Evolution of Ethernet: CSMA/CD to TRILL

Radia Perlman
Intel Labs
radia.perlman@intel.com
radia@alum.mit.edu

Evolution of Ethernet: CSMA/CD to TRILL

And other Networking Topics

Radia Perlman
Intel Labs
radia.perlman@intel.com
radia@alum.mit.edu

Networking is really confusing

- What exactly is Ethernet?
- Why do we need both Ethernet and IP?
- What is this whole “layer 3 vs layer 2” thing about?

Perlman’s View of Network Layers

- Based on OSI layers…

Perlman's View of Network Layers

- Layer 1: Physical

Perlman's View of Network Layers

- Layer 1: Physical
- Layer 2: Data Link: Neighbor-neighbor

Perlman's View of Network Layers

- Layer 1: Physical
- Layer 2: Data Link: Neighbor-neighbor
- Layer 3: Network: create path, forward

Perlman’s View of Network Layers

- Layer 1: Physical
- Layer 2: Data Link: Neighbor-neighbor
- Layer 3: Network: create path, forward
- Layer 4: “Transport”: end-to-end reordering, error recovery

Perlman’s View of Network Layers

- Layer 1: Physical
- Layer 2: Data Link: Neighbor-neighbor
- Layer 3: Network: create path, forward
- Layer 4: “Transport”: end-to-end reordering, error recovery
- Layers 5 and above:

Perlman’s View of Network Layers

- Layer 1: Physical
- Layer 2: Data Link: Neighbor-neighbor
- Layer 3: Network: create path, forward
- Layer 4: “Transport”: end-to-end reordering, error recovery
- Layers 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*

Things I’ll talk about

- Addressing (hierarchical, flat)
- Switch forwarding tables based on
  - Destination address
    - Direct lookup
    - Hash
    - Longest prefix match
  - Path
- Creating forwarding tables (central, distributed)

Address Issues

- Name, ID, Address, Route

Address Issues

- Name, ID, Address, Route
  - Name: human-friendly, location-independent
  - ID: computer-friendly, location-independent
  - Address
    - If dest moves, address changes
    - But same address works from any source
  - Route
    - Dependent on location of source as well as dest!

Flat vs Hierarchical Addresses

- Flat: address doesn’t change when you move (so I’d call it an ID, but oh well…)
- Hierarchical: something like
  - Planet, country, state, city
- Ethernet addresses are flat, IP addresses are hierarchical

So, what’s the difference between layer 2 and layer 3?

Original Ethernet Invention

- CSMA/CD
  - – CS: carrier sense
    - Don’t interrupt if someone’s talking!
  - – MA: multiple access
    - You are sharing the airwaves so be polite!
  - – CD: collision detect
    - If someone else starts talking while you are talking, stop talking—people can’t listen to multiple people talking at once!

CSMA/CD

- Lots of papers about limited “goodput” due to collisions
- Limited scalability (distance, number of stations)

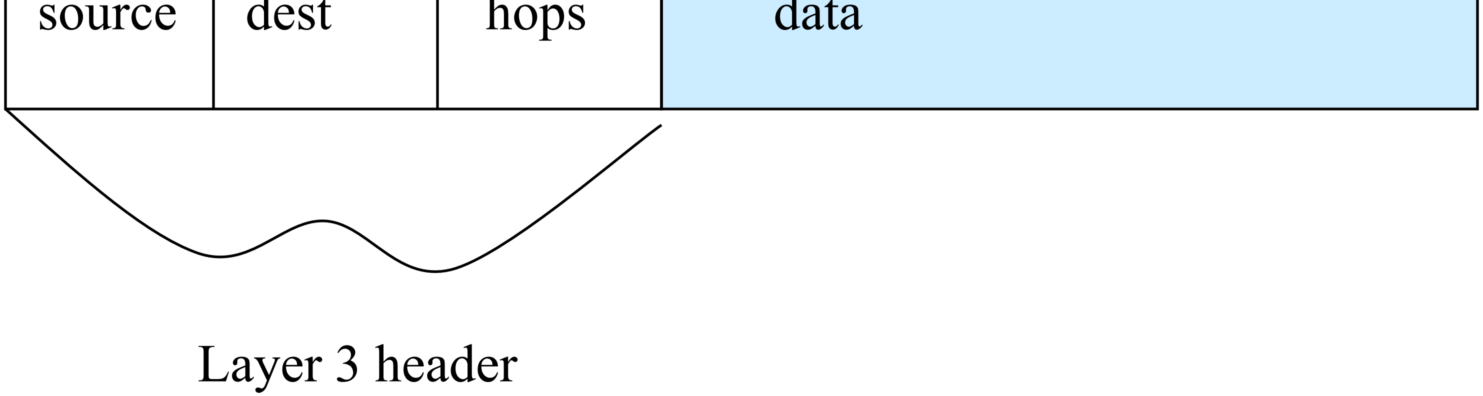
CSMA/CD

- Lots of papers about limited “goodput” due to collisions
- Limited scalability (distance, number of stations)
- But Ethernet hasn’t been CSMA/CD for years!

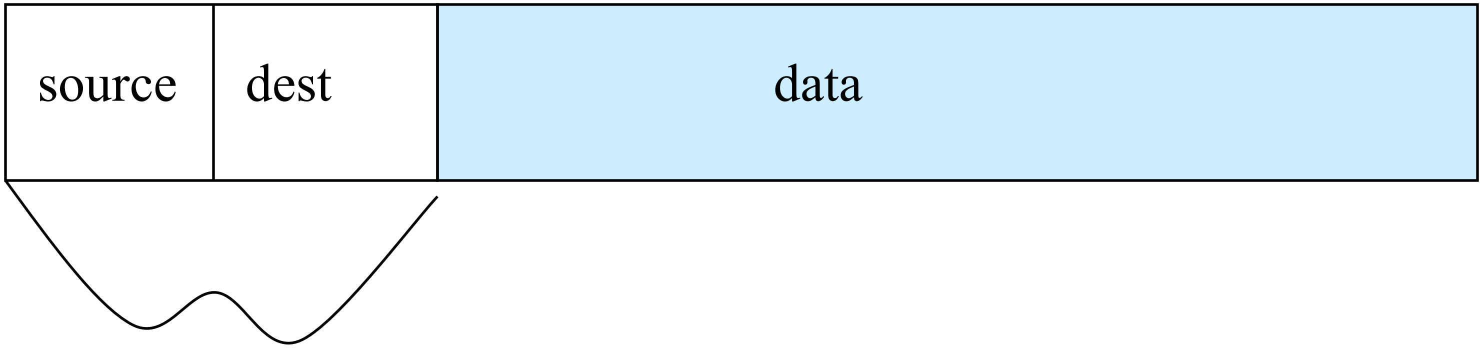
Layer 3 (e.g., IPv4, IPv6, DECnet, Appletalk, IPX, etc.)

- 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 header

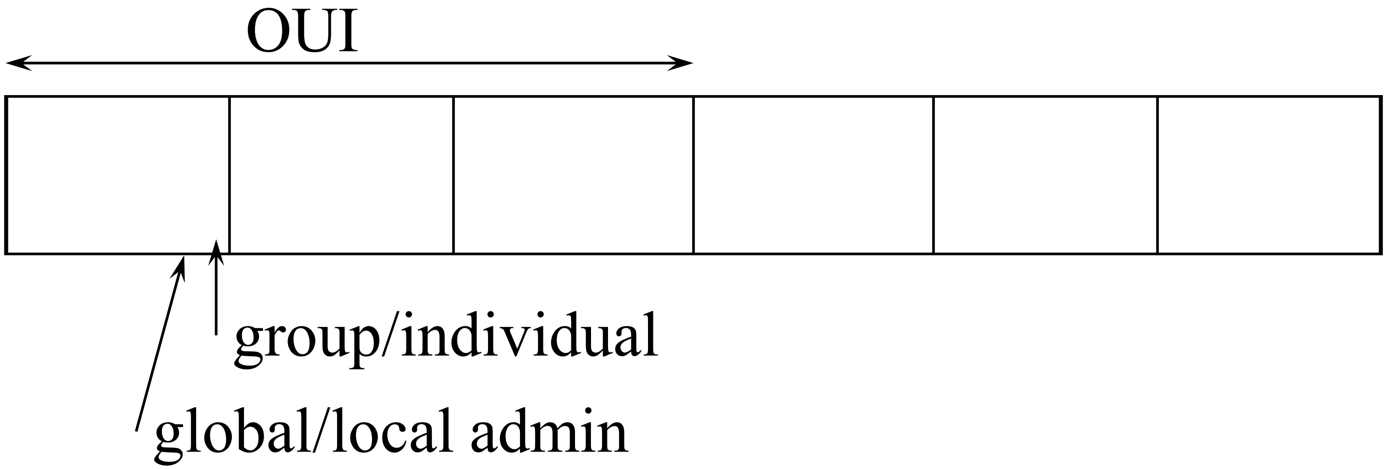
Autoconfiguration

- Ethernet philosophy: plug and play
- Worst part of configuration: addresses
- They wanted each device to have its own address
- Decided on 6 byte addresses, even though the technology as originally invented was only for connecting, say, 1000 nodes

Unique addresses

- Two proposals
  - Pick an address at random
  - Administer them centrally (now done by IEEE) and have manufacturer created devices with permanent addresses in ROM

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

- They look hierarchical
- But they are flat
- The hierarchy is only for ease of assignment

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)

So where did Ethernet bridges come from?

When I started

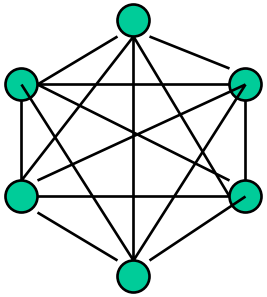
- Layer 3 had source, destination addresses
- Layer 2 was just point-to-point links (mostly)

Then…Ethernet

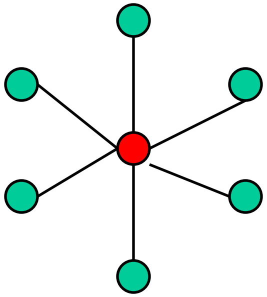
Ethernet…

- Should have been called “Etherlink”
- New kind of link, requiring some adjustment to the routing protocol, e.g.,
  - Routing algorithm overhead proportional to number of links
  - So, for link state routing, I created “pseudonodes”

Instead of:

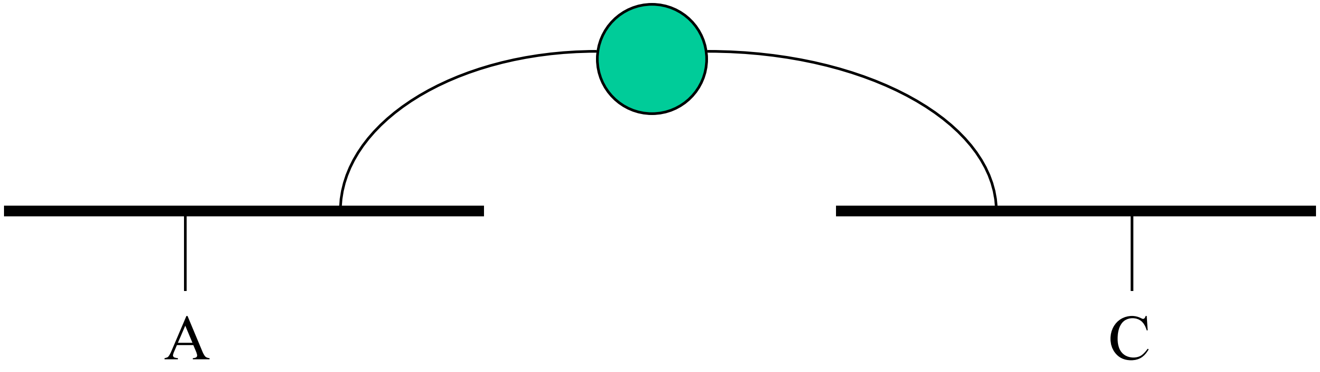


Use pseudonode



Problem Statement

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



Why routers won’t work

- Router knows about one layer 3 protocol
- And the endnode has to implement that!

Constraint at that time for “magic box at layer 2”

- Must not modify Ethernet packet in any way
- Hard limit on size of packet

Basic idea

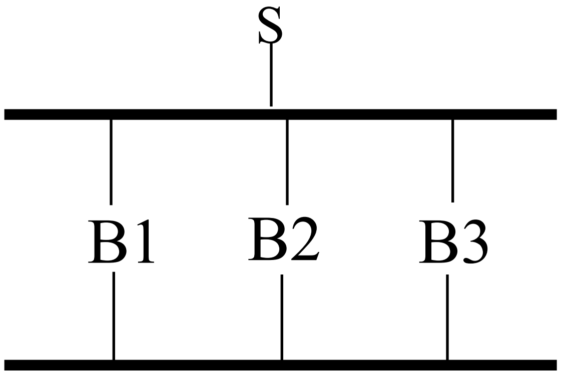
- Listen promiscuously
- Learn location of source address based on source address in packet and port from which packet received
- Forward based on learned location of destination

What’s different between this and a repeater?

- no collisions
- with learning, can use more aggregate bandwidth than on any one link
  - Repeater forwards immediately…can’t look at destination address before forwarding
- no artifacts of LAN technology (# of stations in ring, distance of CSMA/CD)

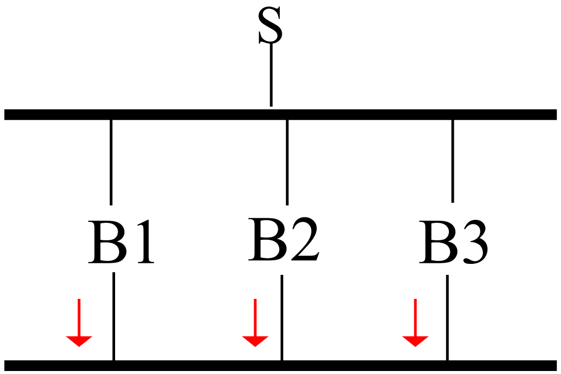
But loops are a disaster

- No hop count
- Exponential proliferation



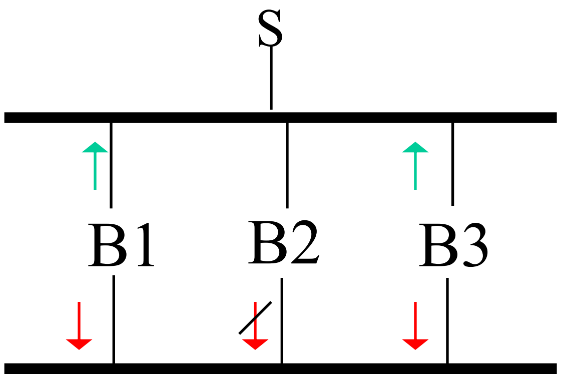
But loops are a disaster

- No hop count
- Exponential proliferation



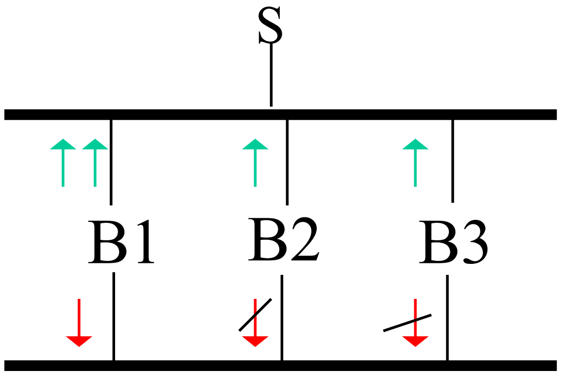
But loops are a disaster

- No hop count
- Exponential proliferation



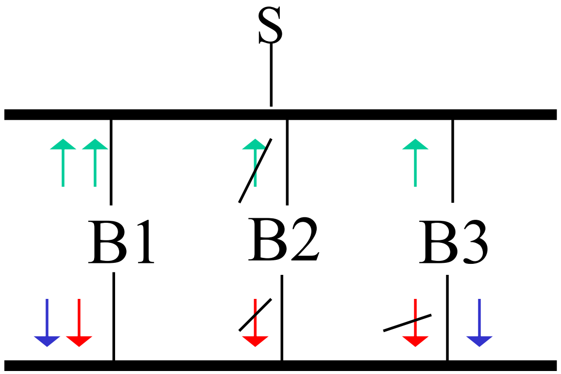
But loops are a disaster

- 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.

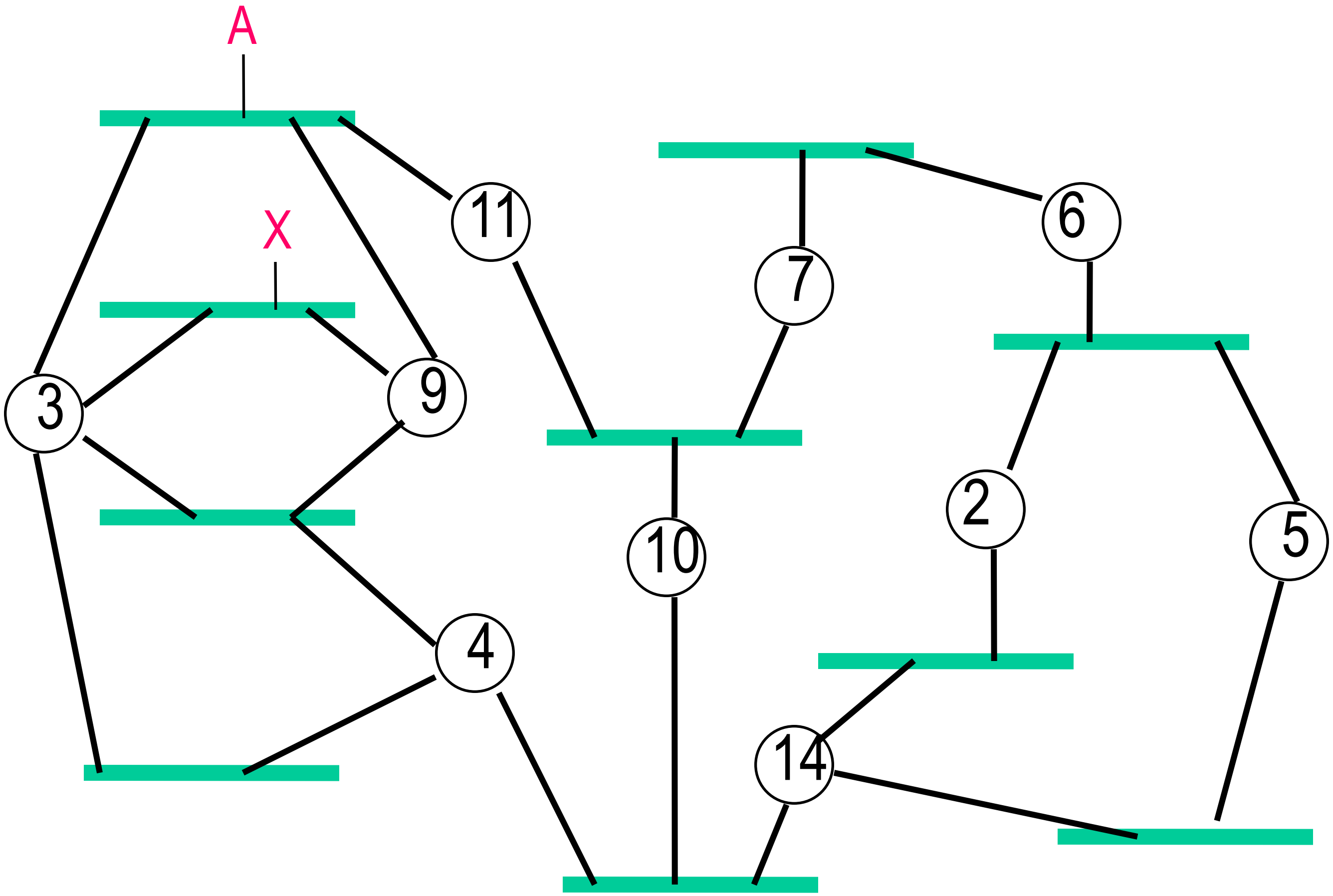
A tree which must be sure to span
So packets can reach every LAN.

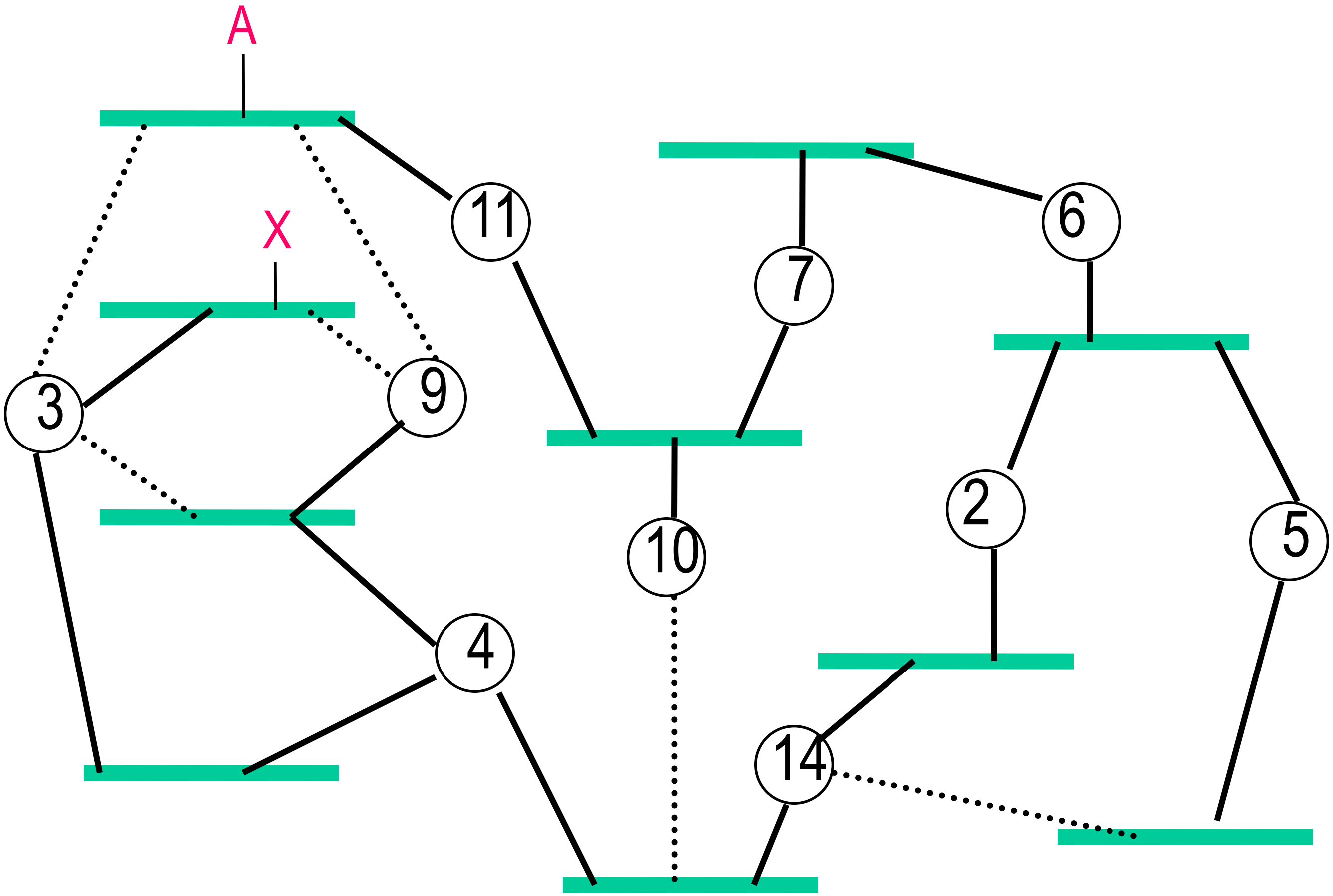
First the Root must be selected
By ID it is elected.

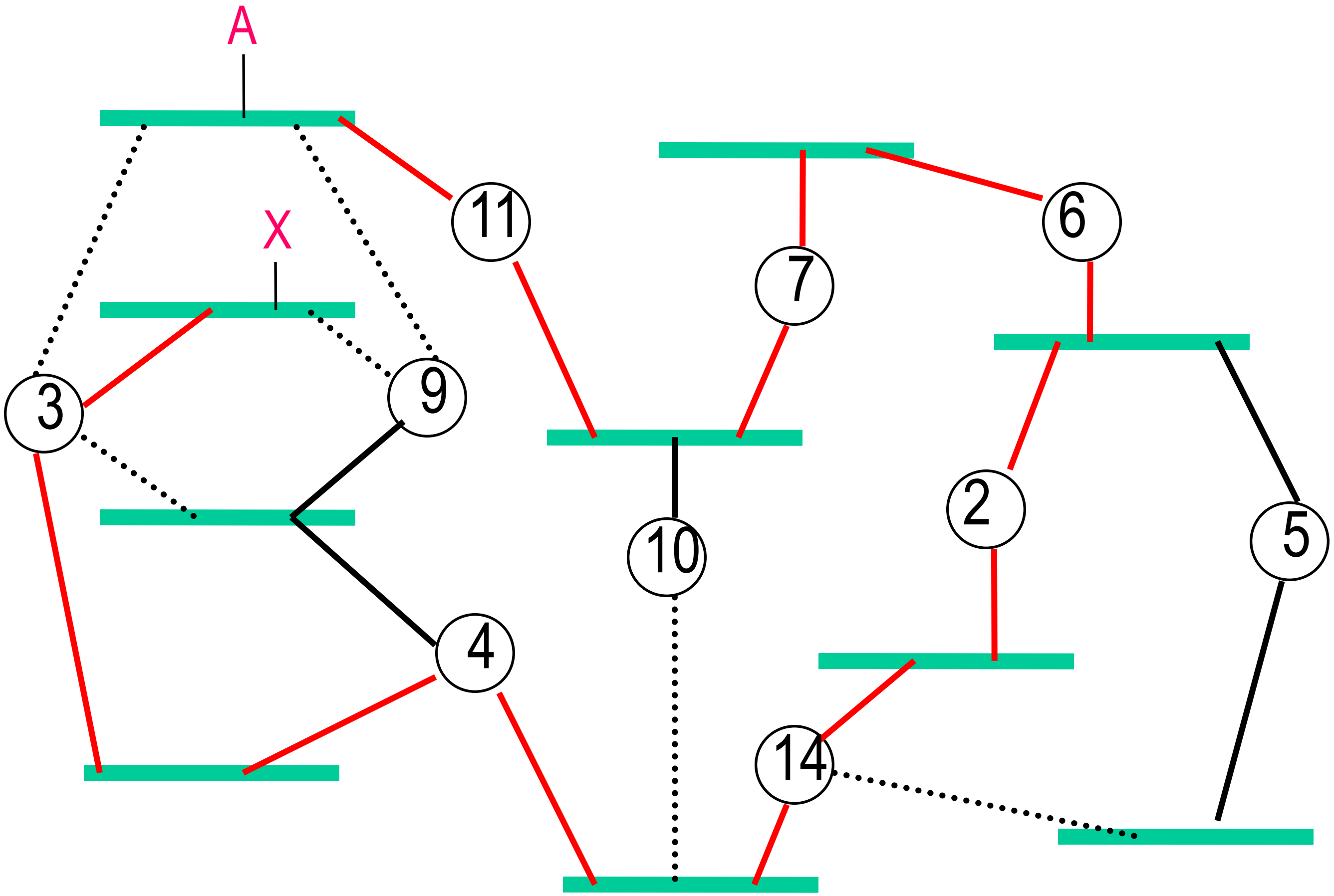
Least cost paths from Root are traced
In the tree these paths are placed.

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

Radia Perlman


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

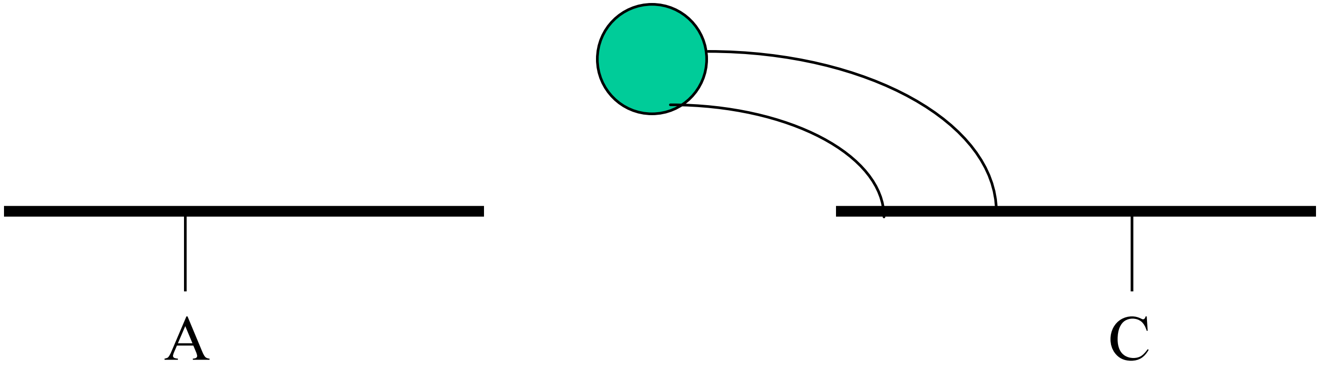

A
X
11
6
7
3
9
2
5
10
4
14


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

Bother with spanning tree?

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

First Bridge Sold



How spanning tree works

- Each bridge starts out with an ID (e.g., a MAC address it owns)
- Bridge B transmits spanning tree message:
  - ID of who B thinks is Root
  - Cost from B to Root
  - B’s ID
  - Other stuff (e.g., port, spanning tree parameters)

Remember “best” spanning tree message on each port p

- Best is numerically smallest
  - Root ID | cost to Root | ID of X’mitter | port ID
- If you are the Root, best is
  - Your ID | 0 | your ID | port ID
- So memory requirement for switch S to run spanning tree is only size of spanning tree message (about 50 bytes) * number of ports on S

Pick the Root

- Choose numerically smallest root ID from
  - Received spanning tree messages
  - Your own ID

Calculate your cost to Root

- 0 if you think you are the Root
- Else, smallest {cost of port p + cost reported by neighbor on that port}

Which ports are in the tree?

- Ports on which your spanning tree message is “best”
- Single one that is your best path to Root

Why is this a tree?

- Tree needs:
  - Unique Root
  - Every node (other than Root) needs unique parent
- Consider two types of nodes: links, and bridges
- Unique parent of link: bridge with best spanning tree message
- Unique parent of bridge: port giving best path to Root

A few extra interesting things

- Example things you can configure
  - Bridge priority
  - Link cost
  - Max-age
- Why this protocol is unstable if bridges cannot keep up with wire speed on receive

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.

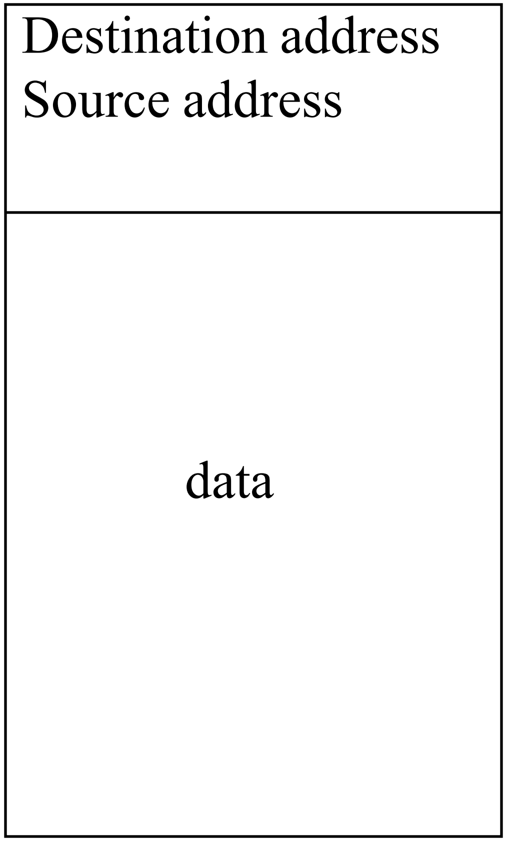
Note some things about bridges

- Certainly don’t get optimal source/destination paths
- Temporary loops are a disaster
  - No hop count
  - Exponential proliferation
- Inherently unstable
- 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!

Review



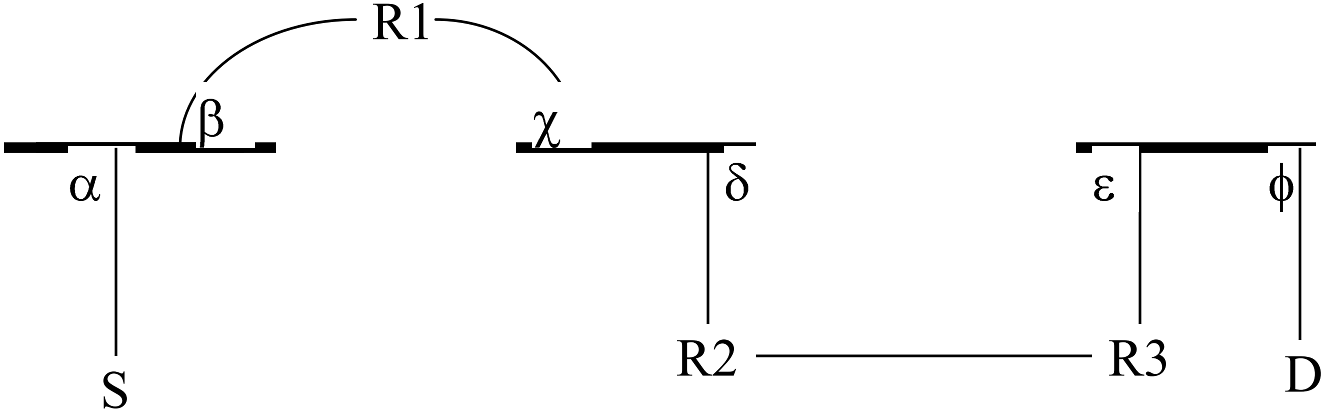
Destination address
Source address

data

When I started

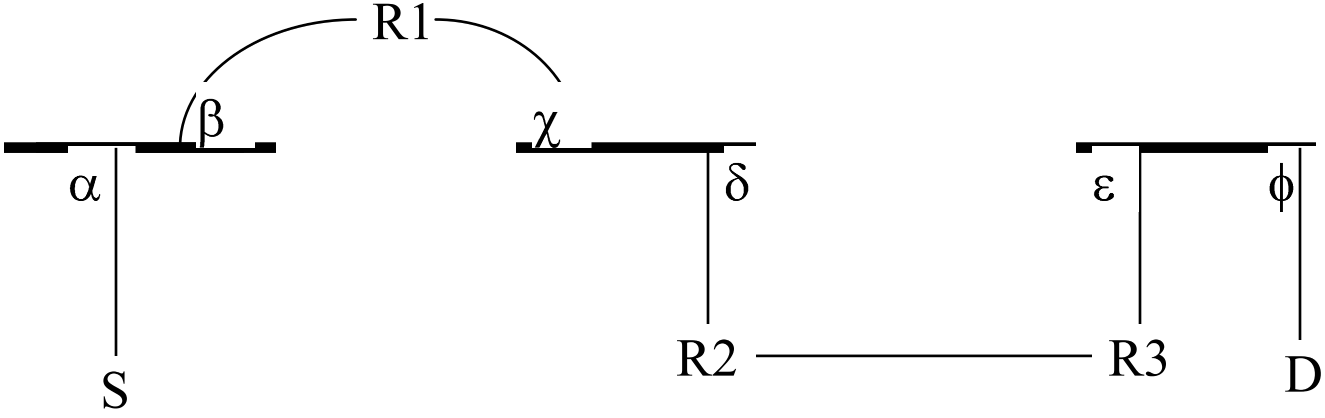
- Layer 3 had source, destination addresses
- Layer 2 was just point-to-point links (mostly)
- If layer 2 is multiaccess, then need two headers:
  - Layer 3 has ultimate source, destination
  - Layer 2 has next hop source, destination

Hdrs inside hdrs



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

Hdrs inside hdrs

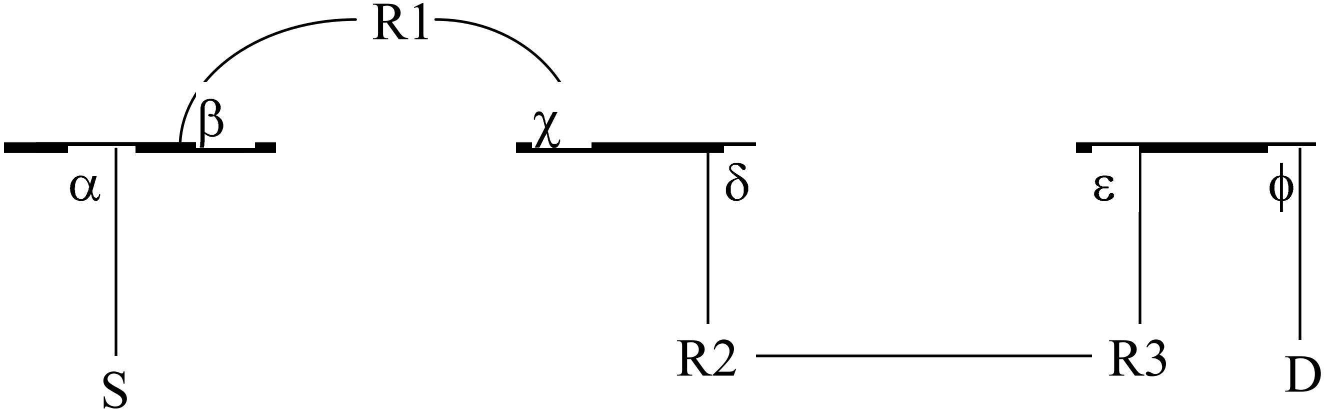


S:

Layer 2 hdr	Layer 3 hdr	

Dest=β
Source=α
	Dest=D
	Source=S	

Hdrs inside hdrs

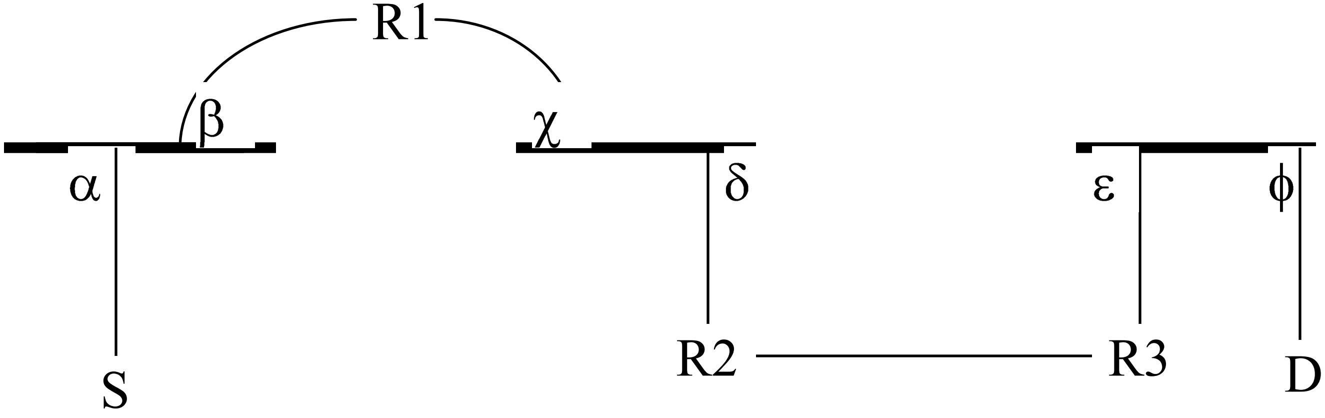


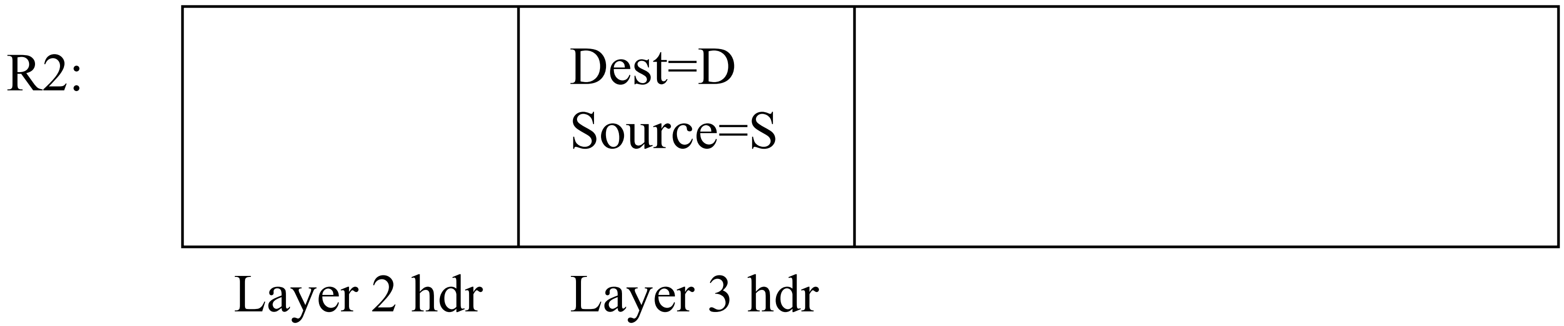
R1:

Layer 2 hdr	Layer 3 hdr	

Dest=δ
Source=χ
	Dest=D
	Source=S	

Hdrs inside hdrs



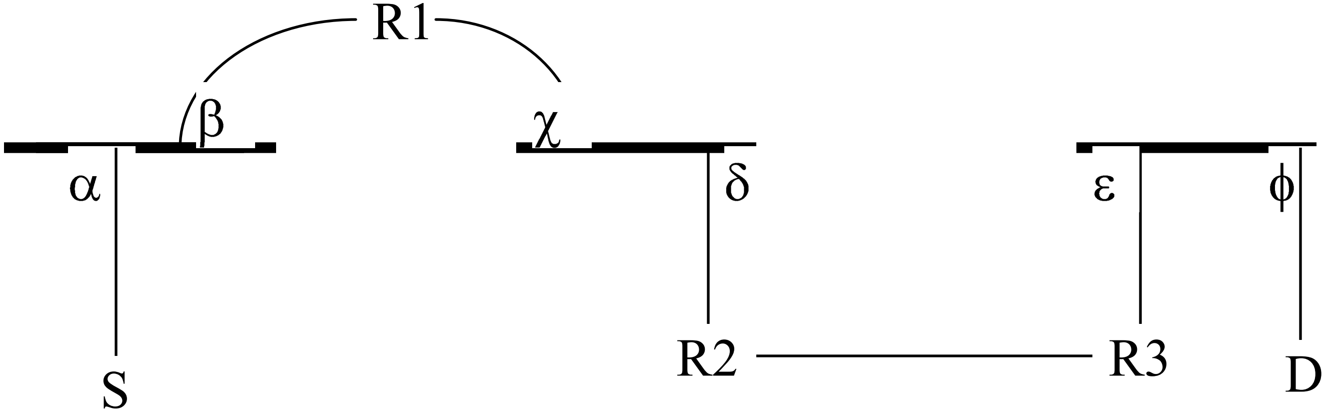

R2:

Dest=D
Source=S

Layer 2 hdr

Layer 3 hdr

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

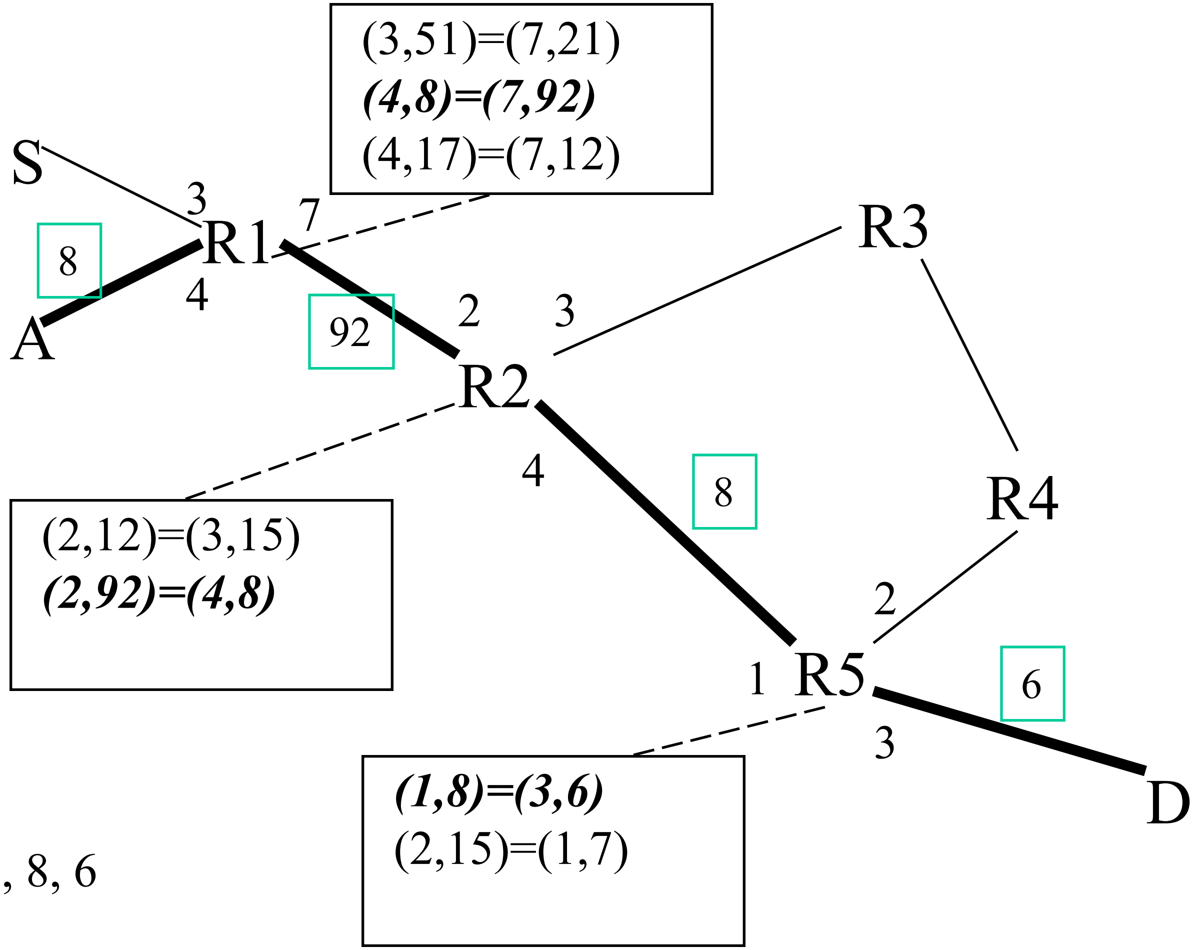
Network Layer

- 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



(3,51)=(7,21)
(4,8)=(7,92)
(4,17)=(7,12)

(2,12)=(3,15)
(2,92)=(4,8)

(1,8)=(3,6)
(2,15)=(1,7)

label=8, 92, 8, 6

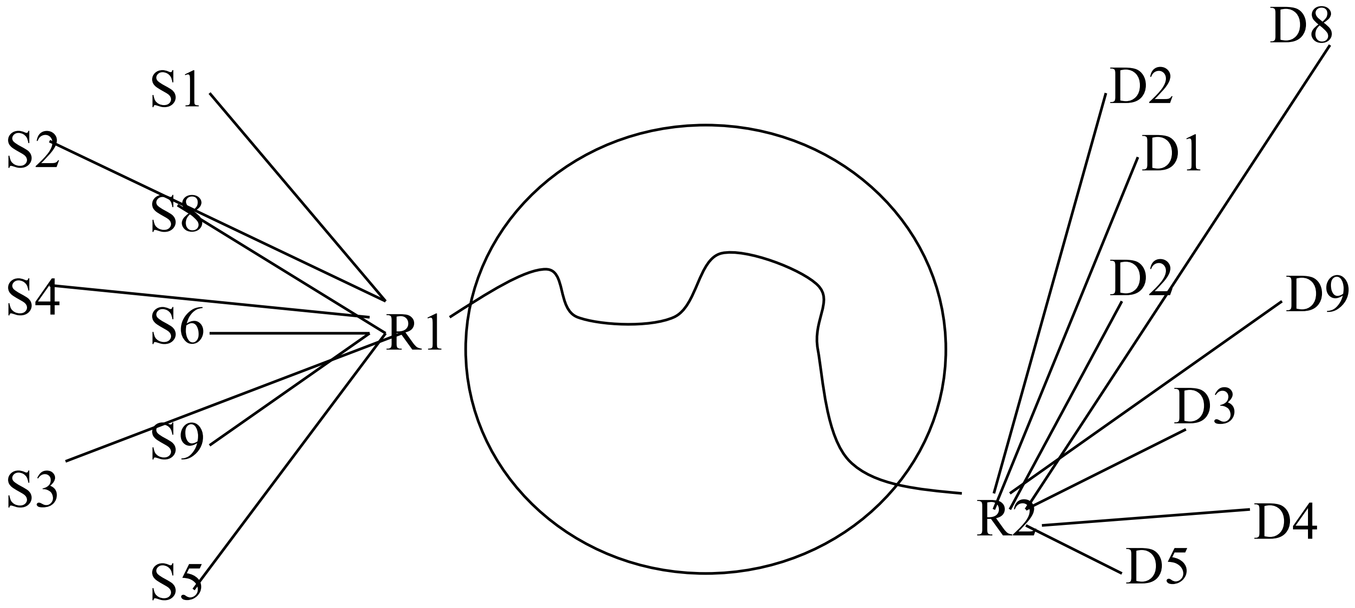
Lots of connection-oriented networks

- 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

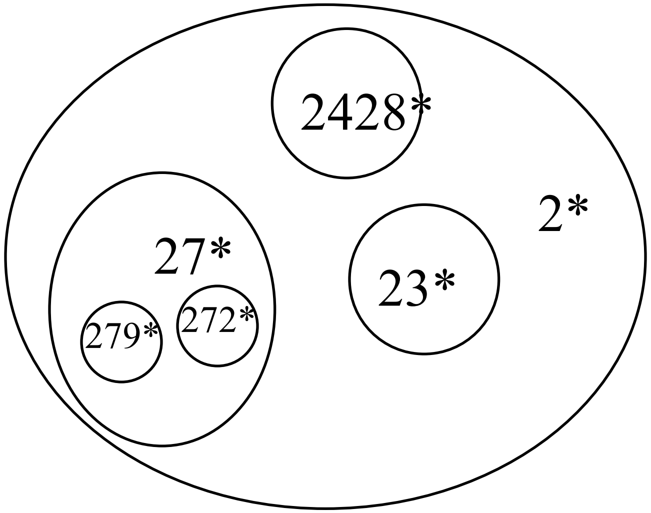
- Destination, source, hop count
- Maybe other stuff
  - fragmentation
  - options (e.g., source routing)
  - error reports
  - special service requests (priority, custom routes)
  - congestion indication
- Real diff: size of addresses

Addresses

- 802 address “flat”, though assigned with OUI/rest. No topological significance
- layer 3 addresses: locator/node : topologically hierarchical address
  - IPv4, IPv6, IPX, AppleTalk: unique “locator” for each 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



Hierarchy

- Makes network much more scalable
- Allows forwarding tables to be much smaller
- But paths are no longer optimal
    - Enter circle as soon as possible
    - Not necessarily best place for the specific destination inside the circle

Fixed hierarchy vs longest prefix match

- Fixed: If top portion = yours, route based on rest
- Longest prefix match: flexible boundaries
- Comparison
  - Lookup easier with fixed boundaries
  - Longest prefix match allows regions of different sizes without wasting bits of address by allocating maximum # of bits at each level

Address Prefix Routing

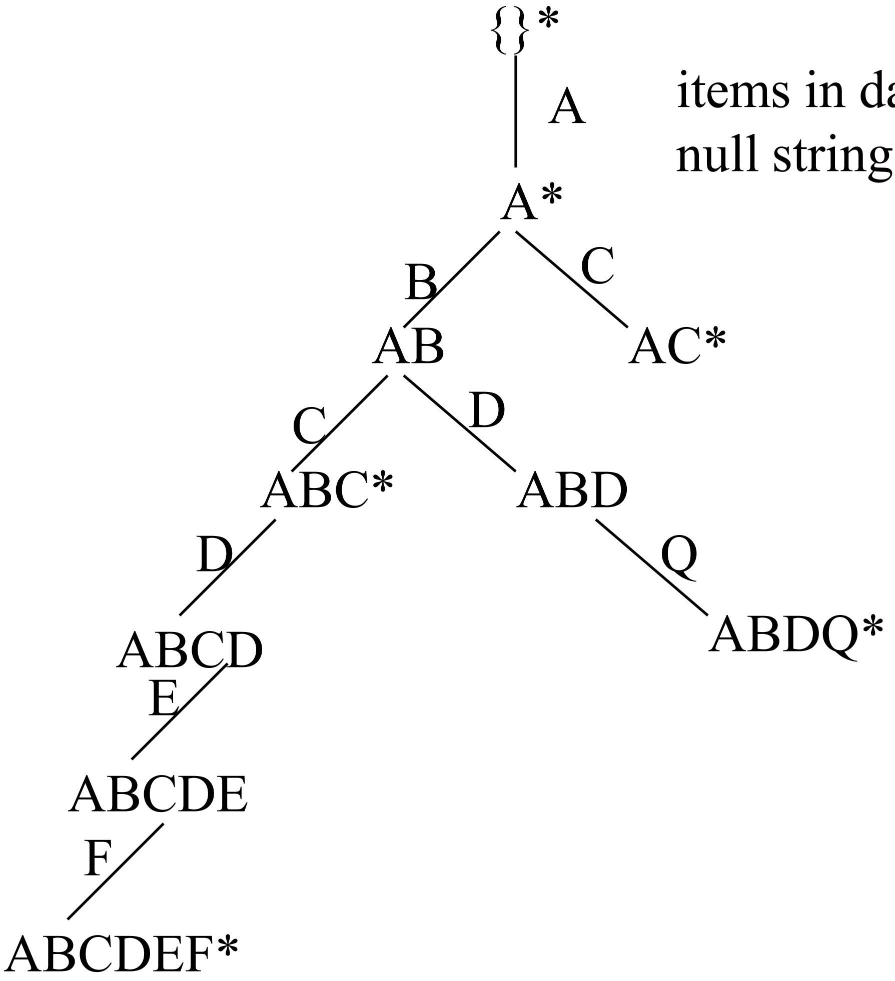
- Given destination address, want to find longest prefix match in forwarding table
- Two basic algorithms
  - TRIE
  - modified binary search

How to do Longest Prefix search

TRIE

- Character-by-character search
- “Character” might be single bit
- “*” means match
- remember last time “*” seen
- once nowhere to go, last “*” is longest prefix match

TRIE



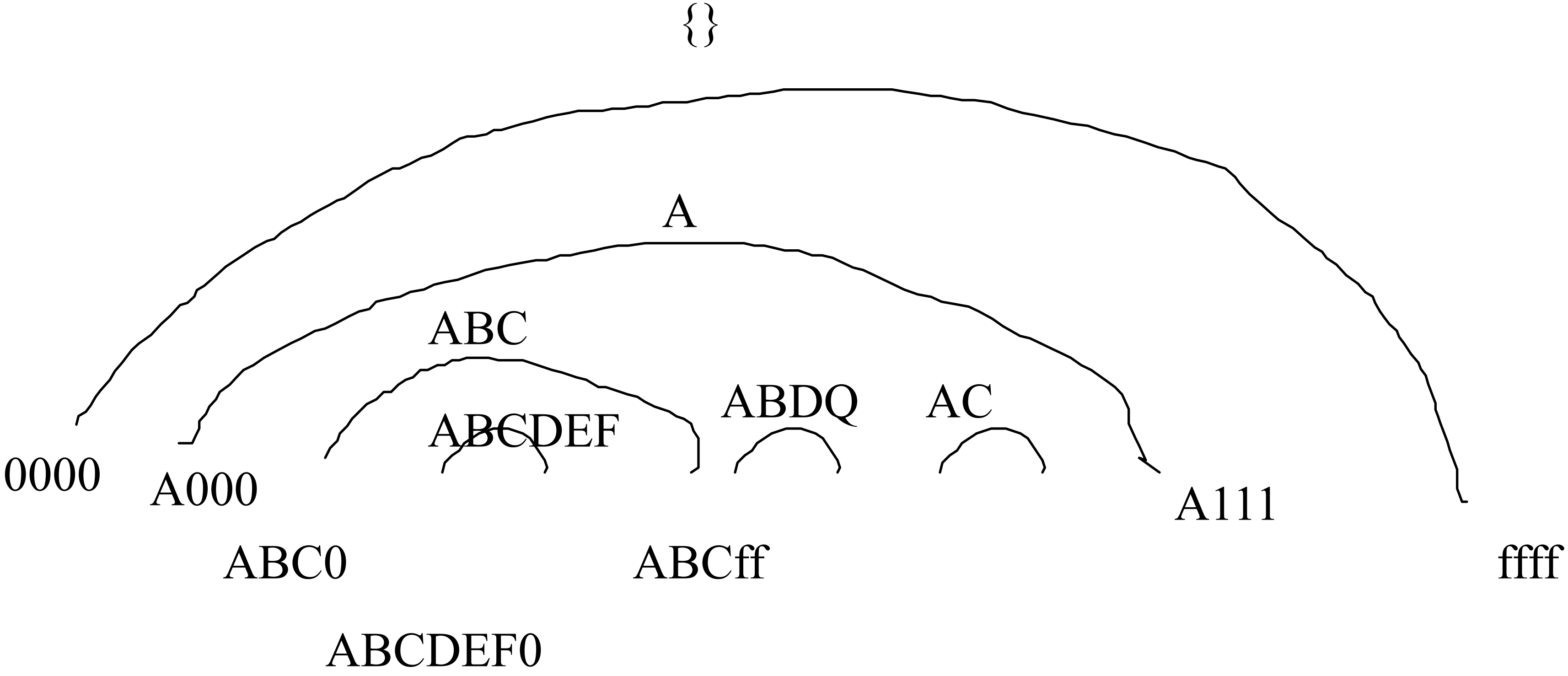
items in database:
null string, A, ABC, ABCDEF, ABDQ, AC

Binary search

- Create ranges
- Take each prefix
  - pad with 0’s for low order of range
  - pad with 1’s for hi order of range
- Sort them
- Find where destination address fits

Binary Search

items: {}, A, ABC, ABCDEF, ABDQ, AC



Forwarding Decision

- Switch makes decision on how to forward based on:
  - Information in packet
  - Forwarding table

Next topics to discuss

- What is in a forwarding table
- How to create forwarding tables
- How to do address lookup

What’s in a forwarding table

- Flat destination, small (like TRILL)
  - Direct lookup → {output ports}
- Flat destination, large (like Ethernet)
  - Hash → {output ports}
- Prefix (like IP)
  - “longest matching prefix” → {output ports}
- Path (like MPLS)
- ((input port, label) → (output port, new label))

Size of forwarding table

- Destination
  - O(# of destinations)
- Path-based
  - O(# of communicating pairs)
  - So..if you want n^2 communicating parties, forwarding table would be the square!
  - And if you want path diversity…exponential!

Why did ATM use path-based?

- Assumptions
  - # of actively communicating pairs much smaller than total number of destinations
  - OK to have latency to set up path when A first decides to talk to B
  - OK to give “fast busy signal” if some switch doesn’t have resources for a new connection

Why did MPLS use path-based?

- Longest prefix match hard
- So, give neighbor a shorthand
  - In the future, when you forward that kind of packet to me, use this label

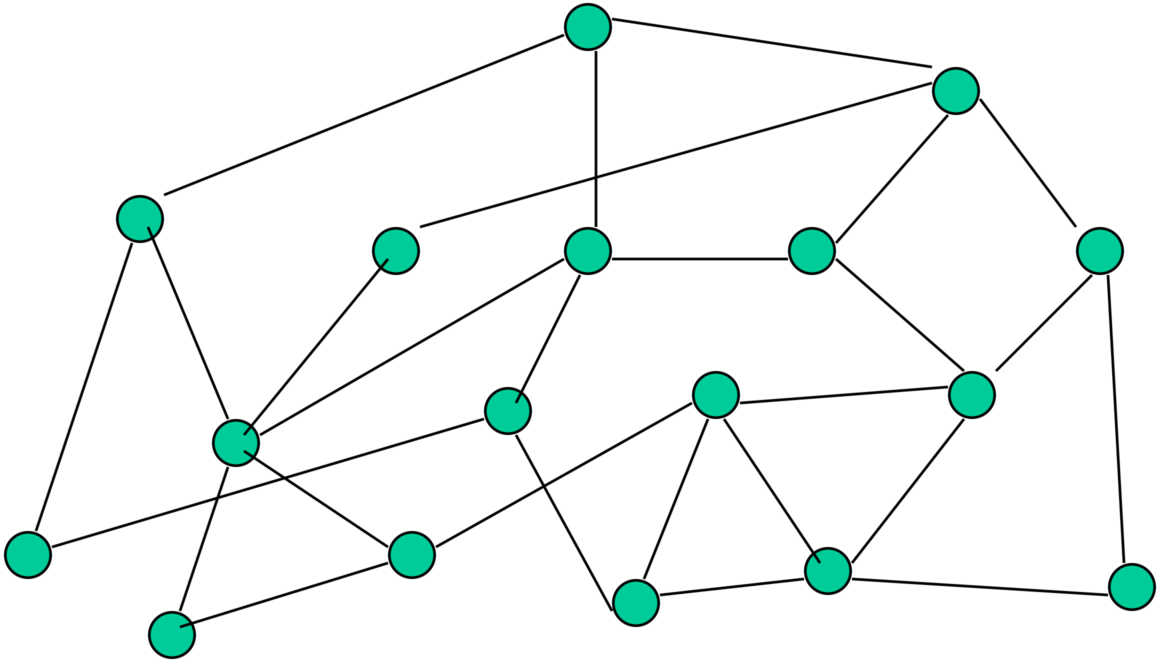
Why did MPLS use path-based?

- Longest prefix match hard
- So, give neighbor a shorthand
  - In the future, when you forward that kind of packet to me, use this label
- Would have been better to be dest-based
- But what about traffic engineering?
  - Dest-based can still lock down some paths: have a few special “destinations” for fixed path

Where does forwarding table come from?

- Distributed algorithm
- Configured
- Central fabric manager

New topic: Routing Algorithms

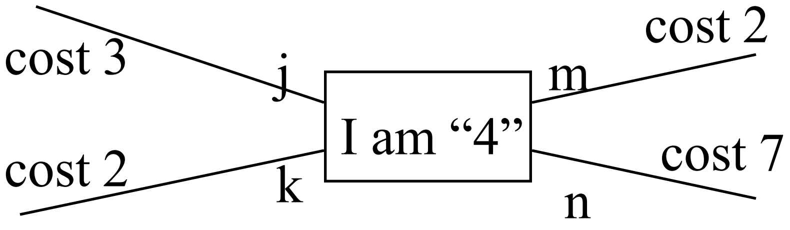


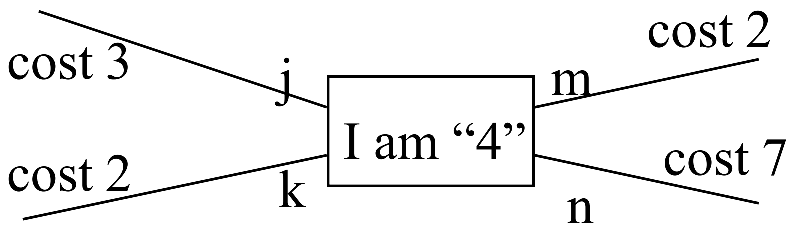
Distributed Routing Protocols

- 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

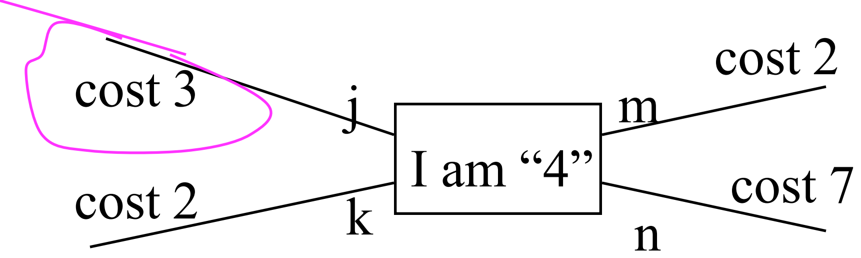
6	2	0	7	8	5	8	12	11	3	2

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

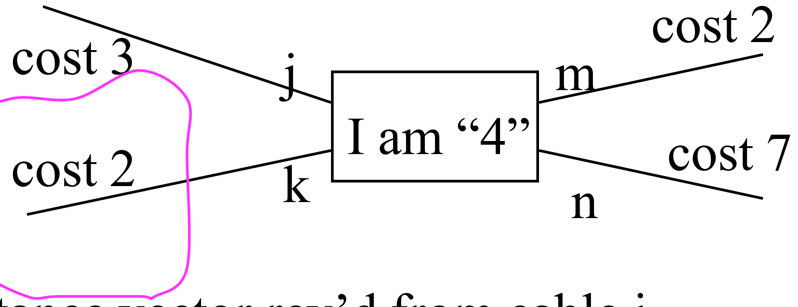
cost 7	6	2	0	7	8	5	8	12	11	3	2

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

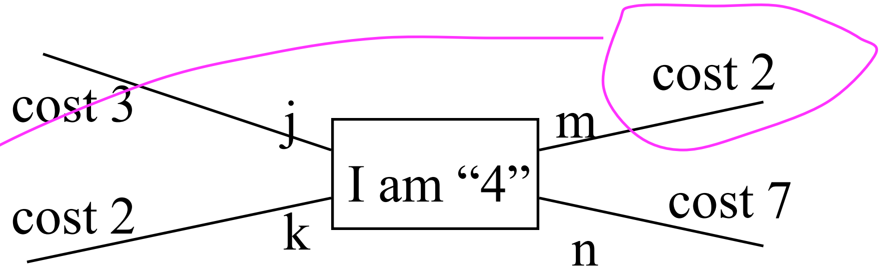
cost 7	6	2	0	7	8	5	8	12	11	3	2

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

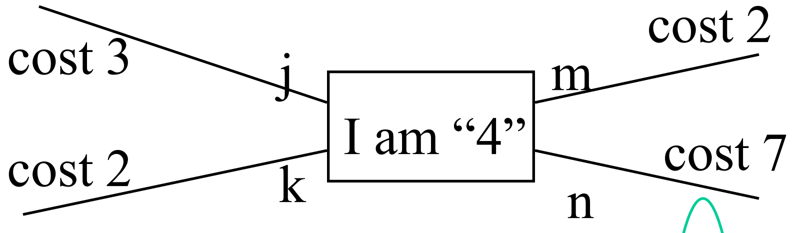
cost 7	6	2	0	7	8	5	8	12	11	3	2

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

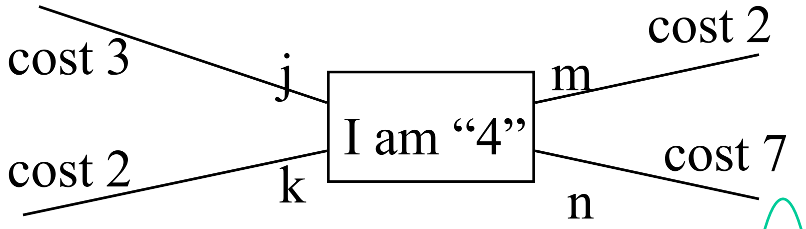
cost 7	6	2	0	7	8	5	8	12	11	3	2

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

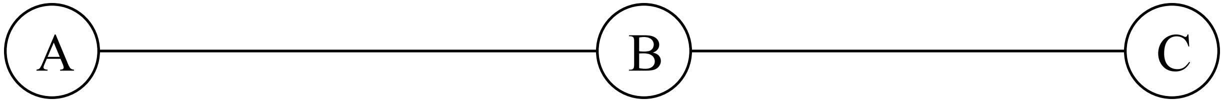
your own calculated distance vector

2	6	5	0	12	8	6	19	3	?	?

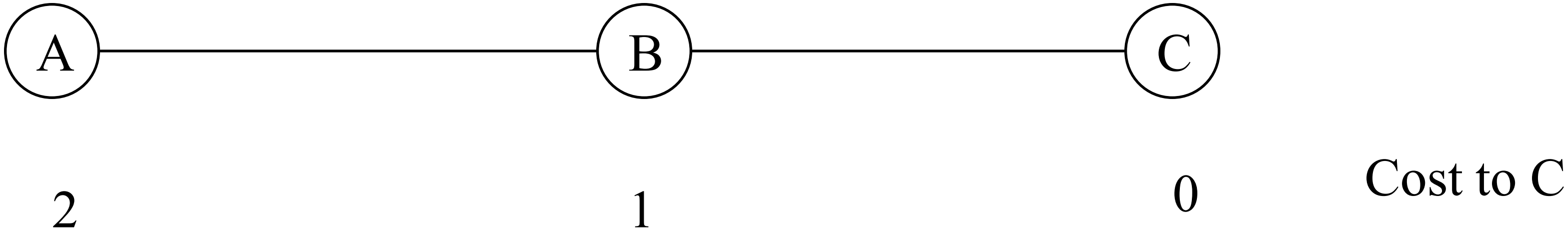
your own calculated forwarding table

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

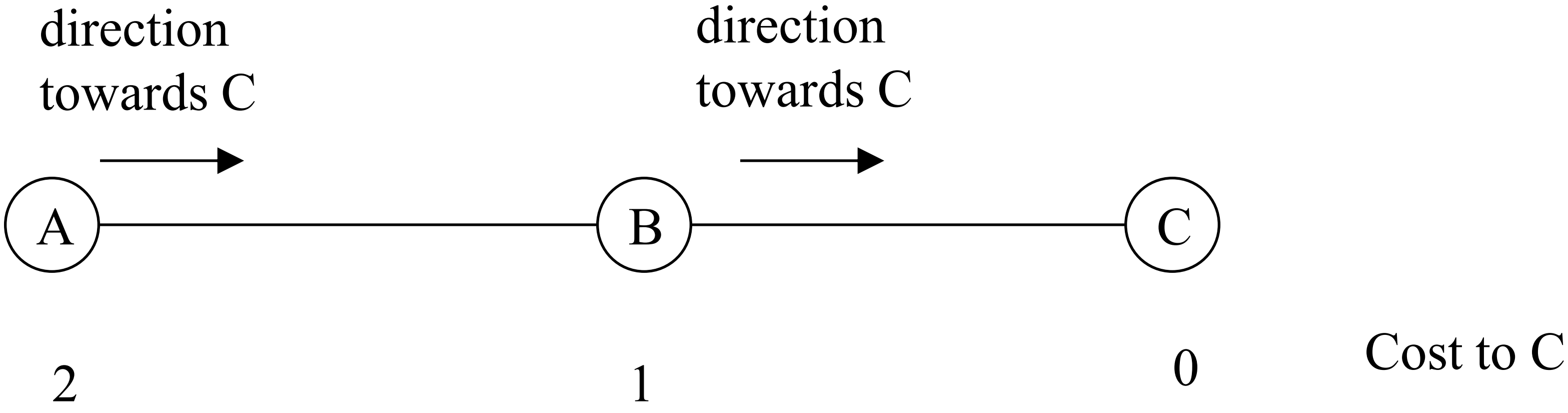
Looping Problem



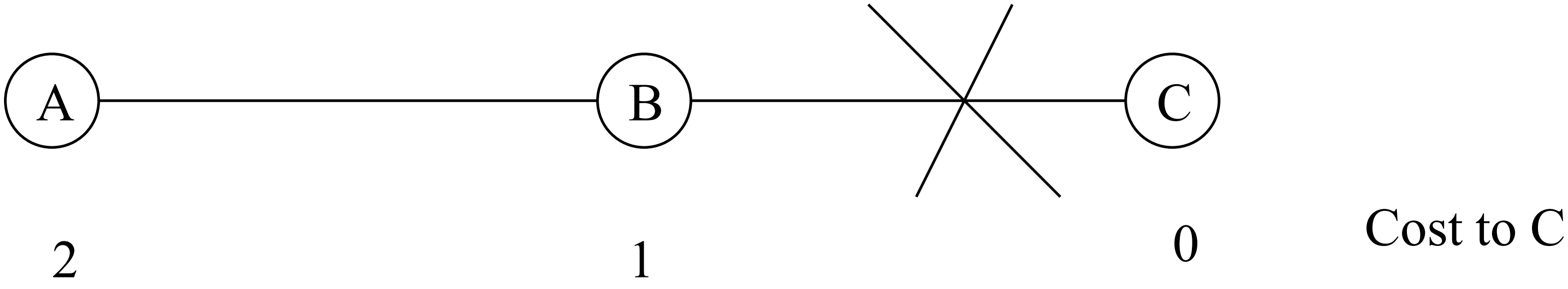
Looping Problem



Looping Problem

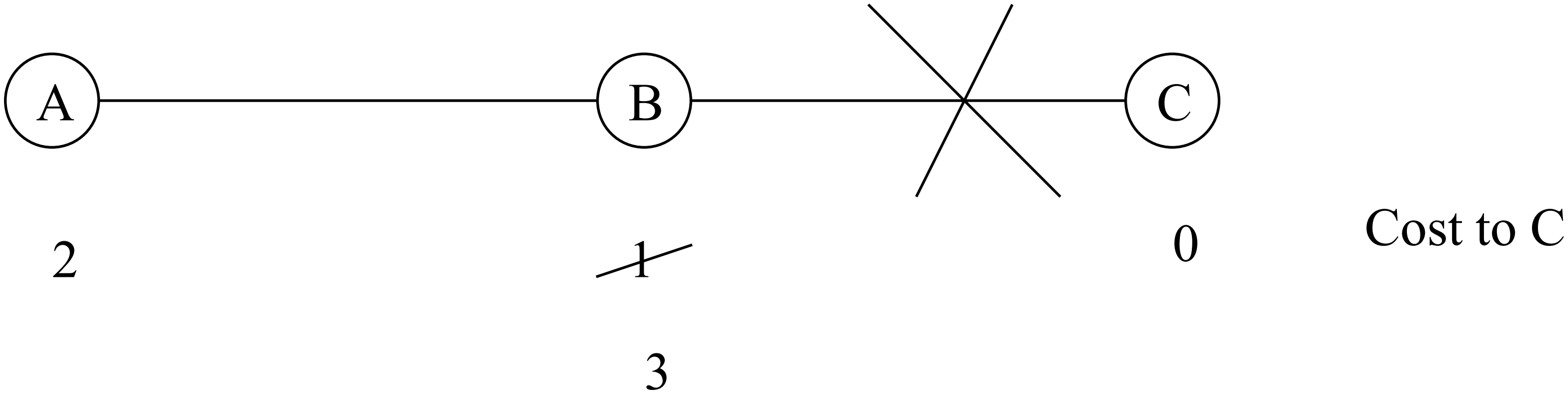


Looping Problem

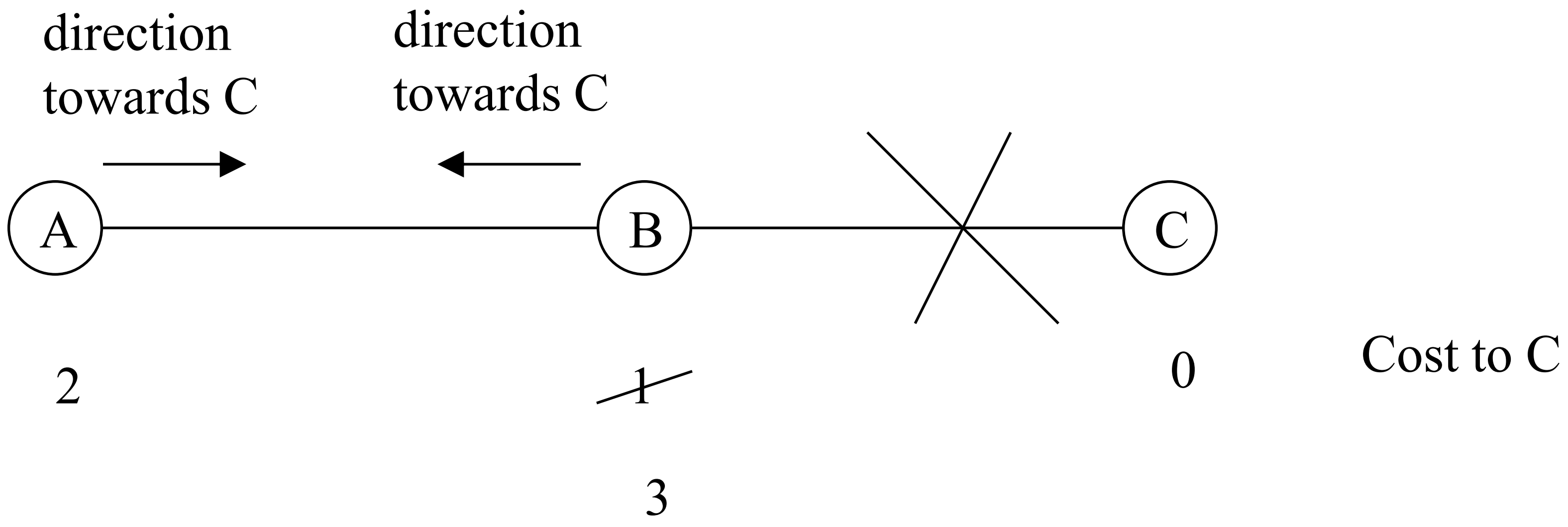


What is B’s cost to C now?

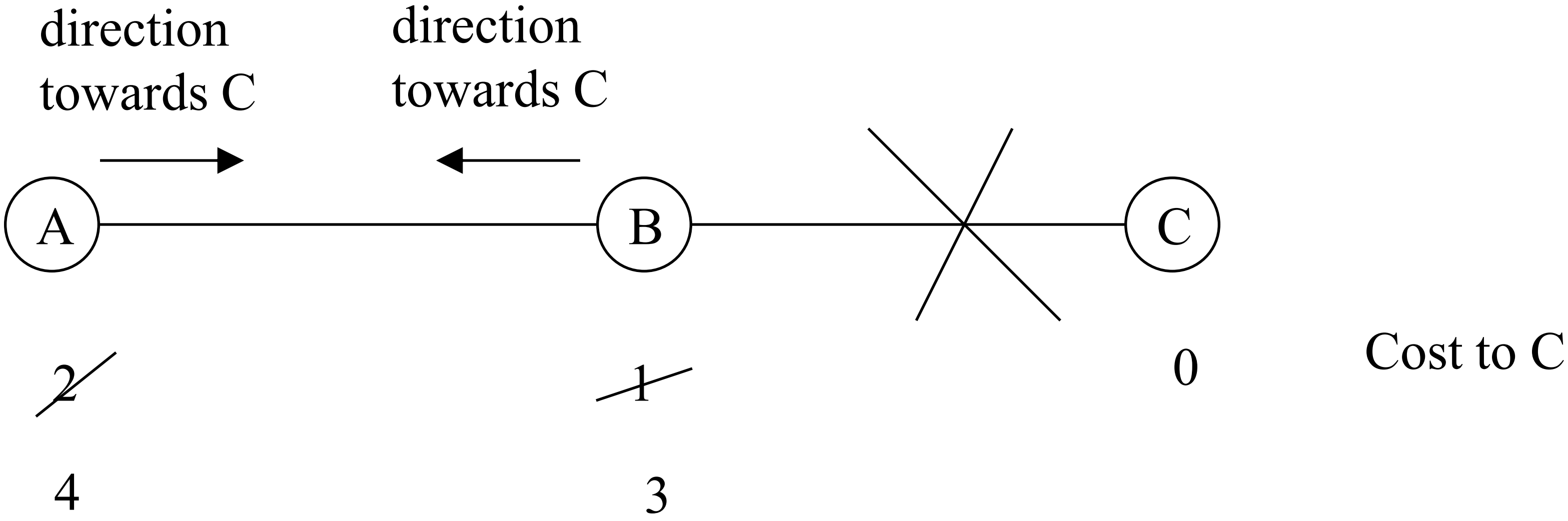
Looping Problem



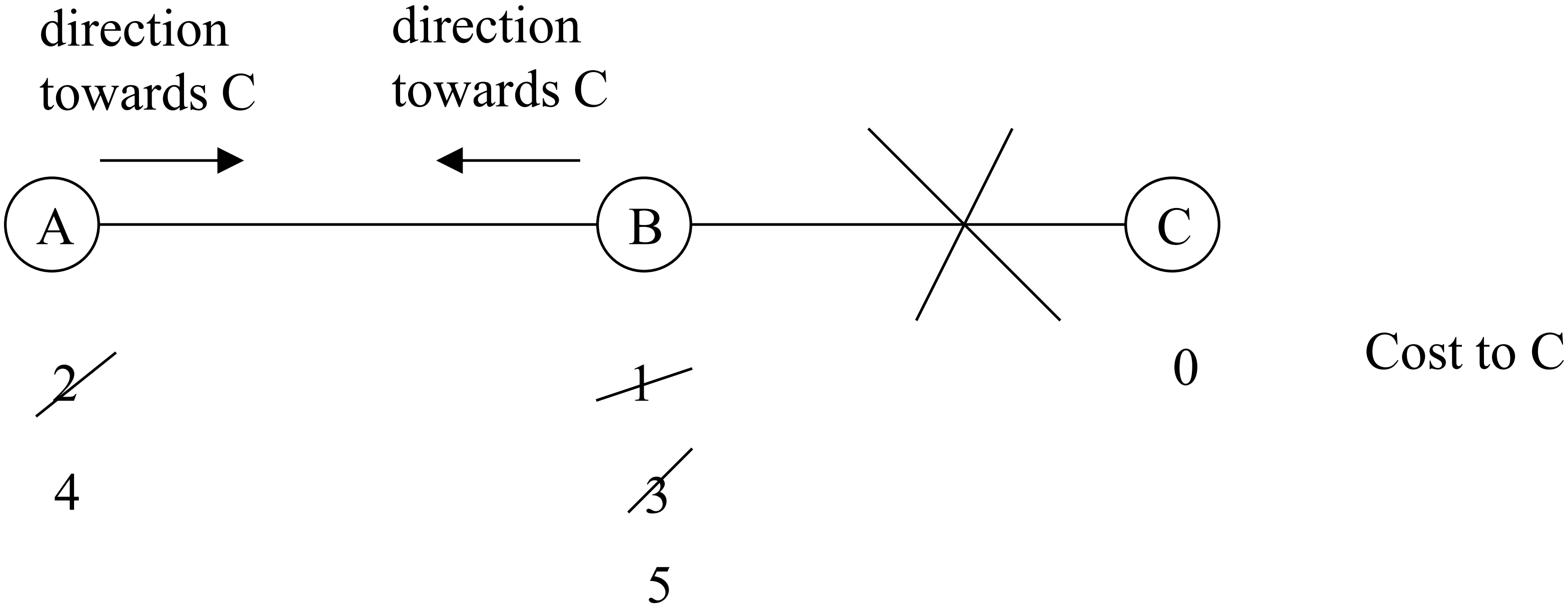
Looping Problem



Looping Problem



Looping Problem



direction towards C

direction towards C

~~2~~

4

~~1~~

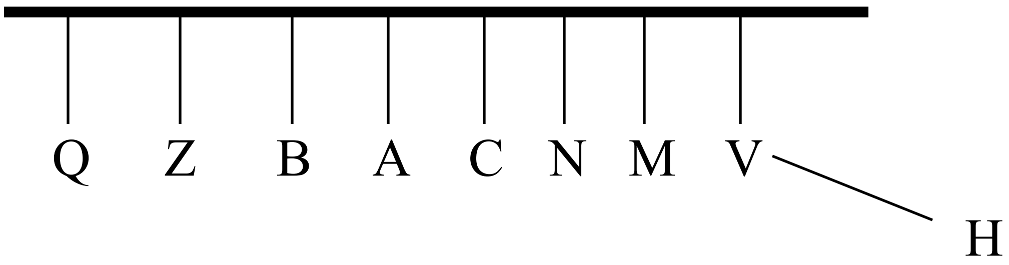
~~3~~

5

0

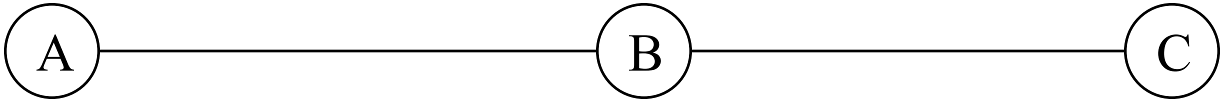
Cost to C

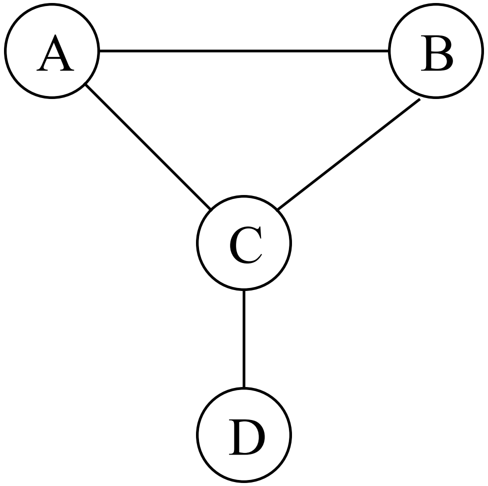
Looping Problem worse with high connectivity



Split Horizon: one of several optimizations

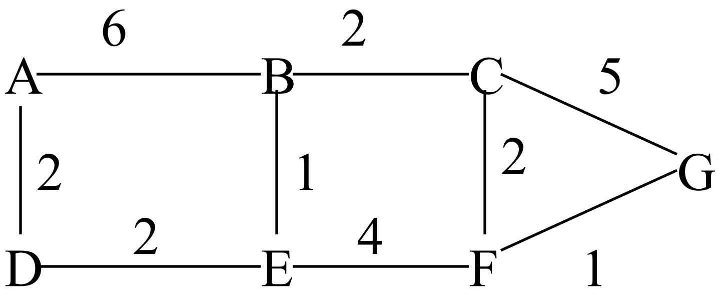
Don’t tell neighbor N you can reach D if you’d forward to D through N





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)



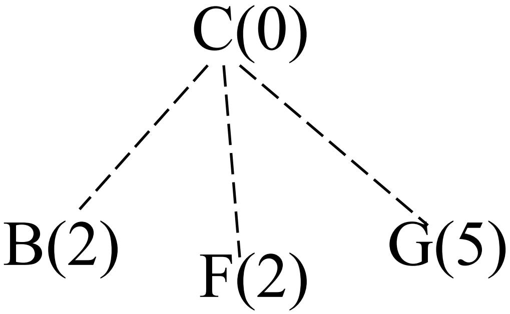
A	B	C	D	E	F	G
B/6	A/6	B/2	A/2	B/1	C/2	C/5
D/2	C/2	F/2	E/2	D/2	E/4	F/1
	E/1	G/5		F/4	G/1	

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

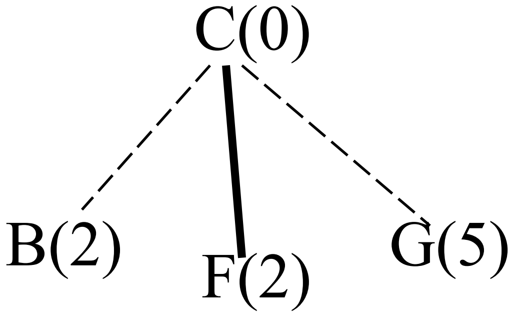
Look at LSP of new tree node

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	



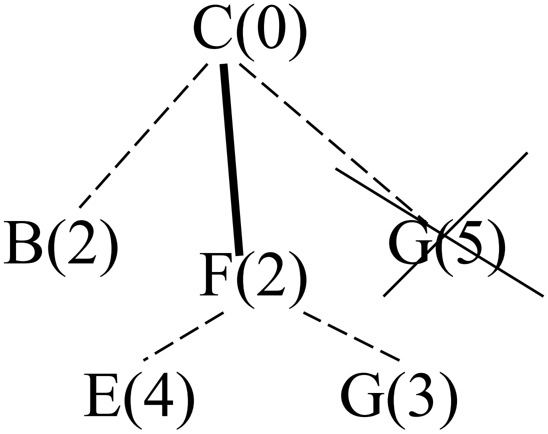
Make shortest TENT solid

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	



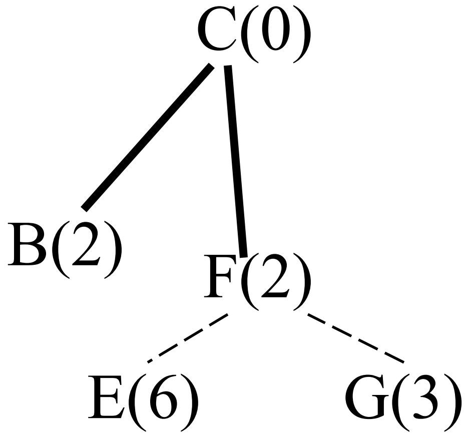
Look at LSP of newest tree node

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	



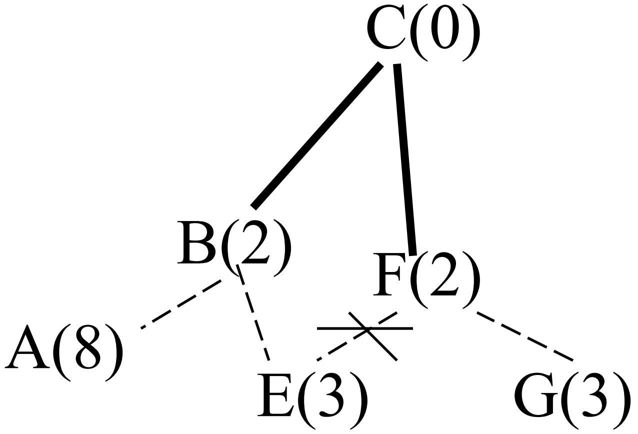
Make shortest TENT solid

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	



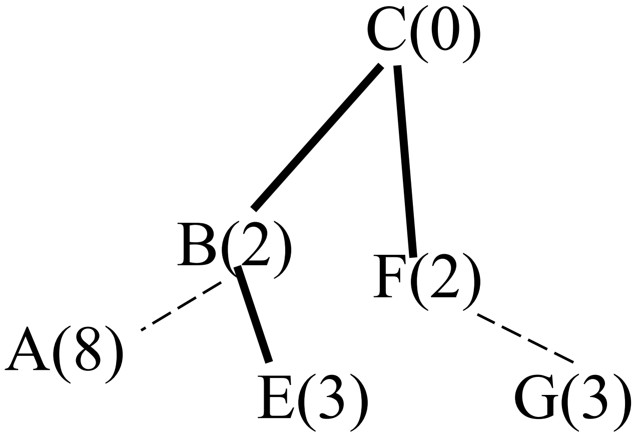
Look at LSP of newest tree node

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	



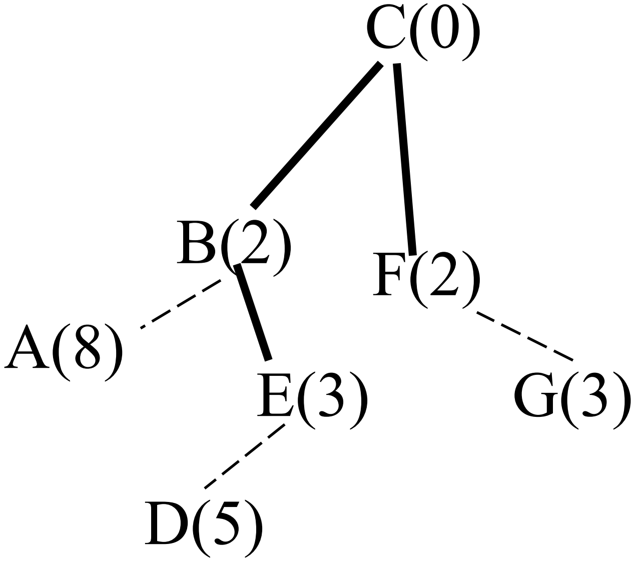
Make shortest TENT solid

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	



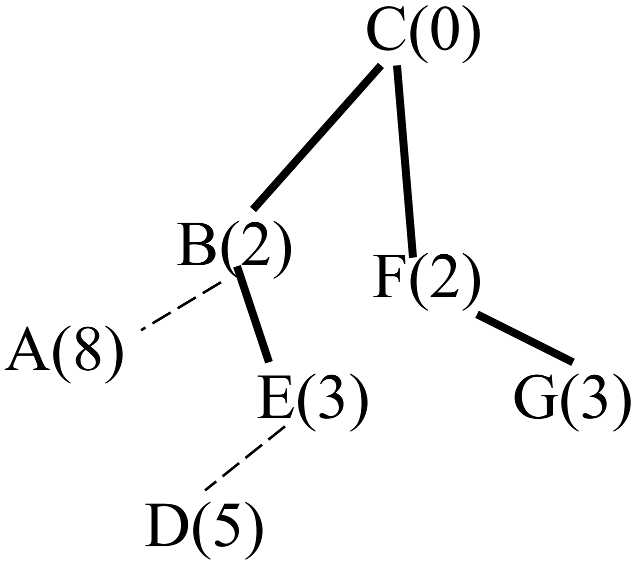
Look at LSP of newest tree node

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	



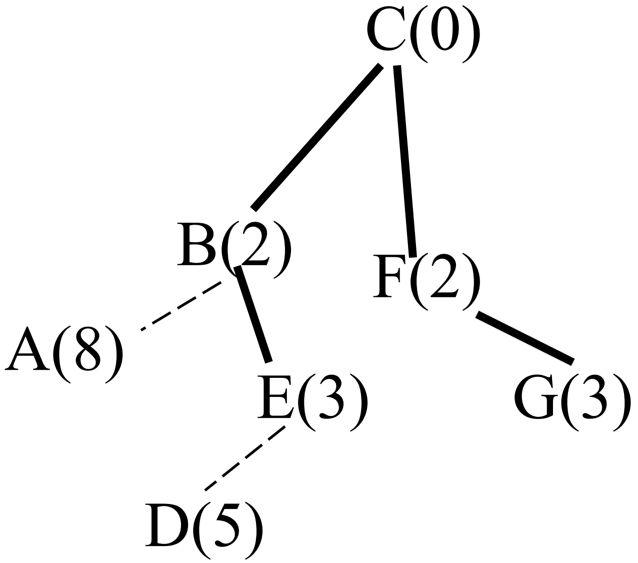
Make shortest TENT solid

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	



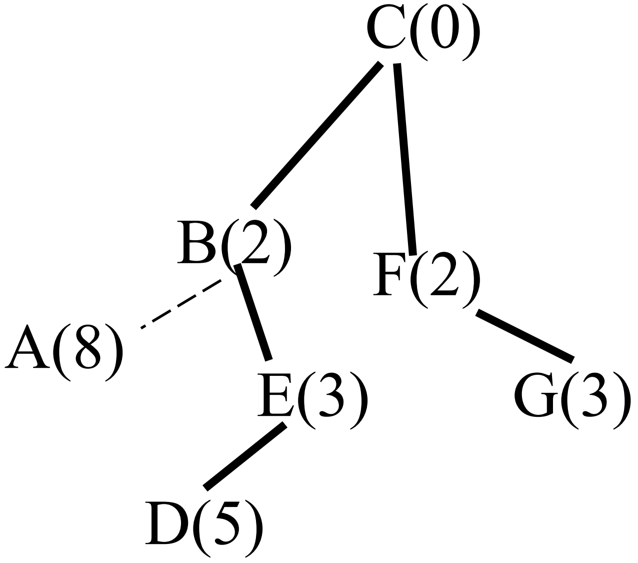
Look at newest tree node's LSP

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	



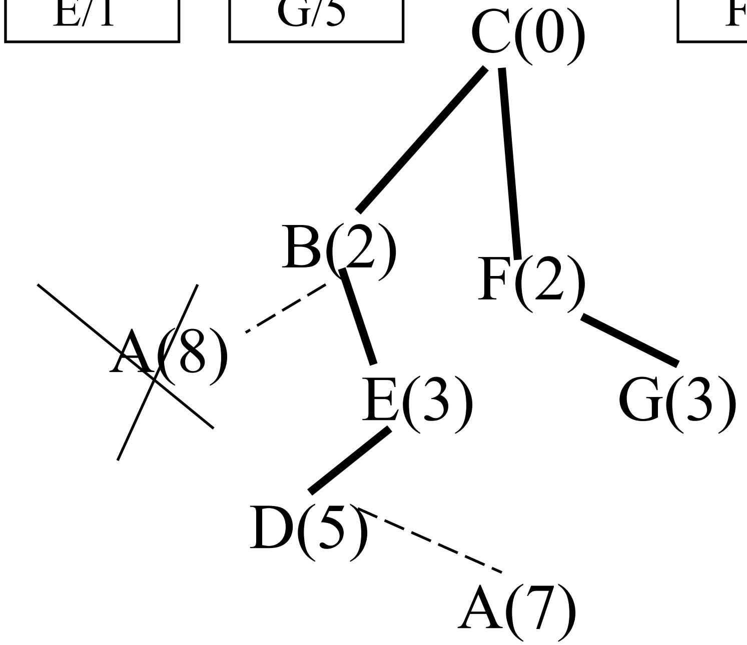
Make shortest TENT solid

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	



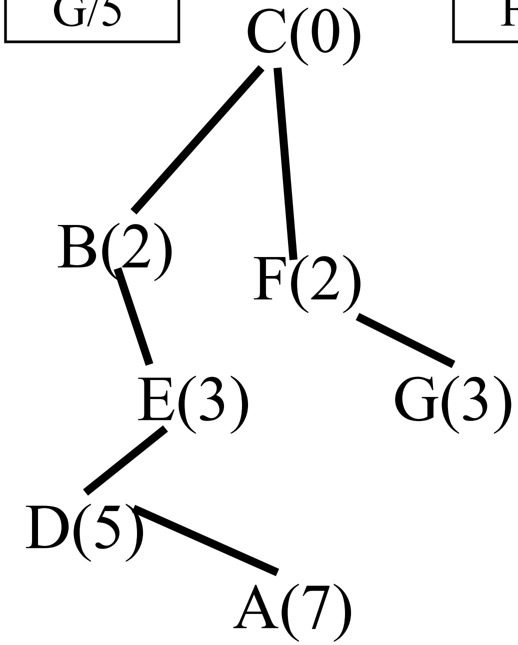
Look at newest node’s LSP

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	



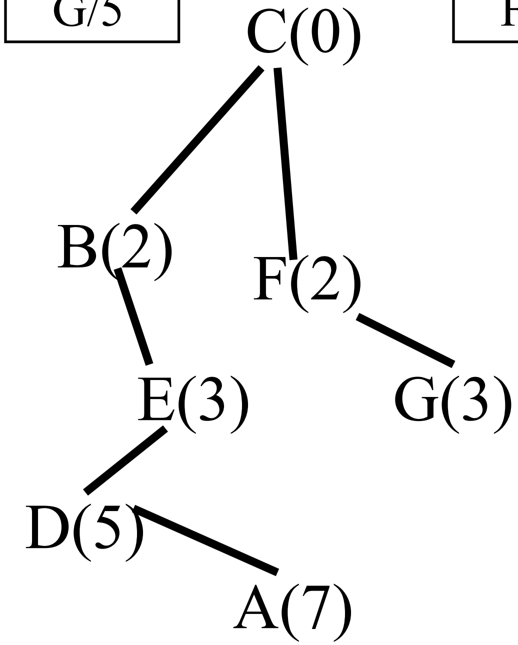
Make shortest TENT solid

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!

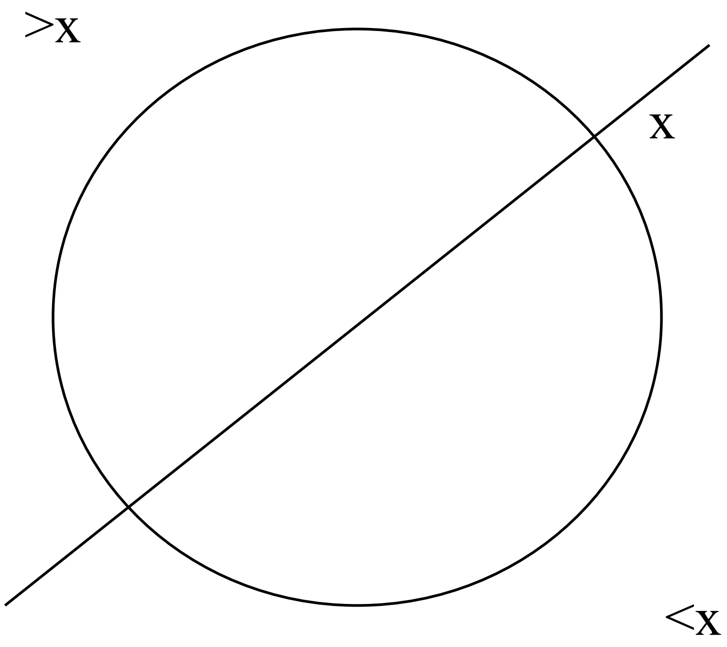
A	B	C	D	E	F	G
B/6	A/6	B/2	A/2	B/1	C/2	C/5
D/2	C/2	F/2	E/2	D/2	E/4	F/1
	E/1	G/5		F/4	G/1	



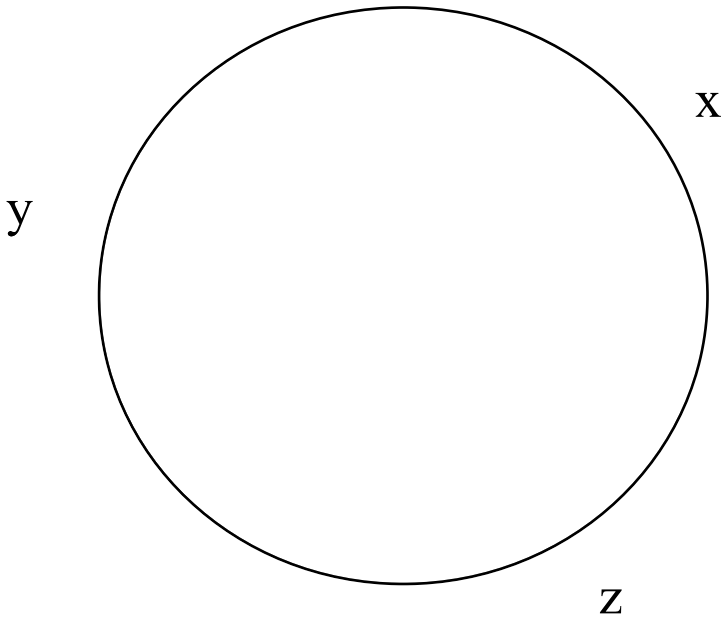
ARPANET: first link state protocol: unstable!

- Their algorithm for flooding link state packets was unstable
- Sounds simple:
  - LSP has sequence number
  - R2 rcvs LSP from source S, seq # x
  - R2 has LSP from S with seq # y
  - If x > y, overwrite and flood, else discard

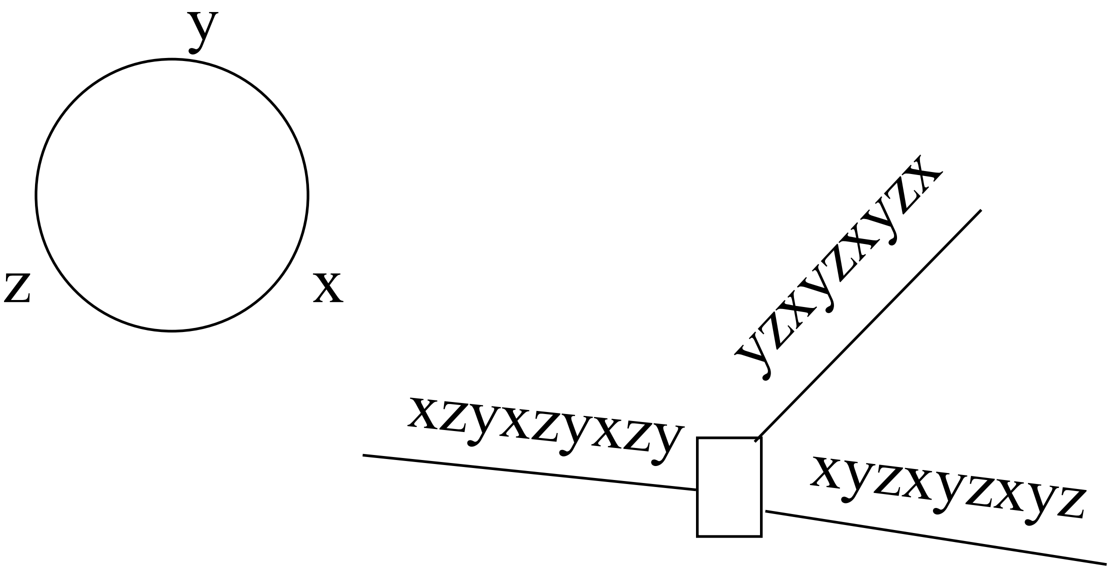
Arithmetic in circular space



$x < y < z$



ARPANET disaster



Diagnosing and Fixing the Net

- Luck!

Diagnosing and Fixing the Net

- Luck!
- Network “didn’t work”: managed from one place

Diagnosing and Fixing the Net

- Luck!
- Network “didn’t work”: managed from one place
- Tried rebooting their router…didn’t help

Diagnosing and Fixing the Net

- Luck!
- Network “didn’t work”: managed from one place
- Tried rebooting their router…didn’t help
- Did core dump…queue filled with LSPs from “Fred”, with sequence #s x, y, z

Diagnosing and Fixing the Net

- Luck!
- Network “didn’t work”: managed from one place
- Tried rebooting their router…didn’t help
- Did core dump…queue filled with LSPs from “Fred”, with sequence #s x, y, z
- How to fix?

Routing Robustness

- “self-stabilizing” link state protocol…but only after sick/evil node gone
- My thesis: robust even if some of the routers currently attached are evil. More than securing the routing *protocol*: it deals with packet delivery

Other interesting IS-IS details

- Finding neighbors on a shared medium
  - Multicast “Hello”, listing who you’ve heard Hellos from
  - X lists Y as neighbor in X’s LSP iff X hears Y’s Hello, and Y lists X as neighbor in Hello
- How to send LSPs reliably over shared link
  - Individual acks could be problematic
  - So IS-IS has elected master transmit CSNP periodically (“complete sequence numbers packet”), which summarizes LSP database

Other interesting IS-IS details

- Some packets can be too large to fit into a single Ethernet frame
- Typical IP-style fragmentation requires
  - Reassembling before processing
  - Retransmitting entire packet if one fragment gets lost
- IS-IS avoids this – carefully encode so each “fragment” can be processed!
  - Hello neighbor list: sort neighbors, “this fragment contains nbrs 493 through 875”
  - CSNP: “this fragment covers LSPs with ID x through y”

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)

How to create forwarding table

- Configuration, fixed
  - Certainly least overhead, if topology isn’t dynamic
- Distributed vs centralized
  - Distributed will react to changes more quickly

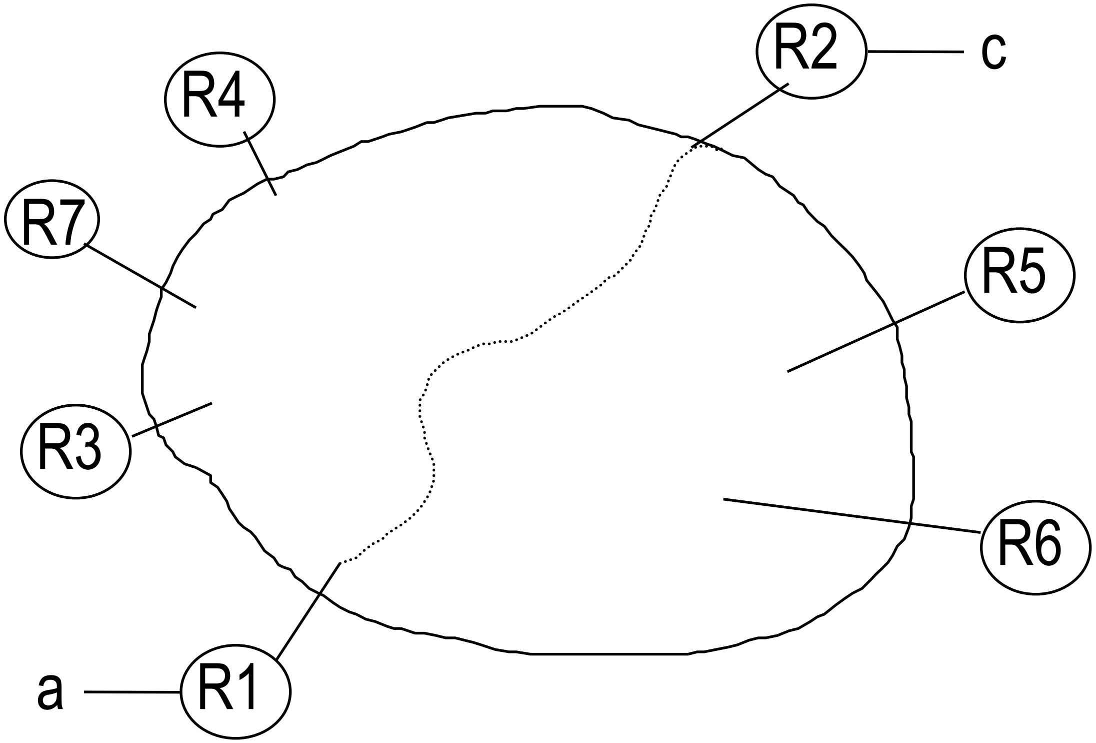
TRILL working group in IETF

- 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

Basic TRILL concept



Basic TRILL concept

- TRILL switches find each other (perhaps with bridges in between)
- Calculate paths to other TRILL switches
- First TRILL switch tunnels to last TRILL switch
- Reason for extra header:
  - Forwarding table in TRILL switches just size of # of TRILL switches
  - Layer 3-like header (hop count)
  - Small, easy to look up, addresses

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)

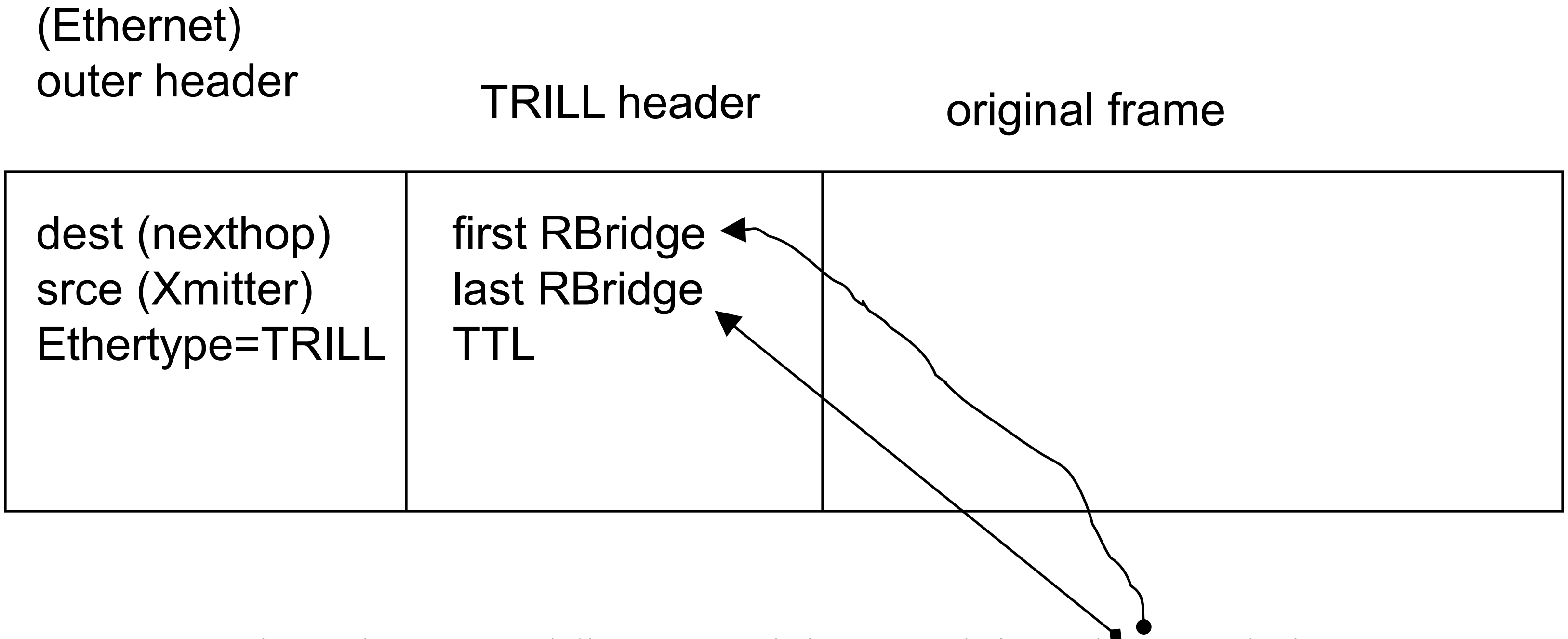
Routing inside campus

- 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

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

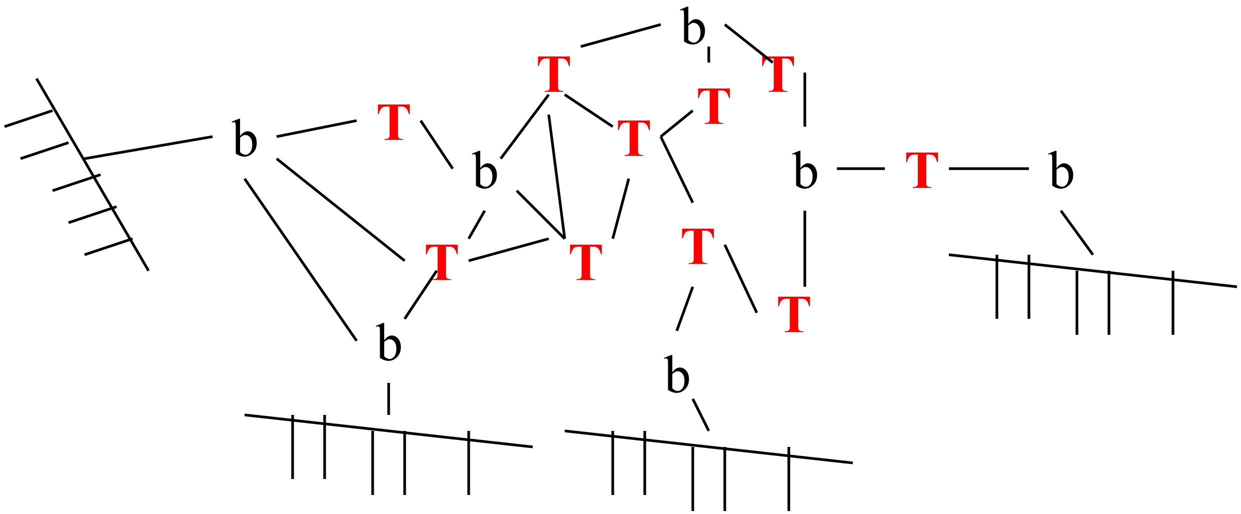
TRILL header specifies RBridges with 2-byte nicknames

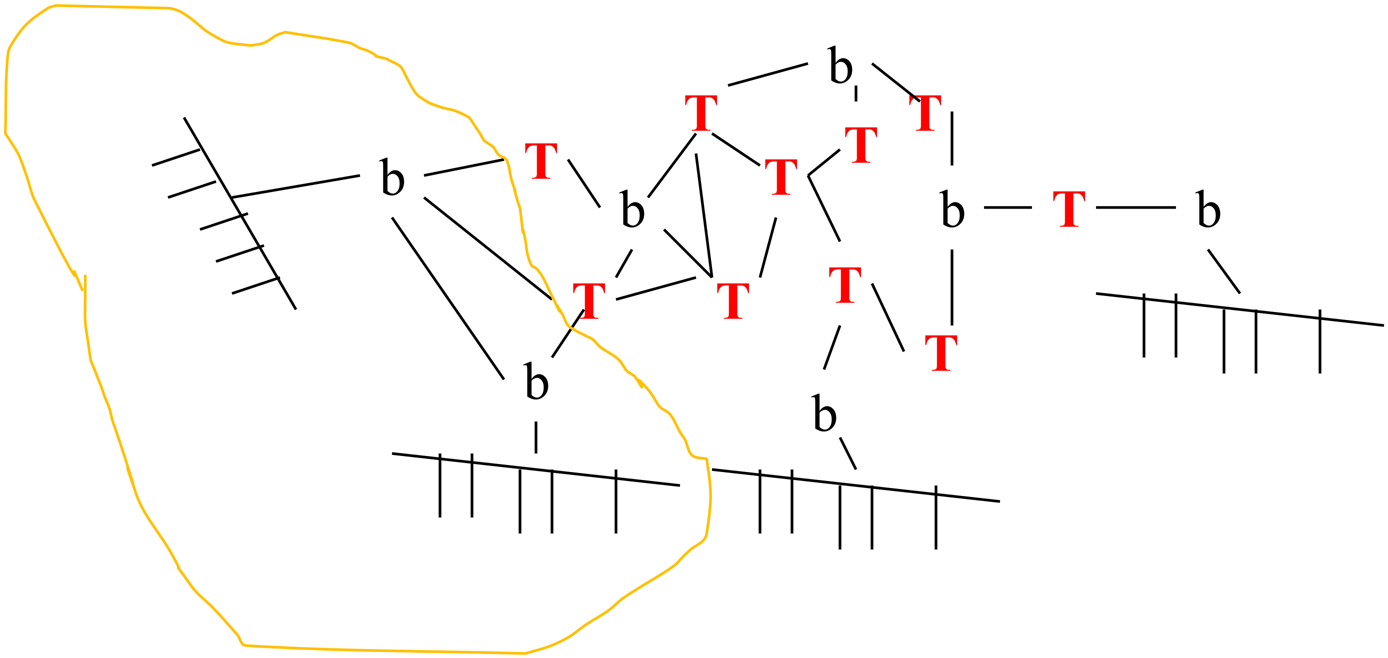
2-byte Nicknames

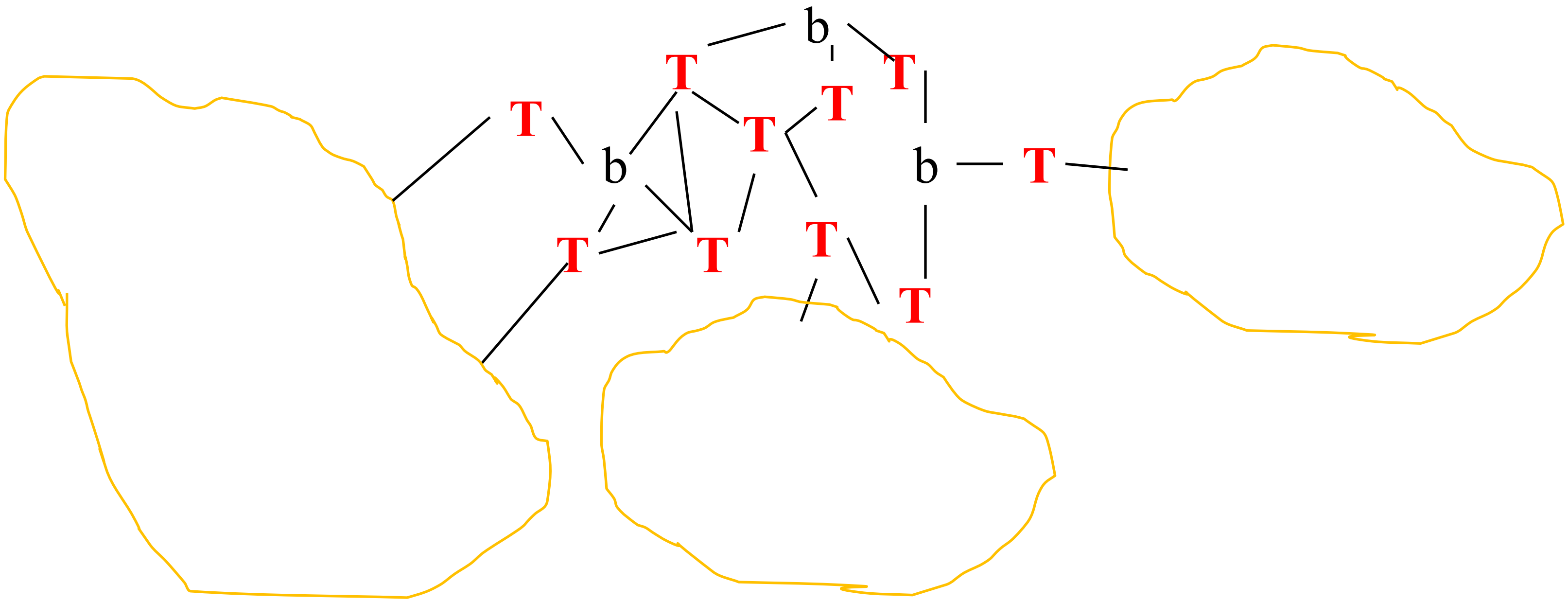
- Saves hdr room, faster fwd’ing
- Dynamically acquired
- Choose unused #, announce in LSP
- If collision, IDs and priorities break tie
- Loser chooses another nickname
- Configured nicknames higher priority

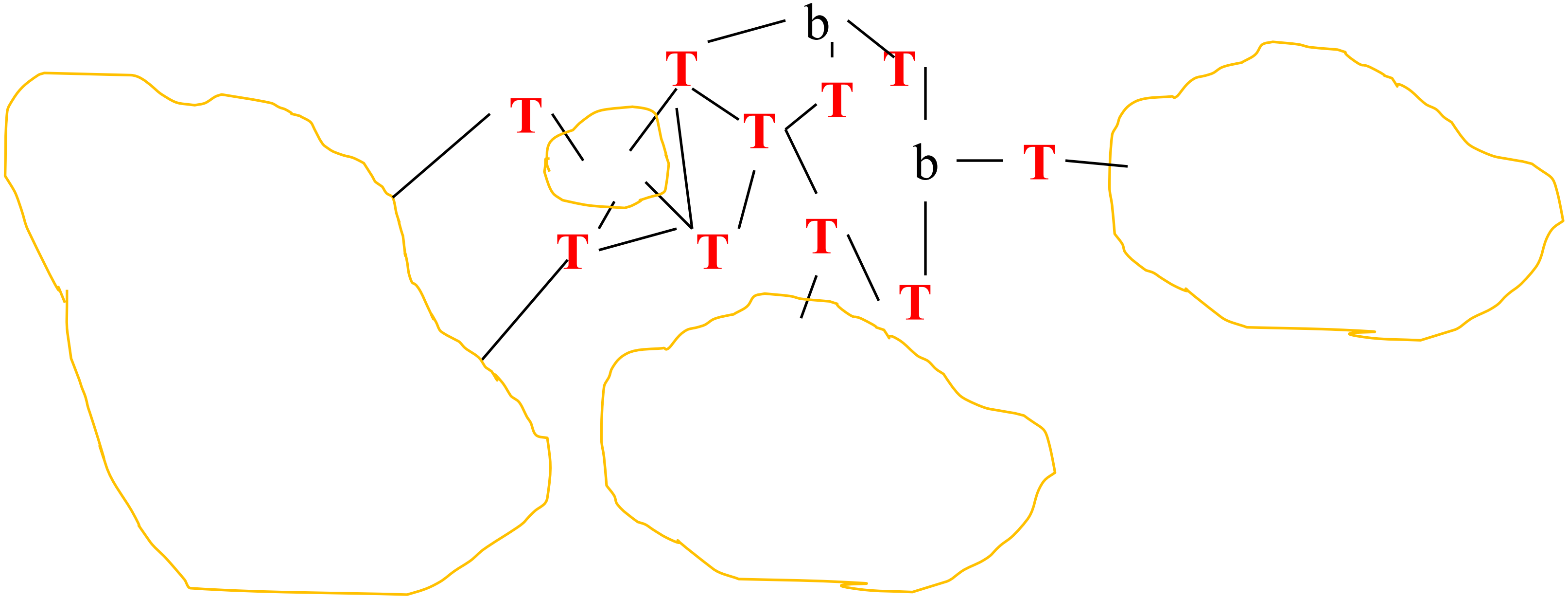
Form network of TRILL switches

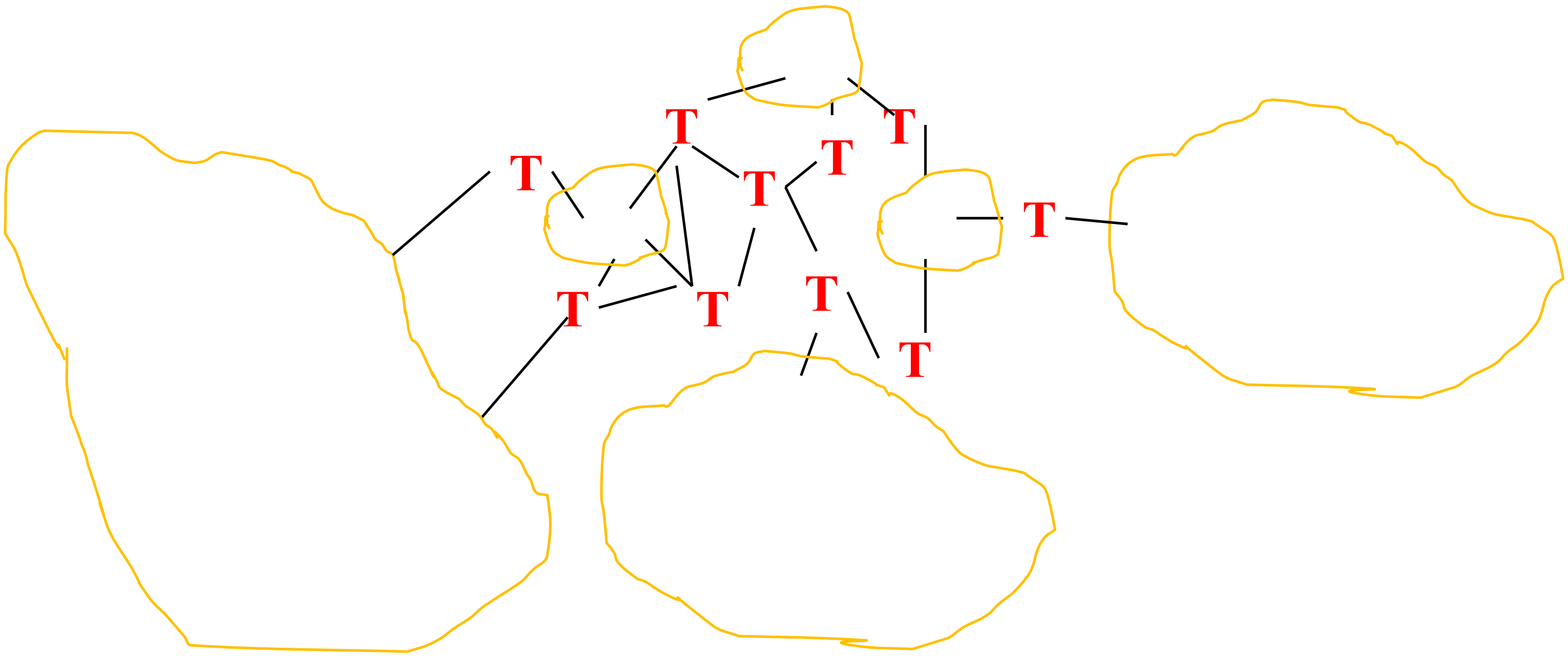
- TRILL switches find each other if:
  - Directly connected with pt-to-pt
  - Both connected to same Ethernet island
- Do “link state protocol” among TRILL switches to calculate paths to other TRILL switches

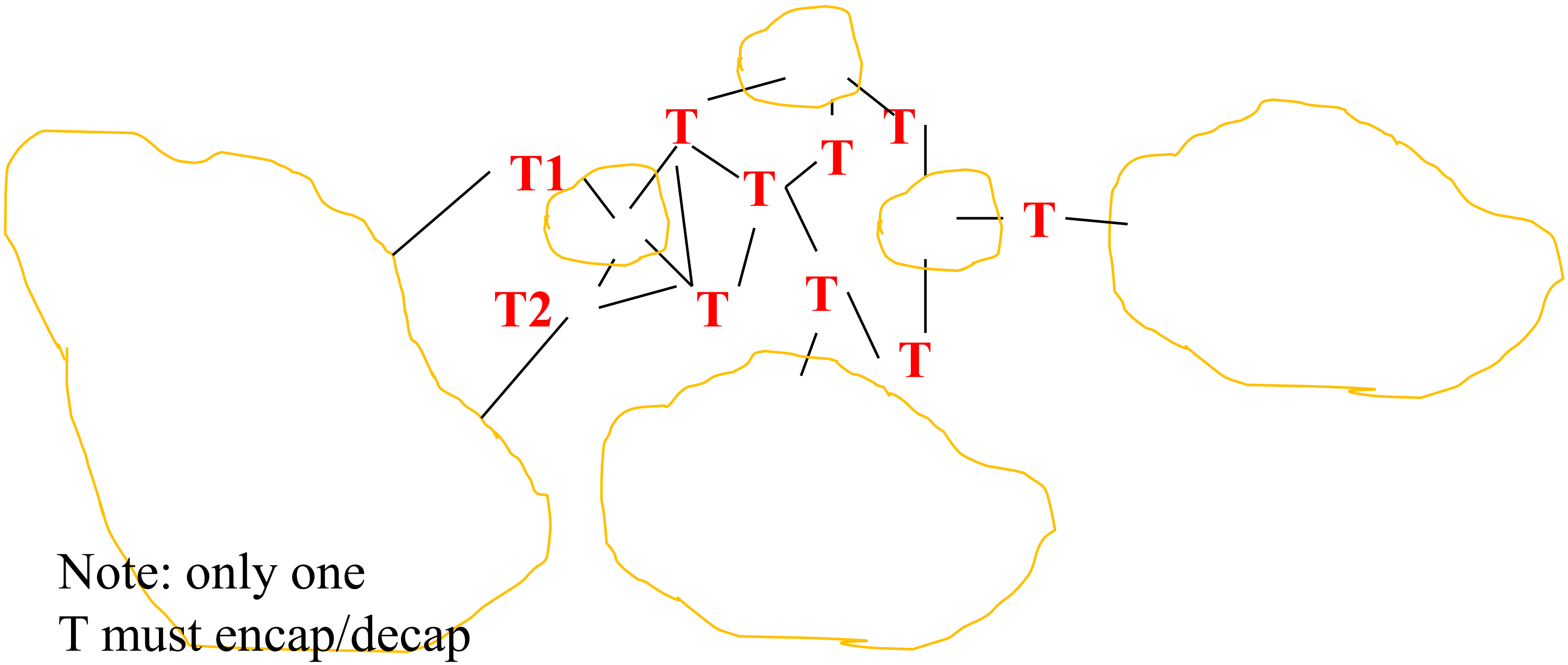

b
T
b
T
T
b
T
T
b
T
b
T
T
b
T
b
b
T
T
b


b
T
b
T
T
T
T
b
b
T
b
T
T
T
T
b
b


b
b
b
T
T
T
T
T
T
T
T
T
T
T


b
T
T
T
T
T
T
b
T
T
T
T
T


T
T
T
T
T
T
T
T
T
T
T



Note: only one
T must encap/decap
So T1 and T2 must
Find each other and
coordinate

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)

Other possibilities

- Configuration of (MAC addresses, location) into switches
- Directory listing (IP, MAC, switch location)
  - Consulted by first switch, or hypervisor, or VM, or application
  - No reason endnode couldn’t encapsulate into TRILL header, using switch’s nickname as “first switch”

Directory

- Could act as the DHCP server (knows (IP, MAC) because it hands them out..can learn switch location based on encapsulated DHCP request
- But what if MAC moves? Short DHCP leases?
- Could remember who requested an entry, and tell them if info changes

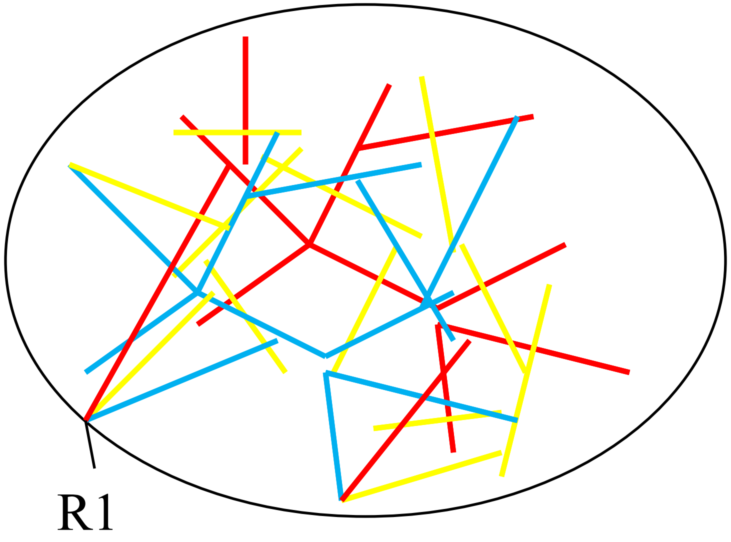
Use of “first” and “last” RBridge in TRILL header

- 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

TRILL and Multicast

- For spreading multicast traffic around, campus computes several trees
- "Last TRILL switch" field in TRILL header specifies which tree to send on
- Traffic filtered in the core based on VLAN, and IP multicast addresses

Multiple trees for multicast



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

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

Other networking topics

Some recently-coined buzzwords

- OpenFlow
- Software Defined Networking

Latency

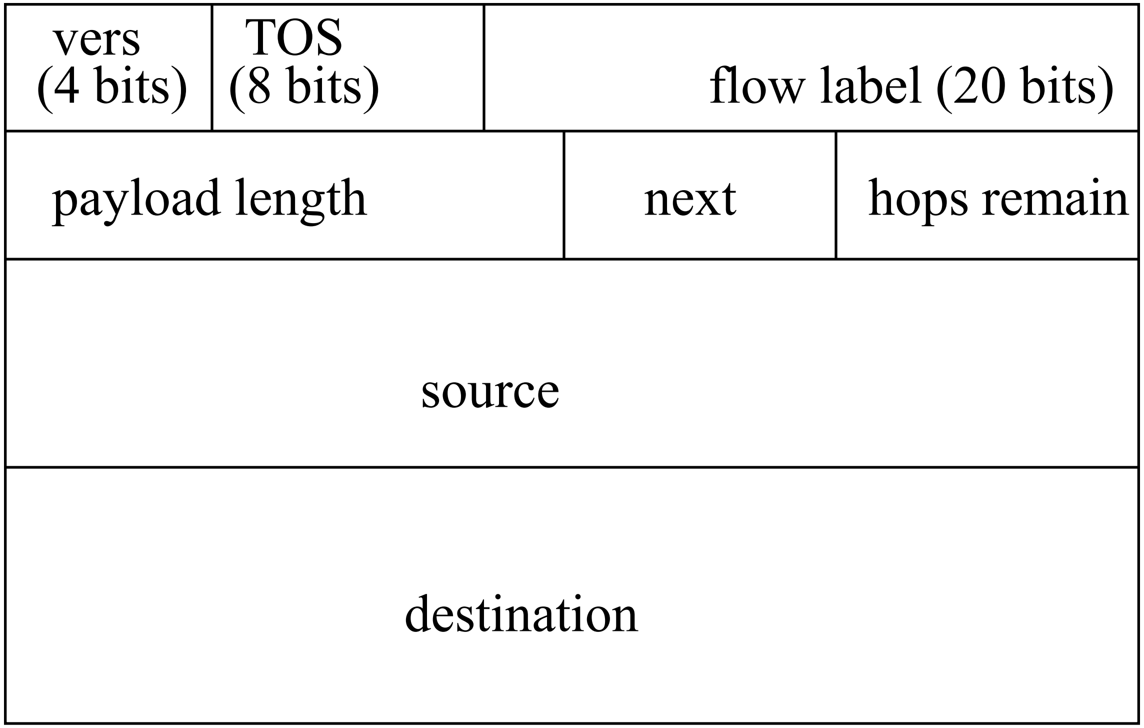
Latency

- Cut-through vs store-and-forward
  - Somewhat complicated by different speed links
  - Even if all links same speed, if you interleave packets, a congested link is the same as a slower-speed link
  - You don’t know until the end if there was a bit error, so you’ll have fragments wandering around
  - Note: you can’t start cut-through until you can make a forwarding decision

IPv4 data packet

	version	hdr lnth
	TOS	
2	total length	
2	pkt id	
	df mf	offset
	offset (cont’d)	
	TTL (time to live)	
	protocol	
2	hdr checksum	
4	source	
4	destination	

IPv6



vers (4 bits)

TOS (8 bits)

flow label (20 bits)

payload length

next

hops remain

source

destination

Another example:

- TCP has a checksum…in the header…
  - Can’t start transmitting until you see all the data

Large Control Messages

What if message is too large to fit in a link frame?

- Usual technique: use IP fragmentation

What if message is too large to fit in a link frame?

- Usual technique: use IP fragmentation
- Problem
  - Can’t process until message is reassembled
  - If one fragment is lost, have to retransmit the entire thing

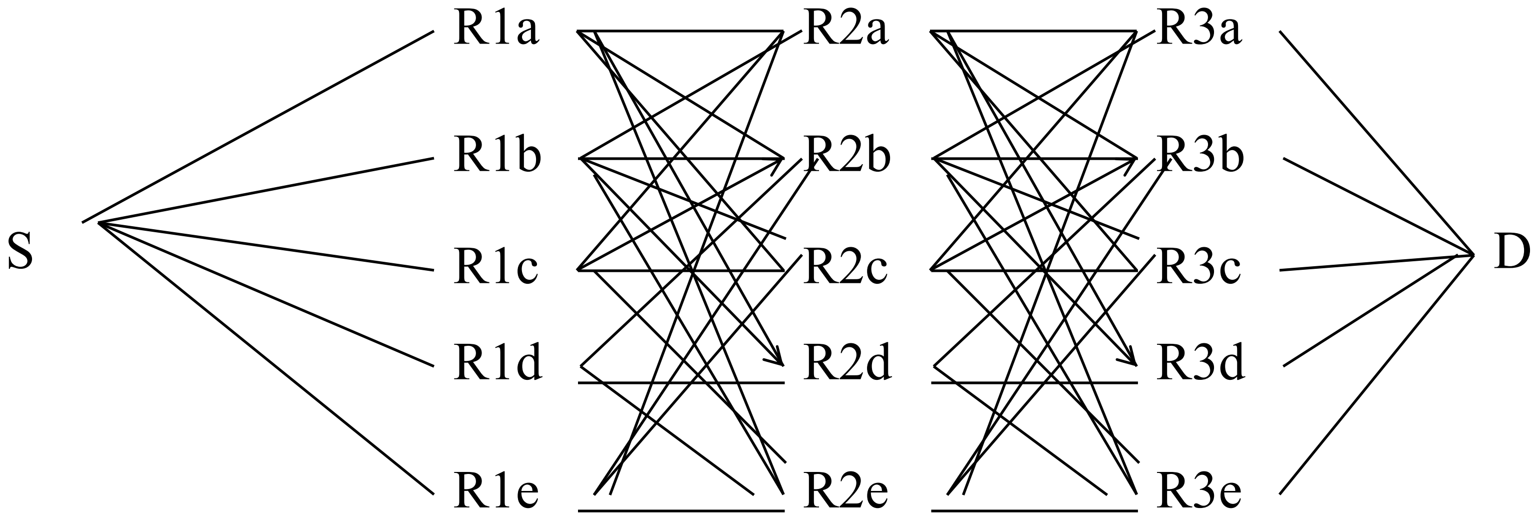
IS-IS trick

- Encode message into pieces, each of which is self-describing
- Example:
  - – Hello lists all neighbors…suppose too many?
    - Sort them
    - Have each Hello list a subset “neighbors with IDs between x and y”

Keeping packets in order

- Today’s routers/switches stand on their head to keep things in order
- Customers would complain if they reordered
- Because endnode implementations freak out

Exploiting parallel paths



How to keep packets in order

- Infiniband:
  - forwarding table has one port /destination
  - For multiple paths, assign destination multiple addresses, occupy multiple fwd-tbl entries
- IP/TRILL
  - Forwarding table has multiple ports/dest
  - Do hash of (source, TCP ports, …) to always choose same next hop
  - IEEE is proposing an “entropy” field. IPv6 “flow-ID”

What’s the entropy/flow-ID field for?

- Source (or first switch, or whatever) computes hash of whatever fields…saves intermediate switches the work of doing deep packet parsing
- Source can also diversify its traffic paths if the source knows which things can be reordered within its conversation, even if all packets use the same TCP ports

In-order delivery constraint lowers fabric performance

- Constrains all pkts of flow on same path
- What if the fabric has lots of parallel paths?
- Wouldn’t it be better to let packets exploit parallel paths, even for the same flow?
- Wouldn’t it be better if a switch could avoid congested links by choosing a less loaded “next hop”?

Chicken and Egg Problem

- Switches carefully engineered not to reorder, because endnode implementations don’t cope
- Endnode implementations (even TCP, whose job it was to reorder!) are lazy and assume fabric keeps order

Congestion

Congestion

- I was always pleased not to think about it
- Then the “DEC bit”
  - “congestion experienced”
- I was told to put it into the DECnet spec…

Network Heresy

- TCP model of congestion is wrong

What’s wrong with TCP?

- Years of research assuming flows really long-lived
- Exponential decrease/additive increase of window size to settle into having n flows share one bottleneck equally
- Conservative toe-in-water when start so as not to take more than your share
- If this was ever true, no longer at these speeds

What’s wrong with TCP?

- Years of research assuming flows really long-lived
- Exponential decrease/additive increase of window size to settle into having n flows share one bottleneck equally
- Conservative toe-in-water when start so as not to take more than your share
- If this was ever true, no longer at these speeds
- Ironic work-around: open multiple TCP connections to the same destination!

Another solution: backpressure for “lossless fabrics”

- Credit-based flow control
- DCB (data center bridging): pause/resume
- Both do roughly the same thing, but pause/resume takes more buffers

Losslessness is not free

- It requires backpressure
- Which would be OK if there were separate buffer pools for every flow
- But if there is shared fate (as in “pause everything on a link” in data center bridging), then flows will be unnecessarily slowed – congestion spreading
- It also requires deadlock-free routes

Parting words

- This stuff is all fuzzy
- Don’t believe everything you hear
- To be continued (with more jokes) tomorrow (and during the conference)

Questions?