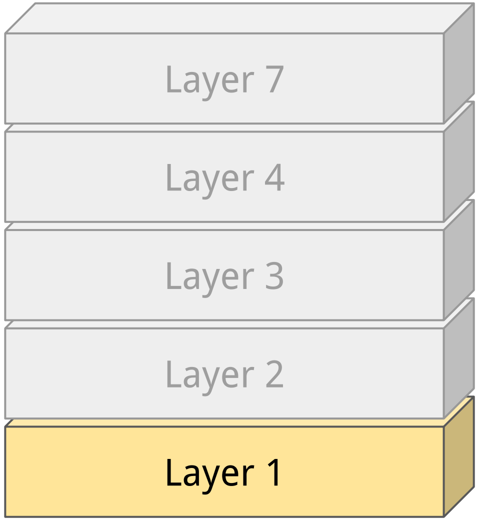
Forwarding & Ethernet

Fall 2024
[cs168.io](cs168.io)

Rob Shakir

Thanks to Murphy McCauley for some of the material!

Looking at Layers Again



Application → DNS, HTTP, …

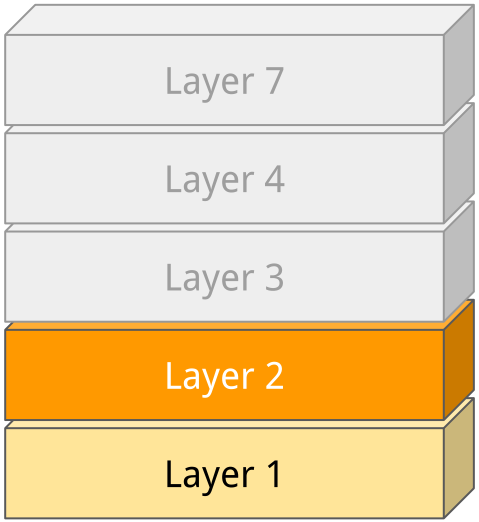
Transport → TCP

Network → IP

Data Link → Ethernet

Physical → Optical fibre, copper

Looking at Layers Again

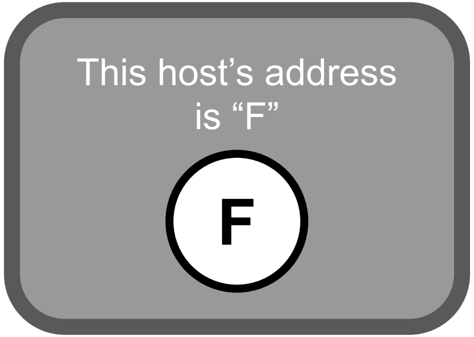


- Application → DNS, HTTP, …
- Transport → TCP
- Network → IP
- **Data Link → Ethernet**
- Physical → Optical fibre, copper

Lecture Plan

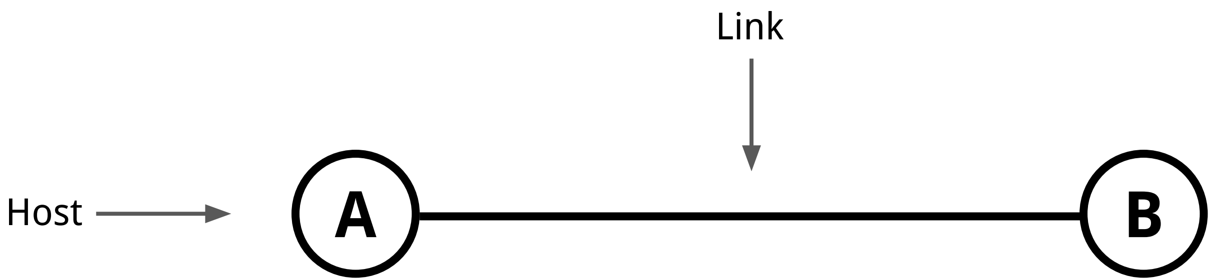
- We're going to start at the bottom of our set of layers.
    - And work-up, covering the layers in more detail.
- Today & next lecture:
    - Layer 2 & Ethernet.
    - How do packets get forwarded?
- Next week **- forwarding and routing**.
    - How do we know where packets need to go?

Drawing Hosts…

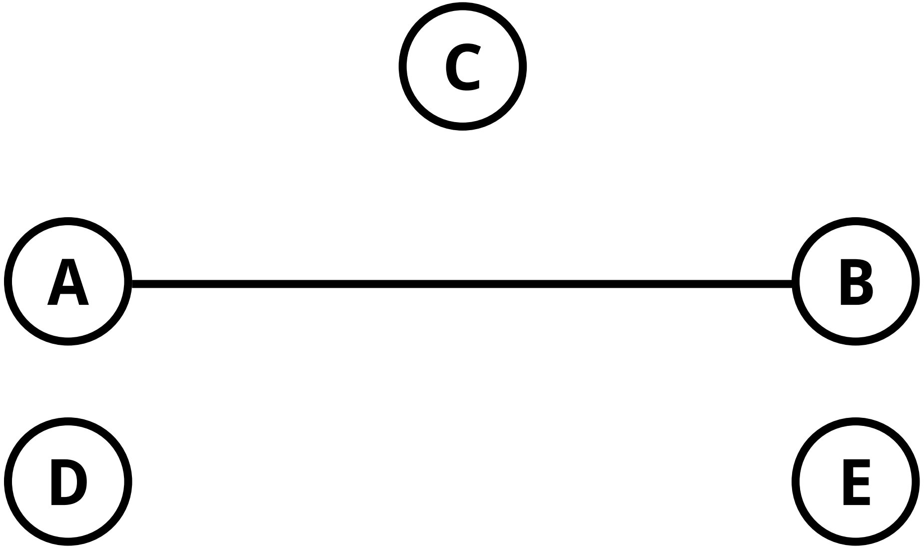


Connecting Nodes Together

Connecting Nodes Together

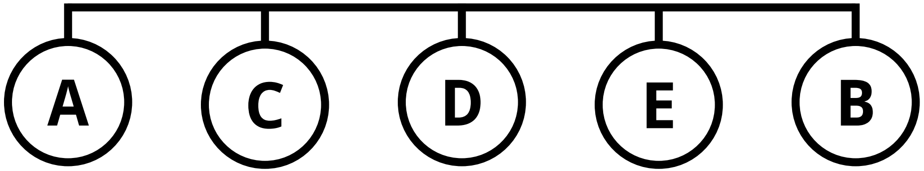


Connecting Nodes Together



Now what?

How can we connect this new host?



Naïve solution - connect all our hosts together
via a shared medium.

Some history - ALOHANet

- Norman Abramson had a problem at the University of Hawaii in 1968.
    - How do we allow people on other islands access to the U of H computer?

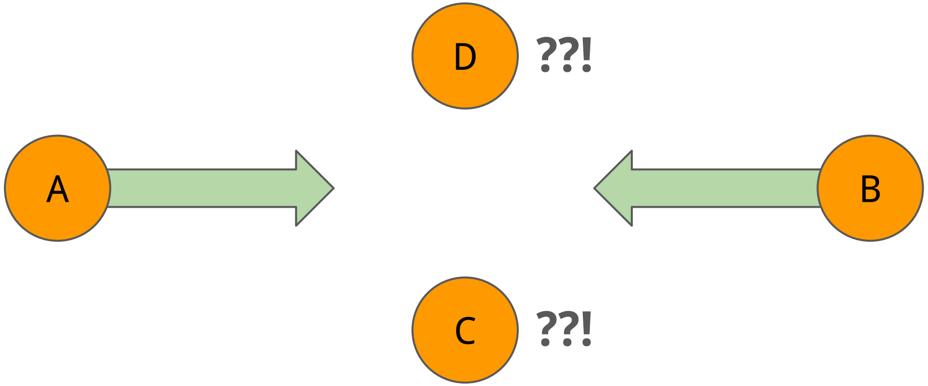
- ALOHANet
    - Additive Links On-line Hawaii Area
    - Wireless communications from terminals (computers) on other islands.
    - Hugely influential.

- ALOHANet was wireless - in radio networks there is a *shared medium* (the electromagnetic spectrum).

Shared Media

- In a network with a *shared medium*, then transmissions from different nodes may interfere or *collide* with each other.

- We need a way to allocate the medium to everyone wanting to use it…
  - A **multiple access protocol**.



Common Multiple Access Protocol approaches

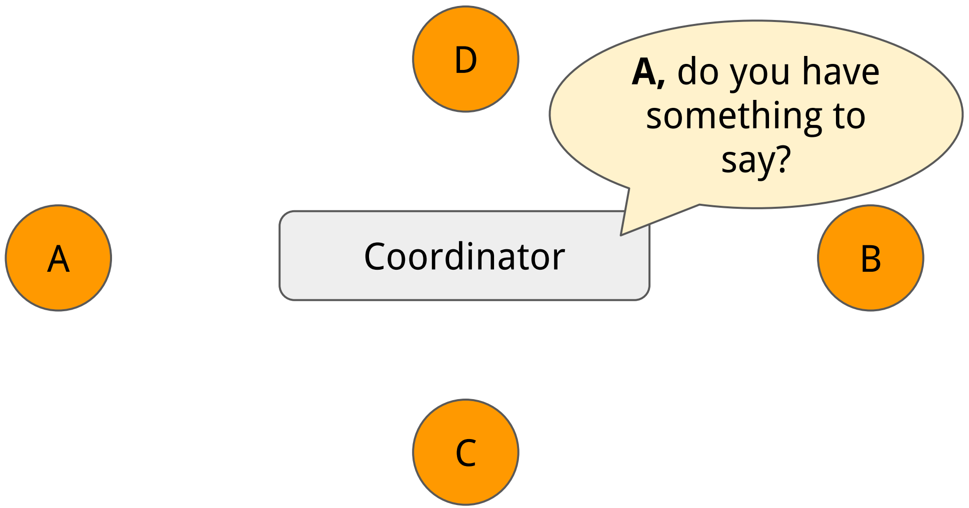
- Divide the medium by frequency – **frequency-division multiplexing**.
  - Give each connected node some slice of frequencies.
  - Can be wasteful – only a specific amount of frequency to allocate.
  - Not everyone has something to say all the time (many frequencies idle).

- Divide the medium by time - **time-division multiplexing.**
  - Divide time into fixed slots and allocate them to each connected node.
  - Same downside – only so much time, many slots are idle.

- Alternative: can connected nodes take turns?

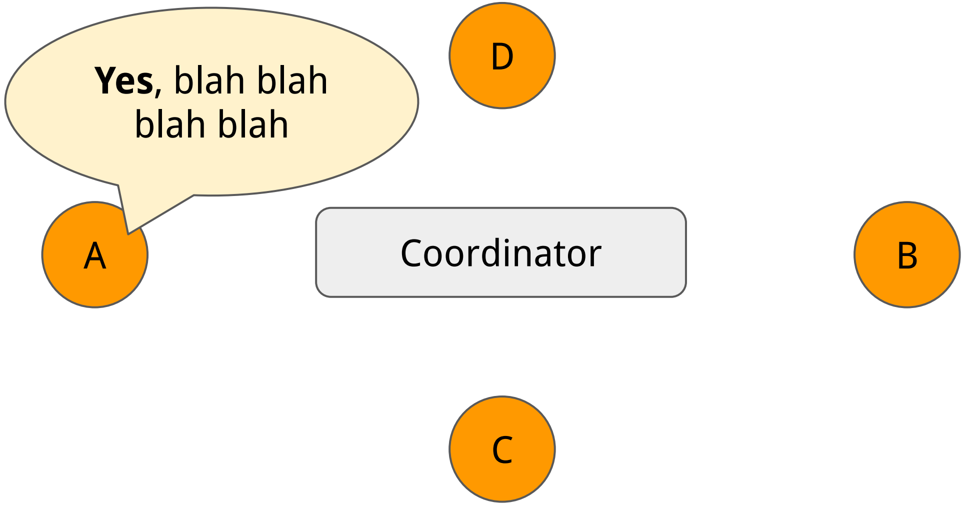
Turn-taking Schemes

- Polling protocols.
  - A coordinator decides when each connected node can speak.
  - e.g., Bluetooth



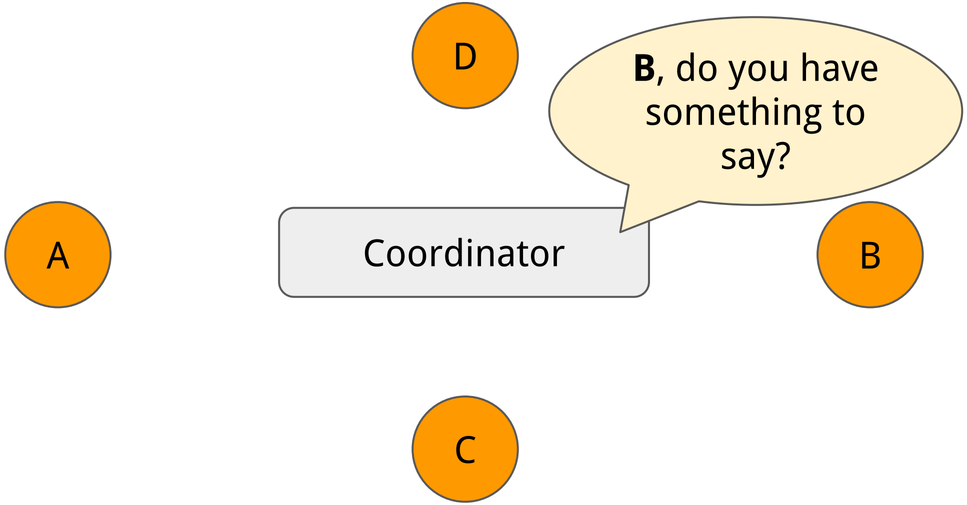
Turn-taking Schemes

- Polling protocols.
  - A coordinator decides when each connected node can speak.
  - e.g., Bluetooth



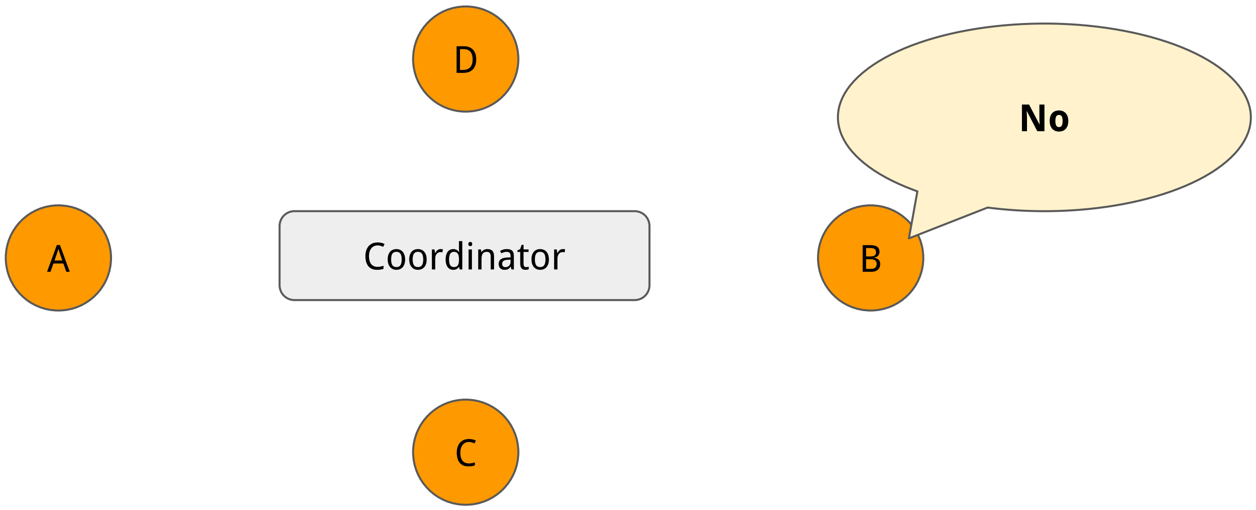
Turn-taking Schemes

- Polling protocols.
  - A coordinator decides when each connected node can speak.
  - e.g., Bluetooth



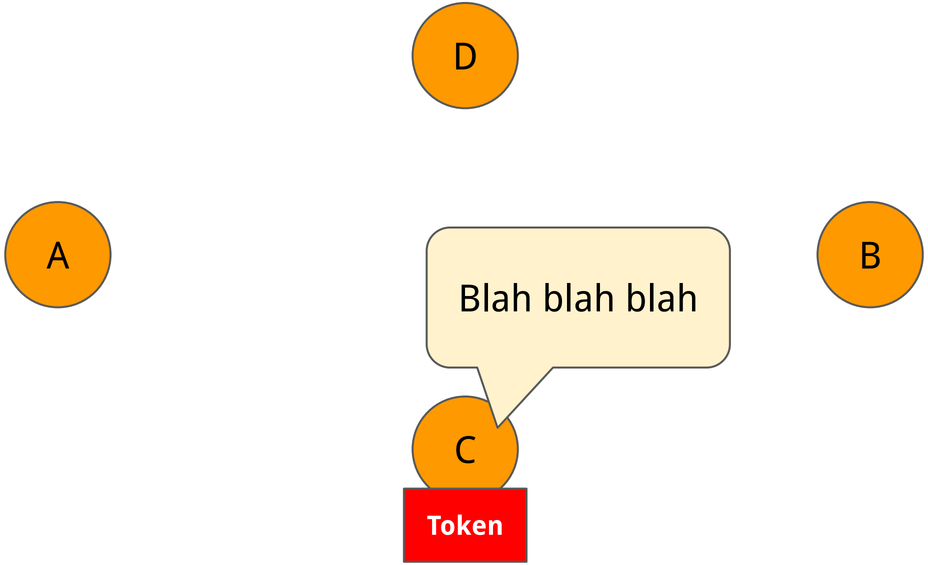
Turn-taking Schemes

- Polling protocols.
  - A coordinator decides when each connected node can speak.
  - e.g., Bluetooth



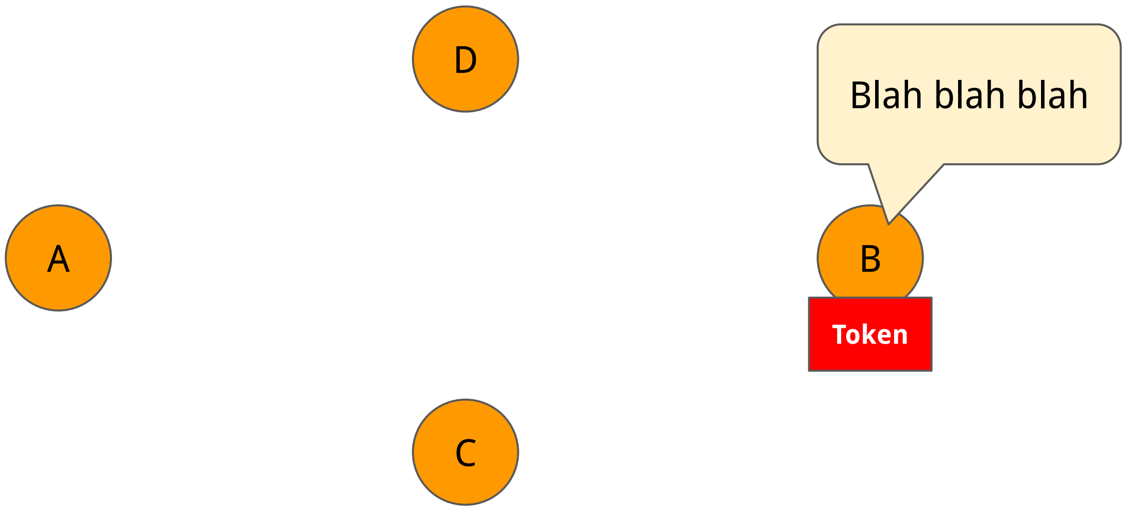
Turn-taking Schemes

- Token-passing
  - Virtual “token” passed around, only the holder can transmit.
  - IBM Token Ring and FDDI.



Turn-taking Schemes

- Token-passing
  - Virtual “token” passed around, only the holder can transmit.
  - IBM Token Ring and FDDI.



Alternative – *Random Access*

- Both of these mechanisms are **partitioning approaches**.
  - Essentially, we are dividing by time – but dynamically.
  - Require some form of inter-node communication.

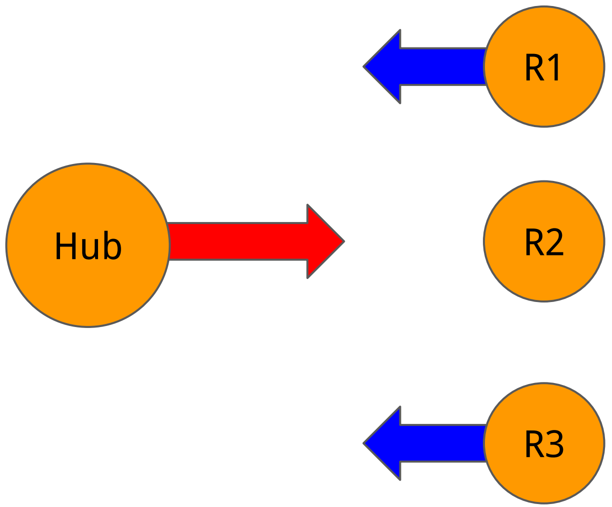
- An alternate idea – just allow for nodes to talk when they have something to say.
  - And deal with collisions when they occur.

- Used by ALOHANet and then later in Ethernet.

ALOHANet's Random Access

- Hub node on Oahu.
- Remote nodes across Hawaii.

- Used two frequencies:
    - **Hub transmits on its own frequency.**
        - Only one sender – no collisions.
        - All remote nodes listen to this frequency.
    - **All remote sites transmit on one frequency.**
        - May collide.
        - Only the hub listens to the remote frequency.



ALOHANet: Pure ALOHA random access scheme

- If remote has a packet – just send it.
  - No *a priori* coordination among remote sites.

- When the hub gets a packet – send ACK.

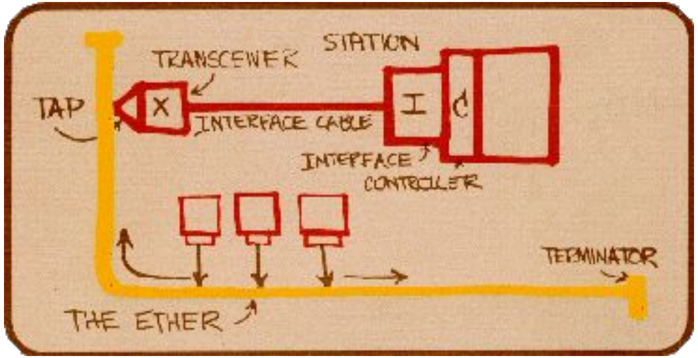
- If two remote sites transmitted at once, collisions results in a corrupted packet.
  - Hub doesn’t ACK!

- If a remote sender doesn’t get the expected ACK – then:
  - Wait a random amount of time.
  - Then resend, probably avoiding collisions this time.

Questions?

Ethernet

- Invented in 1973 at the Palo Alto Research Center (PARC).
- Originally aimed to allow computers to share printers and files.
- Has continued to be iterated on for the last 50 years.
    - Speeds in 1980 were 10**M**bps.
    - Speeds in 2024 are 800**G**bps.



https://www.ieee802.org/3/ethernet_diag.html

Ethernet and CSMA

- Ethernet – used as the most common wired *Data Link* protocol.

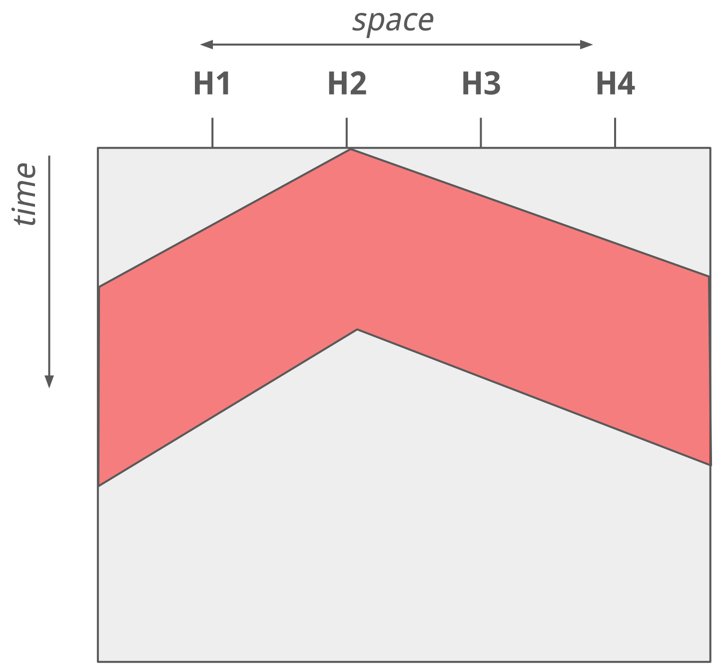
- Refined the ALOHA multiple access protocol to allow access to a shared Ethernet bus resulting in ***Carrier Sense Multiple Access*** **(CSMA).**

- Where ALOHA is rude, CSMA is polite.
    - Rather than just starting talking, and dealing with collisions…
    - CSMA listens first, and then starts to talk when it is quiet.
    - “Listen” means sensing the signal (carrier) on the shared medium.

Ethernet: CSMA and propagation delay

- CSMA does not necessarily avoid collisions – because of **propagation delay**.

- t=0:
    - H2 transmits.
    - Signal propagates through the shared media.



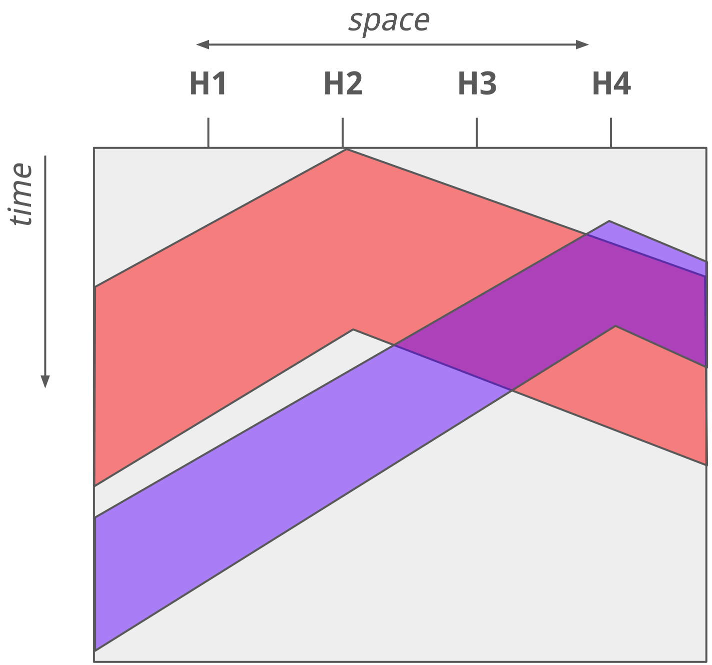
Ethernet: CSMA and propagation delay

- CSMA does not necessarily avoid collisions – because of **propagation delay**.

- t=0:
  - H2 transmits.
  - Signal propagates through the shared media.

- t=2:
  - H3 has heard, won't transmit.
  - H4 has not heard – it's safe to transmit!
    - Signal propagates as time goes by
    - …and collides with H2's signal.

- Solution: CSMA/CD.



Ethernet: CSMA/CD

- *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD).
- Modification to the previous approach:
  - Listen <u>whilst</u> you talk.
  - If you start hearing something whilst you are still transmitting **– stop!**
    - Hence - detect the collision.
- Some additional complexities – but this is the core idea.
- What do we do after detecting a collision?

Ethernet: CSMA/CD

- After collision – wait a random amount of time and retransmit.

- If the link has many senders who want to talk (has high contention) we may keep colliding.

- Use randomised *binary exponential backoff…*
    - If retransmit after collision also collides, wait up to twice as long.
    - Continue doubling for every subsequent collision.
    - Retransmits fast when possible, slowing down where necessary.

Questions?

Forwarding & Ethernet Continued

Fall 2024
cs168.io

Rob Shakir

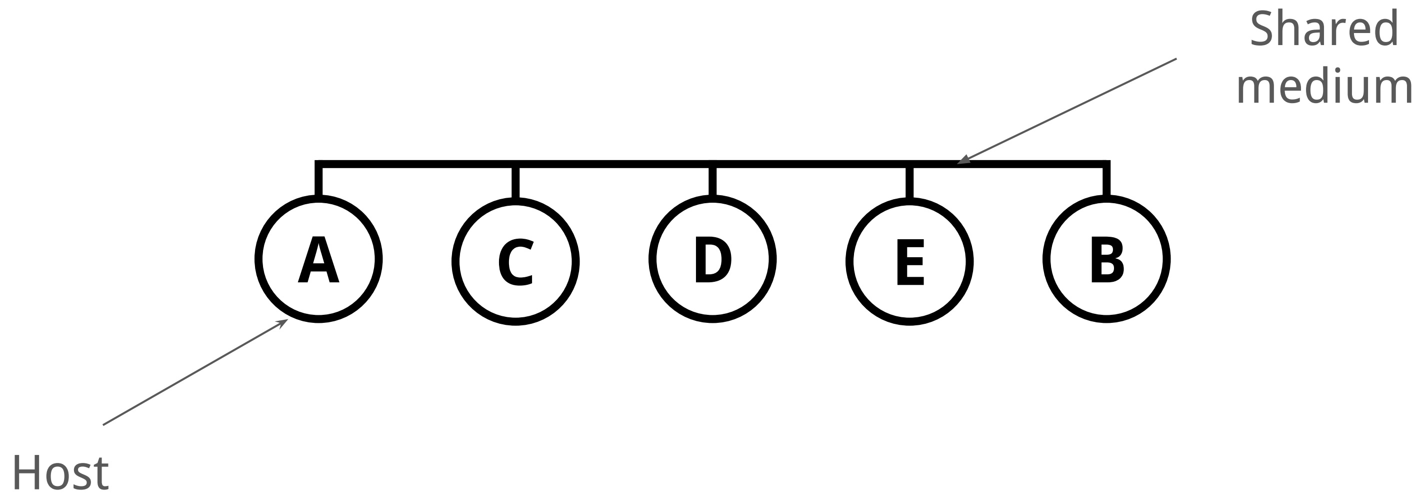
Thanks to Murphy McCauley for some of the material!

Recap

- Started to think about Layer 2 networking.

Recap

- Started to think about Layer 2 networking.



Transmission onto a Shared Medium

- We discussed that hosts need to have a means to determine <u>how</u> to share the medium.
  - We can use strict partitioning approaches – by time or frequency.
  - Or more dynamic ways to share time.

- Dynamic mechanisms can either:
  - Have inter-host messaging (e.g., a coordinator, or a token) to avoid collisions.
  - Or use a means to deal with collisions when they do occur.

- Ethernet uses *Carrier Sense Multiple Access with Collision Detection.*
  - Which allows a host to both detect when another host is transmitting
  - And detect when collisions occur – and back off.

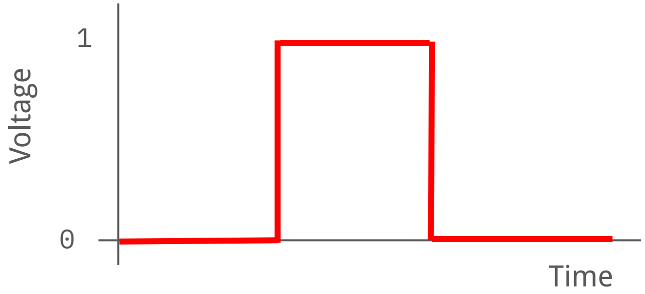
Clarifying L1 and L2

- Questions after last lecture – how does this relate to packets?
- Let’s look at what Layer 1 and Layer 2 are.

Clarifying L1 and L2

- Questions after last lecture – how does this relate to <u>packets</u>?

- Let's look at what Layer 1 and Layer 2 are.

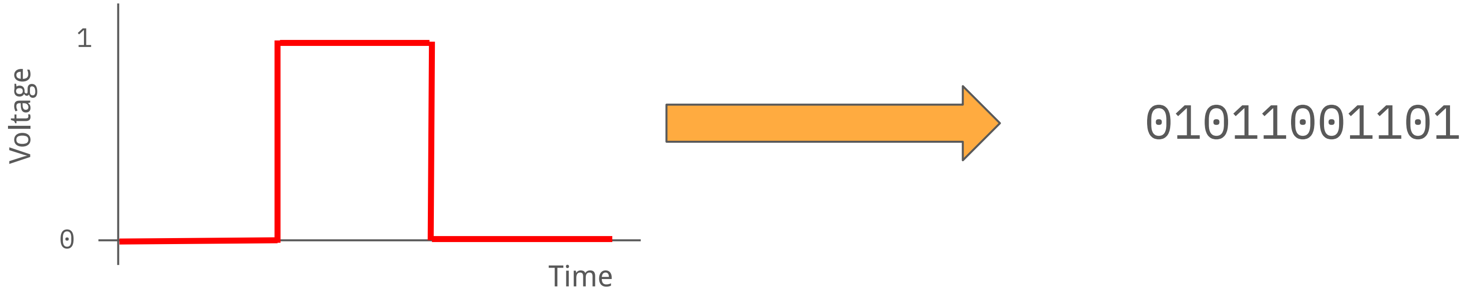


- L1: How to send data onto a particular medium (fibre, copper, radio).

- How to take some data and turn it into a signal that can be interpreted at the other end of the connection.

Clarifying L1 and L2

- Questions after last lecture – how does this relate to <u>packets</u>?
- Let's look at what Layer 1 and Layer 2 are.



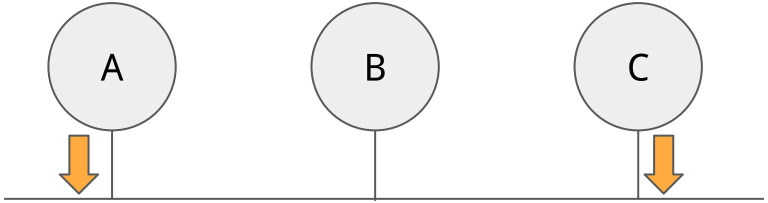
We can think of the "output" of Layer 1 as a string of bits.

Clarifying L1 and L2

- Layer 2 then takes this string of bits and makes it into a *packet*.

Simple host-to-host.

If A transmits to B ⇒ string of bits is a valid packet (using the same parsing rules).



Shared medium.

Collisions can occur – string of bits might not be a valid packet.

Layer 2 defines the rules for how to deal with this (e.g., CSMA/CD).

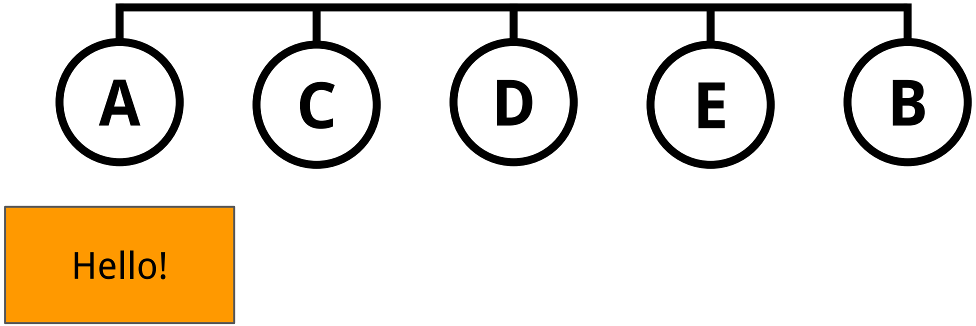
Clarifying L1 and L2

- Layer 2 protocols such as Ethernet define how we deal with sending and receiving a string of bits, <u>independently</u> of what the underlying Layer 1 protocol looks like.

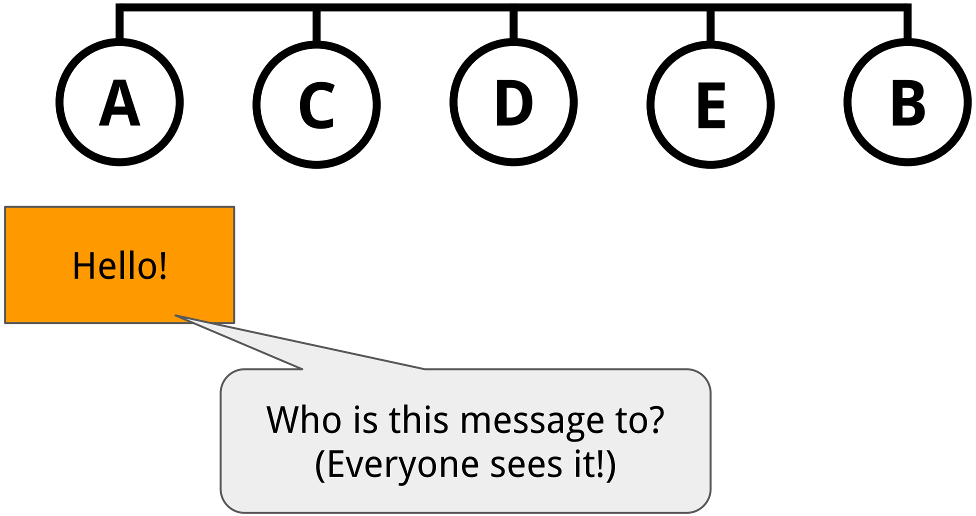
- The input to Layer 2 is a set of bits, that we then parse into an Ethernet *frame.*

Questions?

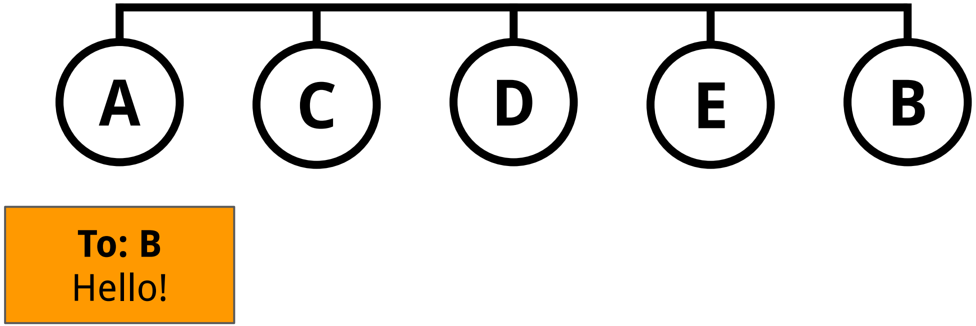
Sending messages between computers on an Ethernet.



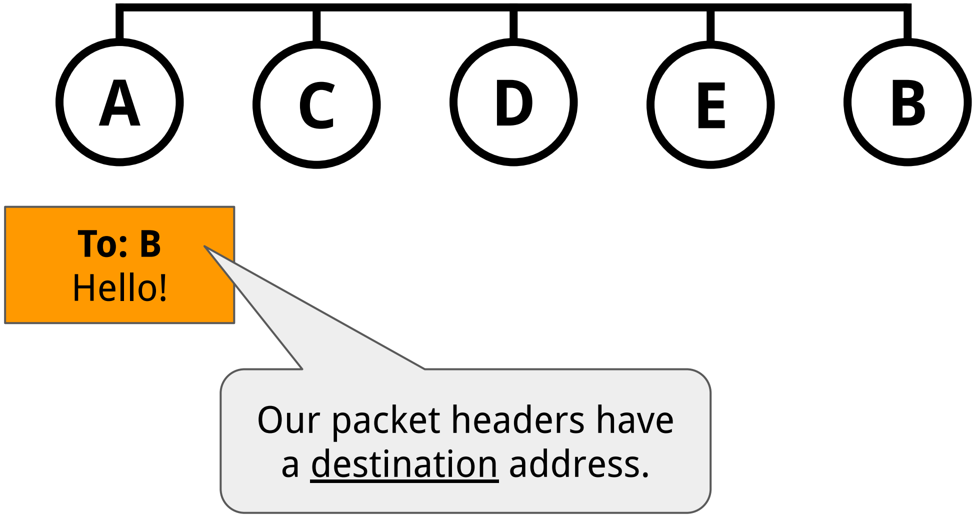
Sending messages between computers on an Ethernet.



Sending messages between computers on an Ethernet.



Sending messages between computers on an Ethernet.



Recall from earlier, Packets

- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain...

Metadata (headers)					Data/Payload
Src Addr	**Dst Addr**	**Type**	**Version**	**...**	**<html><head><title>My Website</title><head> ...**

Recall from earlier, Packets

- Packet has…
    - Payload (the actual data)
    - Headers (metadata)
        - Must* contain a **destination address**.

Metadata (headers)					Data/Payload
Src Addr	**Dst Addr**	**Type**	**Version**	**…**	**<html><head><title>My Website</title><head> …**

Recall from earlier, Packets

- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain a **destination address**.
      - …which implies that **a host has an address**!
        - Or more than one! (Why?)

Metadata (headers)					**Data/Payload**
Src Addr	**Dst Addr**	**Type**	**Version**	**...**	**<html><head><title>My Website</title><head> ...**

Recall from earlier, Packets

- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain a **destination address**.
      - …which implies that **a host has an address**!
        - Or more than one! (Why?)
        - **For now, one address per host.**

Metadata (headers)					Data/Payload
Src Addr	Dst Addr	Type	Version	...	<html><head><title>My Website</title><head> ...

Ethernet Addressing

- If I send a signal (shout in this room) – everyone gets the message.

- But we do want some way to be able to identify the destination of a particular message.
  - e.g., just talk to one person in the room – not talk to everyone!

- We therefore need some form of addressing to be able to identify different hosts connected to the same medium.
  - Like we would use a *name* within this room to talk to one another.

Ethernet: Addresses

- Ethernet has Media Access Control (MAC) addresses.
  - These are Layer 2 addresses – we don't need to know anything about what is inside the Ethernet *Frame* (i.e., it doesn't matter whether it's IPv4, IPv6, or even IP at all!)
  - We'll talk about what can come inside this frame — at the moment it is a bunch of bytes.

- MAC addresses are 48-bits.
  - Usually shown as six two-digit hex numbers with colons.
  - Sometimes referred to as `ether` or `link` addresses.

```
▸ ifconfig en0
en0:
flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST>
mtu 1500
        options=400<CHANNEL_IO>
        ether f8:ff:c2:2b:36:16
```

```
rjs@jumphost:~$ ip link show ens4
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460
qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 42:01:0a:8a:00:03 brd ff:ff:ff:ff:ff:ff
    altname enp0s4
```

Ethernet: MAC Addresses

- MAC addresses are allocated according to organisation.
  - Usually the manufacturer of the Ethernet network interface card (NIC).
- Typically stored permanently in the NIC (“burned in”) .
  - Often can be overridden by software.
- Structure:
  - Two bits of flags (we won’t discuss this)
  - 22-bits identifying company/organisation (e.g., device manufacturer)
  - 24-bits of identifying space.
- Usually supposed to be globally unique.
  - You might plug your computer in *anywhere*…

Ethernet: Types of communication

- We will typically talk about **unicast.**
    - Send to any one recipient.

- There are other models that we might care about:
    - Speak to everyone in the room – **broadcast**.
    - Speak to everyone who has joined a group in the room – **multicast**.
    - Send to any one member of a group - **anycast.**

- Ethernet supports both multicast and broadcast.
    - And generally they are not distinguished from each other at the Ethernet level.
    - We will briefly cover these if we get time.

Questions?

Ethernet: Unicast

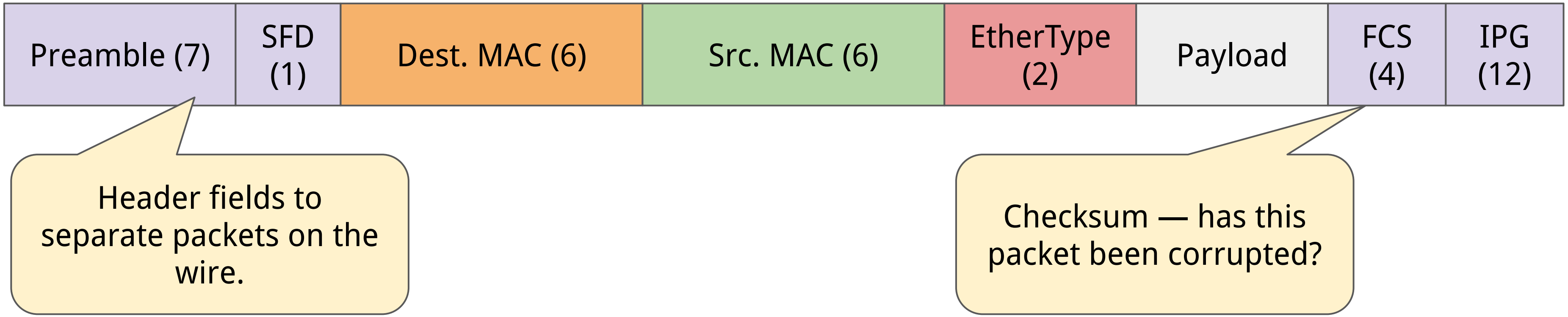
- Unicast is the typical type of communication used on the Internet.
  - A source host wants to talk to a specific destination host.

- The Ethernet header has the fields that allow for unicast forwarding.
  - A data packet in Ethernet is referred to as a *frame*.

Preamble (7)	SFD (1)	Dest. MAC (6)	Src. MAC (6)	EtherType (2)	Payload	FCS (4)	IPG (12)

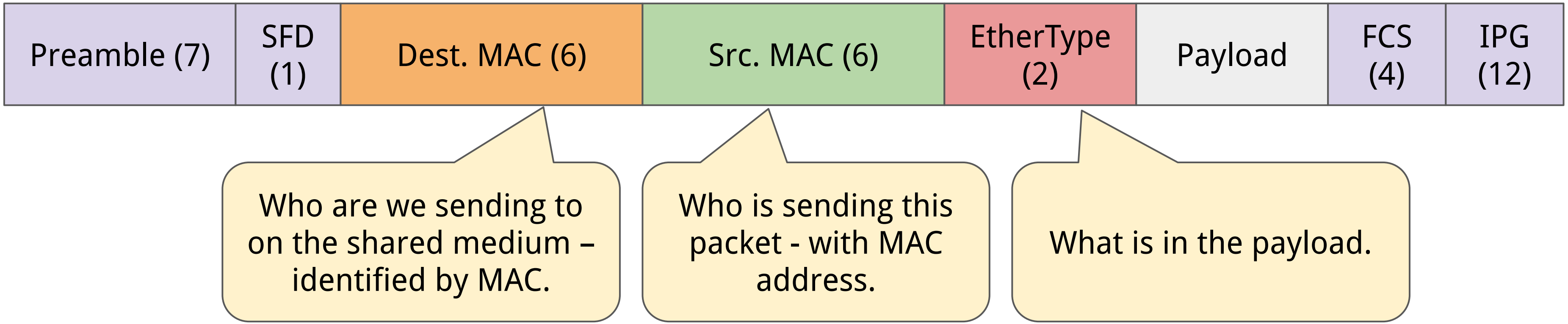
Ethernet: Unicast

- Unicast is the typical type of communication used on the Internet.
  - A source host wants to talk to a specific destination host.
- The Ethernet header has the fields that allow for unicast forwarding.
  - A data packet in Ethernet is referred to as a *frame*.



Ethernet: Unicast

- Unicast is the typical type of communication used on the Internet.
  - A source host wants to talk to a specific destination host.
- The Ethernet header has the fields that allow for unicast forwarding.
  - A data packet in Ethernet is referred to as a *frame*.



Ethernet: Unicast

- To send a packet to a specific destination host - we set the destination MAC to a specific remote machine’s MAC address.
- Packets go to everyone on the shared medium (wire).
- Receivers check the destination MAC to determine whether the packet is destined to them.
    - “Is dst MAC == `42:01:0a:8a:00:03`? It’s for me!”

Ethernet: Broadcast

- Broadcast – send to everyone!
  - Specifically, everyone on the specific Ethernet network…
  - …everyone on the same cable.

- The packet already reaches everyone – they are connected to the *shared media.*
  - We need receivers to listen.

- Broadcast is implemented using the all ones address.
  - `FF:FF:FF:FF:FF:FF`

Ethernet: Multicast

- Multicast – send to all members of a group.
  - Trivial on classic Ethernet – since everyone gets the packet.

- Implemented by having specific addresses – one of the flags in the address set to 1.
  - `01:00:00:00:00:00`
  - Normal addresses all have an even first byte.
  - This 1 is the first bit on the wire – bytes are sent low bit first.

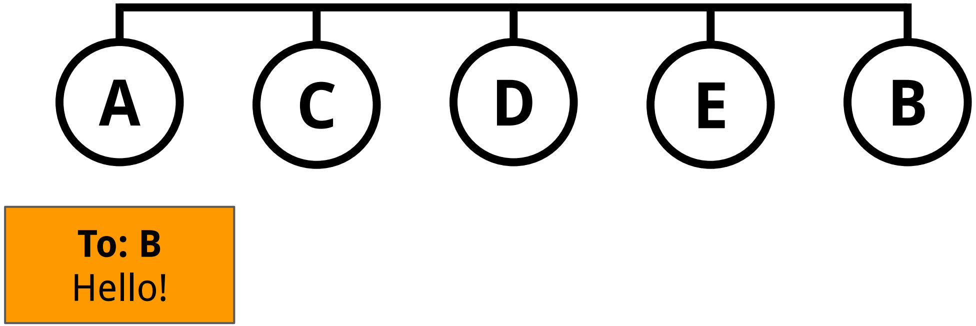
- Broadcast is just a special case of multicast – where everyone is in a group.

Why do we need multicast in a LAN?

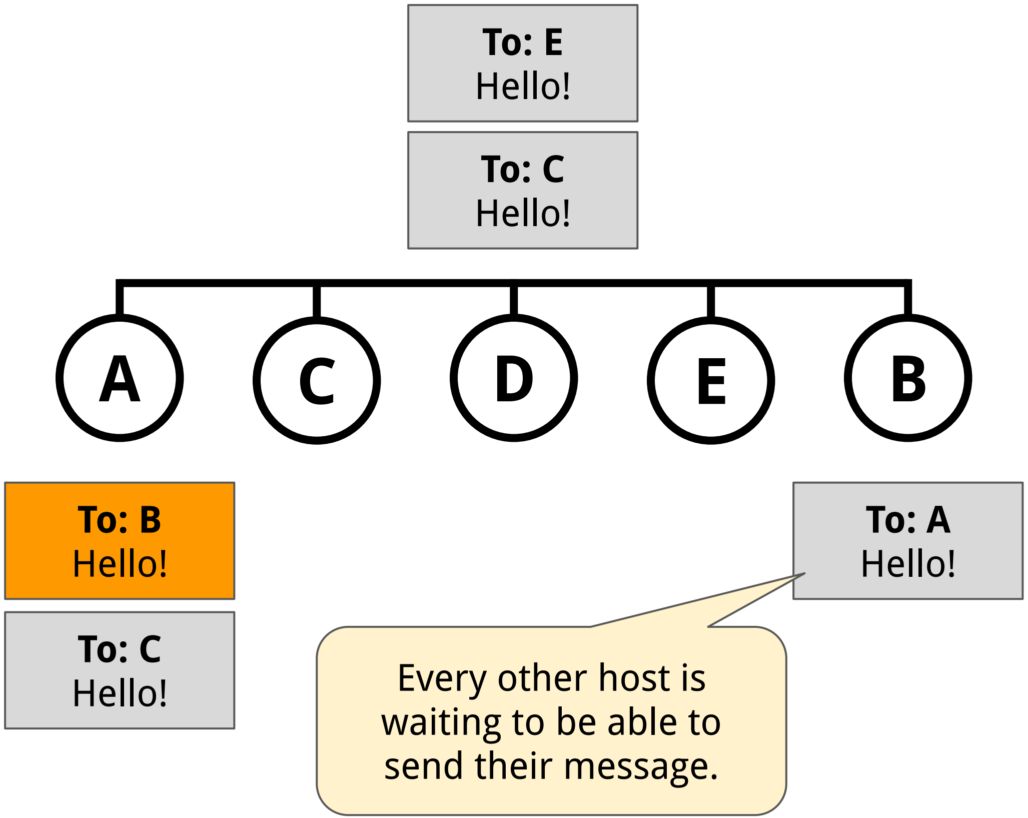
- Apple invention: Bonjour/mDNS.
- iPhone wants to discover any Apple TV, or HomePod that it can play music on.
  - It can actively discover this “hey local Apple products, are there any speakers?”.
  - Sends to a multicast group that all Apple products join by default.
  - Equally, HomePod/Apple TV can advertise “I am an Apple TV!”.
- Actually uses DNS advertisements that are sent to multicast addresses.
  - Using specific types of records – e.g., SRV – to advertise capabilities.

Questions?

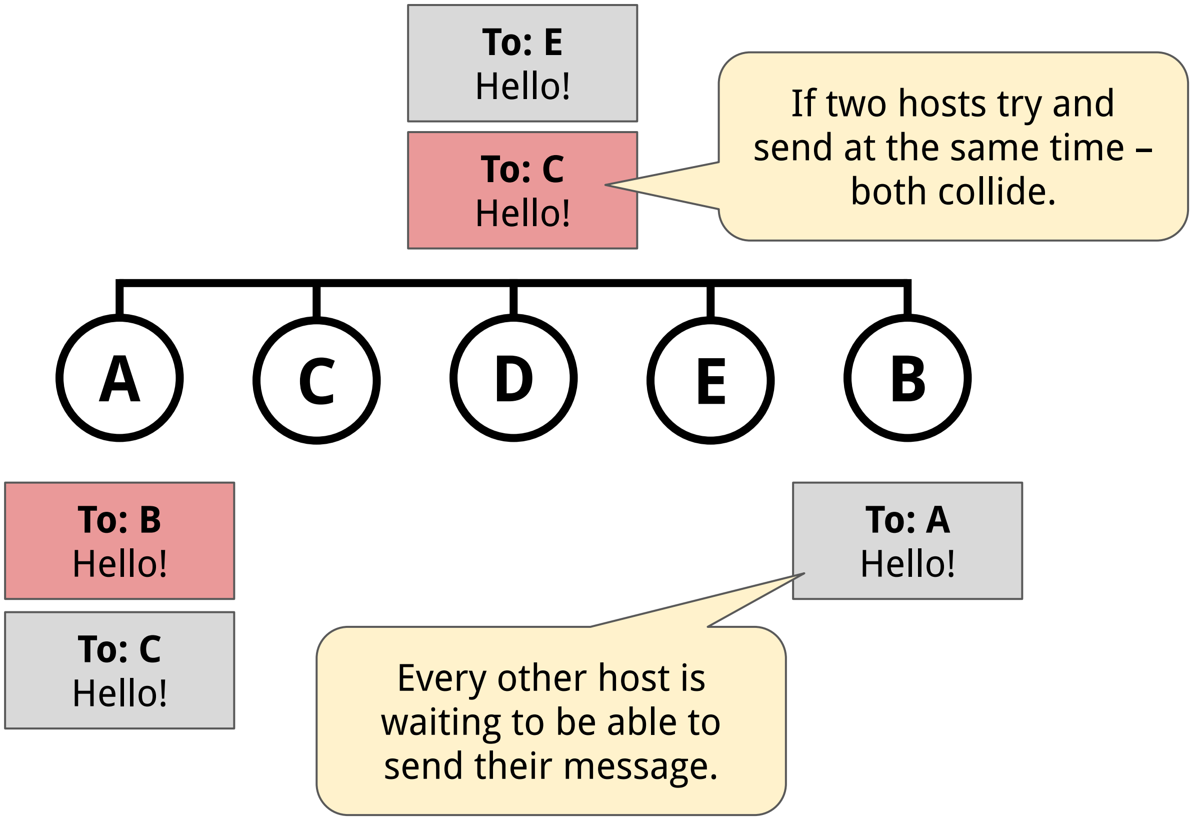
Inefficiencies of a single bus



Inefficiencies of a single bus



Inefficiencies of a single bus



Inefficiencies of a Shared Medium

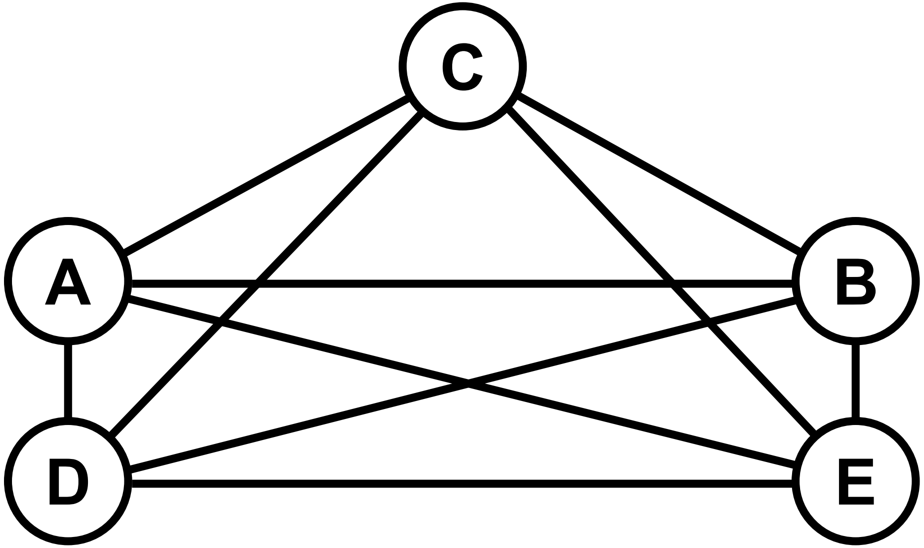
- The best case scenario is that someone is always sending.

- Assuming our shared medium has some bandwidth limit, **B**, this is the best we can do.
  - **B** is dictated based on how we signal packets on the wire.
  - May be do to do with modulation speed, or amount of spectrum available.

- Equally – we need everyone to be connected to the shared medium.
  - Inverse square law – eventually we are not going to be able to propagate the signal!

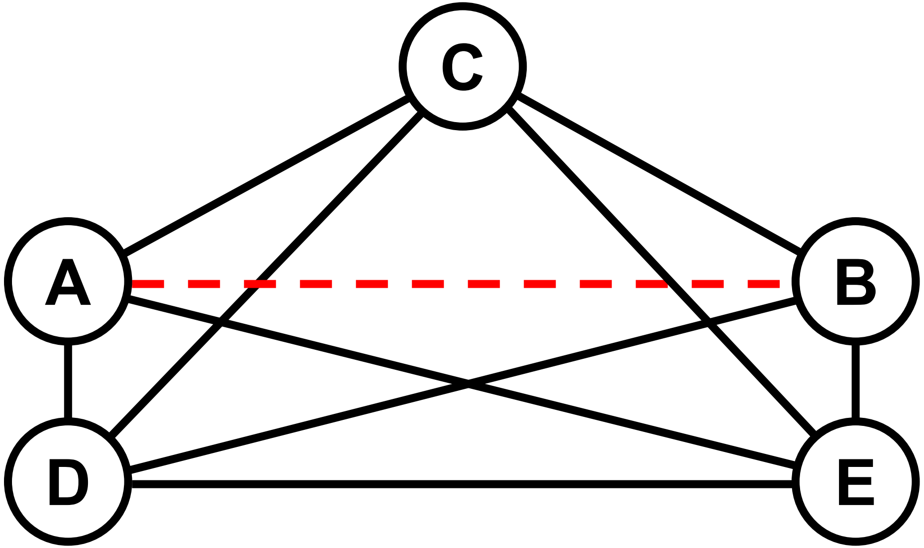
- **So**, can we do better?

Connecting Nodes Together

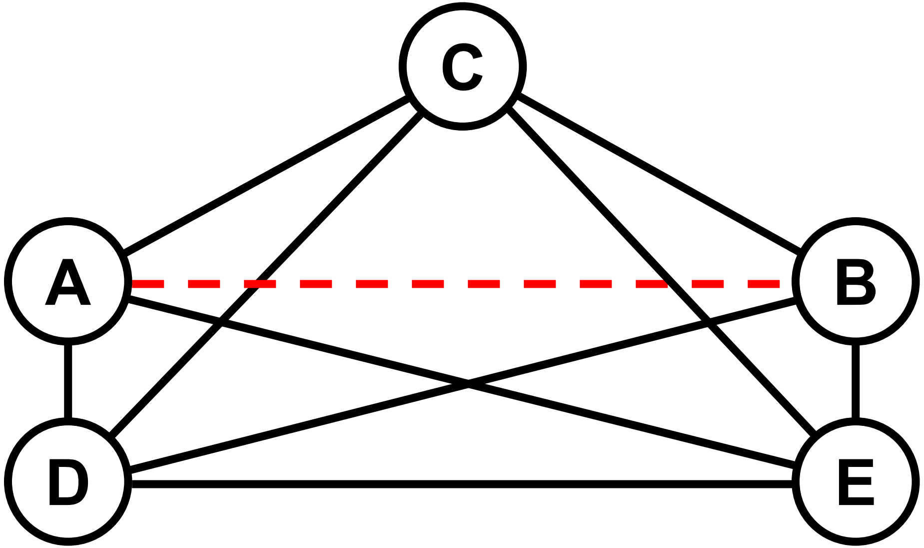


Is there a problem with this approach?

Connecting Nodes Together

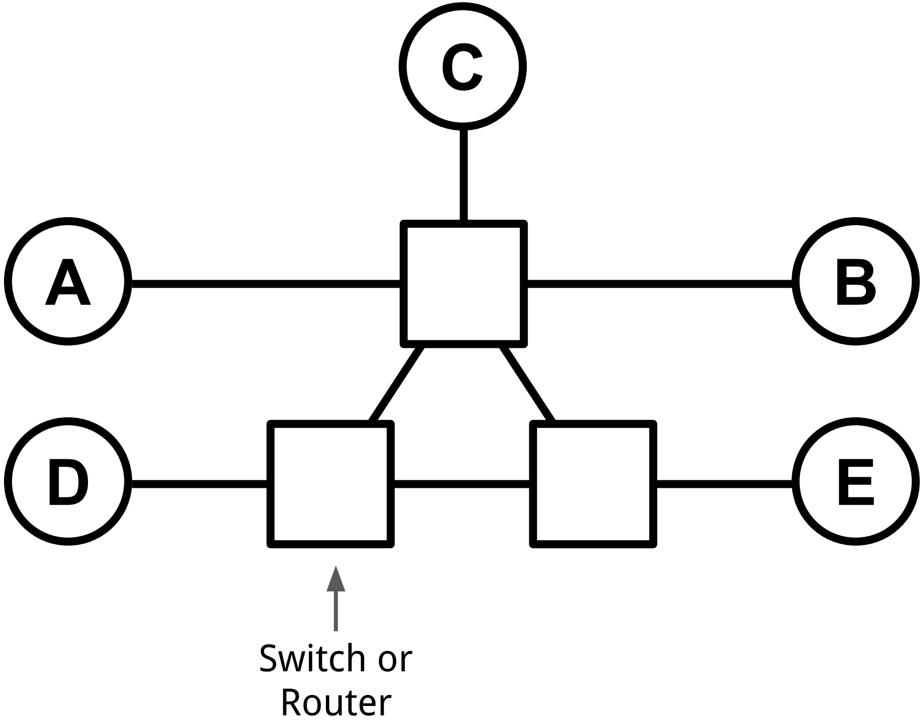


Connecting Nodes Together



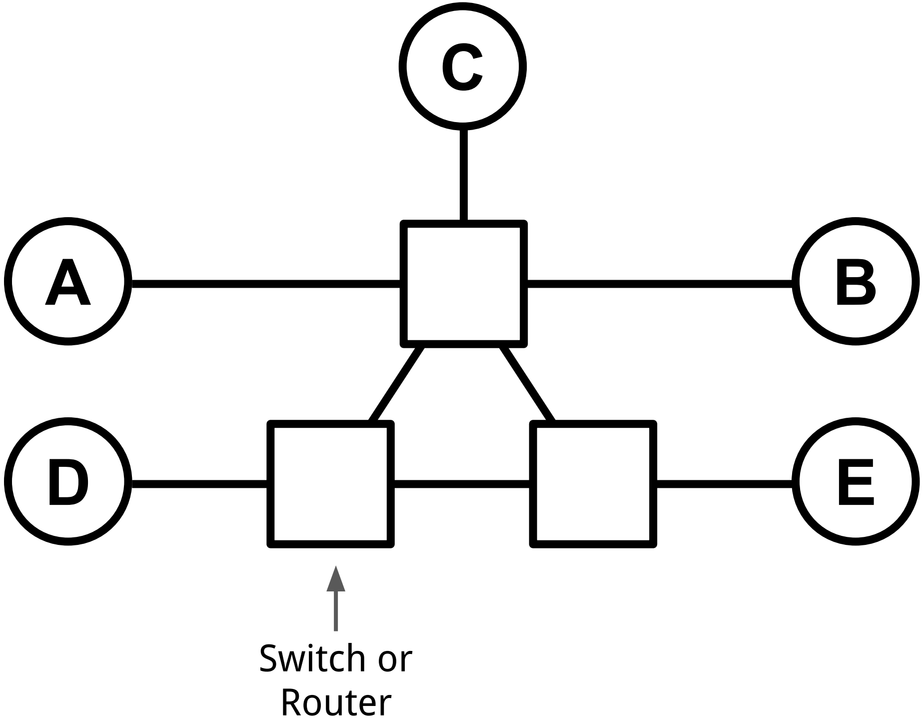
Is there anything *good* about this approach?

Connecting Nodes Together



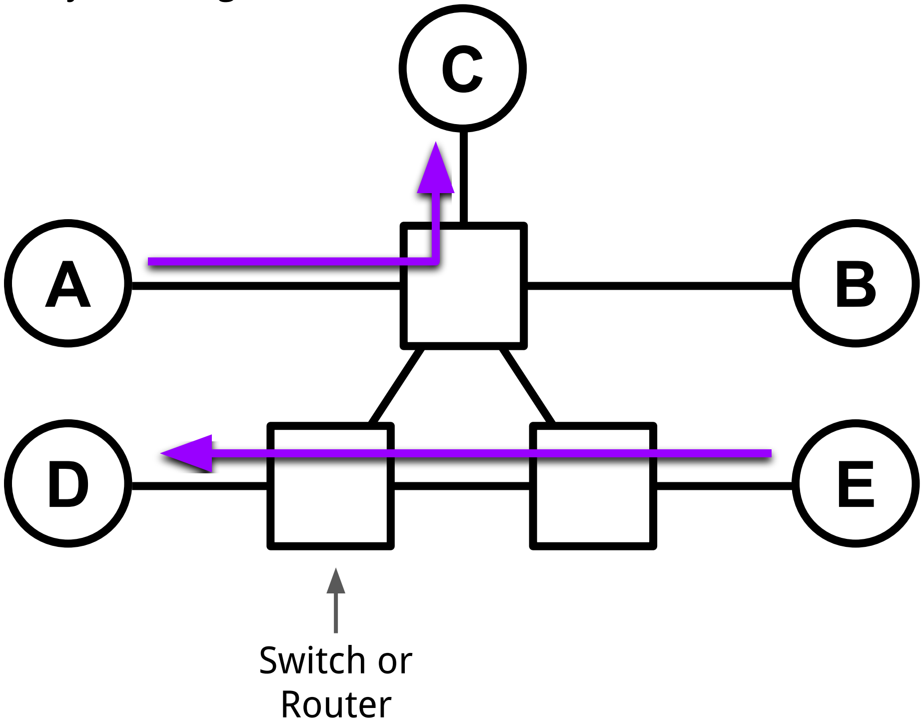
Connecting Nodes Together

- Way fewer links than a full mesh!



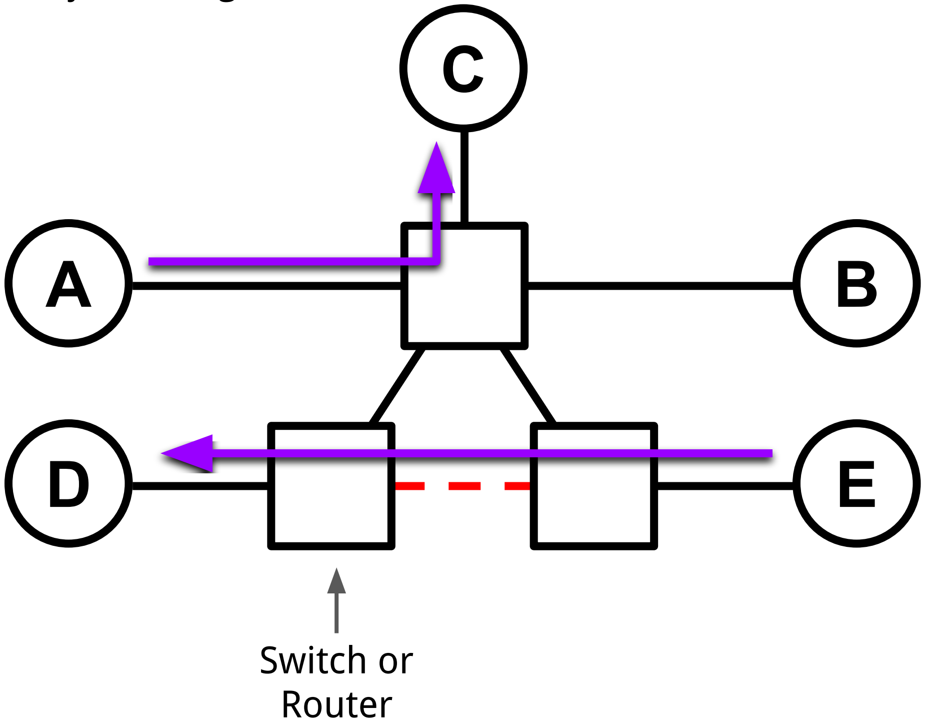
Connecting Nodes Together

- Way fewer links than a full mesh!
- But more capacity than just a single link!



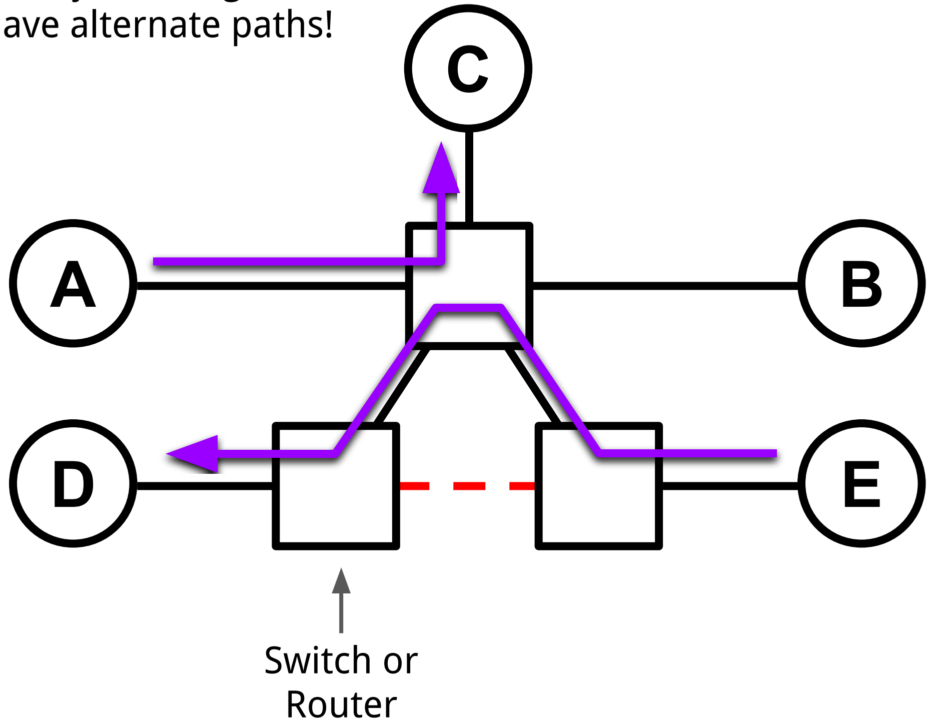
Connecting Nodes Together

- Way fewer links than a full mesh!
- But more capacity than just a single link!



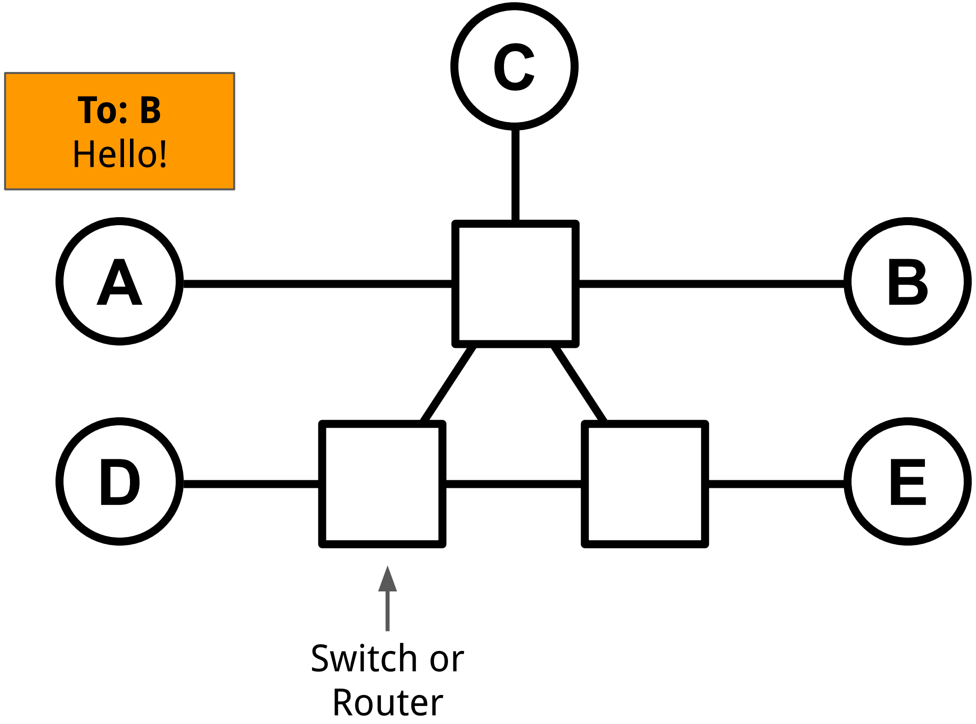
Connecting Nodes Together

- Way fewer links than a full mesh!
- But more capacity than just a single link!
- With the ability to have alternate paths!

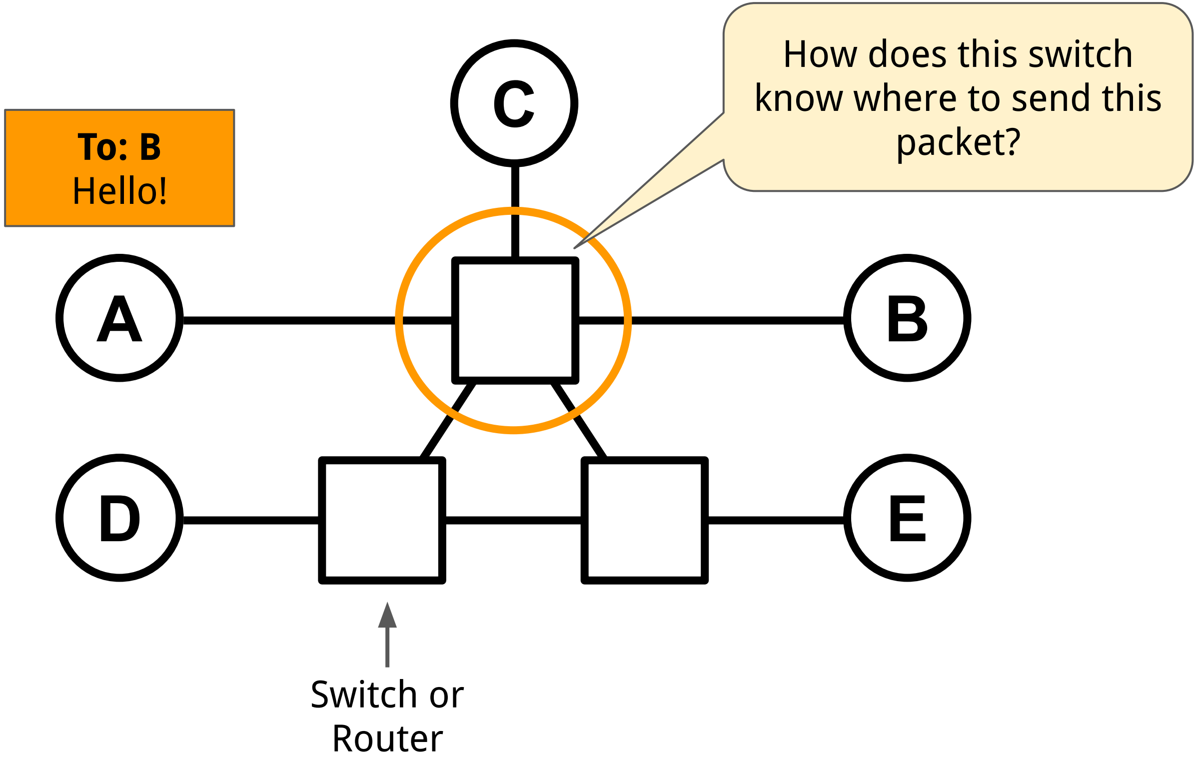


Questions?

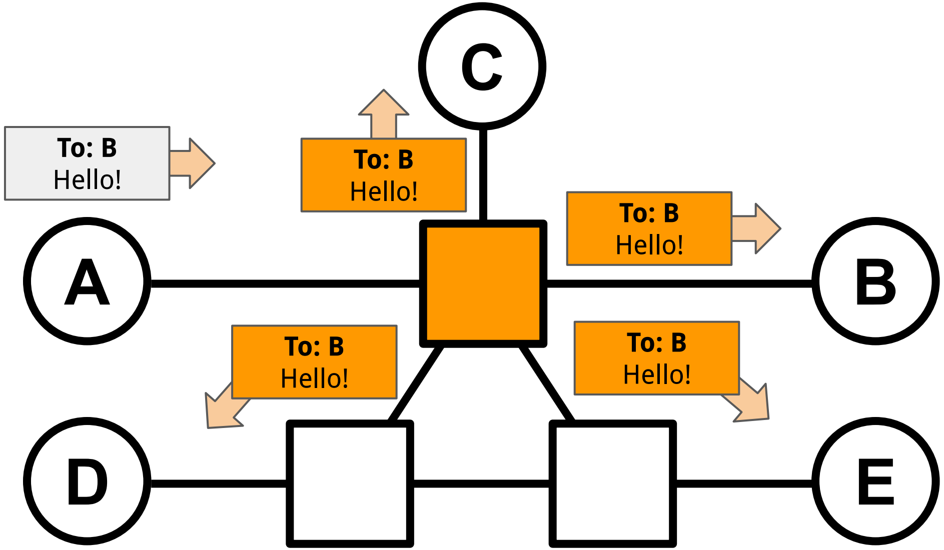
But, we just created a new problem!



But, we just created a new problem!

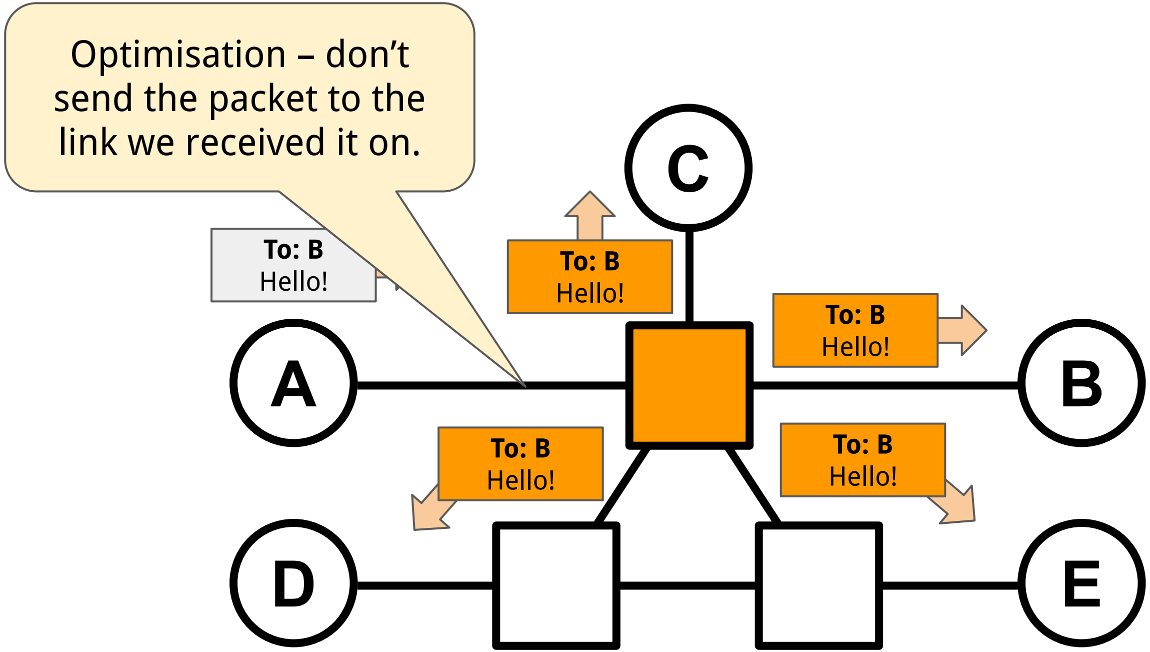


Naïve approach to forwarding



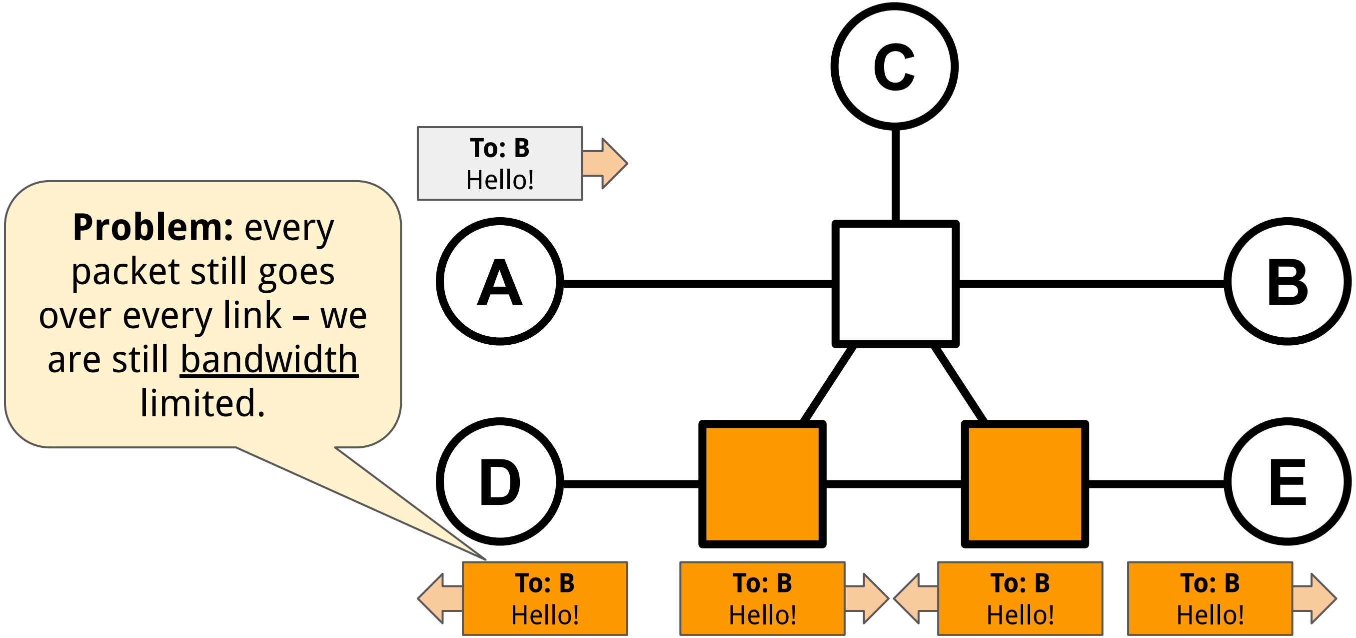
Flooding – send the packet to every port on the switch.

Naïve approach to forwarding

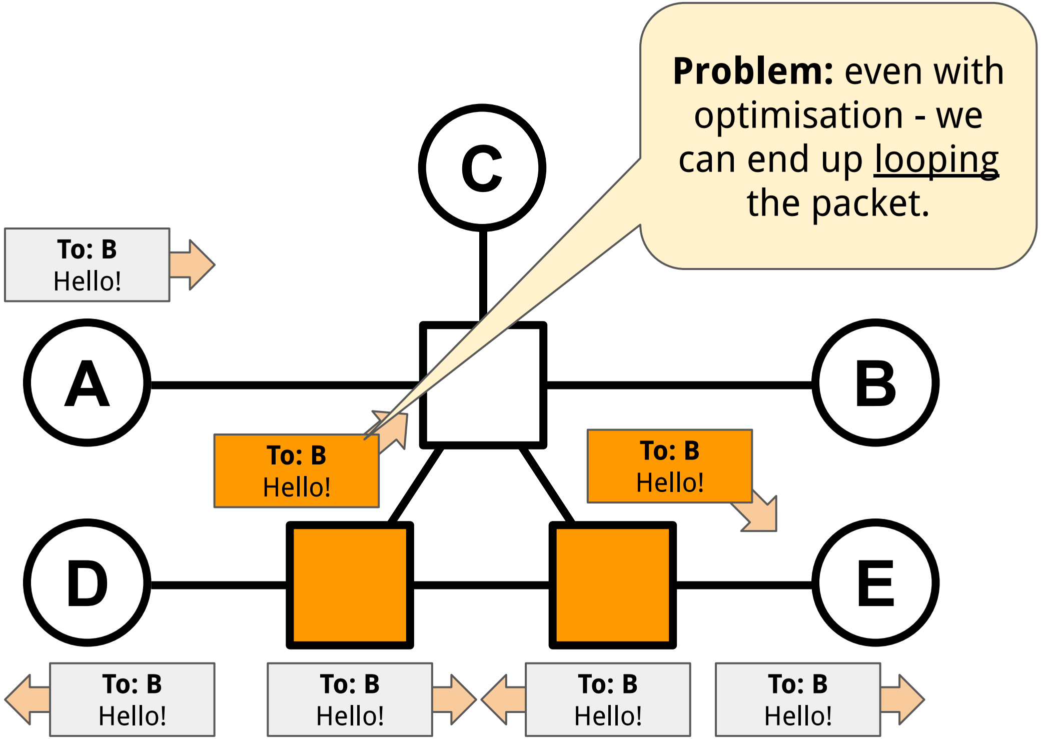


Flooding – send the packet to every* port on the switch.

Naïve approach to forwarding



Naïve approach to forwarding



Questions?

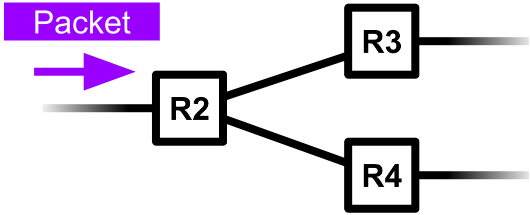
Forwarding challenges

- Our naïve approach is problematic.

- We need to solve two problems:
  - How to avoid wasting bandwidth by sending a packet to everyone even if they are not interested.
  - How to deal with the fact that the *topology* of the network might mean that flooding a packet causes it to be looped.

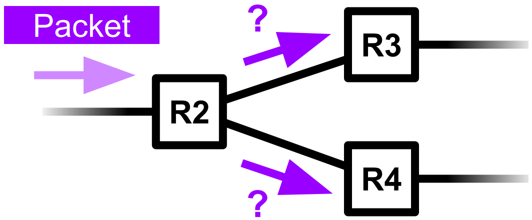
- Let’s start with the first one.

The Challenge of Forwarding



- When packet arrives...

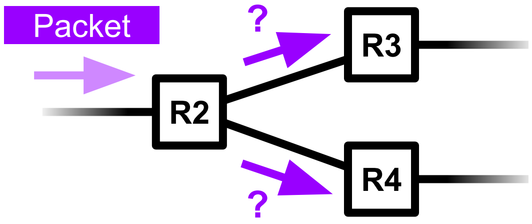
The Challenge of Forwarding



- When packet arrives, switch ***forwards*** it to one of its neighbors

- You need to make the decision about which neighbor *fast* (~nanoseconds)
- Implies the decision process is *simple*

The Challenge of Forwarding

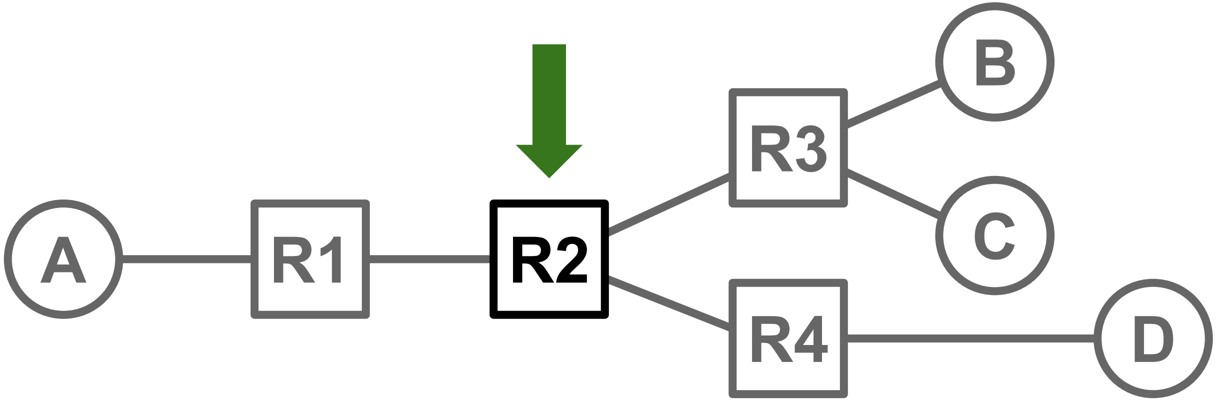


- When packet arrives, switch ***forwards*** it to one of its neighbors

- You need to make the decision about which neighbor *fast* (~nanoseconds)
- Implies the decision process is *simple*

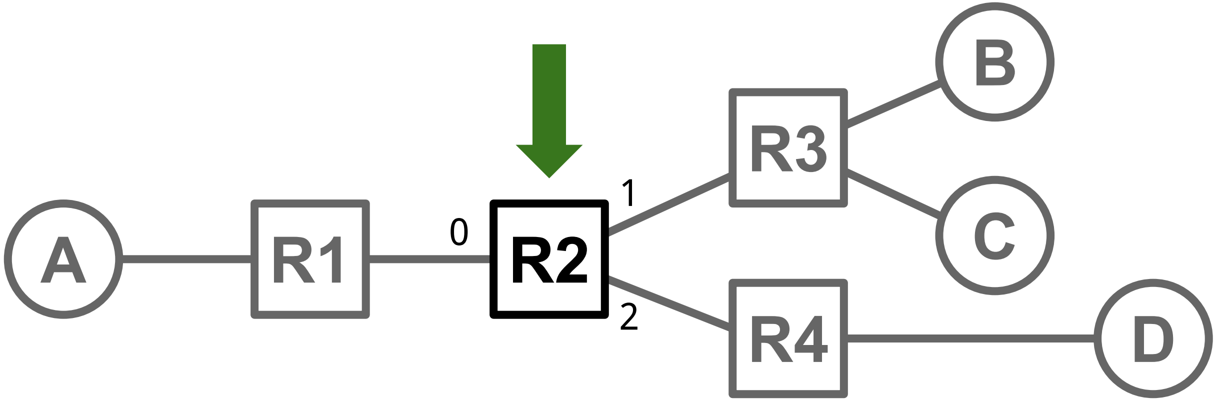
- Solution: Use a table

Forwarding with a Table



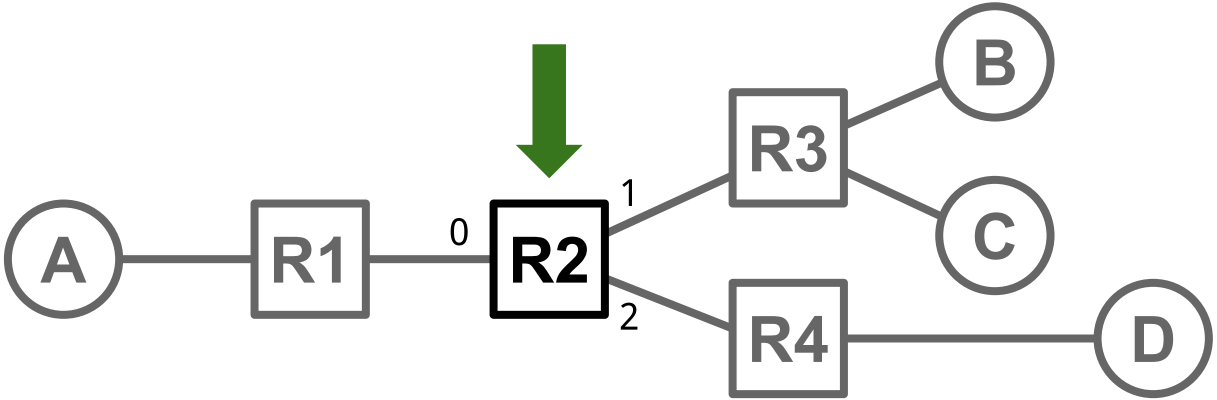
R2's Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

Forwarding with a Table



R2's Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

Forwarding with a Table



R2's Table	
Dst	*NextHop*
A	R1
B	R3
C	R3
D	R4

.. or ..

R2's Table	
Dst	*Port*
A	0
B	1
C	1
D	2

Forwarding with a Table

- Given the tables, decision *depends only on destination field of packet*
- .. we are doing what's called ***destination-based forwarding/routing***
  - Very common
  - An "archetypal" Internet assumption (and basically the default)

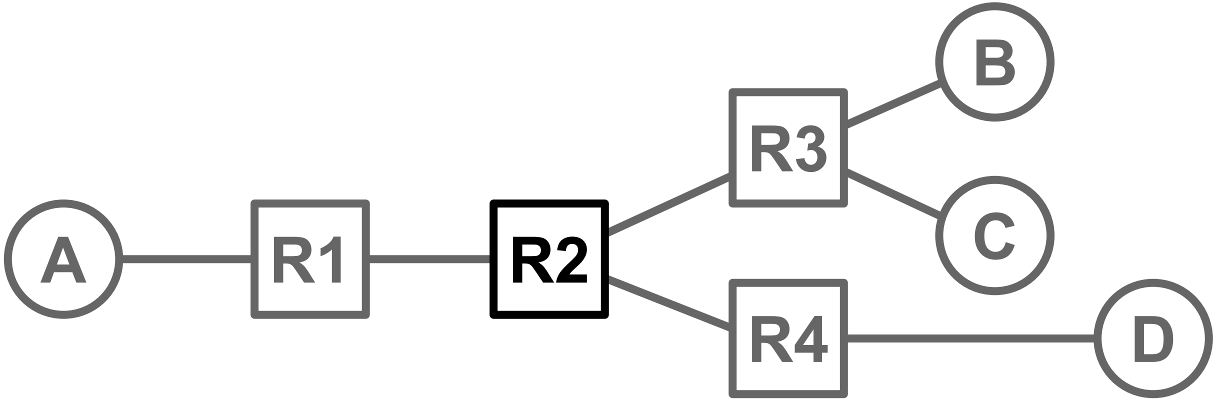
R2's Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

.. or ..

R2's Table	
Dst	***Port***
A	0
B	1
C	1
D	2

Questions?

How do tables get populated?



R2's Table	
Dst	***NextHop***

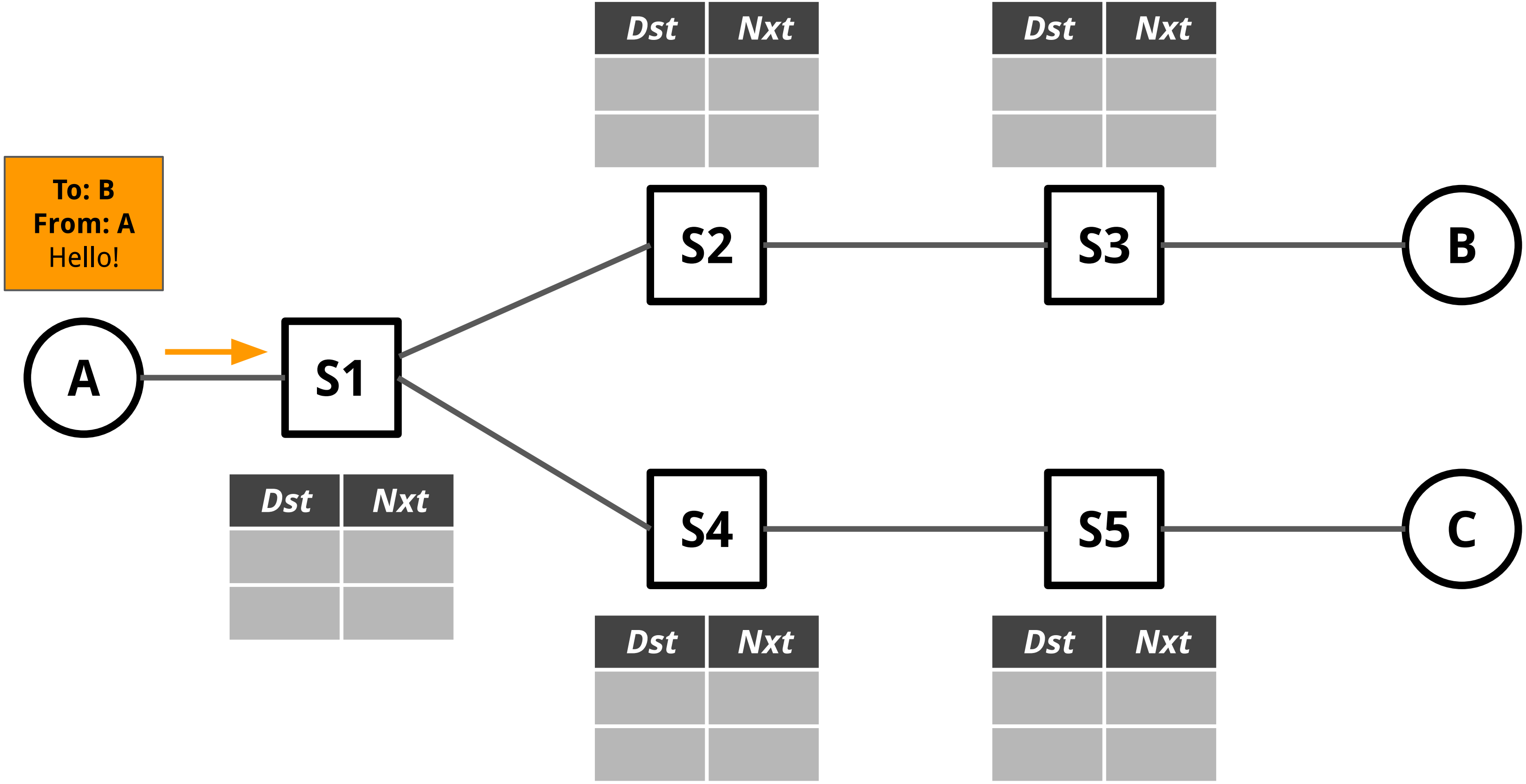
To be able to make our forwarding decision – we need to populate this table.

Approaches to populating forwarding tables

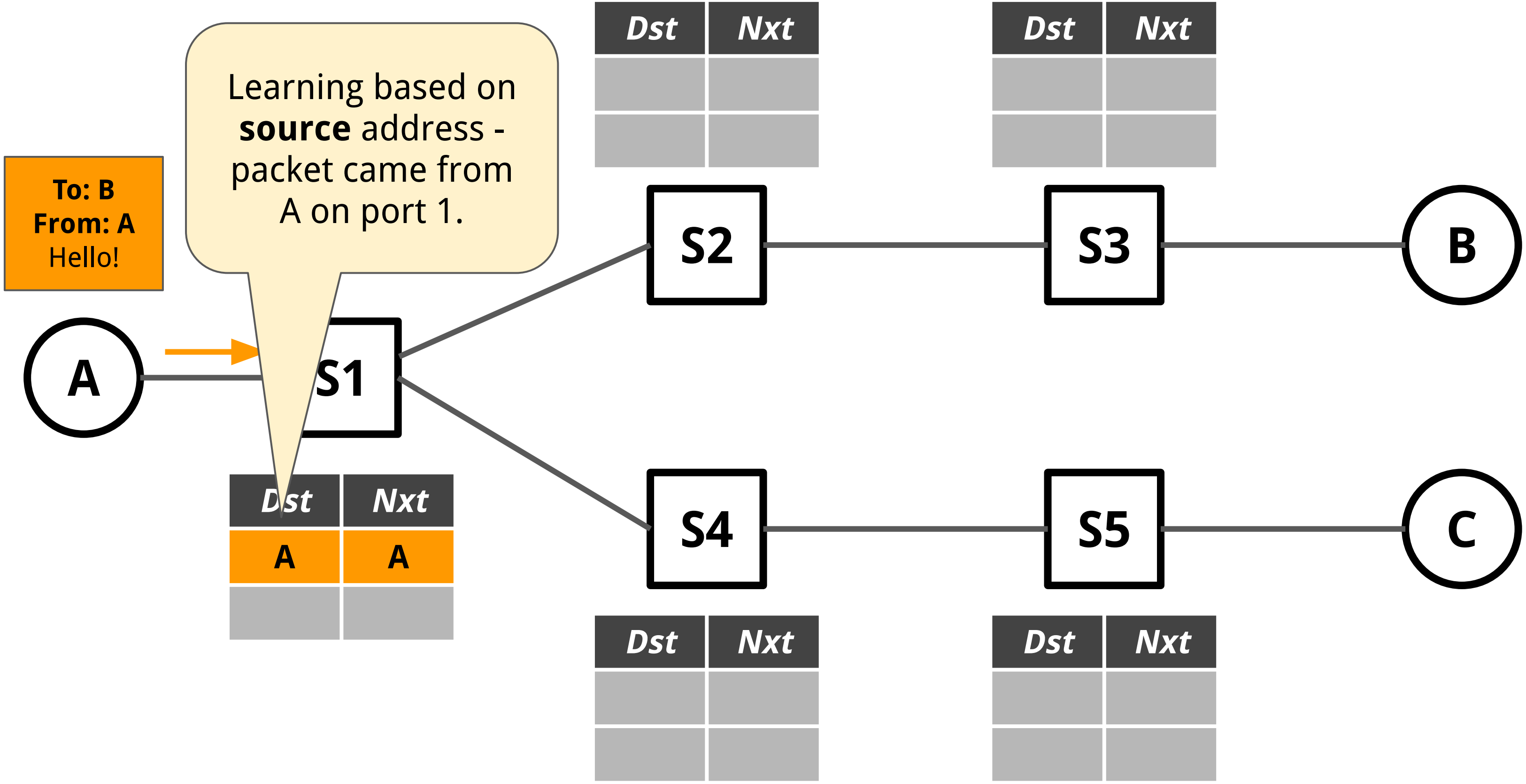
- When a packet comes along – look at where it came from – and use this to help us learn where hosts are connected.

- Very simple!
    - We need to look at the packet that we see on the data plane.
    - No need for any kind of *a priori* knowledge of how the network topology works.

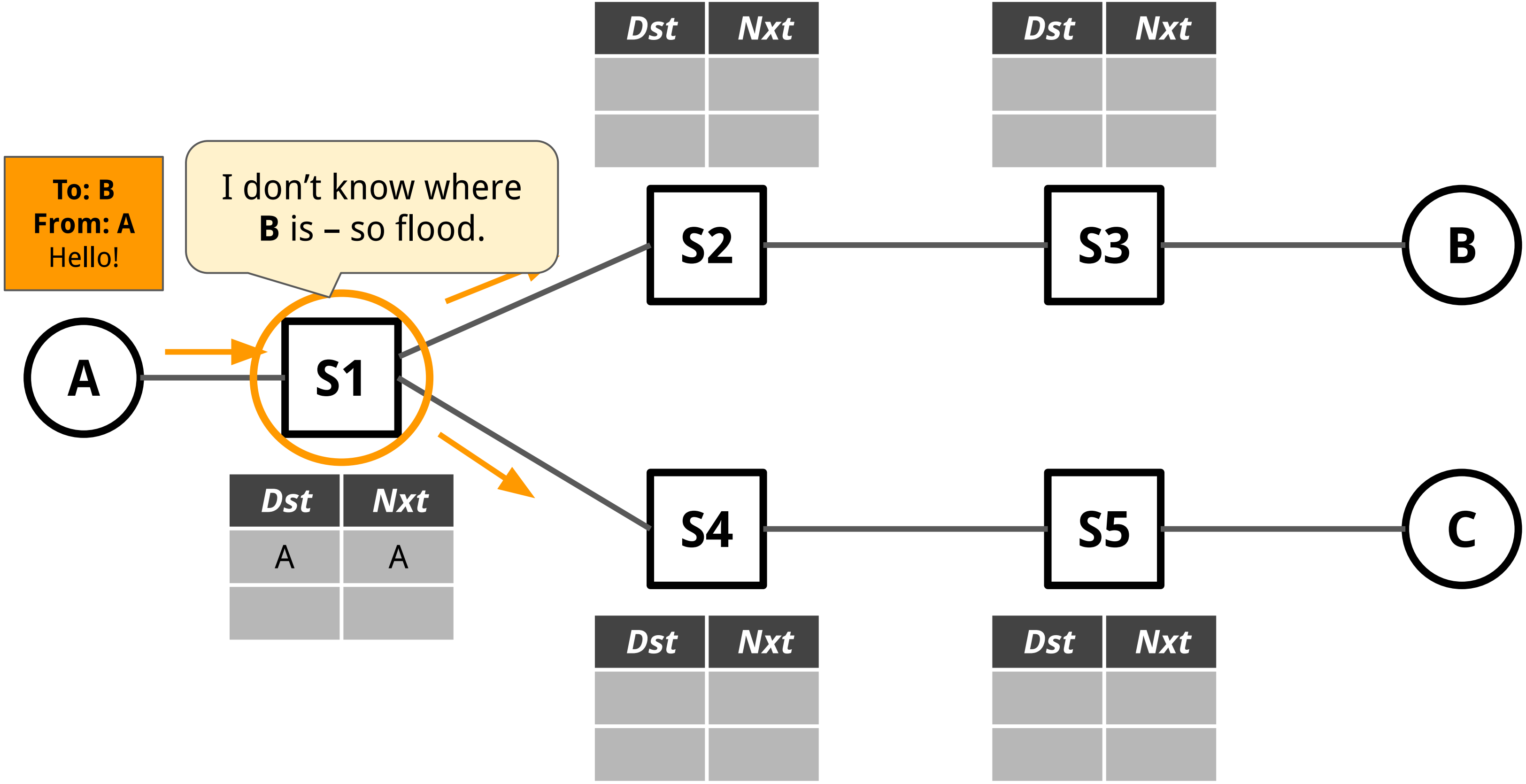
Learning Switches



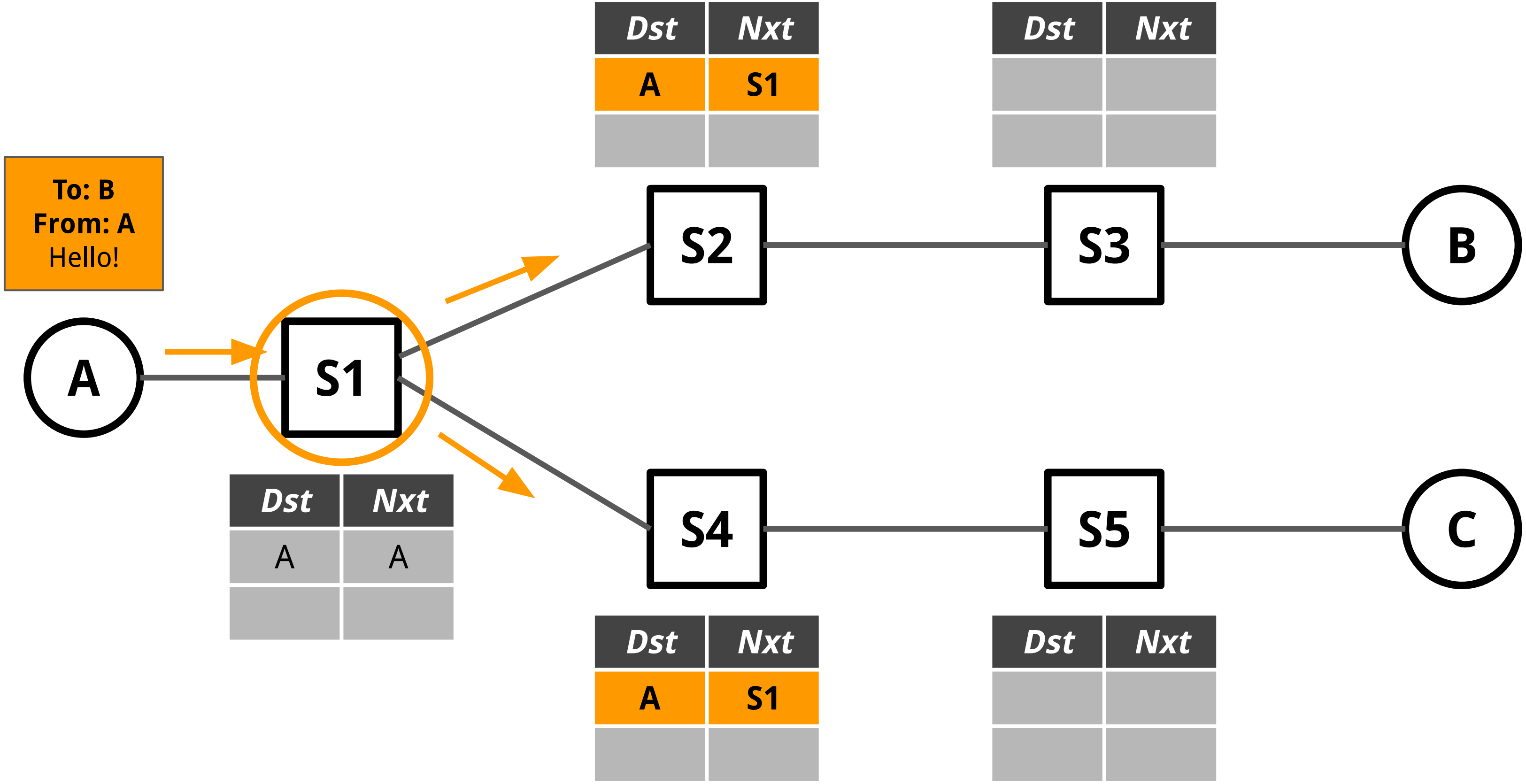
Learning Switches



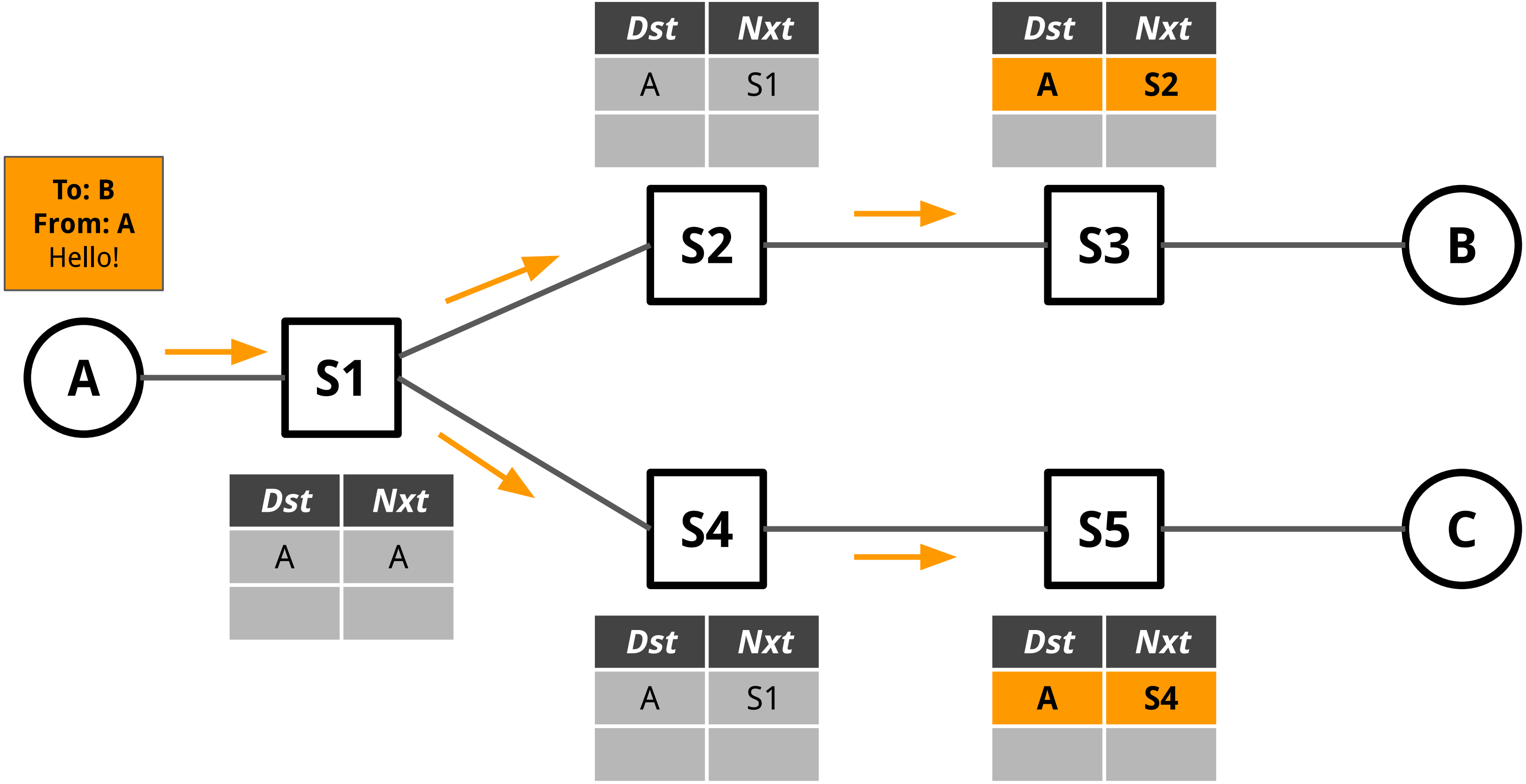
Learning Switches



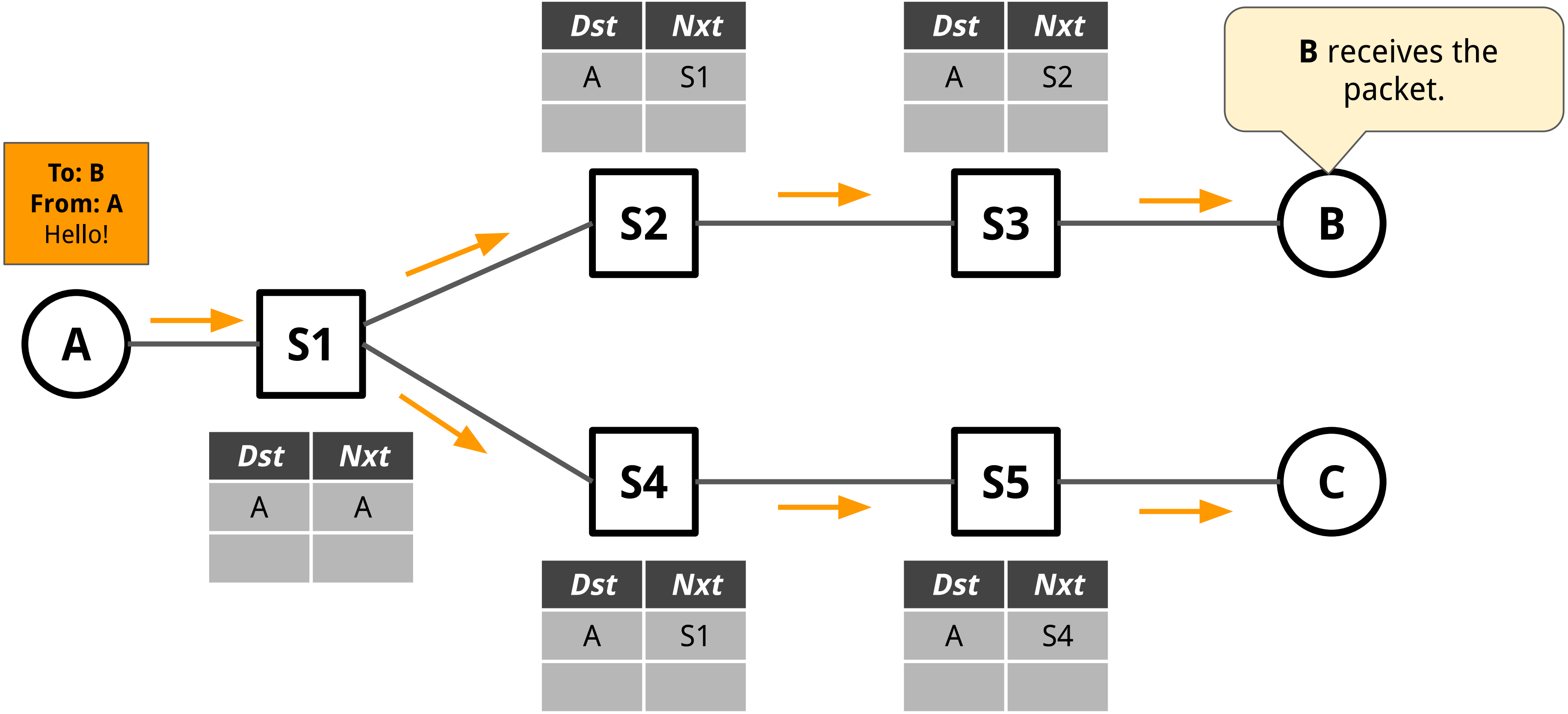
Learning Switches



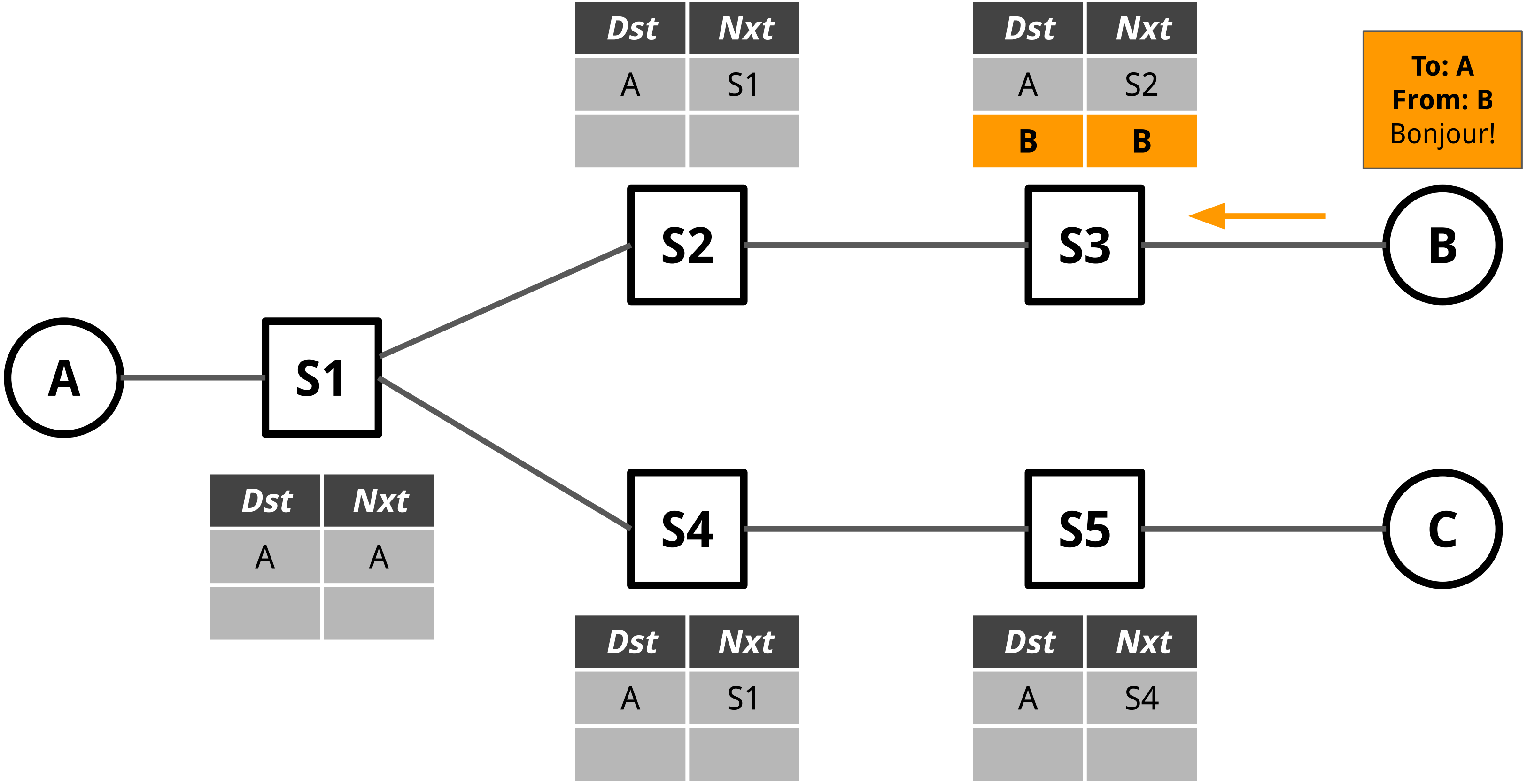
Learning Switches



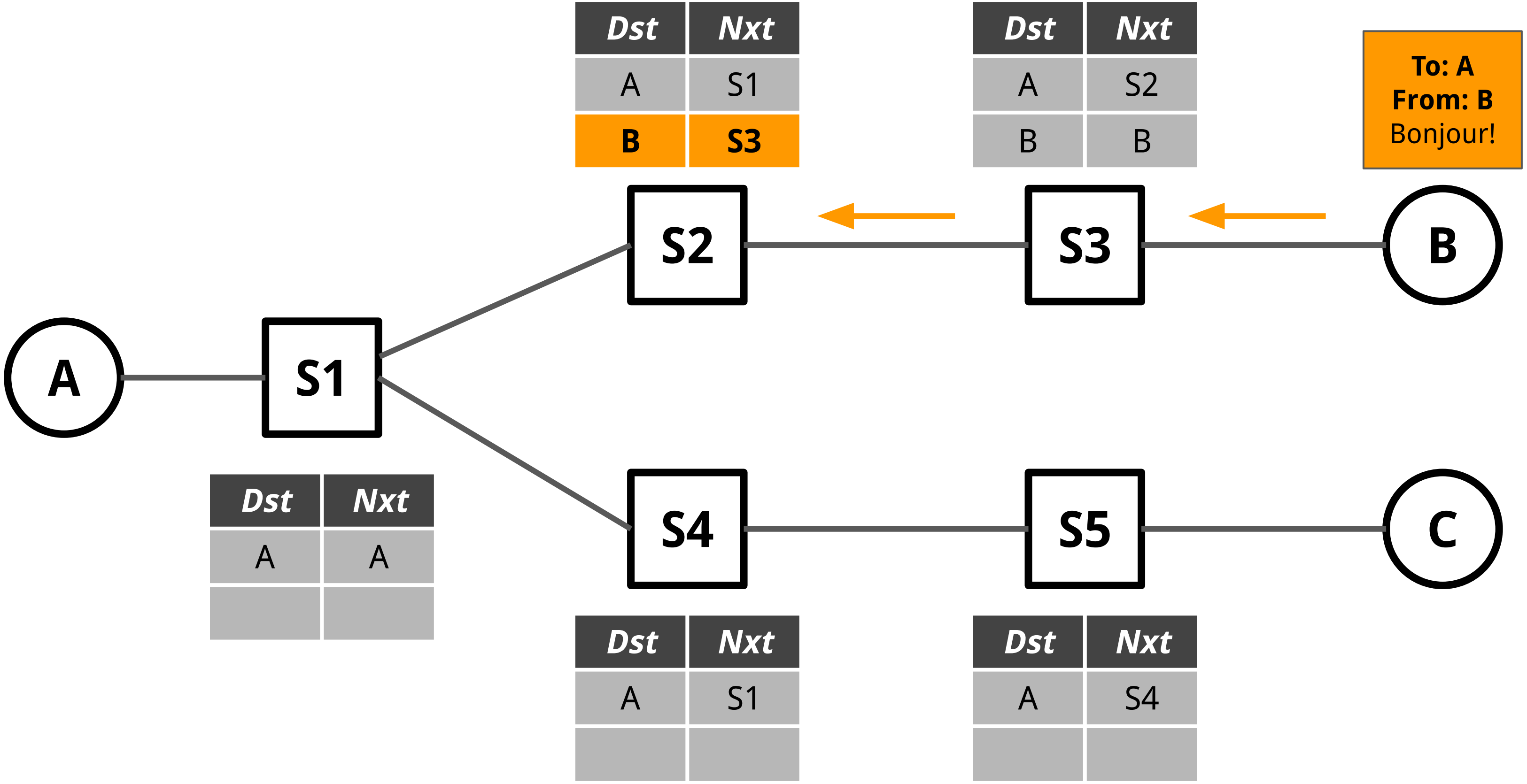
Learning Switches



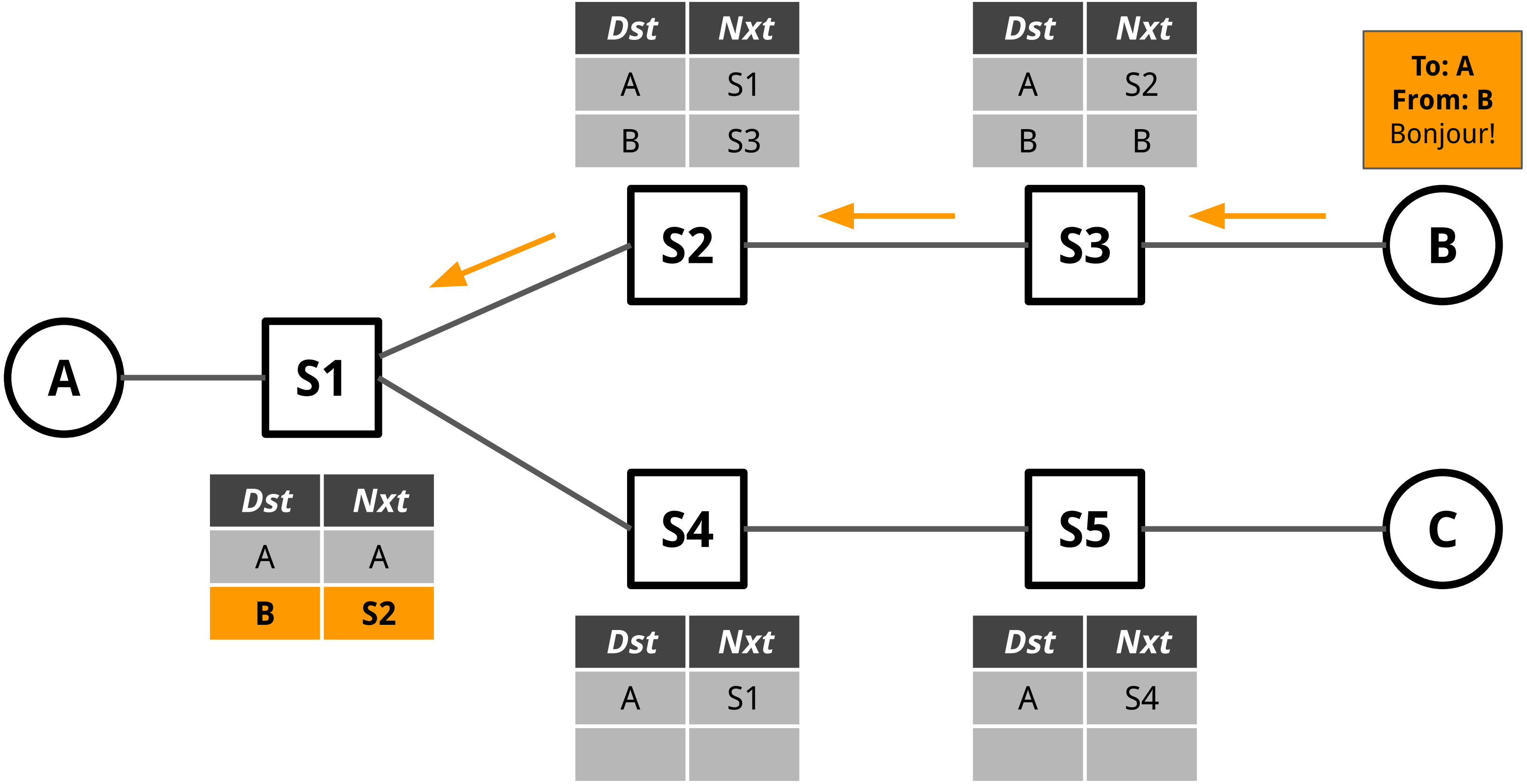
Learning Switches



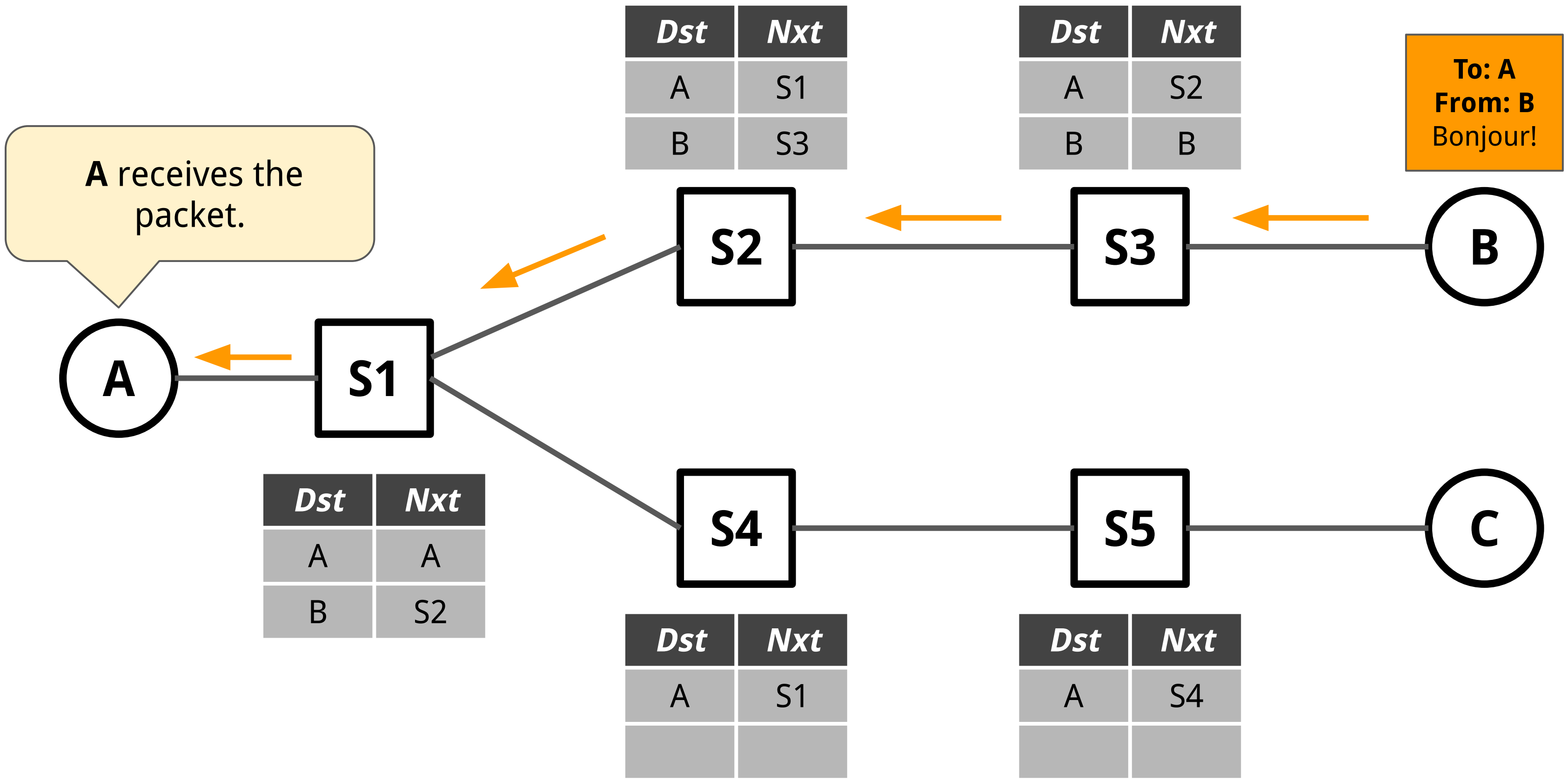
Learning Switches



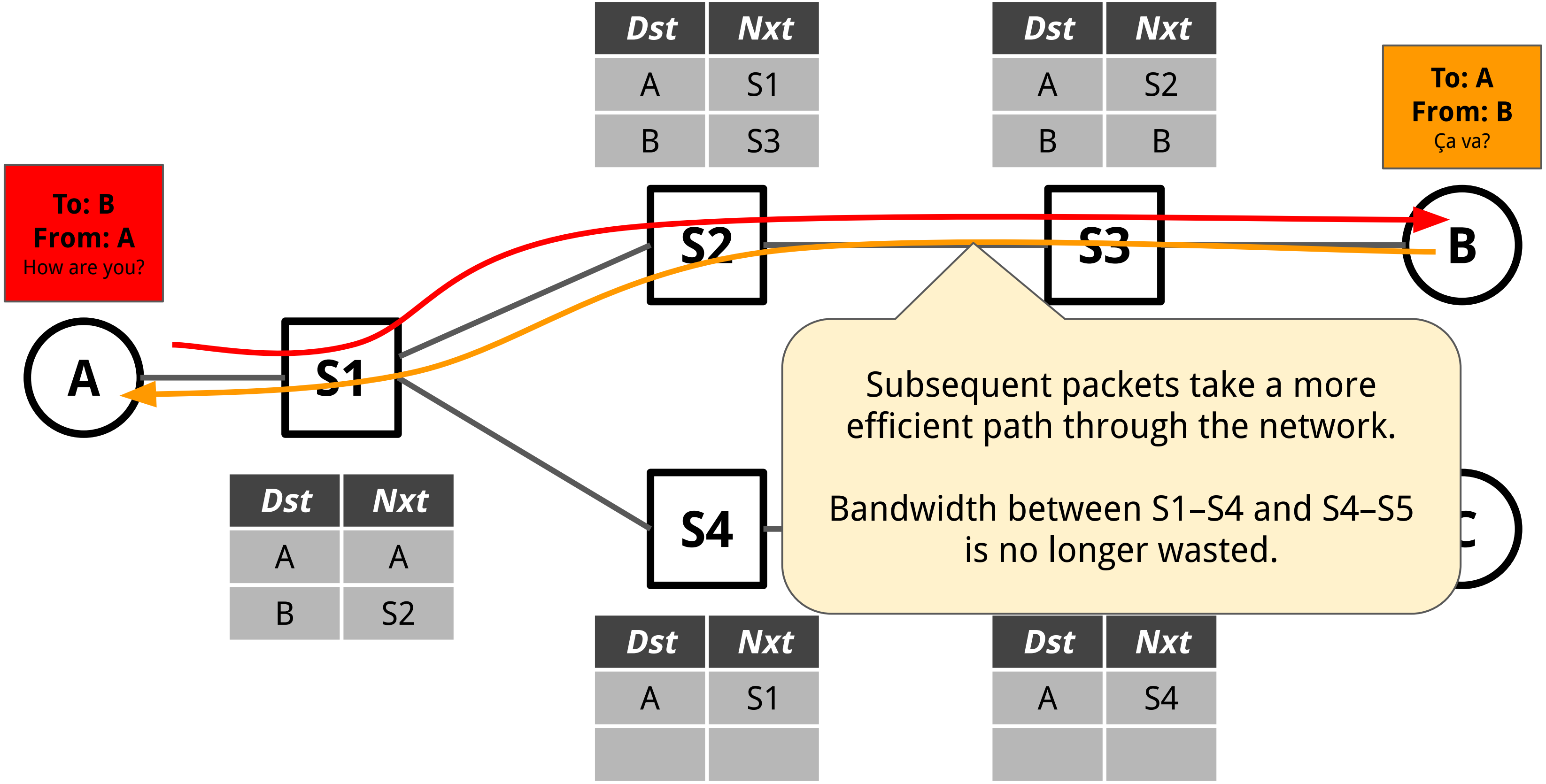
Learning Switches



Learning Switches



Learning Switches



Dst	Nxt
A	S1
B	S3

Dst	Nxt
A	S2
B	B

Dst	Nxt
A	A
B	S2

Dst	Nxt
A	S1

Dst	Nxt
A	S4

Subsequent packets take a more efficient path through the network.

Bandwidth between S1–S4 and S4–S5 is no longer wasted.

Learning Switches

- Each switch decides whether it is going to flood.
  - One switch may flood a packet when it neighbour does not.
  - Based on whether there is a **forwarding table entry**.

- At each switch:
  - If a forwarding table entry exists → send on the port specified.
  - Else → flood out of all ports (except the incoming port) if not.

Learning Switches in Pseudocode

```
on arrival of packet from neighbour previous_hop:
    # Learn
    table[packet.source].next_hop = previous_hop
    table[packet.source].ttl = 600 * time.second

    # Forward
    if packet.destination in table:
        next_hop = table[packet.destination].next_hop
        if next_hop == previous_hop:
            packet.drop()
        else:
            packet.forward_to(next_hop)
    else:
        packet.flood_to_neighbours(except=previous_hop)
```

Learning Switches in Pseudocode

```
on arrival of packet from neighbour previous_hop:
    # Learn
    table[packet.source].next_hop = previous_hop
    table[packet.source].ttl = 600 * time.second

    # Forward
    if packet.destination in table:
        next_hop = table[packet.destination].next_hop
        if next_hop == previous_hop:
            packet.drop()
        else:
            packet.forward_to(next_hop)
    else:
        packet.flood_to_neighbours(except=previous_hop)
```

Why do we need this **time to live?**

Learning Switches in Pseudocode

```
on arrival of packet from neighbour previous_hop:
    # Learn
    table[packet.source].next_hop = previous_hop
    table[packet.source].ttl = 600 * time.second

    # Forward
    if packet.destination in table:
        next_hop = table[packet.destination].next_hop
        if next_hop == previous_hop:
            packet.drop()
        else:
            packet.forward_to(next_hop)
    else:
        packet.flood_to_neighbours(except=previous_hop)
```

Why do we need this **source port check?**

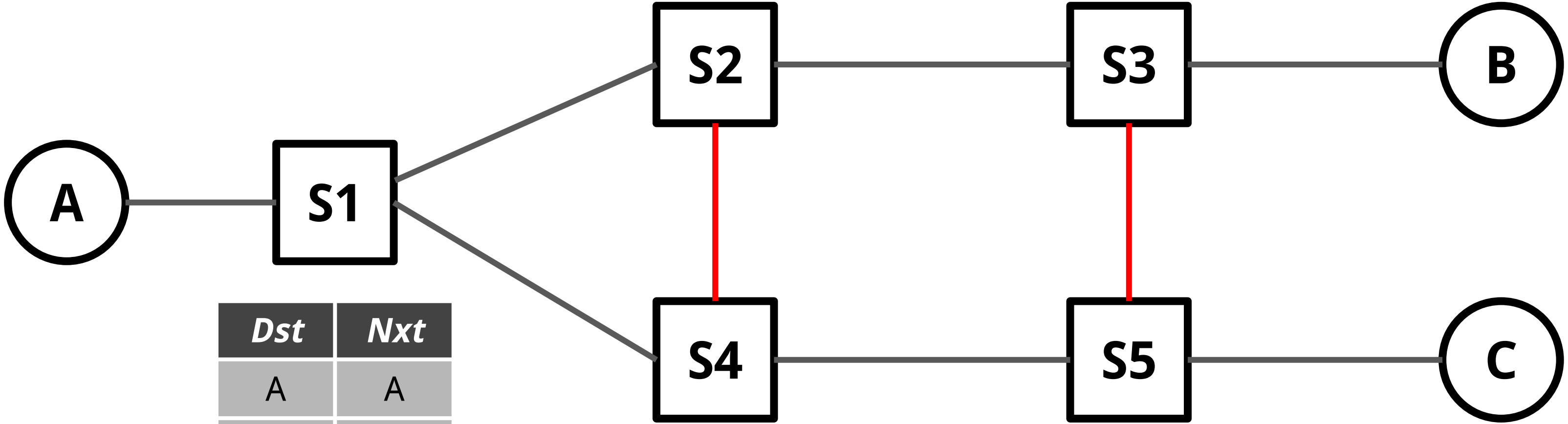
Learning Switches

S2:

Dst	Nxt
A	S1
B	S3

S3:

Dst	Nxt
A	S2
B	B



S1:

Dst	Nxt
A	A
B	S2

S4:

Dst	Nxt
A	S1

S5:

Dst	Nxt
A	S4

Learning Switches

- A major problem with learning switches:
  - Floods when the destination is unknown.
  - … floods have problems when the topology has loops.
- How do we prevent there being looped packets?
- Brute force solution…
  - Remove the loops.
  - Disable links until we have a topology that connects to all hosts but does not have any loops.
  - This is a spanning tree (we'll come back and discuss these more later).

Spanning Tree Protocol

- How do we make a spanning tree from an arbitrary network?
- Step 1: Find a path from every switch to the root.
- Step 2: Disable data delivery on every link not on a path to the root.
- Step 3: When the tree breaks (a link on it fails) start over.

Spanning Tree Protocol

- How do we make a spanning tree from an arbitrary network?
- Step 1: Find a path from every switch to the root.
    - If there are multiple links – how do we choose? We need an idea of a link or a node **preference** or **cost**.
- Step 2: Disable data delivery on every link not on a path to the root.
- Step 3: When the tree breaks (a link on it fails) start over.

Spanning Tree Protocol: Step 1 (Paths to root)

- Step 1: Find the least cost path from every switch to the root.

- Give every switch a unique, orderable ID (based on the Ethernet address)

- Work to find:
  - The root (the switch with the lowest ID)
  - The best path to the root (lowest cost).

Spanning Tree Protocol: Step 1 (Paths to root)

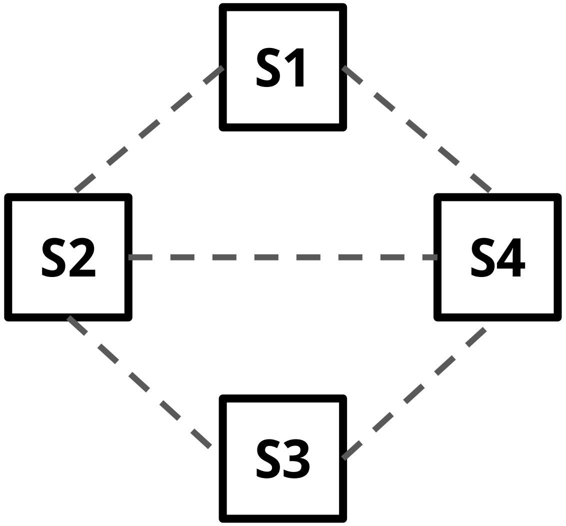
- Start out: all switches think that they are the root.
- Sends a message to its neighbour to say (“`The root is` **`<me>`** `and I can reach it in` **`<zero>`** `hops!`”).

- On receiving a message from a neighbour:
    - First, compare the root ID to what we think the root ID is…
    - If it’s smaller than the current ID – it is a better root, use it as a root.
    - If its larger than the current ID – it is a worse root, ignore it.
        - (We’ll come back to what happens if it’s the same!)
    - … and send a triggered update to your neighbours telling you about your new state.

Spanning Tree Protocol: Step 2 (Disable links)

- Step 2: Disable data delivery on every link not on a shortest path to root.

- Wait, why is this so complicated?
  - It's not as easy as you might think…

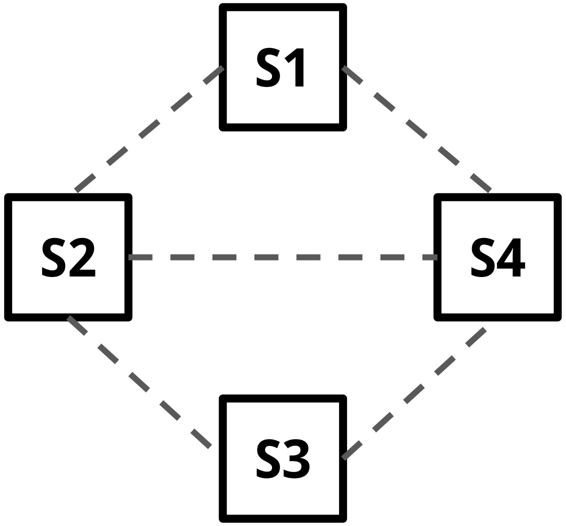


Spanning Tree Protocol: Step 2 (Disable links)

- Step 2: Disable data delivery on every link not on a shortest path to root.

- Wait, why is this so complicated?
    - It's not as easy as you might think…

- Strategy:
    - **Enable** the link along your best path to root
    - **Disable** the other links to switches closer than the root to you.
        - … they are not on your best path
        - … and you can't possibly be on theirs (you are farther than the root than them!)
    - Leave the other links for the other switches to decide
        - … they are all farther from the root than you
        - … so you're closer
        - … so the above enable/disable rules work for them.

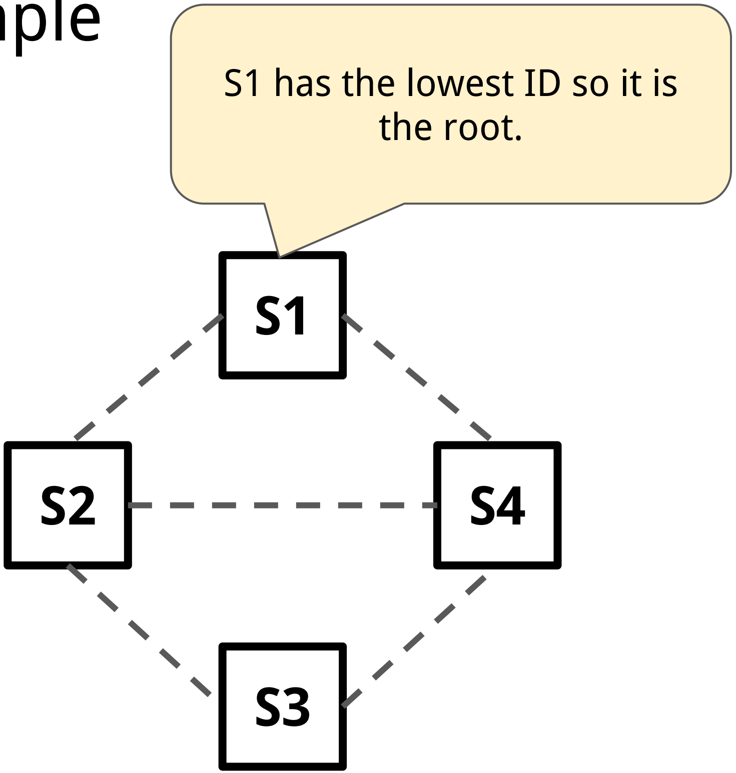


Spanning Tree Protocol: Step 2 Example

- Dashed links are unknown.
- Green links are enabled.
- Red links are disabled.

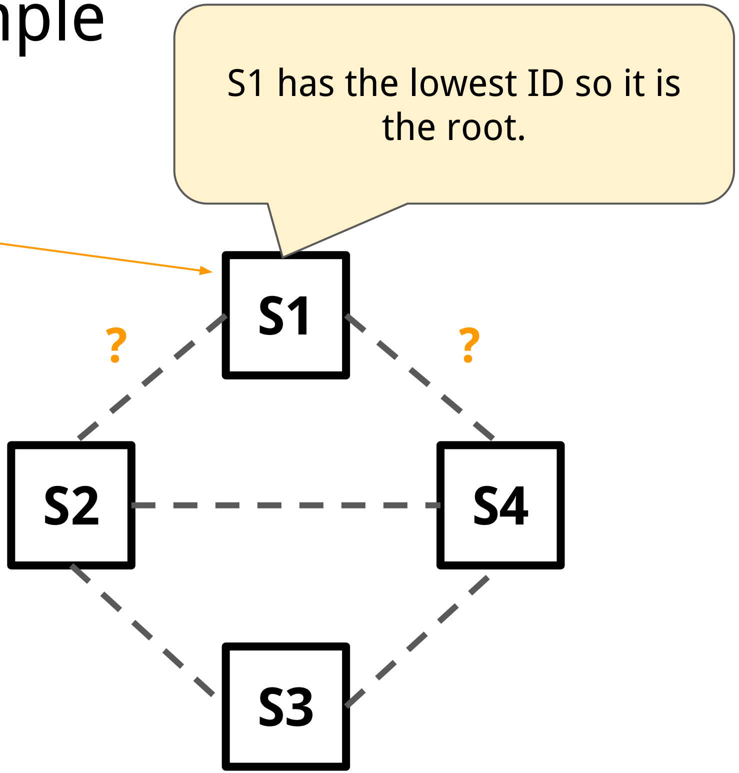
- S1 is the root (lowest ID)

- Step 1 is complete – we know about our neighbours.



Spanning Tree Protocol: Step 2 Example

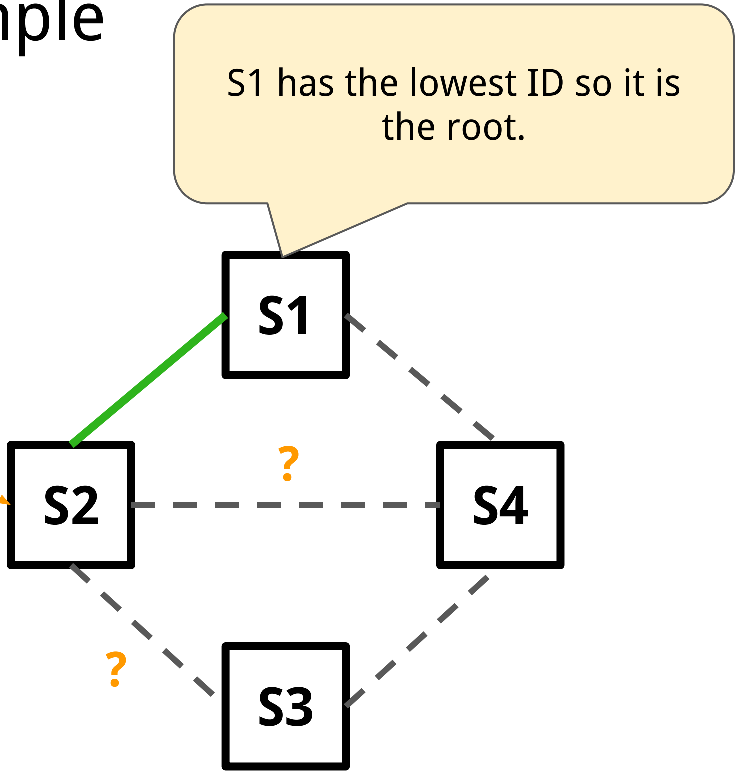
- S1's perspective
  - S1–S2: Unknown
  - S1–S4: Unknown



Spanning Tree Protocol: Step 2 Example

- S2’s perspective
  - S2–S1: Enabled
  - S2–S3: Unknown
  - S2–S4: Unknown

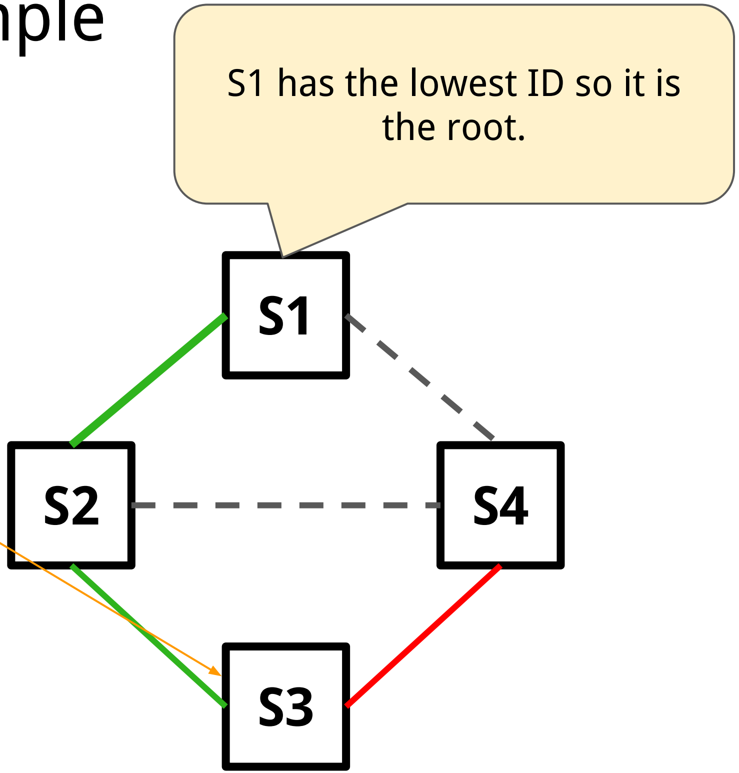
- S4 has the same distance to the root – but has a higher ID – so it’s farther from the root.



Spanning Tree Protocol: Step 2 Example

- S3's perspective
  - S3–S2: Enabled
  - S2–S4: Disabled

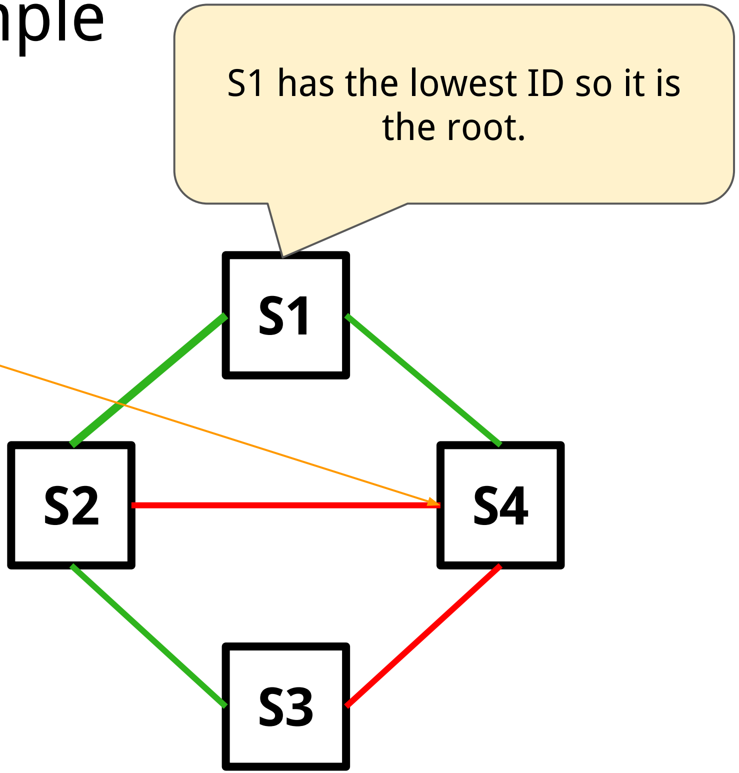
- S2:
  - Closer to the root than S4 (same distance, but lower ID)
- S4:
  - Closer than we are, but not on the shortest path – disable.



Spanning Tree Protocol: Step 2 Example

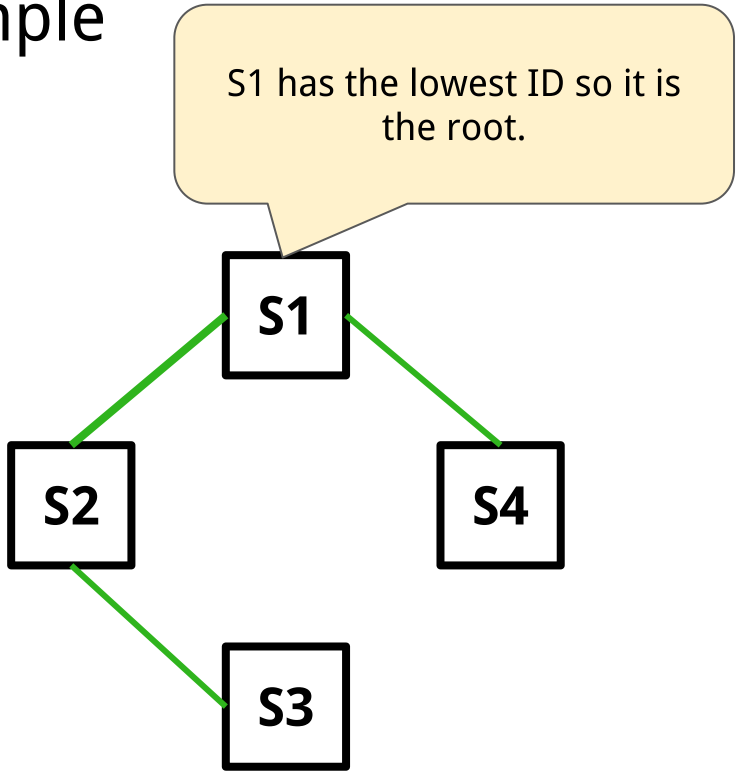
- S4's perspective
  - S4–S1: Enabled
  - S4–S3: Unknown
  - S4–S2: Disabled

- S2:
  - Closer to the root (same distance, but smaller ID).



Spanning Tree Protocol: Step 2 Example

- We've got a spanning tree!
- And it matches the next-hops each switch came up with from Step 1!



Spanning Tree Protocol: Step 2 (Disable links)

- Step 2 Recap…
- No ties when comparing distance – break ties using a switch's ID.
- Each switch:
  - **Enables** the link along the best path to the root (and all links to hosts!)
  - **Disables** every other link to a neighbour closer to the root.
  - Lets the furthest-away neighbour decide the rest!
- In this way - a switch closer doesn't disable a link needed by a switch that's farther.
  - Doesn't require explicit co-ordination (no need to ask "do you need this link?")
  - Exactly one switch is responsible for enabling or disabling each link.

Spanning Tree Protocol: Step 3

- Step 3: When the tree breaks (a link on it fails), start over.

- Starts the process at step 1 to re-discover a spanning tree.

Spanning Tree Protocol

- Notice, we had to exchange messages between switches to have some knowledge of the topology.
    - Done by the switch's **control plane**.

- We had to have some idea of **preference** or **cost** within the topology to decide what to enable/disable.

- **Spanning tree** is a (simple) "routing" protocol.
    - We'll talk more about different protocols and their approaches going forward.
    - Limited in functionality – simply to avoid us having a "loopy" topology that causes problems with learning switches.

Next Time

- Consider some of the problems of our Layer 2 network.

- Introduce more *routing protocols* and discuss different optimisation criteria that they might enable.