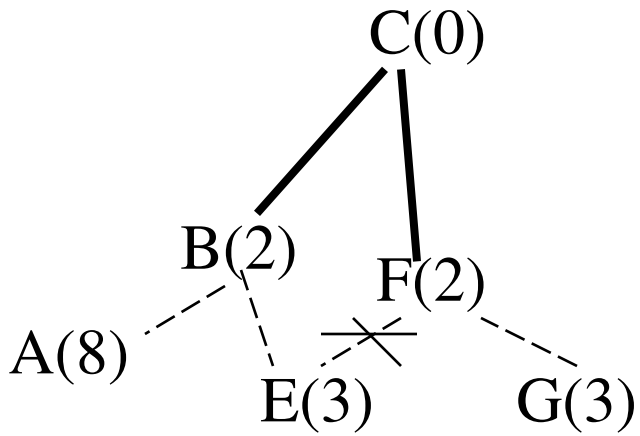
Make shortest TENT solid

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	



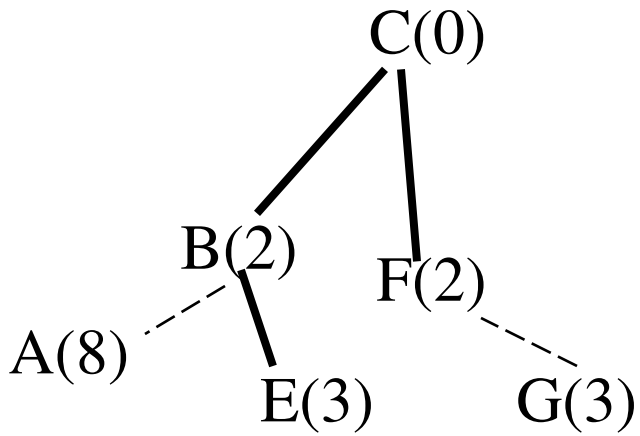
Look at LSP of newest tree node

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	



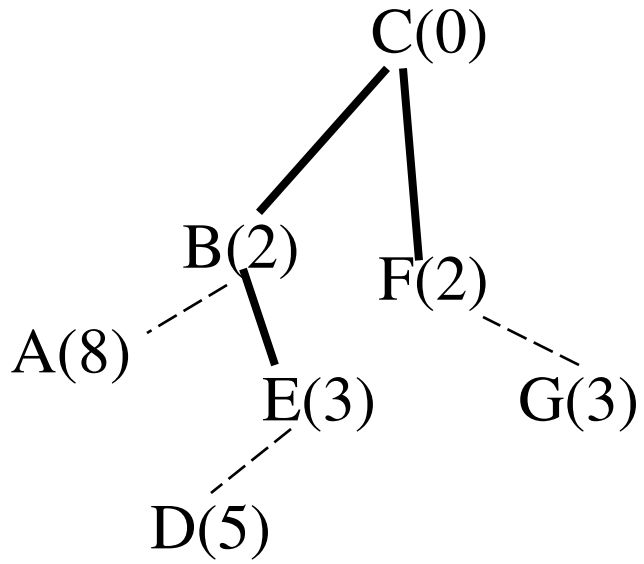
Make shortest TENT solid

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	



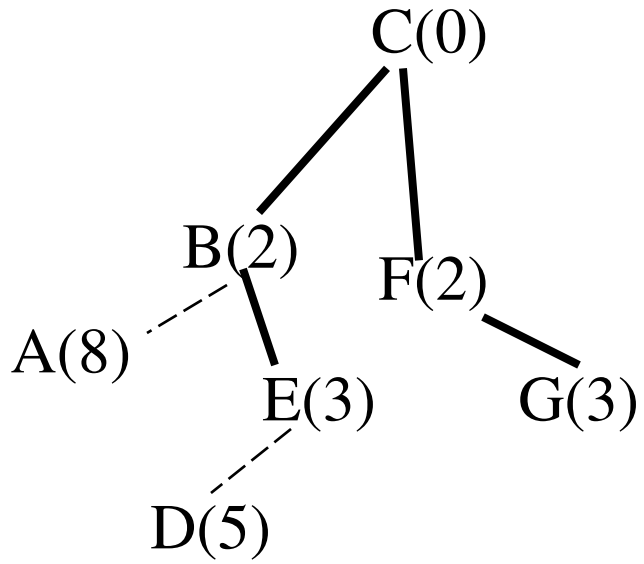
Look at LSP of newest tree node

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	



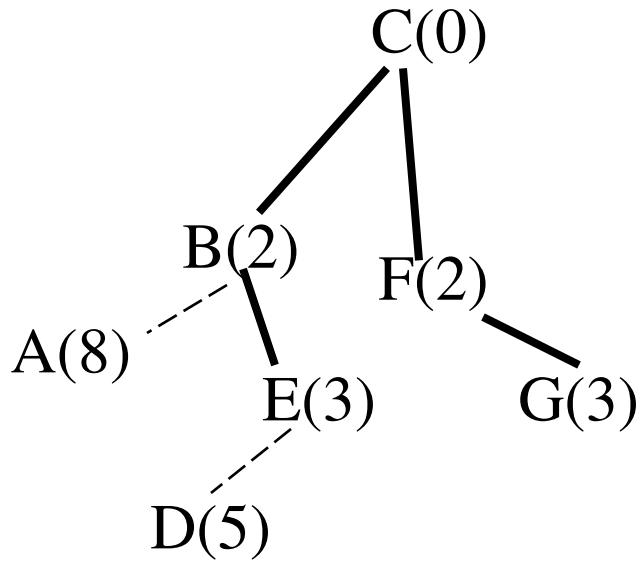
Make shortest TENT solid

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	



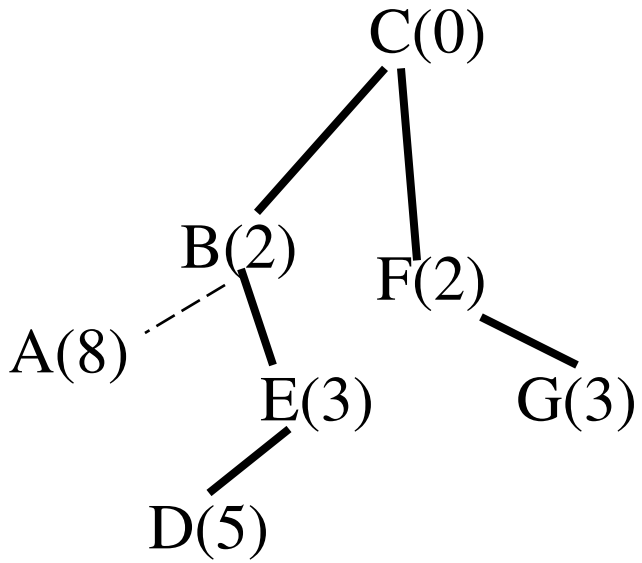
Look at newest tree node's LSP

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	



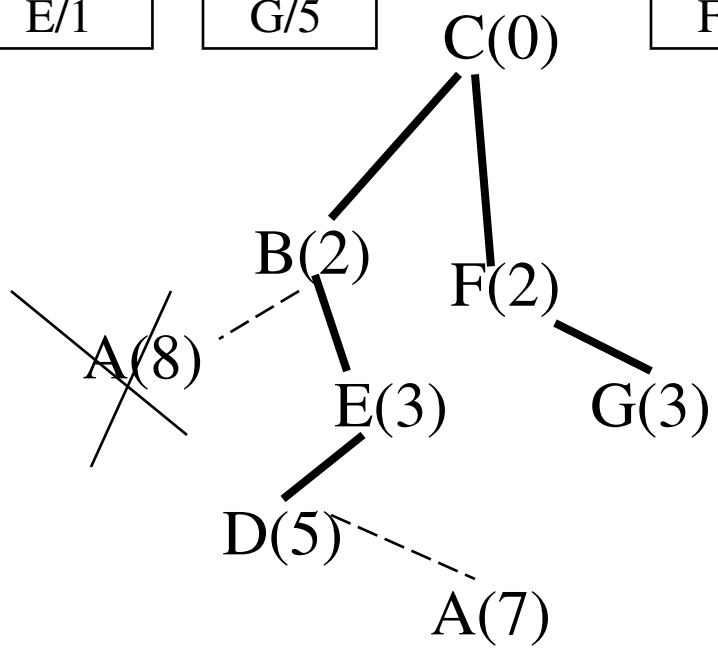
Make shortest TENT solid

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	



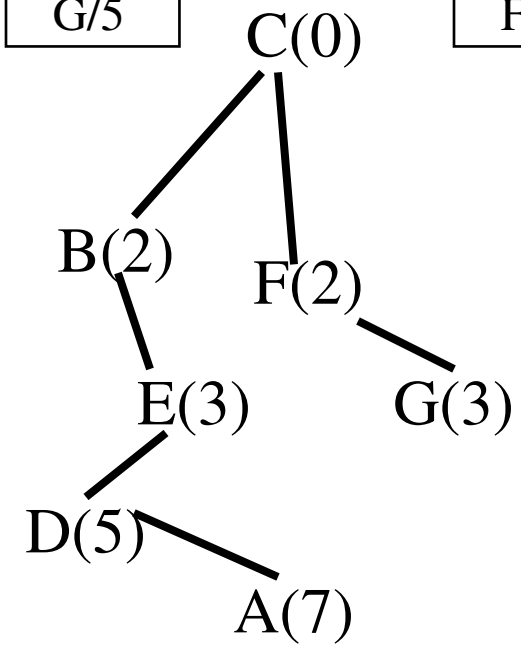
Look at newest node's LSP

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	



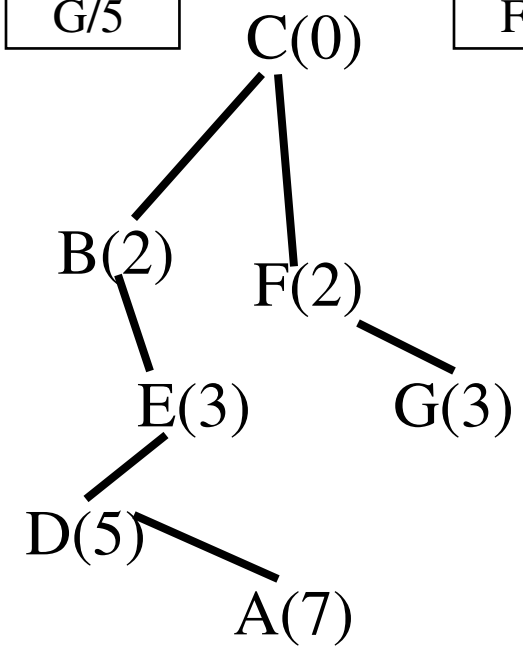
Make shortest TENT solid

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	



Not quite: need forwarding table

- So everything in subtree gets forwarded through that port

“A miracle occurs”

- First link state protocol: ARPANET
- I wanted to do something similar for DECnet
- My manager said “Only if you can prove it’s stable”
- Given a choice between a proof and a counterexample…

Routing Robustness

- I showed how to make link state distribution "self-stabilizing"…but only after the sick or evil node was disconnected
- Later, my thesis was on how to make the routing infrastructure (not just the routing protocol), robust while sick and evil nodes are participating…and it's not that hard

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

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)

Bridges are cool, but…

- Routes are not optimal (spanning tree)
  - STA cuts off redundant paths
  - If A and B are on opposite side of path, they have to take long detour path
- Temporary loops really dangerous
  - no hop count in header
  - proliferation of copies during loops
- Traffic concentration on selected links

Bridge meltdowns

- They do occur (a Boston hospital)
- Lack of receipt of spanning tree msgs tells bridge to turn on link
- So if too much traffic causes spanning tree messages to get lost…
  - loops
  - exponential proliferation of looping packets

What we'd like, part 1: replace bridging with Rbridging

- keep transparency to endnodes
- keep plug-and-play
- have best paths
- eliminate problems with temporary loops
  - have a hop count
  - don't exponentially proliferate packets
- then can converge optimistically (like rtrs)

What we'd like, part 2: true "level 1 routing" for IP

- allow plug-and-play campus sharing a prefix
- allow optimal routing
- don't require any endnode changes (e.g., implement ES-IS)
- Interwork with existing routers and bridges

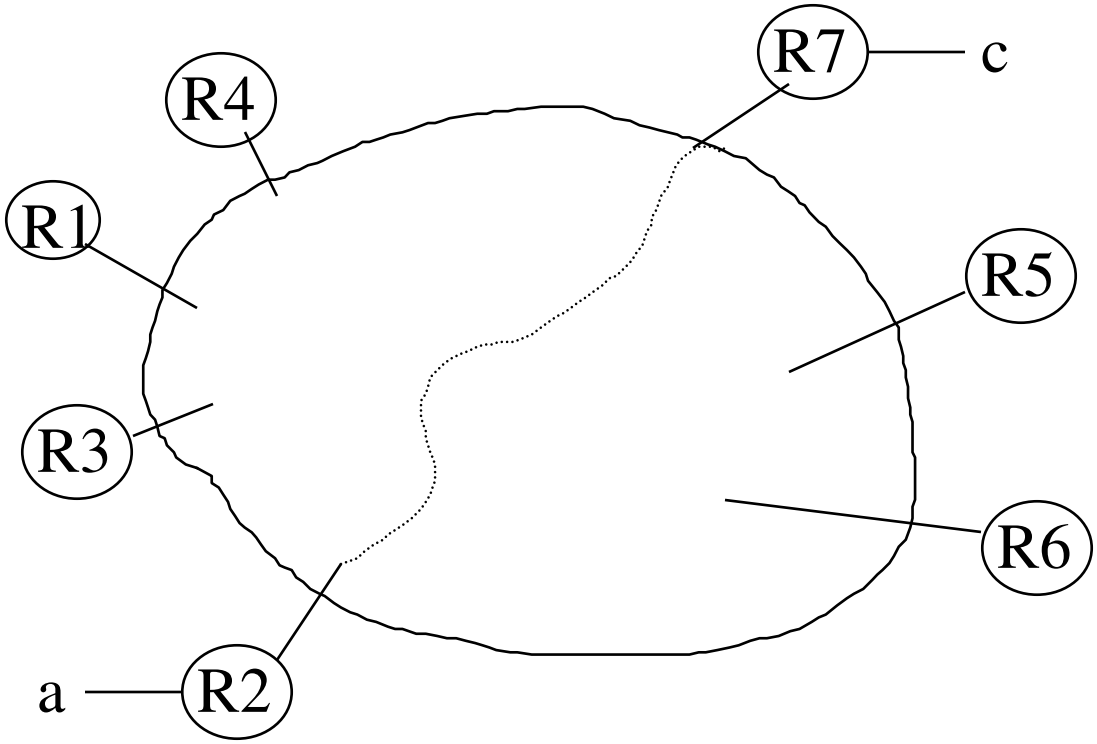
Rbridges

- Compatible with today's bridges and routers
- Like routers, terminate bridges' spanning tree
- Like bridges, glue LANs together to create one IP subnet (or for other protocols, a broadcast domain)
- Like routers, optimal paths, fast convergence, no meltdowns
- Like bridges, plug-and-play

Rbridging layer 2

- Link state protocol among Rbridges (so know how to route to other Rbridges)
- Like bridges, learn location of endnodes from receiving data traffic
- But since traffic on optimal paths, need to distinguish originating traffic from transit
- So encapsulate packet

Rbridging



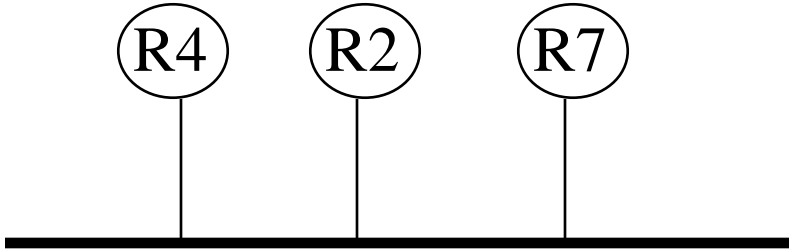
Encapsulation Header

S=Xmitting Rbridge D=Rcving Rbridge pt="transit"	hop count dest RBridge	original pkt (including L2 hdr)

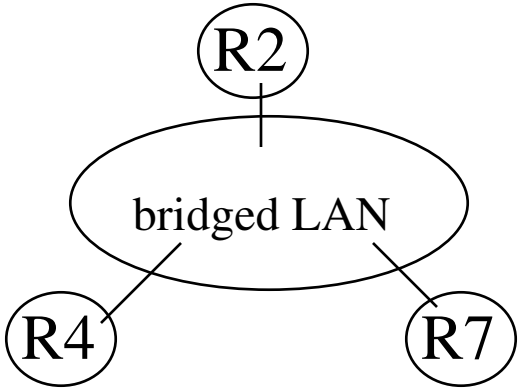
- Outer L2 hdr must not confuse bridges
- So it's just like it would be if the Rbridges were routers
- Need special layer 2 destination address
  - for unknown or multicast layer 2 destinations
  - can be L2 multicast, or any L2 address provided it never gets used as a source address

Rbridges and Bridges

Seems like:



Actually can be:



Endnode Learning

- On shared link, only one Rbridge (DR) can learn and decapsulate onto link
  - otherwise, a “naked” packet will look like the source is on that link
  - have election to choose which Rbridge
- When DR sees naked pkt from S, announces S in its link state info to other Rbridges

Pkt Forwarding: Ingress RBridge

- If D known: look up egress RBridge R2, encapsulate, and forward towards R2
- Else, send to “destination=flood”, meaning send on spanning tree
  - calculated from LS info, not sep protocol
  - each DR decapsulates

Calculating spanning tree

- No reason to have additional protocol to calculate a spanning tree
- The link state database gives enough information for RBridges to deterministically compute a spanning tree
  - Do it from viewpoint of lowest ID RBridge
  - When tie in Dijkstra calculation, choose parent with lowest ID

Possible IP optimization: proxy ARP

- For IP, learn (layer 3, layer 2) from ARP (ND) replies
- Pass around (layer 3, layer 2) pairs in LSP info
- Local RBridge can proxy ARP (i.e., answer ARP reply) if target (layer 3, layer 2) known

Possible IP optimization: tighter aliveness check

- Can check aliveness of attached IP endnodes by sending ARP query
- Can assume endnode alive, until you forward traffic to it, or until someone else claims that endnode

VLANs

- VLAN is a broadcast domain
- So a VLAN A packet must only be forwarded to VLAN A links
- RBridges must announce which VLANs they connect to
- RBridges must be able to flood a VLAN A pkt to just VLAN A links
  - could do it with one spanning tree, and just not send on non-A links
  - or one spanning tree, and filter if no A-links downstream
  - or per-VLAN spanning tree

VLANs

- VLAN A endnodes only need to be learned by RBridges attached to VLAN A
- All RBridges must be able to forward to any other RBridge
- Egress RBridge in the encapsulation header

Conclusions

- Looks to routers like a bridge
  - invisible, plug-and-play
- Looks to bridges like routers
  - terminates spanning tree, broadcast domain

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
- IKEv1, SSL, even IP, 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!