Shortest Path Bridging

Norman Finn

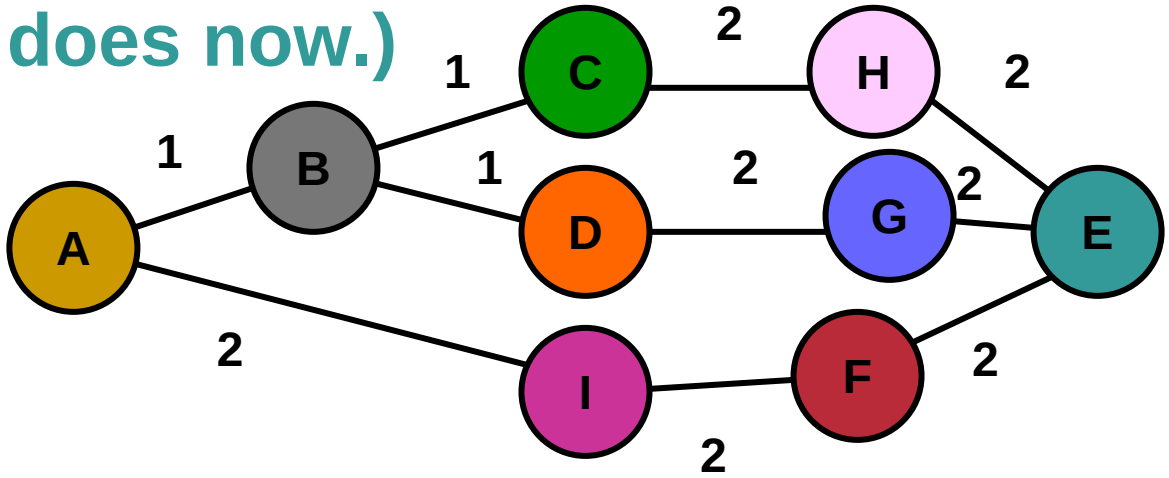
Acknowledgements

- **Francois Tallet and Michael Smith have contributed important elements to this presentation.**

Optimal Bridging Overview

Spanning Tree Uses Sub-Optimal Paths

(Well, it does now.)



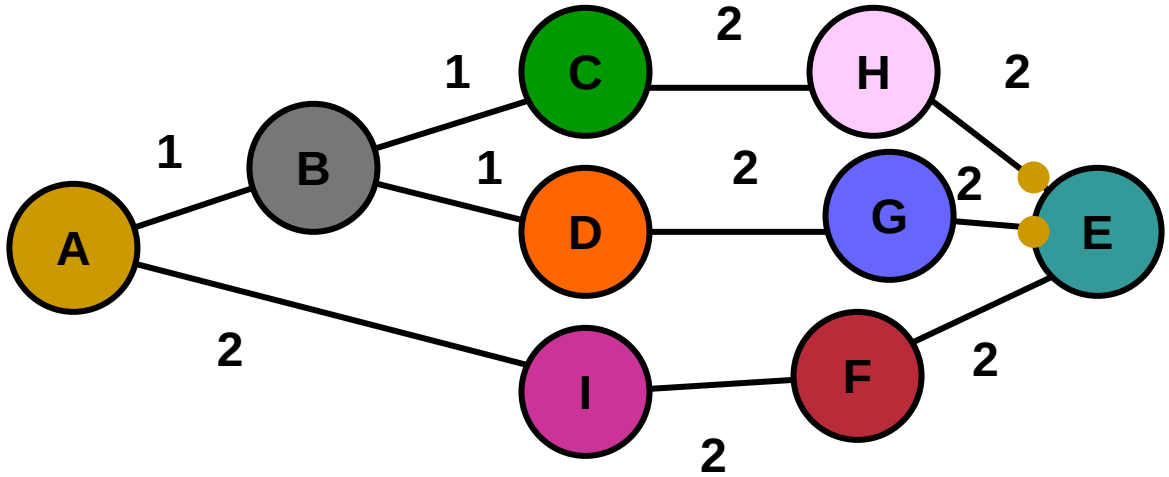
- **9 Bridges A-I, connected as shown.**

  Letters are Bridge IDs. Lower letter (A) is "better" than higher letter (D).

  Numbers are path costs.

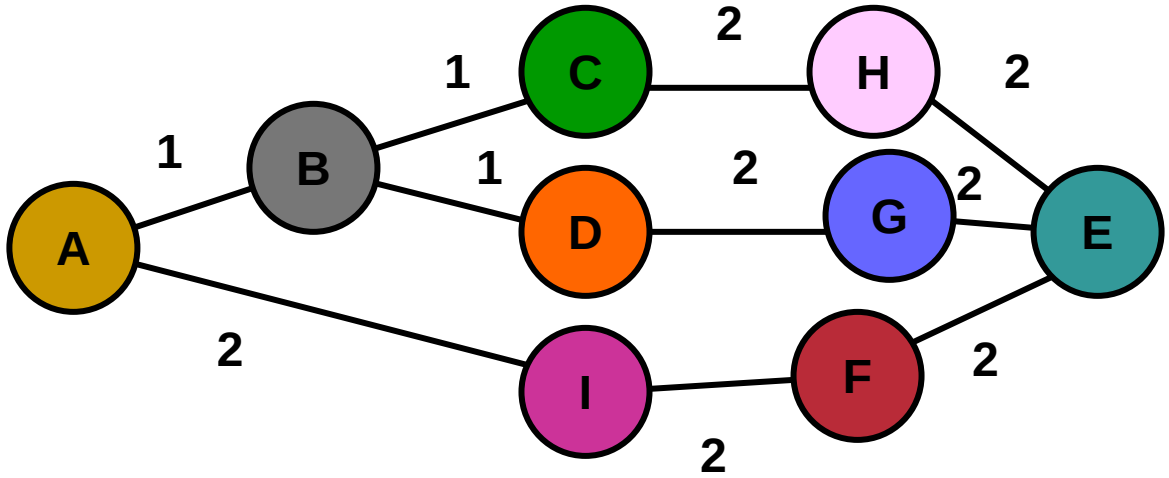
  Each pair of bridges agrees on link's path cost.

Spanning Tree Uses Sub-Optimal Paths



- **Bridge A is the Root Bridge.**
- **Bridge E breaks the two spanning tree loops by blocking the marked ports.**
- **Path from E to G is E-F-I-A-B-D-G.**
- **This clearly qualifies as “sub-optimal.”**

Spanning Tree Per Bridge

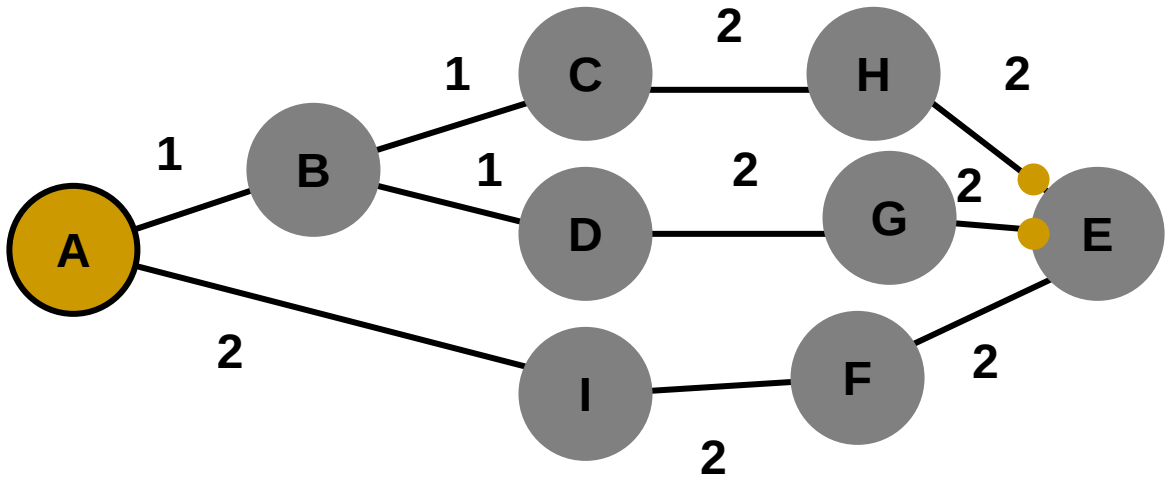


- **Instead of 1 spanning tree, we create 9 spanning trees.**

  Each bridge is the root of its own spanning tree instance (STI).

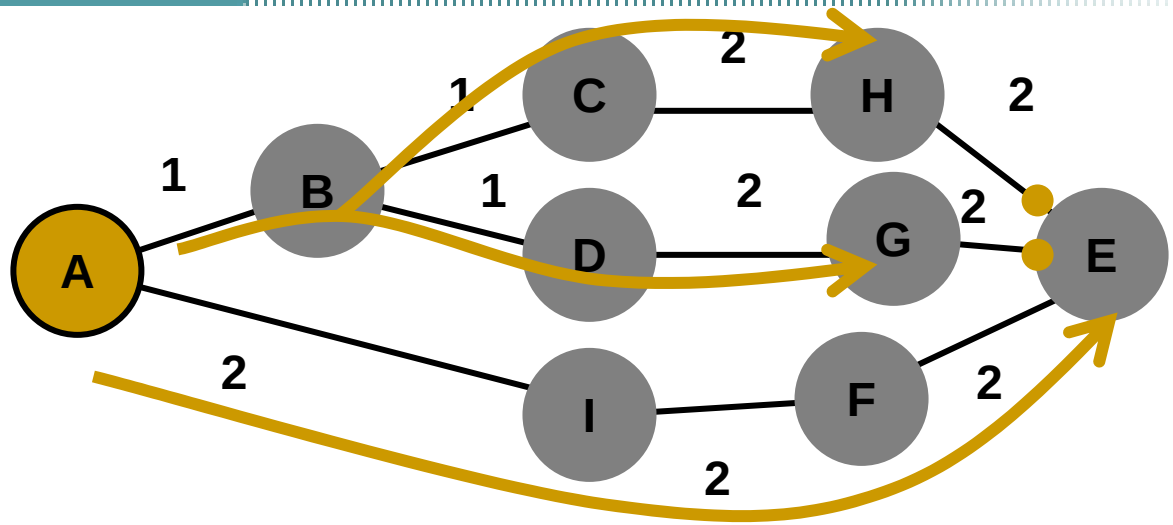
  802.1S (MSTP) supports 64 STIs trivially, 4k with modest effort.

Spanning Tree Per Bridge



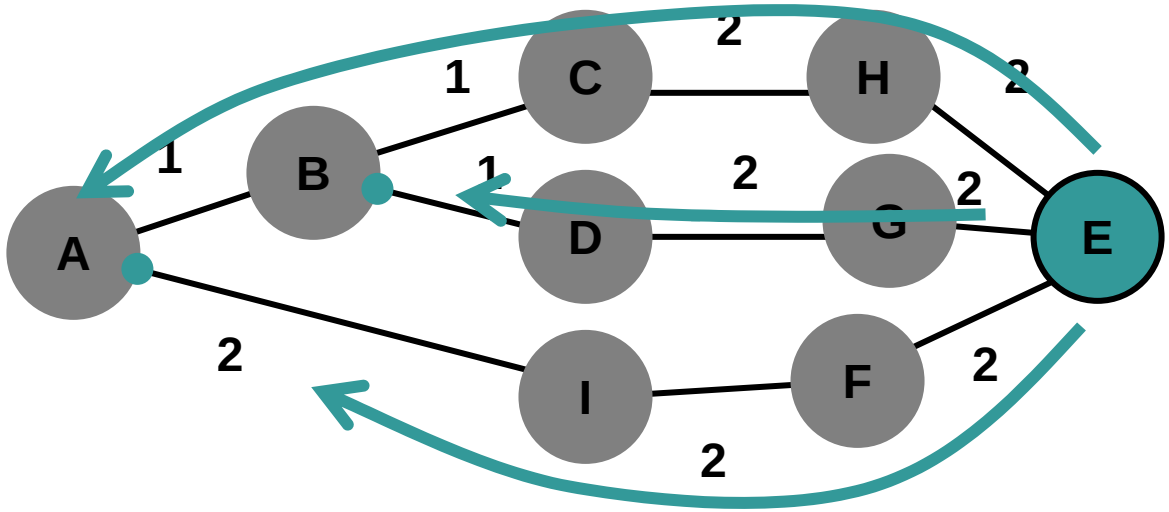
- **We assume MAC-in-MAC. The outer MAC addresses are from the set {A, …, I}.**

Spanning Tree Per Bridge



- **Whenever Bridge A sends a frame, it uses STI A.**
- **Of course, the STI with A as the root is the optimal path is always straight away from A, along the least-cost path.**

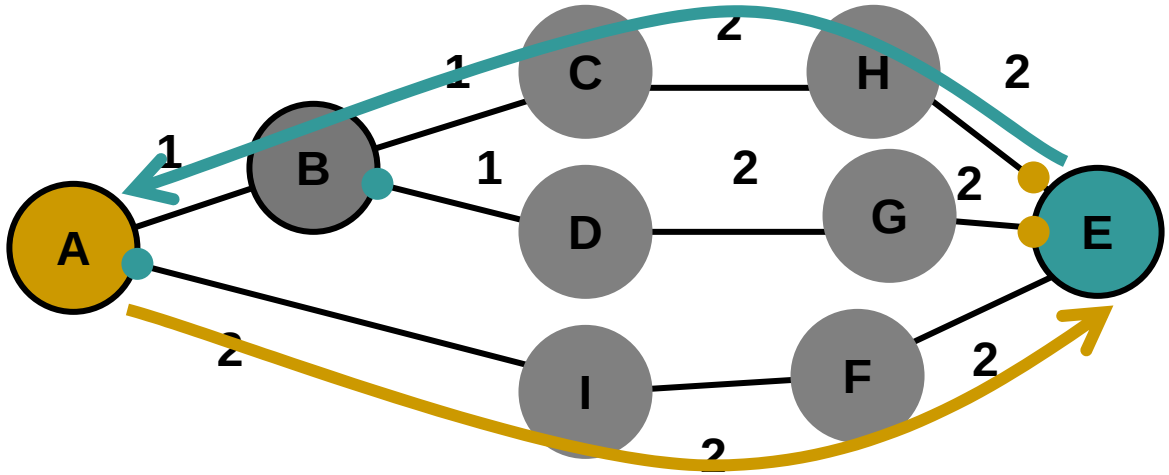
Spanning Tree Per Bridge



- **Whenever Bridge E sends a frame, it uses STI E.**
- **Of course, the STI with E as the root is the optimal path is always straight away from E, along the least-cost path.**

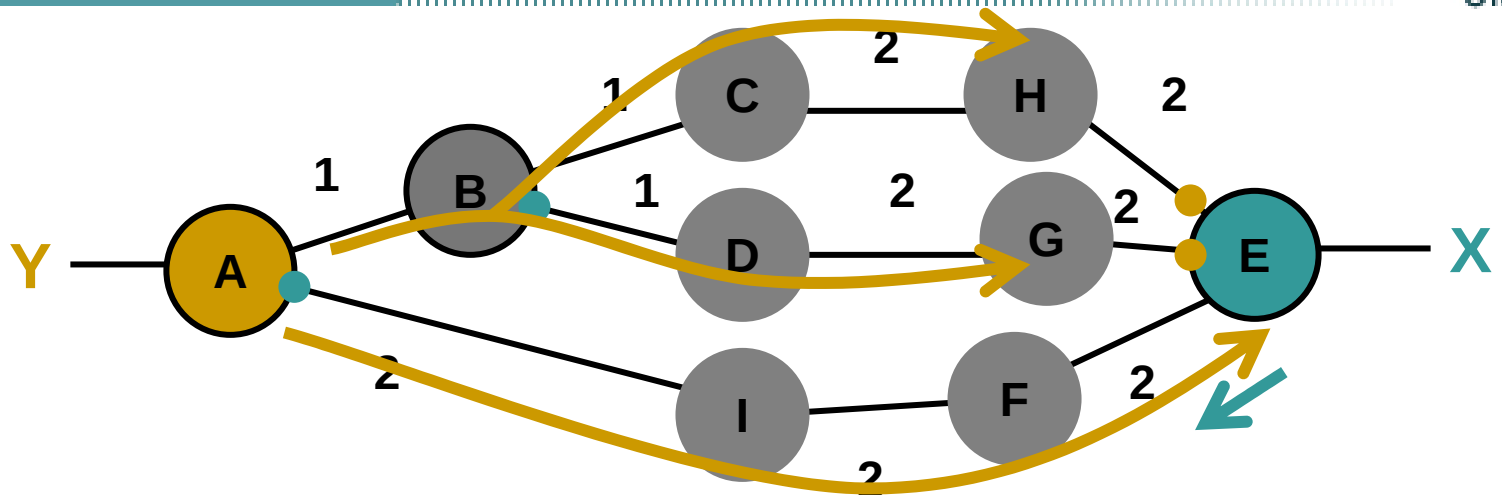
The Problem of Asymmetrical Spanning Trees

Asymmetrical Spanning Trees



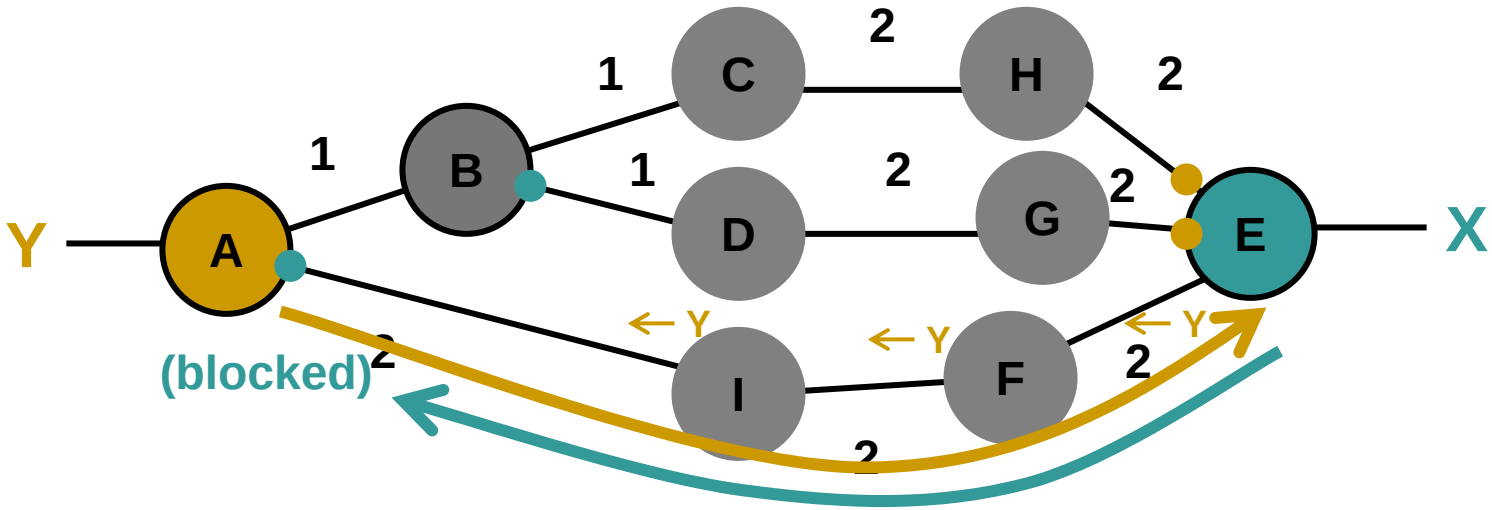
- **Suppose the VLANs rooted on Bridges A and E are blocked as shown.**
- **The path from E to A is E-H-C-B-A.**
- **The path from A to E is A-I-F-E.**

Asymmetrical Spanning Trees



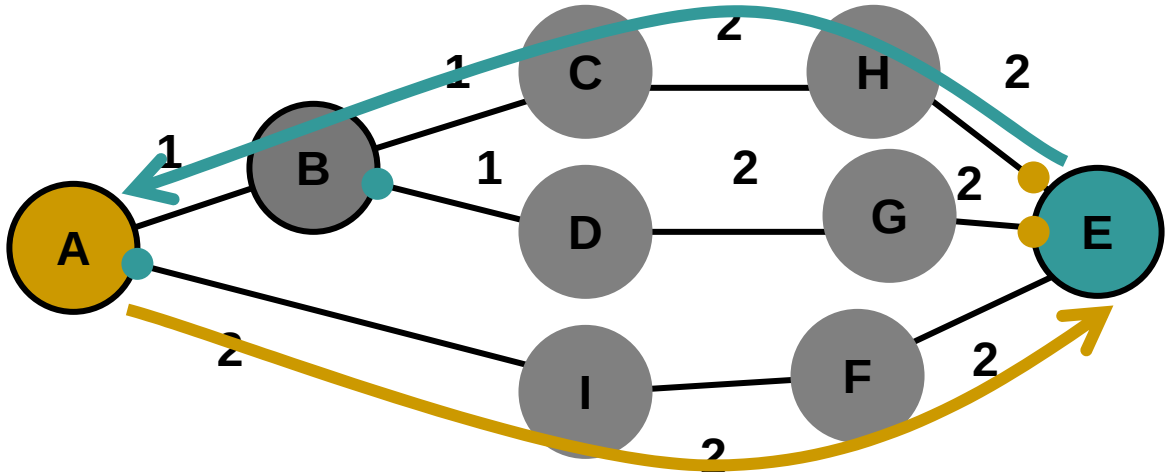
- **A receives a frame from Y for X. Not knowing where X is, it floods the frame on its own MSTI A.**
- **The frame reaches E via the F-E link, and E forwards it everywhere, including X.**

Asymmetrical Spanning Trees



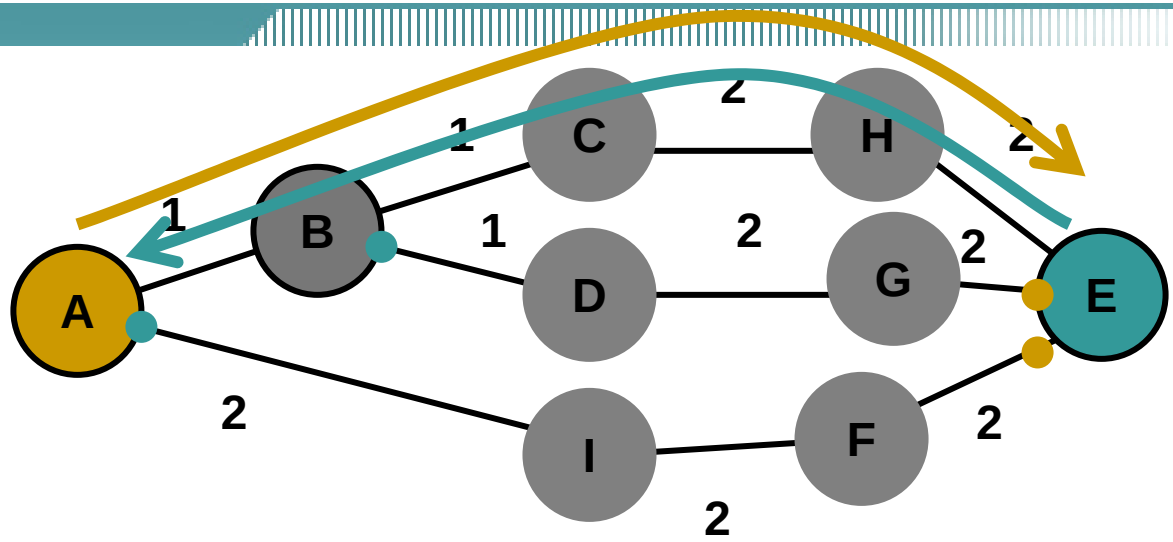
- I, F, and E all learn the direction back to Y.
- When X replies, the frame is sent E-F-I-(A) on B's MSTI.
- A discards the frame, because the port is blocked on that MSTI.

Asymmetrical Spanning Trees



- **If A transmits a frame for E towards B, the frame will hit the blocked port H-E and not get to E.**
- **Learning doesn't work.**

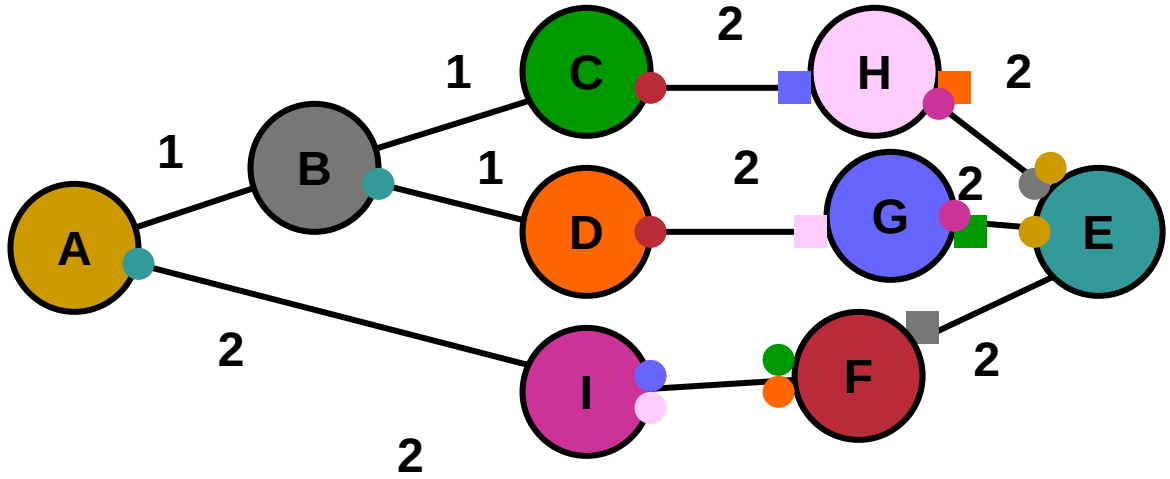
Asymmetrical Spanning Trees



- **But, if the two spanning trees are symmetrical, e.g. if A-B-C-H-E is used for both directions, then learning works just fine.**

Asymmetrical Spanning Trees

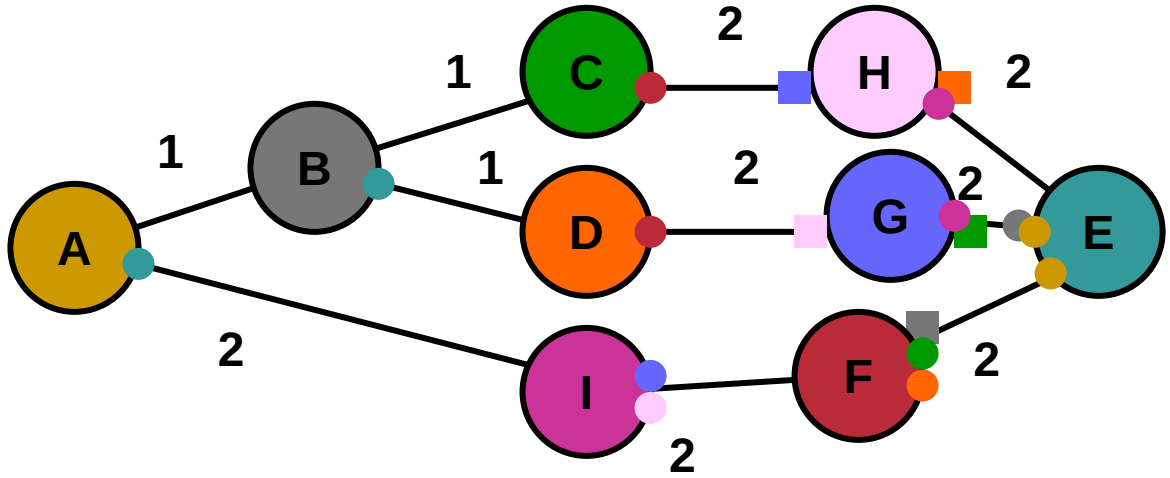
(Squares are safely blocked ports. Spots are potentially asymmetric blocks.)



- This is how the 9 STIs would be created if we do nothing.
- Many STI pairs are compatible, e.g. A-I, B-F, and D-H.
- But some are not, e.g. A-E, D-F, and B-E.

Asymmetrical Spanning Trees

(Squares are safely blocked ports. Spots are potentially asymmetric blocks.)



- **This layout would be perfect.**
- **But, how do we get there?**

Coordinating path costs and priorities

- **Because link costs can be configured, as well as computed from the link speed, the two bridges on the two ends of the link can disagree on the link cost used in the STP algorithm. This would make symmetrization impossible.**
- **Similarly, Bridge Priorities can be configured differently for different spanning tree instances, which makes them asymmetrical.**

Coordinating path costs

- **So, carry a little extra in obMSTP to ensure that both bridges use the same link cost!***

  The bridge advertises its link configured costs in obMSTP BPDUs.

  All bridges on a given LAN use the link costs advertised by the CSTI Designated Bridge.

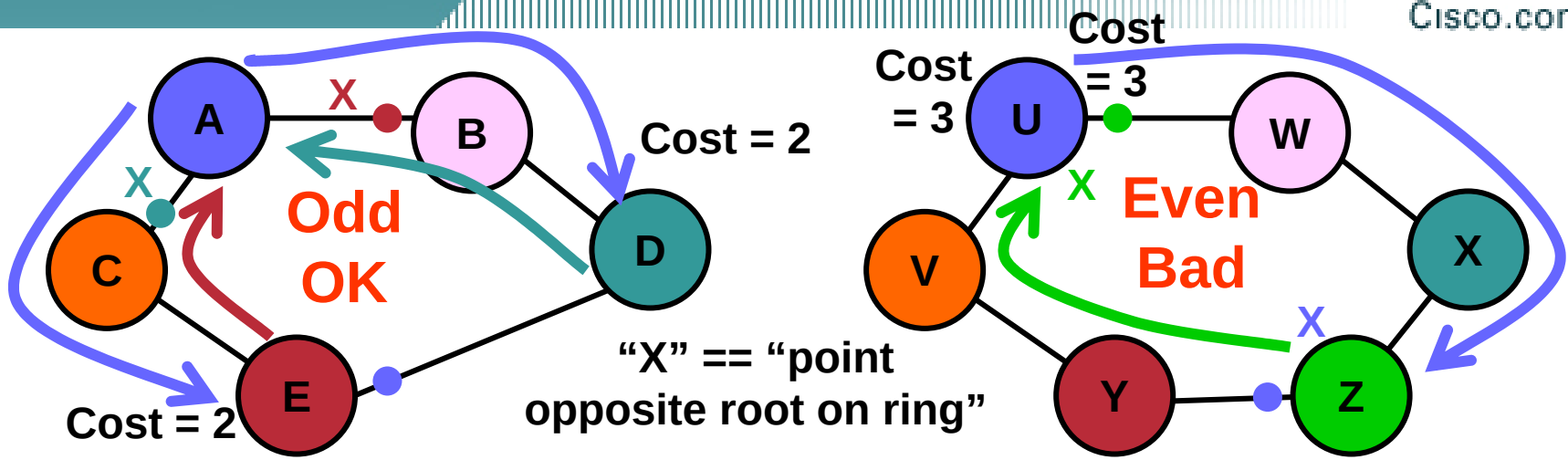
  The CSTI does *not* play this game.

- **One link cost parameter is required for each M-Part.**

- **Also, bridge's bridge priority must be the same in all STIs that must be symmetrical.**

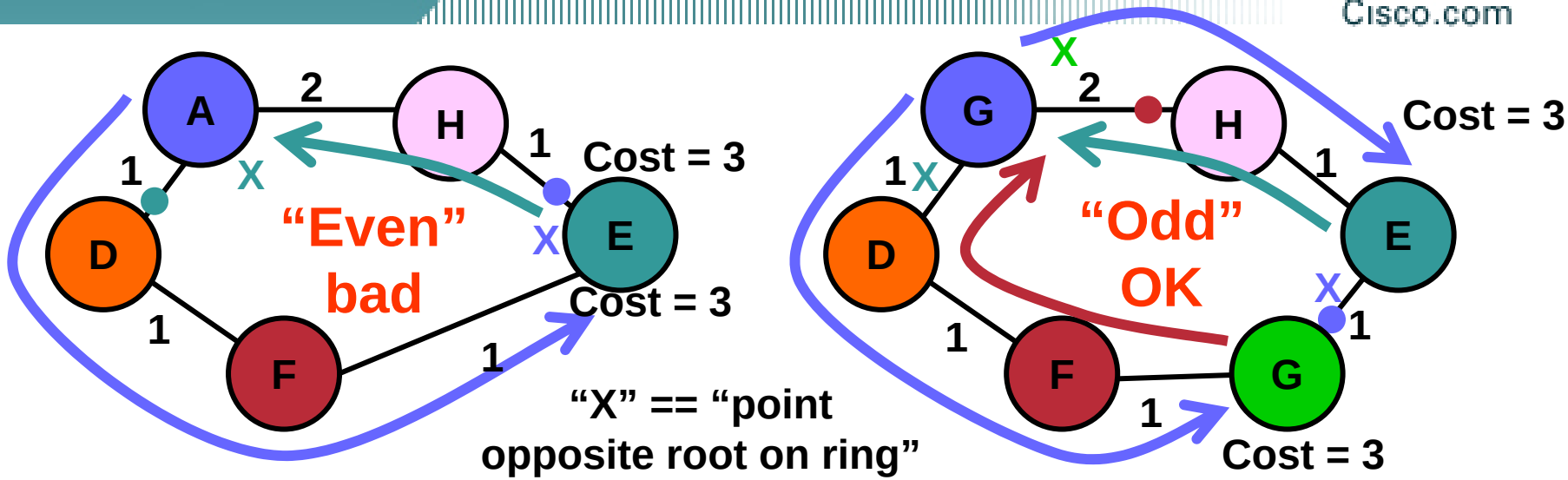
*** There are other solutions, as well.**

Identifying the core of the problem



- **When can STP break loops asymmetrically?**
- **Assuming the same cost per link, a ring with an odd number of nodes cannot have a problem; routes are always symmetrical.**
- **A ring with an even number of nodes has two equal paths from one side to the other, and so can have asymmetrical paths.**

Identifying the core of the problem

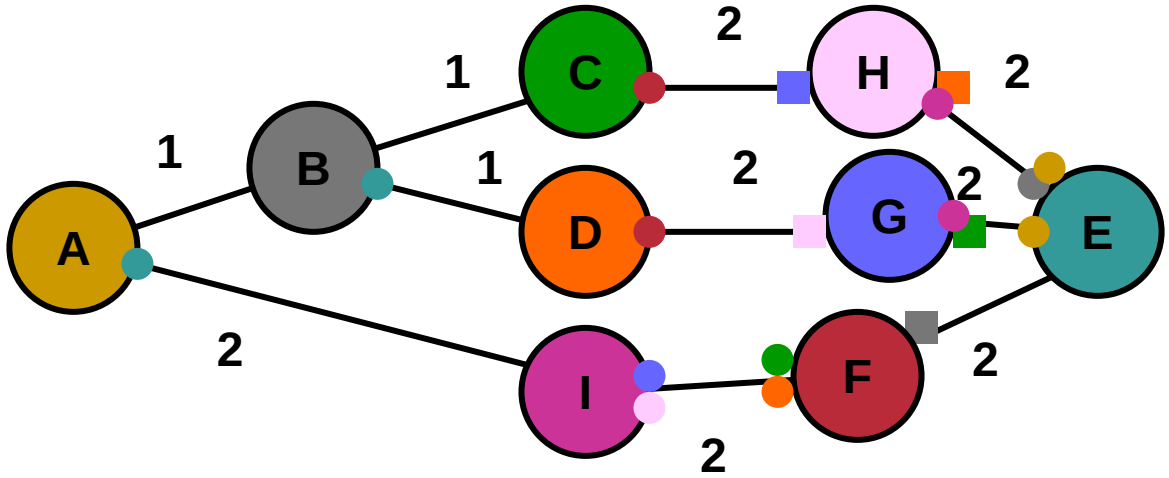


- **The addition of link costs changes the definition of "odd" and "even", but not the nature of the problem.**
- **If the "point opposite me on this loop" (X) is a link, I'm OK, and if it's a bridge, I can have asymmetrical paths.**

But, Root Port selection can be changed!

- **It has been observed often, in IEEE 802.1, that when a bridge has equal root path costs to choose from, any decision it makes for its Root Port is perfectly compatible with the spanning tree algorithms.**

Problems are always visible at both ends



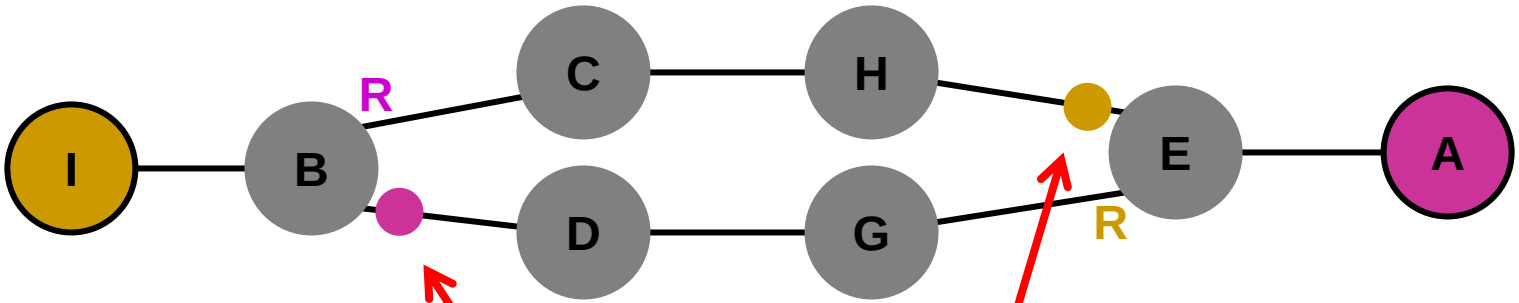
- **As long as the link costs are forced to be symmetrical:**
- **If any Bridge “X” has an equal-cost root port choice on STI Y, then Bridge Y has an equal-cost root port choice on STI X!**

The Path Vector

The Path Vector

- If both ends know about the problem, they should be able to do something about it!
- So, we add, for each STI in the obMSTP BPDU, a Path Vector containing one bit of information about each Bridge in the network.

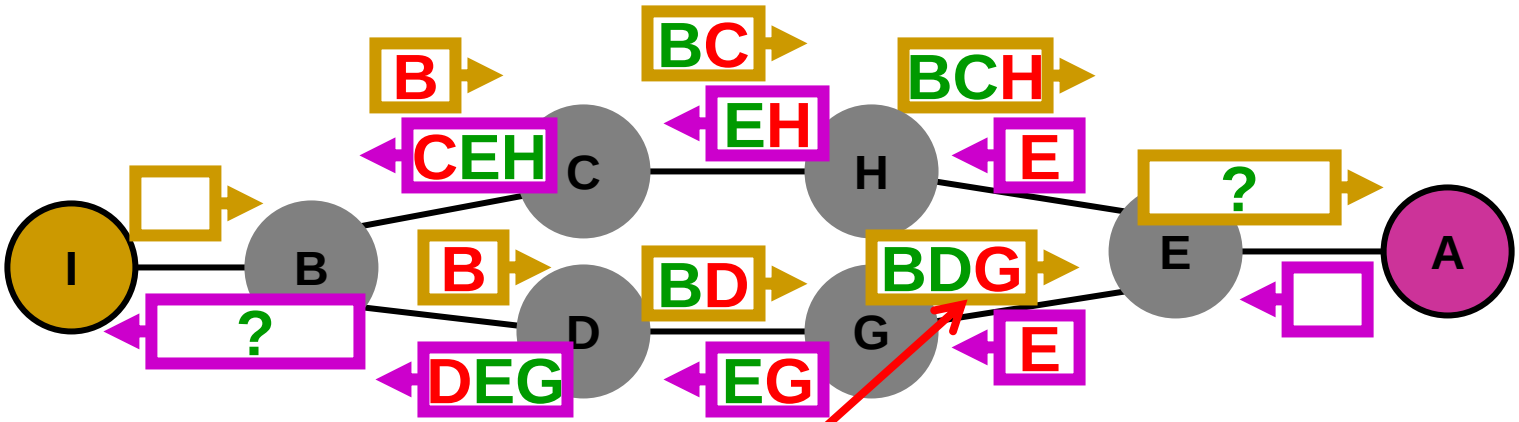
Propagating the Path Vector



- **A slightly different diagram lets us flesh out the details of the Path Vector.**
- **Looking at the STIs of Bridges A and I, we see that Bridges B and E will each want to block one port for one of the two STIs.**

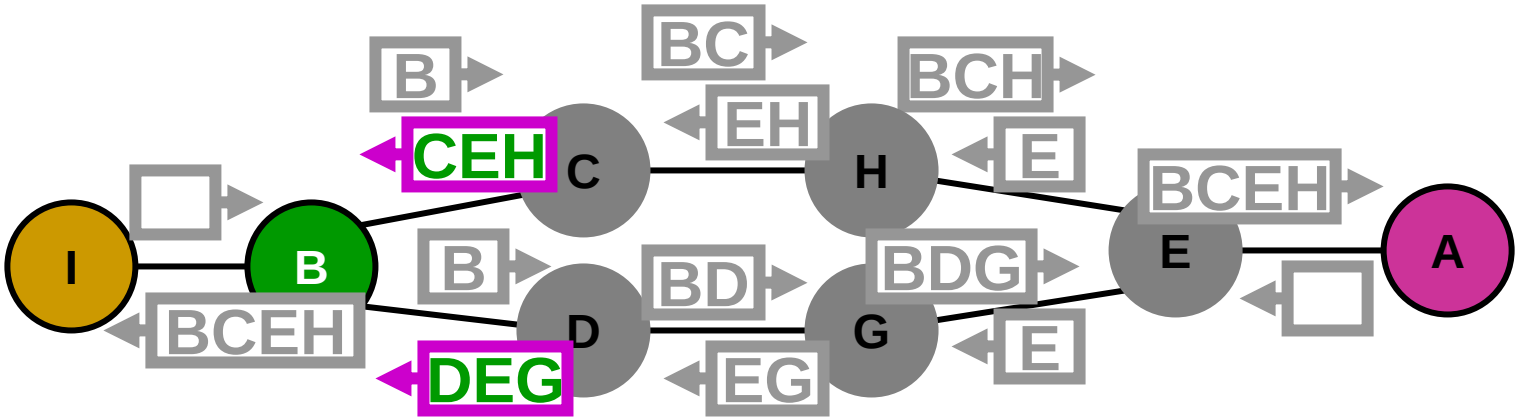
And, they block the wrong ports.

Propagating the Path Vector



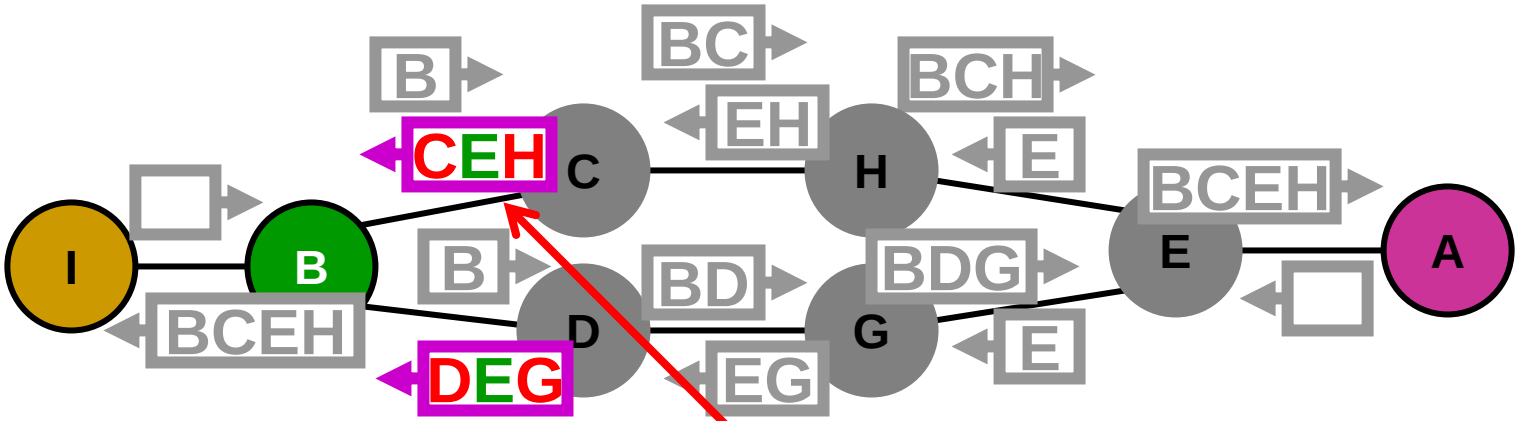
- **Each Bridge, as the Root of its own MSTI, initiates an empty Path Vector.**
- **Each Bridge "owns" one bit in every PV.**
- **As the PV is propagated, each Bridge adds its own bit to every other Bridges' PV.**

Propagating the Path Vector



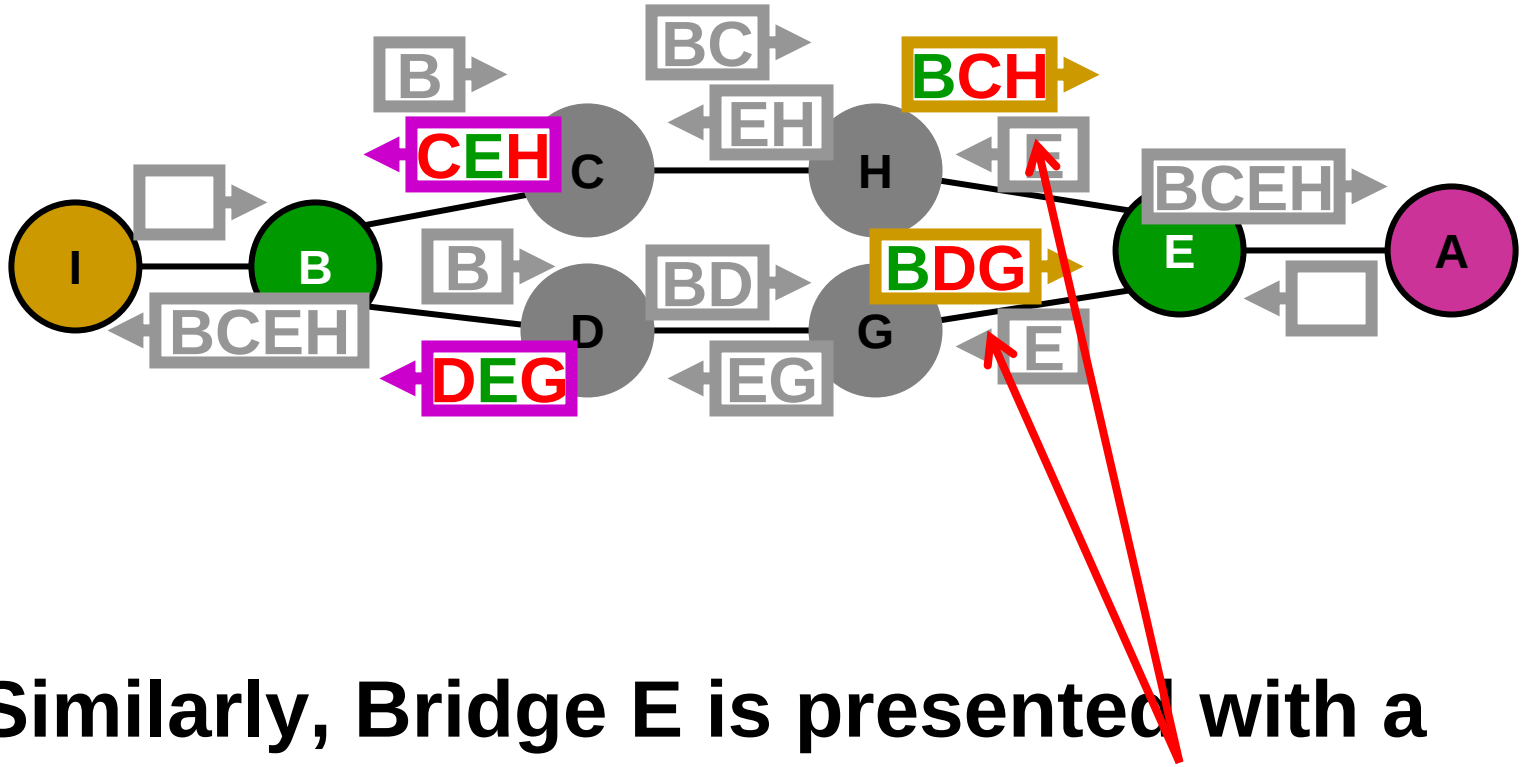
- **Bridge B has two equal-cost paths to Root Bridge A.**
- **B uses the Path Vector, instead of the Bridge IDs (C and D) to pick its Root Port for MSTI A.**

Propagating the Path Vector



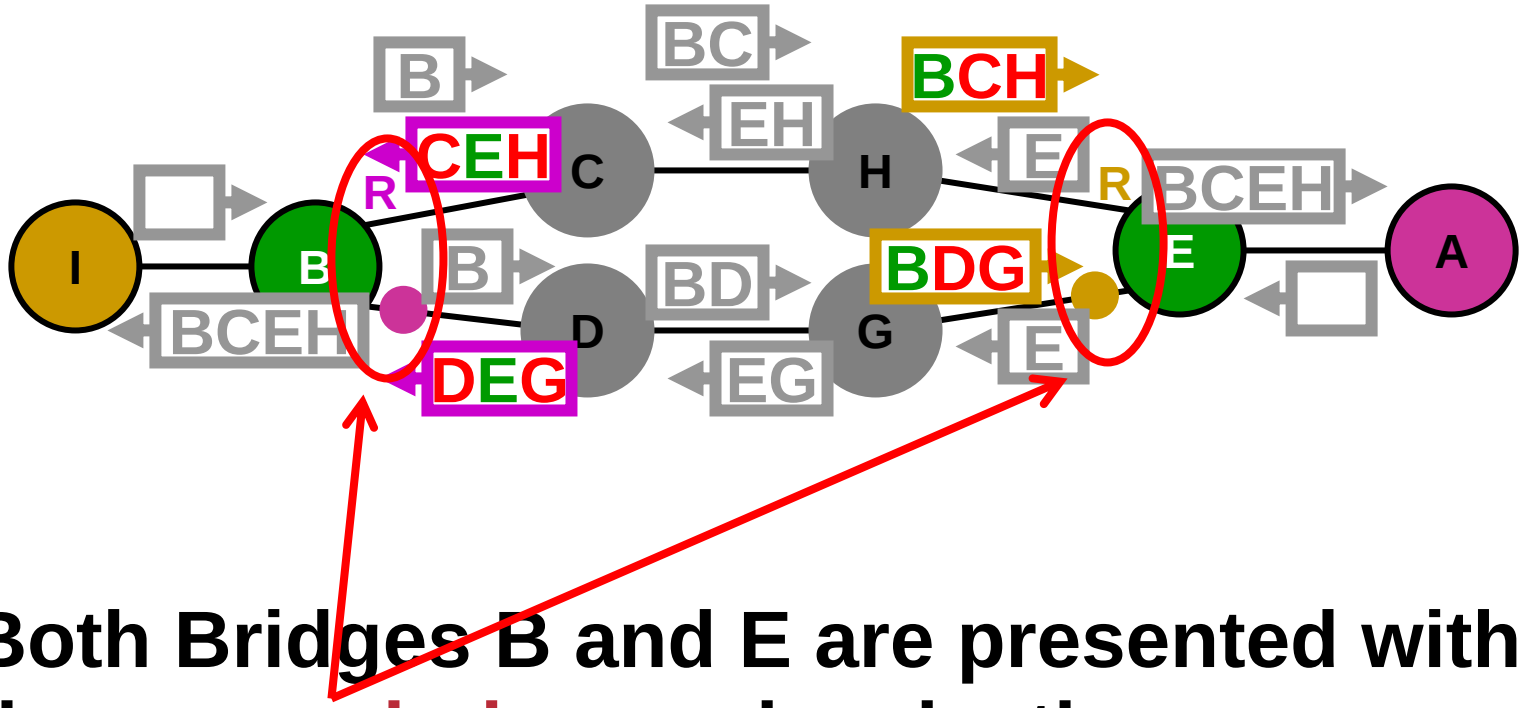
- **Bridge B compares the PVs as binary numbers.**
- **The PVs differ only in the different routes taken across the loop – only Bridges C, D, G, and H can be different in the PVs seen by B on the equal-cost paths.**

Propagating the Path Vector



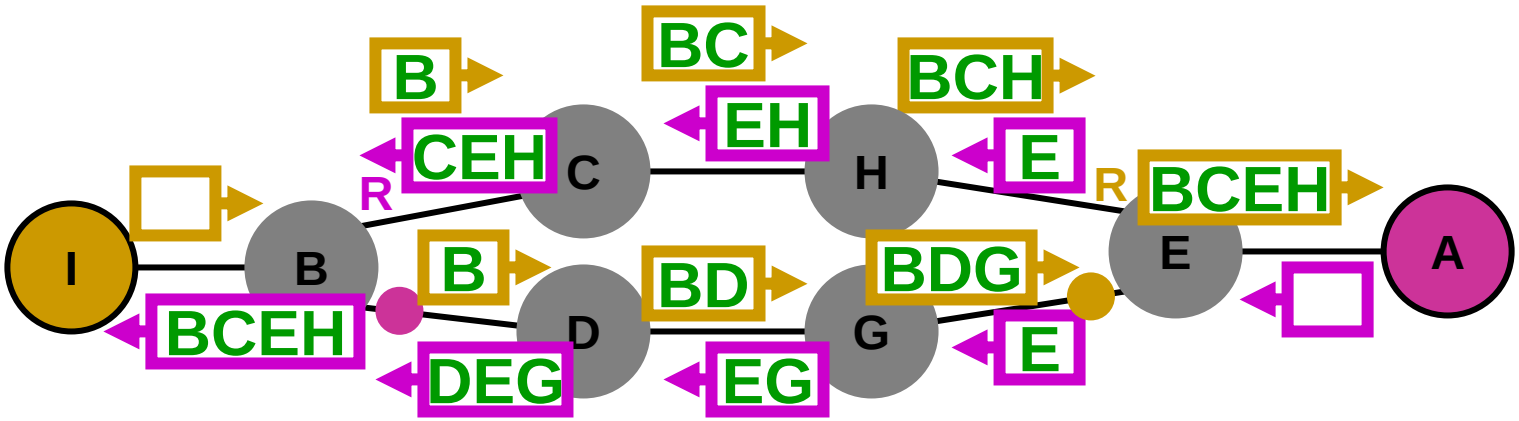
- **Similarly, Bridge E is presented with a choice, and that choice also depends on Bridges C, D, G, and H.**

Propagating the Path Vector



- **Both Bridges B and E are presented with the same choice, and make the same decision.**
- **The Root Ports for MSTIs A and I match!**

Propagating the Path Vector



- **So, the Bridges choose symmetrical paths in a single pass of the Path Vectors across the network!**

Priority Vectors

- **In IEEE Std. 802.1Q-2003, the Regional Root Port for MSTI X is the port on which one is receiving the best priority vector from the Designated Bridge of MSTI X:**

  {Root ID, Root Path Cost, Bridge ID, Port ID}

  where the Bridge ID and Port ID are those of the Designated Bridge in MSTI X.

Priority Vectors

- **We modify this vector by replacing the Bridge ID with the Path Vector in MSTI X's priority vector:**

  {Root ID, Root Path Cost, Bridge ID, Port ID}

  becomes:

  {Root ID, Root Path Cost, Path Vector, Port ID}

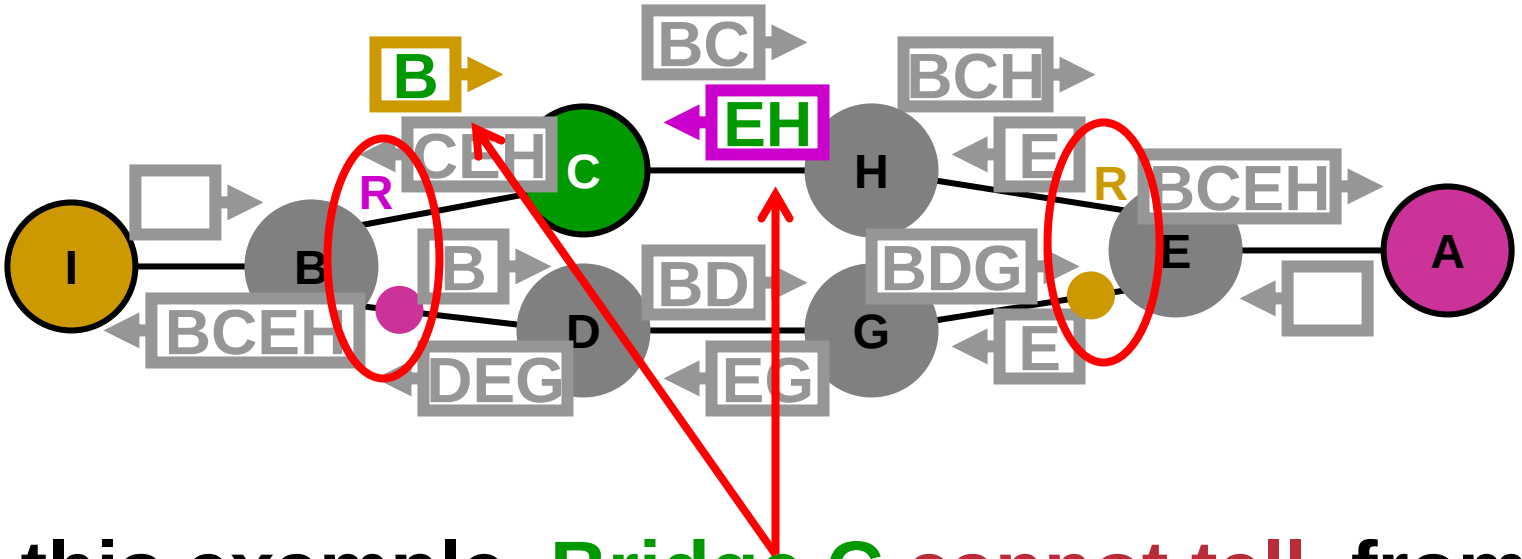
- **(This is how you implement the decision, inside the bridge, that makes the trees symmetrical.)**

The Reflection Vector

The Multicast Problem

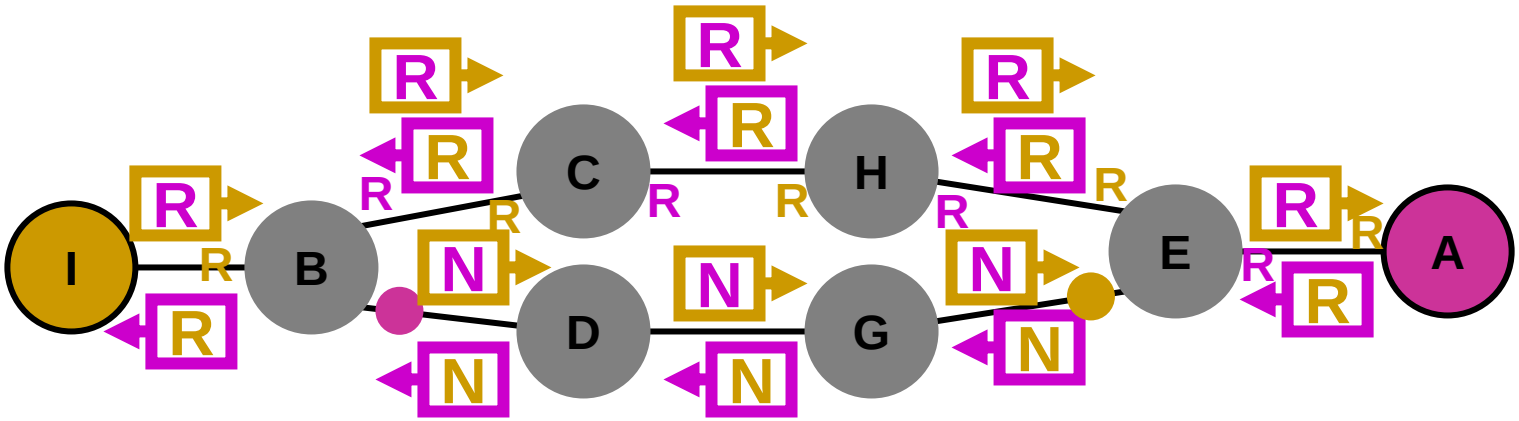
- **Every frame is transmitted on the tree rooted at the originating Bridge.**
- **How does Bridge B decide on which ports to distribute a multicast group address G on MSTI A, given that it knows the Bridge C wants G?**
- **If B knew whether it was on the MSTIs between A and C, it knows the answer to the multicast group address question.**

The Multicast Problem



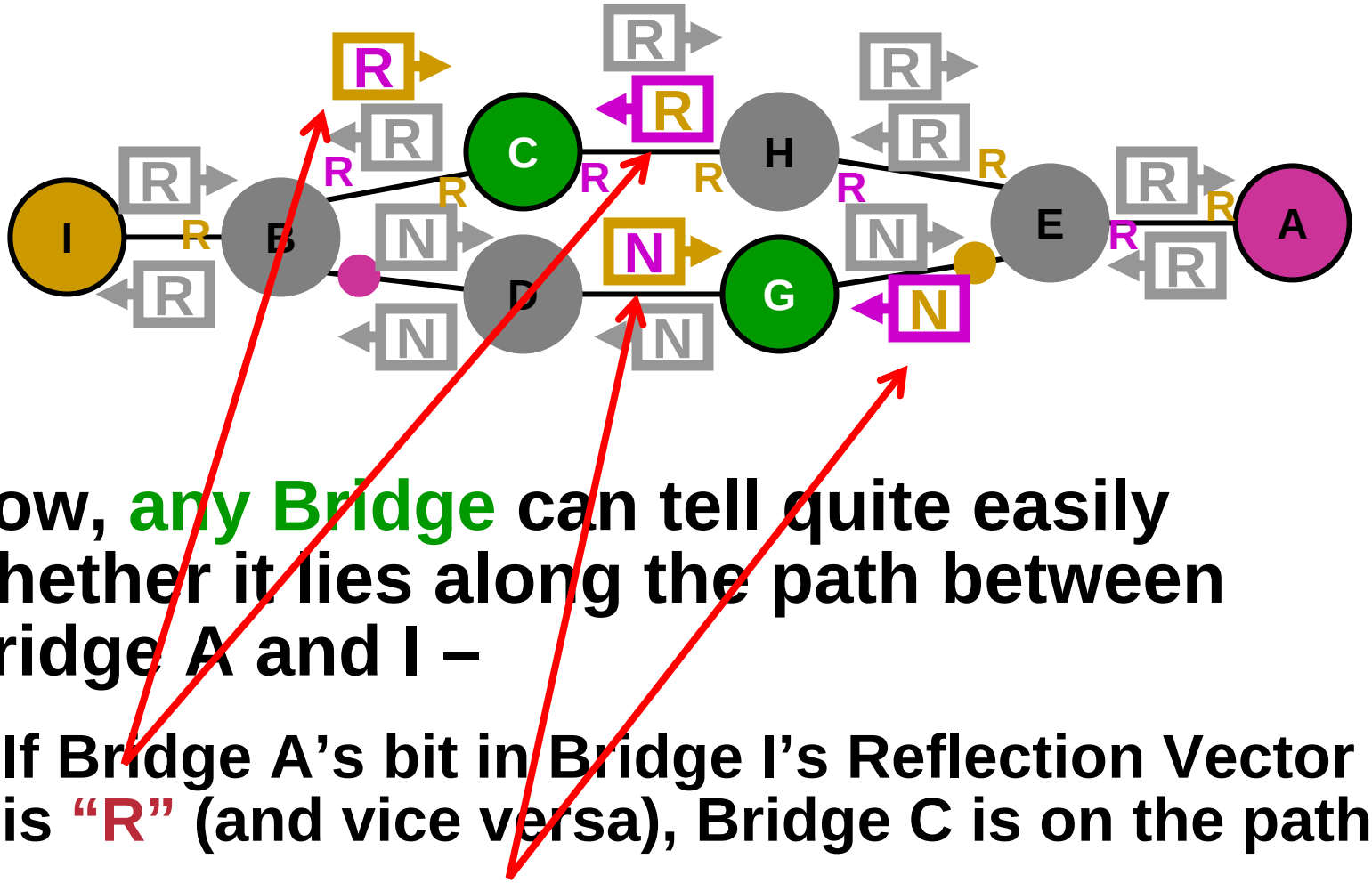
- **In this example, Bridge C cannot tell, from its two inputs, what decision Bridges B and E will make for their equal-cost paths between Bridges A and I.**
- **That is, bridge C cannot tell whether or not it is on the path between Bridge A and I.**

Propagating the Reflection Vector



- **Each Root Bridge initiates a Reflection Vector with one bit for each other MSTI.**
- **The bit says the RV has (R) or has not (N) passed only through Root Ports of the bit's MSTI along the RV's path from the RV's Root Bridge.**

Propagating the Reflection Vector



- **Now, any Bridge can tell quite easily whether it lies along the path between Bridge A and I –**

  If Bridge A's bit in Bridge I's Reflection Vector is "R" (and vice versa), Bridge C is on the path.

  If they have an "N", it is not.

Propagating the Reflection Vector

- **Furthermore, since the Path Vector ensures that all MSTIs are symmetrical, we don't need both Bridge A's bit in Bridge I's Path Vector *and* Bridge I's bit in Bridge A's vector – we only need one of them.**
- **That cuts the Reflection Vector's size in half, which helps keep the BPDU size down.**
- **One frame can hold a BPDU for a 64 node network.**

Generating the Reflection Vector

- **Each Bridge maintains a Reflection Vector for each MSTI. The Reflection Vector has one bit for each other Bridge with MSTID less than the originating Bridge's MSTID.**
- **A 1 indicates "Root in this direction", and a 0 indicates "Root not in this direction".**
- **The Reflection Vector for MSTI X is set to all 1s in the Root Bridge for MSTI X.**

Convergence Times

Dynamic Behavior

- **The worst-case time for MSTP convergence is when a Root Bridge disappears.**
- **This cannot happen now, since if a Root Bridge disappears, its MSTI is irrelevant.**

Dynamic Behavior

- **Not only that, but the Path Vector prevents the "counting to infinity" problem common to distance vector algorithms such as MSTP and RIP (Routing Information Protocol).**
- **If an MSTI message received by Bridge X contains a Path Vector with Bridge X's bit set, it is ignored, since that vector represents stale information that is looping.**

Dynamic Behavior

- **Thus, the “network diameter” (MaxAge) need not be configured to be close to the size of the network in order to prevent excessive hop counts.**
- **This removes another potential source of network failures: a configured “network diameter” that is smaller than the actual size of the network.**

Dynamic Behavior

- **So, the Path Vector typically converges in a single pass across the network, and occasionally, two passes (when stale information loops once).**
- **The Reflection Vector converges in, at most, one additional pass.**

Announcement Packets

Announcement Packets

- **Each Bridge can send Announcement Packets advertising all of the {Group address, Community Part} pairs wanted by its locally-attached edge ports (learned via GMRP or IGMP).**
- **It includes the list of VLAN Community Parts that are required by its locally-attached edge ports (learned via configuration or GVRP).**

Announcement Packets: Alternatives

- **Announcement packet information can be carried several different ways:**
  - **As messages that are ordinary multicasts in the data plane.**
  - **As a hop-by-hop flood, similar to VTP or OSPF.**
  - **Along with Link State routing information.**
  - **As a hop-by-hop flood along the CIST.**

Announcement Packets

- **Every bridge B installs**

  the {Group MAC address, Community Part} pairs in the Filtering Database; and

  the VLAN filters in dynamic VLAN database,

- **based on the facts that:**

  the VID identifies that the frame came from the Root of MSTI A;

  bridge C does (or does not) want that frame's VLAN or {Group, Community}; and

  bridge B is (or is not) on the path from A to C.

{S,G} state

- Let X_C denote the Community Part of VLAN ID X, and X_T denote the MSTI ID to which VLAN ID X is assigned.
- Each bridge acquires, from the announcement packets, a list of {G,C} pairs, which are the G parts of a router's {S,G} state.

{S,G} state

- In order to keep down the number of entries in the bridges' tables, we need to know which bridges can source a given {G,C} pair.
- If endstations cannot transmit to a {G,C} address (except for IGMP, which is intercepted at the edge!), then only routers are sources.
- Otherwise, we must assume that every bridge that wants a {G,C} pair can also source it.

{S,G} state

- **(This brings up the idea that we should also advertise in the announcement packets the presence of routers on each VLAN. If a router is present on a port, then that port wants all multicasts delivered to it. Details are straightforward.)**

{S,G} state

- Let A and D be two MSTIDs such that A < D.
- Then bridge B's bridge variable holding MSTI D's Reflection Vector tells, in bit A, whether or not B is on the path from A to D, and also of course, from D to A.
- For every bridge A that can source {G,C} and every bridge D that wants {G,C} such that bridge B is along the path from A to D, bridge B requires an entry in its multicast forwarding database for {G, D_C+A_T}.
- That entry must specify to output the frames to every port that is a root port for MSTI D.

{S,G} state

- **The downside is that you have more entries in the MAC tables than before the spanning-tree-per-bridge was done.**
- **The upside is that multicast distribution is really, really, efficient in terms of bandwidth.**
- **Note that it is easy to detect that your bridge B is not on the path from source A to destination D, so no entry is required.**

{S,G} state

- **We may note that, if this method is not chosen, then we have to multiply the number of GMRP/MGRP registrations by the number of sources in the network, and each sink bridge has to know what the source bridges are.**
- **(The source information may be available, in which case it is obviously useful to the Reflection Vector method, as well.)**

GVRP state details

- Again, let X_C denote the Community Part of VLAN ID X, and X_T denote the MSTI ID to which VLAN ID X is assigned.
- Then, the GVRP vector bit X on port P is set to pass broadcasts and unicast floods if there exists any bridge D that wants Community X_C, port P is a Designated Port for MSTI X_T, and the reflection vector for MSTI X_T transmitted on that port (received, if $D < X_T$) has bridge D's bit set.

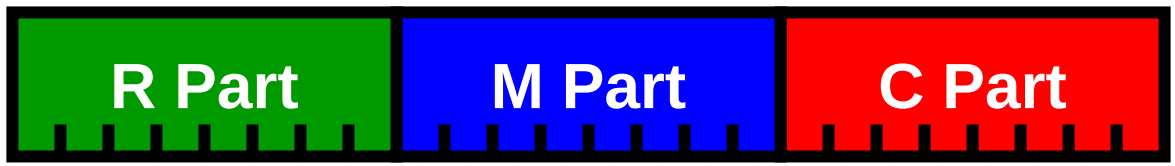
Announcement Packets

- **This plan eliminates the separate multicast distribution control protocol pass(es) that must run, after unicast convergence, in routed networks.**
- **This plan eliminates the similar multicast / VLAN pruning of GVRP, GMRP, or IGMP.**
- **The Announcement Packets contain information that does not change when the backbone topology changes.**
- **Only the *application* of this information to specific ports on each bridge changes when the topology changes.**

VLAN Tagging

Community-Multipath-Root (CMR) Tagging

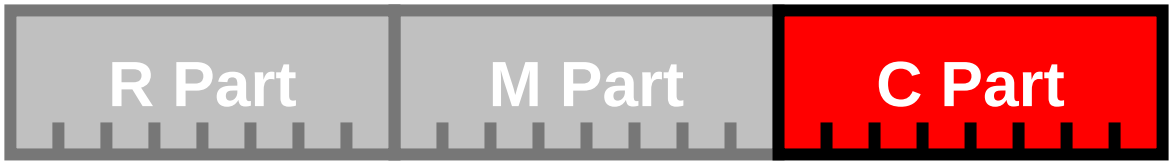
The VLAN tag, like Gaul, can be divided into three parts:



- **The Community Part specifies the broadcast domain (e.g. IP Subnet).**
- **The Multipath Part specifies which set of link cost parameters is used when routing the frame.**
- **The Root Part specifies which Root Bridge is used when routing the frame.**

Community-Multipath-Root (CMR) Tagging

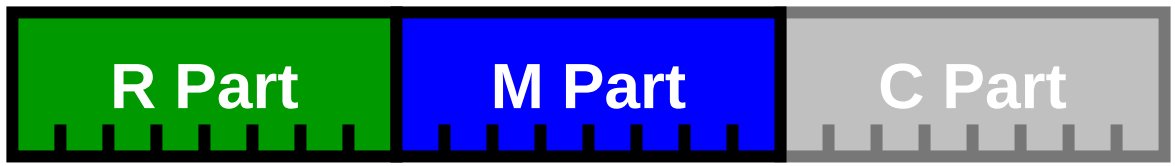
Community Part



- In IEEE Std. 802.1Q-1998, the original VLAN standard, the VLAN tag had only this single function; there was only one Spanning Tree Instance.
- The Community Part identifies who should see the frame; it has no place in the routing of a frame, except as demanded by HSRP/VRRP duplicated MAC addresses.

Community-Multipath-Root (CMR) Tagging

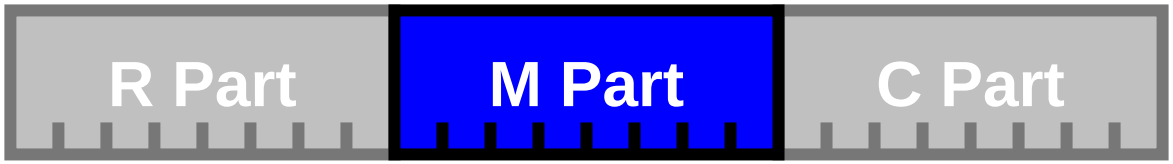
M Part and R Part



- The M Part and R Part, together, select the Spanning Tree Instance to be used for forwarding the frame.
- There are (number of M Parts) * (number of R Parts) separate Spanning Tree Instances.

Community-Multipath-Root (CMR) Tagging

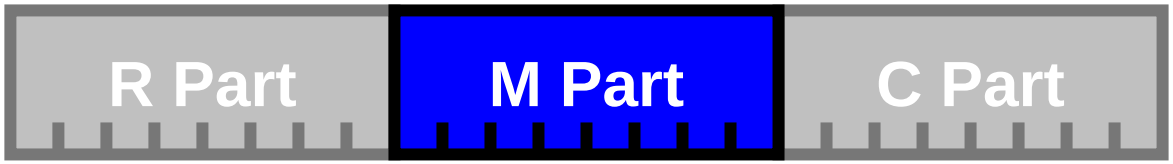
Multipath Part



- **Spanning trees, like OSPF or IS-IS, are constructed by minimizing the sum of the "costs" of the links over which a frame may pass.**
- **Every Spanning Tree Instance with the same Multipath Part must have the same link cost structure.**

Community-Multipath-Root (CMR) Tagging

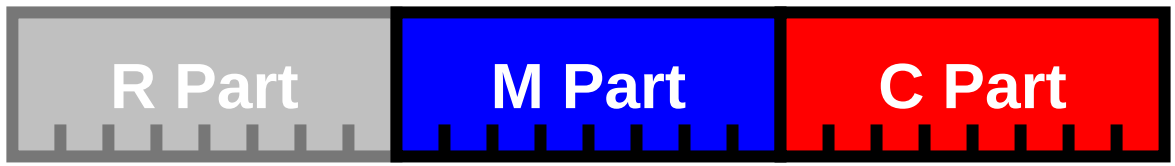
Multipath Part



- Spanning Tree Instances with different Multipath Parts may have different link Costs.
- This allows one to specify alternate paths across the network for different flows.
- The Multipath Part tags every frame with the equivalent of an EtherChannel hash value.

Community-Multipath-Root (CMR) Tagging

C Part and M Part



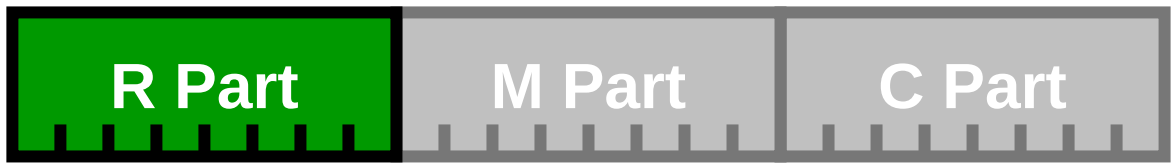
- **The Community Part and Multipath Part, together, determine the Filtering Data Base ID (FID) to use when looking up a MAC address.**

  The Root Part never affects FID selection.

  Multiple Community Part values may, but multiple Multipath Part values must not, map to the same FID. (Private VLANs map to same FID.)

Community-Multipath-Root (CMR) Tagging

Root Part

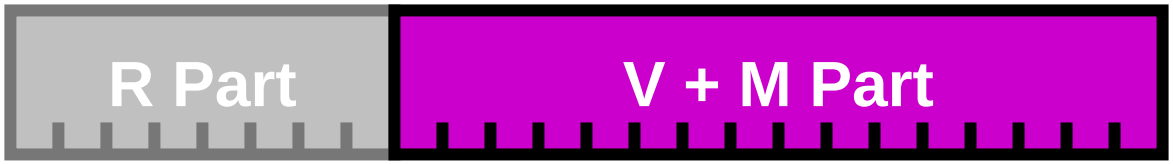


- The Root Part selects which spanning tree Root Bridge is to be used when routing this frame.
- By using the Root Part, perfect routing can be achieved for both unicast and multicast frames, thus avoiding the chief complaint against spanning trees.

Community-Multipath-Root (CMR) Tagging

Cisco.com

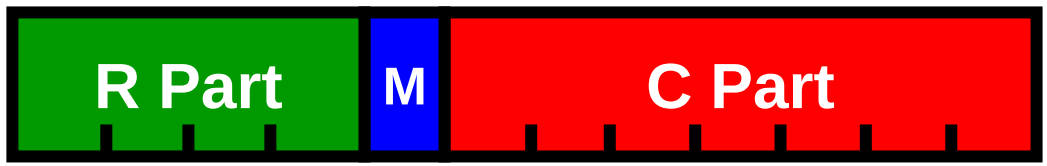
Mixed V+M Part



- **IEEE Std. 802.1s, Multiple Spanning Trees, combines the Community Part and the Multipath Part.**
- **This conflation of the Community Part and the Multipath Part is a common complaint against 802.1s MSTP: MSTP can only divide the multiple routing paths along subnet boundaries.**
- **On the other hand, when this method of selecting the engineered path to use is satisfactory, no bits need be dedicated to the M-Part**

Community-Multipath-Root (CMR) Tagging

Physical representation



- **If we divide up the 12 bits of the existing 802.1Q or new P802.1ad tag into three parts, we maintain compatibility with existing forwarding hardware, but we don’t have much room for the R Part.**

Community-Multipath-Root (CMR) Tagging

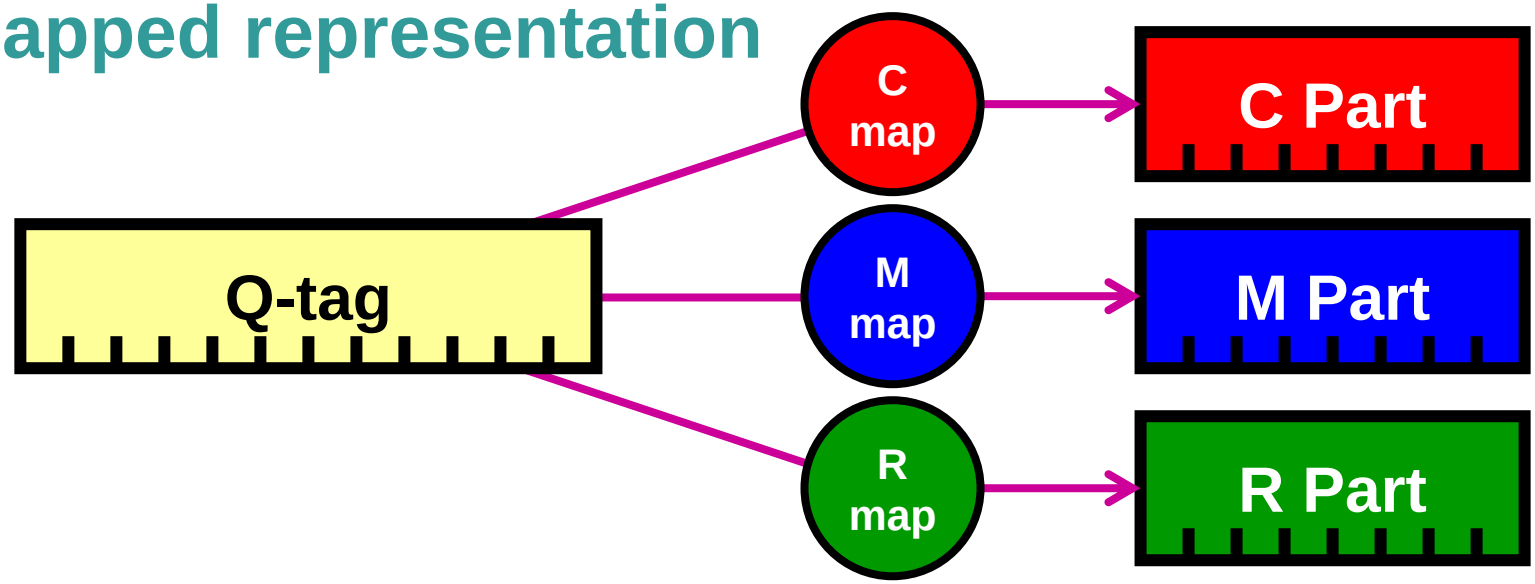
Cisco.com

Physical representation

- **Alternatively, we may place these parts in a brand new, much larger, VLAN tag.**
- **The emerging 802.1ah standard, which is very similar to our MTP tag, should handle this nicely.**

Community-Multipath-Root (CMR) Tagging

Mapped representation



- **Mapping allows more flexibility.**

  For example, if Bridge 6 does not and never will need Community 5, then that combination of R and C values need not exist.

  This can increase the size of the network at a proportionate cost in flexibility.

Data Plane Details

Data Plane: Unicasts

- **MAC address learning and data frame forwarding work exactly as for standard 802.1Q.**
- **Frames received in a Bridge that have no R-Part and M-Part (that specify on which MSTI the frame would be forwarded) are assigned to an MSTI (R-Part and M-Part) by that receiving Bridge.**

Data Plane: Unicasts

- The Root Part is constant for each Bridge.
- The Multipath Part may be constant, or may be a function of the contents of the frame, e.g. a hash of the IP address 5-tuple.
- If the M Part is chosen by a hash, that hash must be symmetrical with respect to source and destination.

Plug and Play

Plug and Play

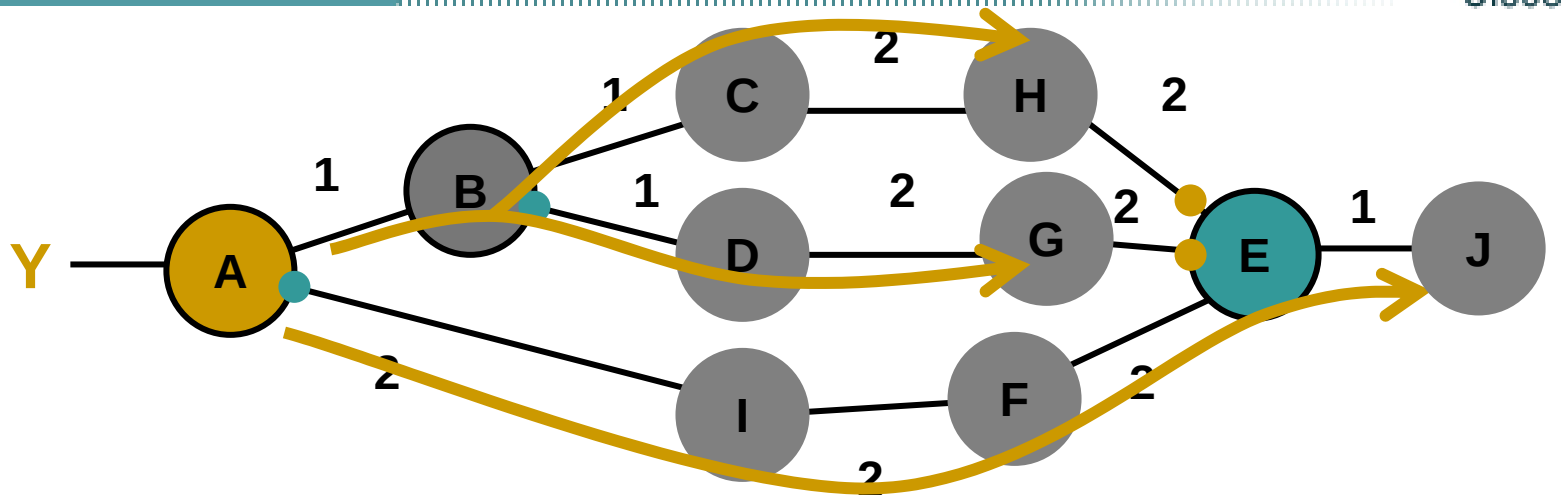
- **Since there are no configured (hierarchical) MAC addresses, there is little that must be configured.**

  Bridges must be assigned MSTIs, however.

  That takes configuration, and that configuration makes plug-and-play difficult.

Link State Protocols

Can we use link state protocols, instead?



- **YES**

Can we use link state protocols, instead?

- **No big deal: JUST ROUTE! Well, you could, ...**
- **Except that a Bridge cannot reliably know what stations are currently attached to it, in order to advertise their MAC addresses.**

  Because hubs, silent listeners, old PC stacks, and old bridges all exist.

- **Except that, since bridges never, ever, deliver multiple copies of multicasts, and essentially never deliver out-of-order frames, there exists (non-TCP) protocols that expect this behavior.**
- **Except that Ethernet frames have no TTL to suppress temporary loops during topology changes.**

Can we use link state protocols, instead?

- Of course, you could use heuristics (and tune them over time) to detect direct-attached stations,
- And you could replace all Customers' protocols that demand in-order, non-duplicated delivery,
- And you could add a TTL field to Ethernet frames ...
- (This is the TRILL approach.)

Can we use link state protocols, instead?

- In short, you could turn Bridges into really bad Routers, that is, Routes with an extremely sparse set of features.
- You could replace all existing bridges, since their hardware can't handle an Ethernet TTL.
- Then, you could spend the next 5 years adding all of the features of Routers into Bridges.

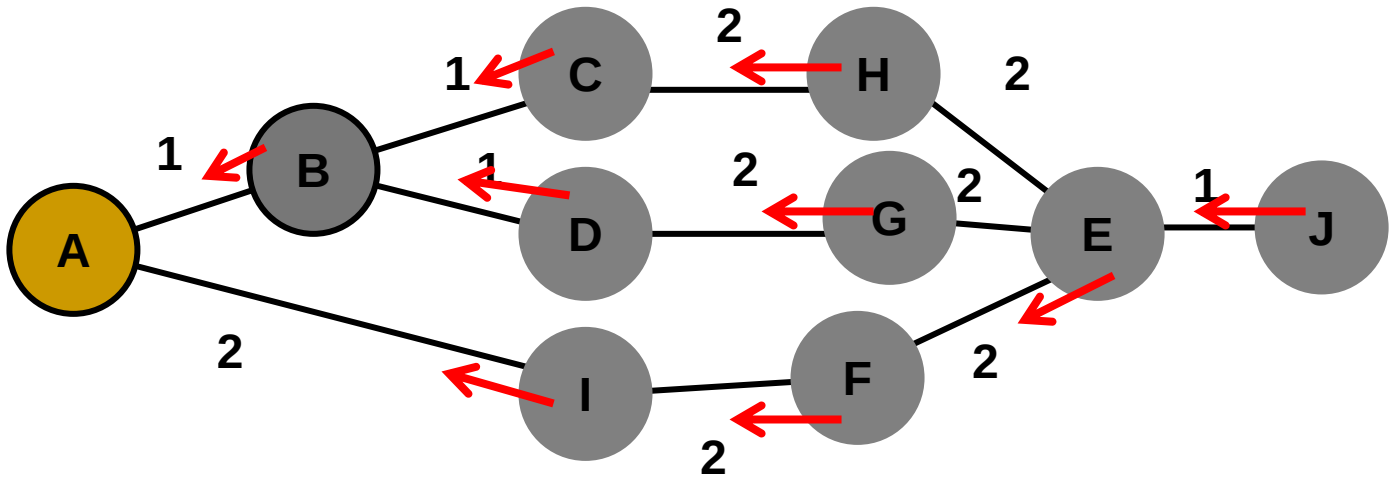
Can we use link state protocols, instead?

- But, why bother?
- If you want to route, no one is stopping you. Route!!

How to use Link State in Bridges

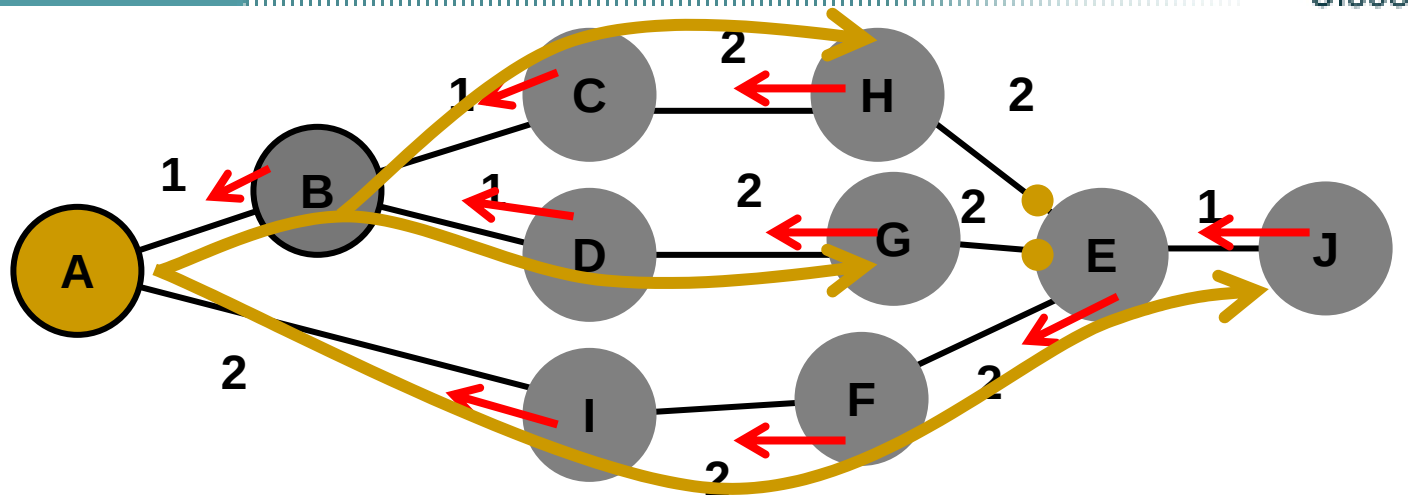
- **A much better plan is to update your Bridges' software to calculate the Optimal Bridging multiple spanning trees using Link State information, instead of MSTP.**
- **Then, you get the advantages of both Routing and Bridging.**
- **(Or, as John says, "I want the AND."**

How to use Link State in Bridges



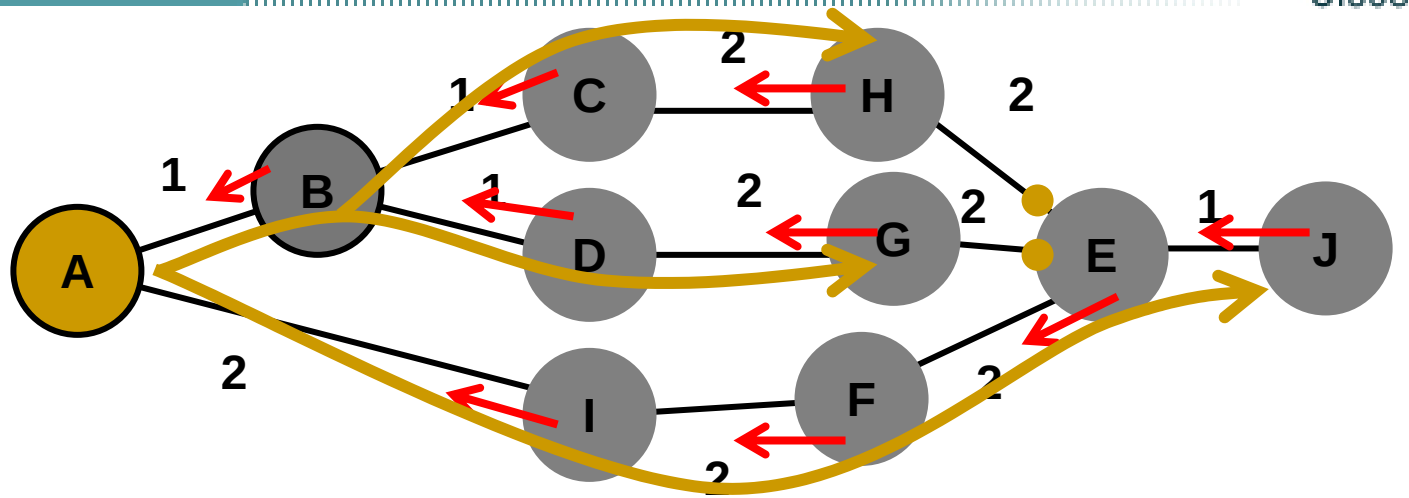
- **The summation of all of the routes to node A.**

How to use Link State in Bridges



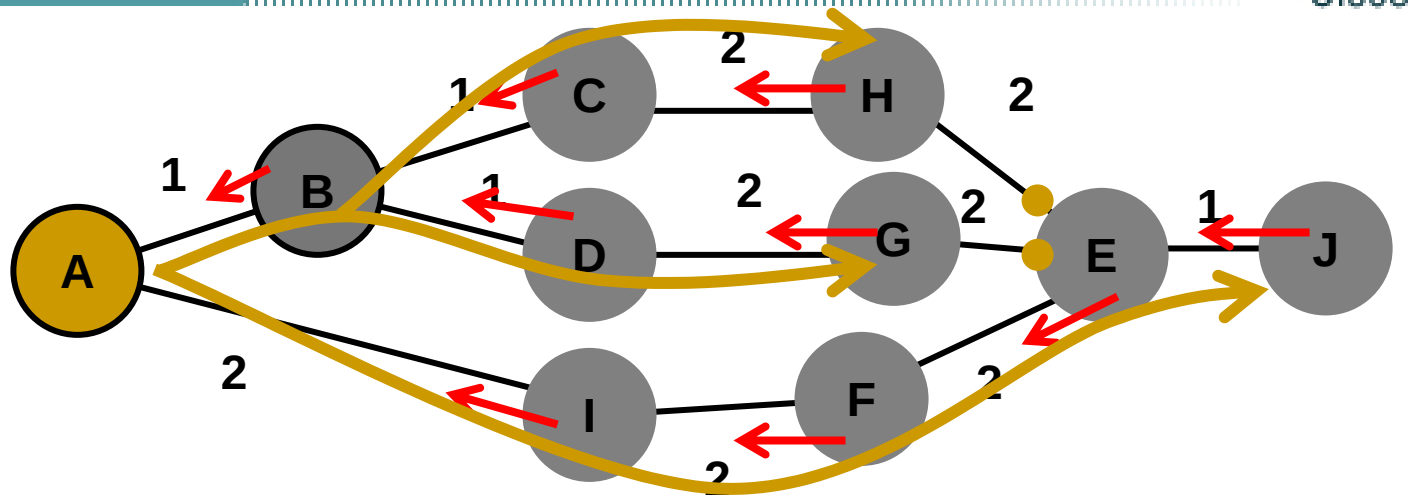
- **Is identical to the spanning tree rooted at node A.**
- **(That's how spanning trees are created.)**

How to use Link State in Bridges



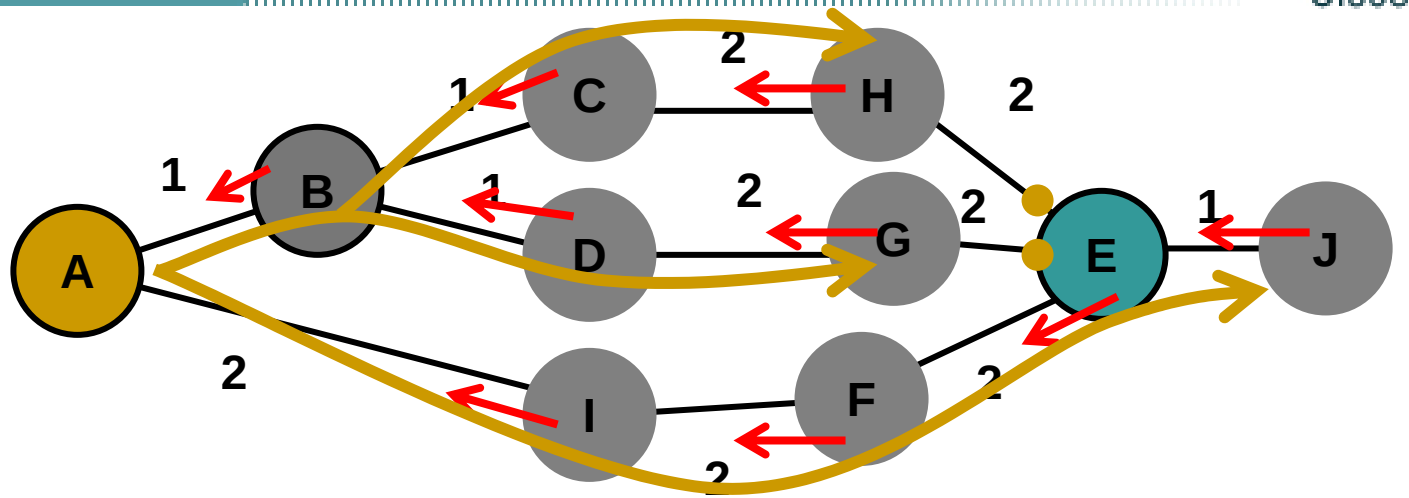
- **So, if Bridge A runs a Dijkstra calculation on the Link State information, it can compute the spanning tree rooted at A.**

How to use Link State in Bridges



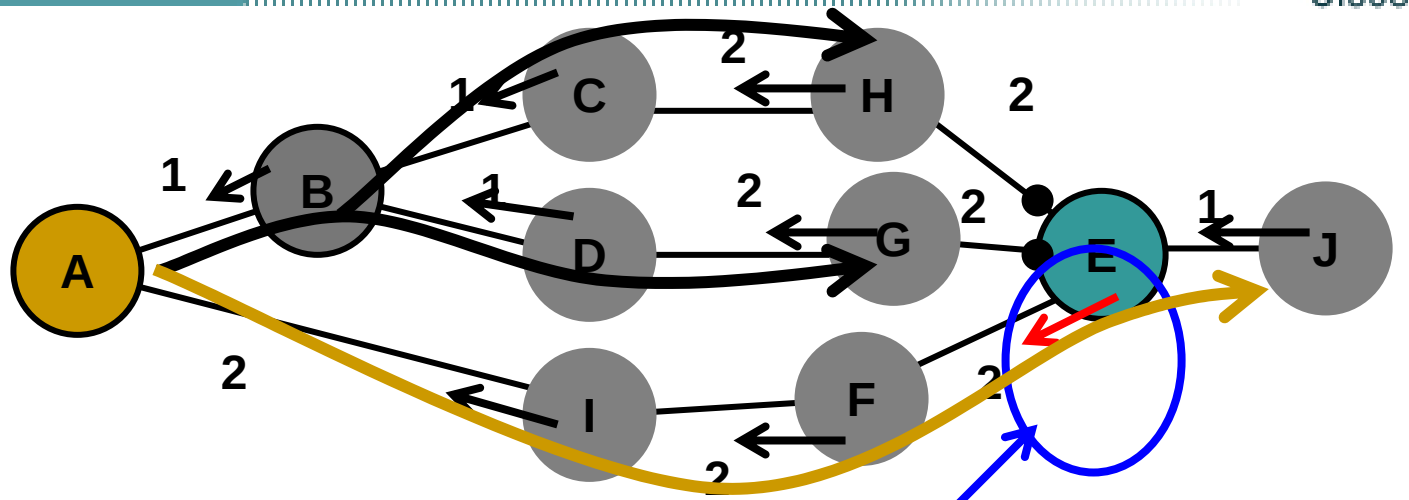
- **Unfortunately, that's irrelevant.**
- **Bridge A already knows it's the Root of its own tree.**
- **Bridge A needs to know its role in the other Bridges' spanning trees.**

Link State trick 1



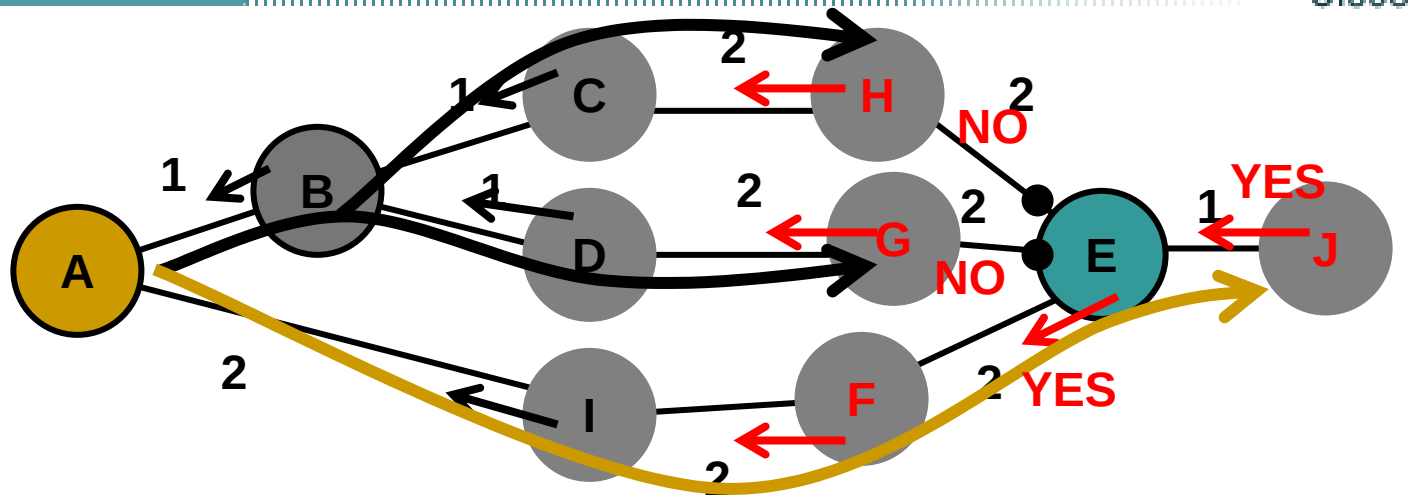
- **Make the Dijkstra calculation symmetrical.**
- **That is, tweak the algorithm slightly so that, if multiple equal-cost paths are found, all Bridges make the same decision.**
- **(This is really easy.)**

Link State trick 1



- **Then, Bridge E's (symmetrical) Dijkstra calculation of the path to Bridge A identifies the "Root Port" for the spanning tree rooted at Bridge A.**

Link State trick 2



- **Bridge E must also perform the Dijkstra calculation for each of its neighbor Bridges, in order to determine which of those links are part of Bridge A's spanning tree.**
- **Except for Bridge B's Root Port to Bridge A, those paths to A that point towards Bridge B are on Bridge A's spanning tree.**

Link State trick 2

- **But what about temporary loops during topology changes? Ethernet frames have no TTL field.**
- **You add interlocks between bridges, just like for MSTP, so that frames are discarded until the topology update is complete.**
- **Note that these interlocks are invoked only when necessary to prevent loops, and usually are not needed.**

Link State trick 3

- **For Multicast and VLAN distribution, you may either:**

  Bridge B may perform the Dijkstra calculation for every Bridge in the network, in order to determine whether it is on the path from A to C for every pair A and C; or

  You may use MVRP/MGRP, which are similar to IGMP, except that 4k groups or VLANs are signaled in a single packet; or

  You may pass the Reflection Vector information around the spanning trees computed by Link State information.

Summary

Summary

- **By modifying either MSTP or by using Link State methods, you may do perfect routing of Ethernet frames, while retaining all of the services offered today by Bridges, at levels of efficiency far exceeding that offered by any alternative technology.**

Summary: “Optimal Bridging” gives you

- **Perfect unicast routing.**
- **Perfect multicast routing.**
- **Preservation of current bridge hardware / knowledge investment.**
- **Very low computational load for control protocols.**
- **Convergence times after network disruptions, including multicast and VLAN pruning, that are better than current bridging convergence times without pruning.**
- **Almost plug and play: no configured addresses.**


CISCO SYSTEMS