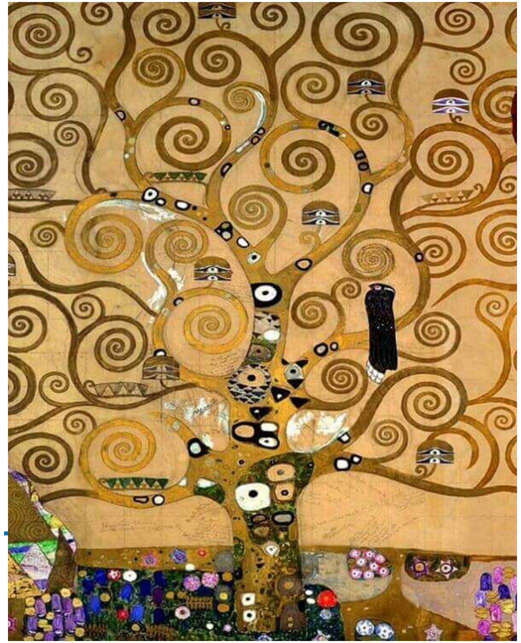


CS168: Intro to Internet Architecture

Flipped Lecture #2:

- Project 1 Intro + Tips
- What I Found Confusing About Routing
  - Split Horizon vs Poison Reverse vs Route Poisoning
  - Learning Switches
  - STP

cs168.io

Alex Krentsel, Narek Galstyan, and Sarah McClure

Berkeley
UNIVERSITY OF CALIFORNIA

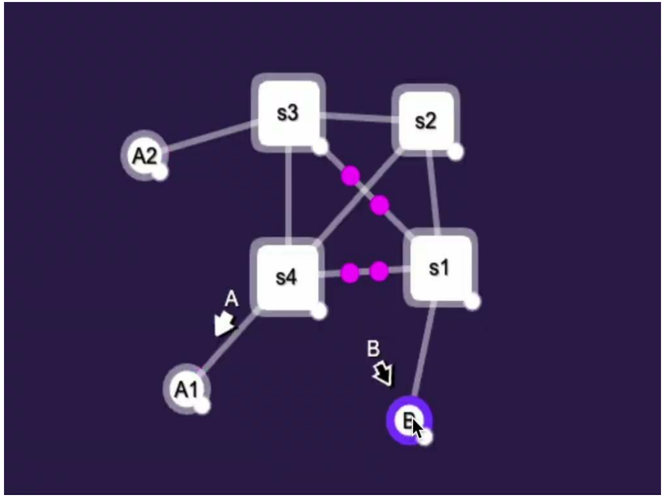
Announcements

- Regular in-person lectures starting from this week - no more pre-recorded videos to watch at home.
- Project 1 is released **today at 12:30pm**, find more info on Ed.

Project 1: Routing

You will implement the Distance Vector protocol!

- Use a network simulator (by Murphy McCauley & others at NetSys) to test, validate, and interact with your router implementation.
- The project is split and scored by 10 stages.
- Goal: guide you through the basics–packet forwarding, route advertisement–to the advanced features, e.g., split horizon, poison reverse, route poisoning, and real-world optimizations.



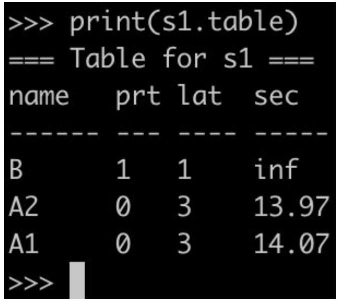
Due: 11:59pm, Oct. 7th. Logistics & OHs will be announced on Ed.

Project TAs:

- Silvery Fu (lead)
- Tenzin Ukyab
- Ken Lien

Project 1 Tips

- Comment your code for each stage - subsequent stages require editing the past code you’ve written.
- Remember that your latency to a dst is your neighbor’s latency to the dst + *your latency to your neighbor*.
- Make sure you understand each scenario across the 10 stages.
  - This project is as much a thinking exercise as a coding one.
  - Use `self.s_log()` liberally
    - May be worth logging when you make changes to a routing table. `setter` methods are very good for doing this :)
- The terminal is a live Python interpreter, use it to inspect state, with `print(node.table)`



```
>>> print(s1.table)
=== Table for s1 ===
name   prt lat  sec
------ --- ---- -----
B      1   1    inf
A2     0   3    13.97
A1     0   3    14.07
>>> 
```

What I Found Confusing about Routing

(which also happen to be core concepts for the project)

Another Perspective on Distance-Vector Routing Concepts

Every semester, students get confused between Split Horizon, Route Poisoning, and Poison Reverse.

Commons questions:

- What combinations can I use?
- Which one is the best?
- Route Poisoning vs Poison Reverse?
- Split Horizon vs Poison Reverse?

Let's clear these up…

Routing Motivation

Core problem behind routing:

- Building the rules used for forwarding packets such that network requirements are satisfied.

Reasonable requirements:

1. Full connectivity
2. Minimized latency (or cost)

Aside: Routing happens in the **control plane**. Forwarding happens in the **data plane**.

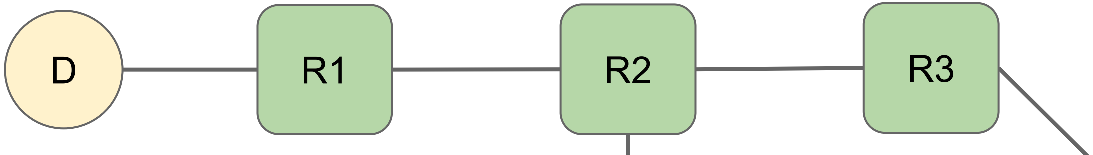
Routing

We’ve discussed Link State (LS) and Distance Vector (DV) protocols as two approaches.

- LS conceptually easier, and not on this project, so let’s focus on DV.

Distance Vector:

- Each node tells its neighbors how far it is from every destination.
- Each node uses the neighbor closest to each destination as its next hop for that dest.



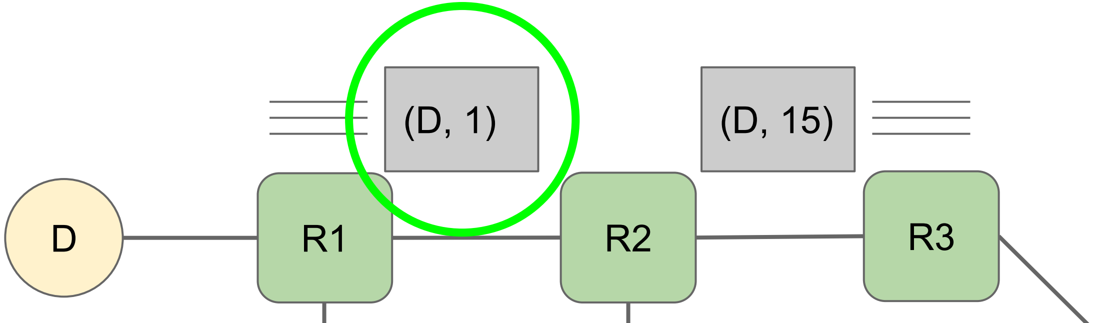
Routing

We've discussed Link State (LS) and Distance Vector (DV) protocols as two approaches.

- LS conceptually easier, and not on this project, so let's focus on DV.

Distance Vector:

- *Speaker*: Each node tells its neighbors how far it is from every destination.
- *Listener*: Each node uses the neighbor closest to each destination as its next hop for that dest.

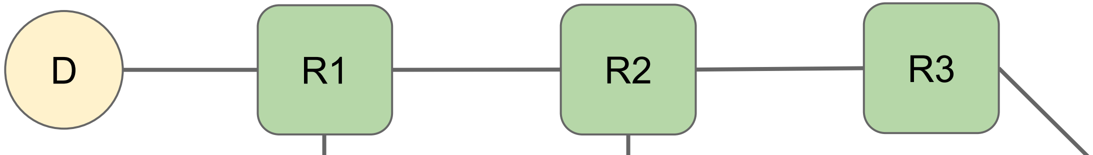


Problems

Problems in the network are caused by invalid network state…

What problems could we run into leading to invalid network state?

- (1) Missing entries
- (2) Loops
  - R1->R2->R1->R2->...->dropped
- (3) Stale network info (entry in a forwarding table that no longer exists in reality)



(1) Missing Entries

What leads to missing entries?

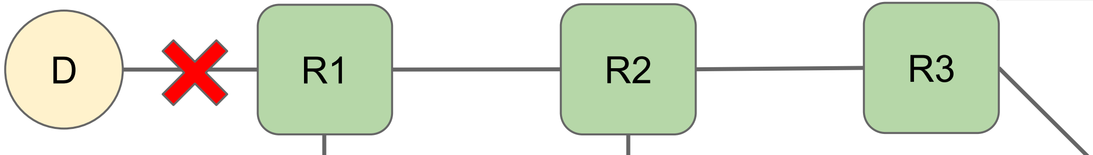
- Advertisement gets dropped

  Retransmit advertisements every X seconds

- Destination is unreachable in the topology

  ??? - only fix is to get that destination reconnected

Destination	Next Hop, Cost
???	???



Problems

What problems could we run into leading to invalid network state?

- (1) Missing entries ✓ **Retransmit Advertisements**
- (2) Loops
  - R1->R2->R1->R2->...->dropped
- (3) Stale network info (entry in a forwarding table that no longer exists in reality)

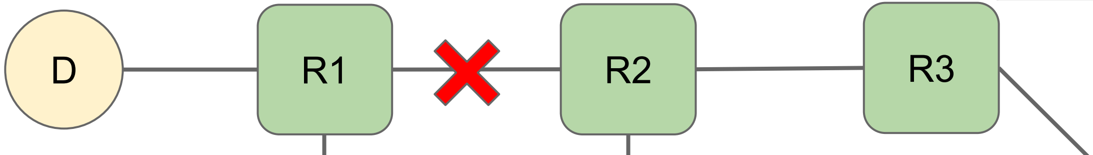
(2) Loops

What leads to (direct) loops?

- 2 routers must have have "accepted" routes through each other!

Destination	Next Hop, Cost
D	R1, 2

Destination	Next Hop, Cost
D	R2, 3



(2) Loops

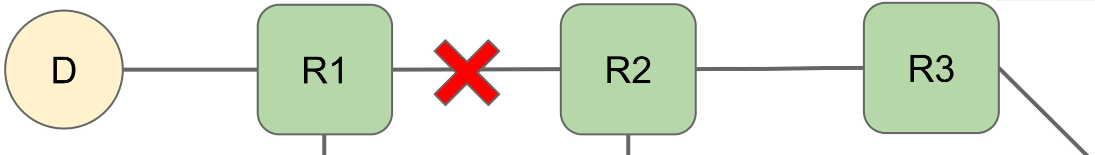
What leads to (direct) loops?

- 2 routers must have have “accepted” routes through each other!

At some point, R2’s route TTL expires…

Destination	Next Hop, Cost
D	

Destination	Next Hop, Cost
D	R2, 3

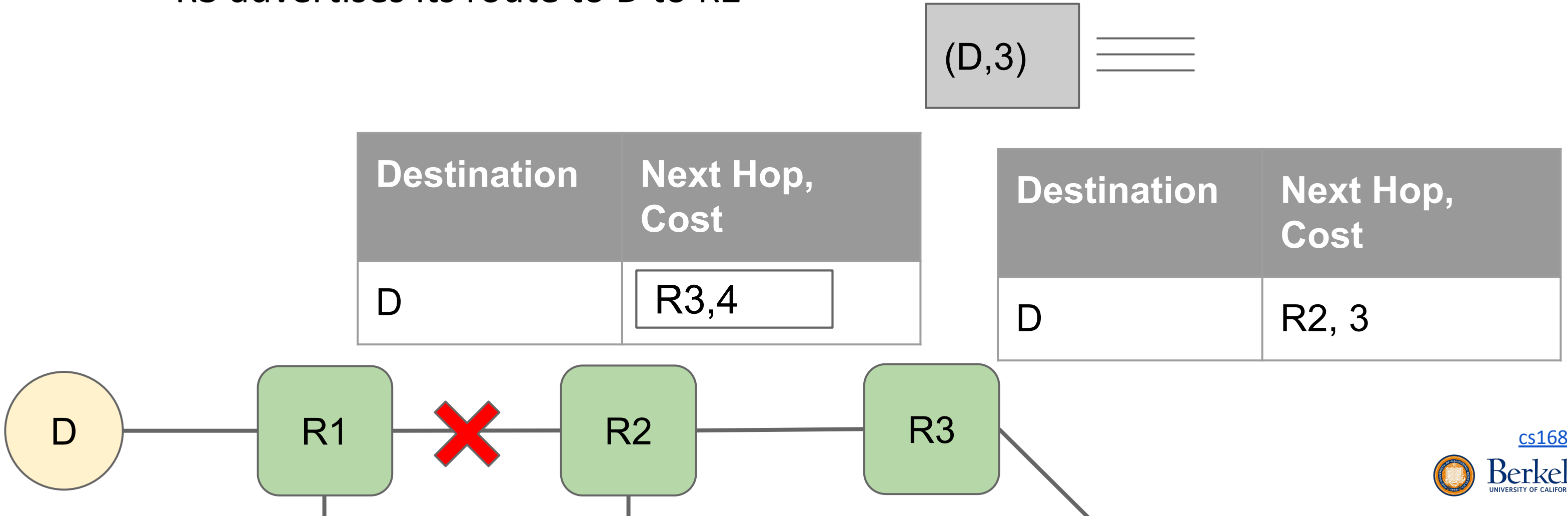


(2) Loops

What leads to (direct) loops?

- 2 routers must have have “accepted” routes through each other!

R3 advertises its route to D to R2



Destination	Next Hop, Cost
D	R3,4

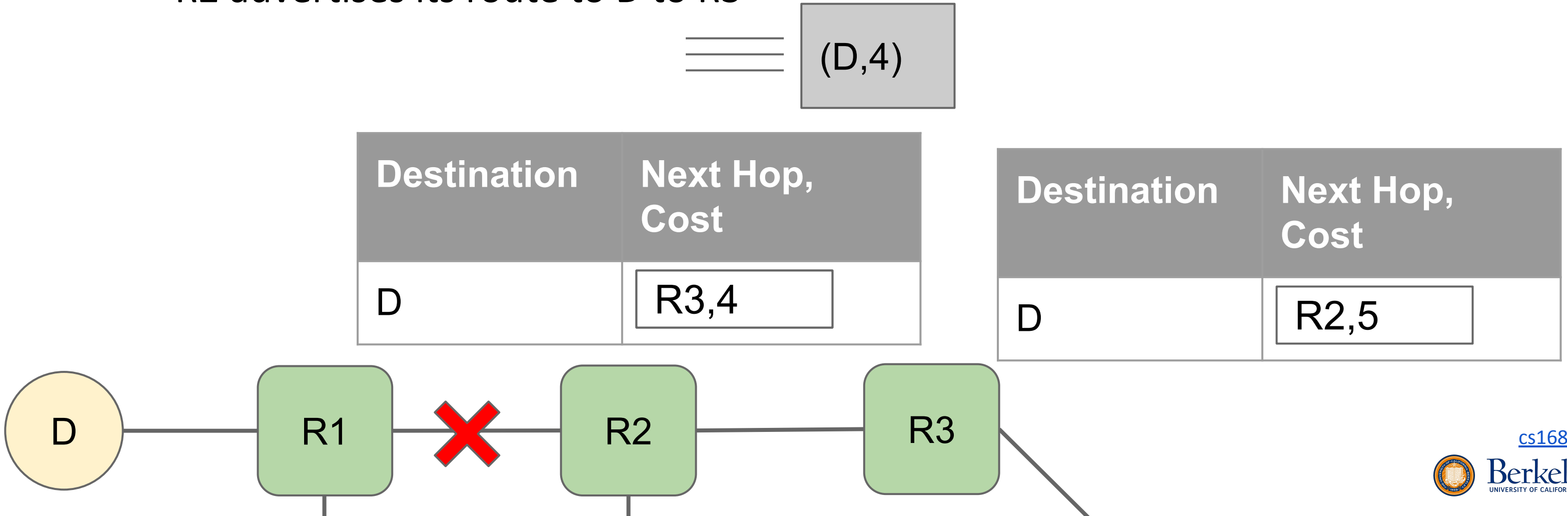
Destination	Next Hop, Cost
D	R2, 3

(2) Loops

What leads to (direct) loops?

- 2 routers must have have “accepted” routes through each other!

R2 advertises its route to D to R3



(D,4)

Destination	Next Hop, Cost
D	R3,4

Destination	Next Hop, Cost
D	R2,5

(2) Loops

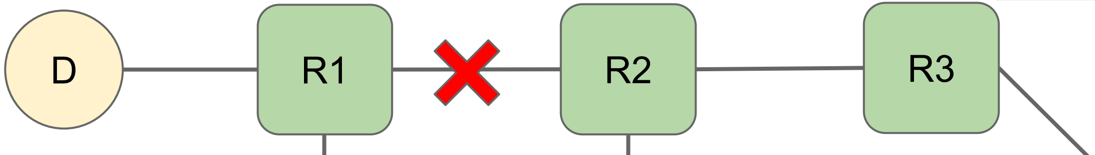
What leads to (direct) loops?

- 2 routers must have have “accepted” routes through each other!

This looping continues forever, counting up to infinity.
This direct looping is what Split Horizon and Poison Reverse attempt to solve.

Destination	Next Hop, Cost
D	R3,8

Destination	Next Hop, Cost
D	R2,7

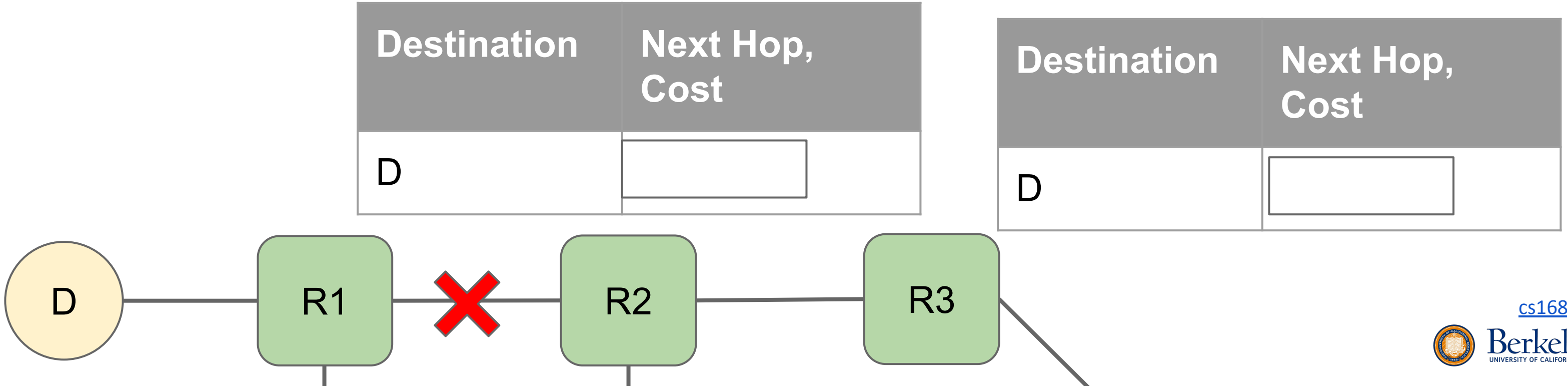


Split Horizon

Split Horizon says:

- If I route through you, I will *not* tell you about my path.

Can’t have a direct loop form if one side never tells the other about the route, right?



Destination	Next Hop, Cost
D	

Destination	Next Hop, Cost
D	

Split Horizon

Split Horizon says:

- If I route through you, I will *not* tell you about my path.

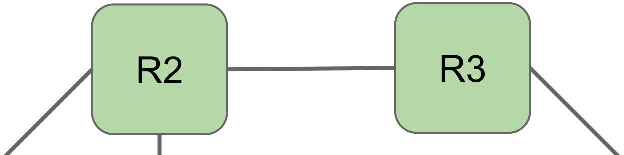
Can’t have a direct loop form if one side never tells the other about the route, right?

- Sort of... There are still some tricky edge cases that could lead to a loop forming - you’ll see one in the project.

If we *do* get into a direct loop, Split Horizon breaks out by letting the Time to Live on the updates expire - loop exists for that long!

Destination	Next Hop, Cost
D	R3, 2

Destination	Next Hop, Cost
D	R2, 3

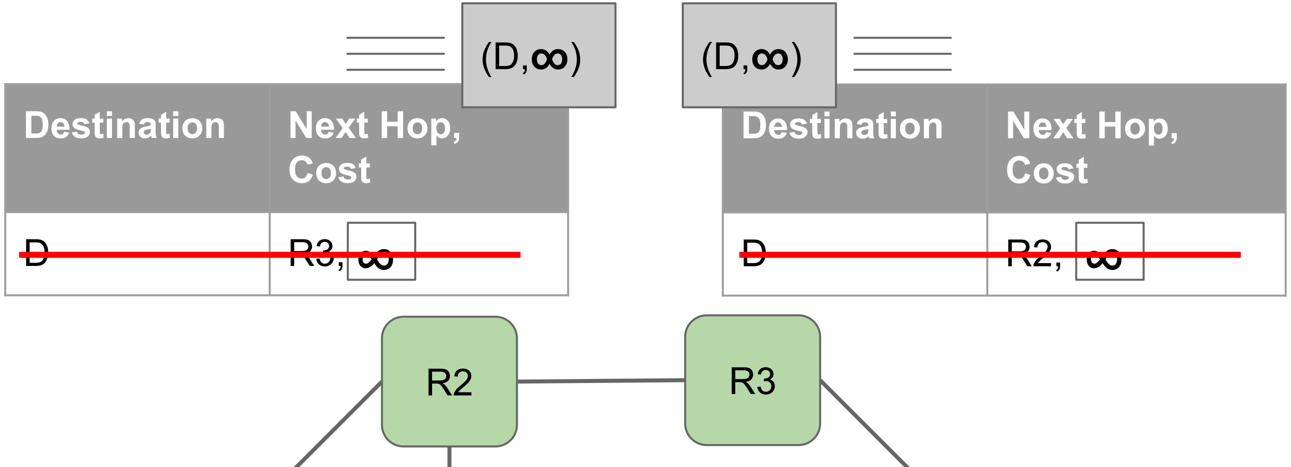


Poison Reverse

Poison Reverse is an improvement on Split Horizon:

- If I route through you, I will lie and tell you *my cost to the dest is* **∞.**
  - All routers agree not to use paths that have cost infinity.

Equally good at preventing direct loops, but *guarantees that in case of a direct loop, forwarding entries are removed at next advertisement, rather than waiting for the next timeout.*



Destination	Next Hop, Cost
~~D~~	~~R3, ∞~~

Destination	Next Hop, Cost
~~D~~	~~R2, ∞~~

A Naming Aside…

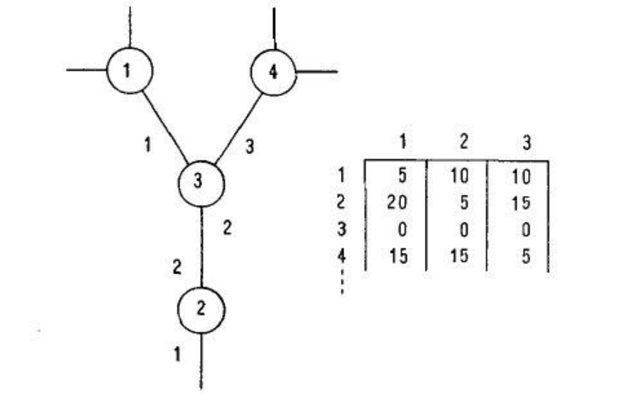
Why is it called "split horizon"?

- Guesses?

From the original paper introducing the concept in 1975...

"The forming of [an update] may be looked upon as describing the **horizon** *seen from that node*…"

- "Omitting [information to particular neighbor] will be called split horizon" (neighbor sees diff. "split")
- "If the same [update] will be transmitted to all the neighbors, the procedure will be referred to as whole horizon."



	1	2	3
1	5	10	10
2	20	5	15
3	0	0	0
4	15	15	5
⋮			

Fig. 7. Part of a network and a delay table in node 3.

Problems

What problems could we run into leading to invalid network state?

- (1) Missing entries ✓ **Retransmit Advertisements**
- (2) Loops ✓ **Split Horizon or Poison Reverse (for direct loops, indirect loops still possible)**
  - R1->R2->R1->R2->...->dropped
- (3) Stale network info (entry in a forwarding table that no longer exists in reality)

(3) Stale Network Info

What leads to stale network info?

- Any change in the network state, while we wait for it to propagate!

Different classes of information take different amounts of time to propagate:

- **Additive Information**: propagates as quickly as the "advertisement frequency"
  - A new router comes up, new link gets added, new host attaches
- **Subtractive Information**: as quickly as the TTL of entries stored in the table
  - A router crashes, a link goes down, hosts detach

TTL >> "advertisement frequency", so how can we improve propagation time for subtractive information…

Route Poisoning

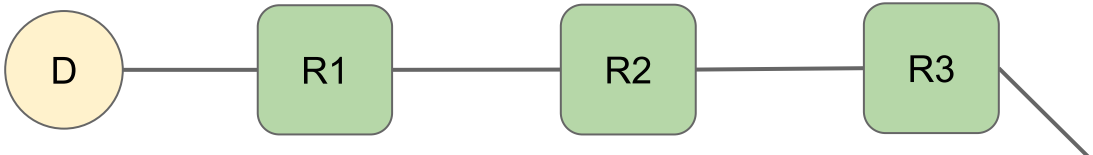
Key Observation we saw with Poison Reverse - you can override a neighbor's TTL period by sending an advertisement with infinite cost. Can we do the same thing when something goes down to reconverge more quickly?

Rule: When you lose a route, instead of just deleting it from your forwarding table, tell all your neighbors your route now has cost **∞**.

Dest	Next Hop, Cost
D	direct, 1

Dest	Next Hop, Cost
D	R1, 2

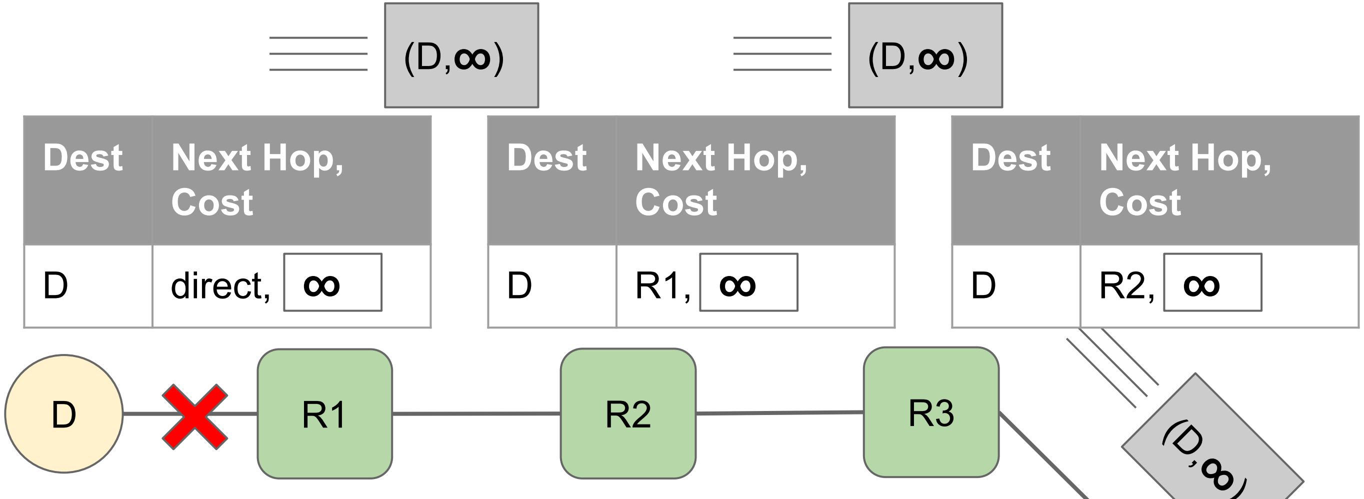
Dest	Next Hop, Cost
D	R2, 3



Route Poisoning

Key Observation we saw with Poison Reverse - you can override a neighbor’s TTL period by sending an advertisement with infinite cost. Can we do the same thing when something goes down to reconverge more quickly?

Rule: When you lose a route, instead of just deleting it from your forwarding table, tell all your neighbors your route now has cost **∞**.



Dest	Next Hop, Cost
D	direct, ∞

Dest	Next Hop, Cost
D	R1, ∞

Dest	Next Hop, Cost
D	R2, ∞

Problems

What problems could we run into leading to invalid network state?

- (1) Missing entries ✓ **Retransmit Advertisements**
- (2) Loops ✓ **Split Horizon or Poison Reverse (for direct loops, indirect loops still possible)**
  - R1->R2->R1->R2->...->dropped
- (3) Stale network info (entry in a forwarding table that no longer exists in reality) ✓ **Route Poisoning**

Frequently Asked Questions

With all of this in mind, back to the common questions:

- What combinations can I use?
- Which one is the best?
- Route Poisoning vs Poison Reverse?

What Combinations Can I Use?

Split horizon and Poison Reverse can’t be used at the same time - one says to say nothing, when the other says to say cost of infinity…

Pick either 1 or 0 for each category of problem:

- Loop Prevention: Split Horizon, Poison Reverse
- Remove Stale Information: Route Poisoning

So to enumerate all of the possibilities:

1. None
2. Poison Reverse (for Loop Prevention), Nothing (for Removing Stale Info)
3. Split Horizon (for Loop Prevention), Nothing (for Removing Stale Info)
4. Nothing (for Loop Prevention), Route Poisoning (for Removing Stale Info)
5. Poison Reverse (for Loop Prevention), Route Poisoning (for Removing Stale Info)
6. Split Horizon (for Loop Prevention), Route Poisoning (for Removing Stale Info)

Frequently Asked Questions

- What combinations can I use?

{Split Horizon, Poison Reverse, Nothing} x {Route Poisoning, Nothing}

- Which one is the best?

- Route Poisoning vs Poison Reverse?

Which One Is Best?

Slightly incorrect assumption in this question…you can't compare all 3.

- Split Horizon and Poison Reverse address one problem
- Route Poisoning address a different, orthogonal problem

Better question: which combination is the best?

- Better to use some strategy for each problem than no strategy
  - Remove Stale Info: *Route Poisoning*
  - Loop Prevention: *Poison Reverse*
    - Split Horizon and Poison Reverse are equally good at preventing loops, but Poison Reverse will kill loops faster than Split Horizon when they do arise.

Only caveat: Poison Reverse sends a bit more control traffic than split horizon, but it's a negligible difference.

Frequently Asked Questions

- What combinations can I use?

{Split Horizon, Poison Reverse, Nothing} x {Route Poisoning, Nothing}

- Which one is the best?

Poison Reverse + Route Poisoning is the best combo.

- Route Poisoning vs Poison Reverse?

Route Poisoning vs Poison Reverse

	Poison Reverse	**Route Poisoning**
"Trigger"	Receiving an advertisement that causes you to go through a new next hop to some destination.	Losing a route to a destination
Problem Being Addressed	Loop prevention, and getting out of loops quickly.	Removing stale routing information.
Mechanism	Send *only the neighbor you will use as next hop* an update saying your distance to D is ∞.	Send *all neighbors* an update saying your distance to D is ∞.

The extent of the *similarity* is that they both use the mechanic of sending an advertisement with a cost of infinity to someone. But for different purposes, and triggered by different reasons!

Frequently Asked Questions

- What combinations can I use?

{Split Horizon, Poison Reverse, Nothing} x {Route Poisoning, Nothing}

- Which one is the best?

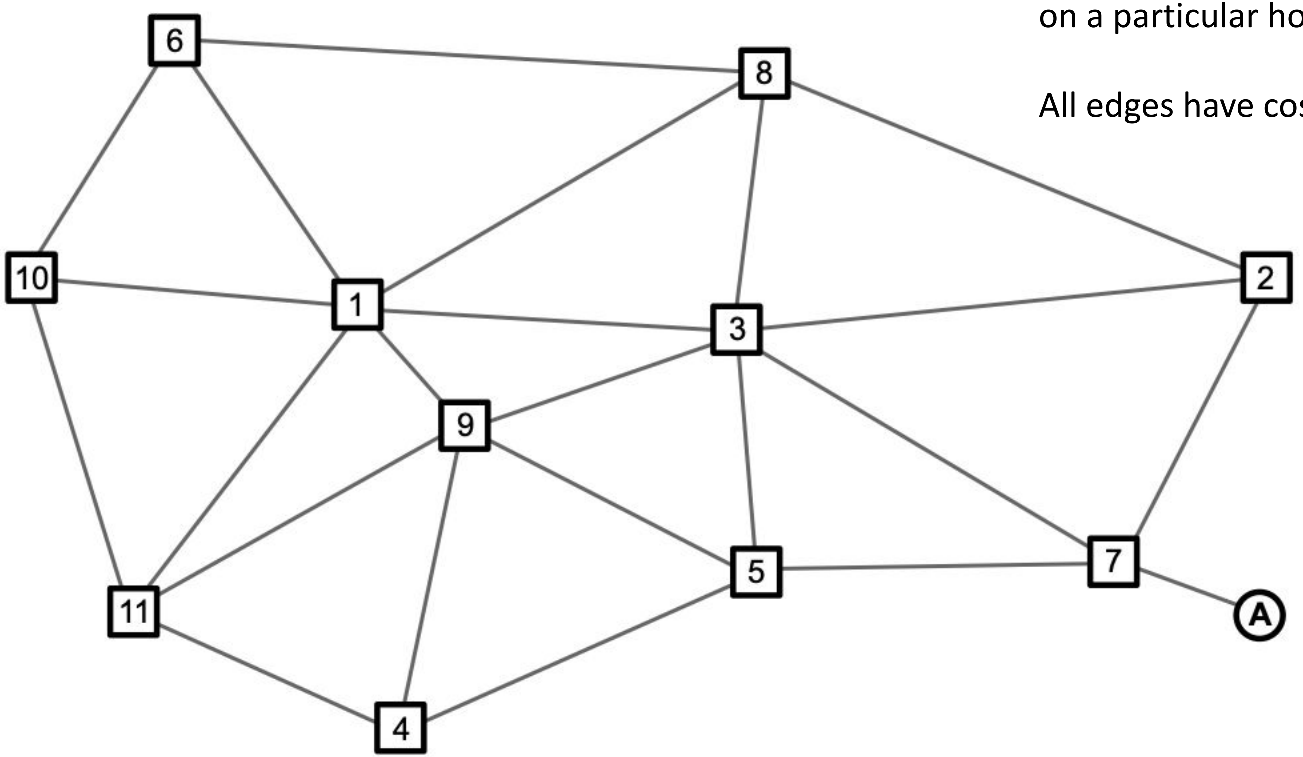
Poison Reverse + Route Poisoning is the best combo.

- Route Poisoning vs Poison Reverse?

Address different problems (though with a similar mechanism - send ∞)

Split Horizon vs Poison Reverse

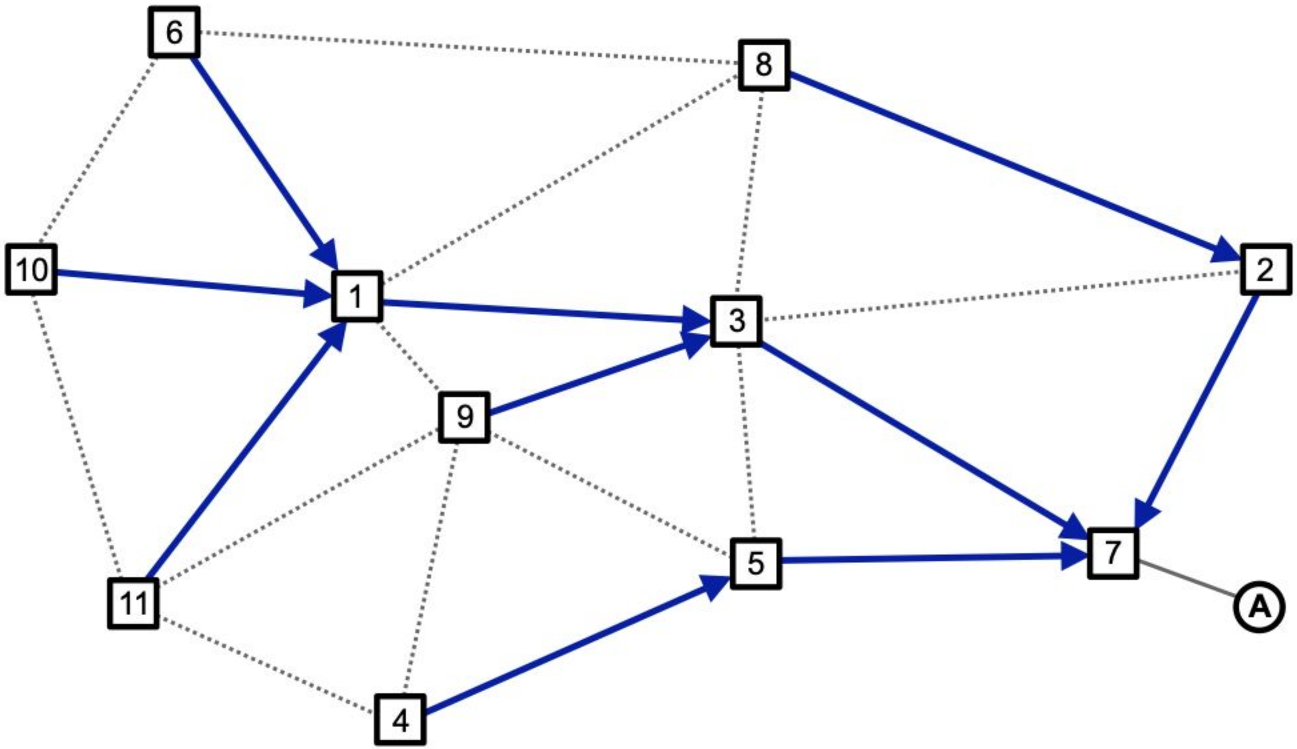
Split Horizon and Poison Reverse



Let's take a network and focus on a particular host as the dst.

All edges have cost 1.

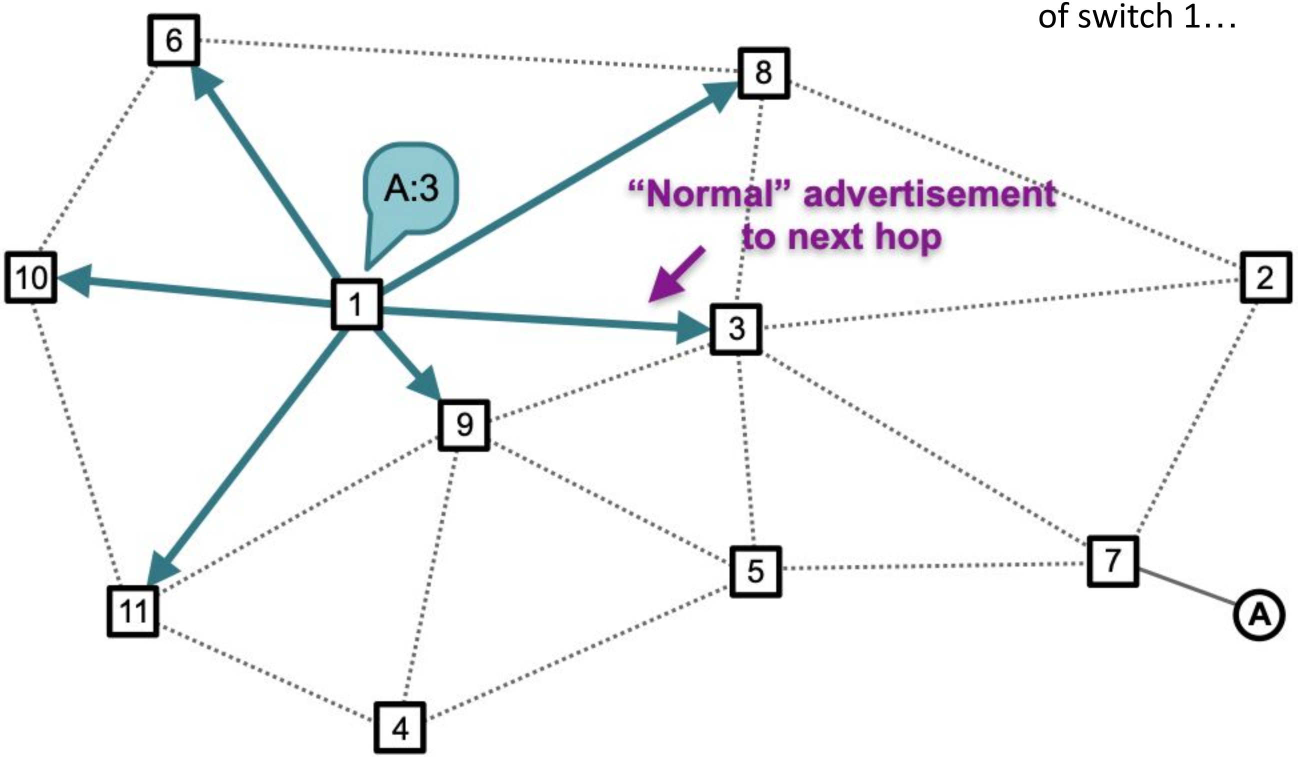
Split Horizon and Poison Reverse



Best paths to A.

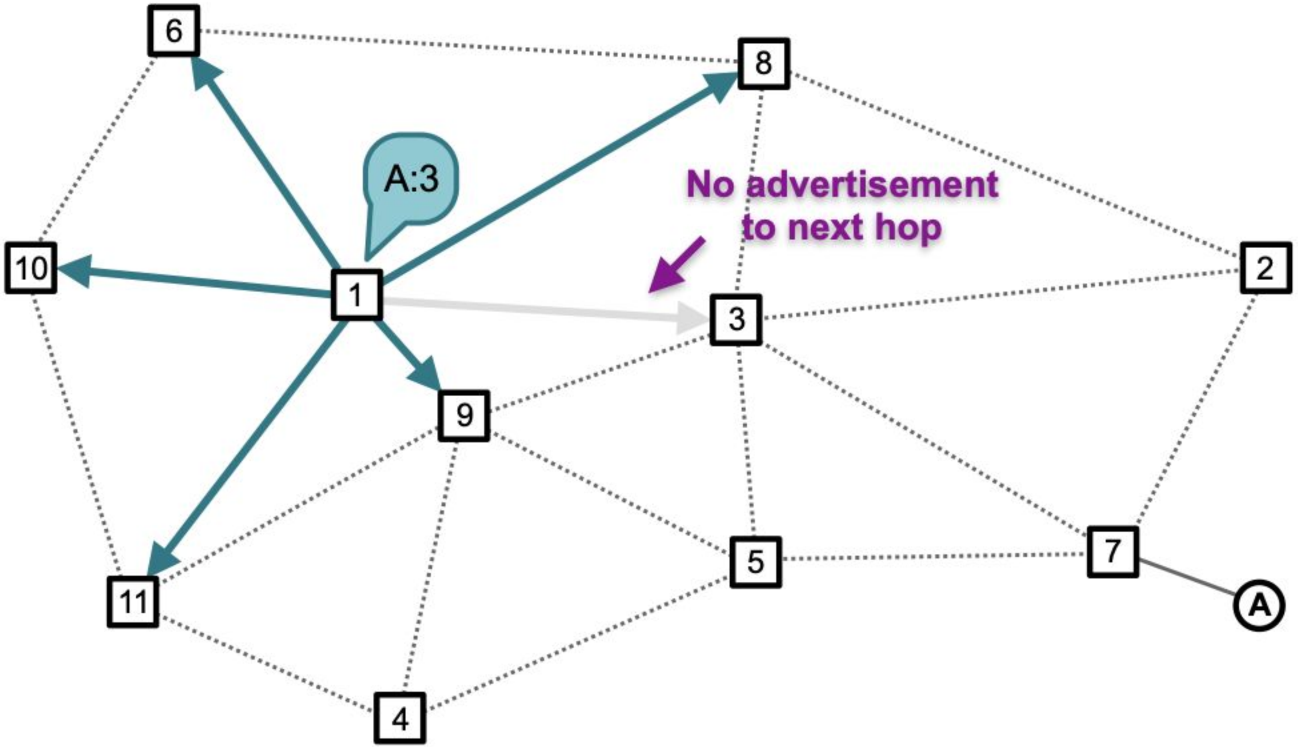
Split Horizon and Poison Reverse

Let's focus on the perspective of switch 1…



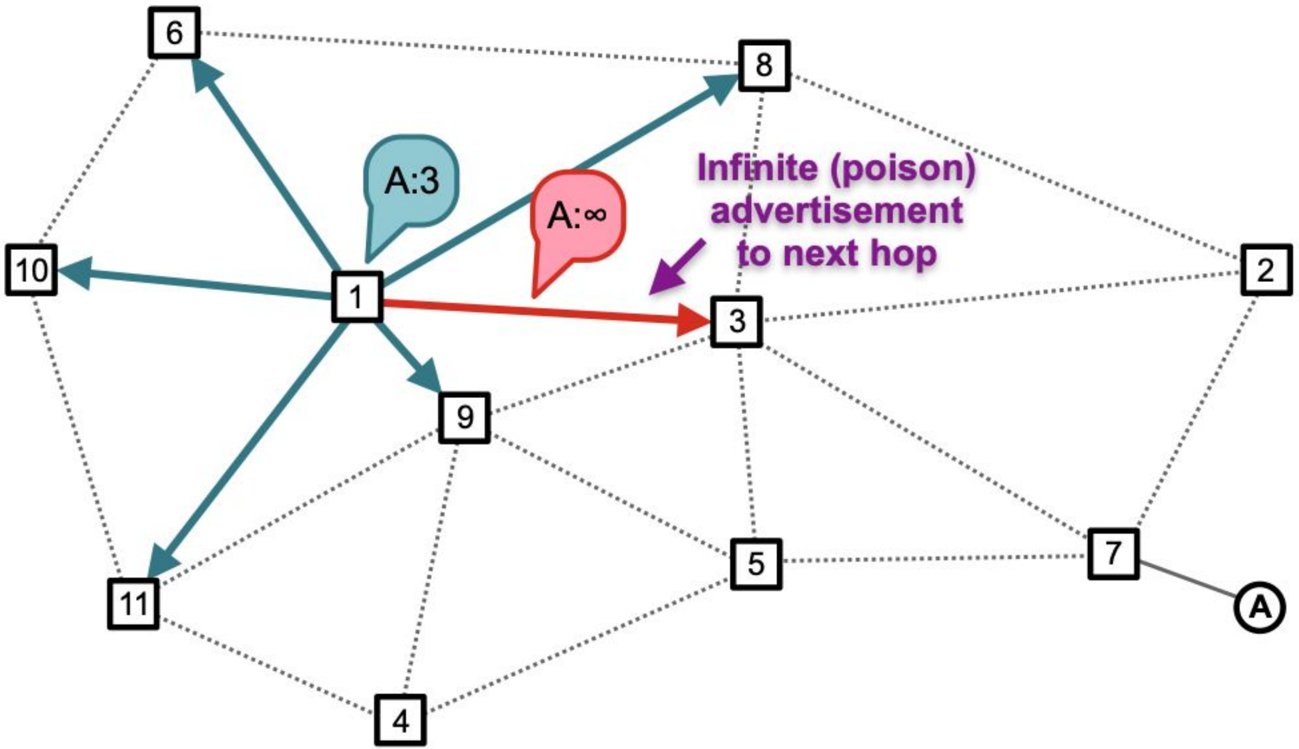
Advertisements from switch 1 with *no split horizon or poison reverse*

Split Horizon and Poison Reverse



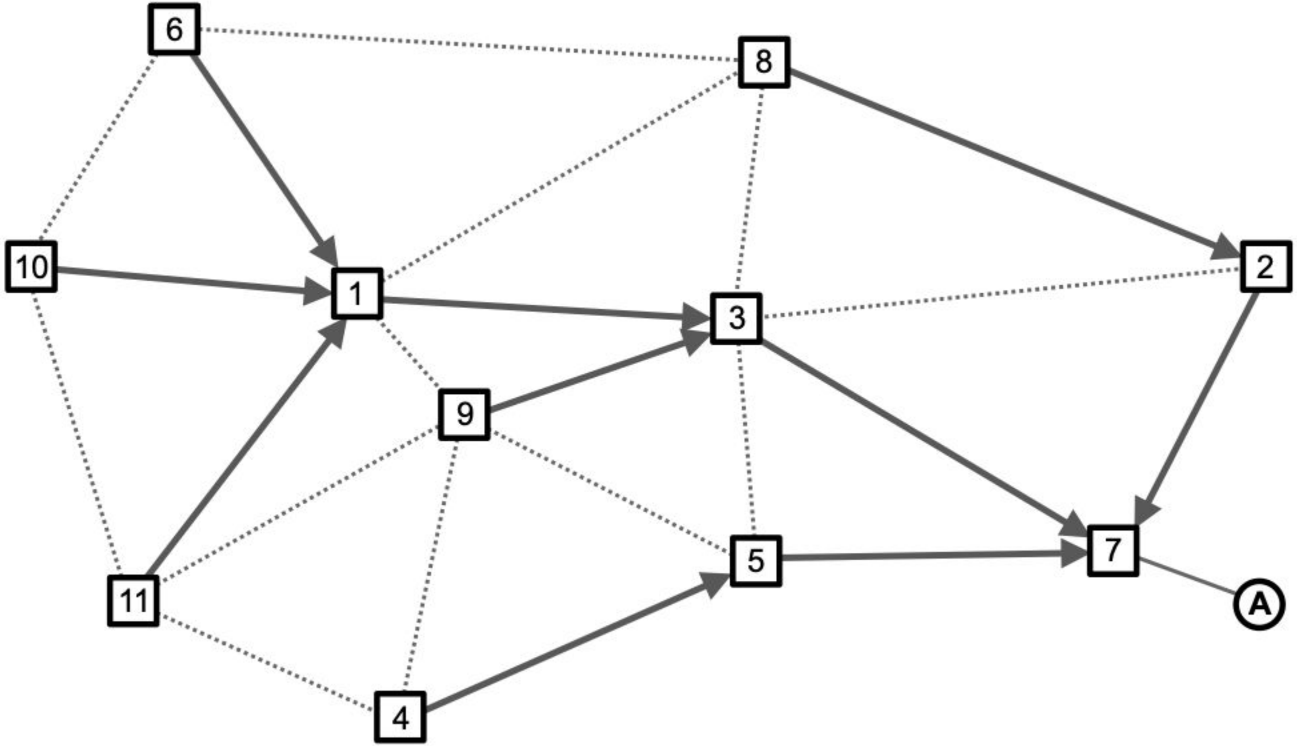
Advertisements from switch 1 with **split horizon**

Split Horizon and Poison Reverse



Advertisements from switch 1 with **poison reverse**

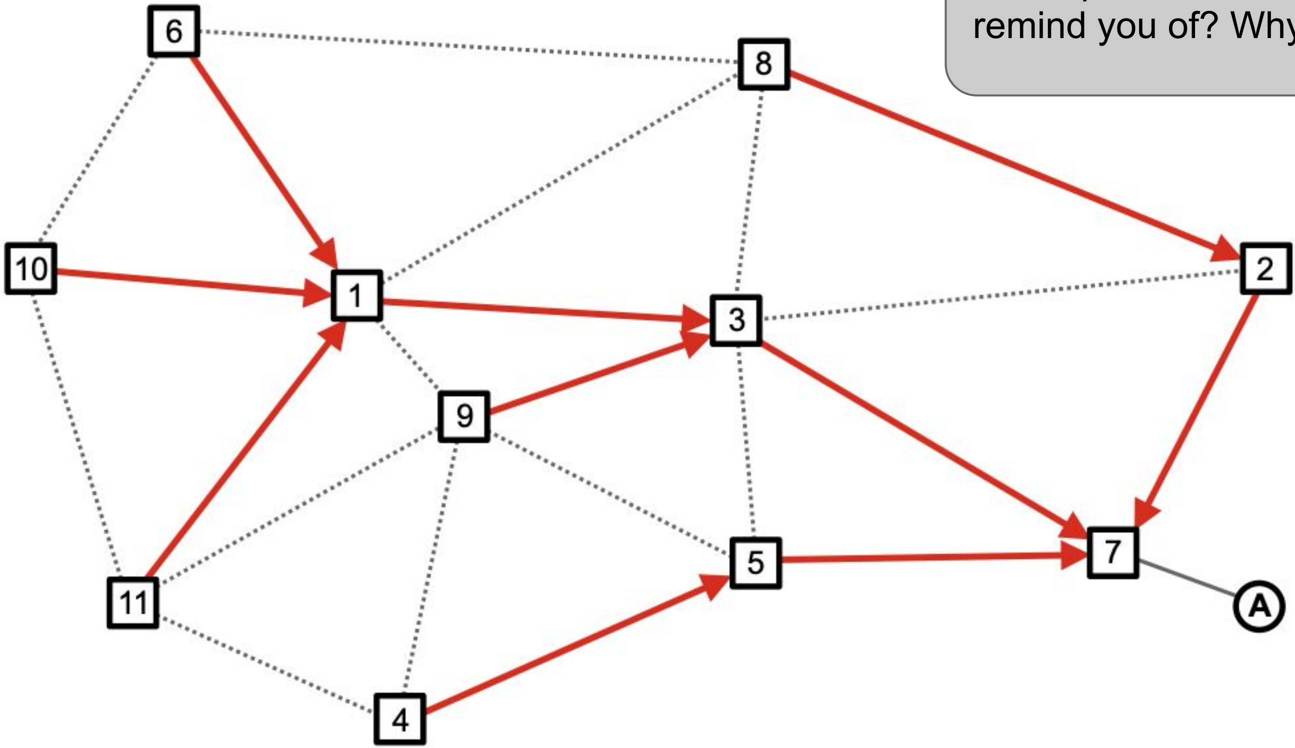
Split Horizon and Poison Reverse



All advertisements from all switches not sent due to split horizon

Split Horizon and Poison Reverse

Concept check: What does this remind you of? Why?



All poison reverse advertisements from all switches.

L2 vs L3 Routing

Routing in L2 vs L3

One more common confusion: we need routing for both L2 and L3, so which of Distance-Vector, Link-State, or Learning Switches do we use in each?

L2: responsible for *local* packet delivery.

L3: response for *global* packet delivery.

(How) is the routing problem different at each layer?

Routing in L2 vs L3

Layer 3

Goals:

- Short paths
- Resilient to failure
- Paths for all destinations in a *(potentially large)* network

Properties:

- Must be scalable -> get a new address when you join a new network -> allows routing to group based on CIDR prefixes

Solutions:

- Distance-Vector
- Link-State

Addressing gave us some static configuration, which we can use to "seed" DV and LS

Layer 2

Goals:

- Short paths
- Resilient to failure
- Paths for all destinations in a *local* network
- **"Plug and play"**

Properties:

- Small network
- Operate with low cost

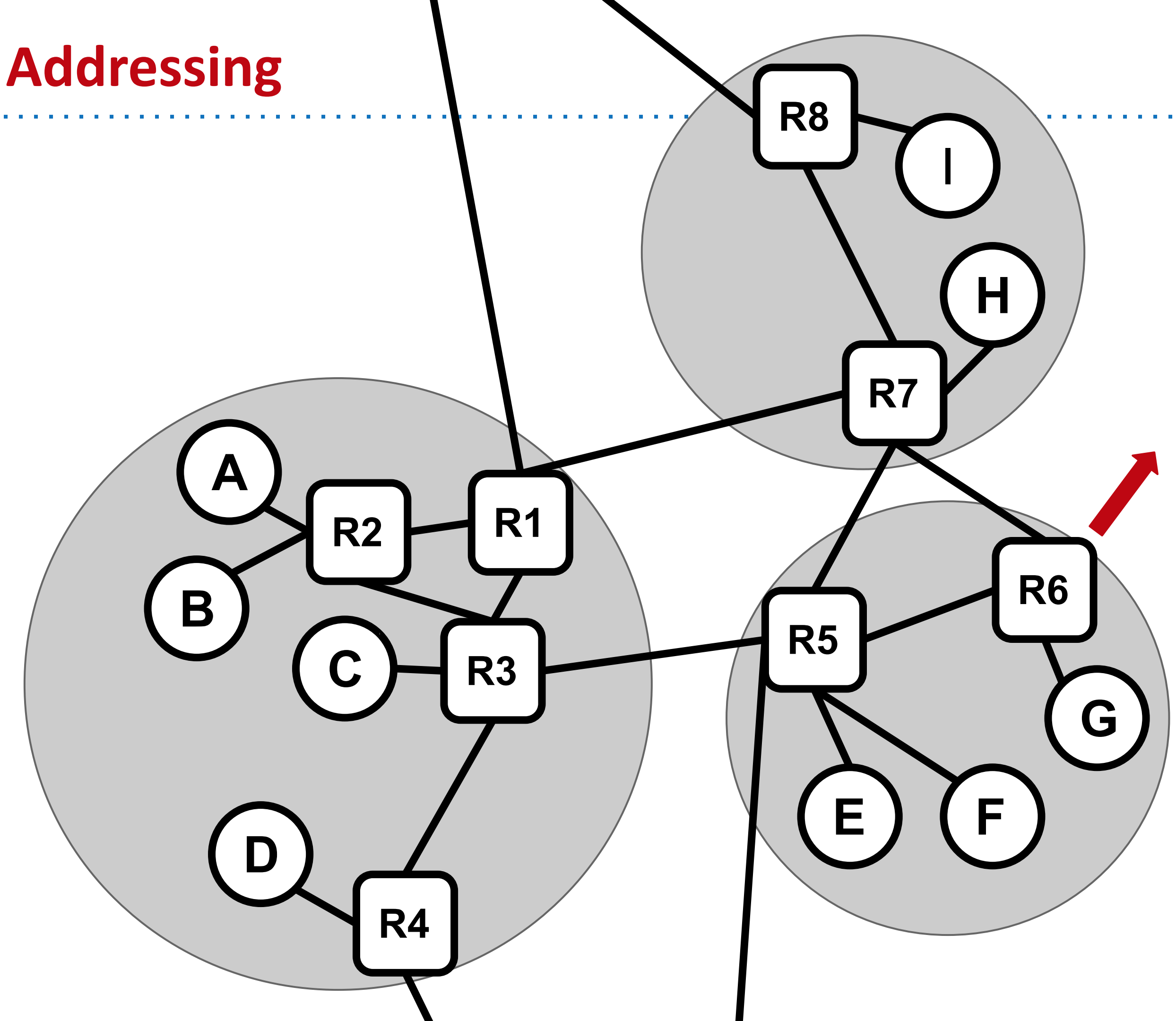
With a small network, we can afford to flood and learn paths to avoid configuration

Solutions:

- Learning switches + STP

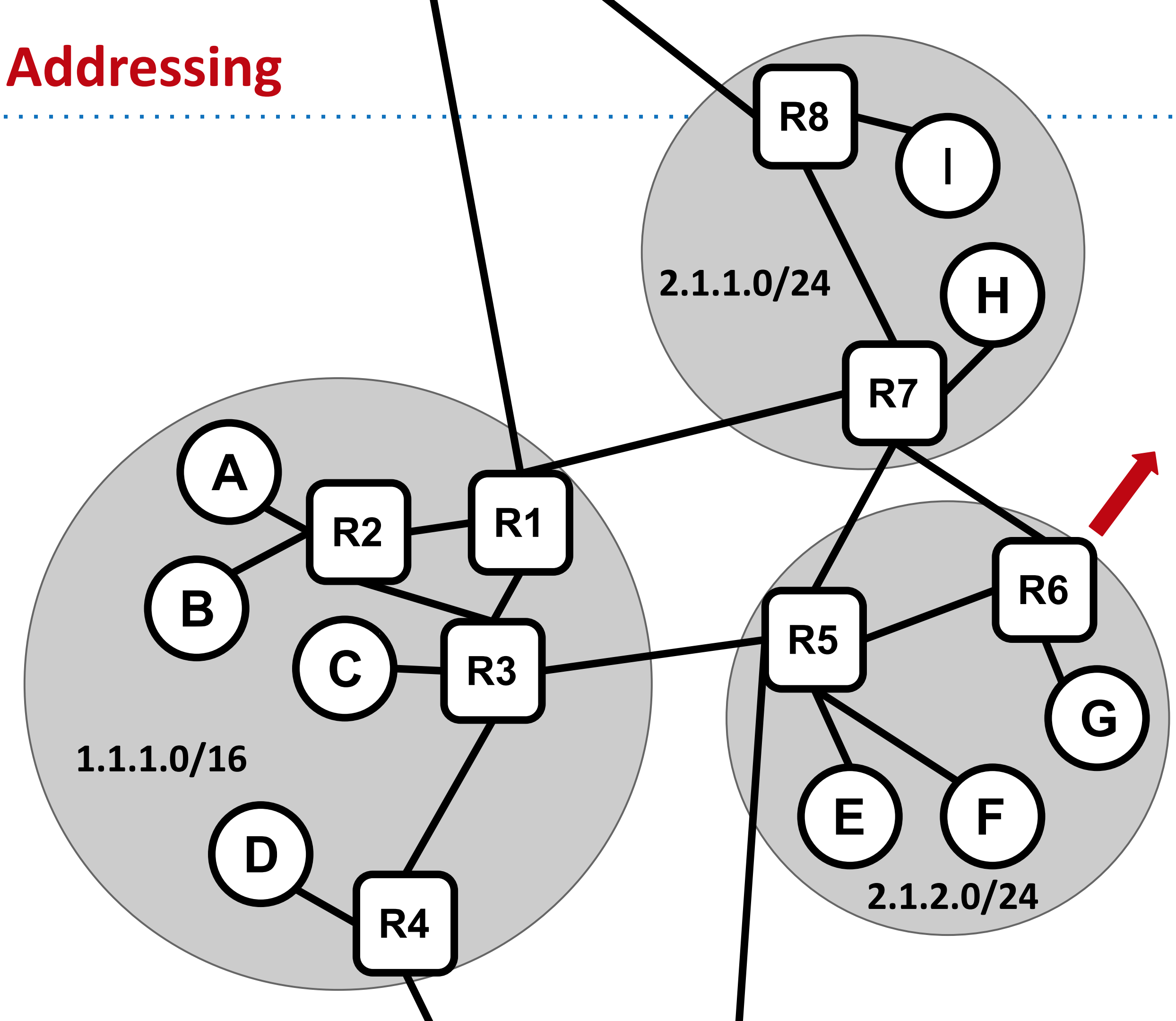
Addressing

Addressing



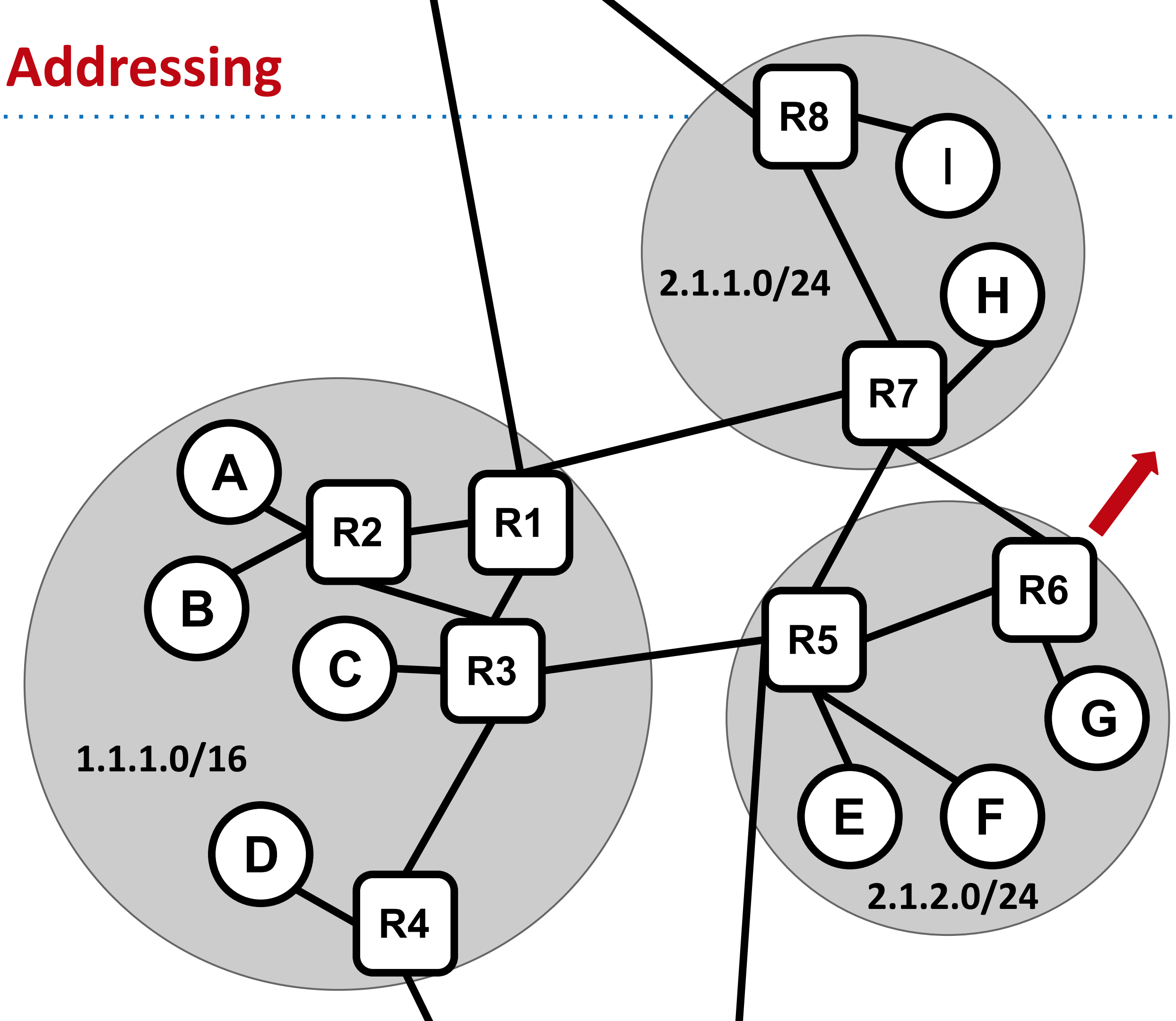
Destination	Next Hop, Cost
A	R5, 4
B	R5, 4
C	R5, 3
D	R5, 4
E	R5, 2
F	R5, 2
H	R7, 2
I	R7, 4
…	

Addressing



Destination	Next Hop, Cost
A	R5, 4
B	R5, 4
C	R5, 3
D	R5, 4
E	R5, 2
F	R5, 2
H	R7, 2
I	R7, 4
…	

Addressing



Destination	Next Hop, Cost
1.1.1.0/16	R5, 2
2.1.1.0/24	R7, 1
2.1.2.0/30	R5, 2
2.1.2.4/30	G, 1

Break

Learning Switches and Spanning Tree Protocol

Narek Galstyan

Learning Switches and Spanning Tree Protocol

Goal for these slides:

- Walk through examples of Learning switch and STP algorithm runs
- Consider corner cases
- Check our understanding of them from lectures

Learning Switches and Spanning Tree Protocol

Goal for these slides:

- Walk through examples of Learning switch and STP algorithm runs
- Consider corner cases
- Check our understanding of them from lectures

Red boxes are questions for you to think about

- ○ And shout out answers!

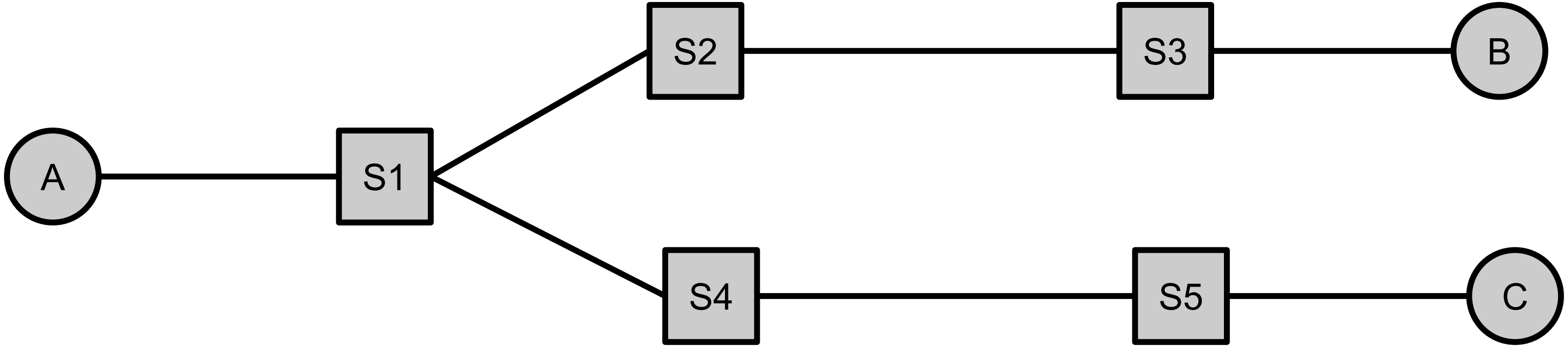
I will have answers in the following slide in a green box

- ○ Please do answer them before

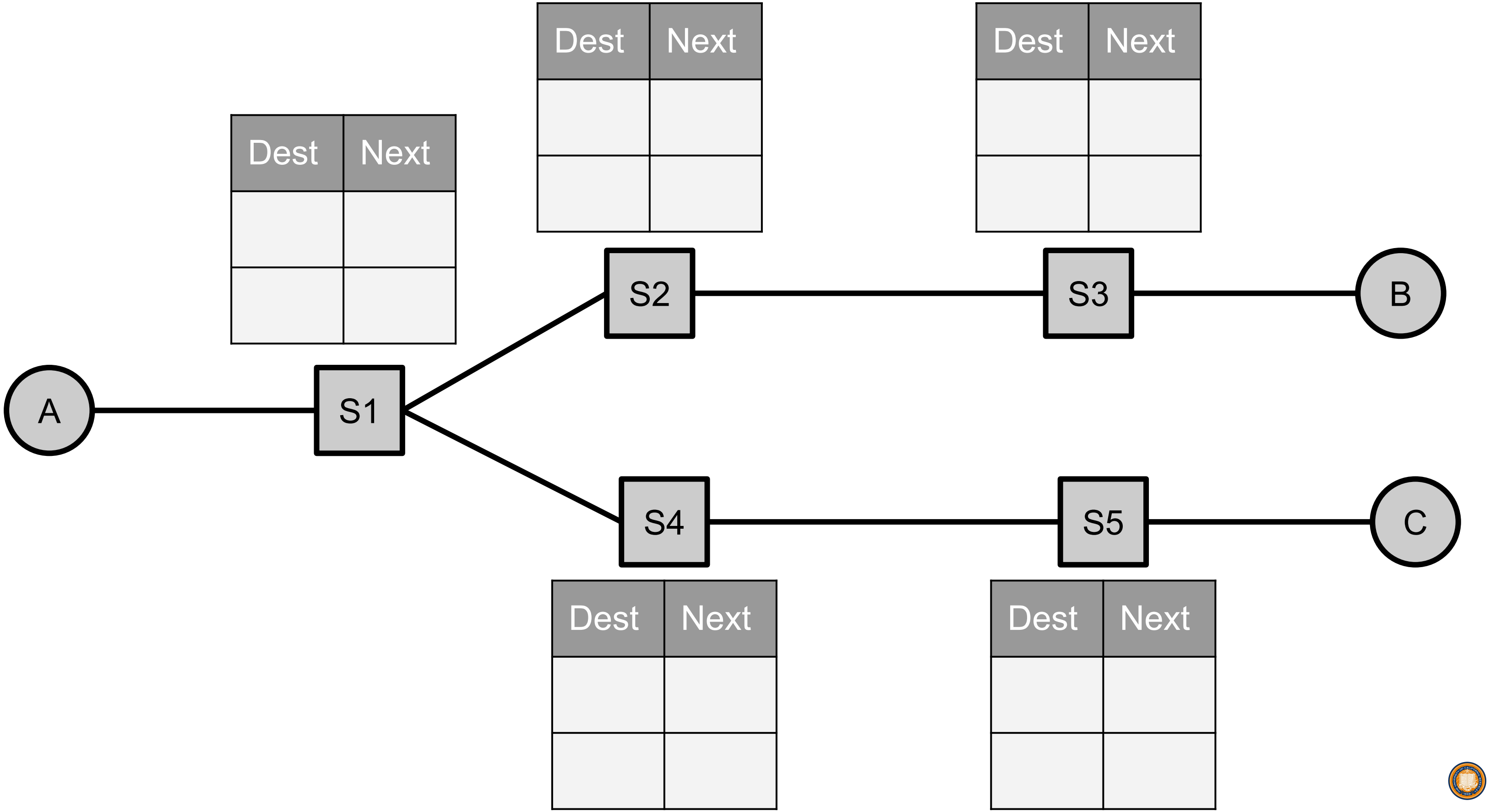
We chose some arbitrary order of message exchanges. What would happen if we chose a different order?

We would reach the same equilibrium!

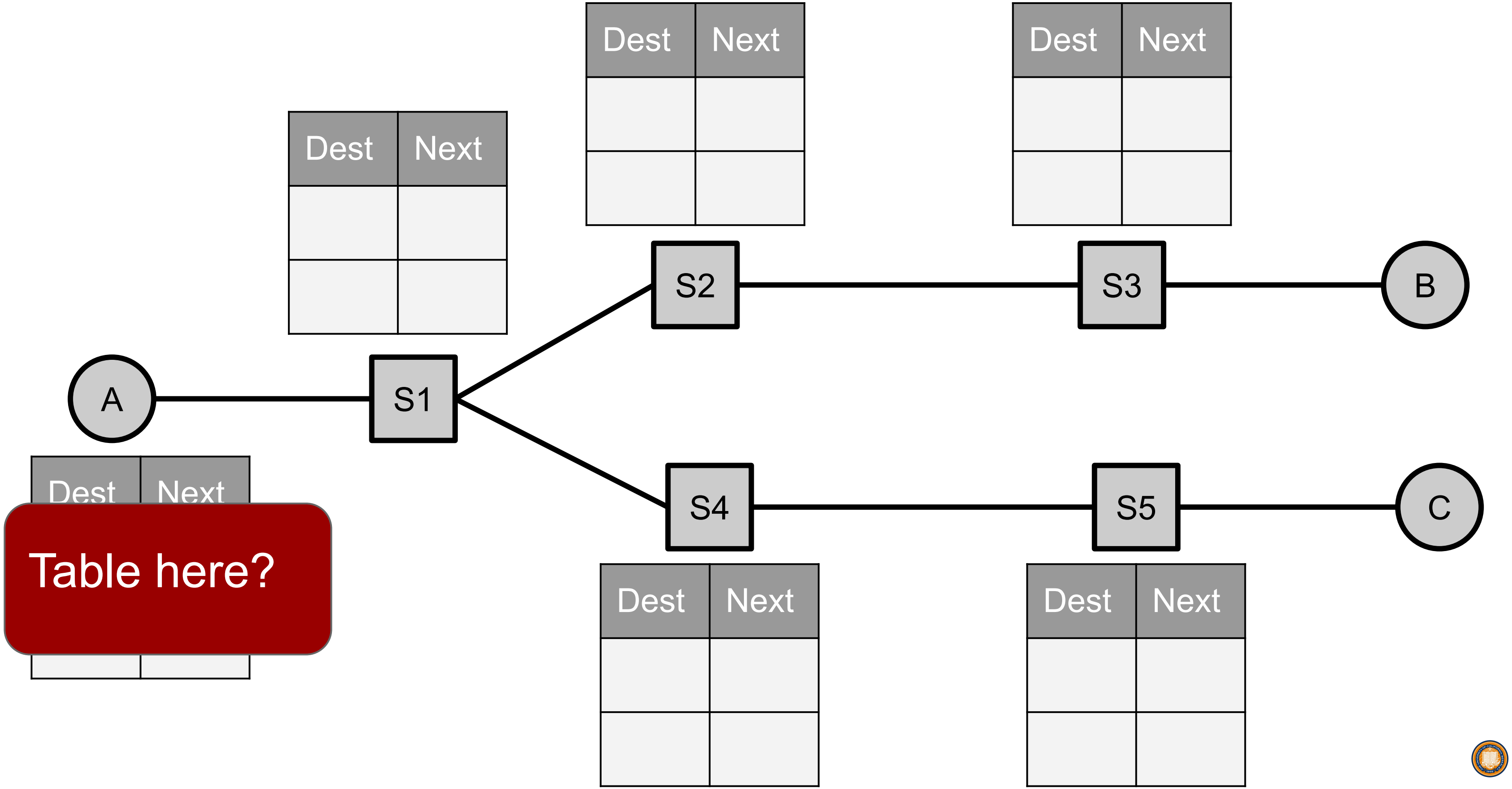
Learning Switches



Learning Switches



Learning Switches



Learning Switches

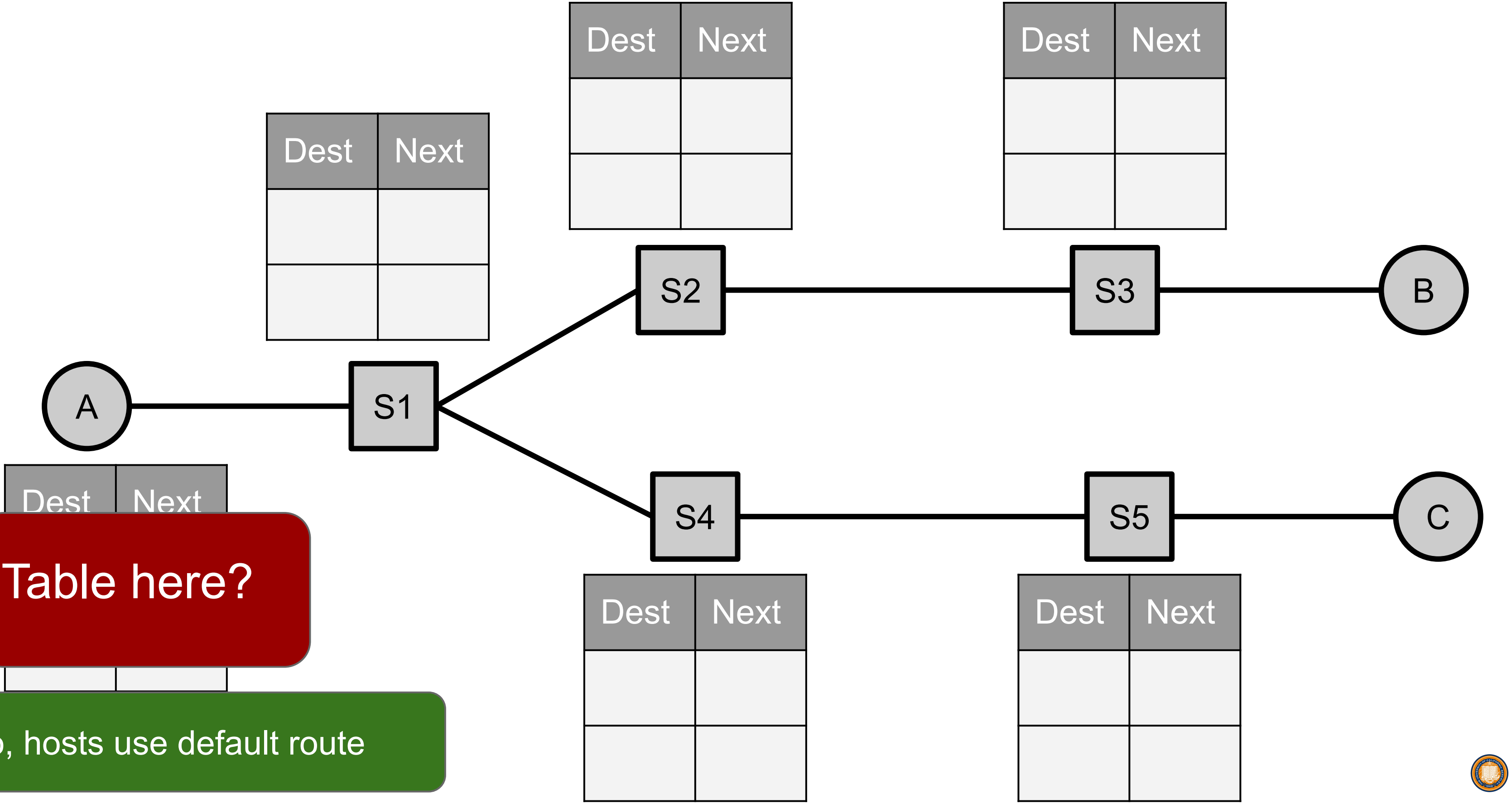
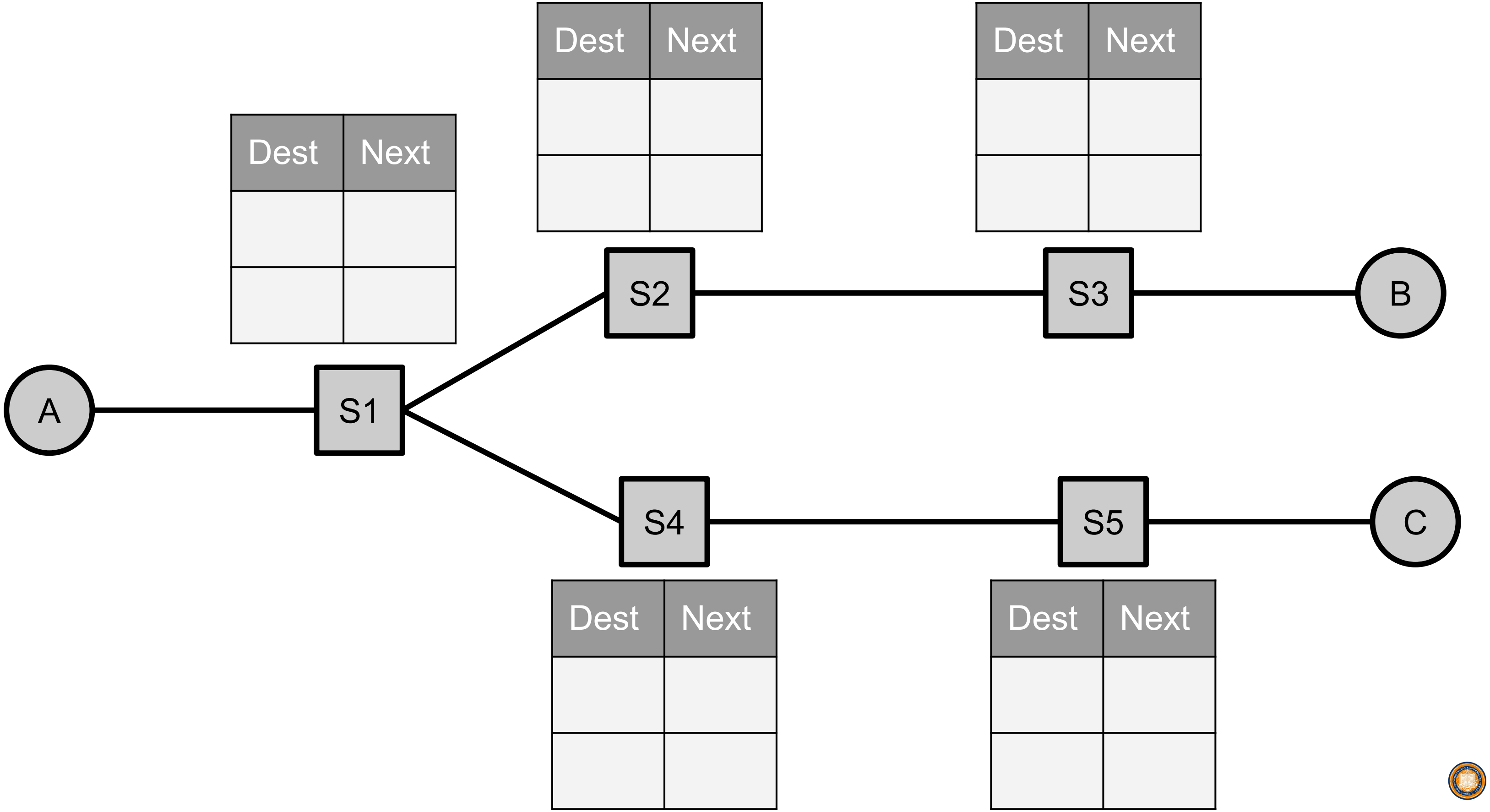


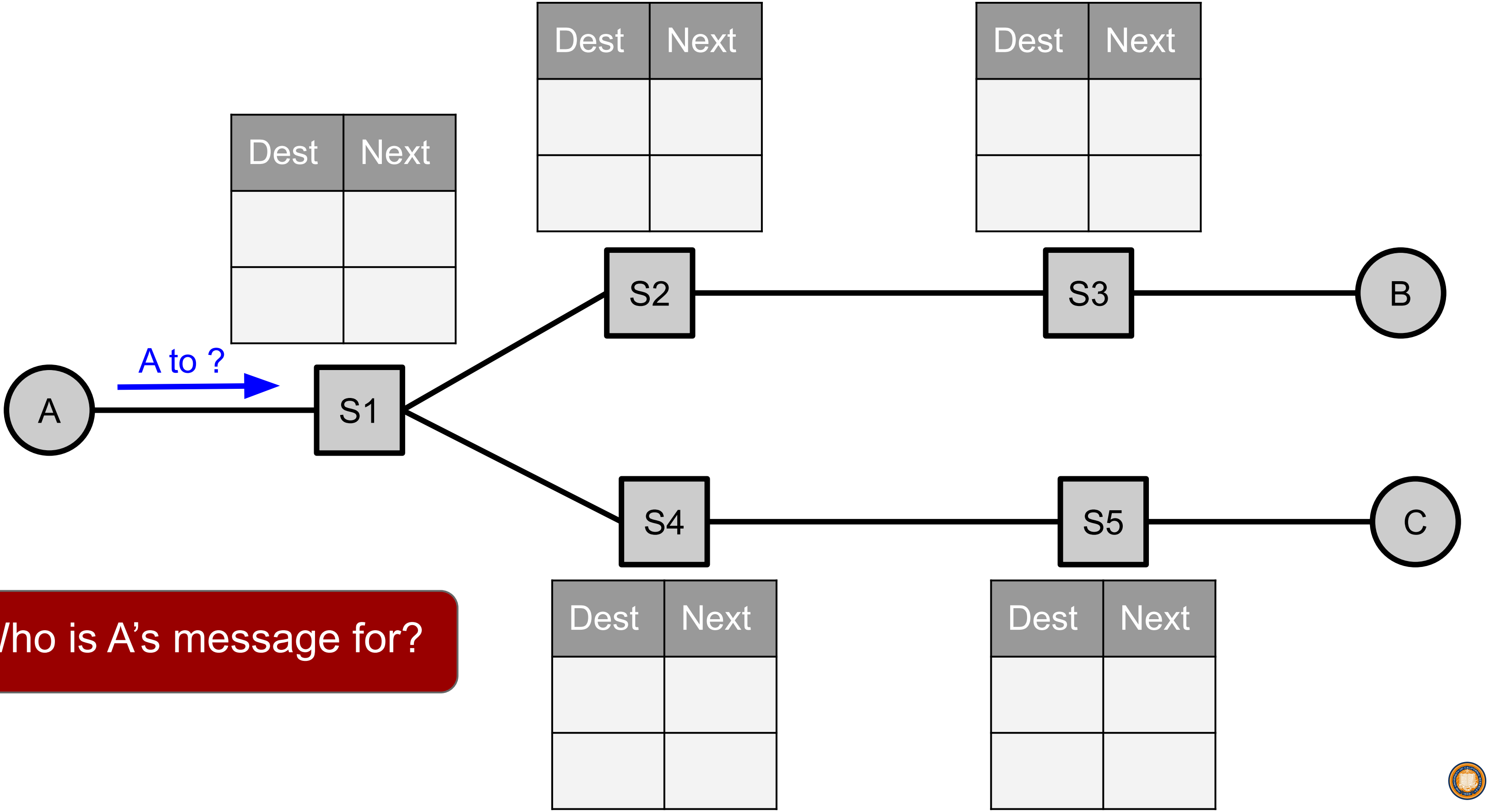
Table here?

No, hosts use default route

Learning Switches



Learning Switches



Dest	Next

Dest	Next

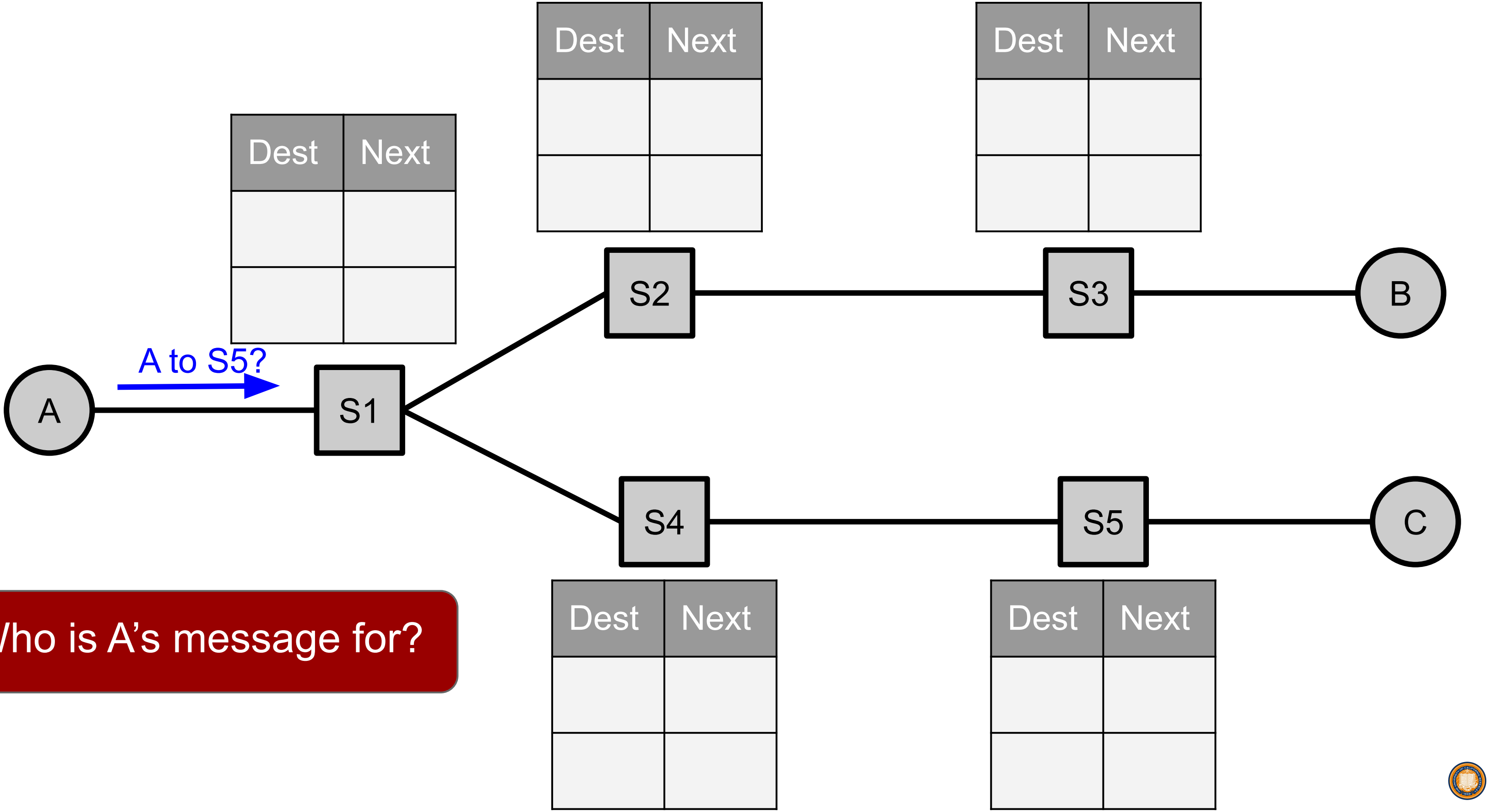
Dest	Next

Dest	Next

Dest	Next

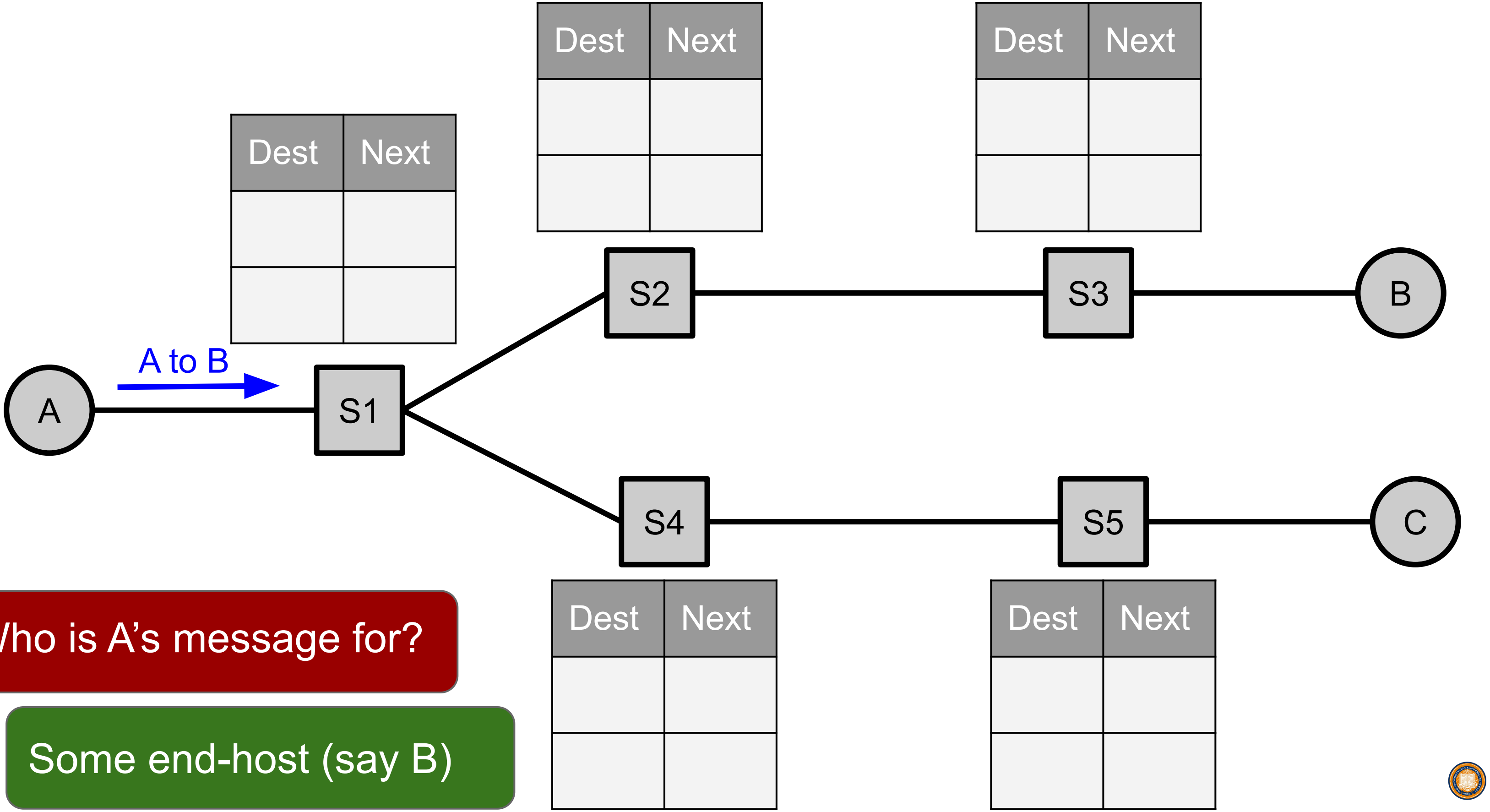
Who is A's message for?

Learning Switches



Who is A's message for?

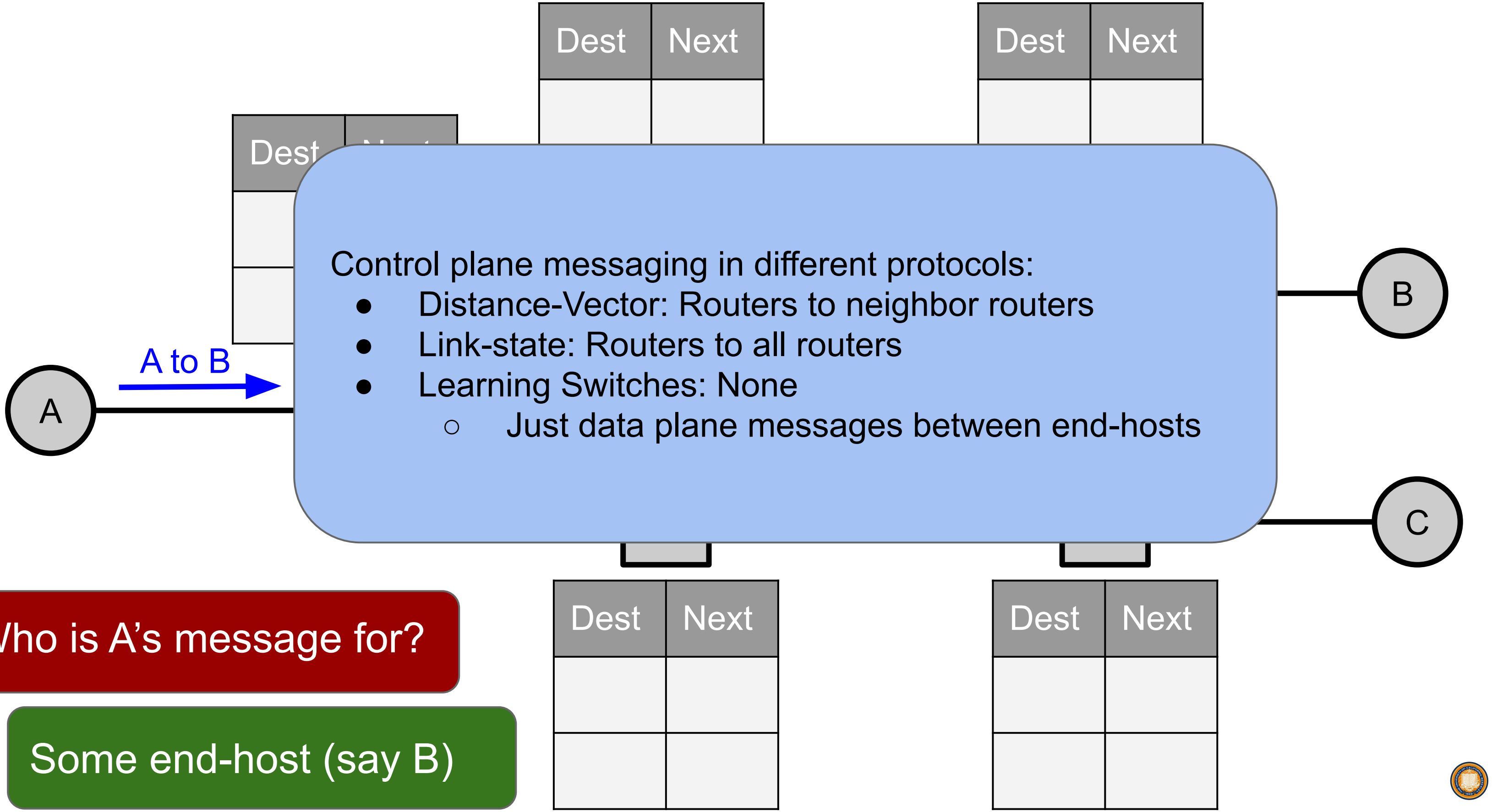
Learning Switches



Who is A's message for?

Some end-host (say B)

Learning Switches



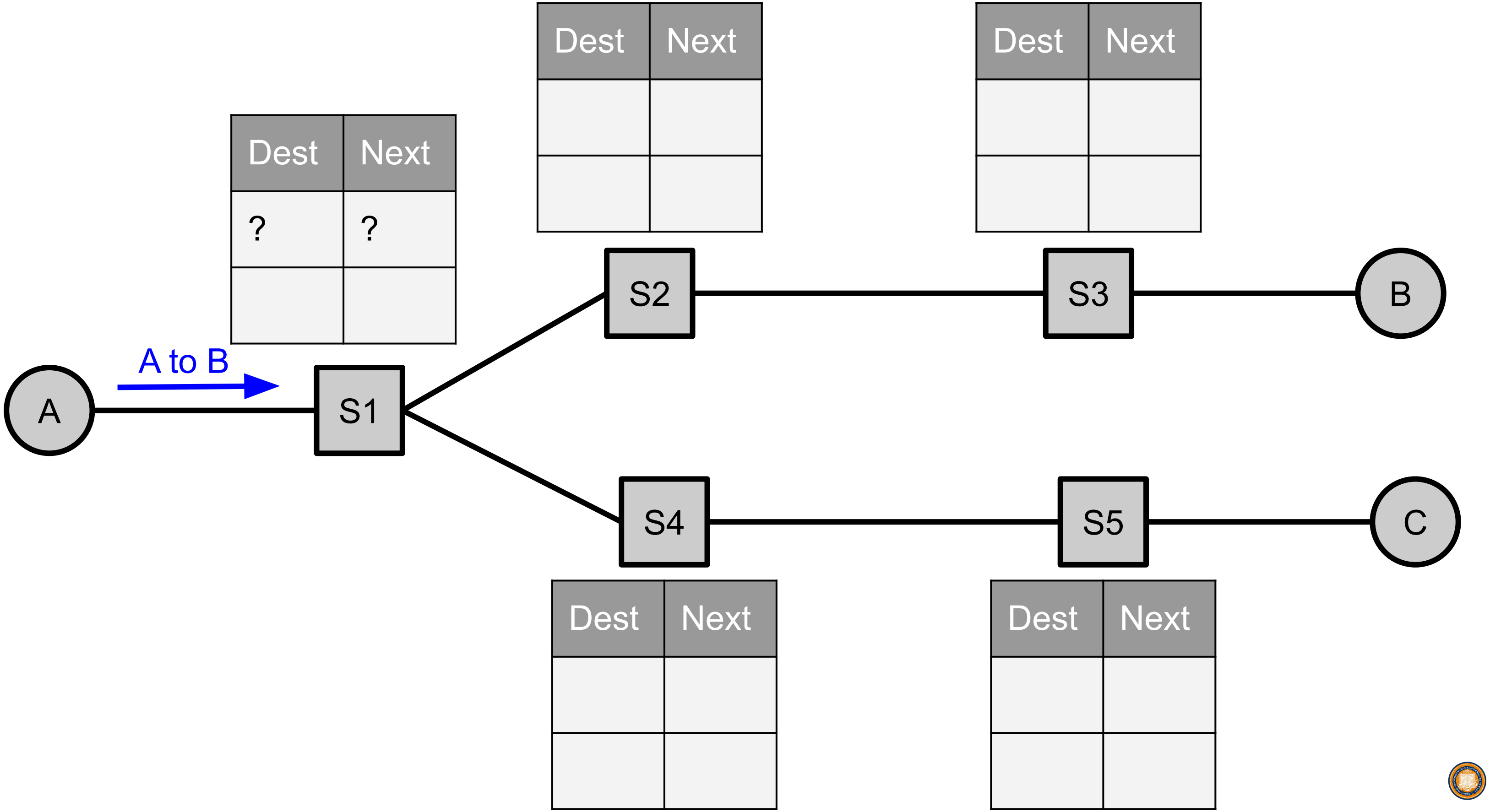
Control plane messaging in different protocols:

- Distance-Vector: Routers to neighbor routers
- Link-state: Routers to all routers
- Learning Switches: None
  - Just data plane messages between end-hosts

Who is A's message for?

Some end-host (say B)

Learning Switches



S1:

Dest	Next
?	?

S2:

Dest	Next

S3:

Dest	Next

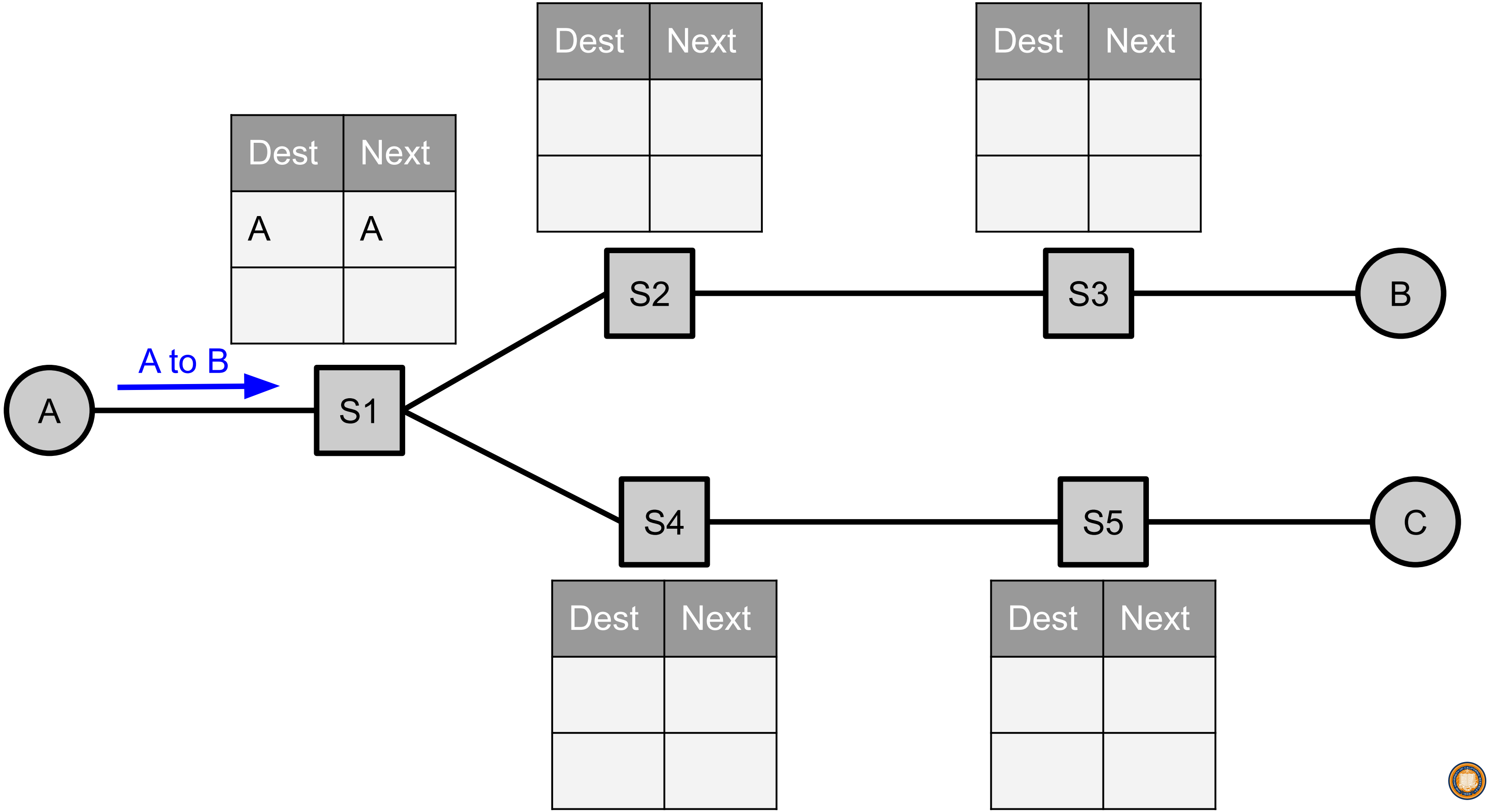
S4:

Dest	Next

S5:

Dest	Next

Learning Switches



Dest	Next

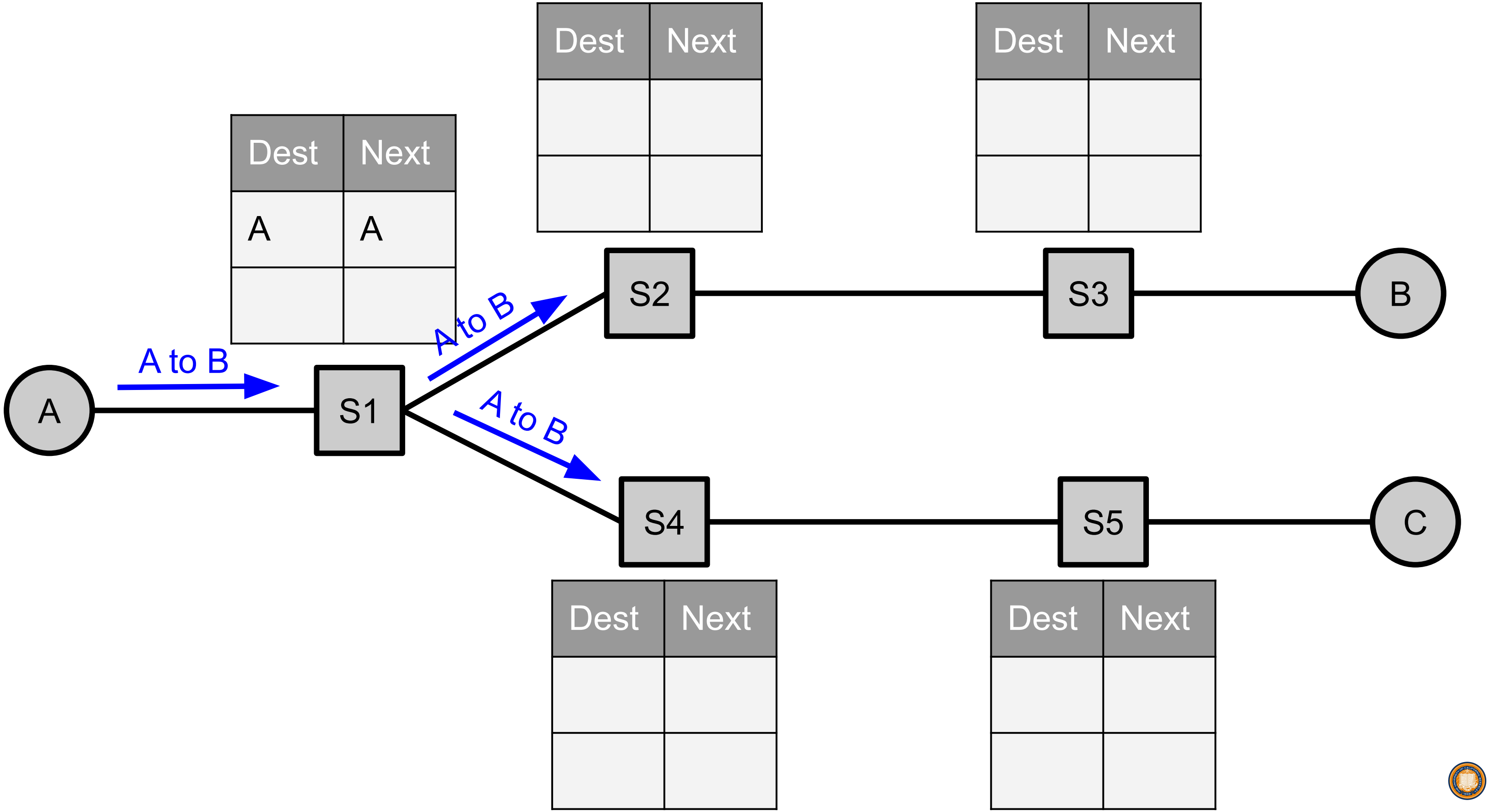
Dest	Next

Dest	Next
A	A

Dest	Next

Dest	Next

Learning Switches



Dest	Next
A	A

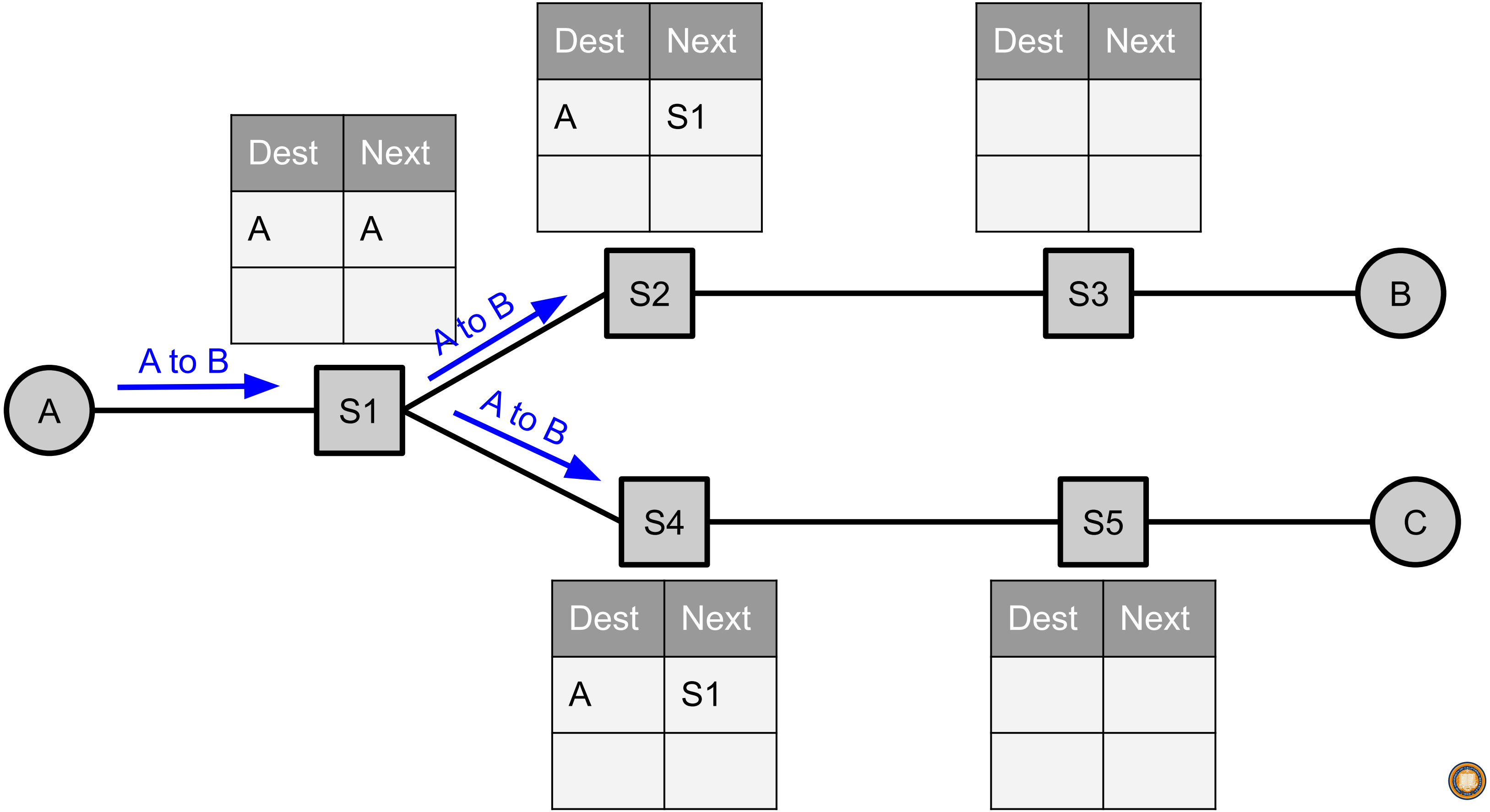
Dest	Next

Dest	Next

Dest	Next

Dest	Next

Learning Switches



S1:

Dest	Next
A	A

S2:

Dest	Next
A	S1

S3:

Dest	Next

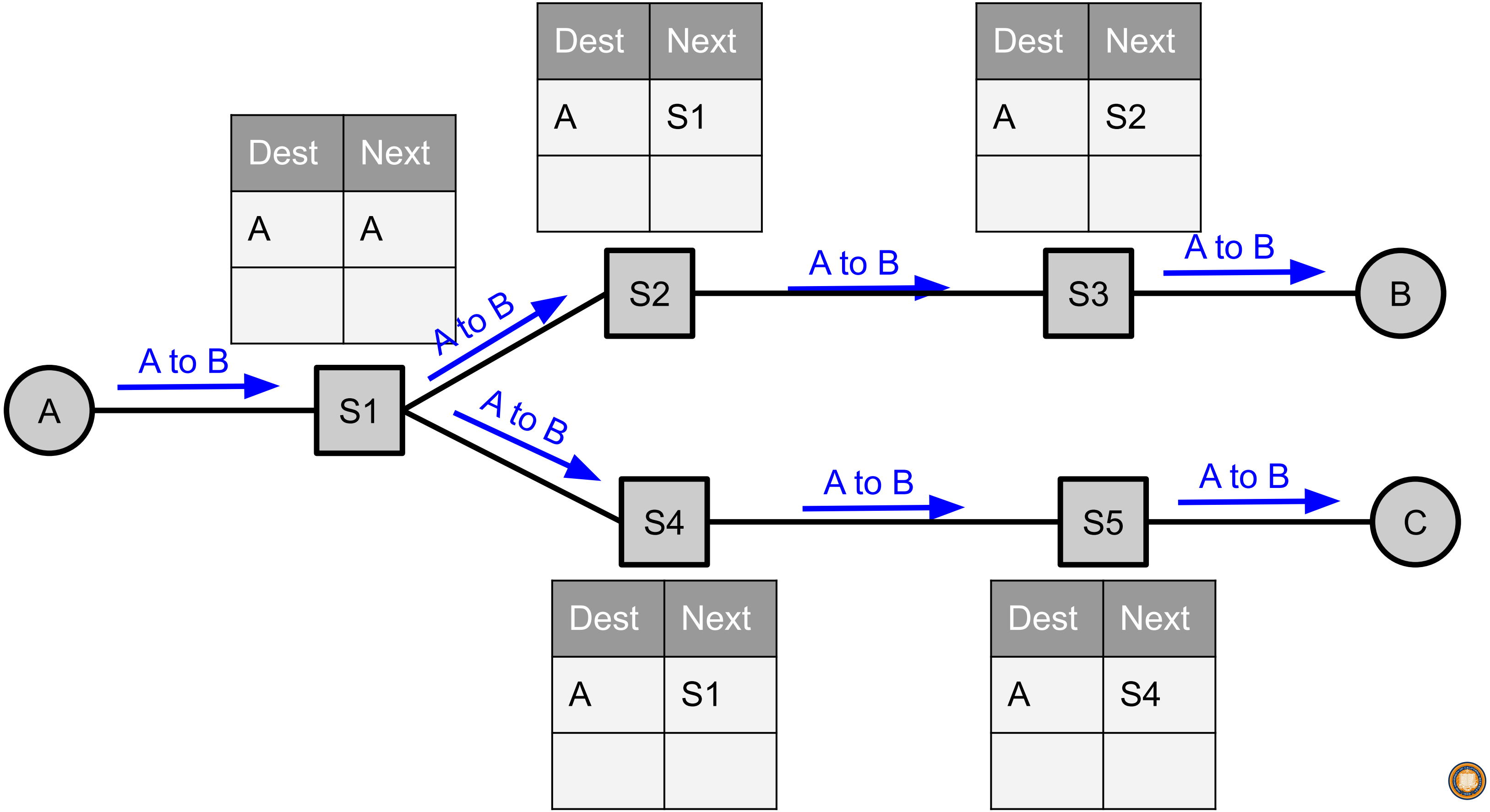
S4:

Dest	Next
A	S1

S5:

Dest	Next

Learning Switches



Dest	Next
A	A

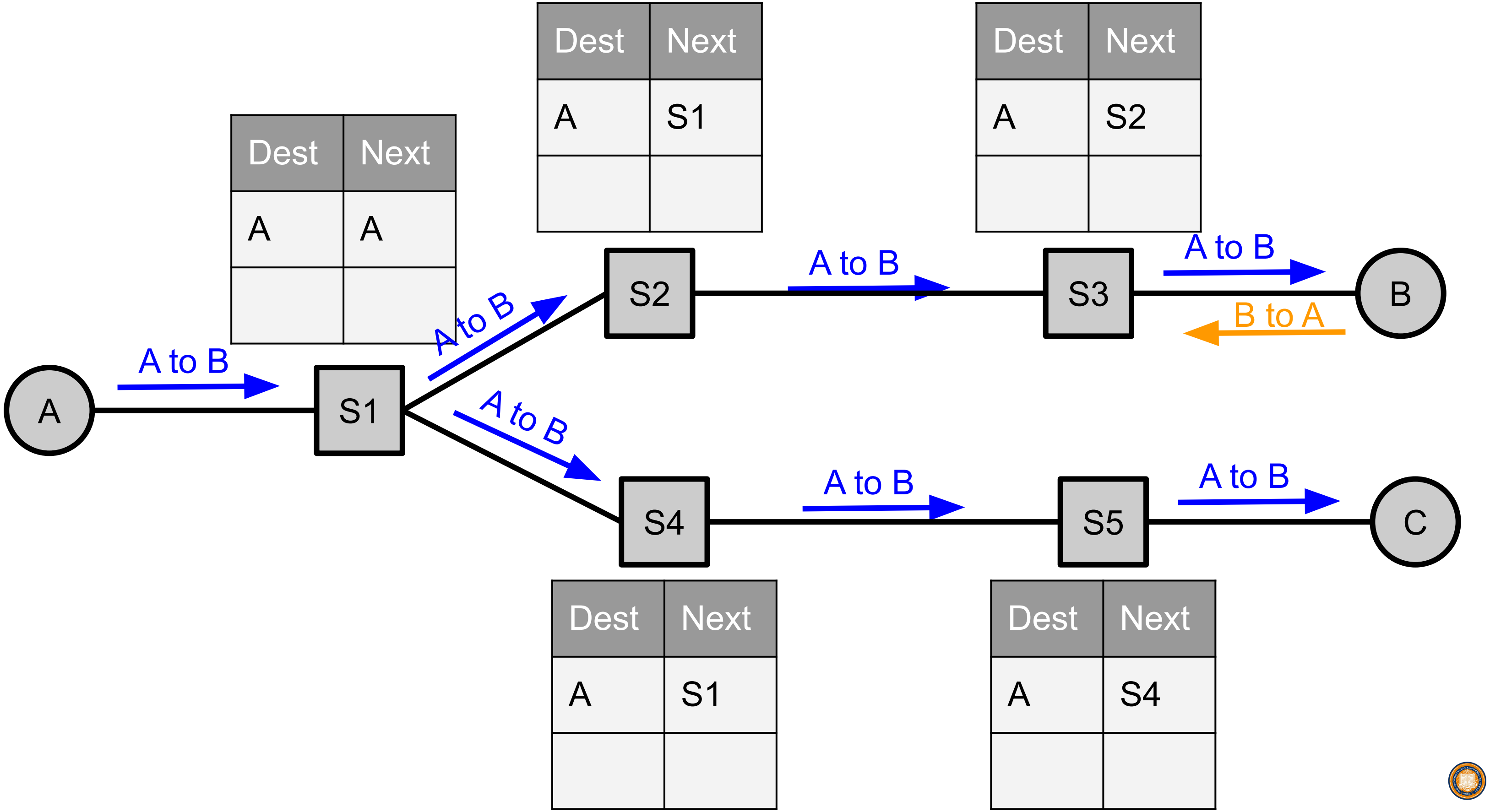
Dest	Next
A	S1

Dest	Next
A	S2

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches



Dest	Next
A	A

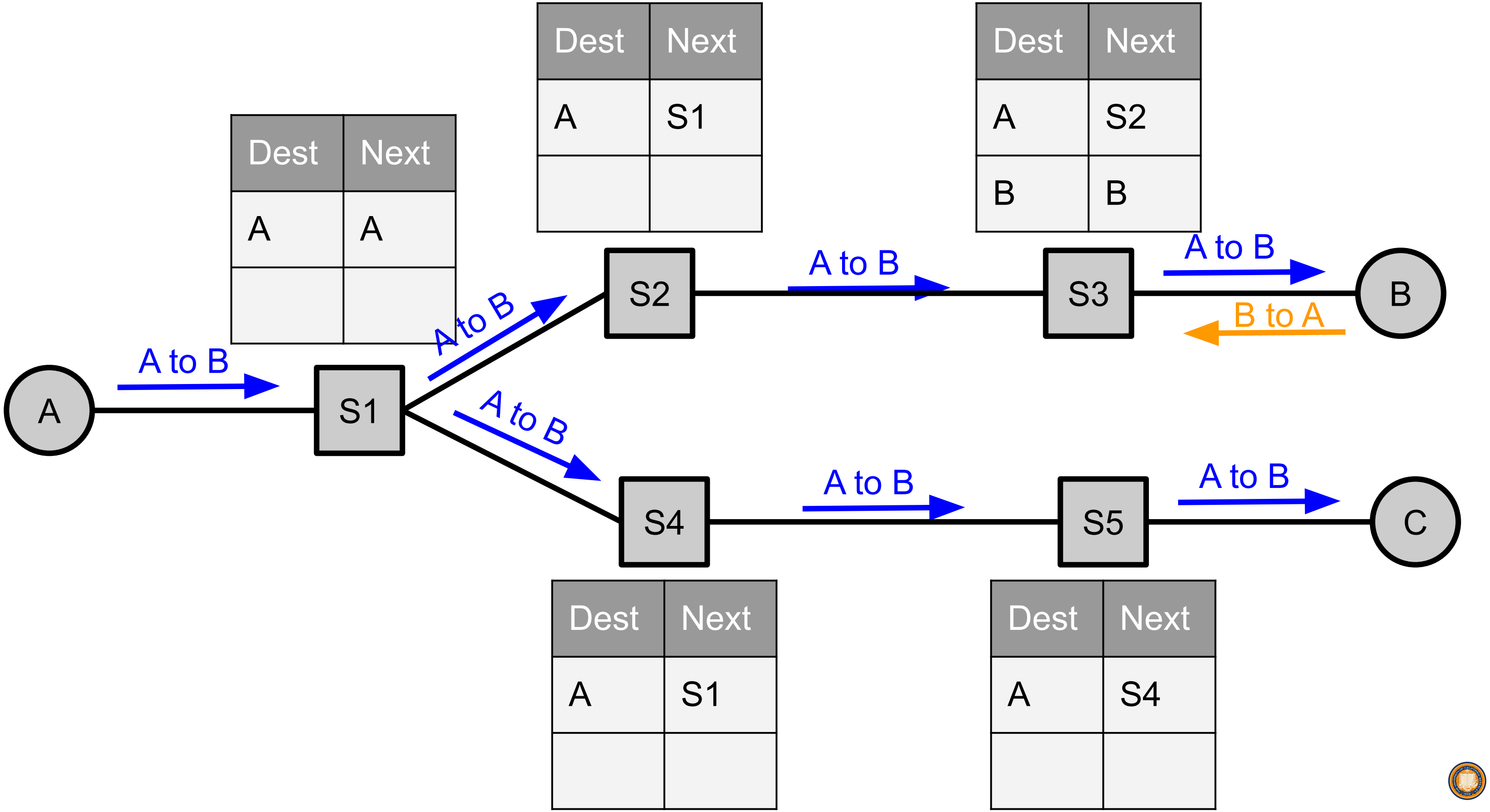
Dest	Next
A	S1

Dest	Next
A	S2

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches



Dest	Next
A	A

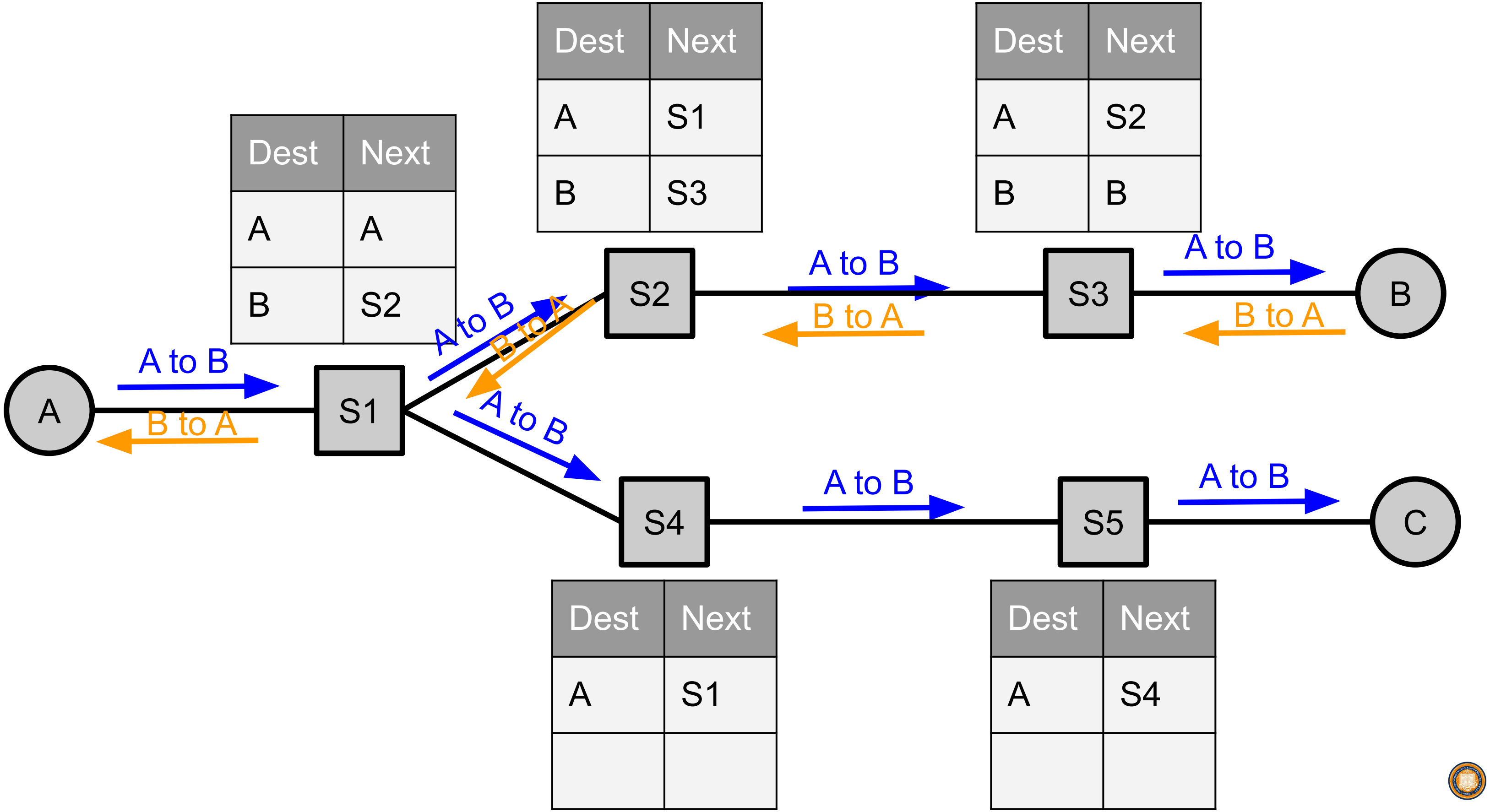
Dest	Next
A	S1

Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches



Dest	Next
A	A
B	S2

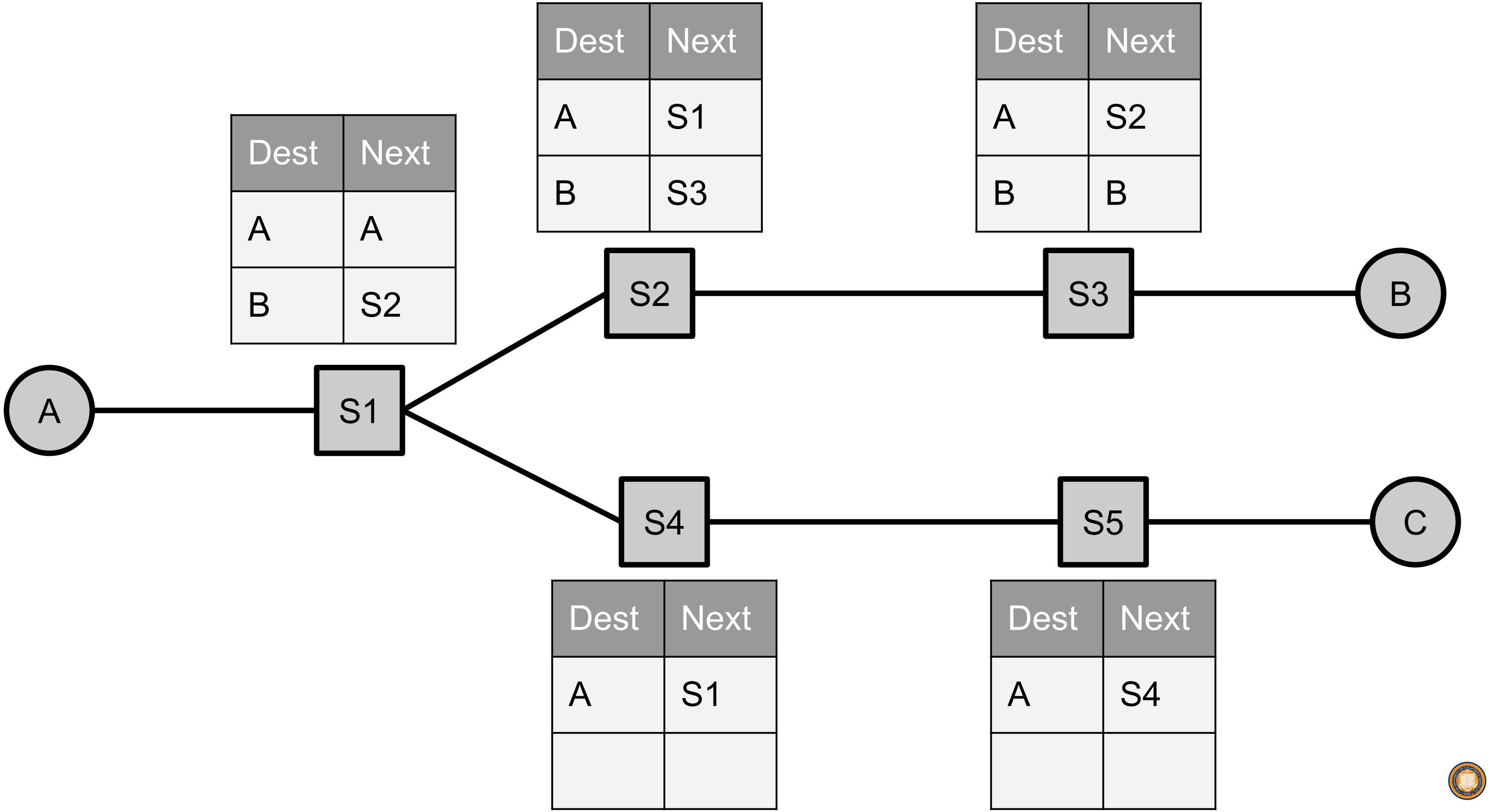
Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches



Dest	Next
A	A
B	S2

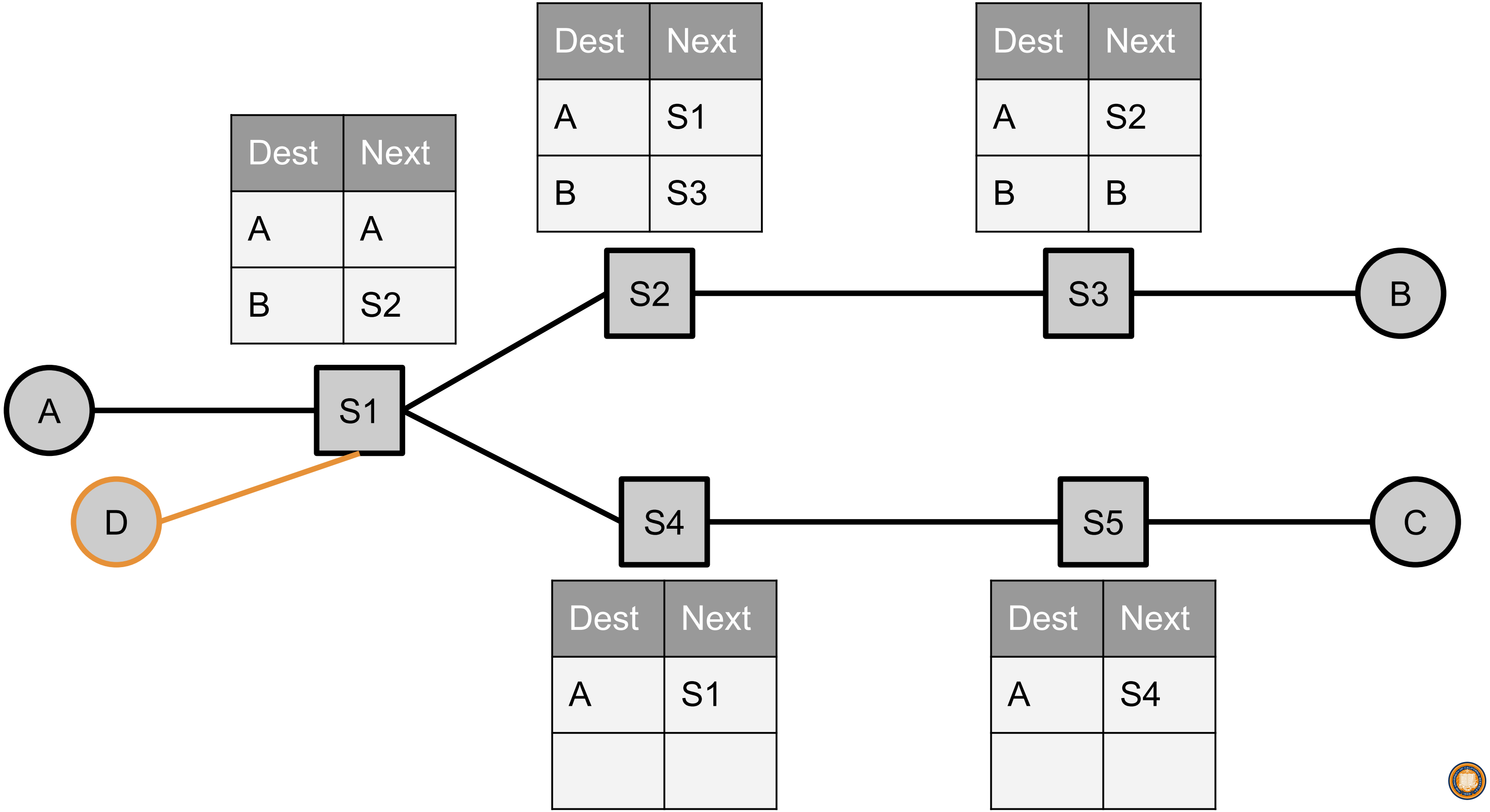
Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches



Dest	Next
A	A
B	S2

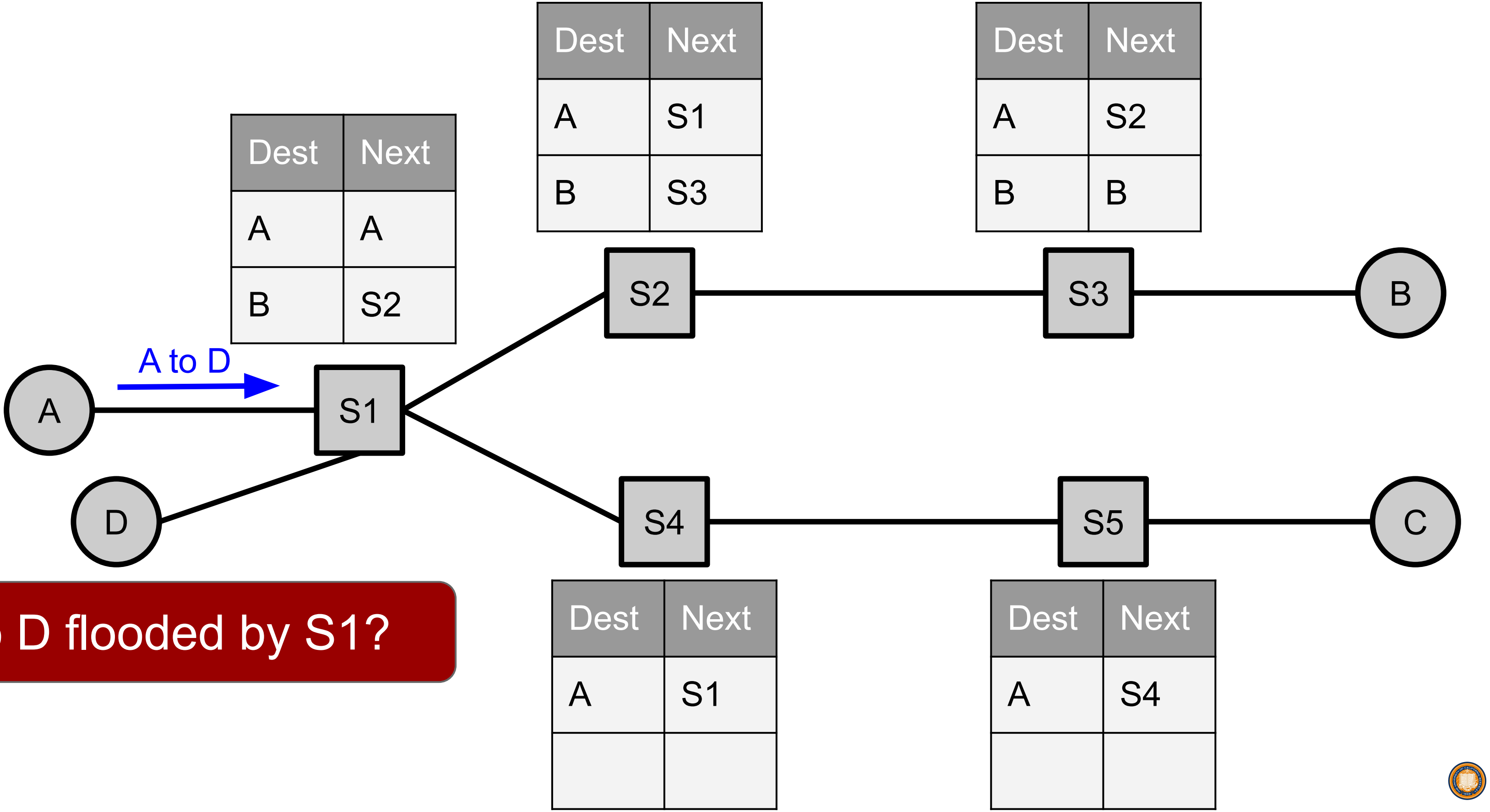
Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

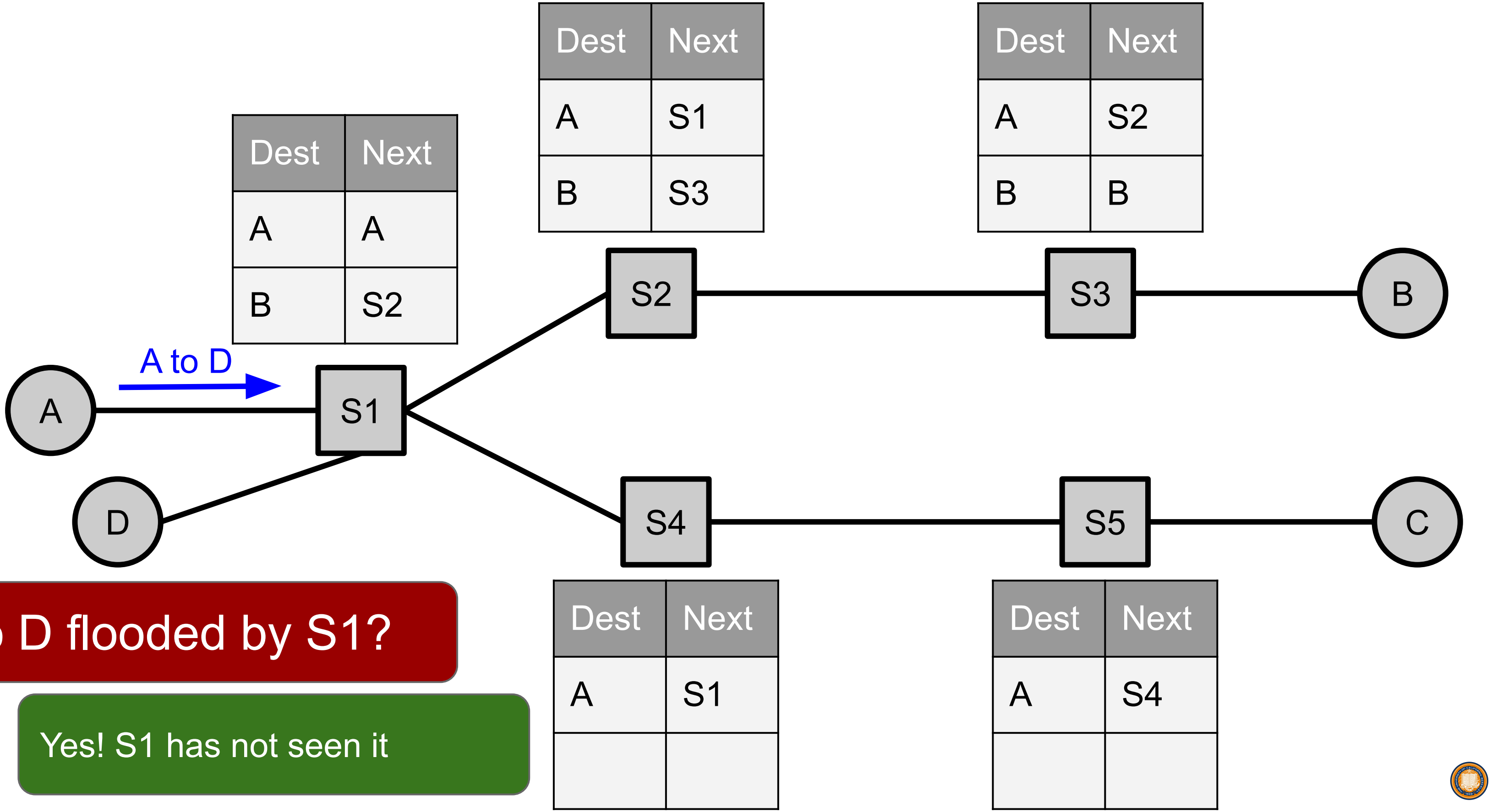
Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

Is A to D flooded by S1?

Learning Switches



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

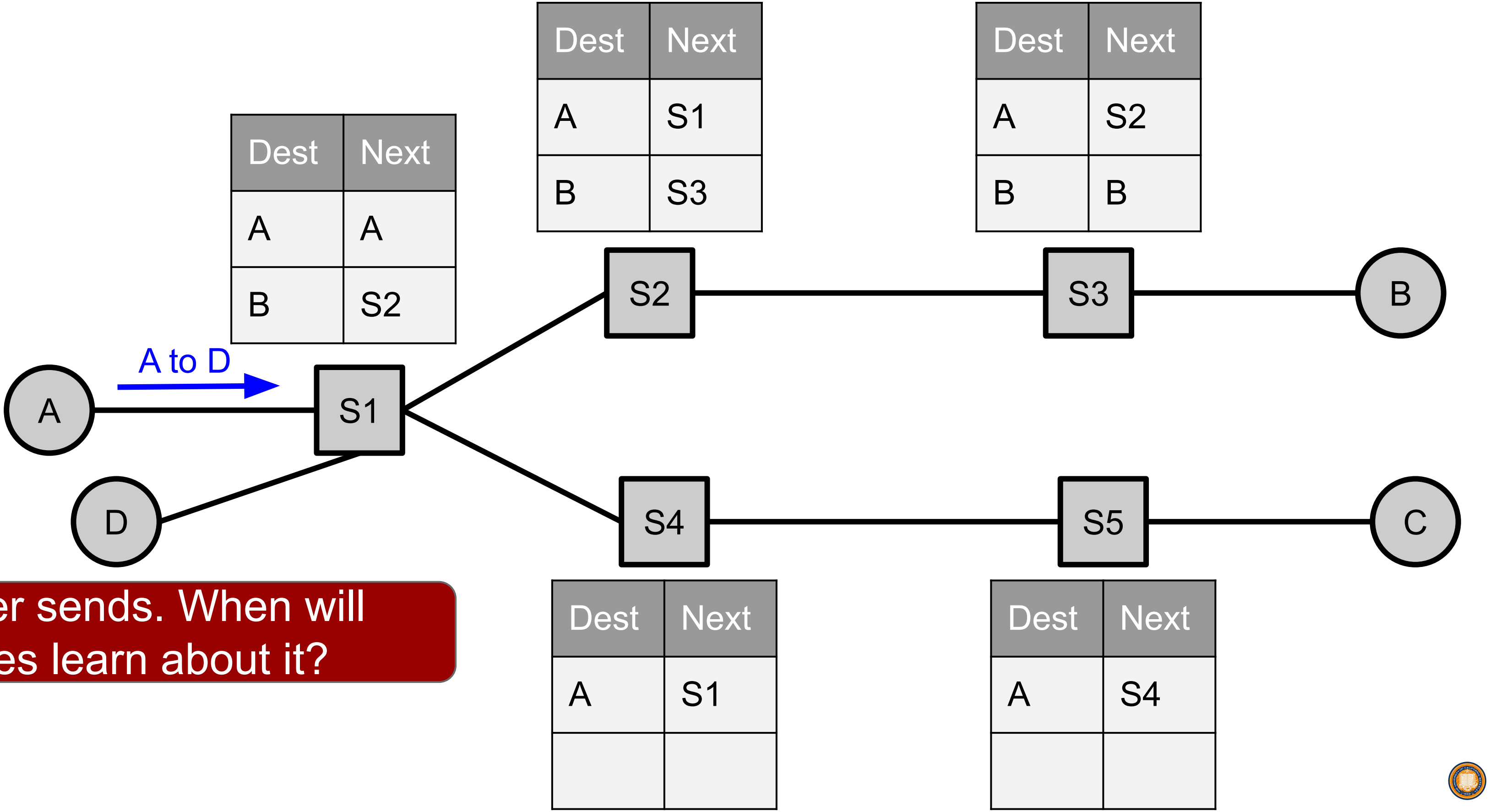
Dest	Next
A	S1

Dest	Next
A	S4

Is A to D flooded by S1?

Yes! S1 has not seen it

Learning Switches



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

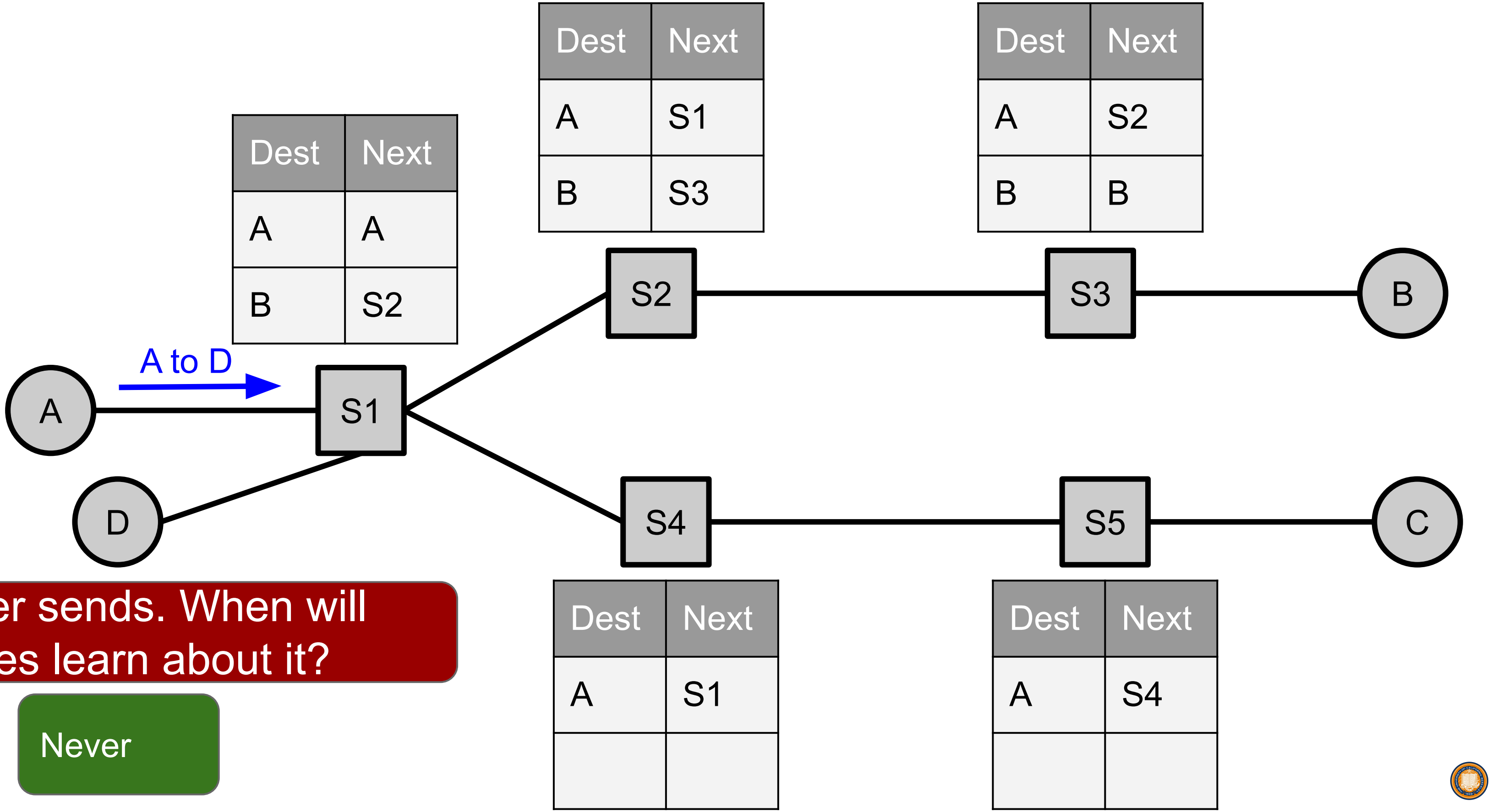
Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

D never sends. When will switches learn about it?

Learning Switches



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

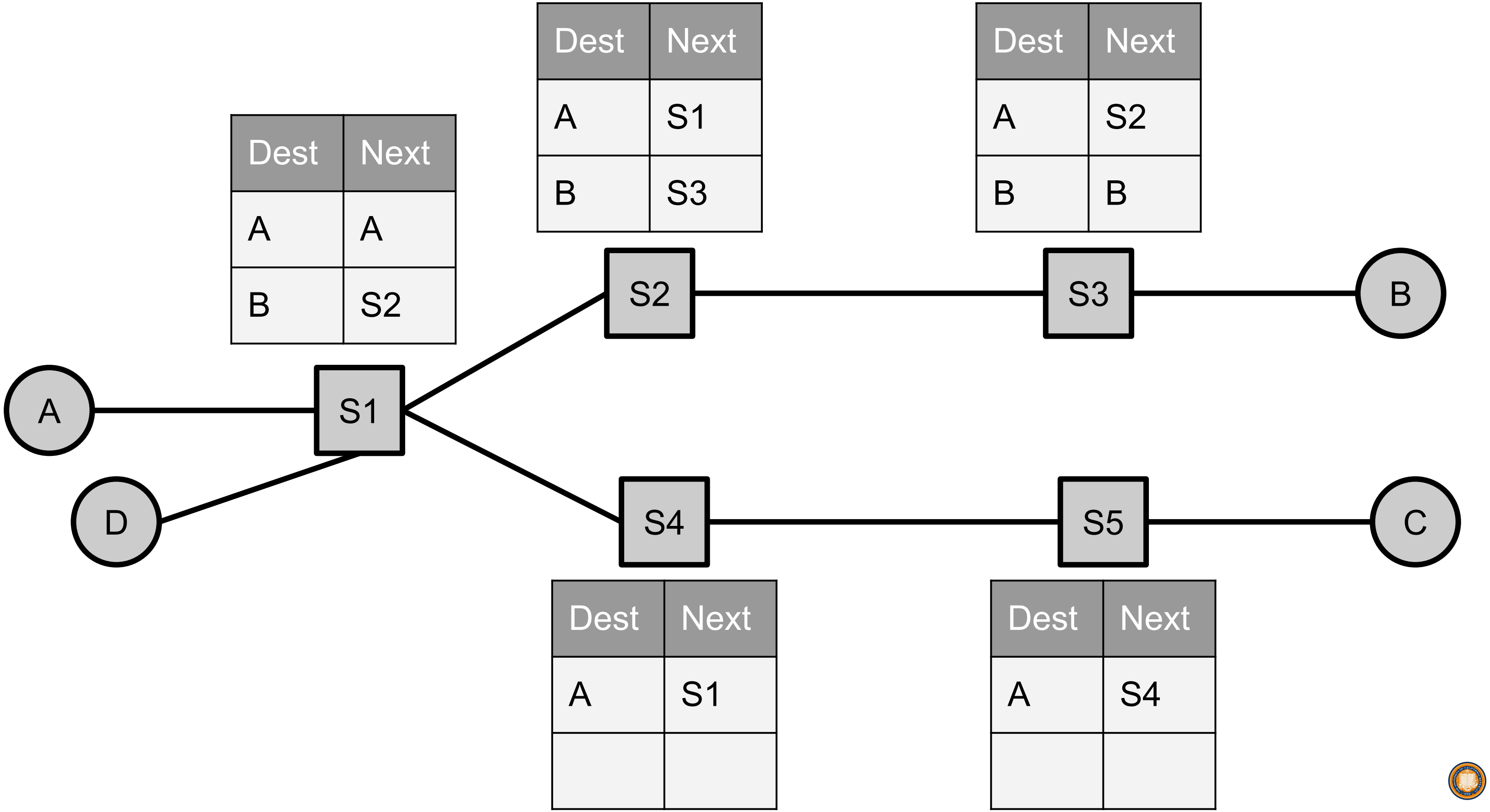
Dest	Next
A	S1

Dest	Next
A	S4

D never sends. When will switches learn about it?

Never

Learning Switches bad news



Dest	Next
A	A
B	S2

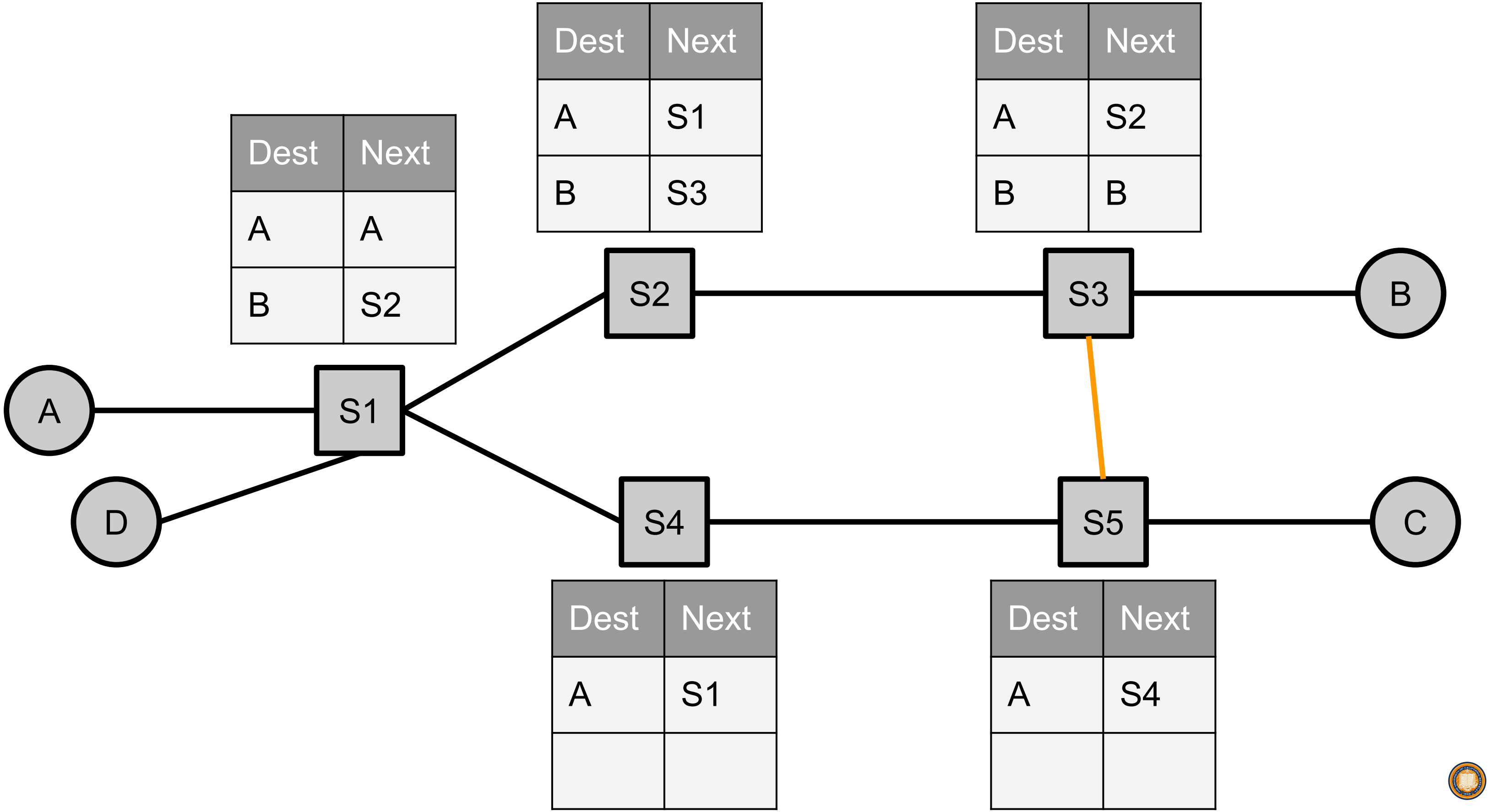
Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches bad news: Loops!



Dest	Next
A	S1
B	S3

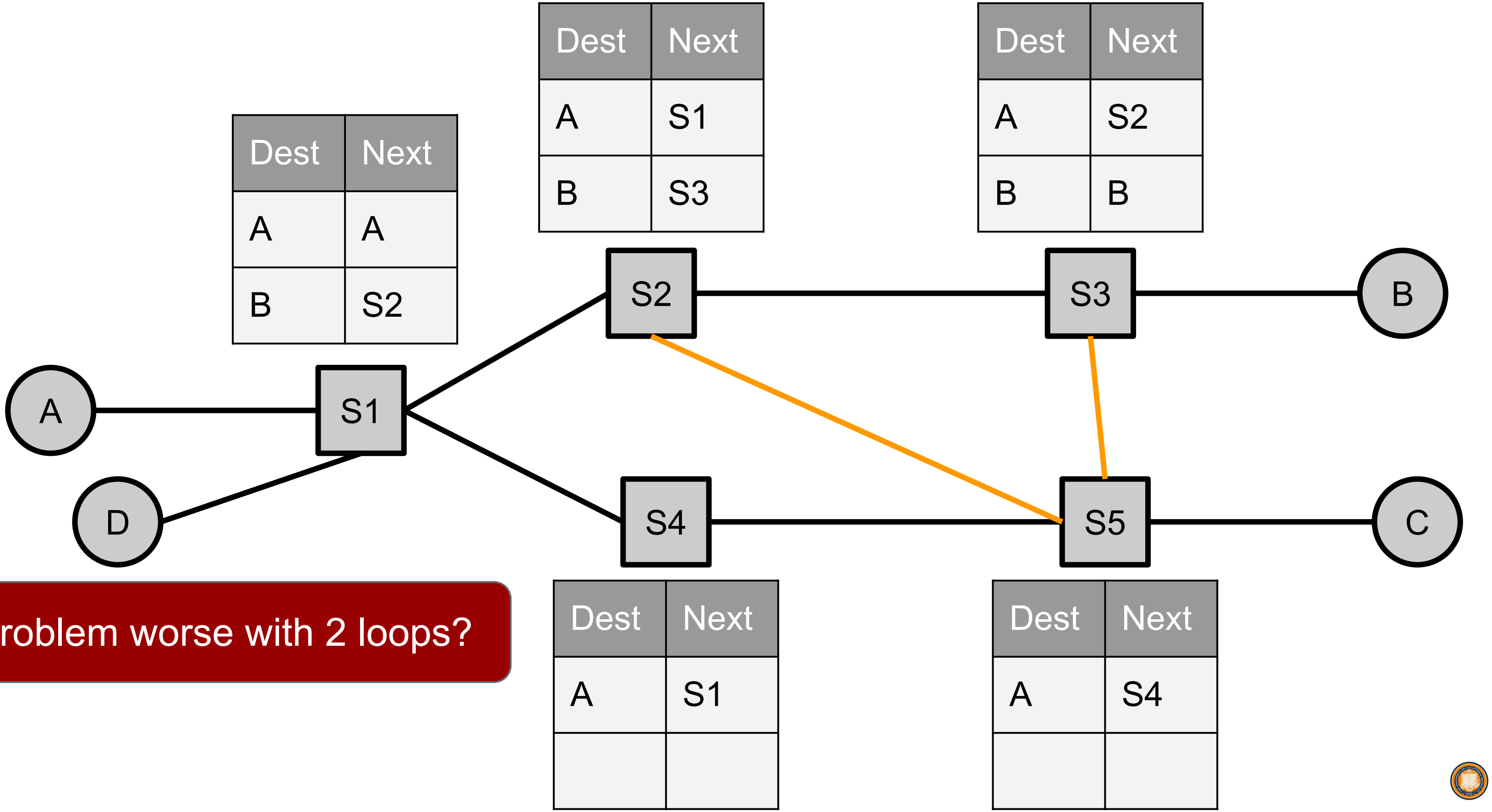
Dest	Next
A	S2
B	B

Dest	Next
A	A
B	S2

Dest	Next
A	S1

Dest	Next
A	S4

Learning Switches bad news: Loops!



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

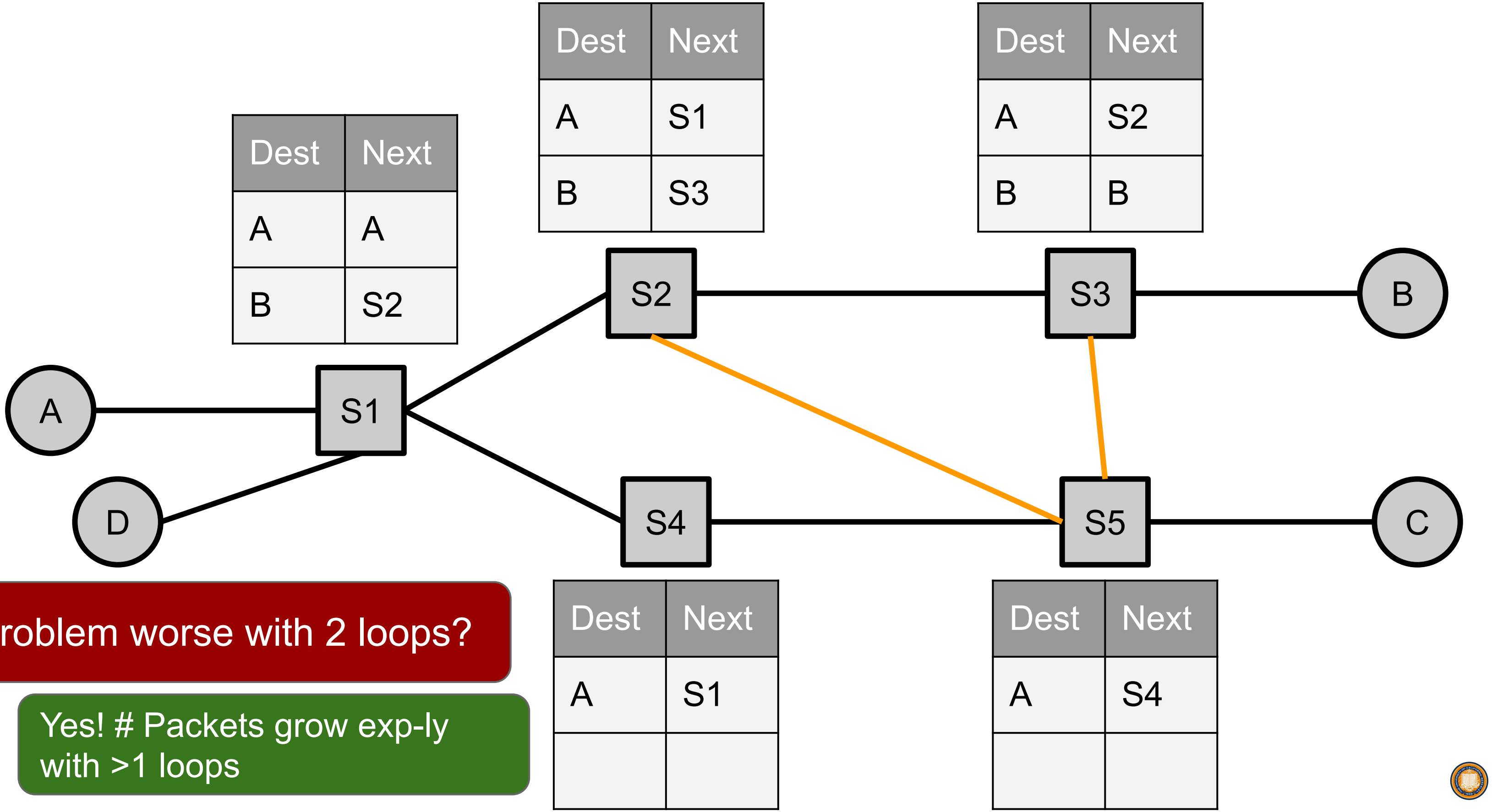
Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

Is the problem worse with 2 loops?

Learning Switches bad news: Loops!



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

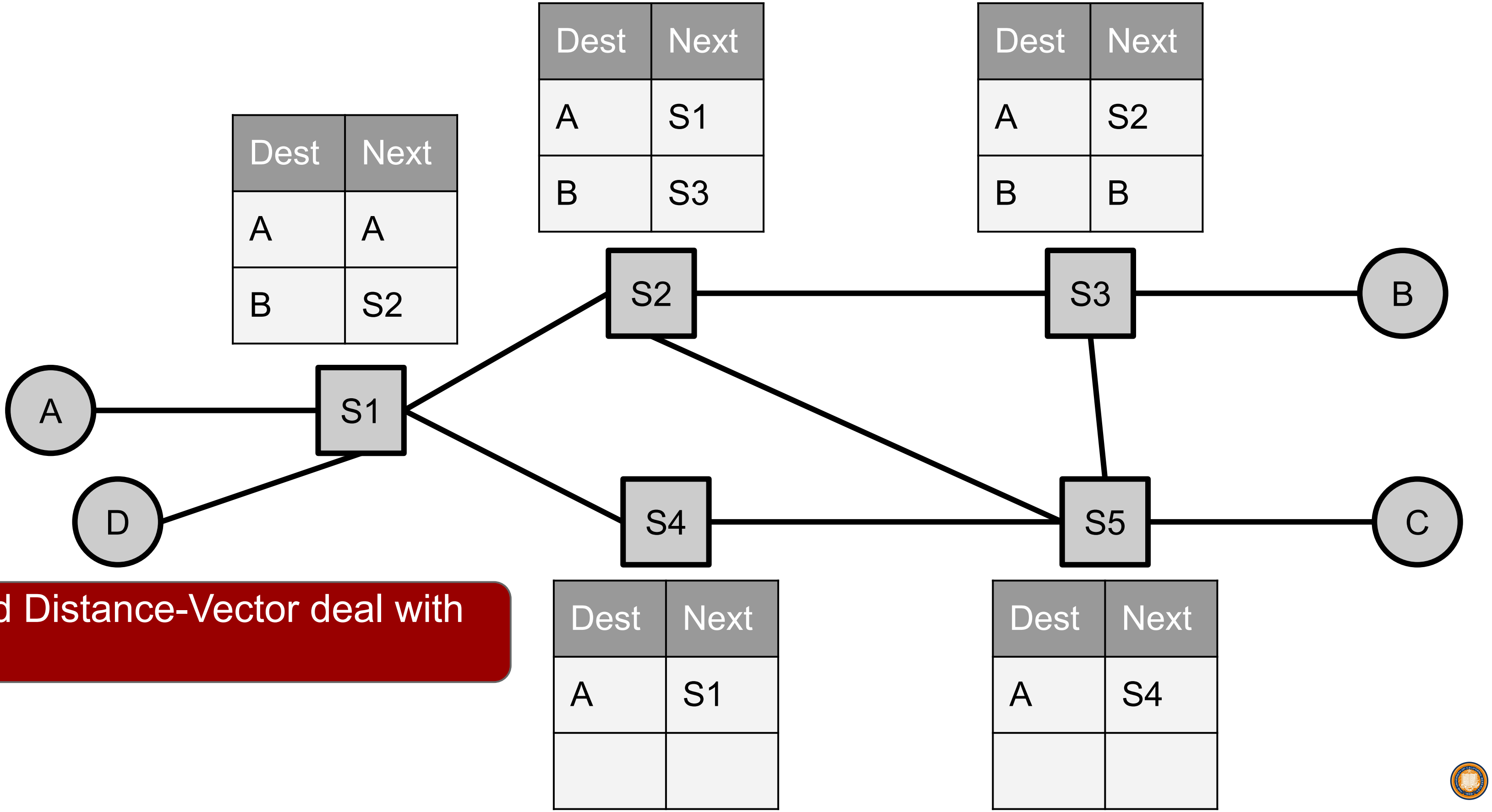
Dest	Next
A	S1

Dest	Next
A	S4

Is the problem worse with 2 loops?

Yes! # Packets grow exp-ly with >1 loops

Learning Switches bad news: Loops!



S1:

Dest	Next
A	A
B	S2

S2:

Dest	Next
A	S1
B	S3

S3:

Dest	Next
A	S2
B	B

S4:

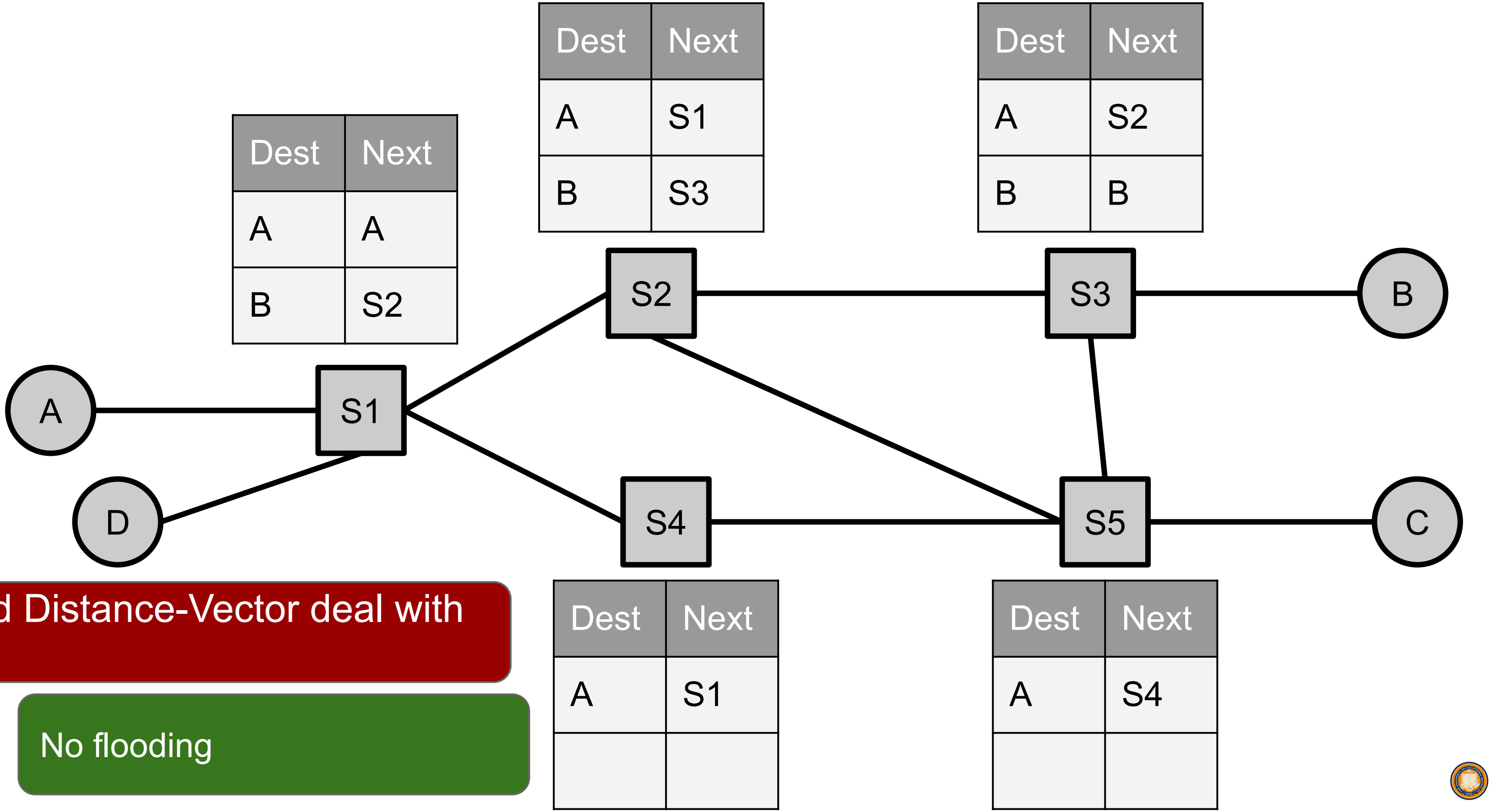
Dest	Next
A	S1

S5:

Dest	Next
A	S4

How did Distance-Vector deal with loops?

Learning Switches bad news: Loops!



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

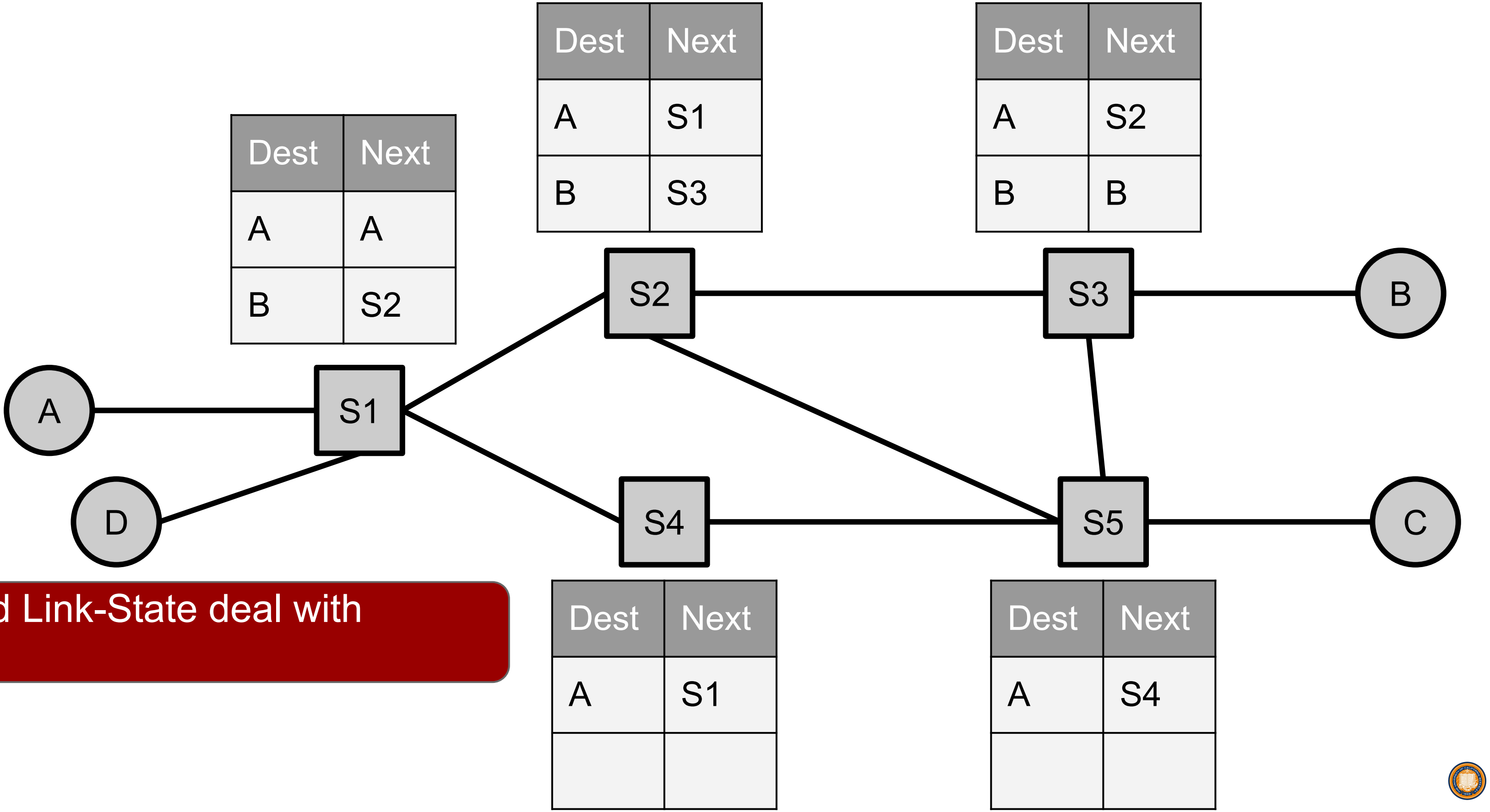
Dest	Next
A	S1

Dest	Next
A	S4

How did Distance-Vector deal with loops?

No flooding

Learning Switches bad news: Loops!



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

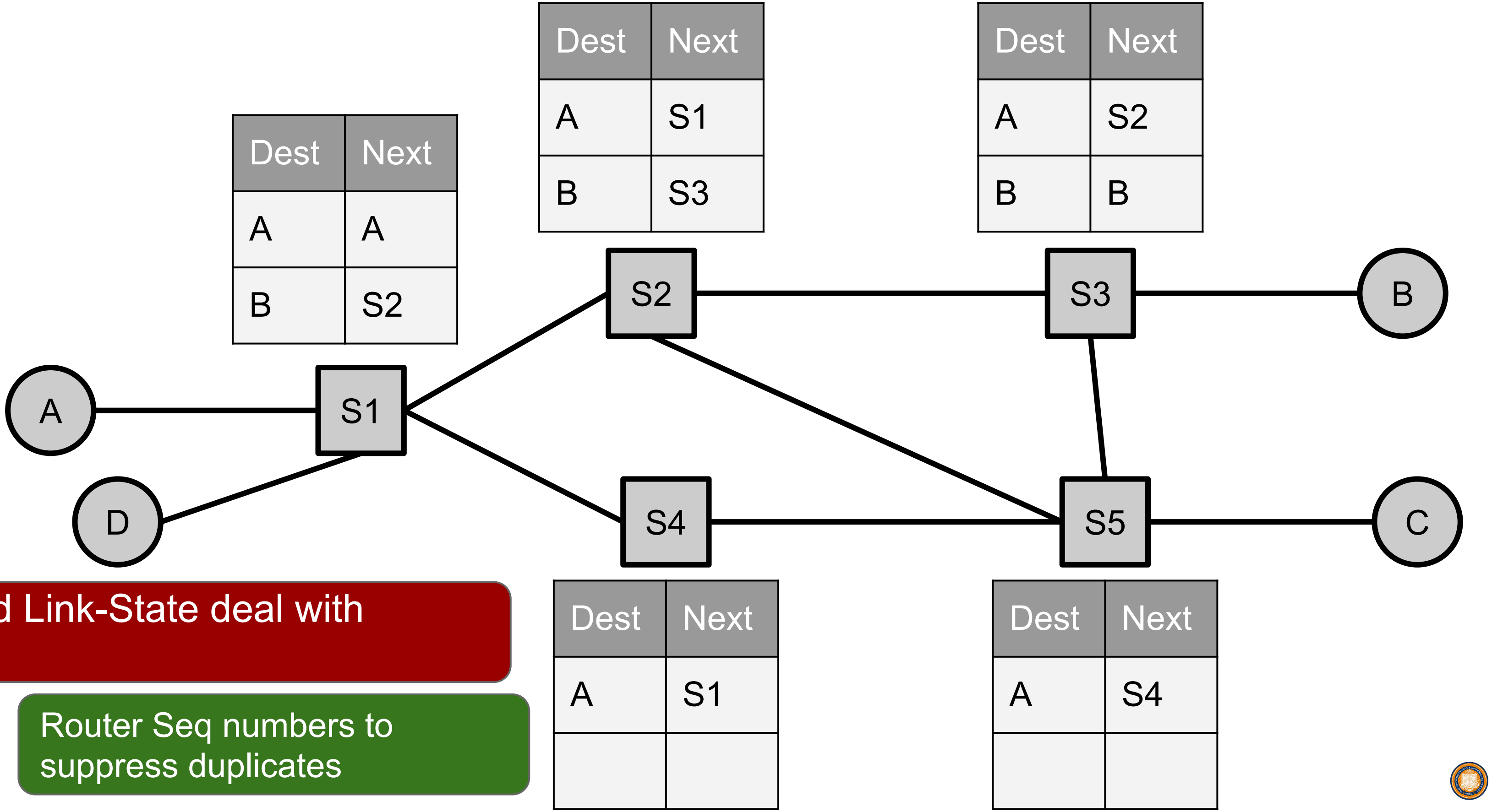
Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

How did Link-State deal with loops?

Learning Switches bad news: Loops!



Dest	Next
A	A
B	S2

Dest	Next
A	S1
B	S3

Dest	Next
A	S2
B	B

Dest	Next
A	S1

Dest	Next
A	S4

How did Link-State deal with loops?

Router Seq numbers to suppress duplicates

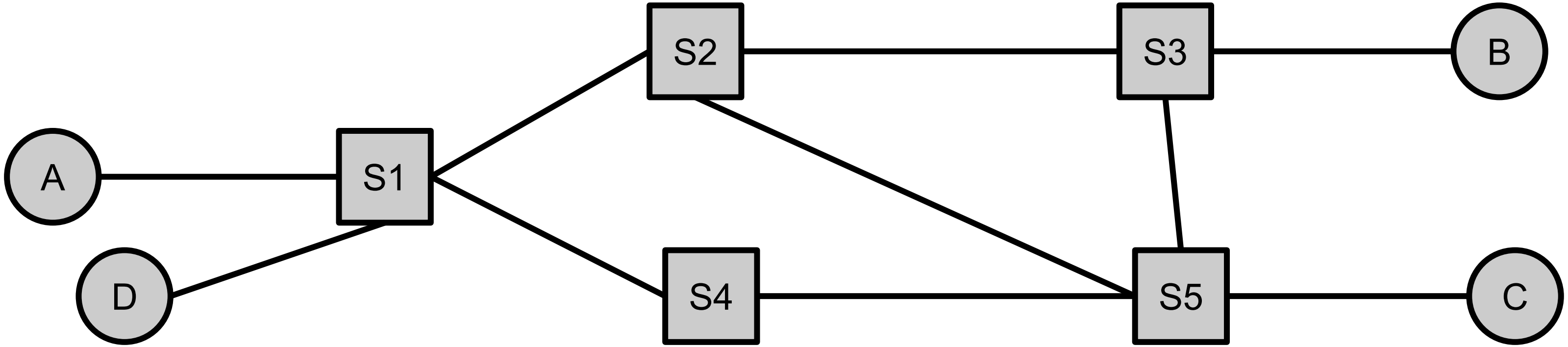
Spanning Tree Protocol (STP)

So far:

- Assumed underlying topology is a tree so it is safe to flood
- STP constructs a loop-free forwarding tree on top of loopy topology
  - Every node starts thinking it is root
  - Advertize (my_id, my_root, dist_to_root) to neighbors

Spanning Tree Protocol

Underlying topology, not good for forwarding in Learning Switches because of loops



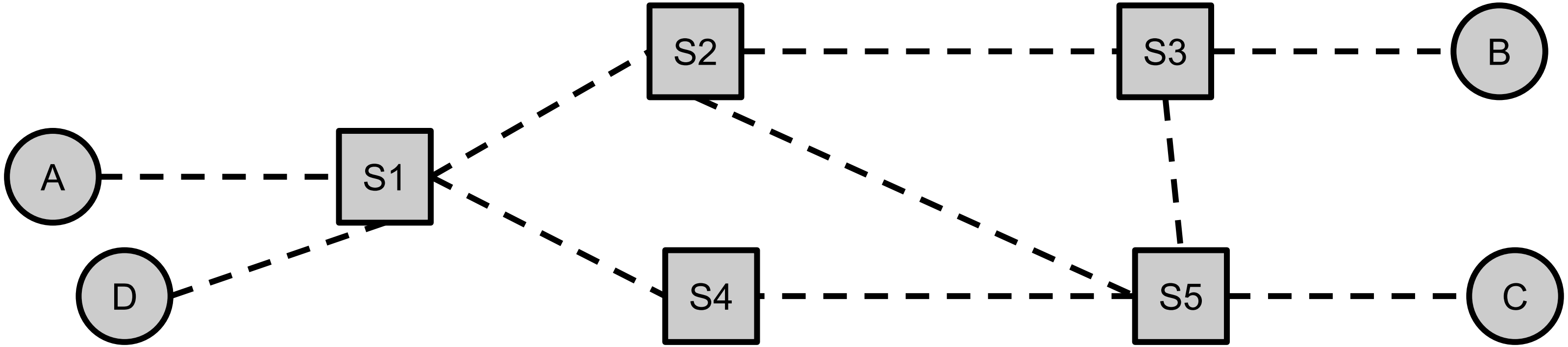
Spanning Tree Protocol

Legend

━━━━ Part of spanning tree

━━━━ Not part of spanning tree

- - - - Unknown



Spanning Tree Protocol

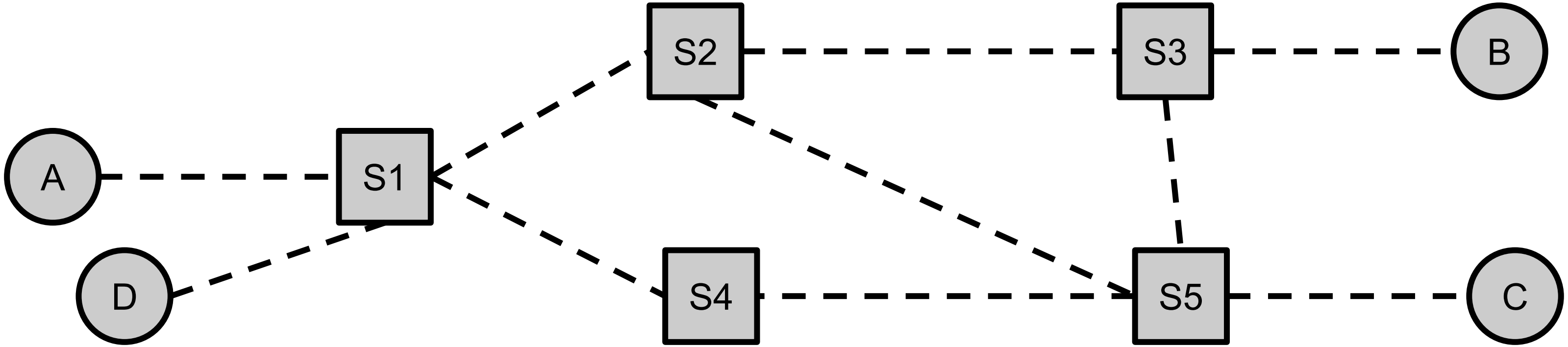
Legend

━━━ Part of spanning tree

🟥 Not part of spanning tree

- - - Unknown

Goal: construct a tree for flooding



Spanning Tree Protocol

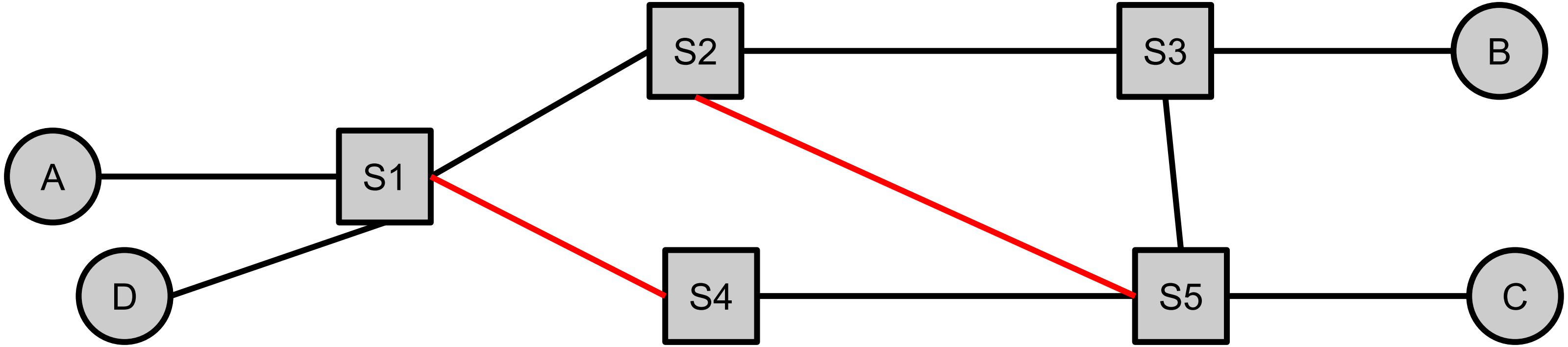
Legend

━━ Part of spanning tree

🟥 Not part of spanning tree

- - - Unknown

Goal: construct a tree for flooding



Valid for forwarding?

Spanning Tree Protocol

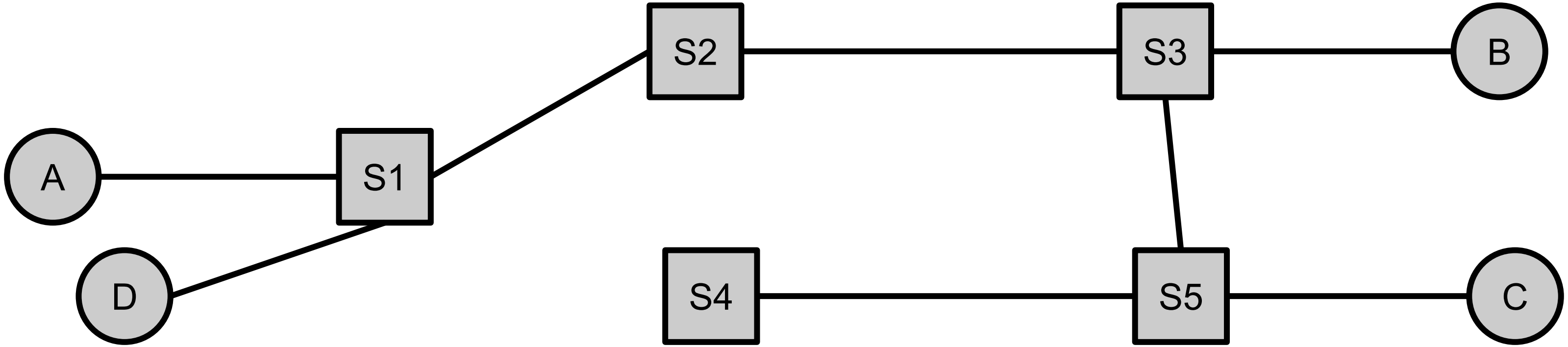
Legend

━━━ Part of spanning tree

━━━ Not part of spanning tree

- - - - Unknown

Goal: construct a tree for flooding



Valid for forwarding?

Spanning Tree Protocol

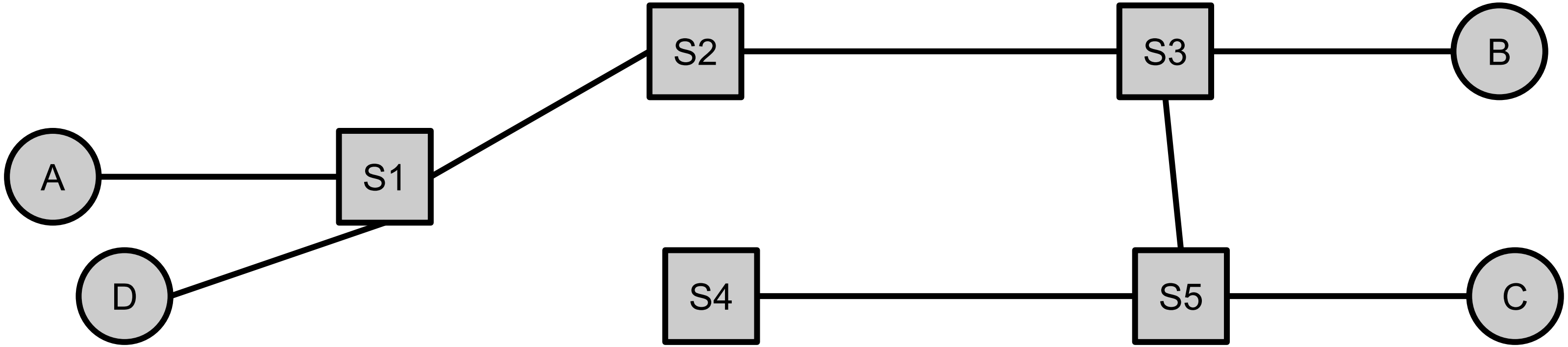
Legend

━━━ Part of spanning tree

<span>━━━</span> Not part of spanning tree

╍╍╍ Unknown

Goal: construct a tree for flooding



Valid for forwarding?

Yes! No loops

Spanning Tree Protocol

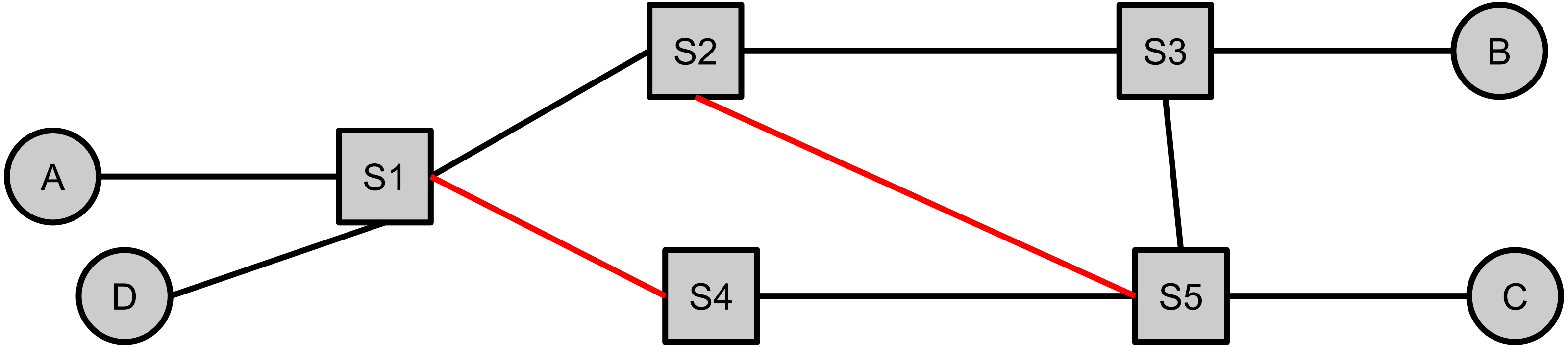
Legend

━━━ Part of spanning tree

━━━ Not part of spanning tree

- - - Unknown

Goal: construct a tree for flooding



Would STP ever produce this solution?

No. S1 not root, wrong tie breaking, not shortest paths

Spanning Tree Protocol

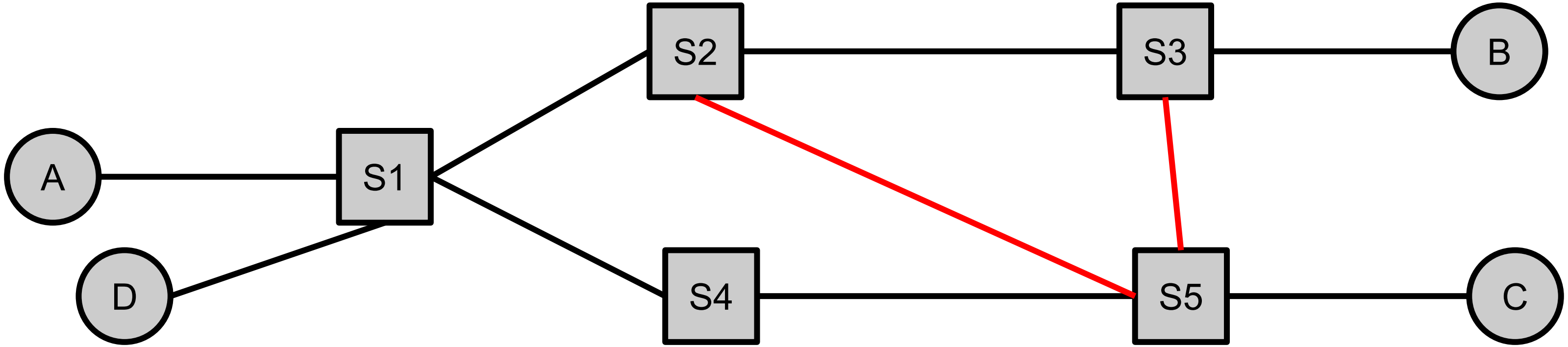
Legend

━━━ Part of spanning tree

🟥 Not part of spanning tree

╍╍╍ Unknown

Goal: construct a tree for flooding



Would STP ever produce this solution?

Spanning Tree Protocol

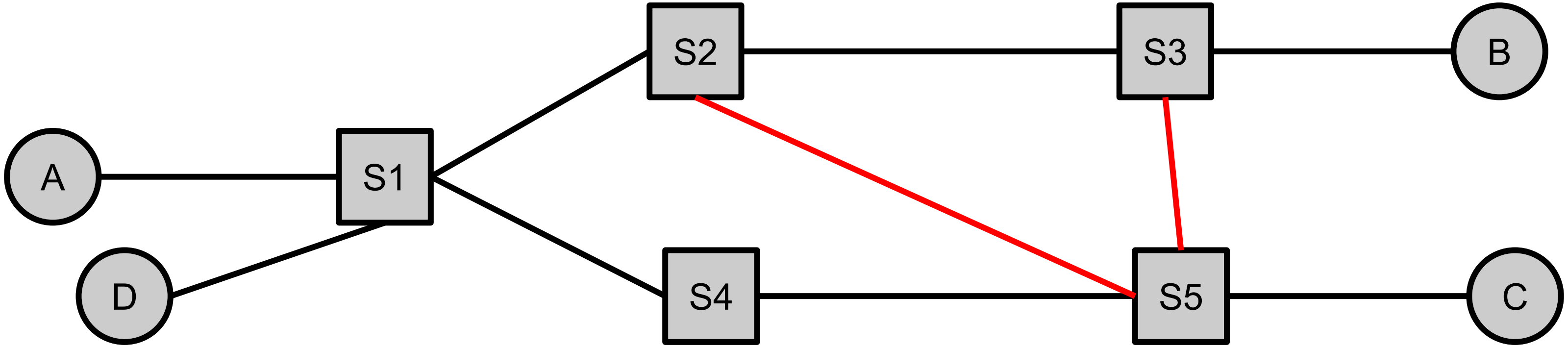
Legend

━━ Part of spanning tree

🟥 Not part of spanning tree

‐ ‐ ‐ Unknown

Goal: construct a tree for flooding



Would STP ever produce this solution?

✅ Shortest paths from all switches to S1

✅ S1 (lowest id) can be considered root

Spanning Tree Protocol

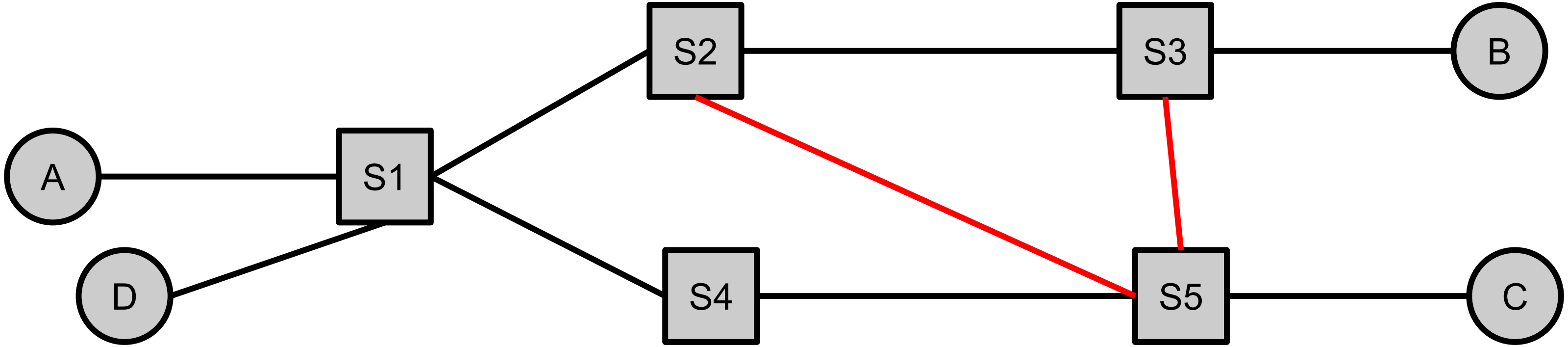
Legend

━━━ Part of spanning tree

━━━ Not part of spanning tree

— — — Unknown

Goal: construct a tree for flooding



Would STP ever produce this solution?

No. S5 would have preferred S5-S2 from S5-S4

✅ Shortest paths from all switches to S1

✅ S1 (lowest id) can be considered root

Spanning Tree Protocol

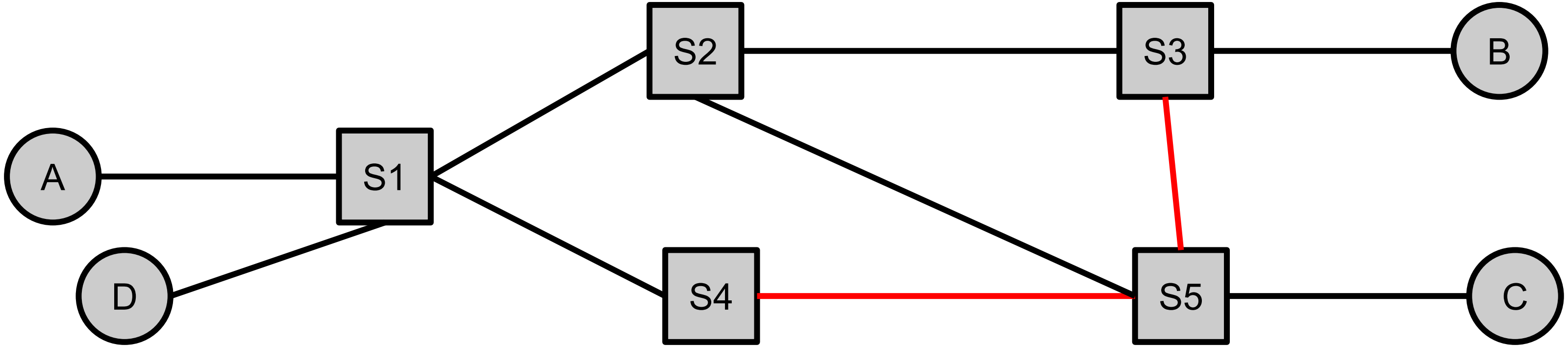
Legend

━━━ Part of spanning tree

🟥 Not part of spanning tree

- - - Unknown

Goal: construct a tree for flooding



Would STP ever produce this solution?

Spanning Tree Protocol

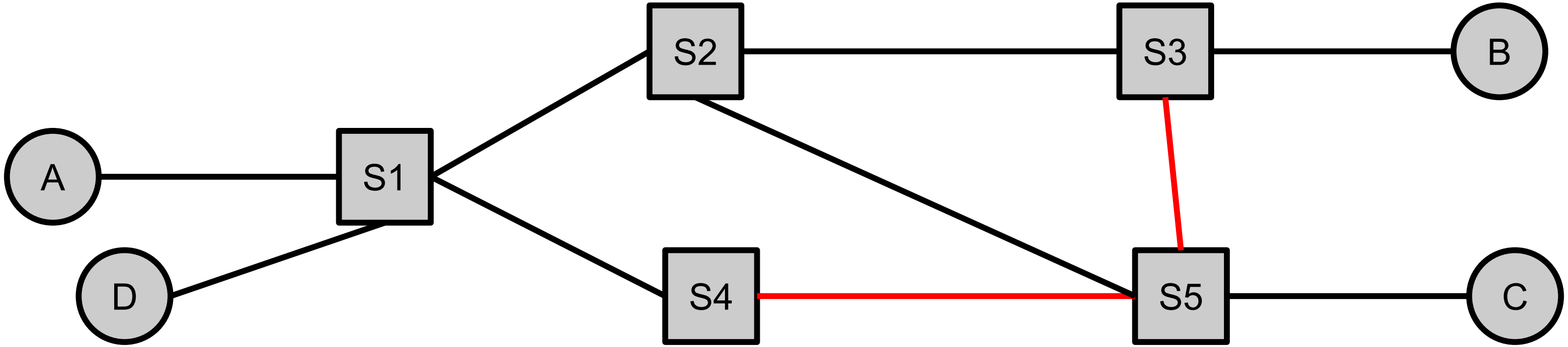
Legend

▬ Part of spanning tree

🟥 Not part of spanning tree

- - - Unknown

Goal: construct a tree for flooding



Would STP ever produce this solution?

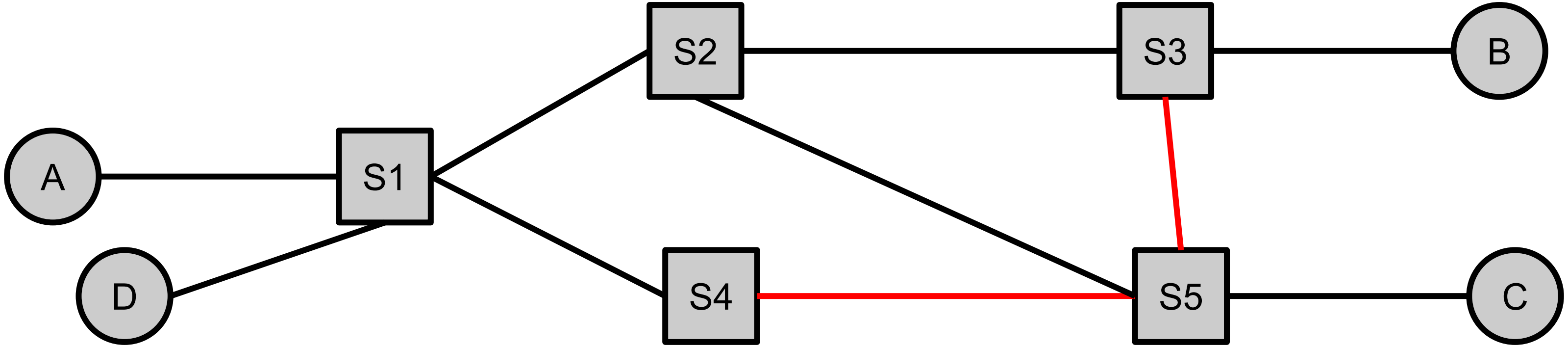
Yes !!

Spanning Tree Protocol

Legend

━━━	Part of spanning tree
🟥	Not part of spanning tree
- - -	Unknown

Goal: construct a tree for flooding



Is this a unique equilibrium for STP or can there be others?

Spanning Tree Protocol

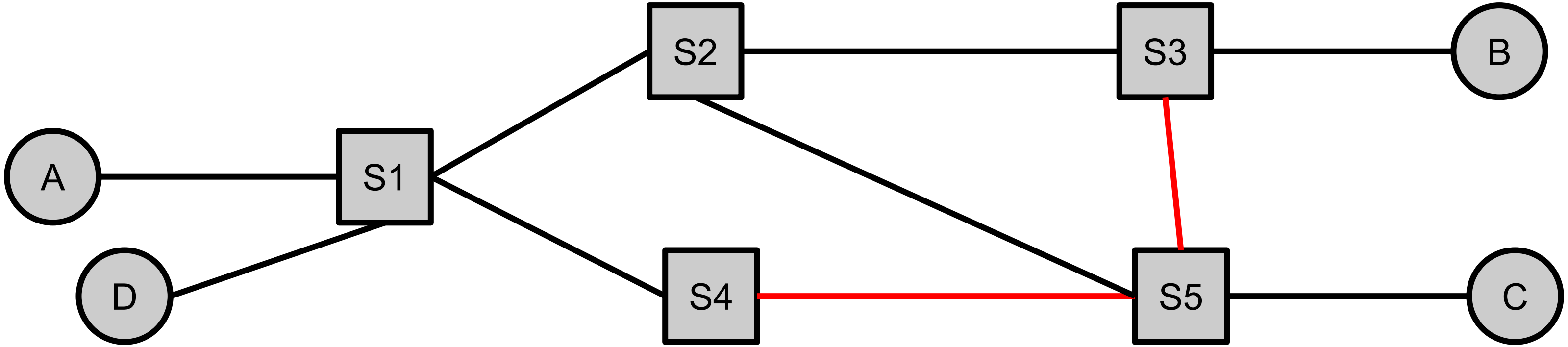
Legend

━━━ Part of spanning tree

━━━ Not part of spanning tree

– – – Unknown

Goal: construct a tree for flooding



Is this a unique equilibrium for STP or can there be others?

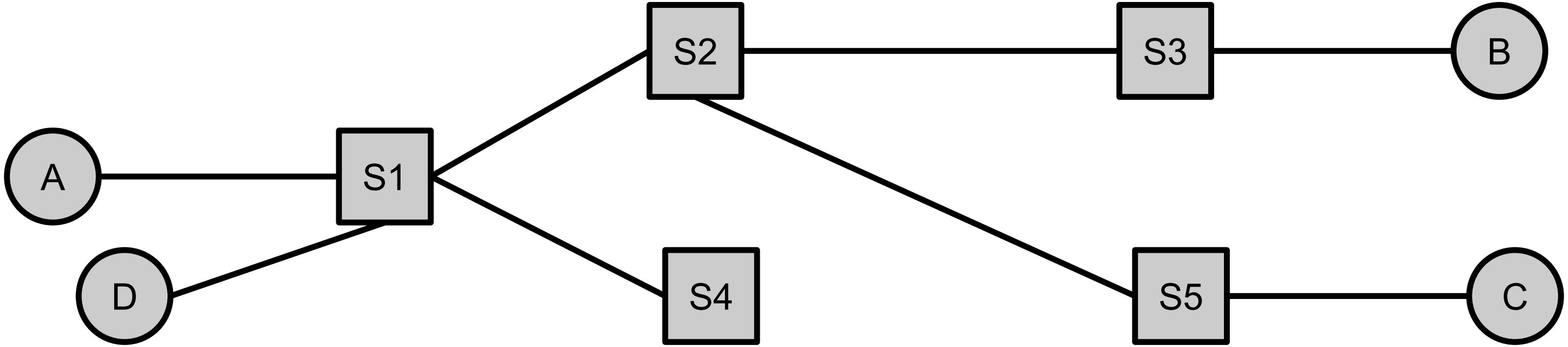
It is unique

Spanning Tree Protocol

Legend

━━	Part of spanning tree
🟥	Not part of spanning tree
╍╍	Unknown

Goal: construct a tree for flooding



Is this a unique equilibrium for STP or can there be others?

It is unique

Let’s run STP step by step

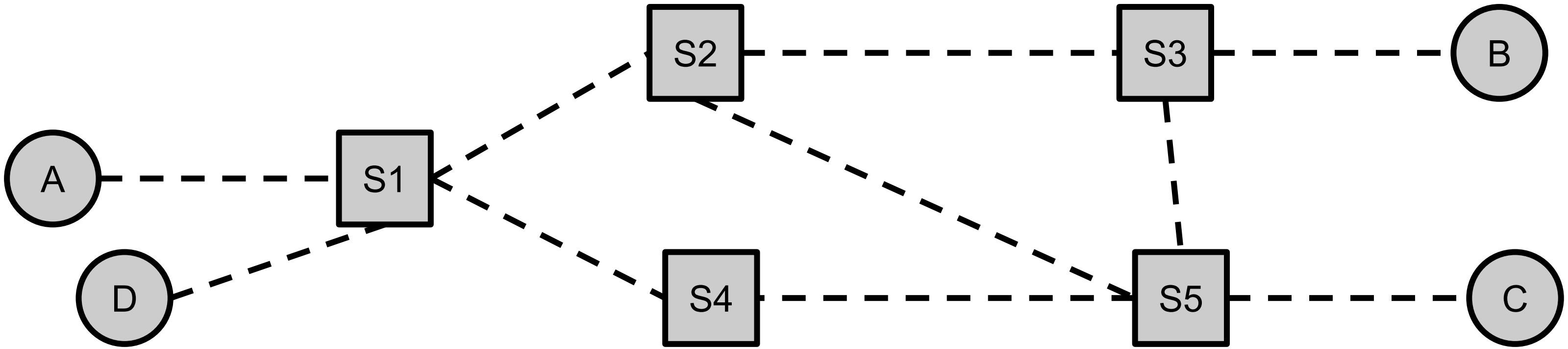
Spanning Tree Protocol

Legend

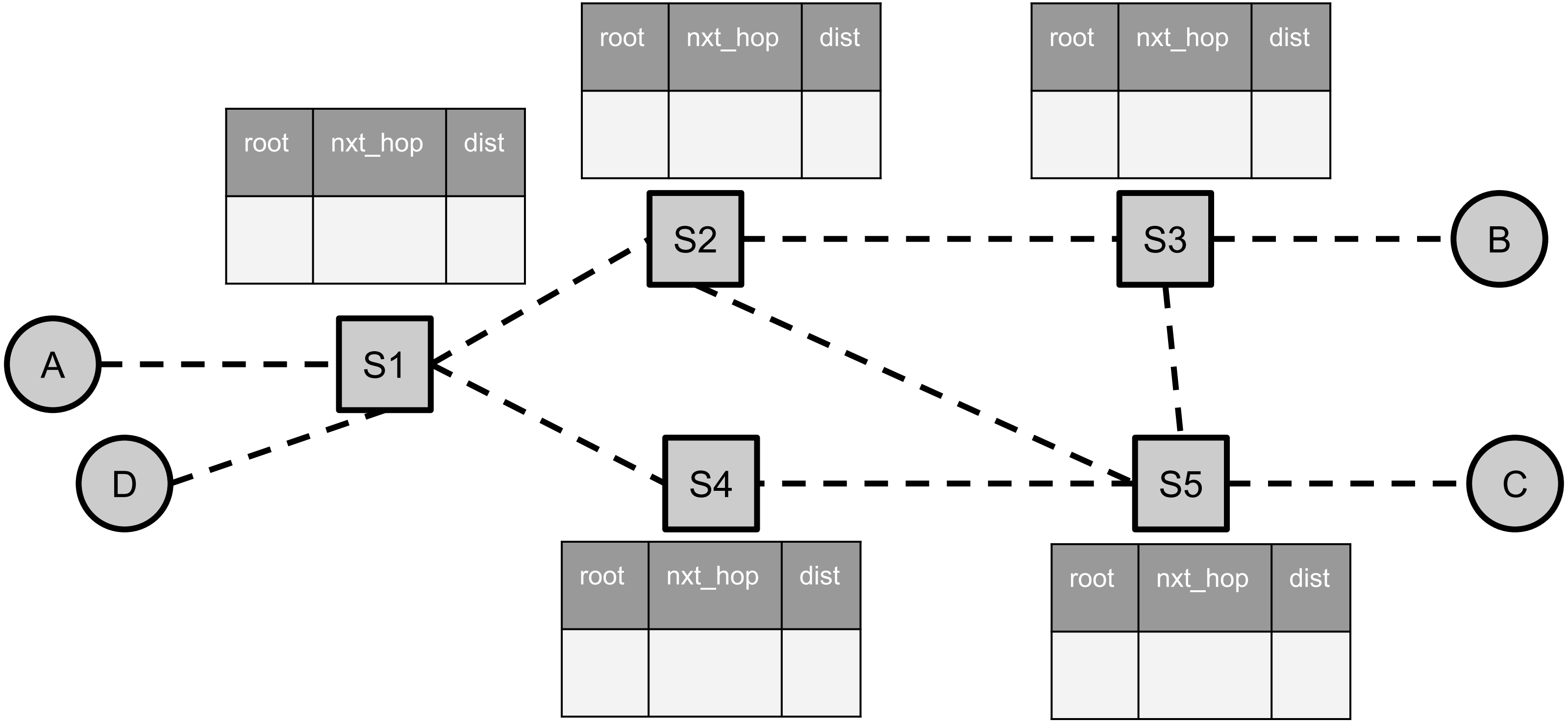
━━━ Part of spanning tree

━━━ Not part of spanning tree

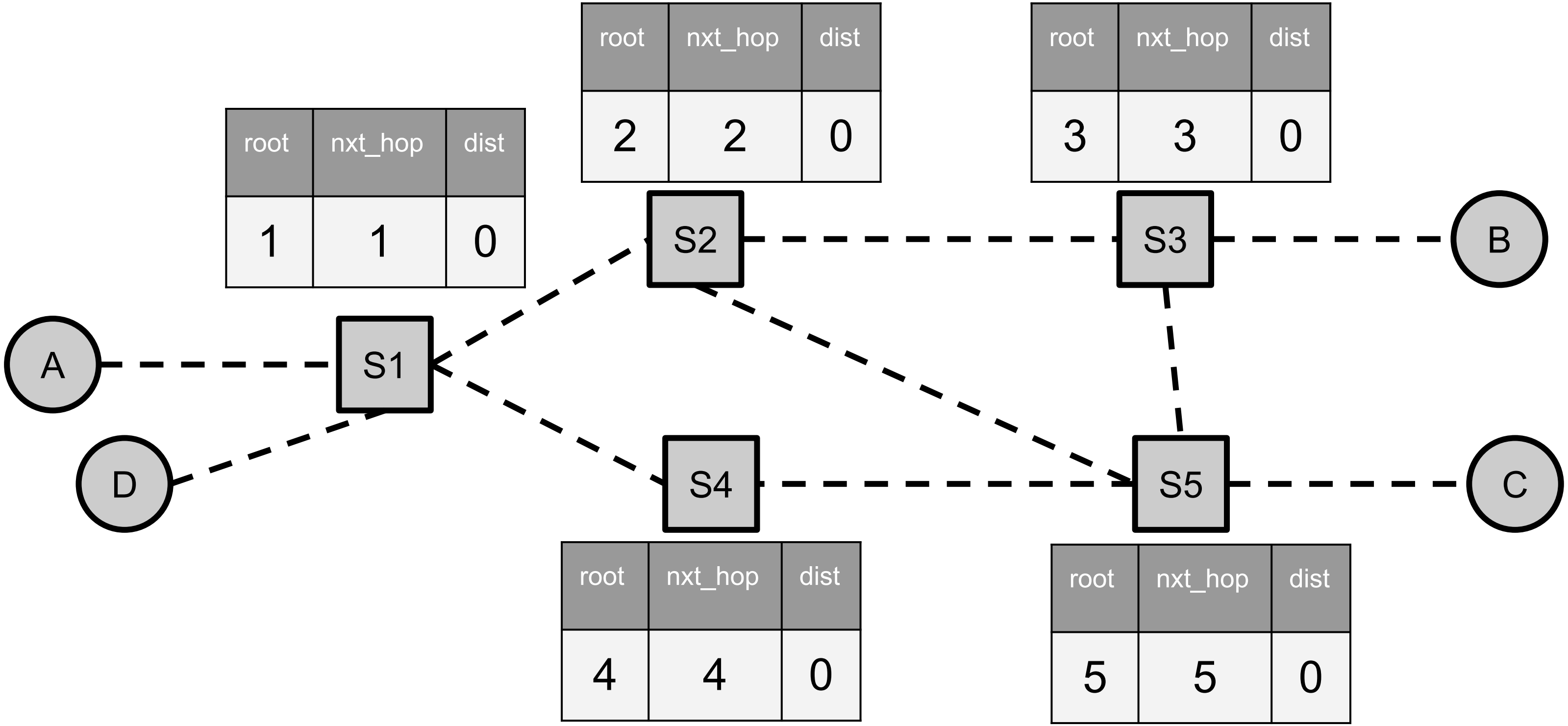
- - - Unknown



Spanning Tree Protocol



Spanning Tree Protocol



S1:

root	nxt_hop	dist
1	1	0

S2:

root	nxt_hop	dist
2	2	0

S3:

root	nxt_hop	dist
3	3	0

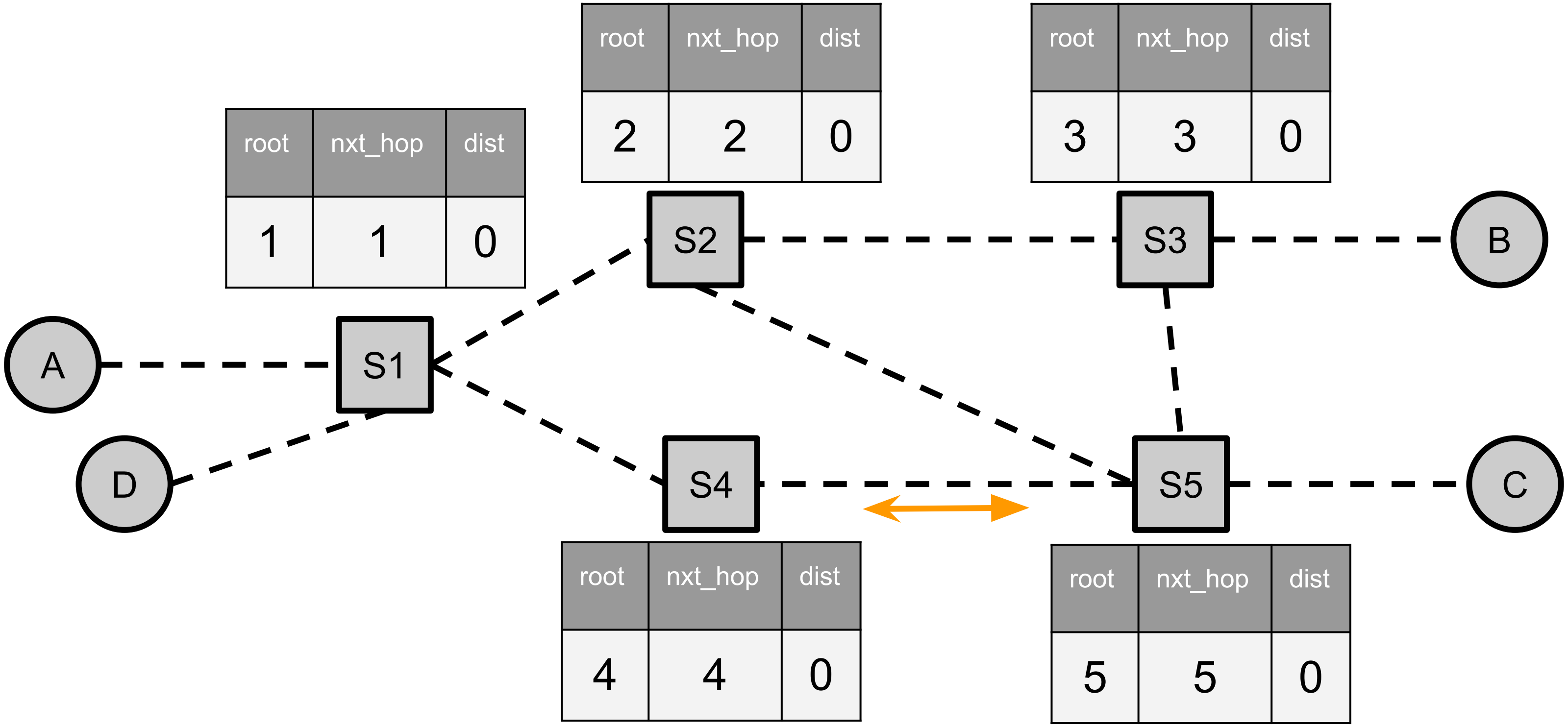
S4:

root	nxt_hop	dist
4	4	0

S5:

root	nxt_hop	dist
5	5	0

Spanning Tree Protocol



S1:

root	nxt_hop	dist
1	1	0

S2:

root	nxt_hop	dist
2	2	0

S3:

root	nxt_hop	dist
3	3	0

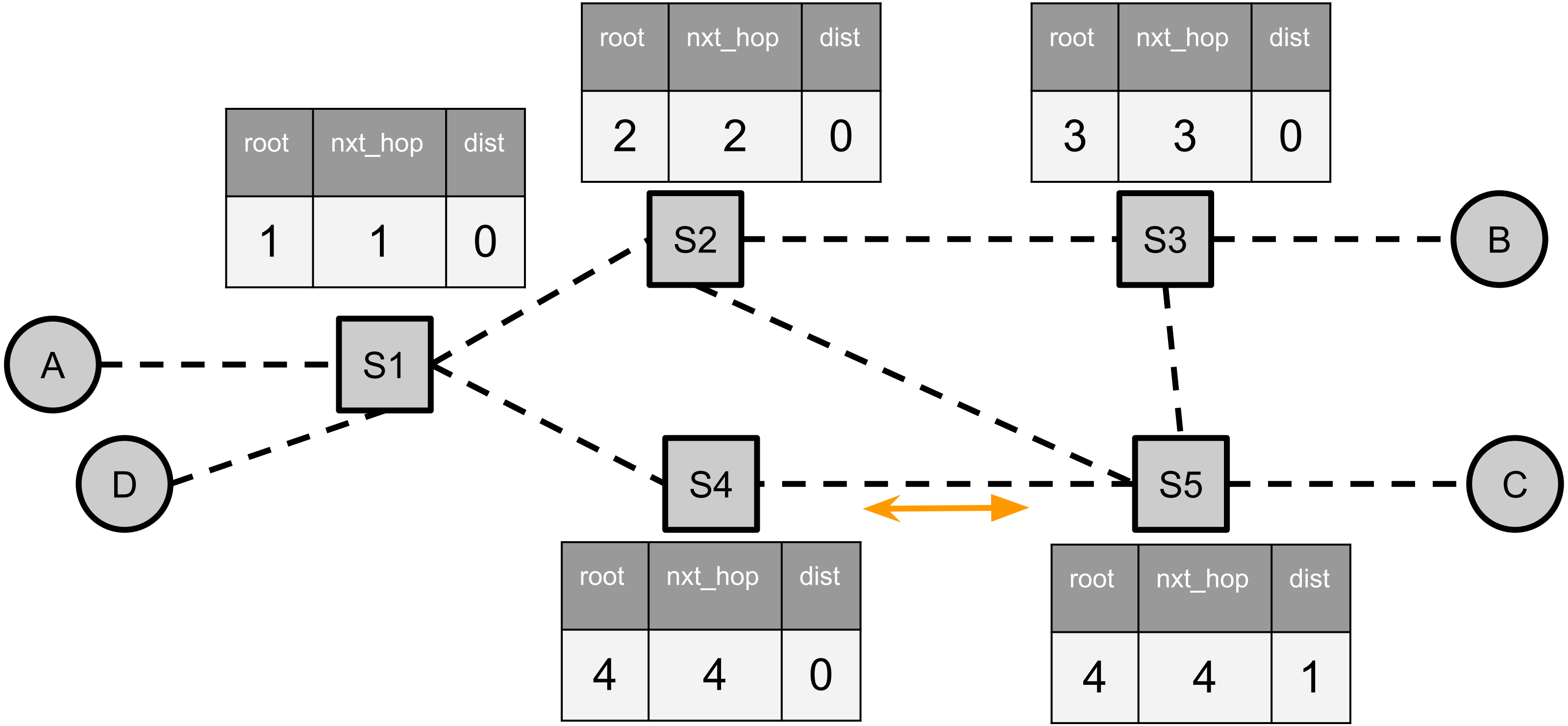
S4:

root	nxt_hop	dist
4	4	0

S5:

root	nxt_hop	dist
5	5	0

Spanning Tree Protocol



root	nxt_hop	dist
1	1	0

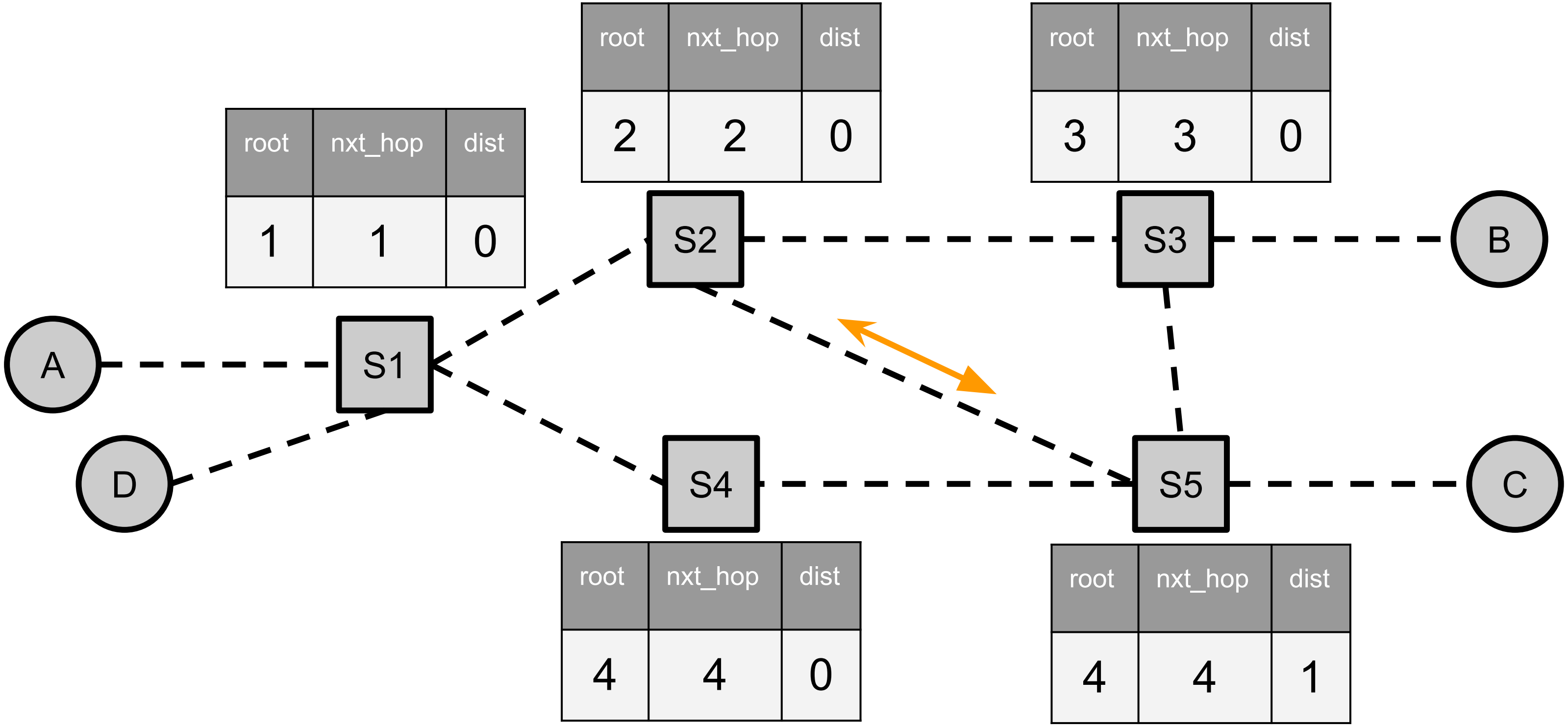
root	nxt_hop	dist
2	2	0

root	nxt_hop	dist
3	3	0

root	nxt_hop	dist
4	4	0

root	nxt_hop	dist
4	4	1

Spanning Tree Protocol



root	nxt_hop	dist
1	1	0

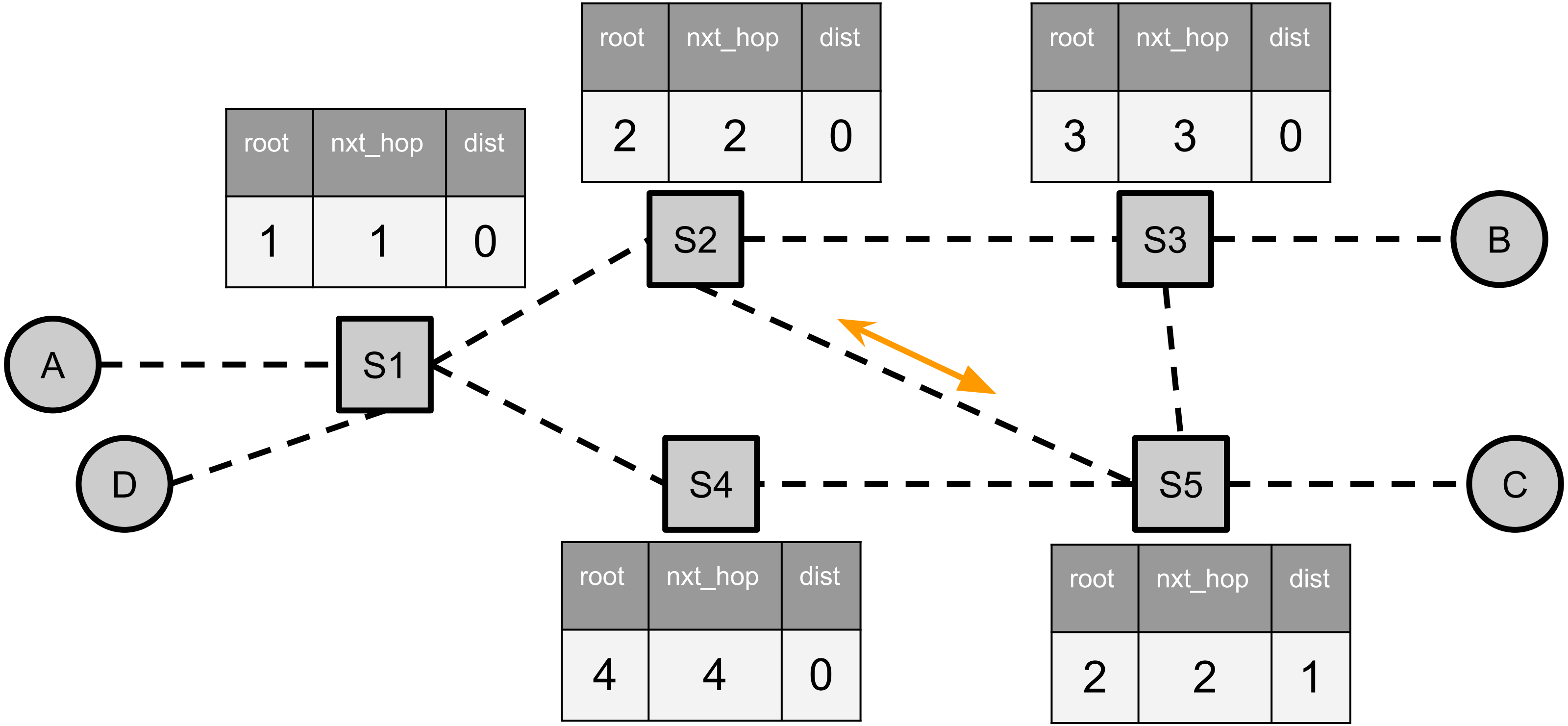
root	nxt_hop	dist
2	2	0

root	nxt_hop	dist
3	3	0

root	nxt_hop	dist
4	4	0

root	nxt_hop	dist
4	4	1

Spanning Tree Protocol



root	nxt_hop	dist
1	1	0

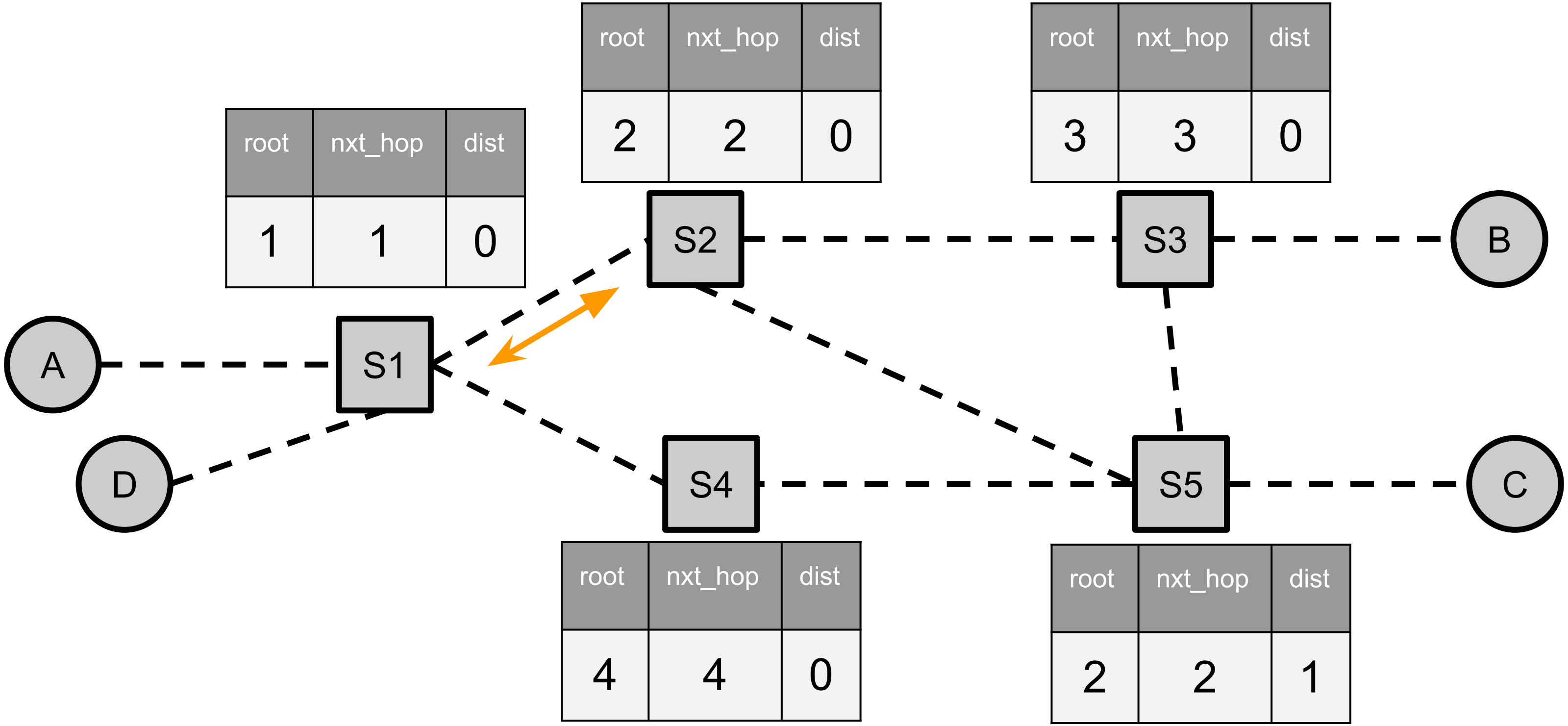
root	nxt_hop	dist
2	2	0

root	nxt_hop	dist
3	3	0

root	nxt_hop	dist
4	4	0

root	nxt_hop	dist
2	2	1

Spanning Tree Protocol



root	nxt_hop	dist
1	1	0

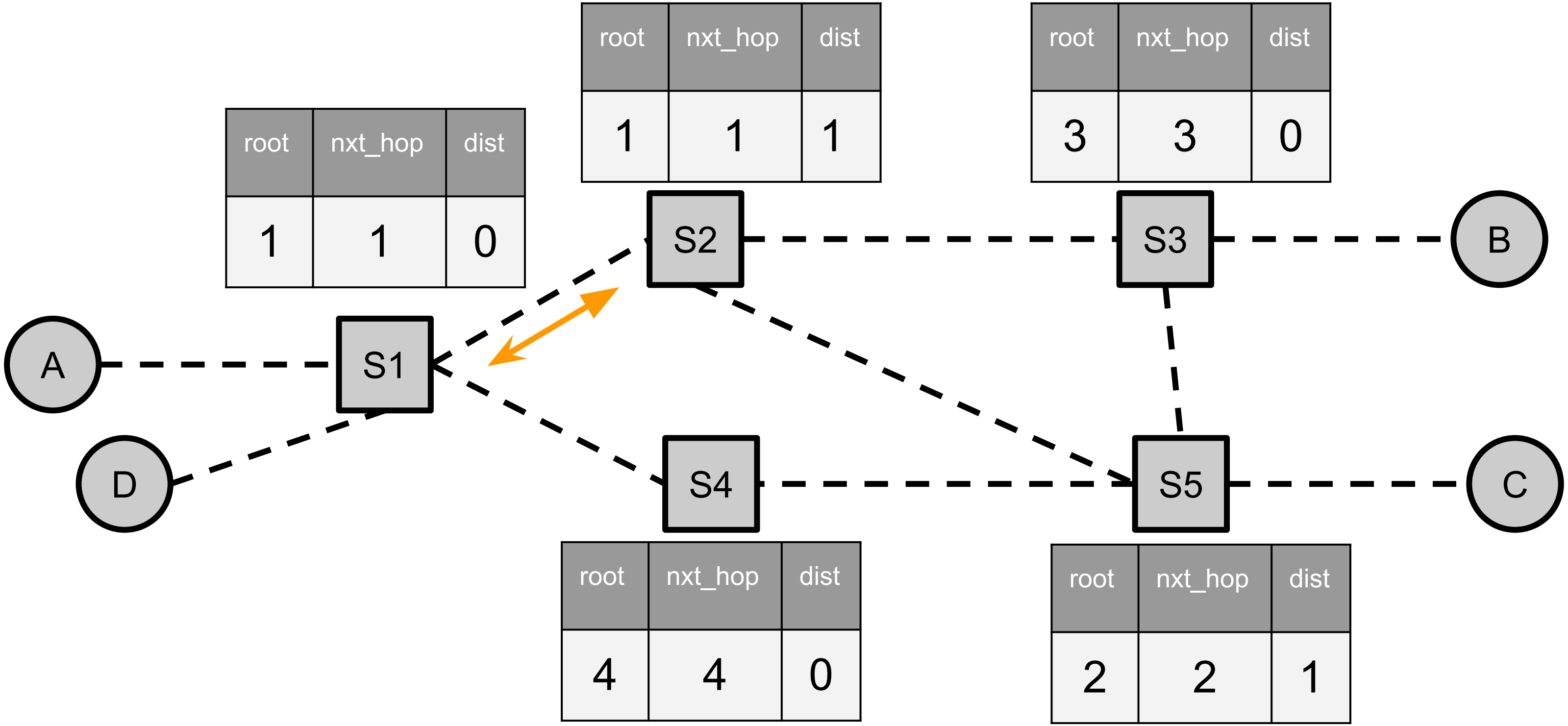
root	nxt_hop	dist
2	2	0

root	nxt_hop	dist
3	3	0

root	nxt_hop	dist
4	4	0

root	nxt_hop	dist
2	2	1

Spanning Tree Protocol



root	nxt_hop	dist
1	1	1

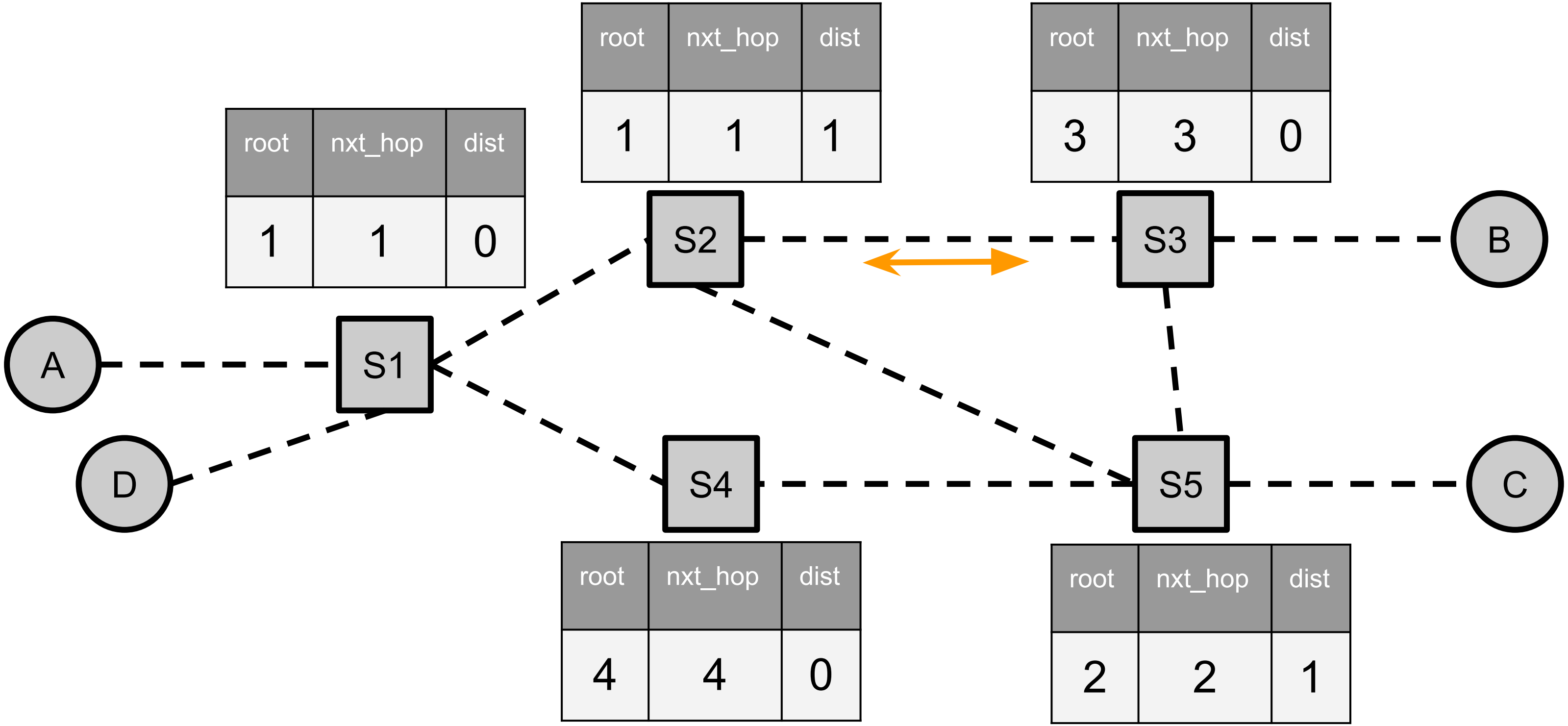
root	nxt_hop	dist
3	3	0

root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
4	4	0

root	nxt_hop	dist
2	2	1

Spanning Tree Protocol



root	nxt_hop	dist
1	1	1

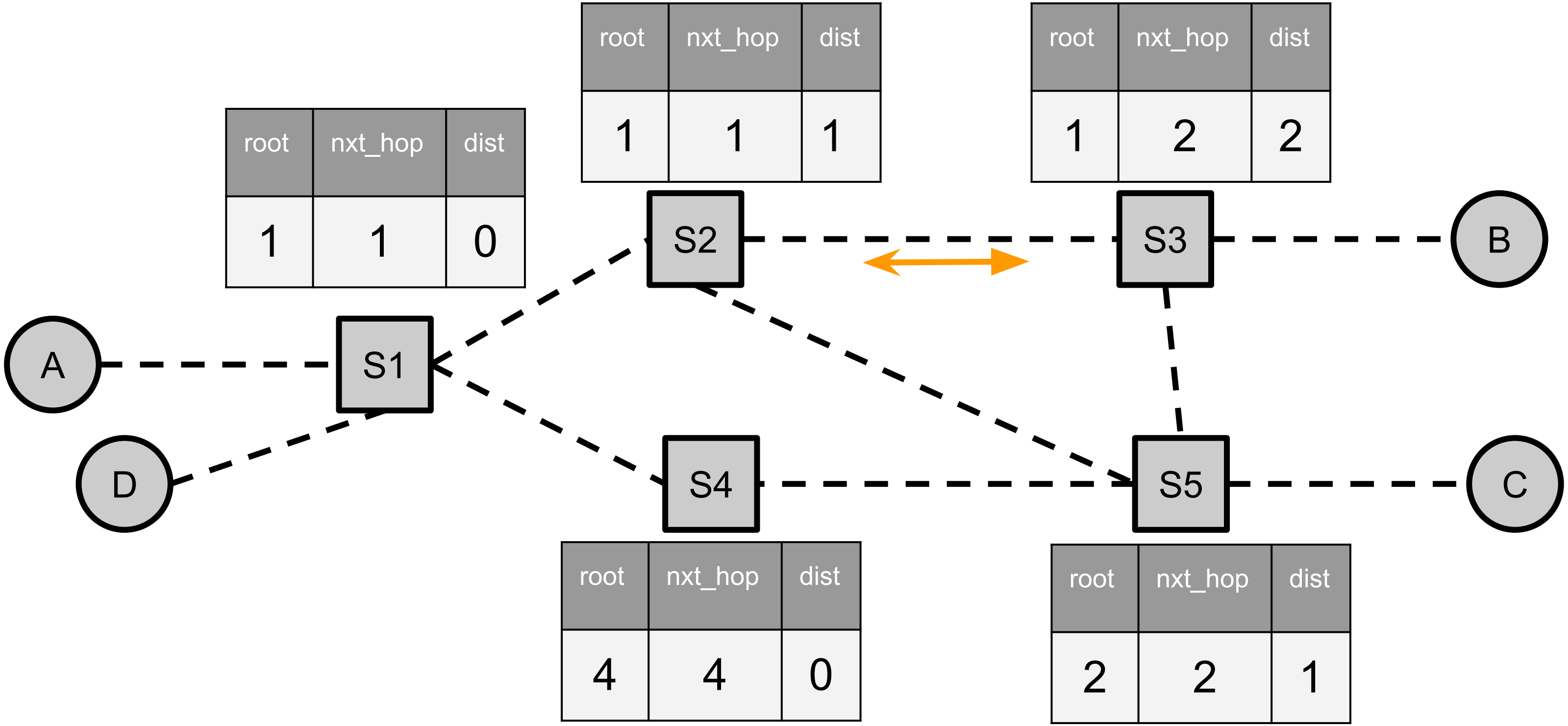
root	nxt_hop	dist
3	3	0

root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
4	4	0

root	nxt_hop	dist
2	2	1

Spanning Tree Protocol



root	nxt_hop	dist
1	1	0

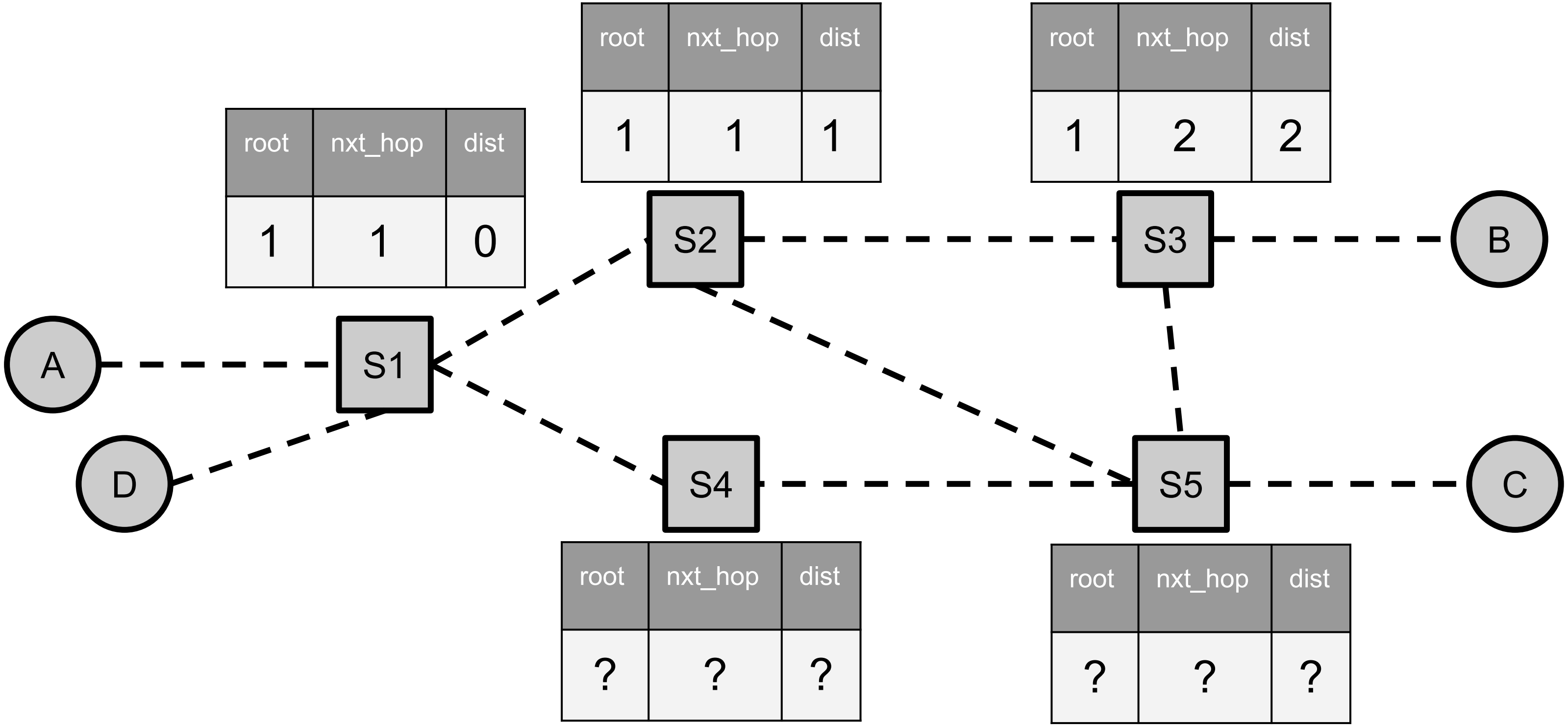
root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
4	4	0

root	nxt_hop	dist
2	2	1

Spanning Tree Protocol



root	nxt_hop	dist
1	1	1

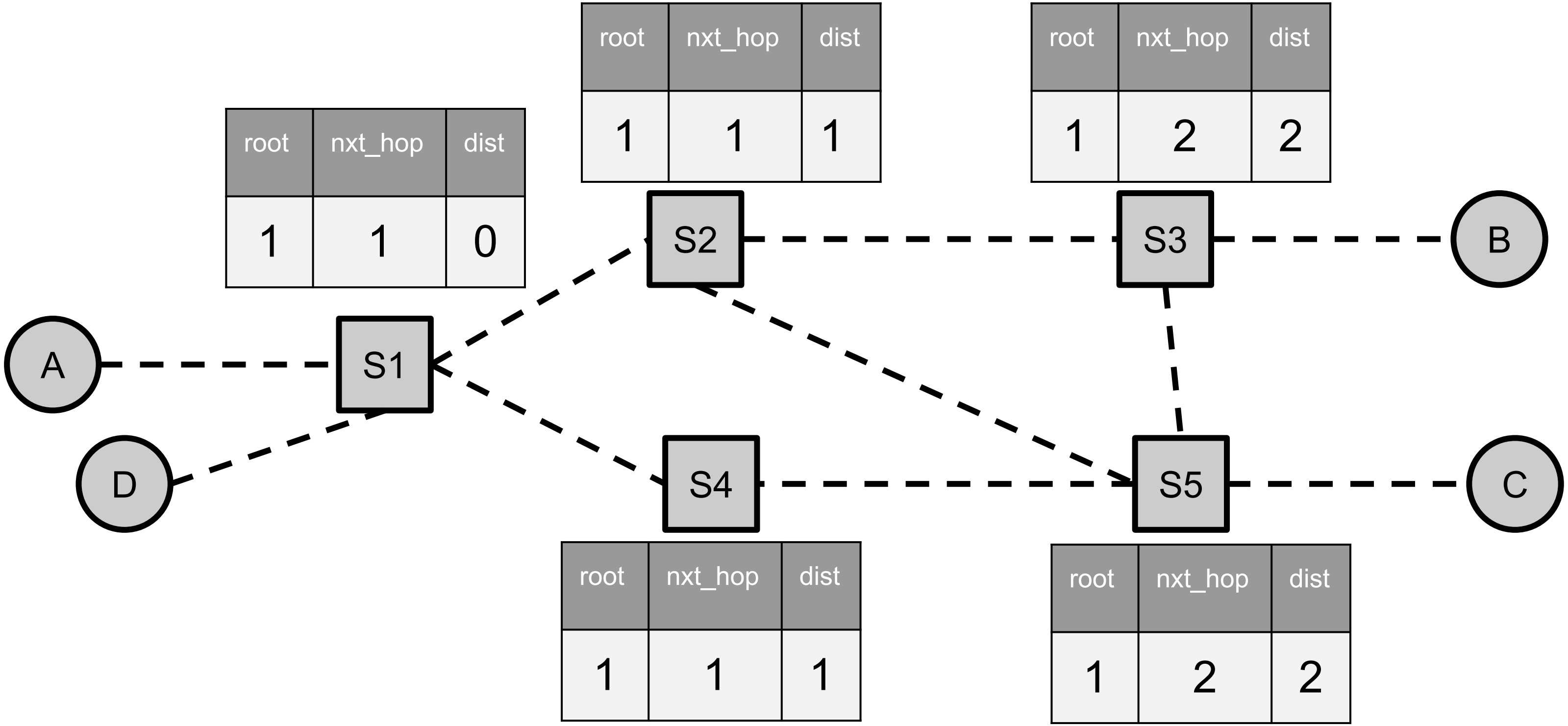
root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
?	?	?

root	nxt_hop	dist
?	?	?

Spanning Tree Protocol



root	nxt_hop	dist
1	1	0

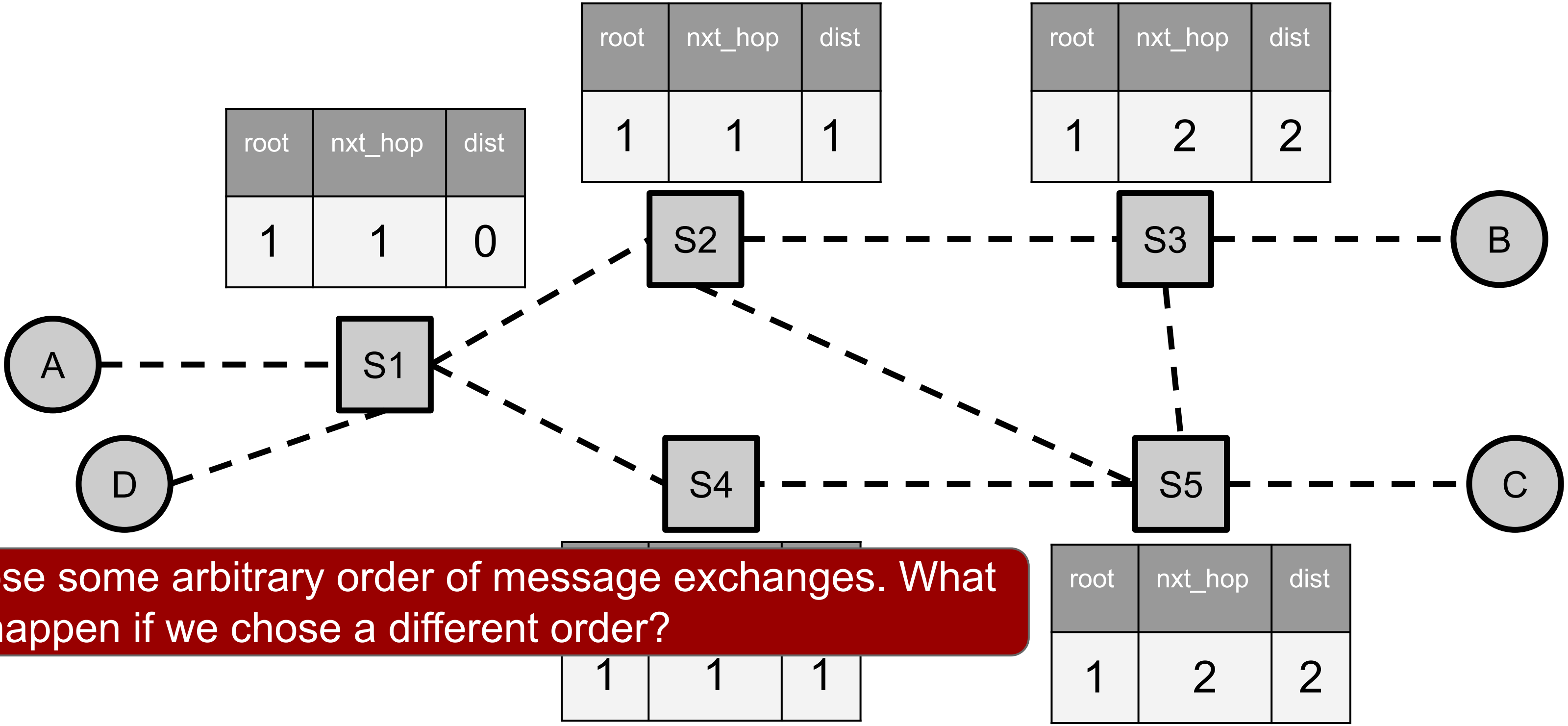
root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Spanning Tree Protocol



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

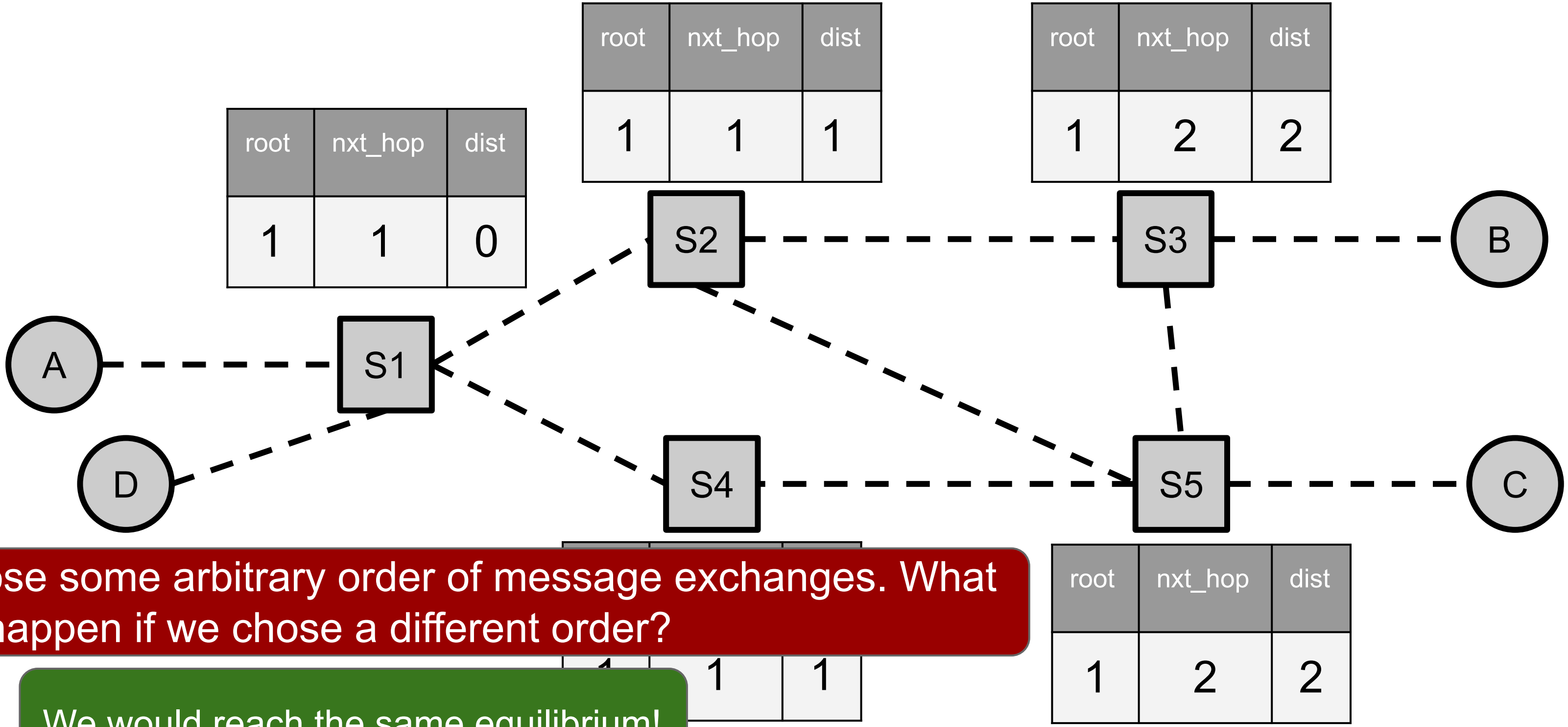
1	1	1

root	nxt_hop	dist
1	2	2

We chose some arbitrary order of message exchanges. What would happen if we chose a different order?

cs168.io

Spanning Tree Protocol



S1:

root	nxt_hop	dist
1	1	0

S2:

root	nxt_hop	dist
1	1	1

S3:

root	nxt_hop	dist
1	2	2

S4:

1	1	1

S5:

root	nxt_hop	dist
1	2	2

We chose some arbitrary order of message exchanges. What would happen if we chose a different order?

We would reach the same equilibrium!

Spanning Tree Protocol

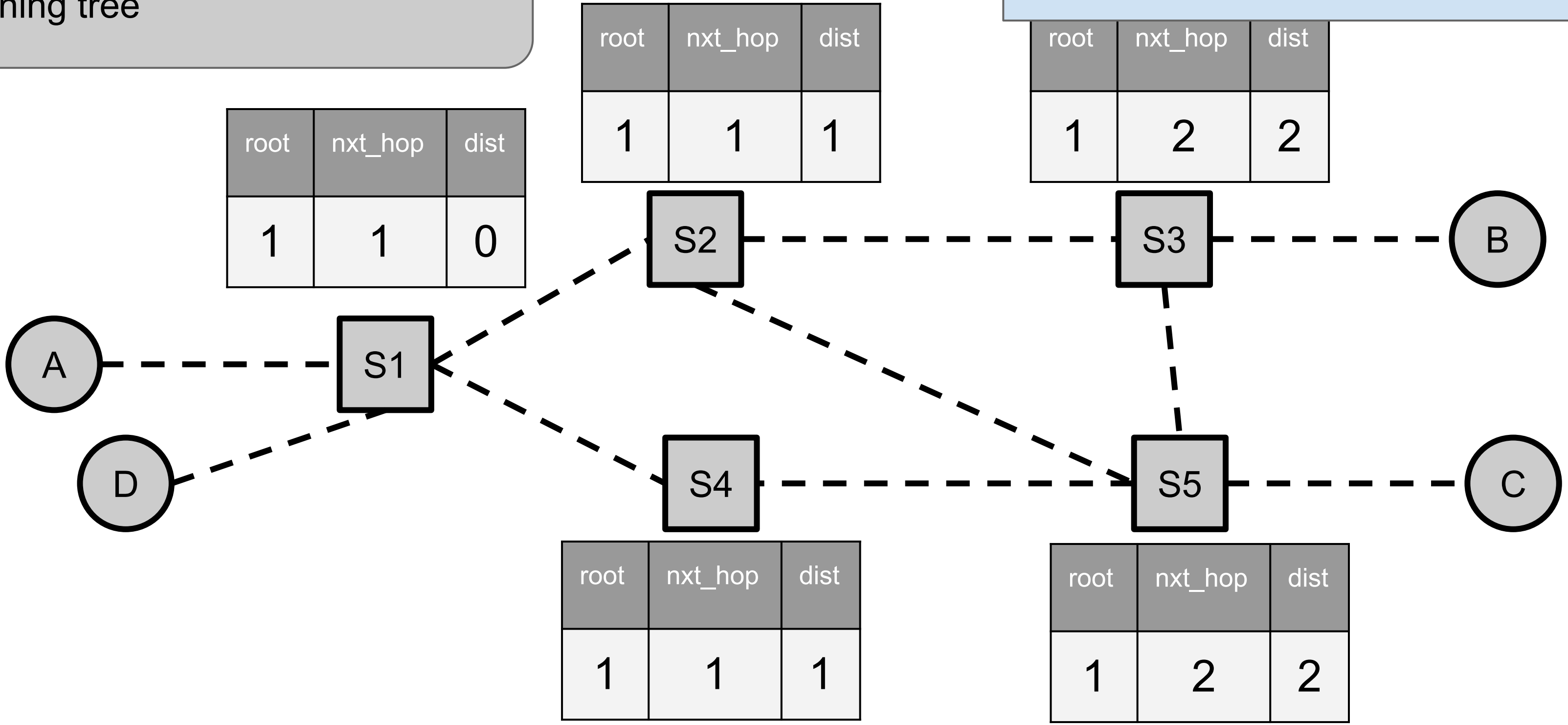
Step 2: Disable all links not on spanning tree

Legend

━ Part of spanning tree

🟥 Not part of spanning tree

╍ Unknown



root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Spanning Tree Protocol

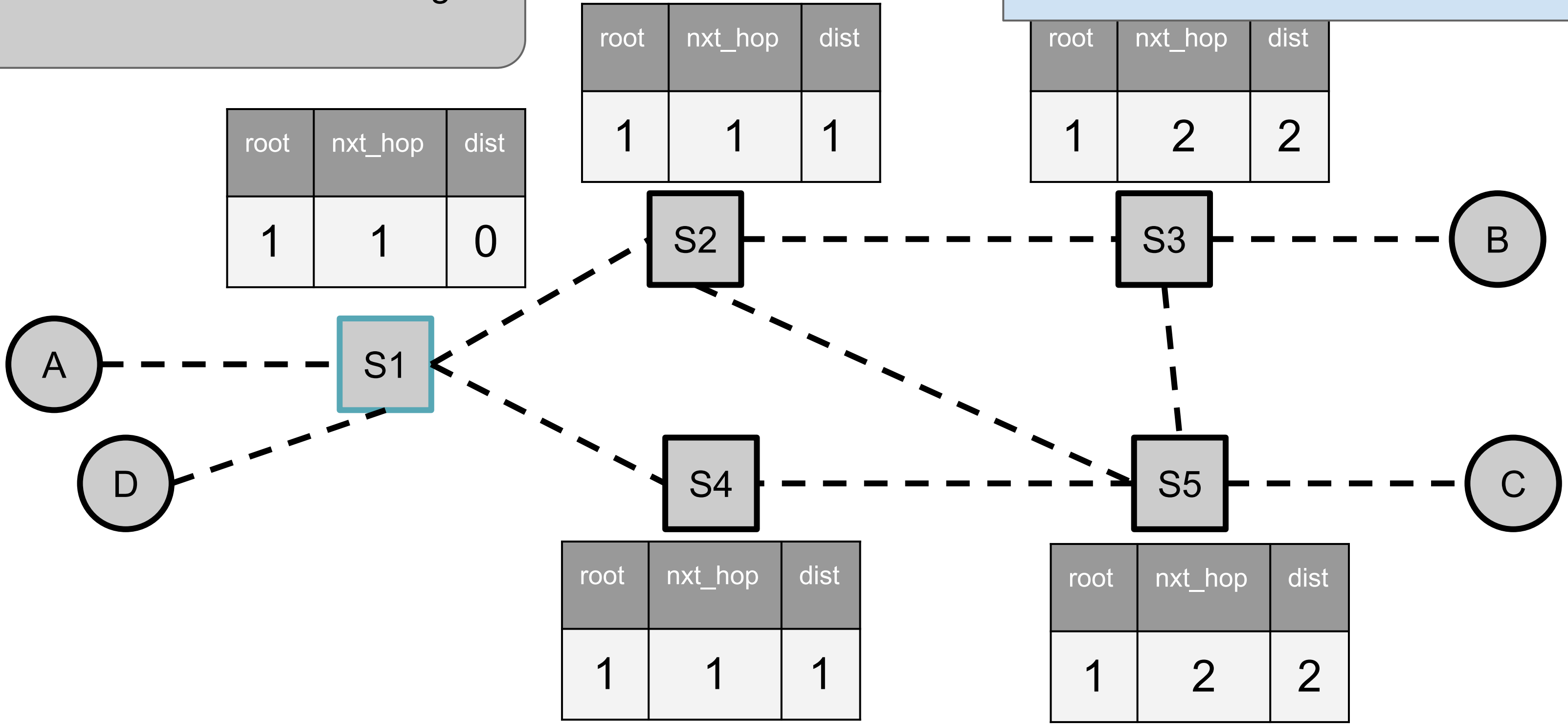
Goal: construct a tree for flooding

Legend

━━━ Part of spanning tree

🟥━━ Not part of spanning tree

- - - Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

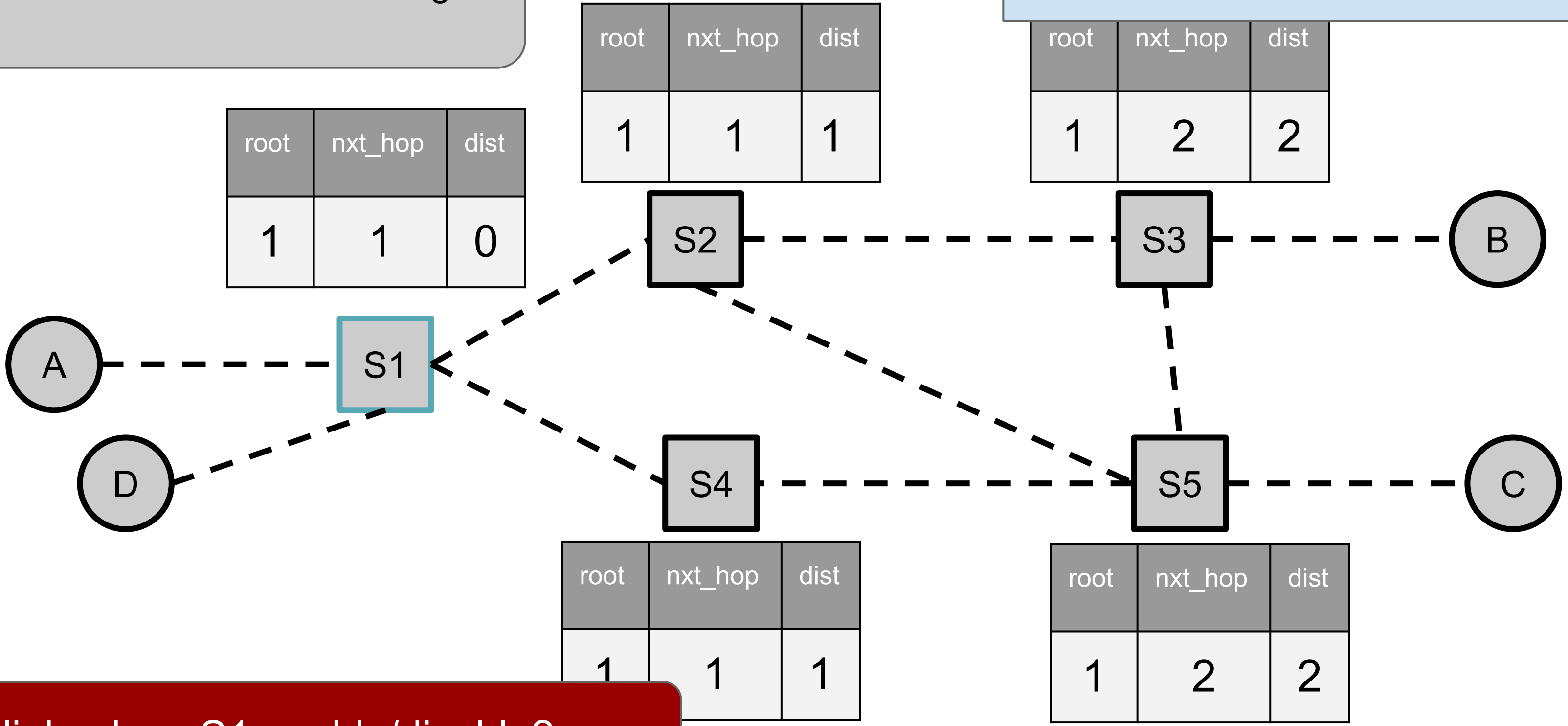
root	nxt_hop	dist
1	2	2

Spanning Tree Protocol

Goal: construct a tree for flooding

Legend

━━━	Part of spanning tree
🟥	Not part of spanning tree
- - -	Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S1 enable/disable?

Spanning Tree Protocol

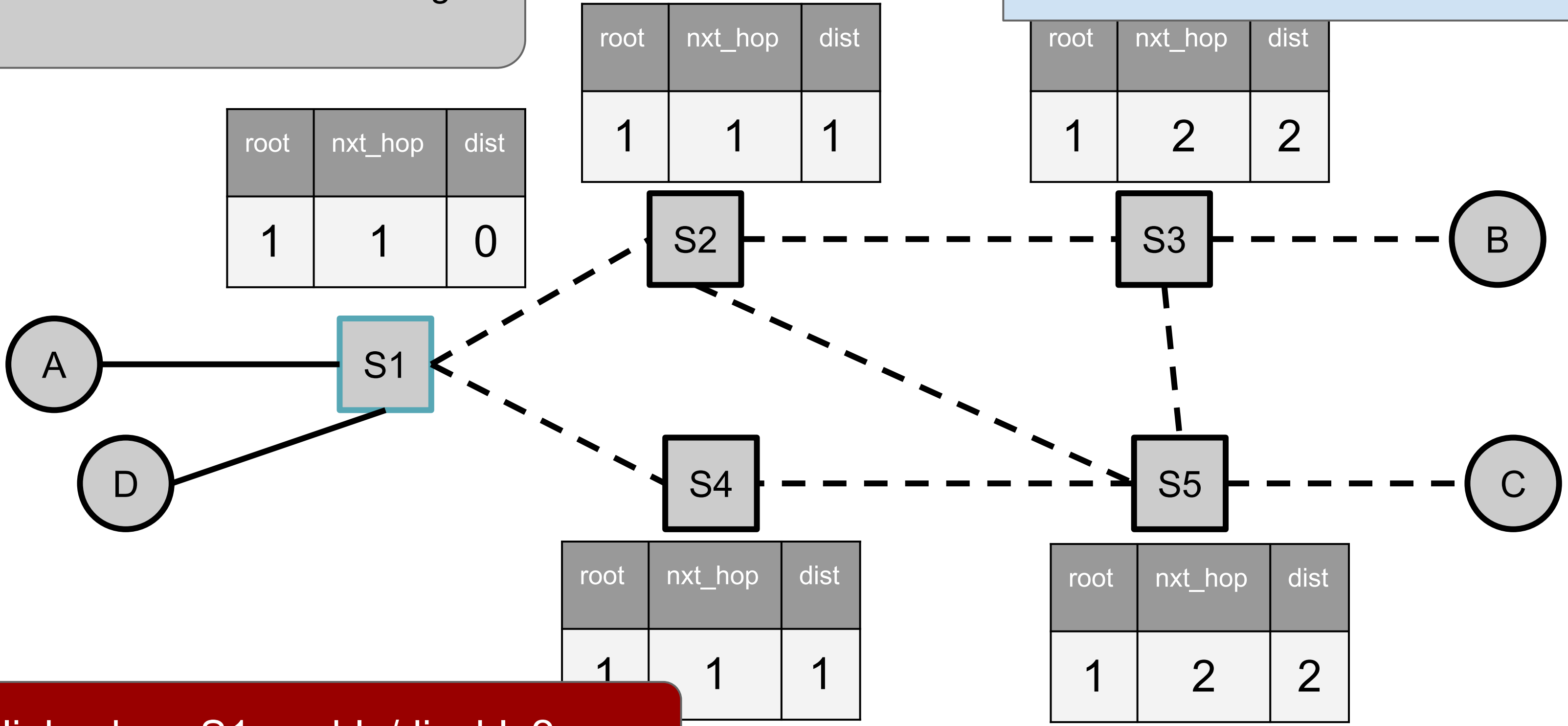
Goal: construct a tree for flooding

Legend

━━━━ Part of spanning tree

━━━━ Not part of spanning tree

- - - - Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

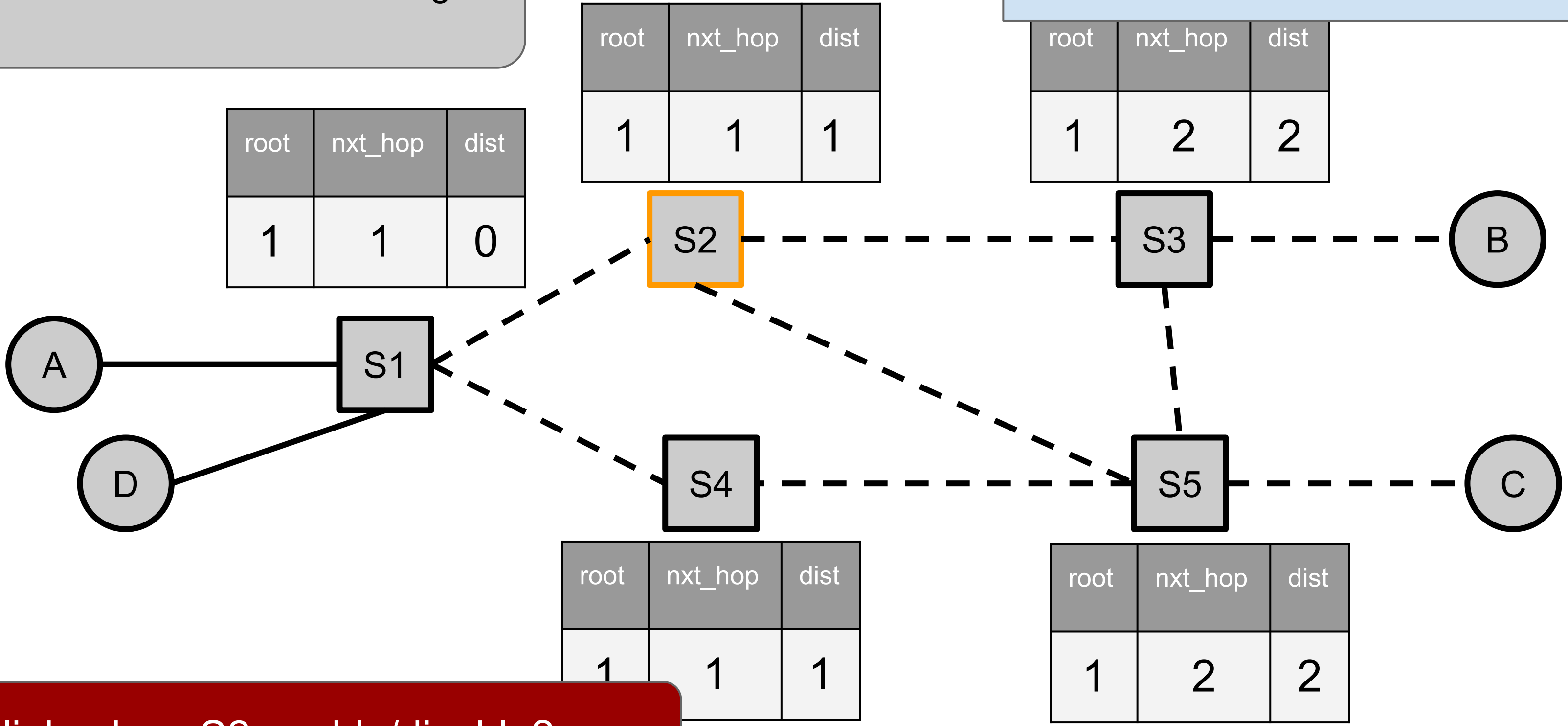
Which links does S1 enable/disable?

Spanning Tree Protocol

Goal: construct a tree for flooding

Legend

- ▬ Part of spanning tree
- 🟥 Not part of spanning tree
- ╌ Unknown



root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	0

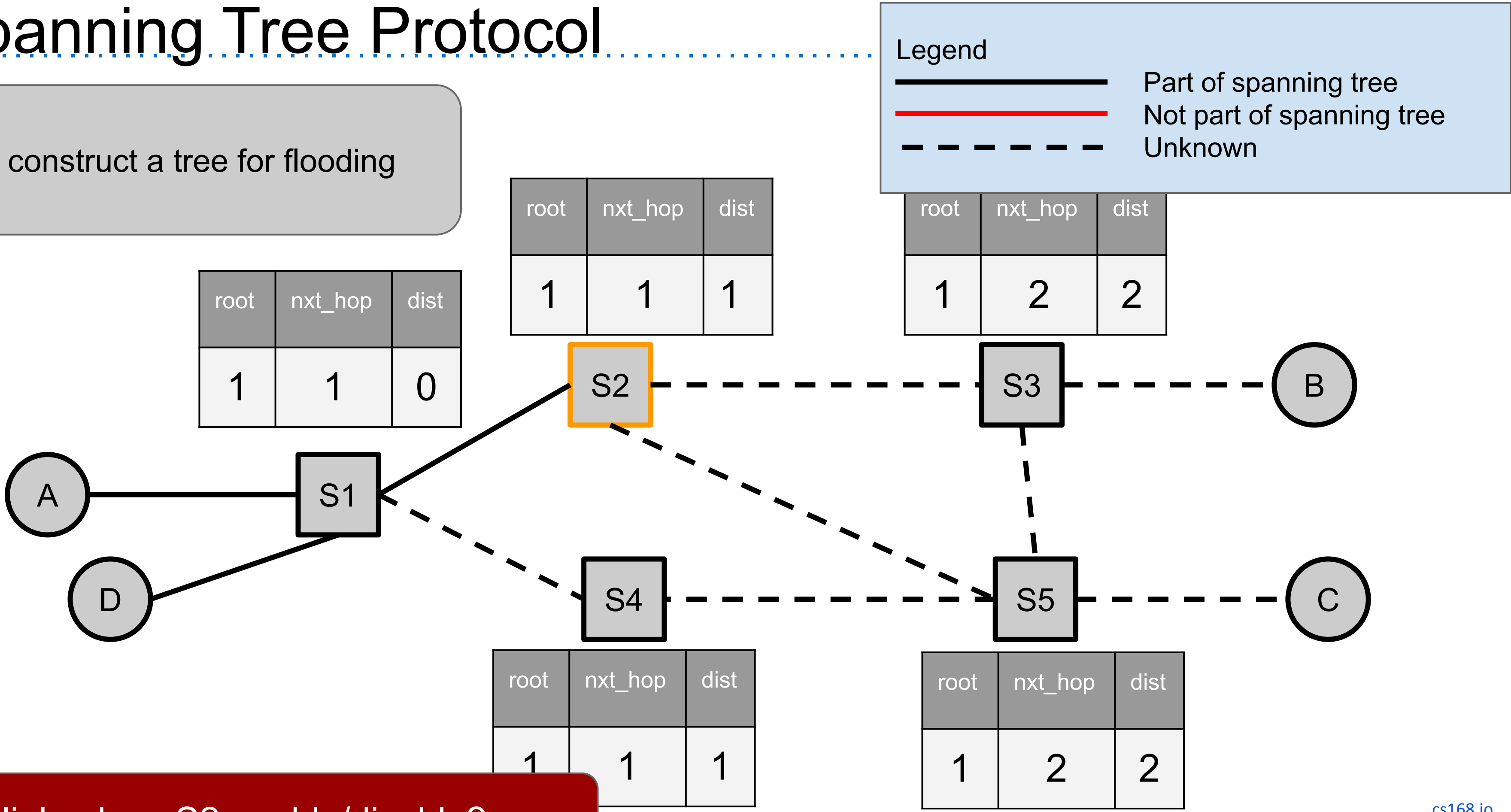
root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S2 enable/disable?

Spanning Tree Protocol

Goal: construct a tree for flooding



Legend

━━	Part of spanning tree
🟥	Not part of spanning tree
- - -	Unknown

root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S2 enable/disable?

Spanning Tree Protocol

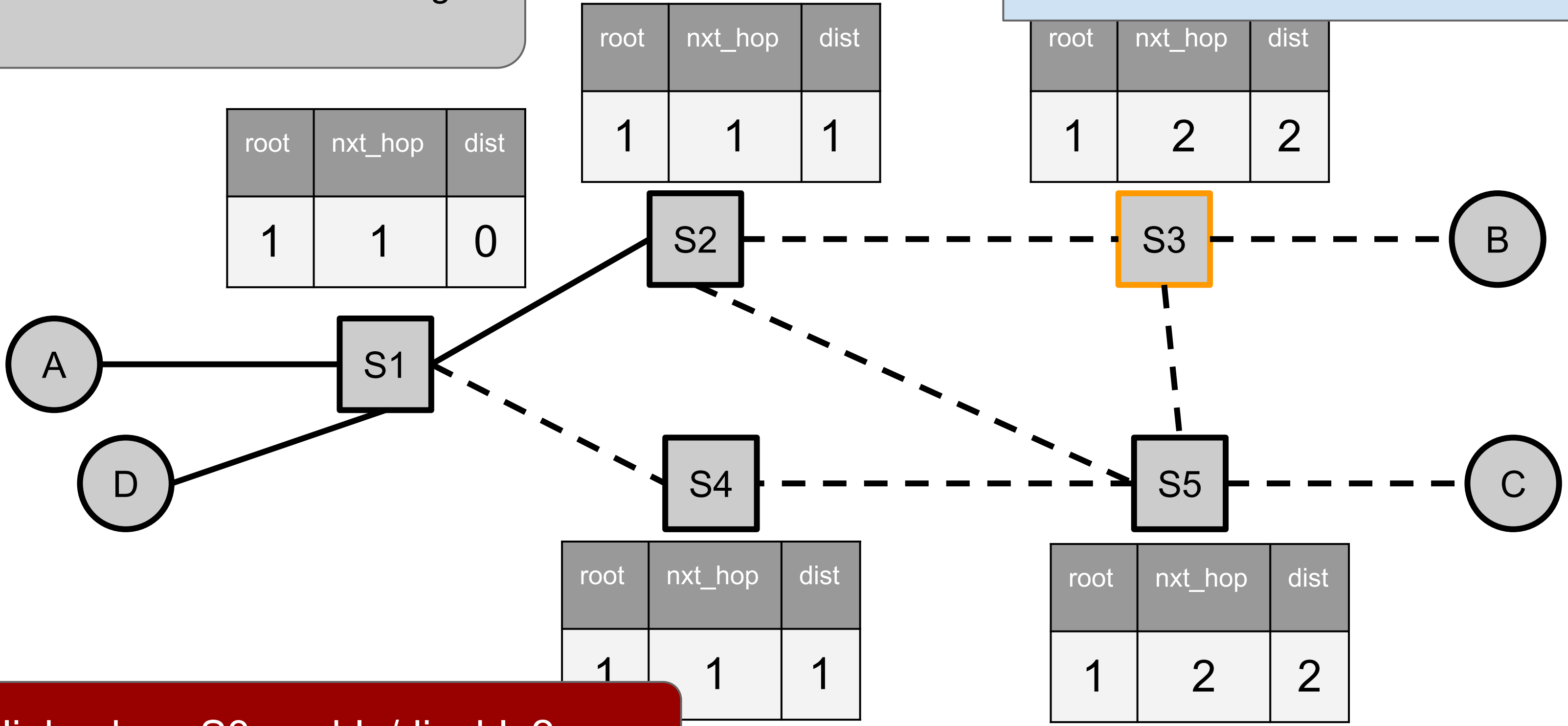
Goal: construct a tree for flooding

Legend

▬ Part of spanning tree

🟥 Not part of spanning tree

- - - Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S3 enable/disable?

Spanning Tree Protocol

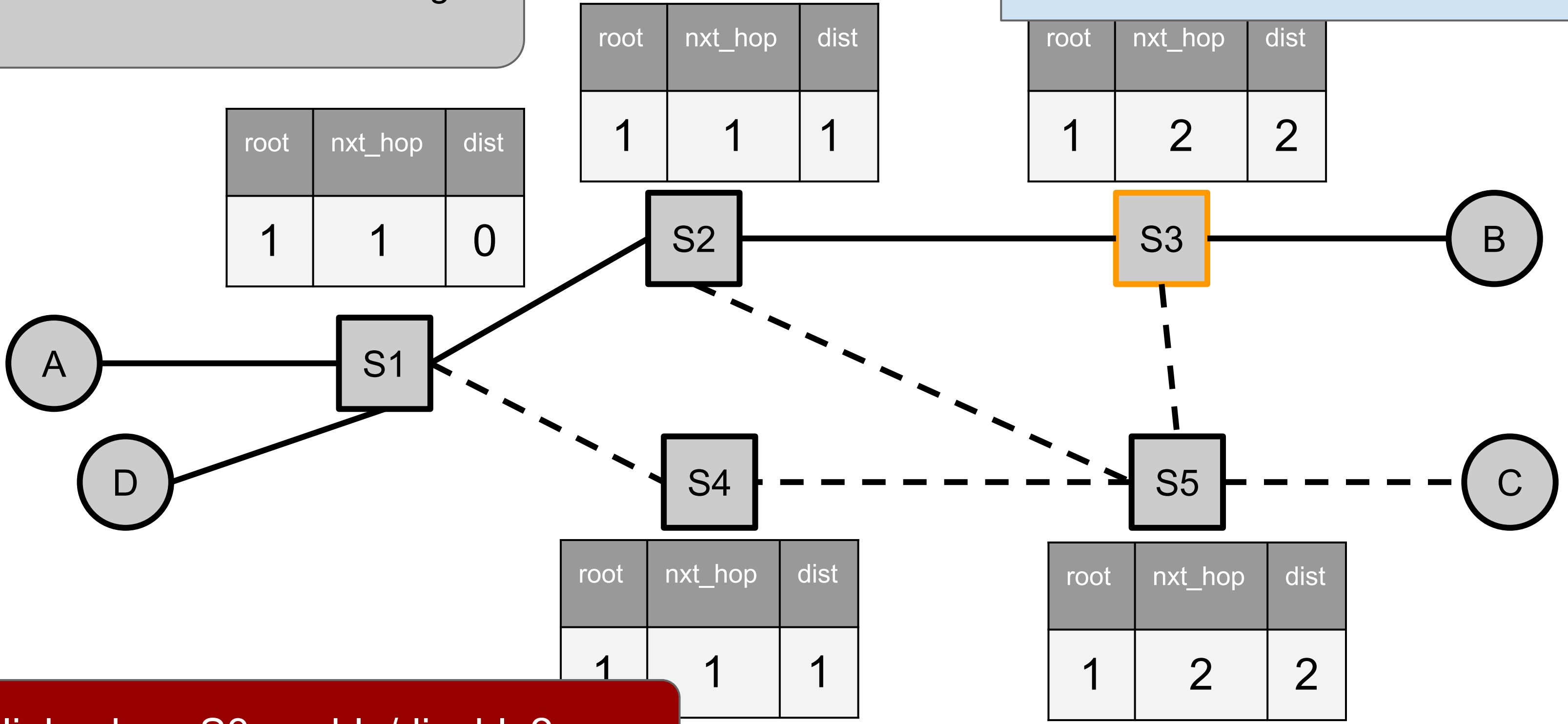
Goal: construct a tree for flooding

Legend

━━━ Part of spanning tree

🟥 Not part of spanning tree

- - - Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S3 enable/disable?

Spanning Tree Protocol

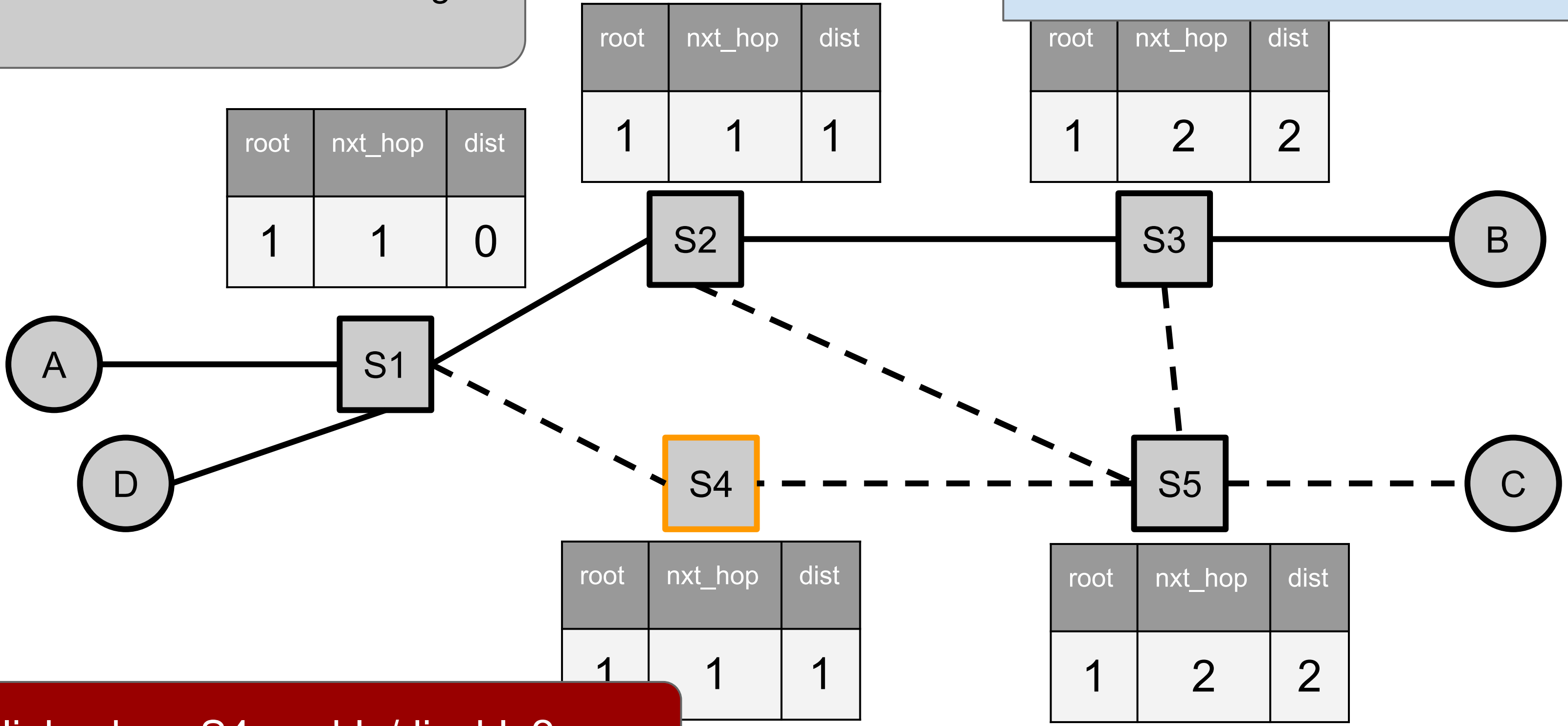
Goal: construct a tree for flooding

Legend

Part of spanning tree

Not part of spanning tree

Unknown



root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S4 enable/disable?

Spanning Tree Protocol

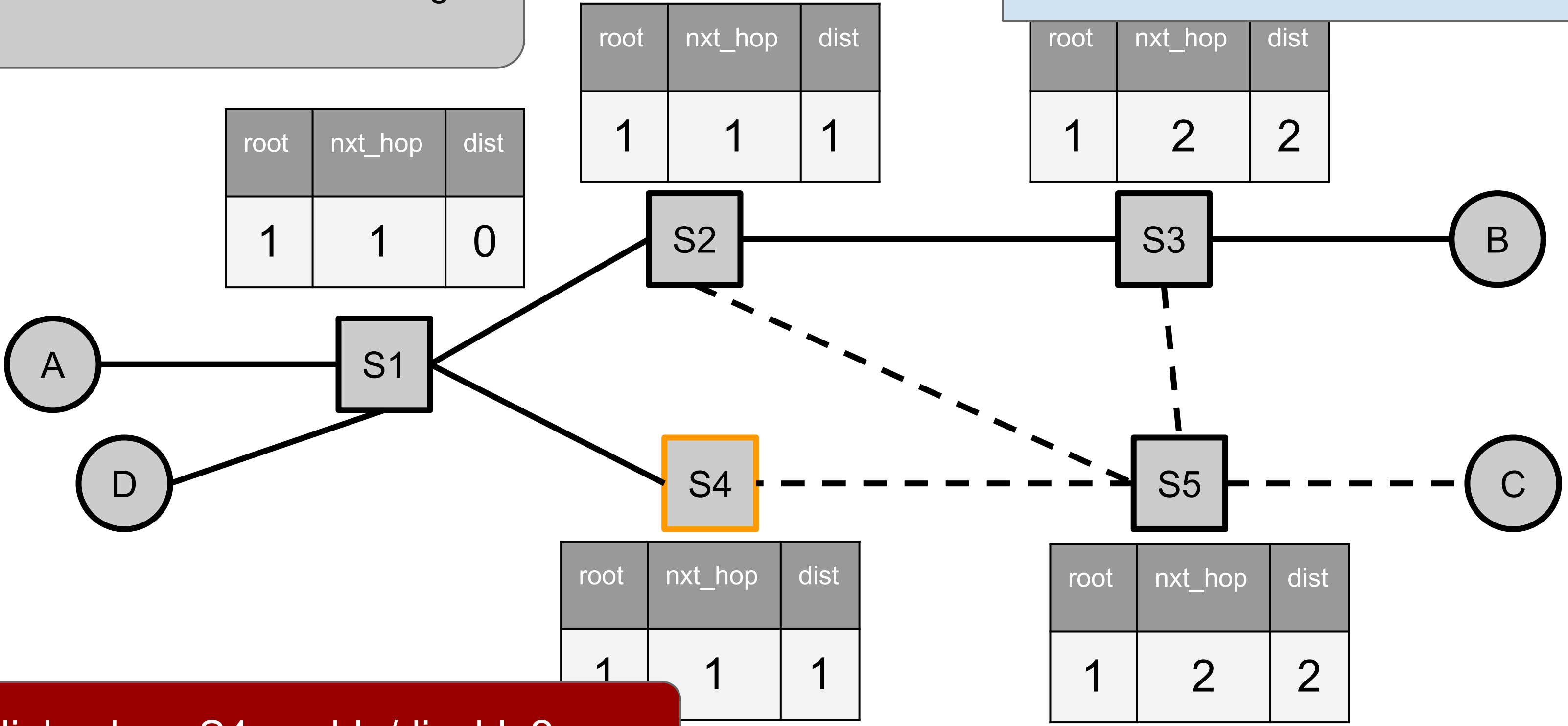
Goal: construct a tree for flooding

Legend

━━━ Part of spanning tree

━━━ Not part of spanning tree

- - - - - - Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S4 enable/disable?

Spanning Tree Protocol

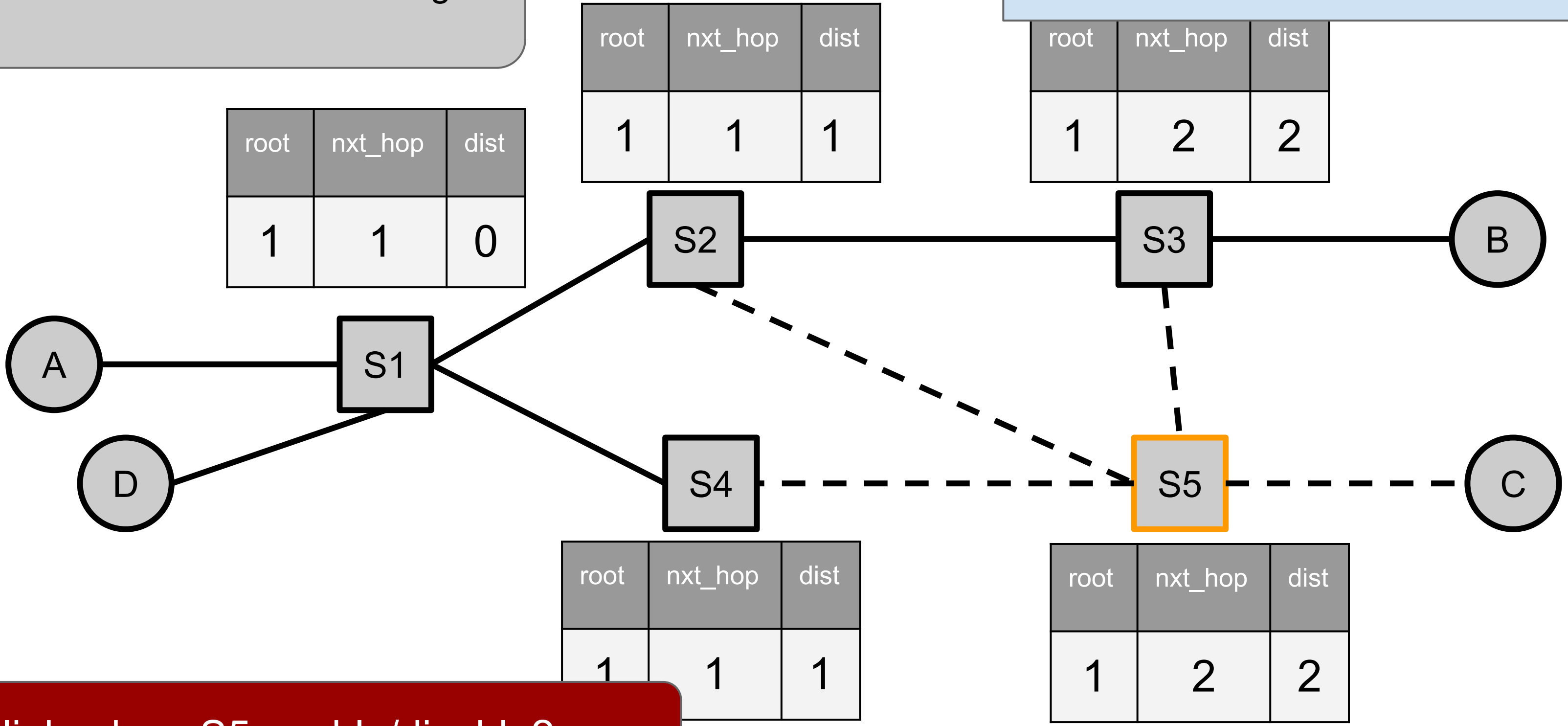
Goal: construct a tree for flooding

Legend

━━━━ Part of spanning tree

━━━━ Not part of spanning tree

╍╍╍╍ Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S5 enable/disable?

Spanning Tree Protocol

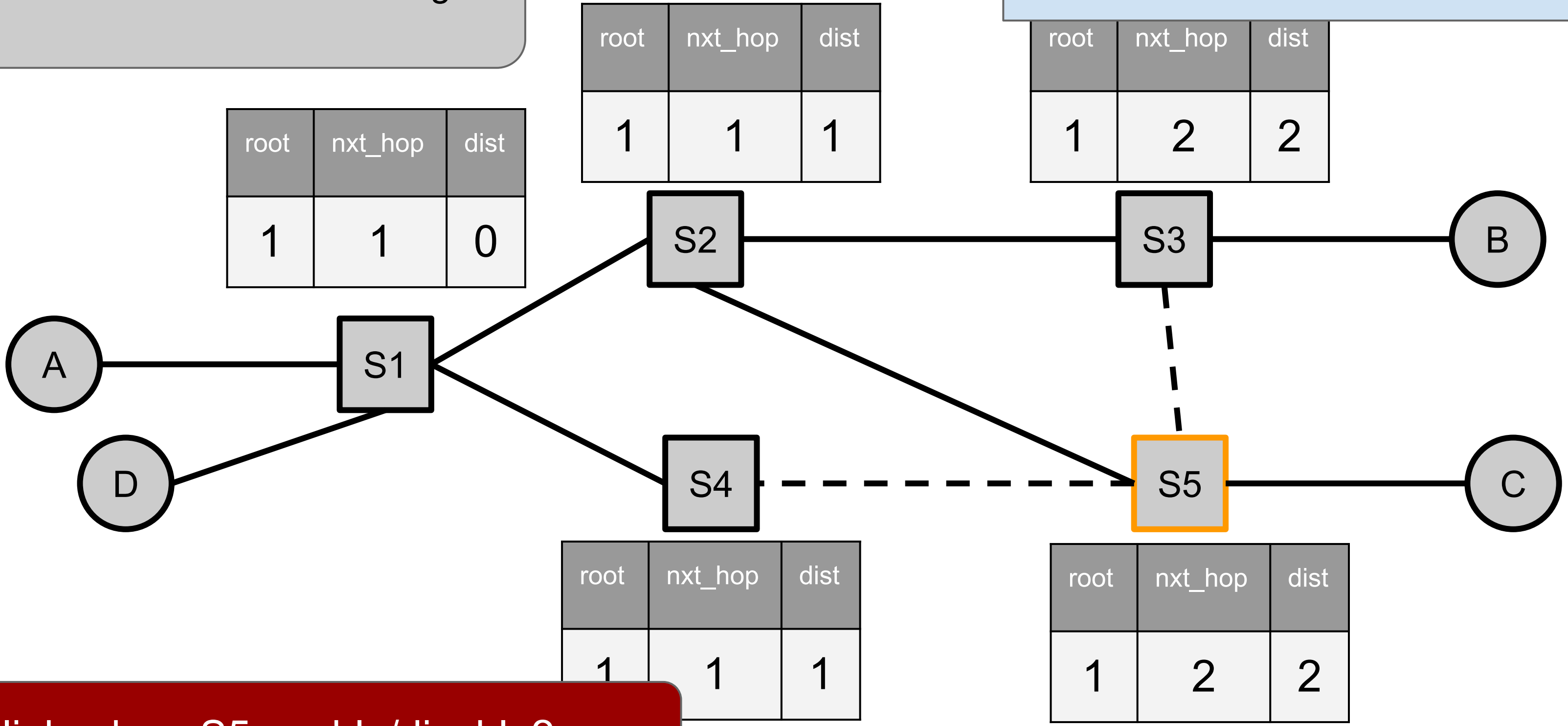
Goal: construct a tree for flooding

Legend

─── Part of spanning tree

─── Not part of spanning tree

- - - Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Which links does S5 enable/disable?

Spanning Tree Protocol

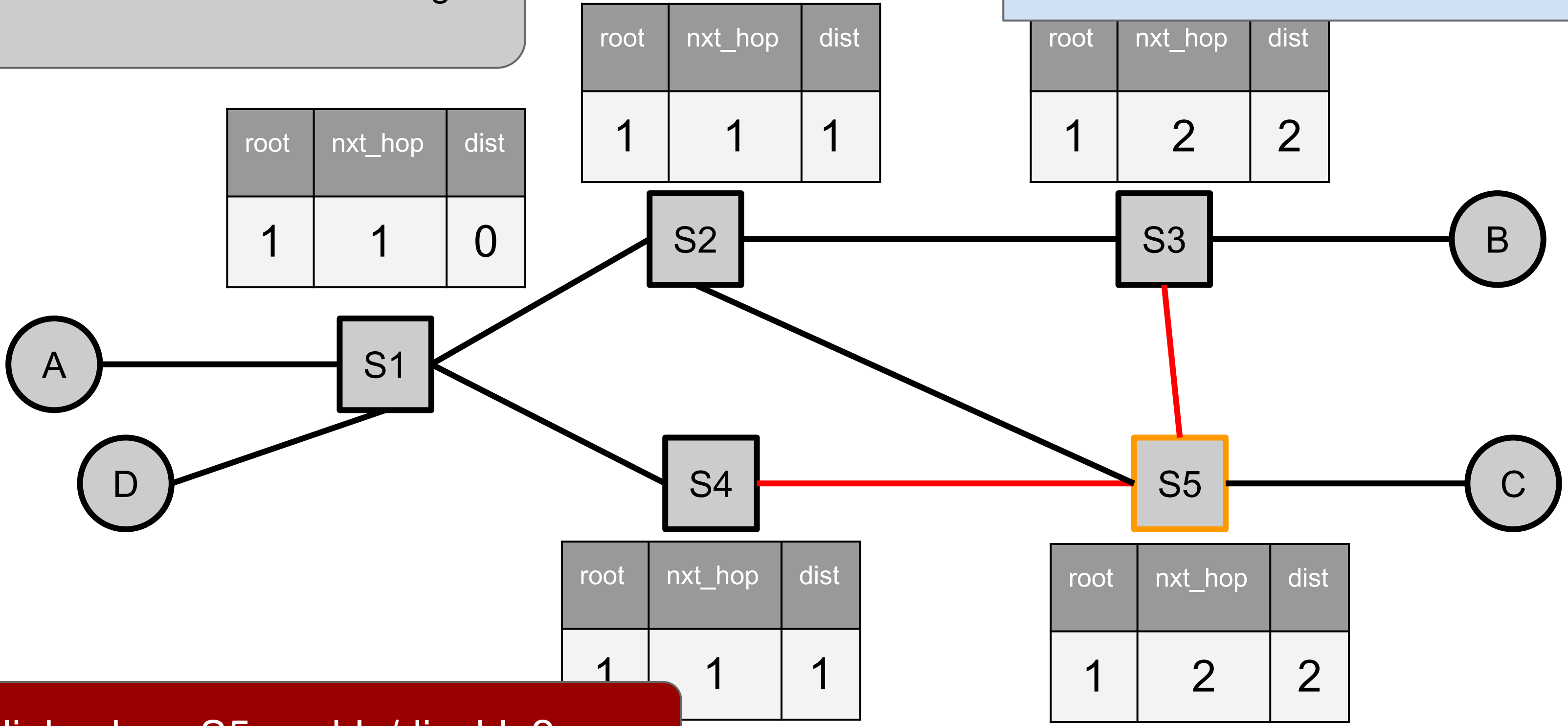
Goal: construct a tree for flooding

Legend

━━━ Part of spanning tree

🟥 Not part of spanning tree

- - - - Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

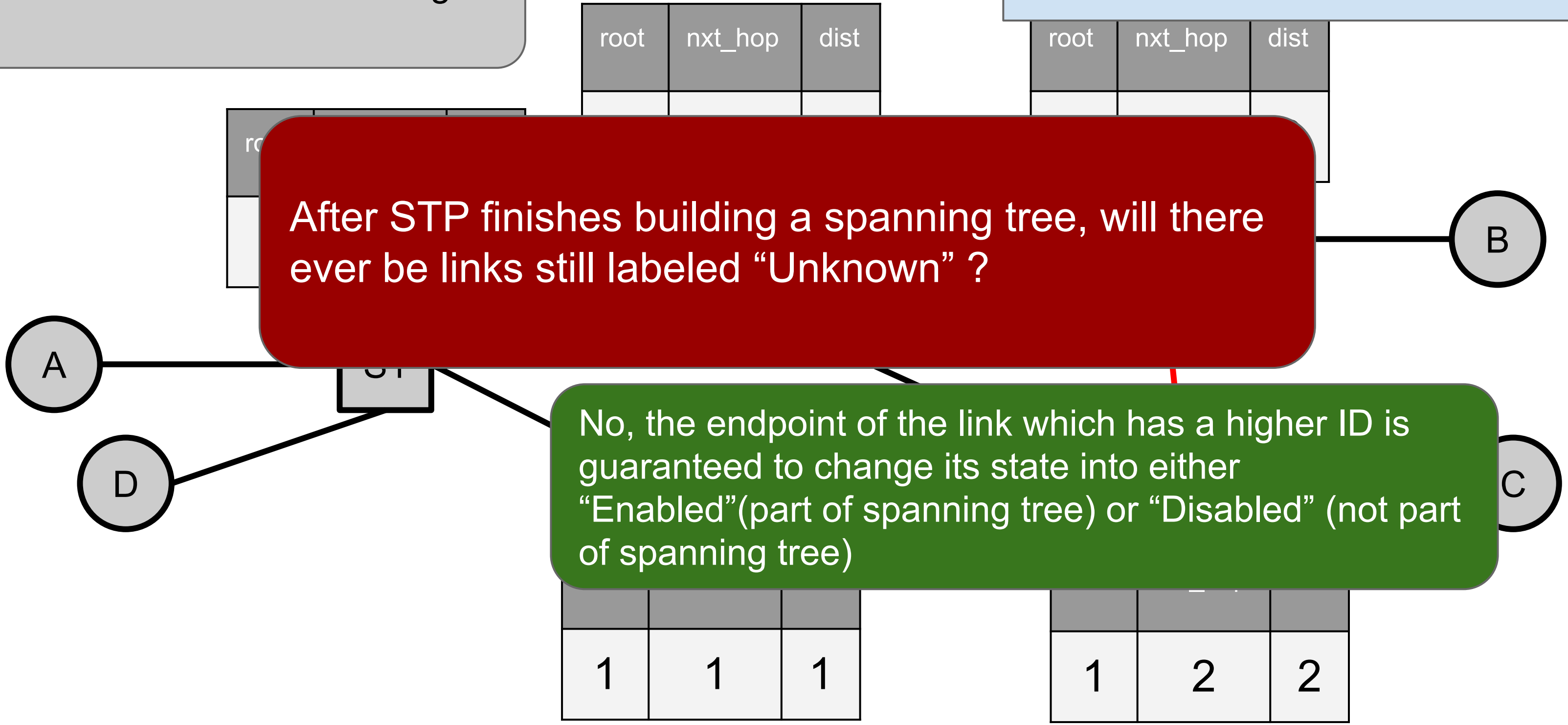
Which links does S5 enable/disable?

Spanning Tree Protocol

Goal: construct a tree for flooding

Legend

━━━━	Part of spanning tree
━━━━	Not part of spanning tree
– – – –	Unknown



1	1	1

1	2	2

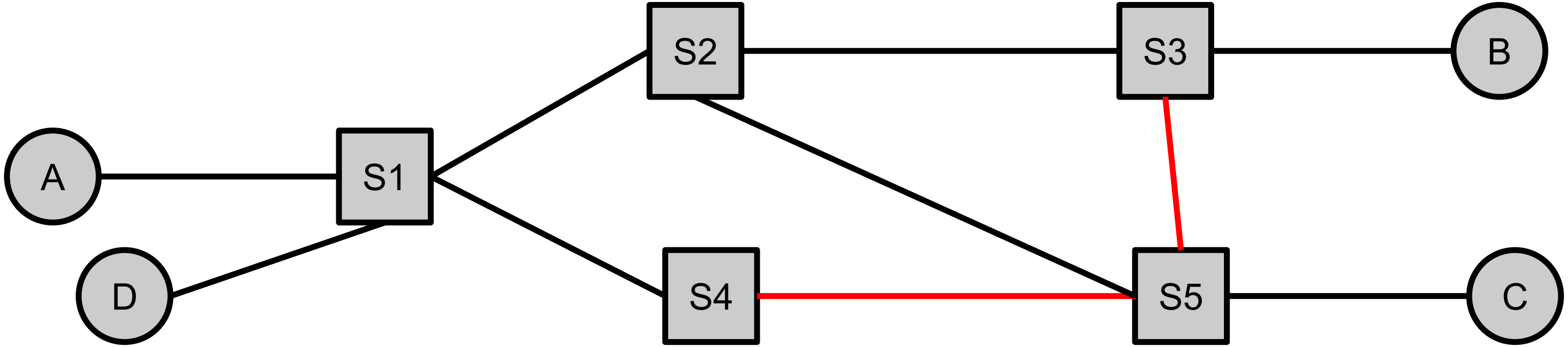
After STP finishes building a spanning tree, will there ever be links still labeled “Unknown” ?

No, the endpoint of the link which has a higher ID is guaranteed to change its state into either “Enabled”(part of spanning tree) or “Disabled” (not part of spanning tree)

Spanning Tree Protocol

Goal: construct a tree for flooding

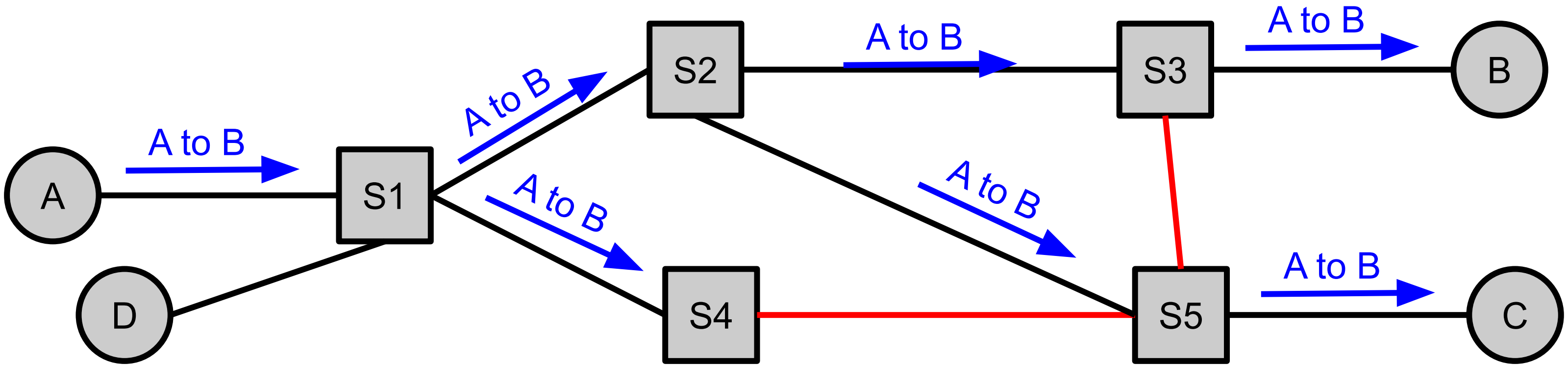
We have a spanning tree now.
Learning switches can use this for flooding!



Spanning Tree Protocol

Goal: construct a tree for flooding

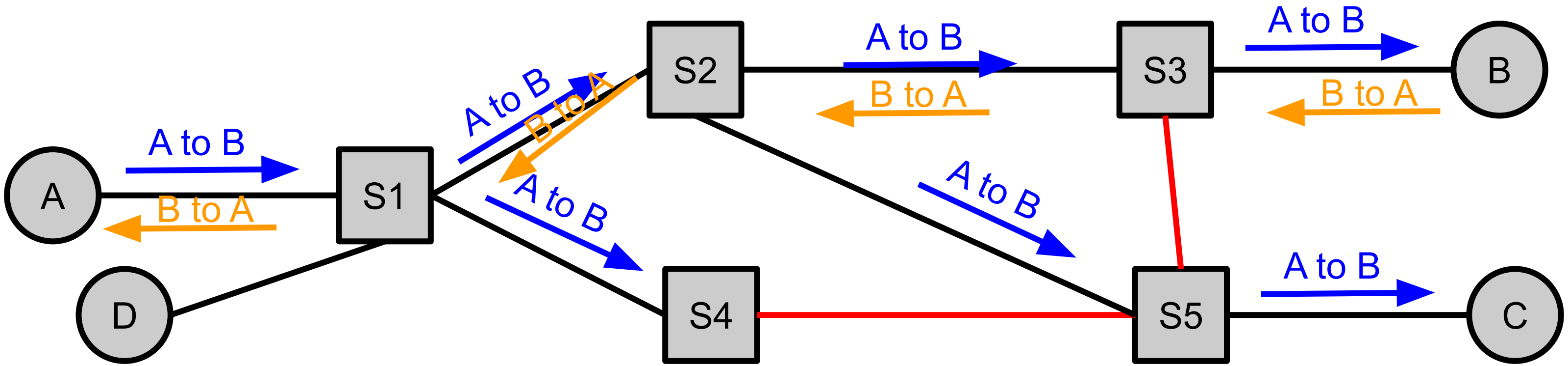
We have a spanning tree now.
Learning switches can use this for flooding!



Spanning Tree Protocol

Goal: construct a tree for flooding

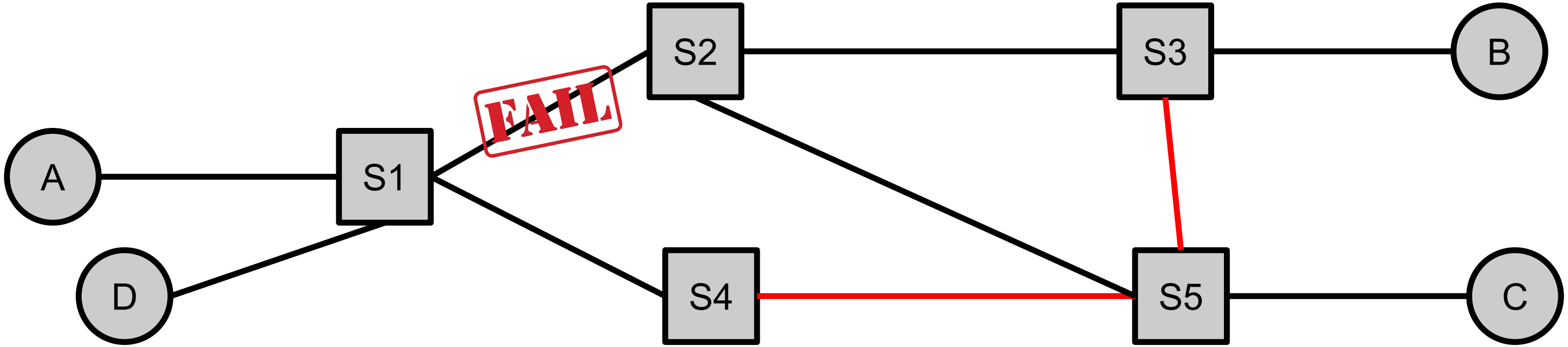
We have a spanning tree now.
Learning switches can use this for flooding!



Spanning Tree Protocol: failures

Legend

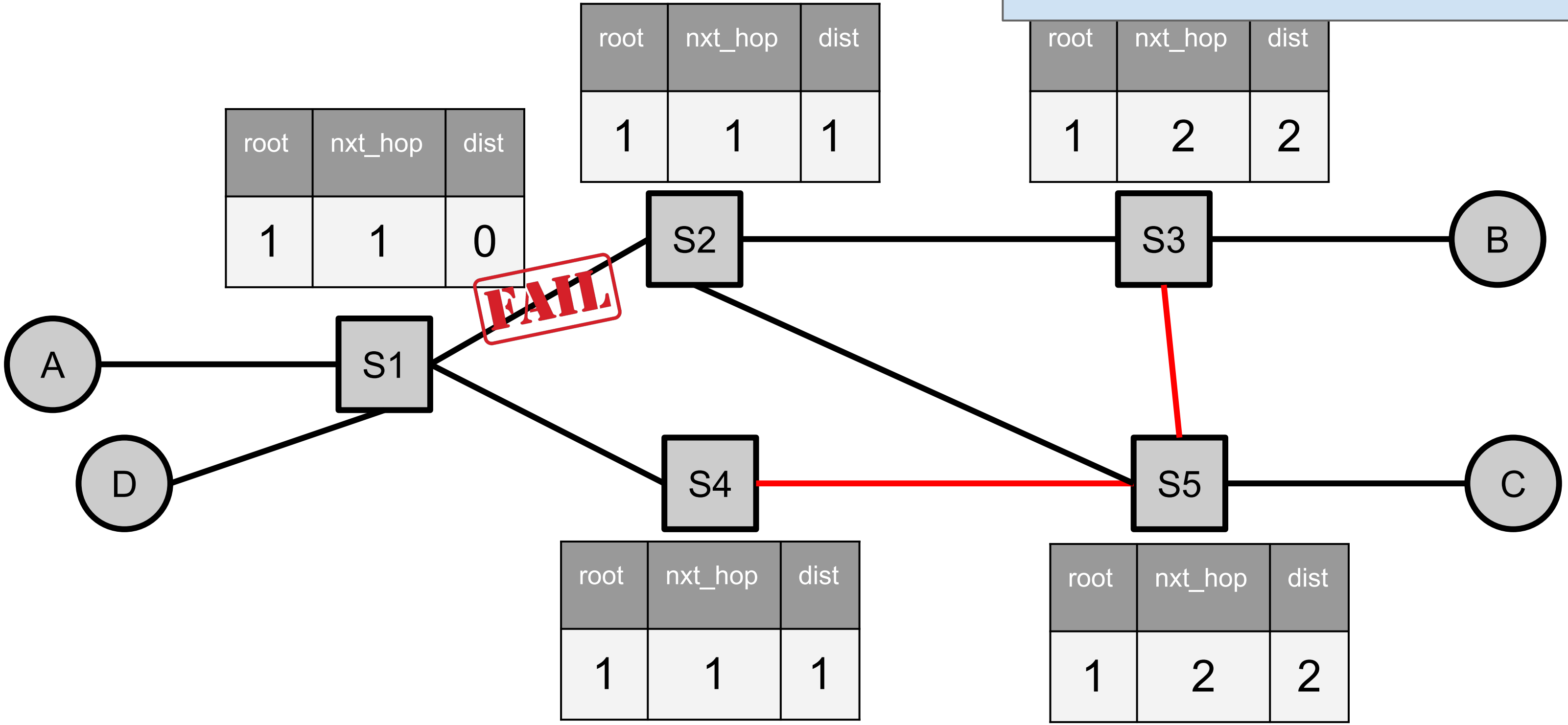
▬	Part of spanning tree
🟥	Not part of spanning tree
- - -	Unknown



Spanning Tree Protocol: failures

Legend

- Part of spanning tree
- Not part of spanning tree
- Unknown



root	nxt_hop	dist
1	1	0

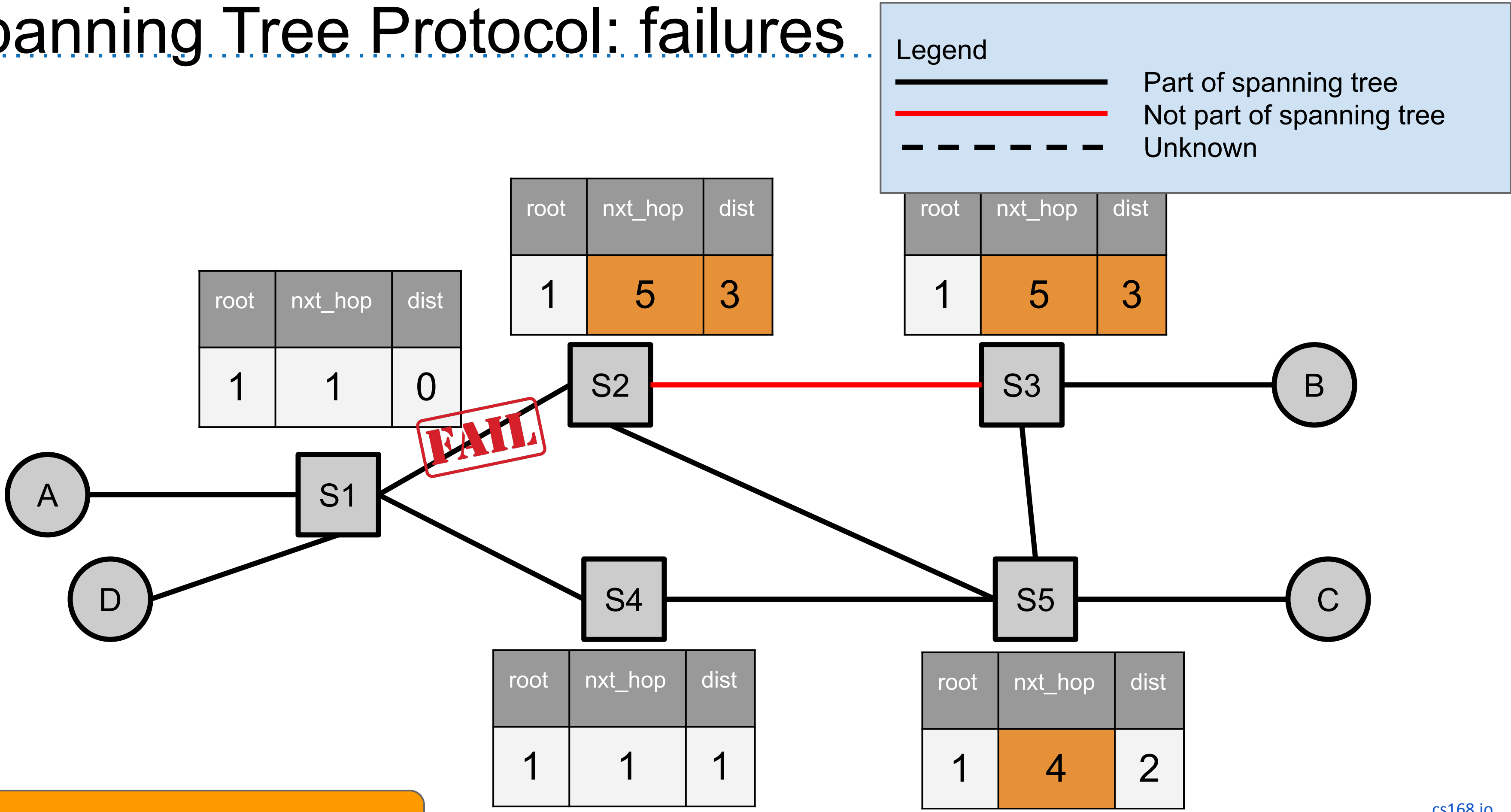
root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	2	2

Spanning Tree Protocol: failures



Legend

— Part of spanning tree

— Not part of spanning tree

- - - Unknown

root	nxt_hop	dist
1	5	3

root	nxt_hop	dist
1	5	3

root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	1	1

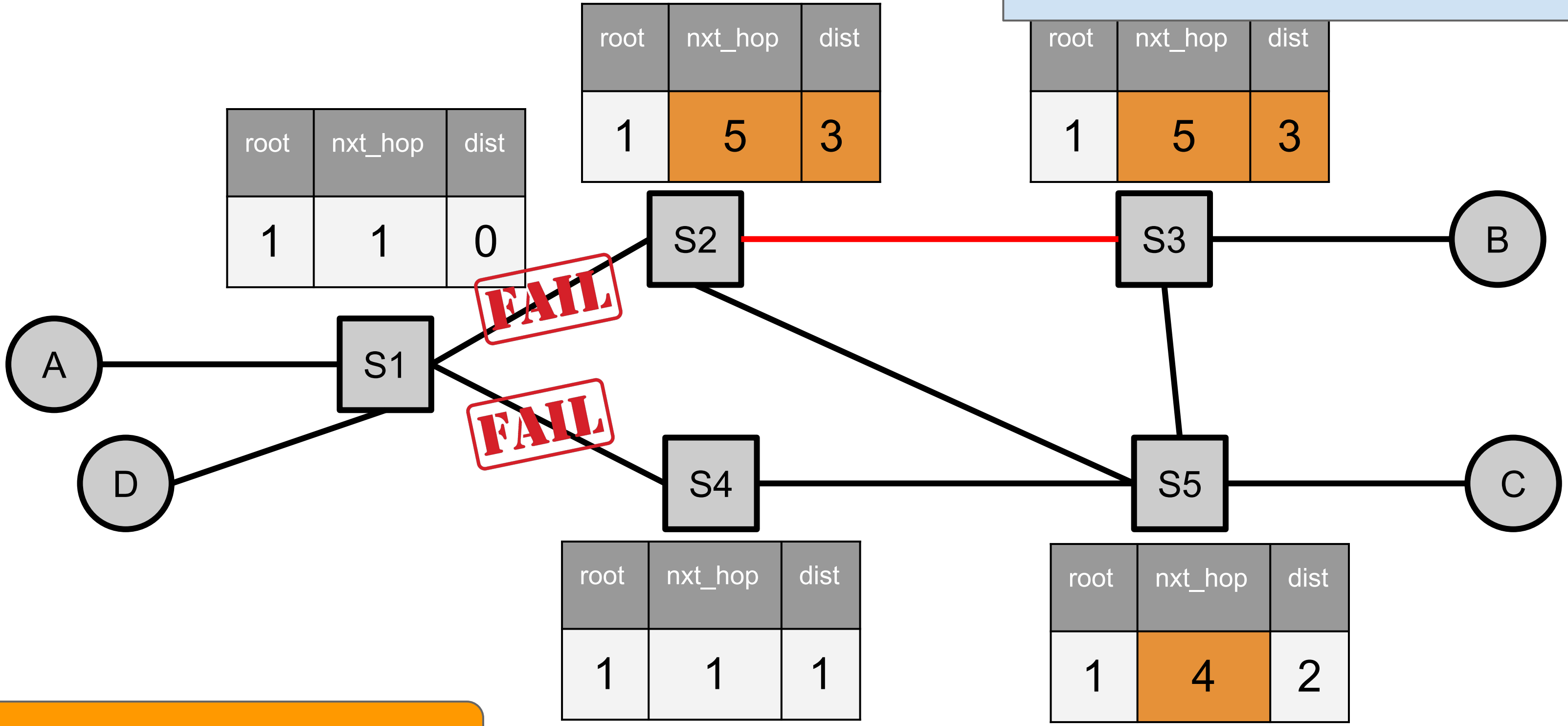
root	nxt_hop	dist
1	4	2

Orange highlights table changes

Spanning Tree Protocol: failures

Legend

- ━━━ Part of spanning tree
- 🟥 Not part of spanning tree
- – – – Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
1	5	3

root	nxt_hop	dist
1	5	3

root	nxt_hop	dist
1	1	1

root	nxt_hop	dist
1	4	2

Orange highlights table changes

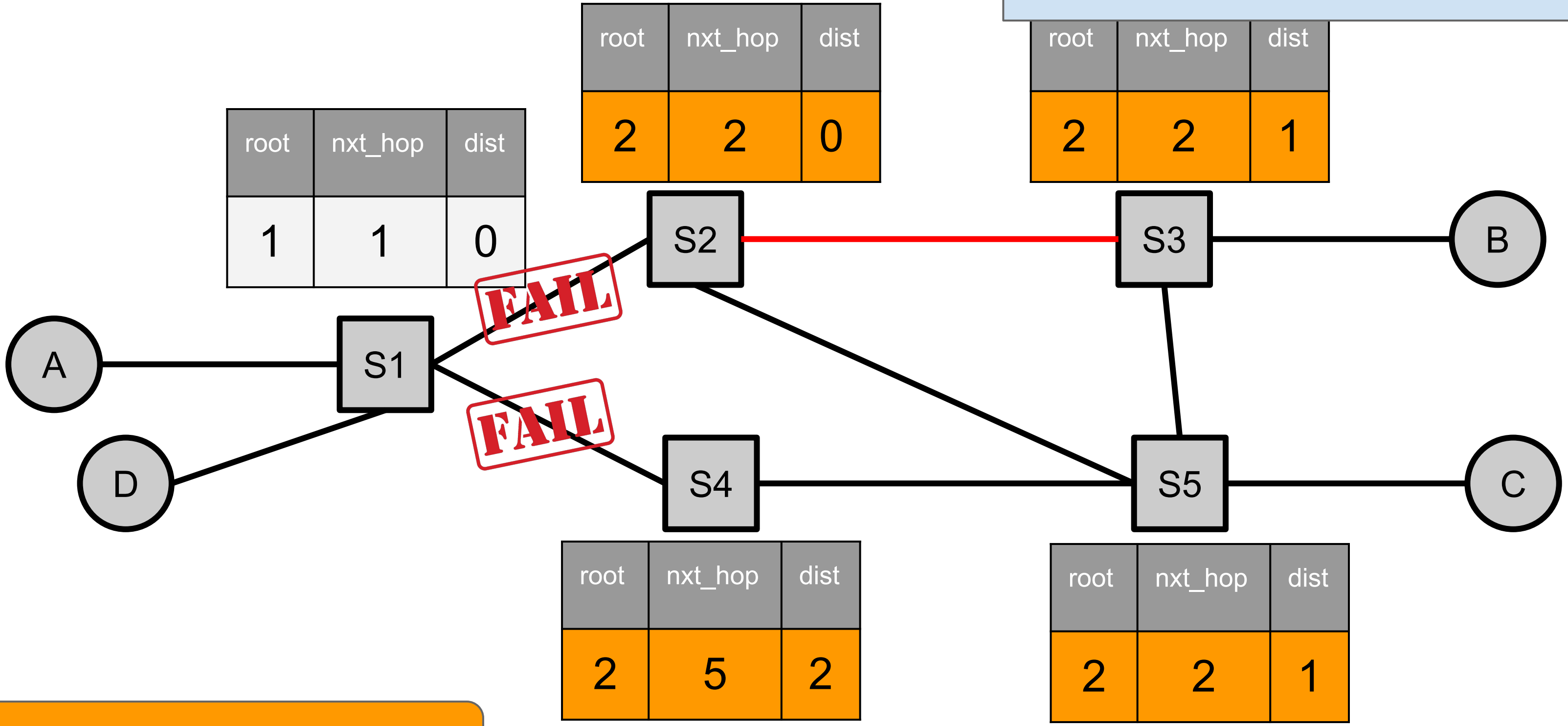
Spanning Tree Protocol: failures

Legend

━━━ Part of spanning tree

━━━ Not part of spanning tree

– – – – – – Unknown



root	nxt_hop	dist
1	1	0

root	nxt_hop	dist
2	2	0

root	nxt_hop	dist
2	2	1

root	nxt_hop	dist
2	5	2

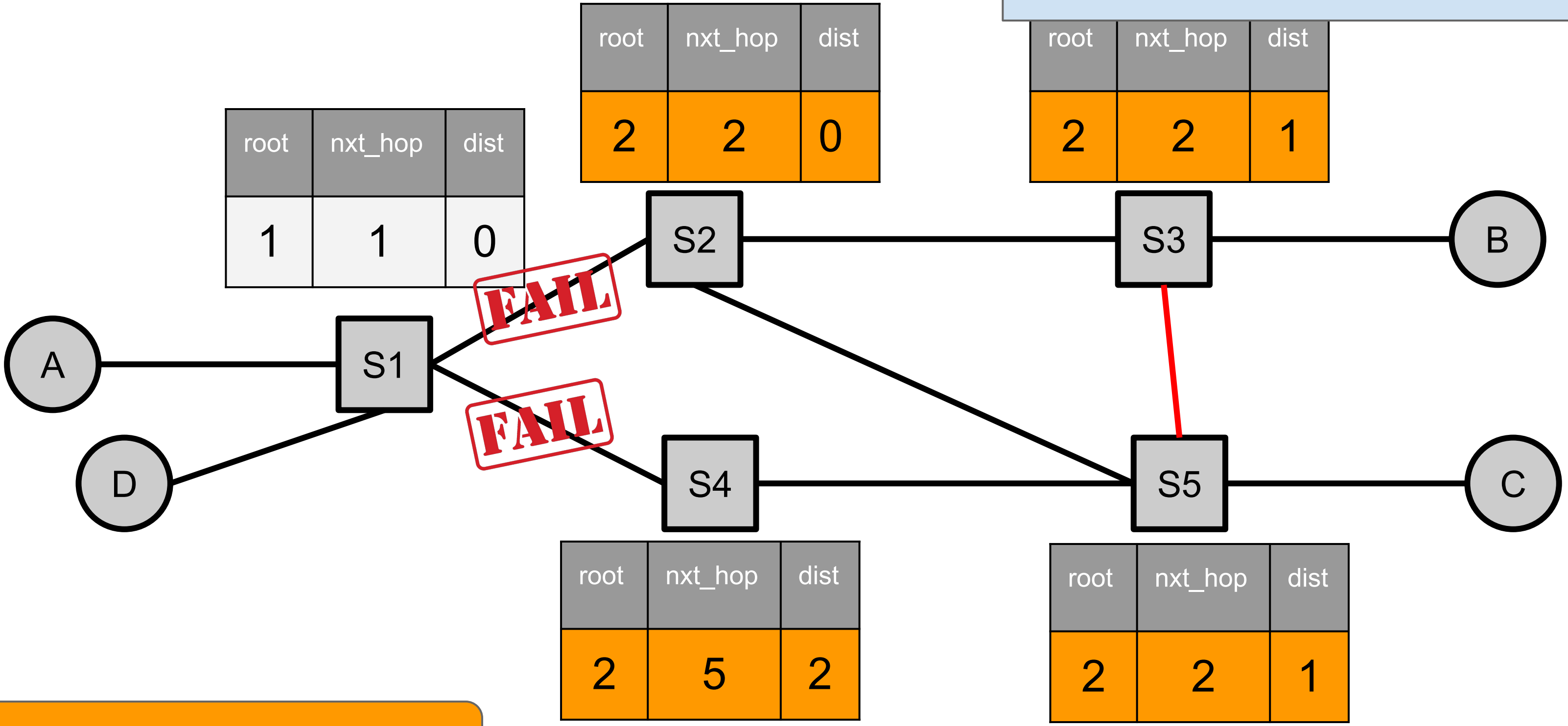
root	nxt_hop	dist
2	2	1

Orange highlights table changes

Spanning Tree Protocol: failures

Legend

▬	Part of spanning tree
🟥	Not part of spanning tree
- - -	Unknown



S1:

root	nxt_hop	dist
1	1	0

S2:

root	nxt_hop	dist
2	2	0

S3:

root	nxt_hop	dist
2	2	1

S4:

root	nxt_hop	dist
2	5	2

S5:

root	nxt_hop	dist
2	2	1

Orange highlights table changes

Spanning Tree Protocol: failures

Note:

After both links out of S1 failed, we still had a valid spanning tree and could in theory continue flooding on it

But STP forced us to recompute spanning tree

Spanning Tree Protocol: failures

Note:

After both links out of S1 failed, we still had a valid spanning tree and could in theory continue flooding on it

But STP forced us to recompute spanning tree

Can we do better?

Spanning Tree Protocol: failures

Note:

After both links out of S1 failed, we still had a valid spanning tree and could in theory continue flooding on it

But STP forced us to recompute spanning tree

Can we do better?

Yes. Beyond the course material. See Murphy's AXE for one example and more references

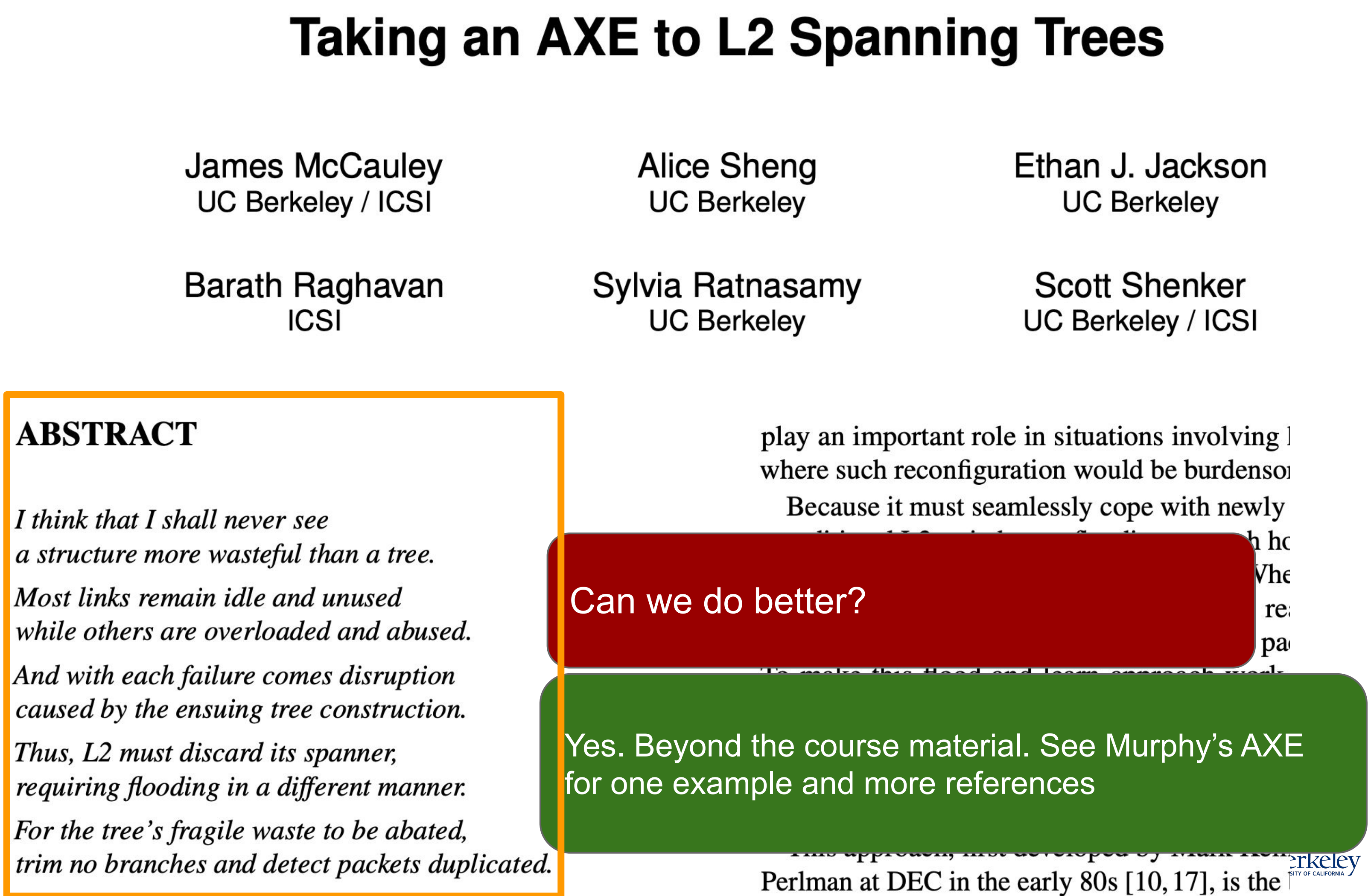
Spann

Note:

Note: Aft

could in

But STP f



Taking an AXE to L2 Spanning Trees

James McCauley
UC Berkeley / ICSI

Alice Sheng
UC Berkeley

Ethan J. Jackson
UC Berkeley

Barath Raghavan
ICSI

Sylvia Ratnasamy
UC Berkeley

Scott Shenker
UC Berkeley / ICSI

ABSTRACT

I think that I shall never see
a structure more wasteful than a tree.

Most links remain idle and unused
while others are overloaded and abused.

And with each failure comes disruption
caused by the ensuing tree construction.

Thus, L2 must discard its spanner,
requiring flooding in a different manner.

For the tree's fragile waste to be abated,
trim no branches and detect packets duplicated.

play an important role in situations involving
where such reconfiguration would be burdenso
Because it must seamlessly cope with newly

To make this flood-and-learn approach work

Perlman at DEC in the early 80s [10, 17], is the

Can we do better?

Yes. Beyond the course material. See Murphy's AXE for one example and more references

Learning Switches and Spanning Tree Protocol (STP)

In short, today we:

- Showed how learning switches learn via flooding in tree-topology
- Introduced loops in topology and realized we need to deal with them
- Introduced STP to "disable" certain links and have a tree for flooding
- Considered different failure scenarios and how STP reacts to them

Attribution

Slides based on Murphy McCauley's slides the Spring 2020 iteration of CS168.

Written by Narek Galstyan, Sarah McClure, and Alex Krentsel.