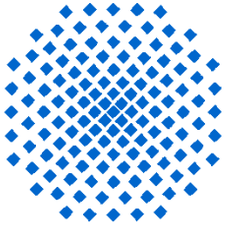


Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Lab-course / Fachpraktikum

Computer Communication: Software-defined Networking

Winter Term 20/21

Assignment 3

Reactive Routing, Adaptive Load Balancing

December 15th, 2020

Sukanya Bhowmik, David Hellmanns

Overview

- **Task 3**
  - **Goals of this task**
  - 3.1 – Centralized ARP Handling **[7 points]**
  - 3.2 – Reactive Routing **[8 points]**
  - 3.3 – Adaptive Link Load Balancing **[5 points]**
- Deadline and Submission

Goals of this Task

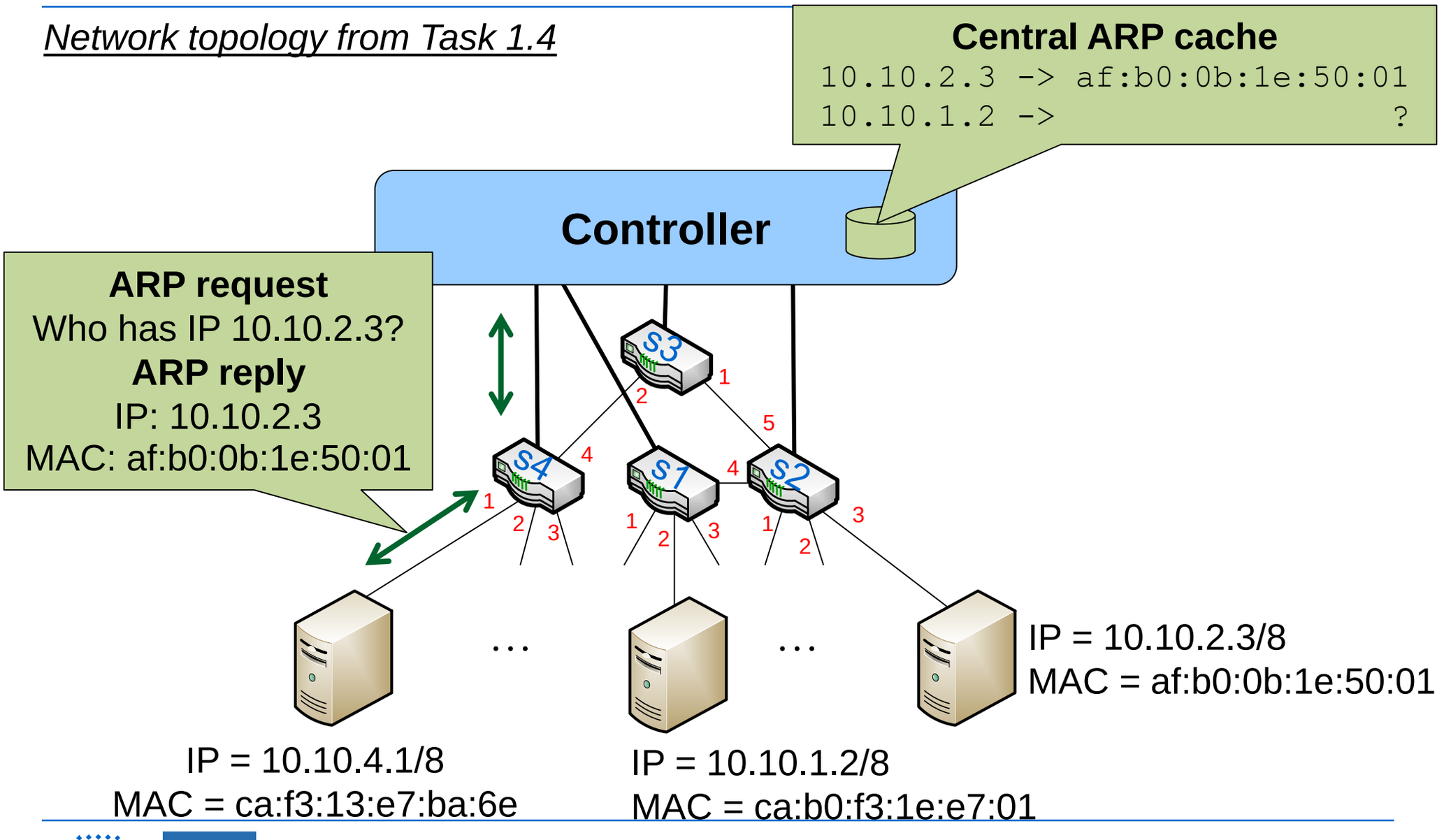
- Build and inject packets from the controller
- Keep data plane state at the controller
- Install flow entries using Java API
- Query traffic statistics from switches

Overview

- **Task 3**
  - Goals of this task
  - **3.1 – Centralized ARP Handling**
  - 3.2 – Reactive Routing
  - 3.3 – Adaptive Link Load Balancing
- Deadline and Submission

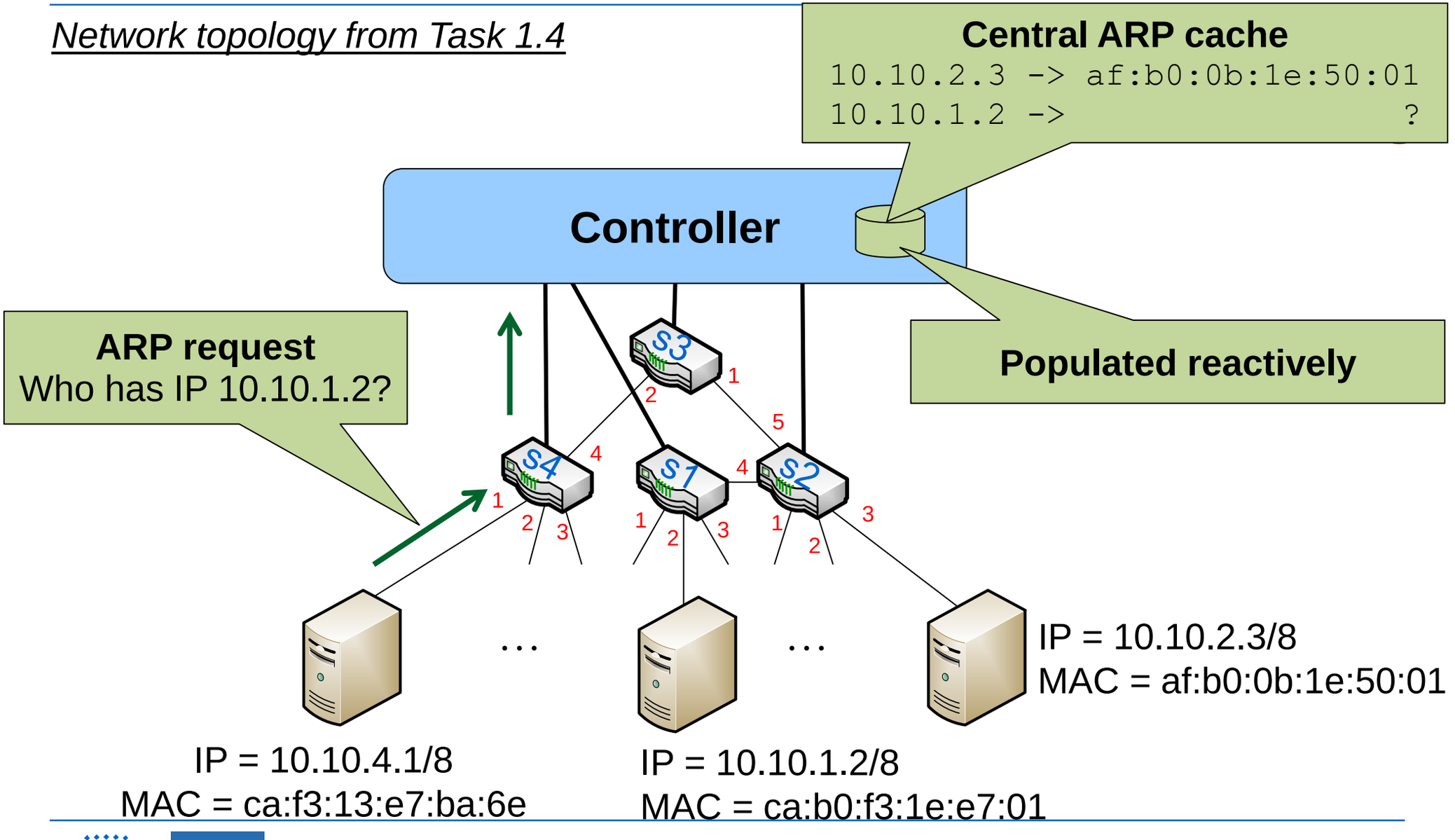
Task 3.1 – Centralized ARP Handling (1)

Network topology from Task 1.4



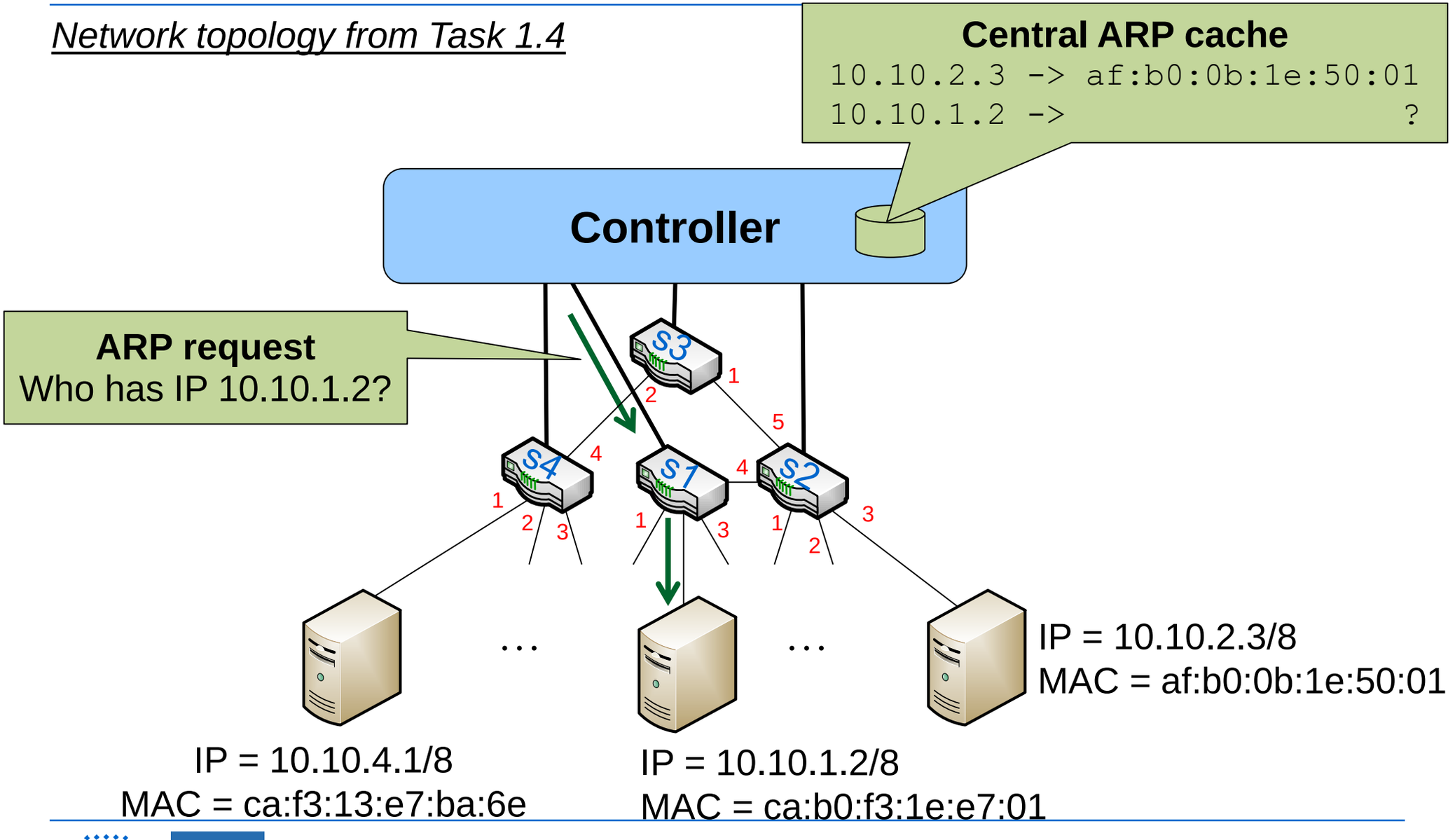
Task 3.1 – Centralized ARP Handling (1)

Network topology from Task 1.4



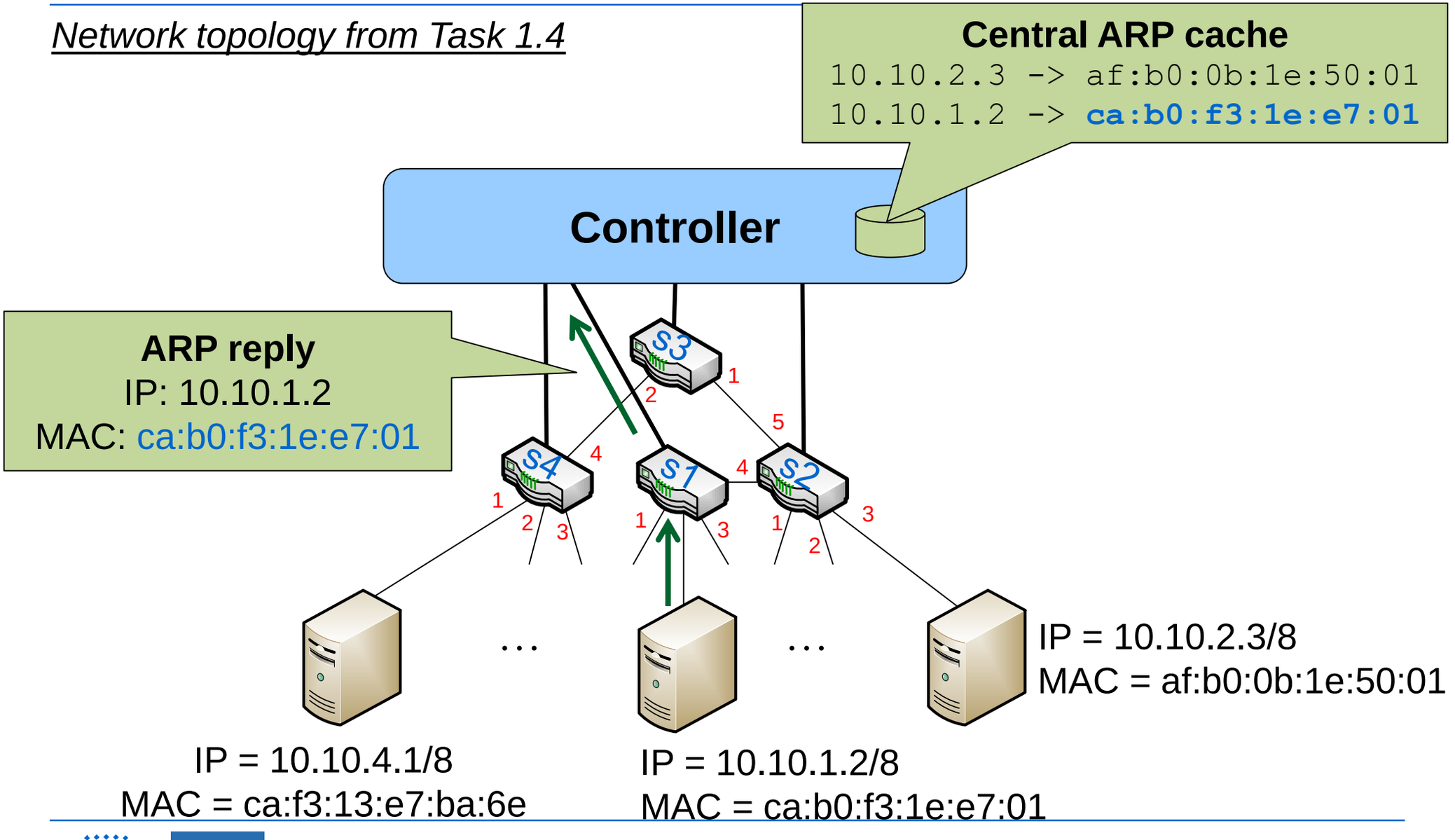
Task 3.1 – Centralized ARP Handling (1)

Network topology from Task 1.4



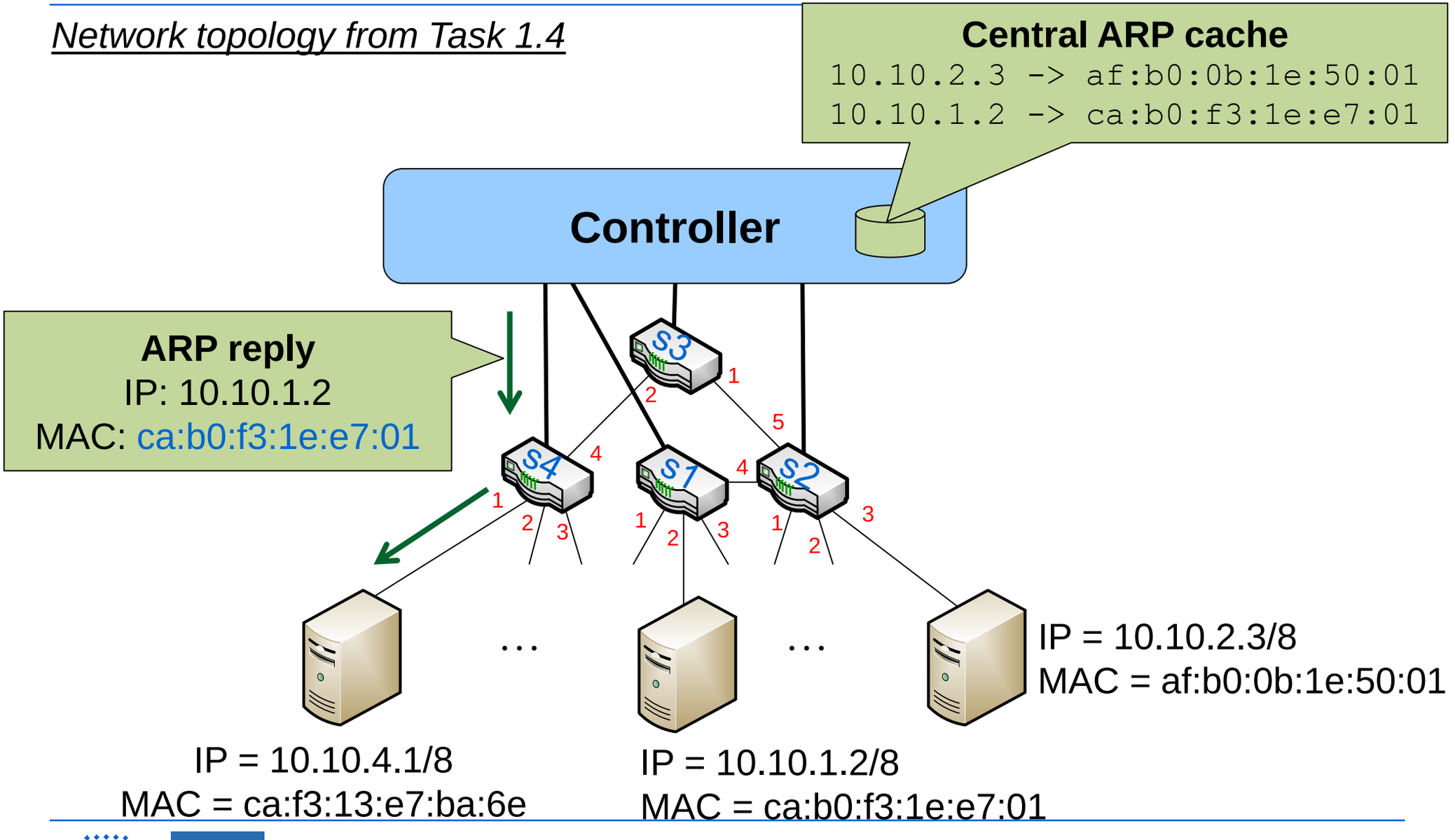
Task 3.1 – Centralized ARP Handling (1)

Network topology from Task 1.4



Task 3.1 – Centralized ARP Handling (1)

Network topology from Task 1.4



Task 3.1 – Centralized ARP Handling (2)

Implement your solution for this task as a Floodlight module ARPHandler.java in the package net.sdnlab.ex3.task31

- Only the controller handles ARP requests / replies from hosts
- You may assume that the topology and host IP addresses are known to the controller
  - Switch redirects incoming ARP request to the controller
  - Controller queries internal ARP cache for MAC address
    - If it contains a corresponding entry, immediately inject appropriate reply
    - Otherwise, first redirect the ARP request to the target host and save the reply to its internal ARP cache, before injecting reply
- To deliver IP packets normally, install static flow table entries for IP packets using Floodlight's Java API! (cf. Task 2.3, Tutorial 3)

Task 3.1 – Centralized ARP Handling (5)

- Use Floodlight's `ARP` packet type, e.g.

```
if (eth.getEtherType() == EthType.ARP) {
  ARP arp = (ARP) eth.getPayload();
  if (arp.getOpCode() == ArpOpcode.REQUEST)
    arp.<get/set><Sender/Target><Hardware/Protocol>Address()
... }
```

- Test your solution with the topology from Task 1.4:
  - The topology of Task 1.4 is implemented in the file **`~/ex3/task14topo.py`**. Use the following command (without the --mac option!) to start mininet:

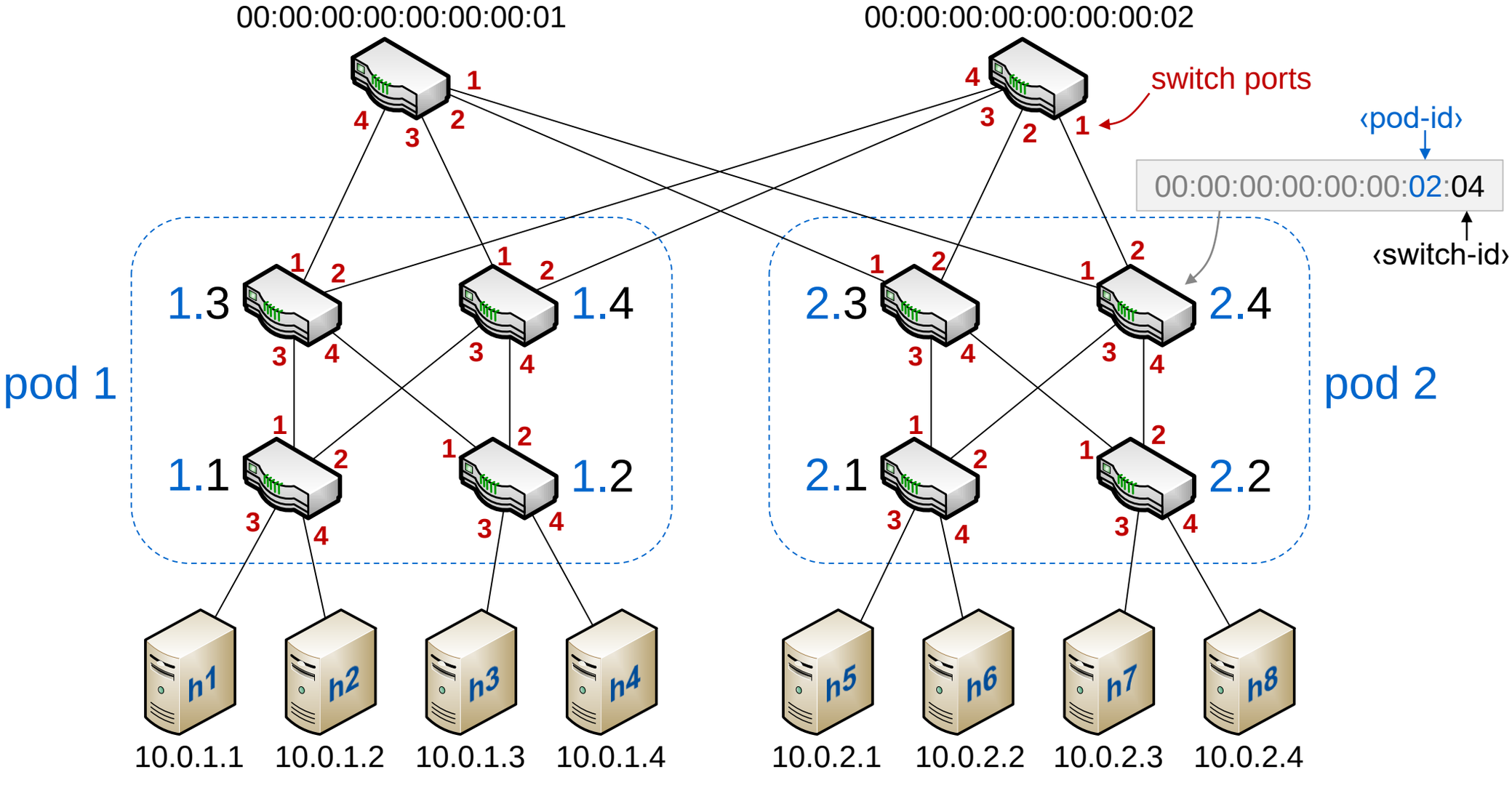
```
~$ sudo mn --switch ovsk --controller remote,port=6653
   --custom ~/ex3/task14topo.py --topo task14topo
```

  - Afterwards, run **`pingall`** in the Mininet CLI

Overview

- **Task 3**
  - Goals of this task
  - 3.1 – Centralized ARP Handling
  - **3.2 – Reactive Routing**
  - 3.3 – Adaptive Link Load Balancing
- Deadline and Submission

Task 3.2 – Reactive Routing



Task 3.2 – Reactive Routing

- For the reactive routing task, use the topology shown on previous slide (**`~/ex3/fattree.py`**) to route IP packets reactively
- To test your solution, run mininet as follows

```
~$ sudo mn --switch ovsk --controller remote,port=6653
   --custom ~/ex3/fattree.py --topo fattree --arp
```

- ***Note*** *that you can assume for this and the remaining task that the ARP caches of all hosts are filled! You don't have to handle ARP requests or implement MAC flooding!*

Task 3.2 – Reactive Routing

- Implement your solution for this task as a Floodlight module Reactive.java in the package net.sdnlab.ex3.task32
  - The Controller only sets up IP (layer 3 match) forwarding entries
  - Routes should be calculated centrally at the controller using Dijkstra's shortest path algorithm (min. hops) in a reactive manner
    - Inject the first packet directly at the target switch
    - Please indicate the source of the Dijkstra implementation you use! E.g., you may copy and modify the dijkstra() method from net.floodlightcontroller.topology.TopologyInstance.java
  - You should use Floodlight's topology information base (ILinkDiscoveryService, cf. Tutorial 5) for core links, but you may hard-code edge links (i.e., switch/port for each host)

Task 3.2 – Reactive Routing

- Hint for the usage of **ILinkDiscoveryService**
  - Wait until topology is detected
  - Use *return Command.CONTINUE*
    - Otherwise topology will not be detected correctly!

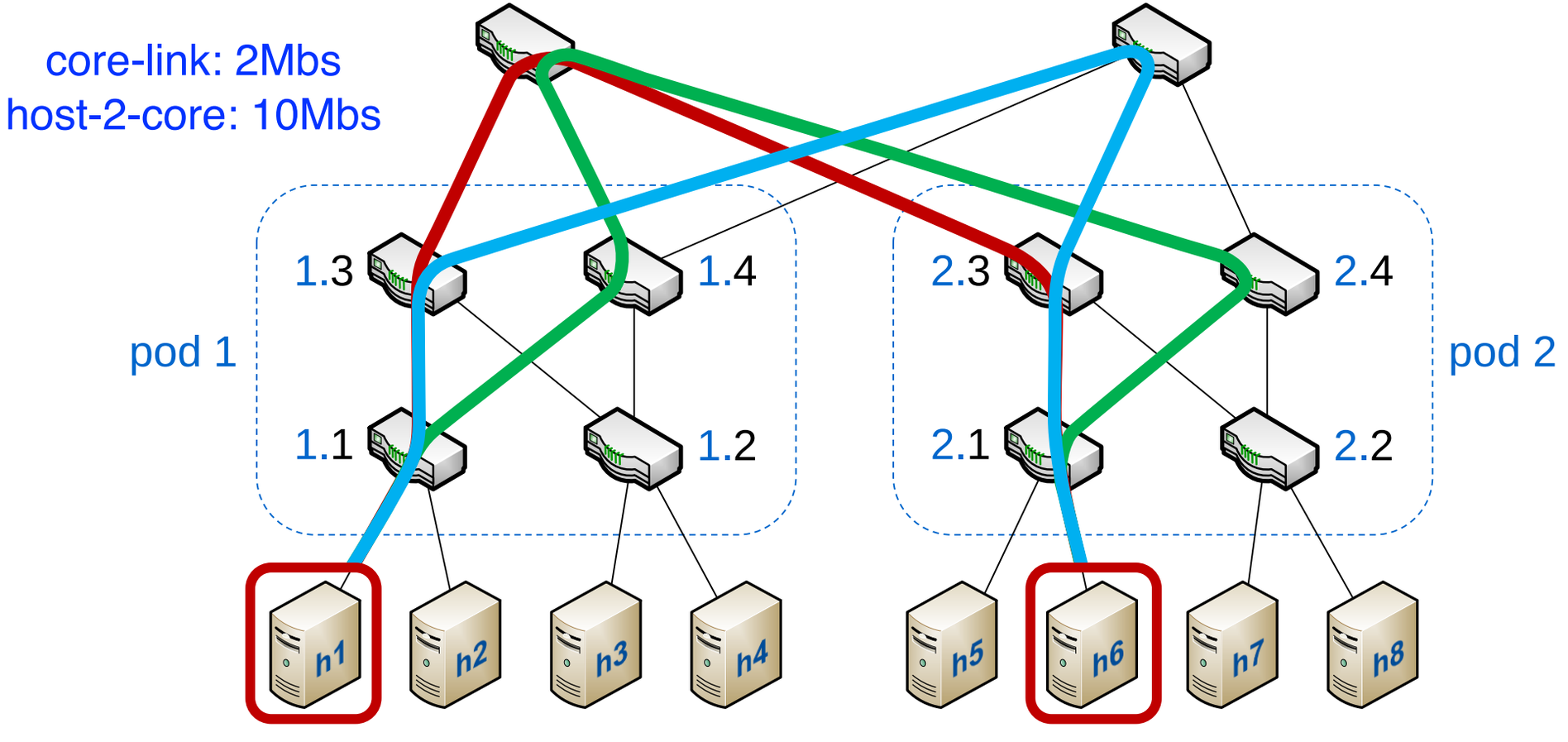
Overview

- **Task 3**
  - Goals of this task
  - 3.1 – Centralized ARP Handling
  - 3.2 – Reactive Routing
  - **3.3 – Adaptive Link Load Balancing**
- Deadline and Submission

Task 3.3 – Adaptive Load Balancing

Equal cost (multi-)paths:

Problem: shortest path commonly finds only **one**… → bottlenecks



Task 3.3 – Adaptive Load Balancing

- Therefore, the solution from Subtask 3.2 is to be extended to route different TCP flows (even between the same hosts) over different paths to distribute the load evenly over all links, therefore,…
  - … use estimated bandwidth based on port counters as link weight
  - … match flows based on **both** layer 3 **and** layer 4:
    - Src/dst IP addresses
    - Src/dst TCP ports

Task 3.3 – Adaptive Load Balancing

- Implement your solution for this task as a Floodlight module Adaptive.java in the package net.sdnlab.ex3.task33
  - Use IStatisticsService as a dependency. This module periodically queries port statistics and calculates bandwidth estimates, e.g.

```
IStatisticsService stats = …;
SwitchPortBandwidth spbw = stats.getBandwidthConsumption(dpid,p);
weight = spbw.getBitsPerSecondTx() + 1;    // ensure weight > 0!
```

  - Enable statistics collection, either in startUp() with stats.collectStatistics(true) (preferred) or in .properties
  - Reduce statistics collection interval from 10 seconds **to 2 seconds** in your .properties file for this task
  - Consulting (java)doc of the StatisticsService is recommended

Task 3.3 – Adaptive Load Balancing

- Print debug information each time a new flow is installed:
  1. Flow specification (srcIP, dstIP, srcPort, dstPort)
  2. Selected routing path for this flow (sequence of visited switches)
  3. Edge weights of each edge on this path (can be converted to different unit for better readability)

- Test your solution with `sudo ~/ex3/mininet3.py`
  - This loads the fat tree topology with bandwith limits (max. 10Mbit/s on edge links and max. 2Mbit/s on core links)
  - Run `loadtest` in the Mininet CLI, which measures the throughput on four connections between two hosts (h1 and h8), waiting 2 seconds between the start of new flows

Task 3.3 – Comparison

- Compare the adaptive routing solution of the two tasks Task 3.2 and Task 3.3 with regard to throughput.
- For both solutions, start the Mininet script:

```
~$ sudo ~/ex3/mininet3.py
```

this will also fill the ARP caches of all hosts

- Use loadtest provided in the CLI of this Mininet instance:

```
mininet> loadtest
```

  - Compare the **overall throughput** for both routing approaches
  - Briefly explain the numbers for both approaches!

Loadtest Output

```
mininet> loadtest
*** Throughput test between h1 and h8
  * This should take at least 40 seconds (possibly much longer!)
    - Starting flows [====]
    - Waiting for flows to terminate...
  * Format:   [ ID] Interval       Transfer     Bandwidth
  * Flow  1: [ 43]  0.0-48.0 sec  [illegible] KBytes  [illegible] Kbits/sec
  * Flow  2: [ 43]  0.0-41.2 sec  [illegible] KBytes  [illegible] Kbits/sec
  * Flow  3: [ 43]  0.0-36.3 sec  [illegible] KBytes  [illegible] Kbits/sec
  * Flow  4: [ 43]  0.0-20.5 sec  [illegible] KBytes  [illegible] Kbits/sec
*** Summary: [ 47]  0.0-98.1 sec  [illegible] KBytes  [illegible] Kbits/sec
```

Overall throughput

Overview

- Task 3
- **Deadline and Submission**

Deadline and Submission

- When (submission deadline): January 12th 2021, 08:00am
- How: Via ILIAS system
  - One submission per group
  1. One document (PDF)
     - Describing the commands you executed to solve the tasks
     - Showing the output
     - Explanation
  2. Archive of source package net.sdnlab.ex3
- Be prepared to show a live demo to the supervisor during the next meeting