UNIVERSITY OF WATERLOO

CS 856: Programmable Networks

Lecture 1: SDN and OpenFlow

Mina Tahmasbi Arashloo

Winter 2023

Logistics

- Join Slack and HotCRP
  - Invitations were sent on Tuesday
  - Make sure to check your spam folder as well
- Sign up for 10-minute paper presentations
  - The link to the spreadsheet will be sent on Slack.
  - Sign up for **4 papers** you are interested to present.
  - If there are already 3 people signed up, try to sign up for other papers
  - 2 papers will be assigned to each person.

Logistics

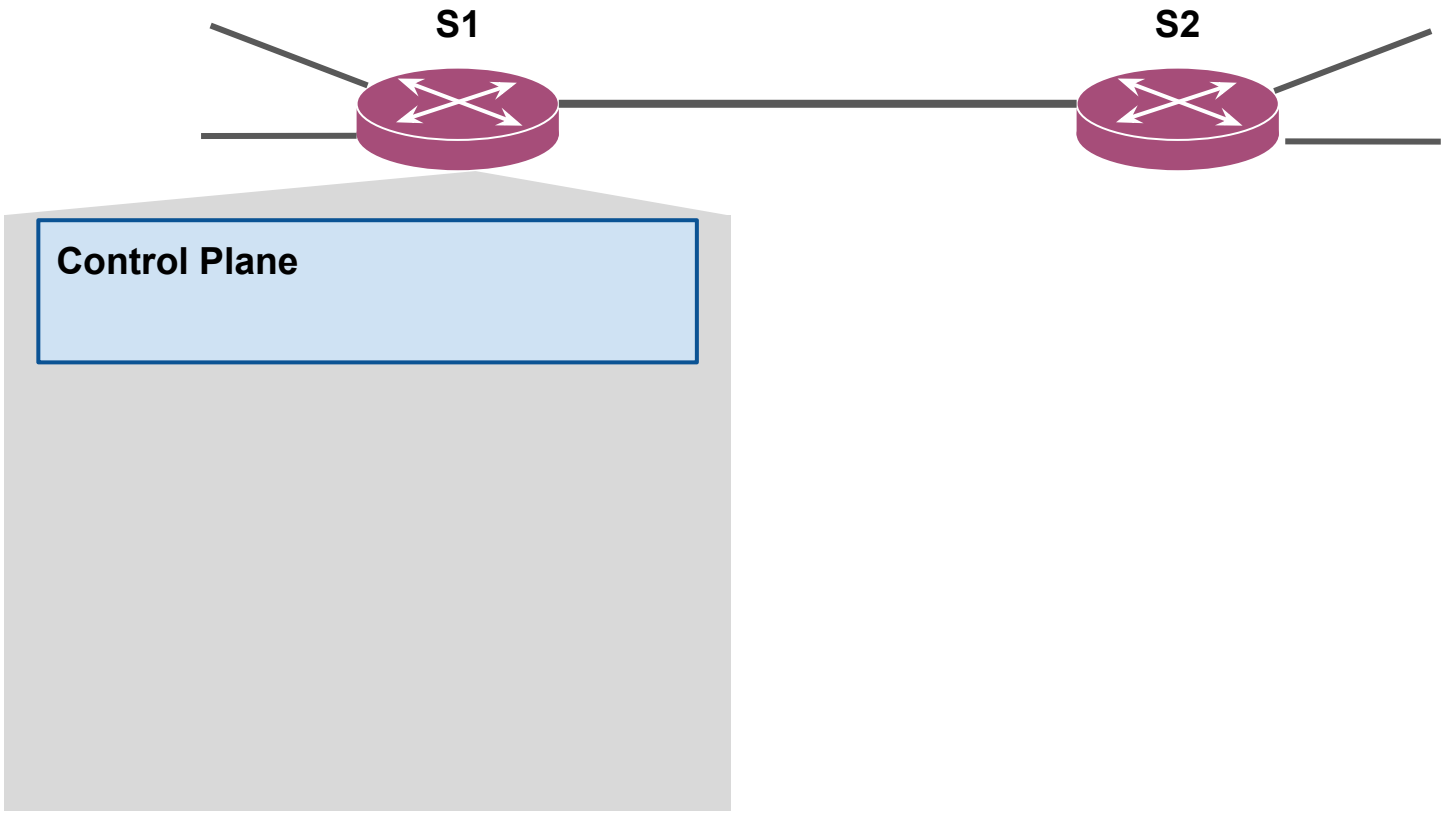
- If you need help with project ideas, come find me after class
- First round of reviews are due **Monday at 5pm**.

Traditional networks - distributed control

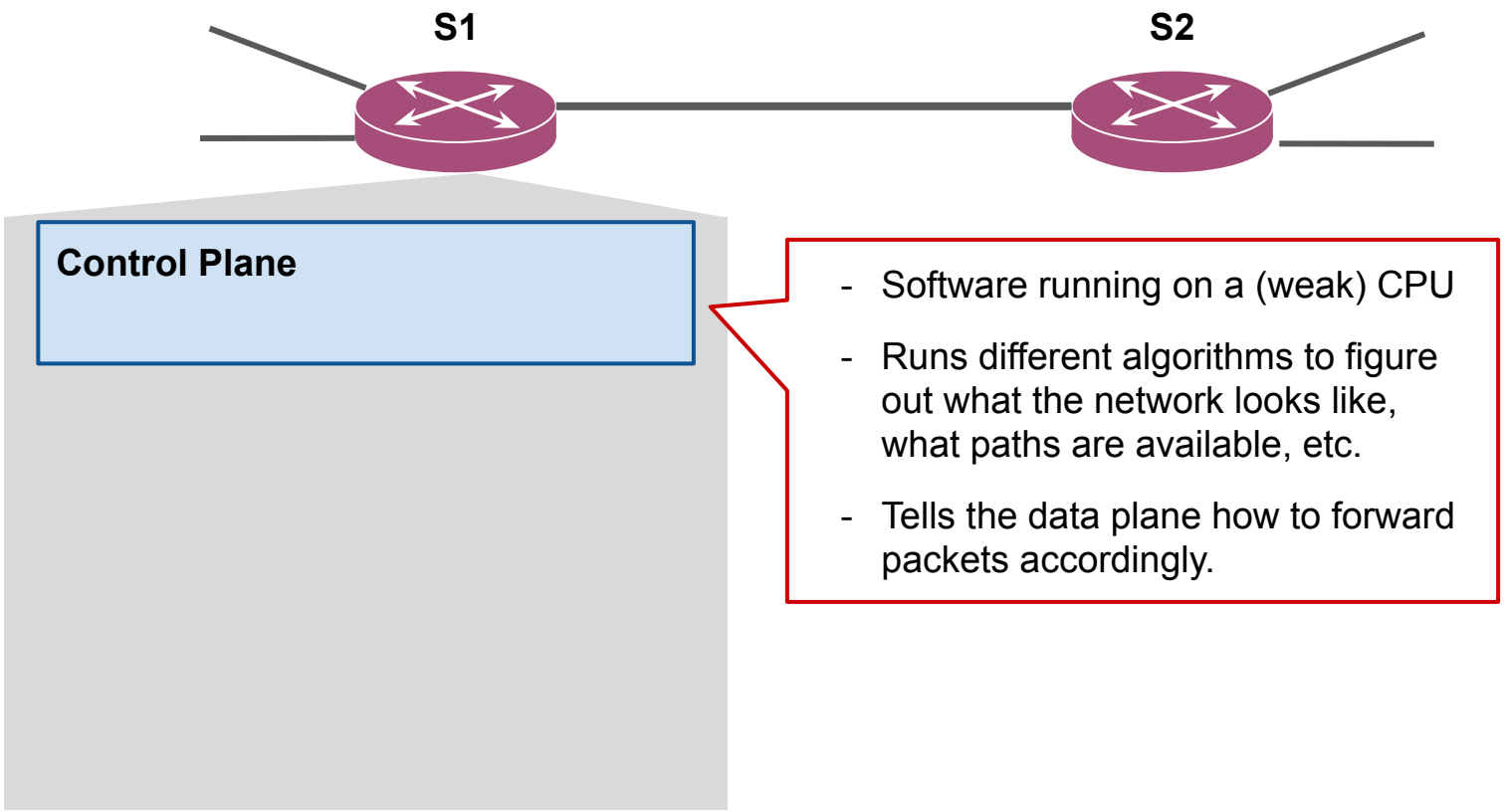
- Each device (switch, router, etc.) runs its own instance of a network algorithm/protocol.
- The devices communicate with each other to figure out how to forward packets.
- Going forward, we'll use the word *switch* as a generic term to refer to network devices that forward traffic

Traditional networks - distributed control

Traditional networks - distributed control



Traditional networks - distributed control

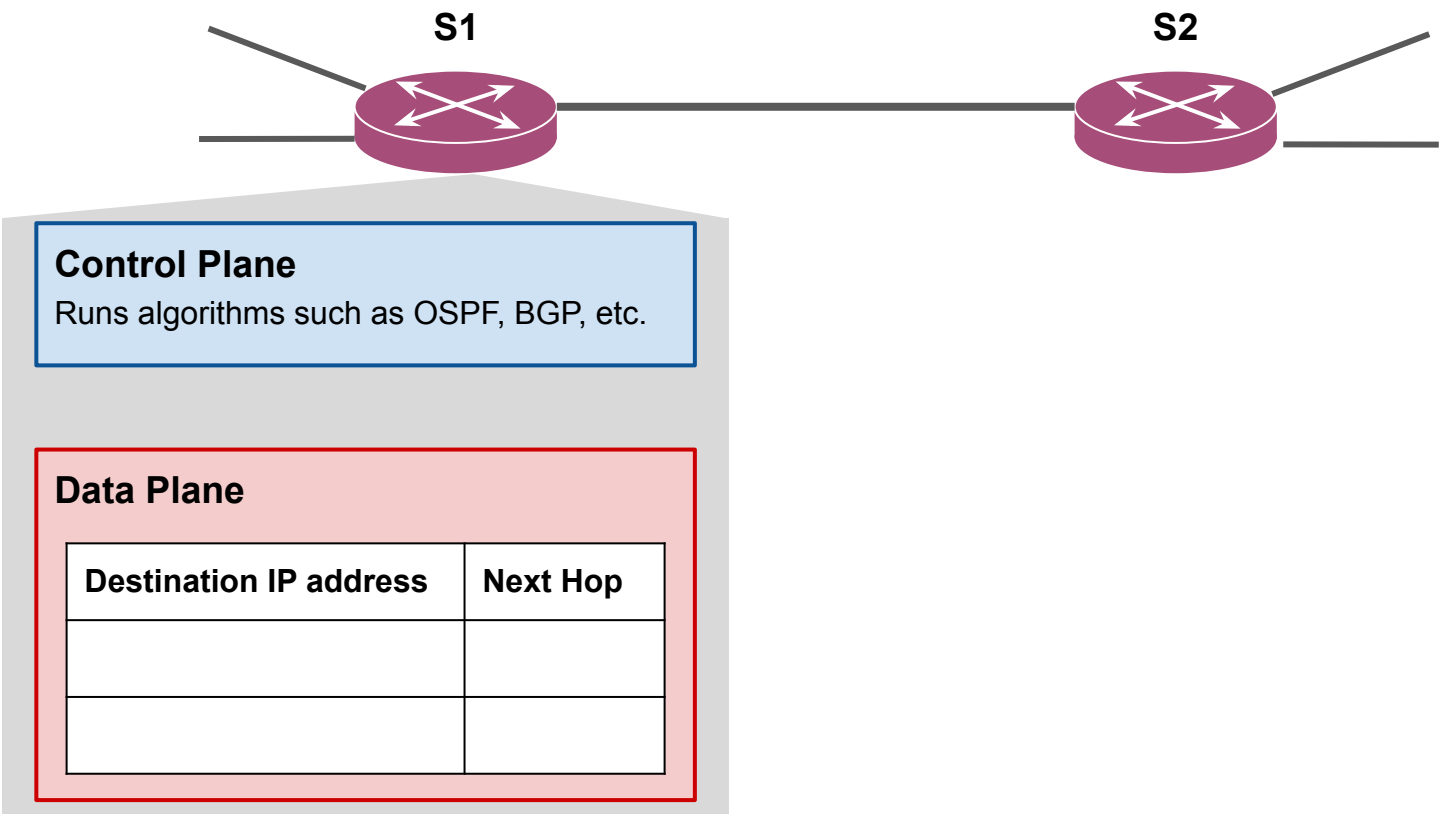


Traditional networks - distributed control

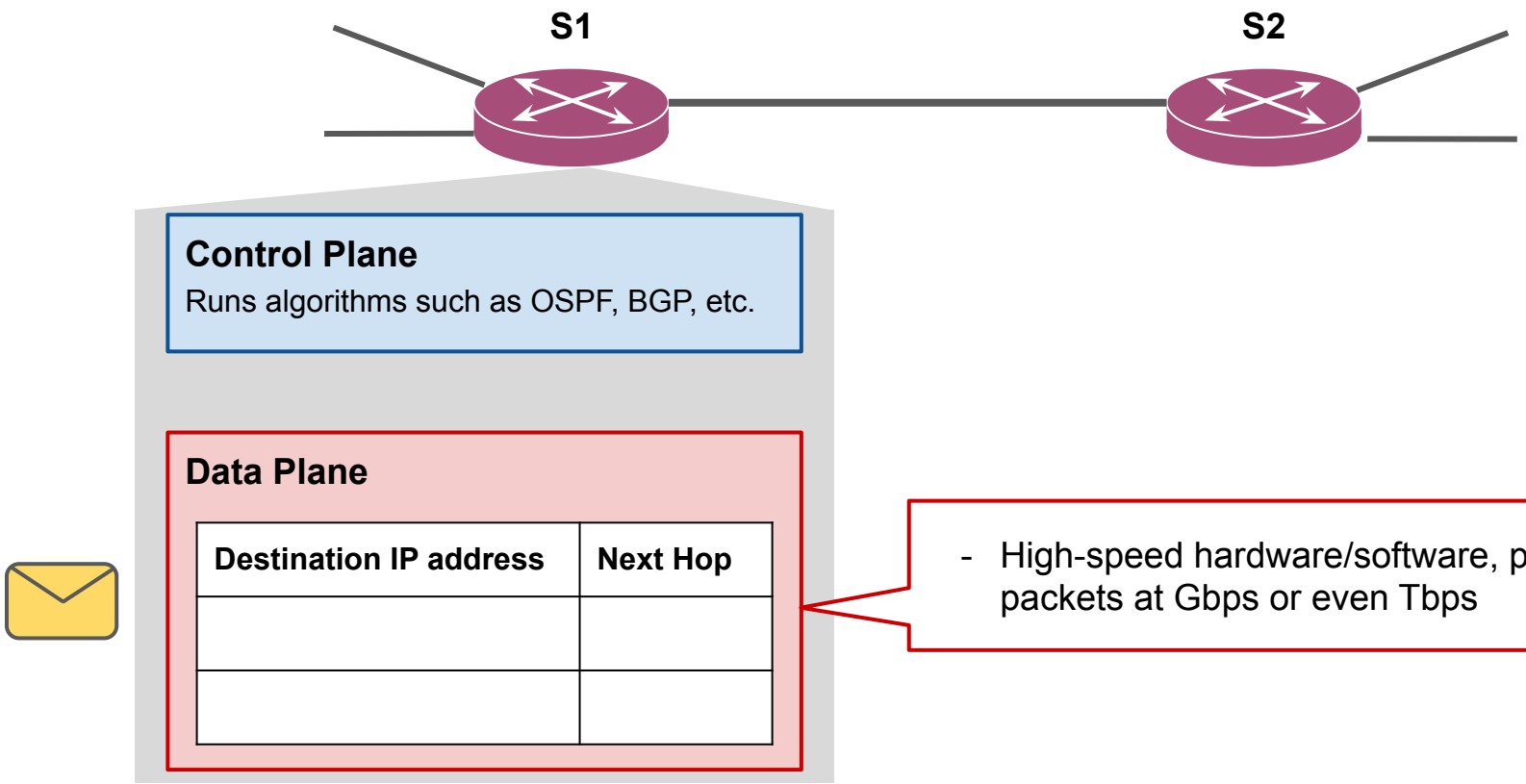
Control Plane
Runs algorithms such as OSPF, BGP, etc.

- Software running on a (weak) CPU
- Runs different algorithms to figure out what the network looks like, what paths are available, etc.
- Tells the data plane how to forward packets accordingly.

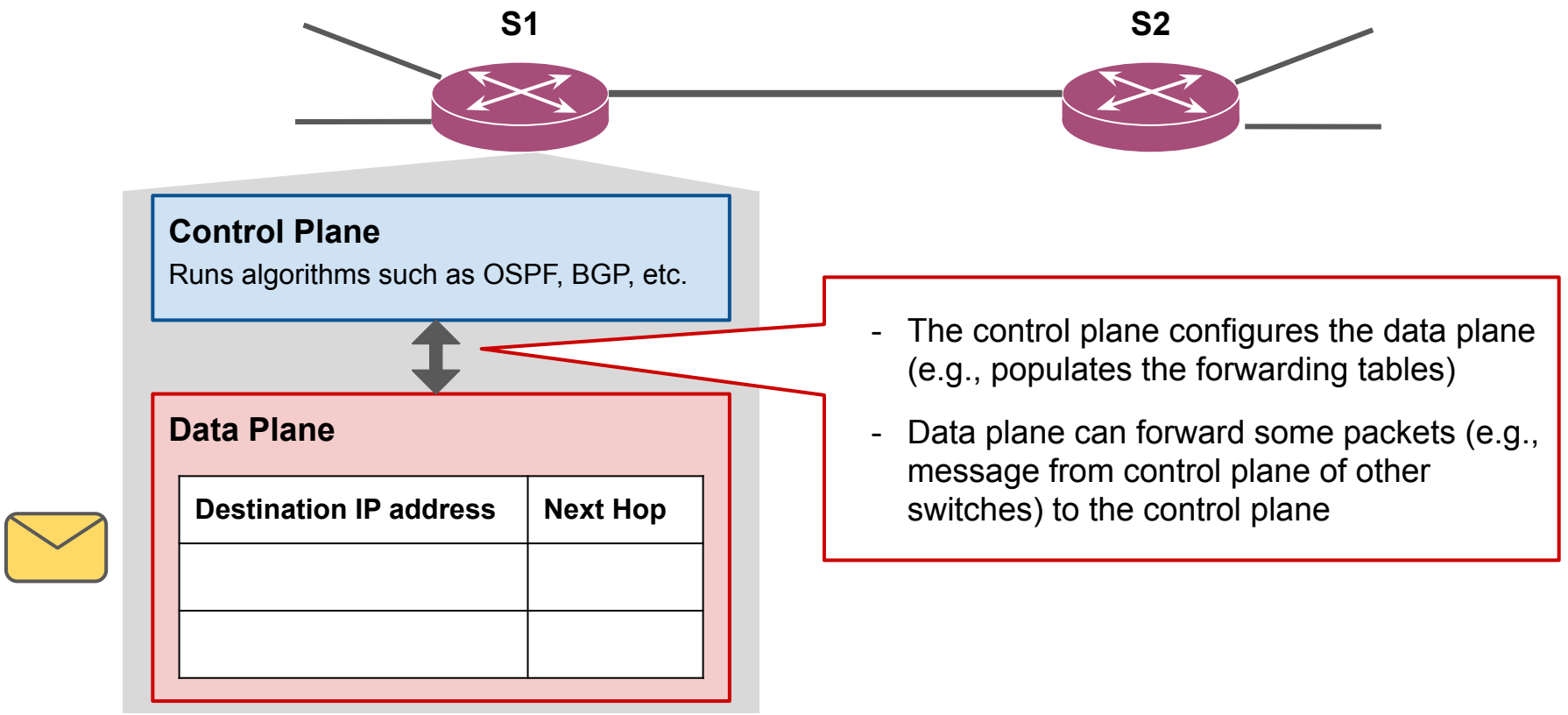
Traditional networks - distributed control



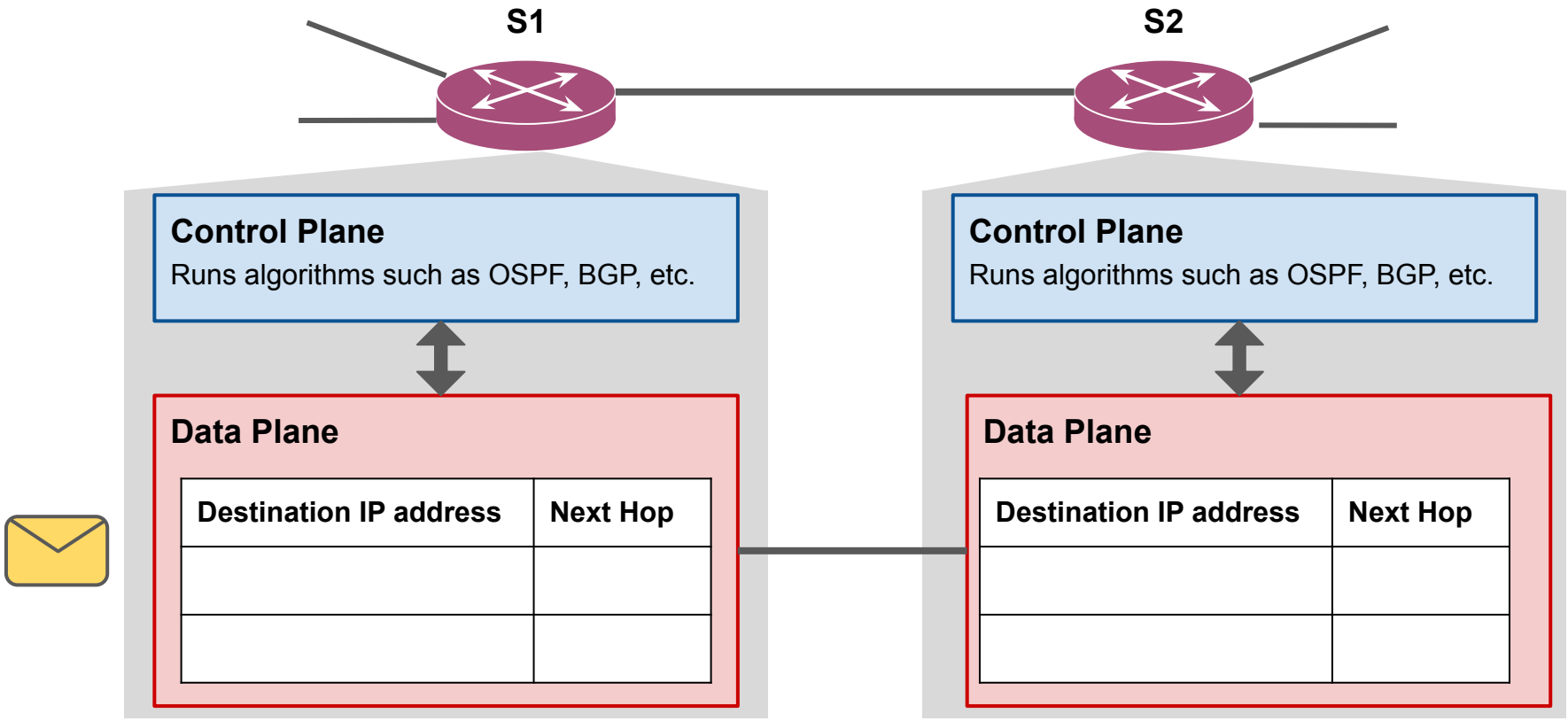
Traditional networks - distributed control



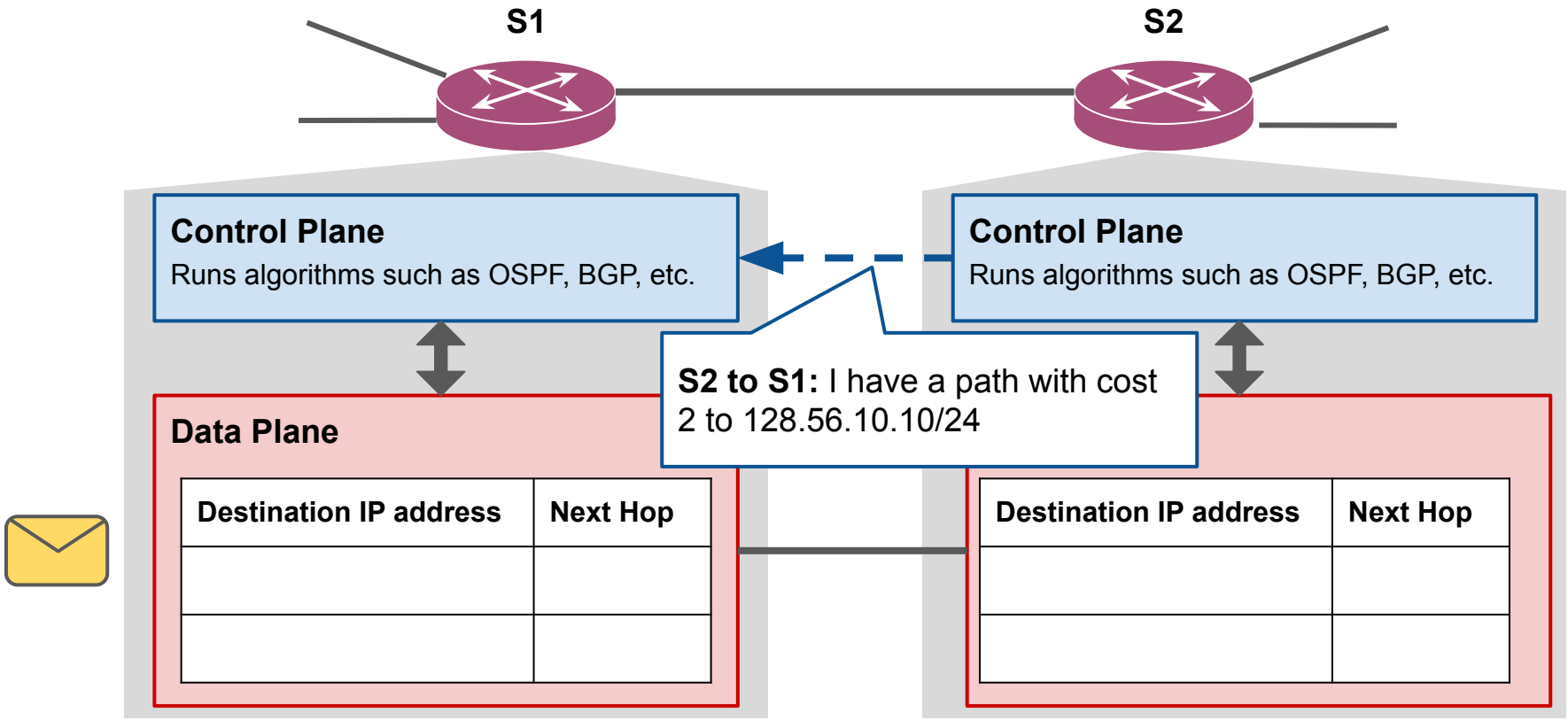
Traditional networks - distributed control



Traditional networks - distributed control

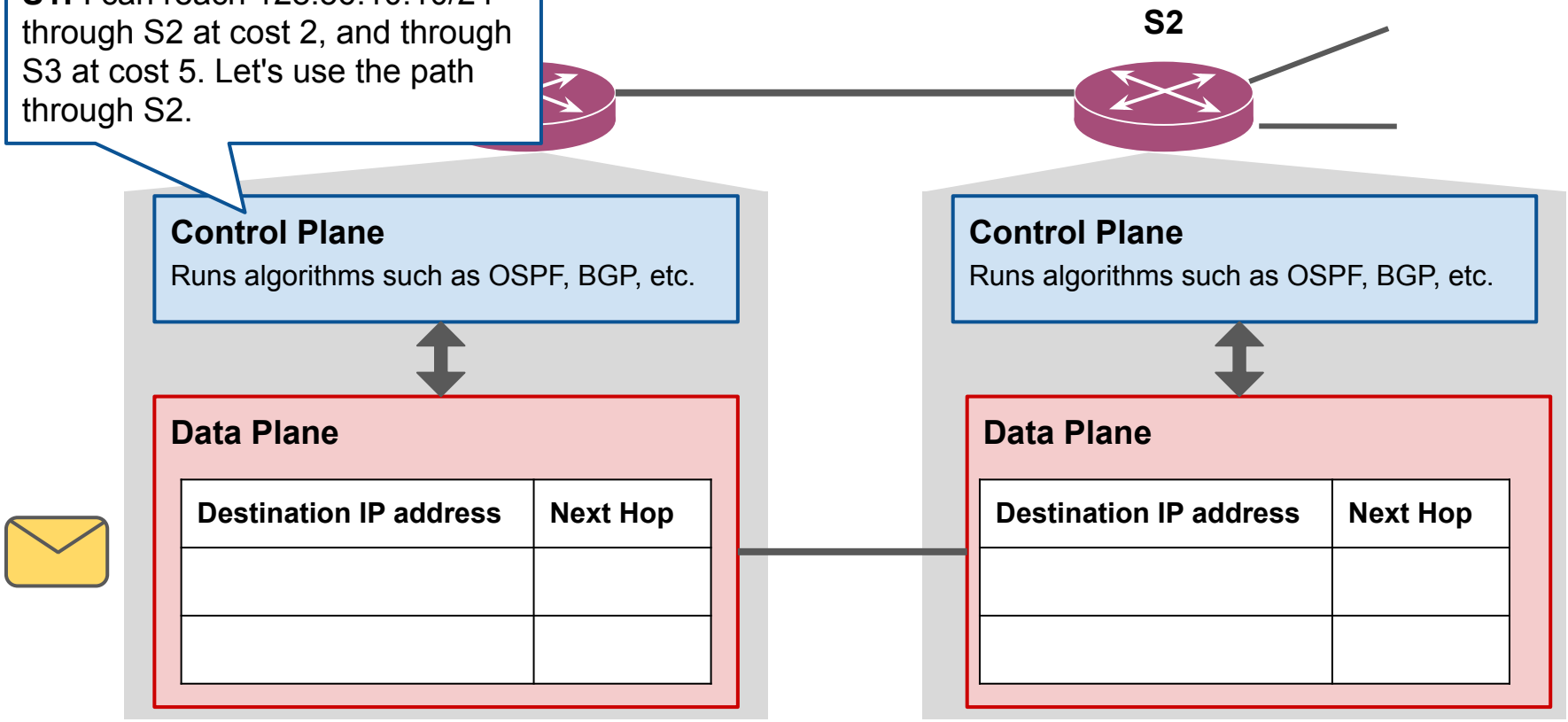


Traditional networks - distributed control

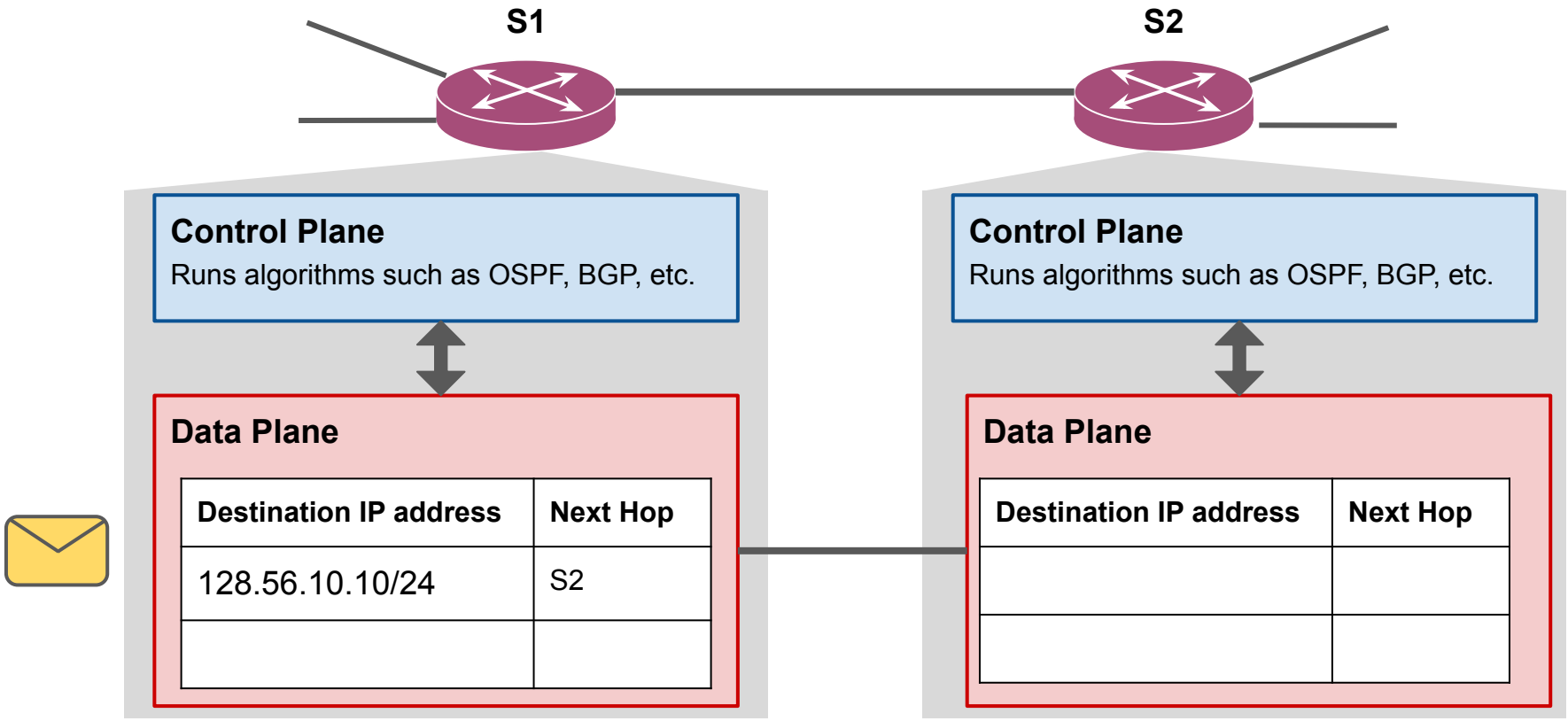


Traditional networks - distributed control

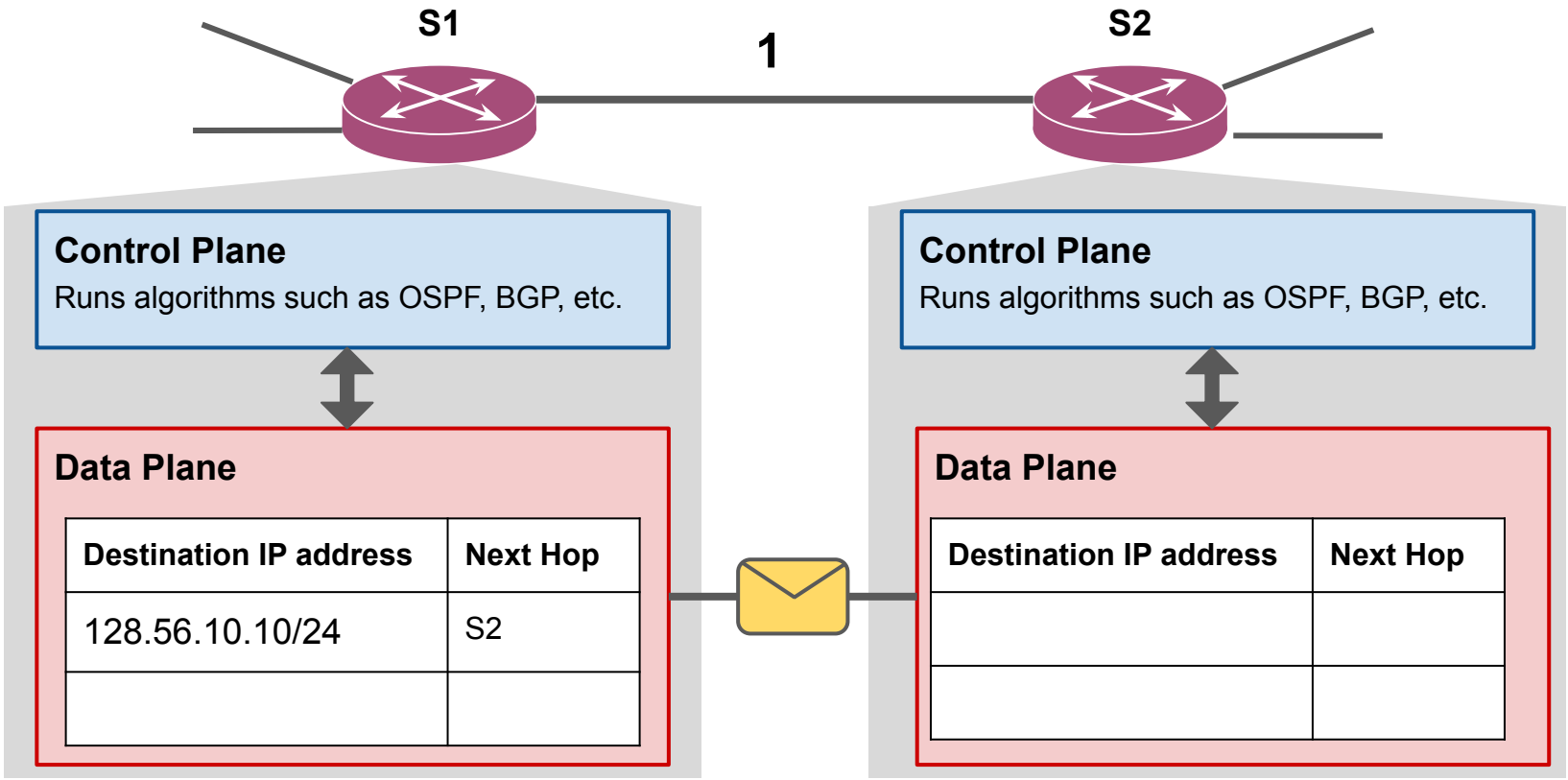
S1: I can reach 128.56.10.10/24 through S2 at cost 2, and through S3 at cost 5. Let's use the path through S2.



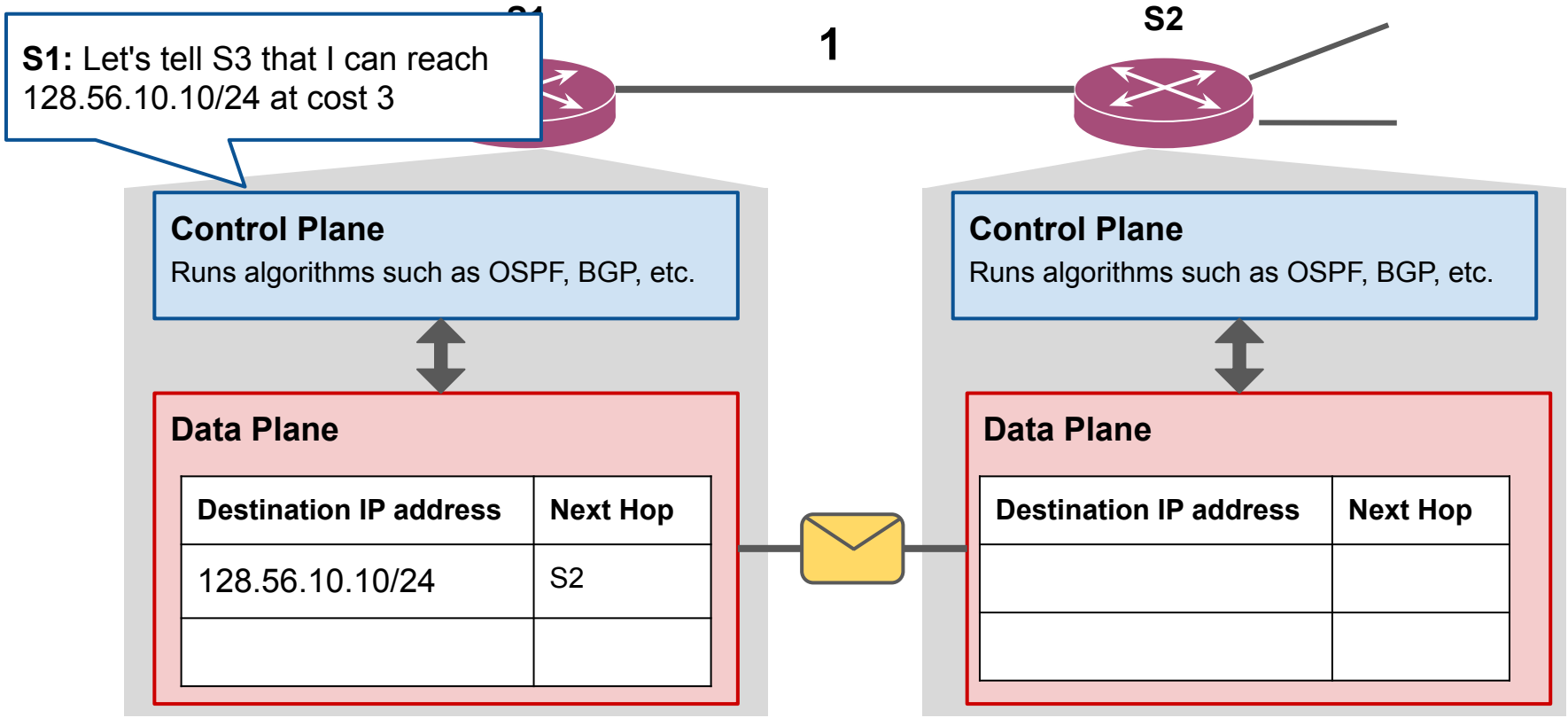
Traditional networks - distributed control



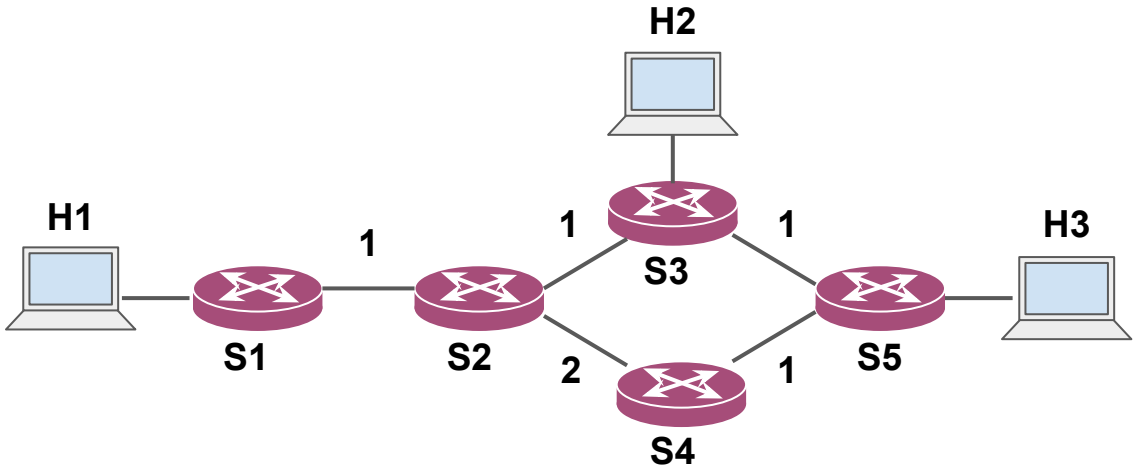
Traditional networks - distributed control



Traditional networks - distributed control

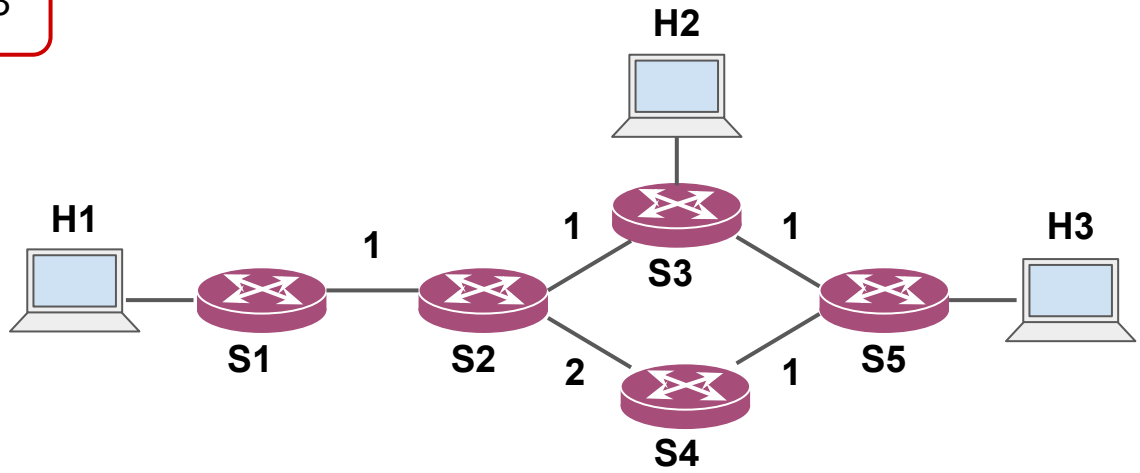


Traditional networks - "Indirect" path selection

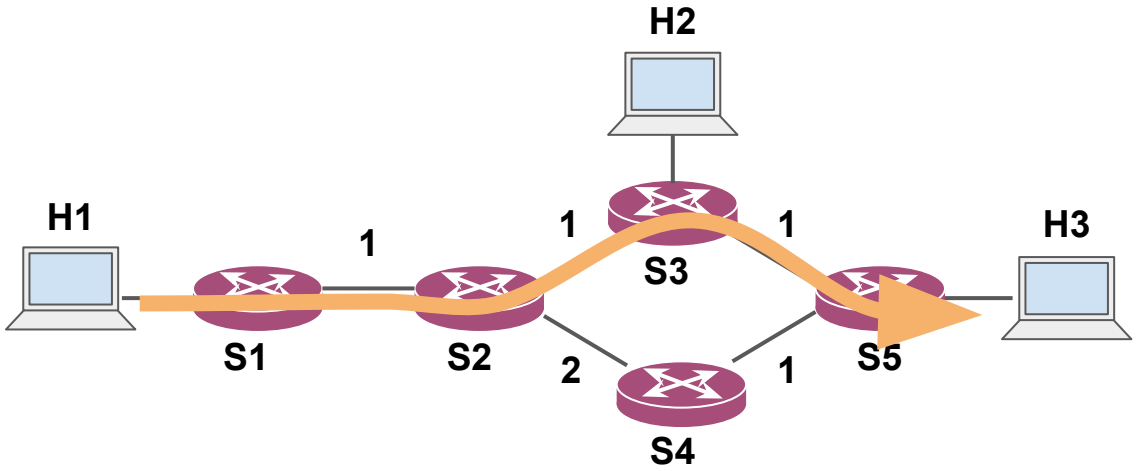


Traditional networks - "Indirect" path selection

H1 starts sending traffic to H3

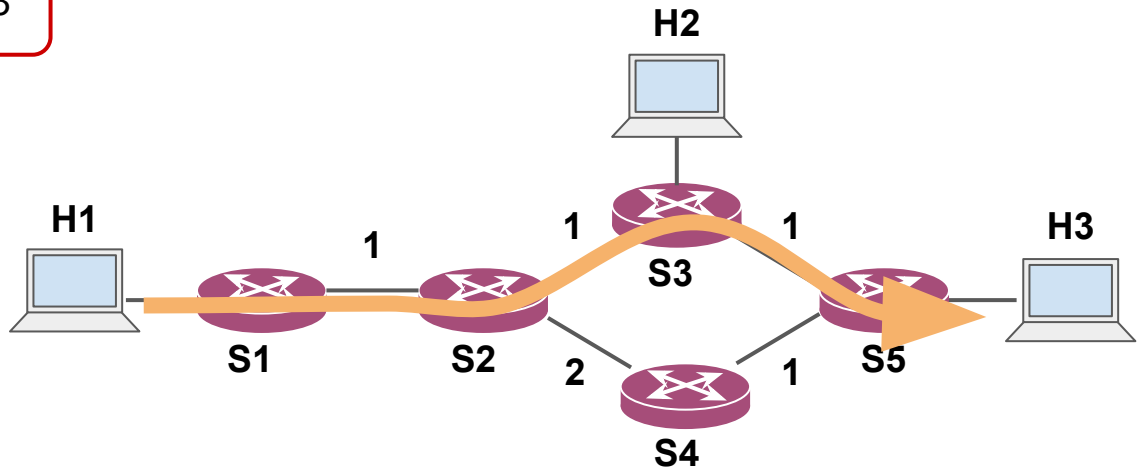


Traditional networks - "Indirect" path selection

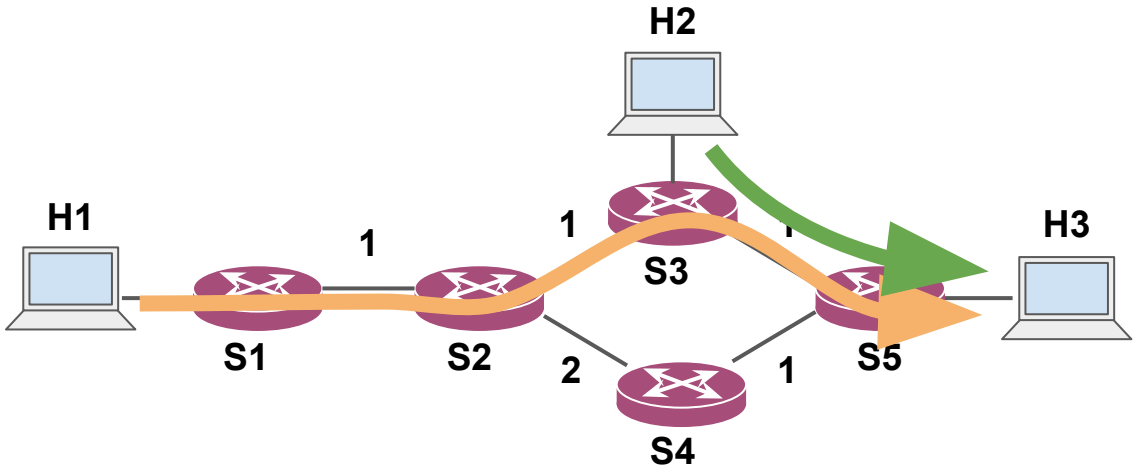


Traditional networks - "Indirect" path selection

H2 starts sending traffic to H3

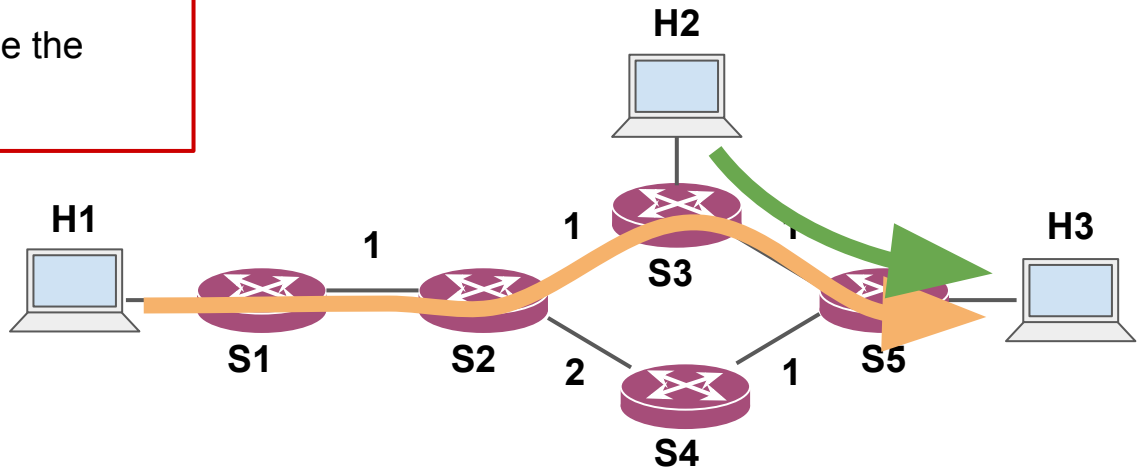


Traditional networks - "Indirect" path selection

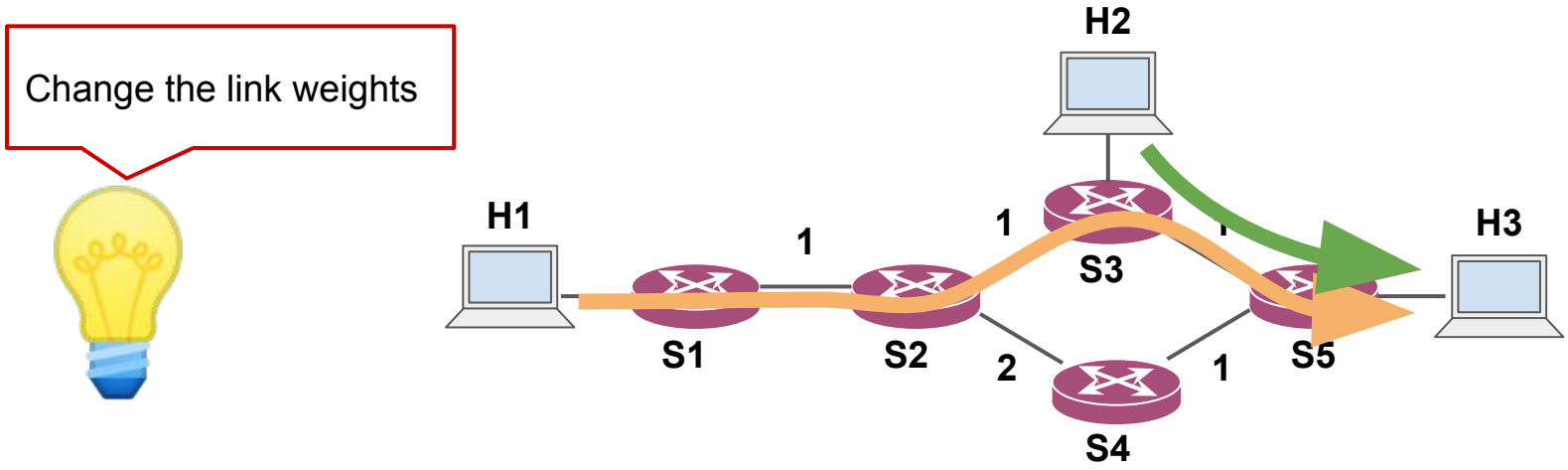


Traditional networks - "Indirect" path selection

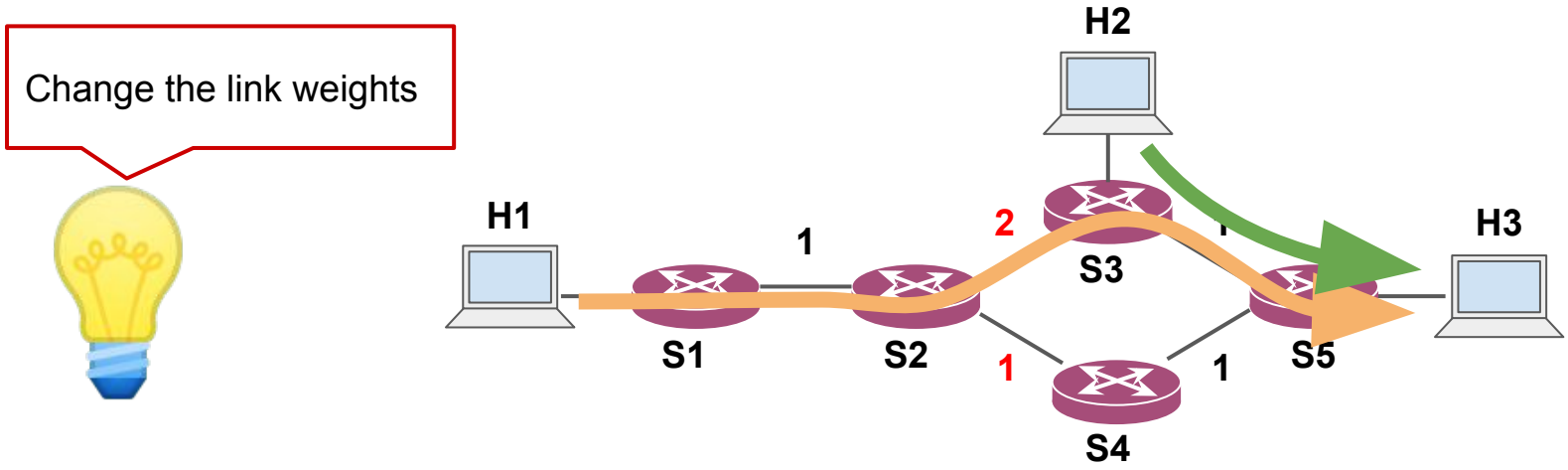
How can we get H1 to use the bottom path?



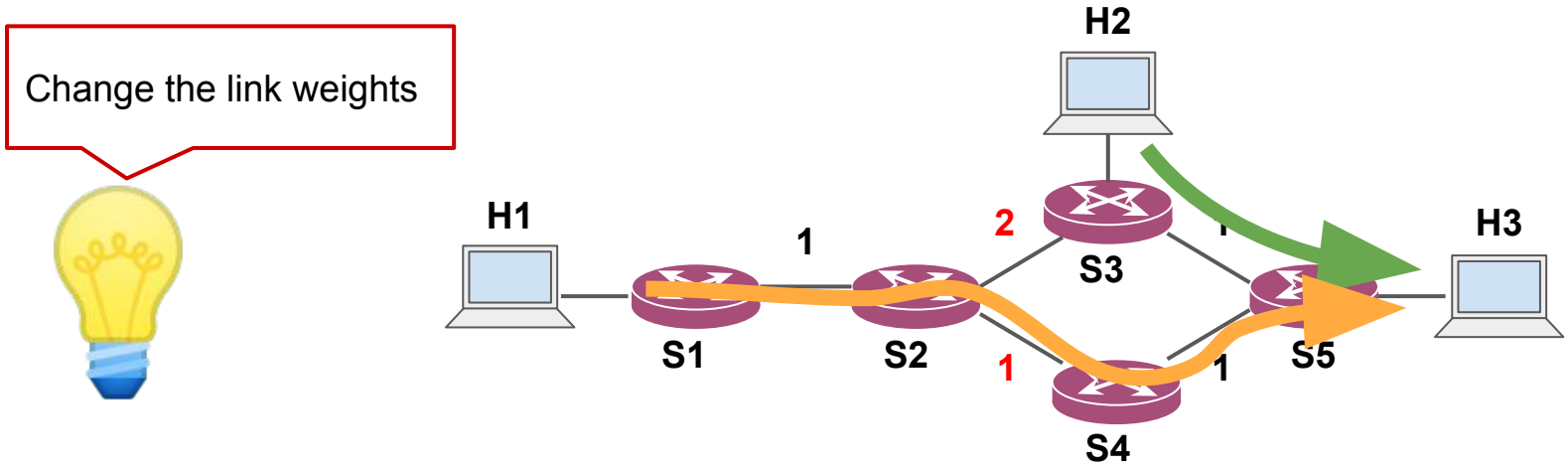
Traditional networks - "Indirect" path selection



Traditional networks - "Indirect" path selection



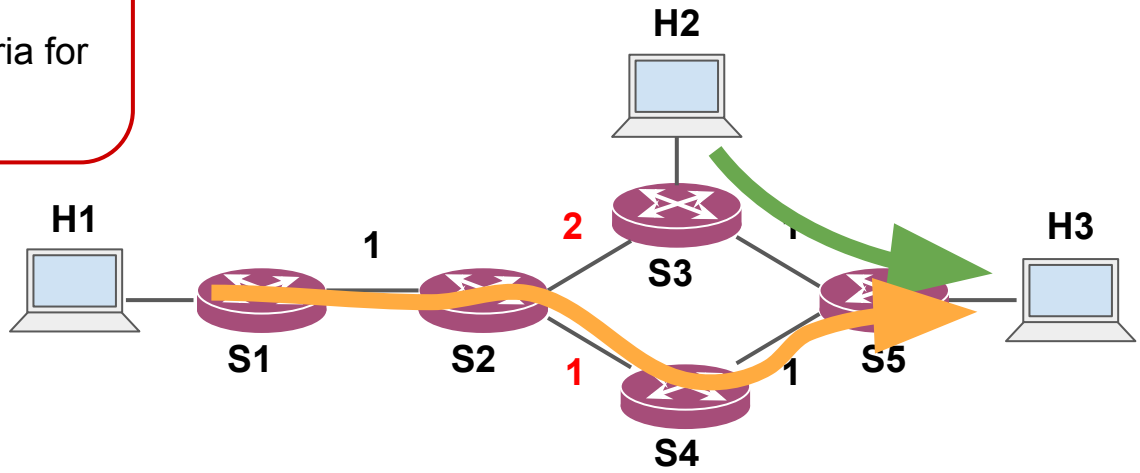
Traditional networks - "Indirect" path selection



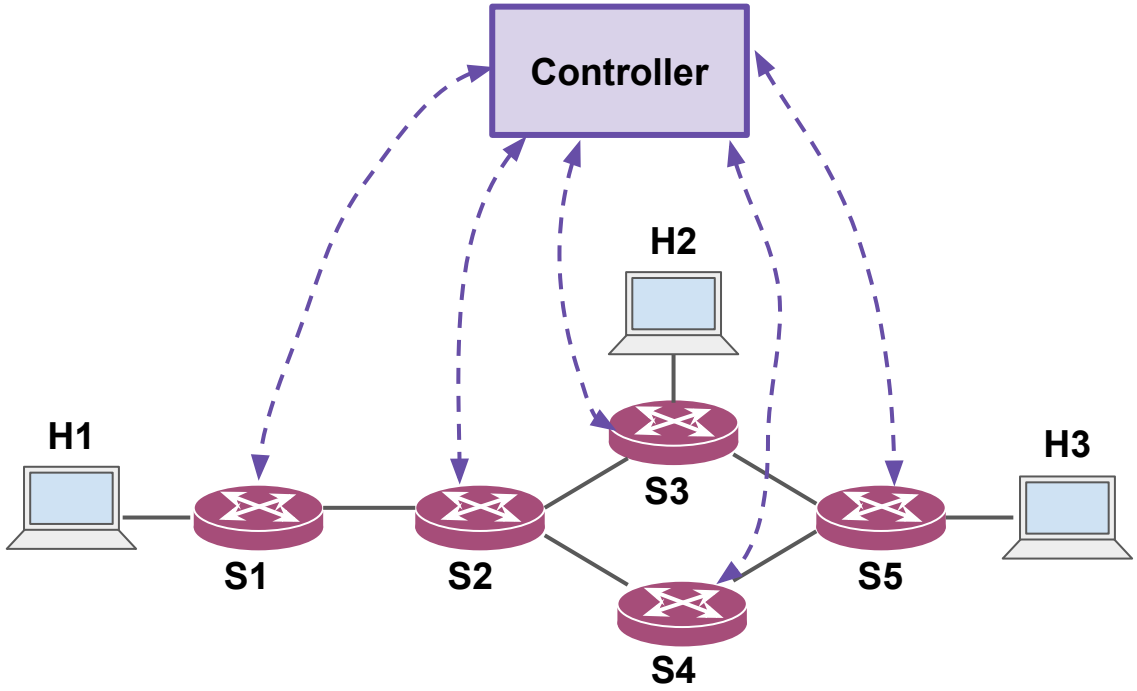
Traditional networks - "Indirect" path selection

What if our network had thousands of devices and links?

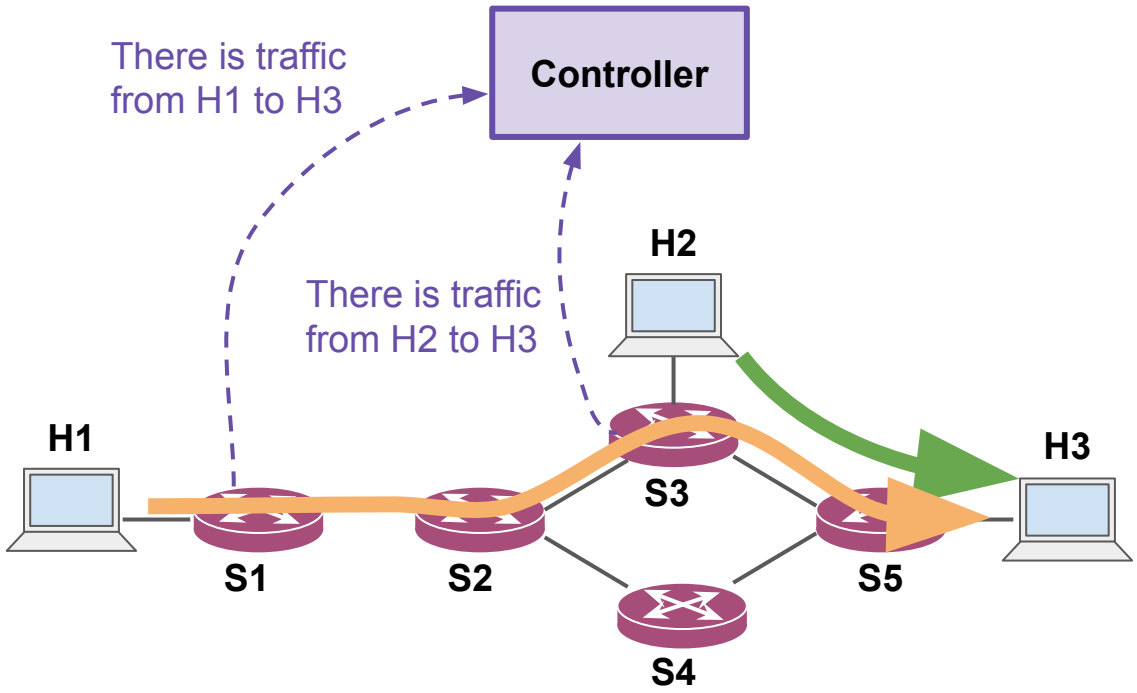
What if we had more complex criteria for selecting forwarding paths?



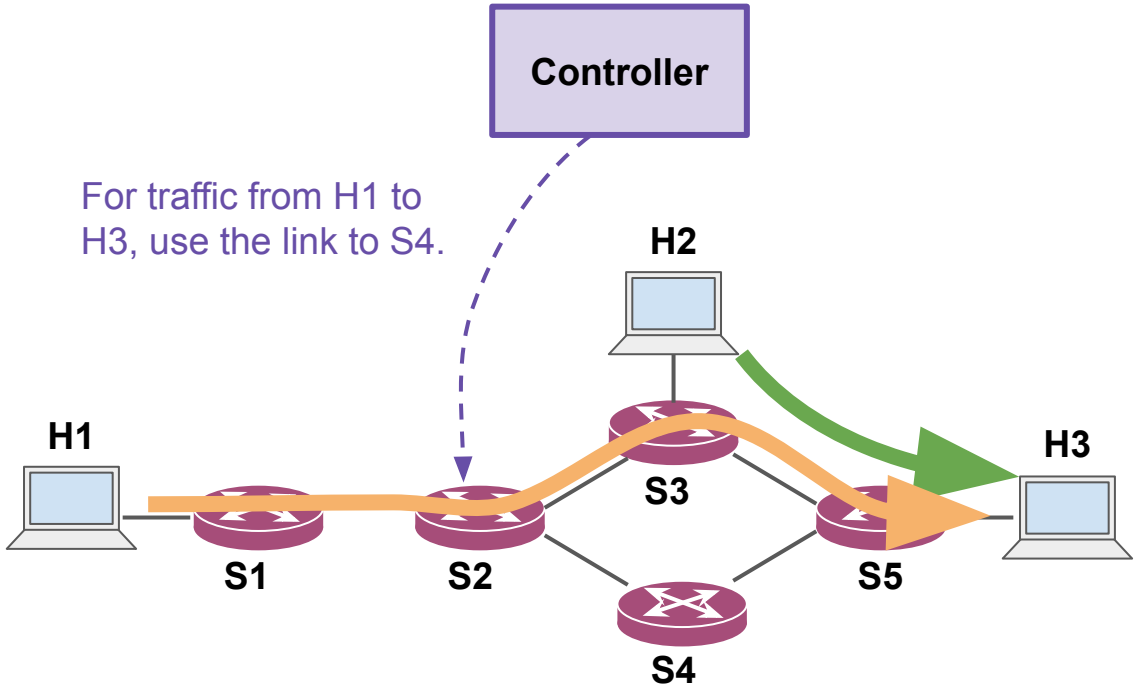
What if we had global visibility and direct control?



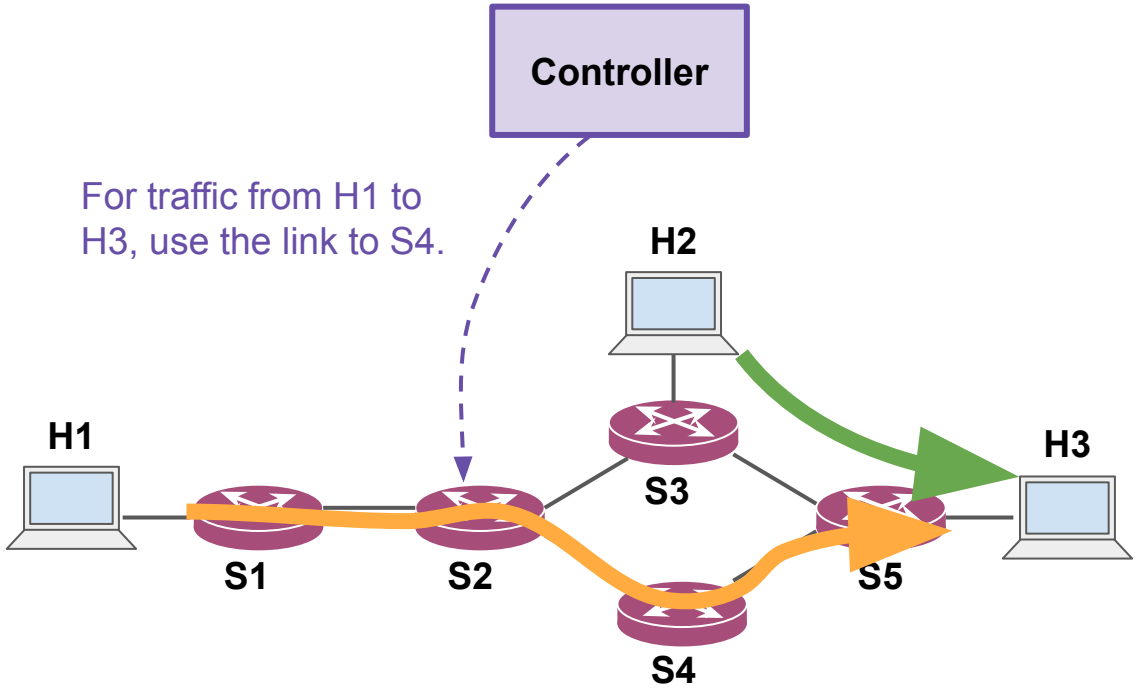
What if we had global visibility and direct control?



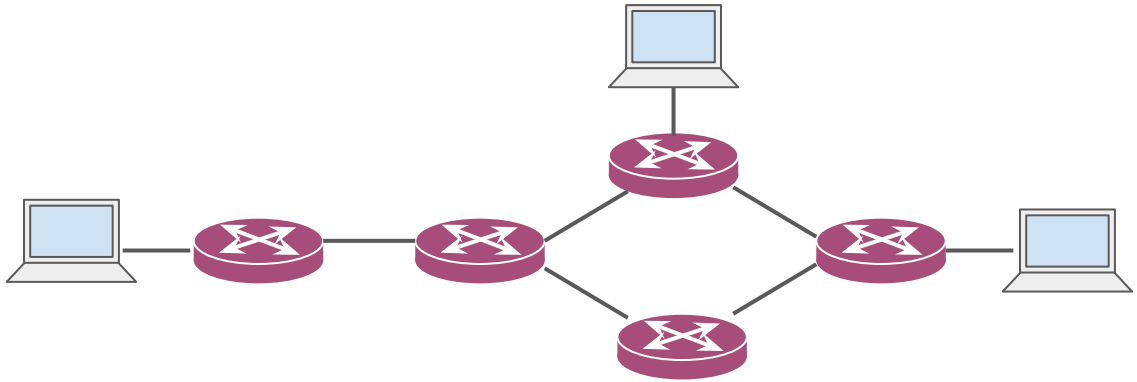
What if we had global visibility and direct control?



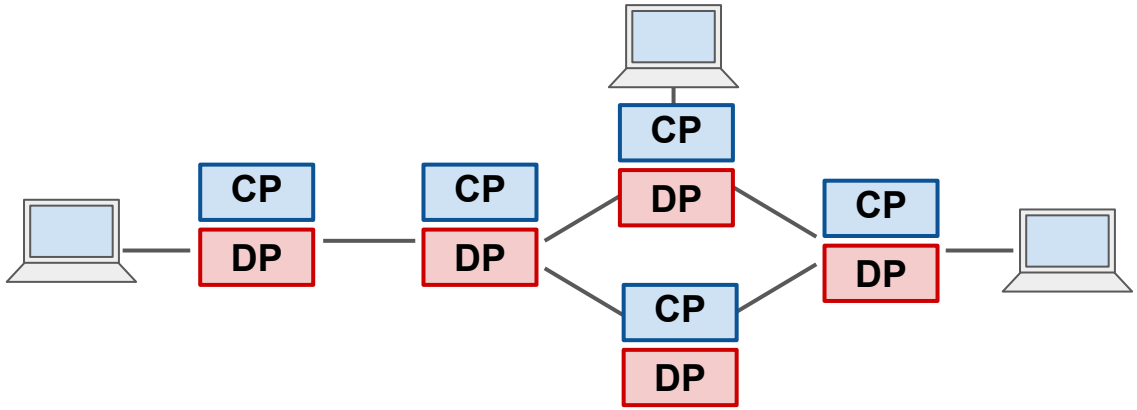
What if we had global visibility and direct control?



Software-Defined Networking (SDN)

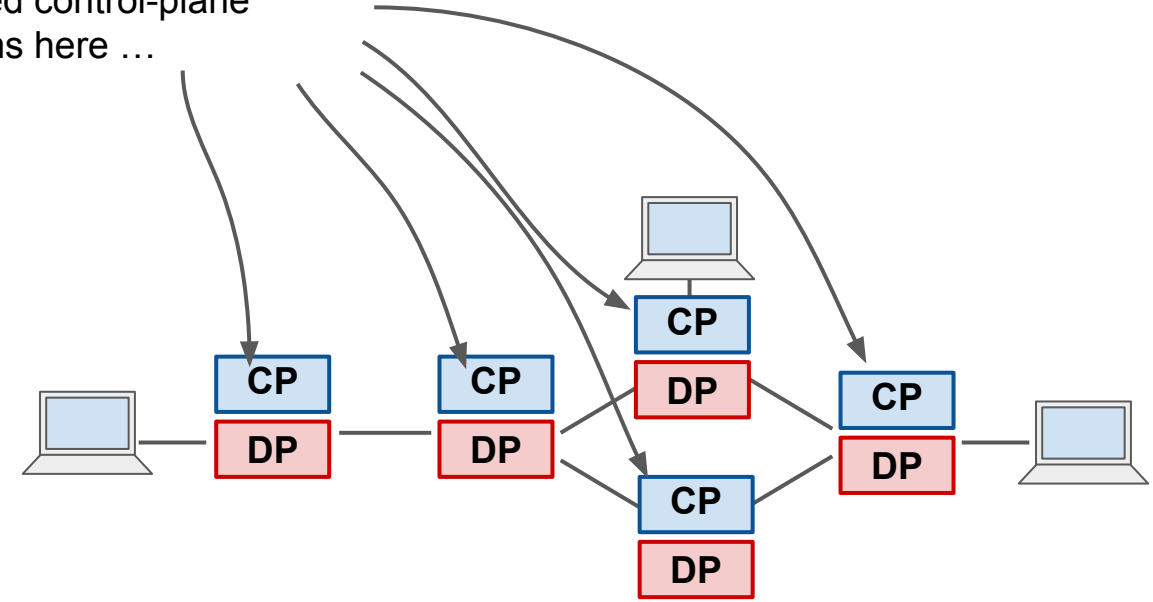


Software-Defined Networking (SDN)



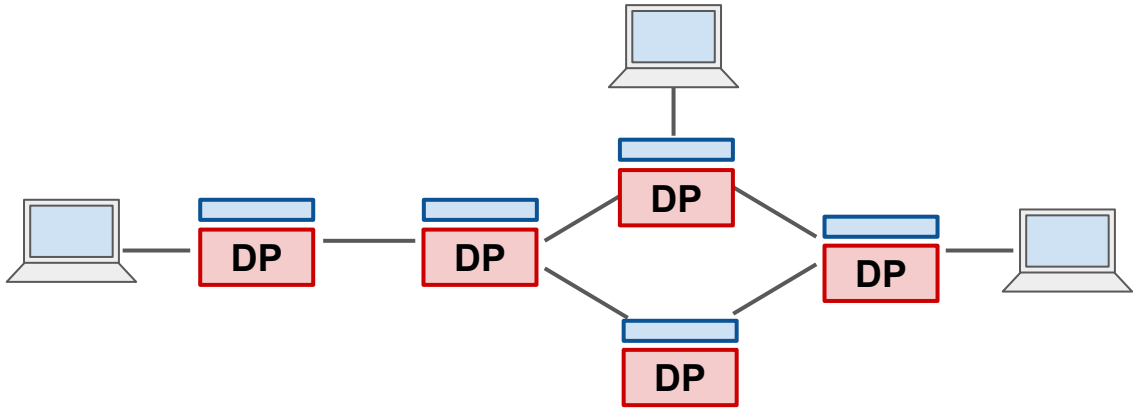
Software-Defined Networking (SDN)

Instead of running (complex) distributed control-plane algorithms here …

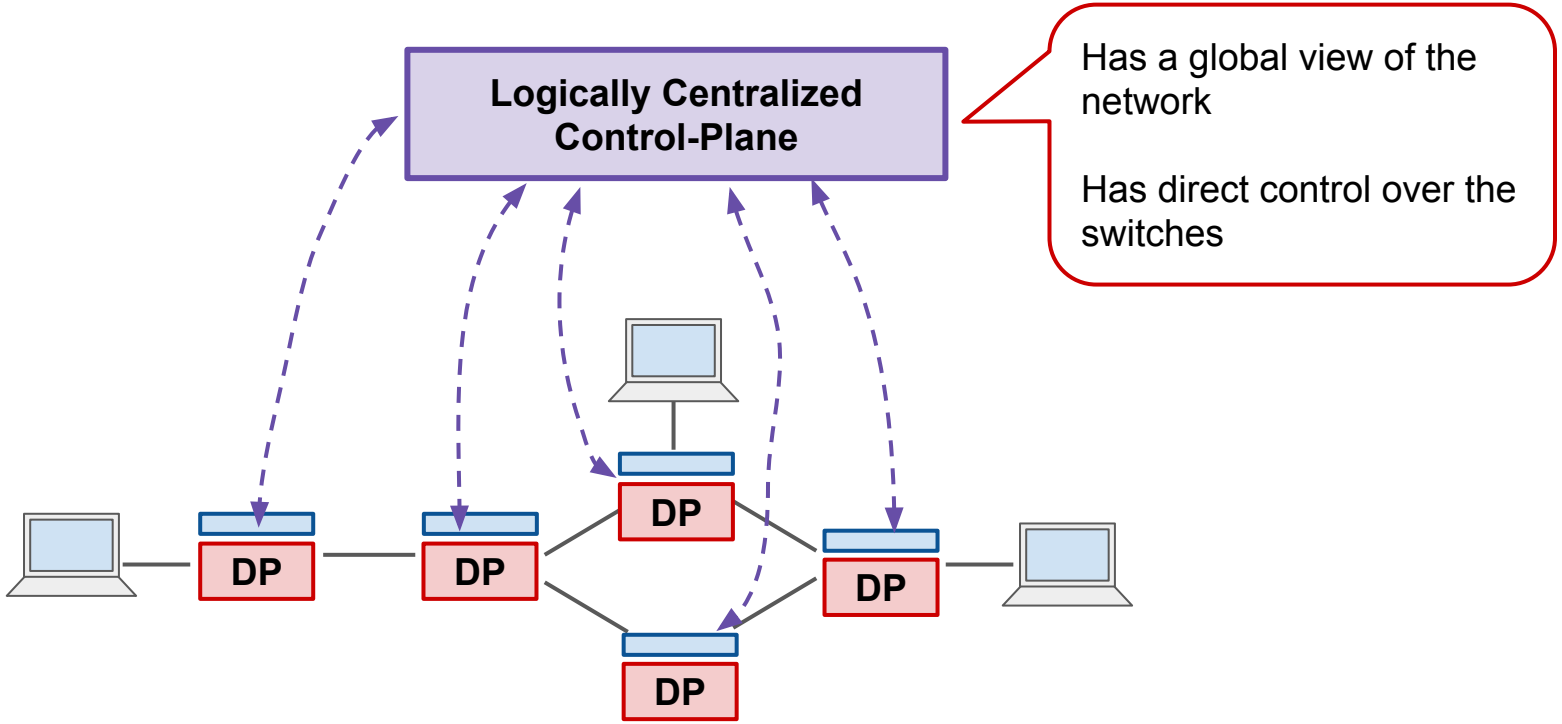


Software-Defined Networking (SDN)

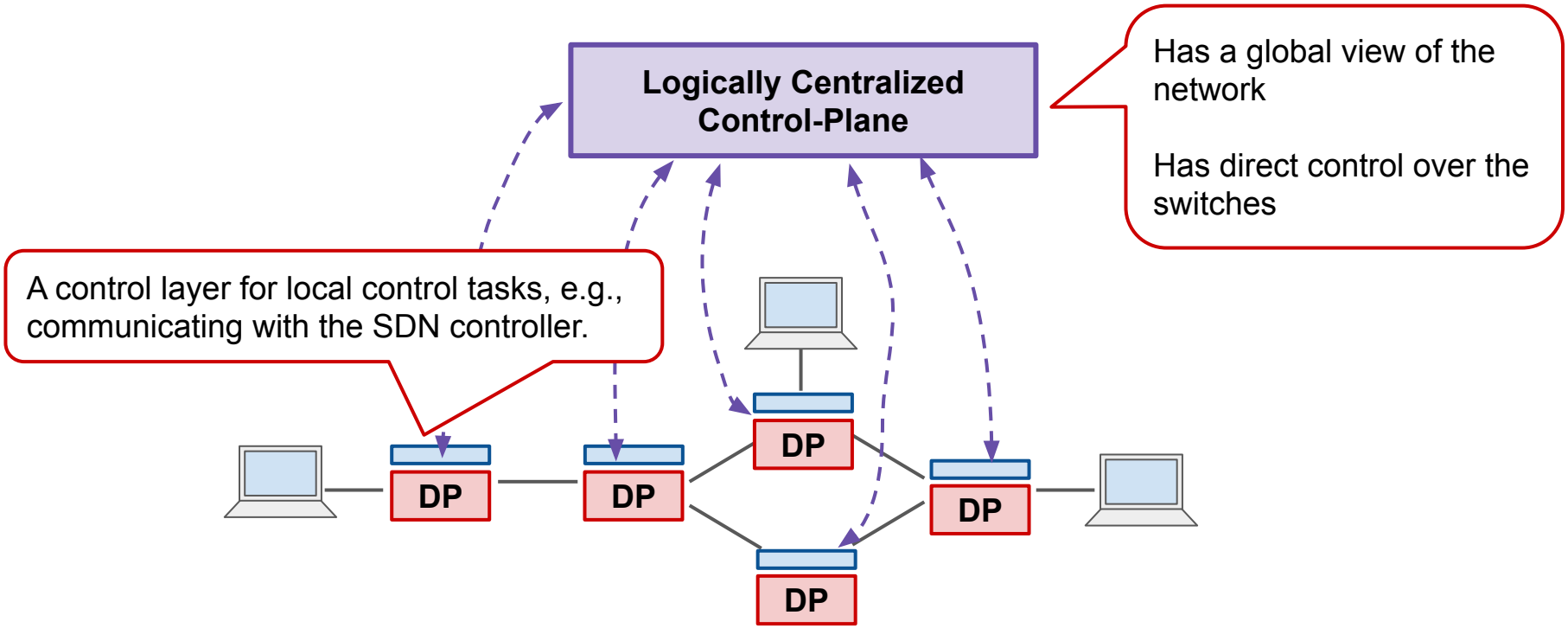
Run them here, and directly tell the switch data plane how to forward traffic.



Software-Defined Networking (SDN)



Software-Defined Networking (SDN)



Software-Defined Networking (SDN)

- SDN provides global visibility and direct control
- Why software-defined?
- Because the "software" running on the SDN controller will "define" the behavior of the network
    - As opposed to the interactions of several instances of a distributed protocol.

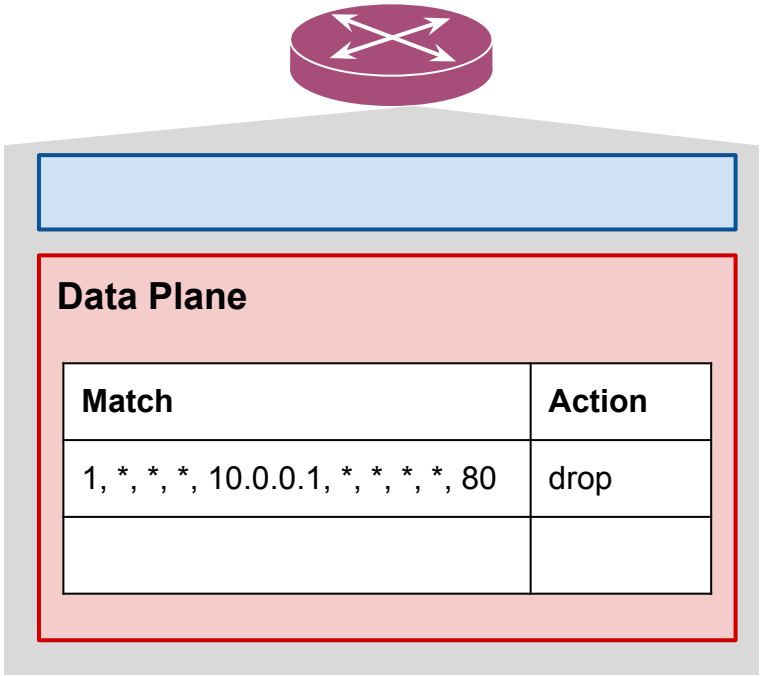
The controller-switch communication

- The controller-switch communication is an integral part of SDN
- An early (and quite popular) proposal for such a communication protocol was OpenFlow

OpenFlow - The early days

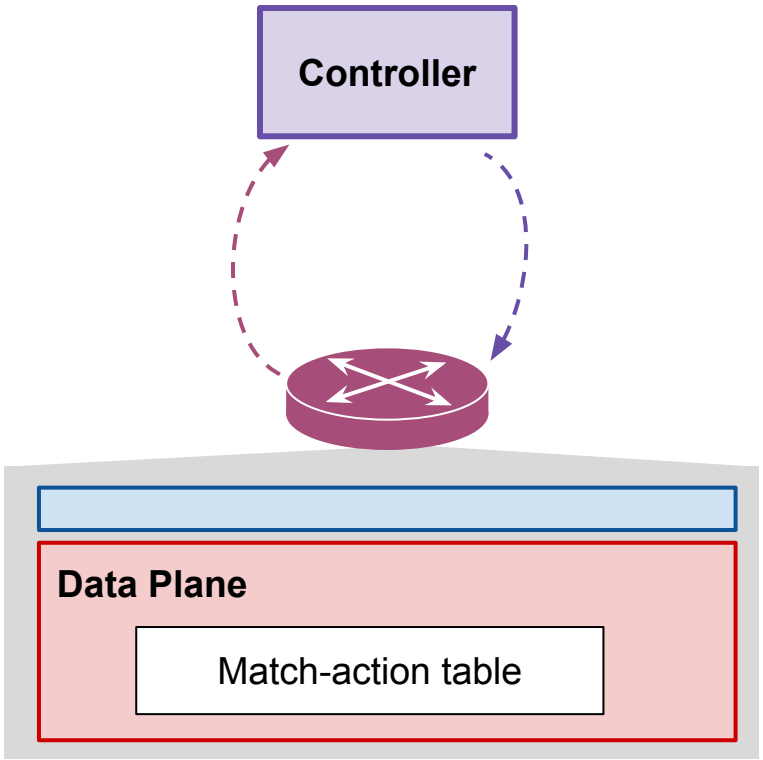
- Abstracts the switch data plane as one big look-up table
- When a packet comes in
  - Extract the relevant headers from it
  - See if it matches any table entries
  - Execute the corresponding action

OpenFlow - The early days



- Match
  - Input port
  - Ethernet header fields (src, dst, type)
  - Some IP header fields (src, dst, proto)
  - Some TCP header fields (src port, dst port)
- Action
  - drop
  - forward to port N
  - send to controller
  - modify the value of a field

OpenFlow - The early days



- Switch to Controller
  - connect
  - disconnect
  - status of ports
  - packet (e.g., if it matches no rules)
  - traffic statistics
- Controller to Switch
  - add/remove/modify table entries
  - packet
  - request traffic statistics

OpenFlow - The early days

- OpenFlow became quite popular
- It was simple
- Yet, it captured the essence of how many network devices process packets
    - MAC learning, IP forwarding, Access control (ACL), NAT, …

OpenFlow - Today

- It has become a lot more complicated
  - Multiple tables
  - More headers, actions, etc.
  - Bundled communication messages
  - …
- It is still a popular abstraction for configuring different components of switches, routers, network interface cards (NICs), middleboxes, etc.

Paper 1: A clean slate 4D approach to network control …

- This paper was published in 2005.
  - SDN didn't exist back then.
  - The community was starting to think more seriously about direct control and global visibility.
- From the paper's introduction: *"Our goal for this paper is not to prove that 4D is the best approach. [...] Rather, by presenting a specific design alternative that is radically different from today's approach, [...], we want to highlight the issues that need to be considered in a clean slate design of network control and management."*
- Keep the above quote in mind when reading the paper. Do you think they achieved their goal?

Paper 1: A clean slate 4D approach to network control …

- There are a lot of acronyms in networking 😅
- If you can't figure out what they mean, ask on Slack
- Examples from this paper
  - Autonomous Systems (AS)
  - OSPF, IS-IS, and EIGRP
  - MPLS (and tunneling in general)
  - FIB
  - …

Paper 2: Frenetic: A network programming language

- This paper was published in 2011
- SDN and OpenFlow had been around for a couple of years.
- OpenFlow had done a great job of abstracting away the low-level details of the data plane.
- But, people were realizing that we may need higher-level abstractions on top of OpenFlow to make programming the network easier.

Paper 2: Frenetic: A network programming language

- This paper proposes *a new programming language* to describe network policies and queries.
  - Policies describe how packets should b processed in the network
  - Queries describe what kind of information operators would like to get from the network.

Paper 2: Frenetic: A network programming language

- Operators describe policies and queries in this language on the controller.
- Frenetic's runtime system makes sure the controller and switches interact in a way that makes those policies and queries happen.

Paper 2: Frenetic: A network programming language

- When proposing a new language, the authors describe its *syntax* and *semantics*
- Syntax specifies what valid programs look like
  - In text with examples
  - More formally as a grammar
- Semantics specifies what a program means
  - i.e., what happens when you execute the program

Paper 2: Frenetic: A network programming language

Queries	q	::= `Select`(a) * `Where`(fp) * `GroupBy`$([qh_1, \ldots, qh_n])$ * `SplitWhen`$([qh_1, \ldots, qh_n])$ * `Every`(n) * `Limit`(n)
Aggregates	a	::= `packets` | `sizes` | `counts`
Headers	qh	::= `inport` | `srcmac` | `dstmac` | `ethtype` | `vlan` | `srcip` | `dstip` | `protocol` | `srcport` | `dstport` | `switch`
Patterns	fp	::= `true_fp`() | qh`_fp`(n) | `and_fp`$([fp_1, \ldots, fp_n])$ | `or_fp`$([fp_1, \ldots, fp_n])$ | `diff_fp`(fp_1, fp_2) | `not_fp`(fp)

Figure 3. Frenetic query syntax

Additional Resources

- The original OpenFlow paper (2008)
- NetKAT: A Frenetic-like network programming language, but with a heavier mathematical foundation and treatment (2014)
  - There has been a long line of research (still ongoing) on NetKat-like family of network programming languages.

Don't forget 🙂

- Join Slack and HotCRP
- Sign up for 10-minute paper presentations
- If you need help with project ideas, come find me after class
- First round of reviews are due **Monday at 5pm**.