

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 4
Content-based Routing
January 12th, 2020

Sukanya Bhowmik, David Hellmanns

Overview

- **Task 4**
 - **Goals of this task**
 - 4.1 – Pub-Sub Routing [**6 points**]
 - 4.2 – Content-based Routing [**7 points**]
 - 4.3 – REST Interface for Content-based Routing [**7 points**]
- Deadline and Submission



Goals of this Task

Perform Content-based Routing for OpenFlow Networks:

- For static subscriptions via StaticEntryPusher (without & with in-network filtering)
- For dynamic subscriptions via own REST Interface



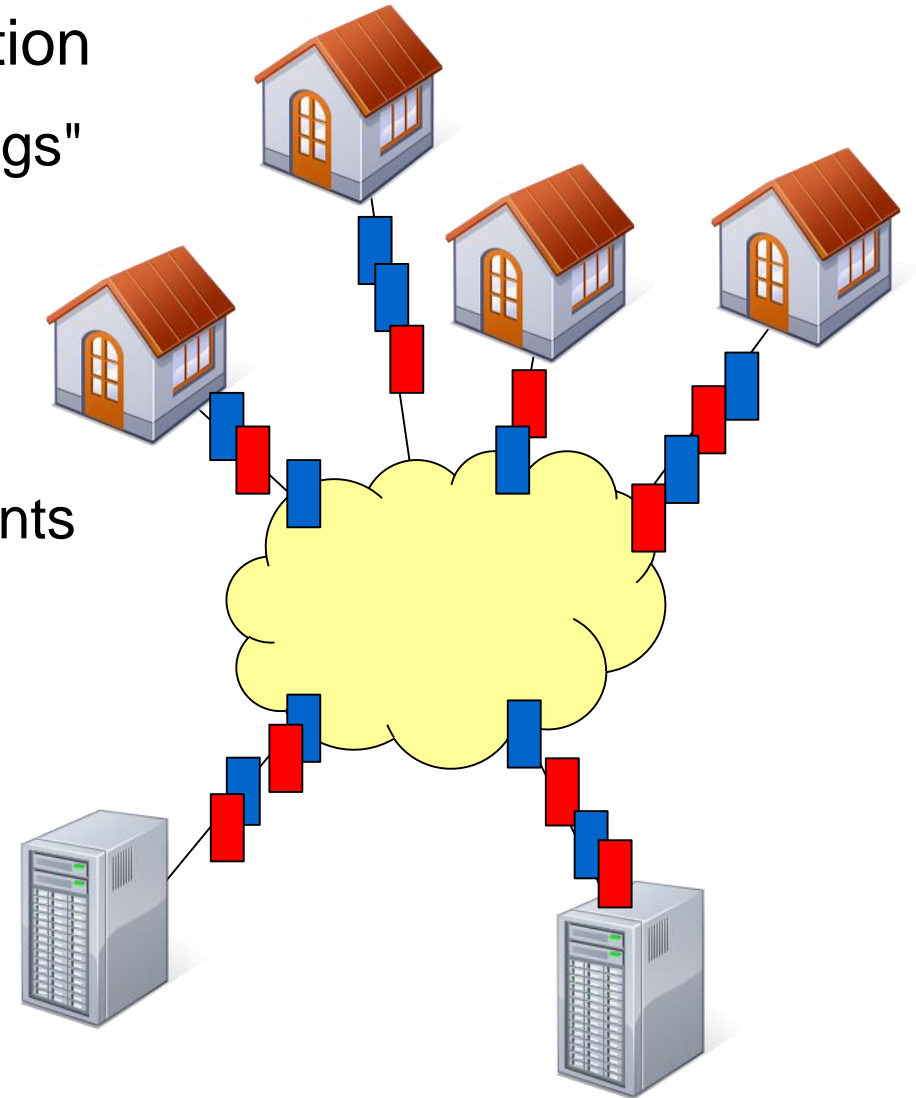
Overview

- **Task 4**
 - Goals of this task
 - **4.1 – Pub-Sub Routing**
 - 4.2 – Content-based Routing
 - 4.3 – REST Interface for Content-based Routing
- Deadline and Submission



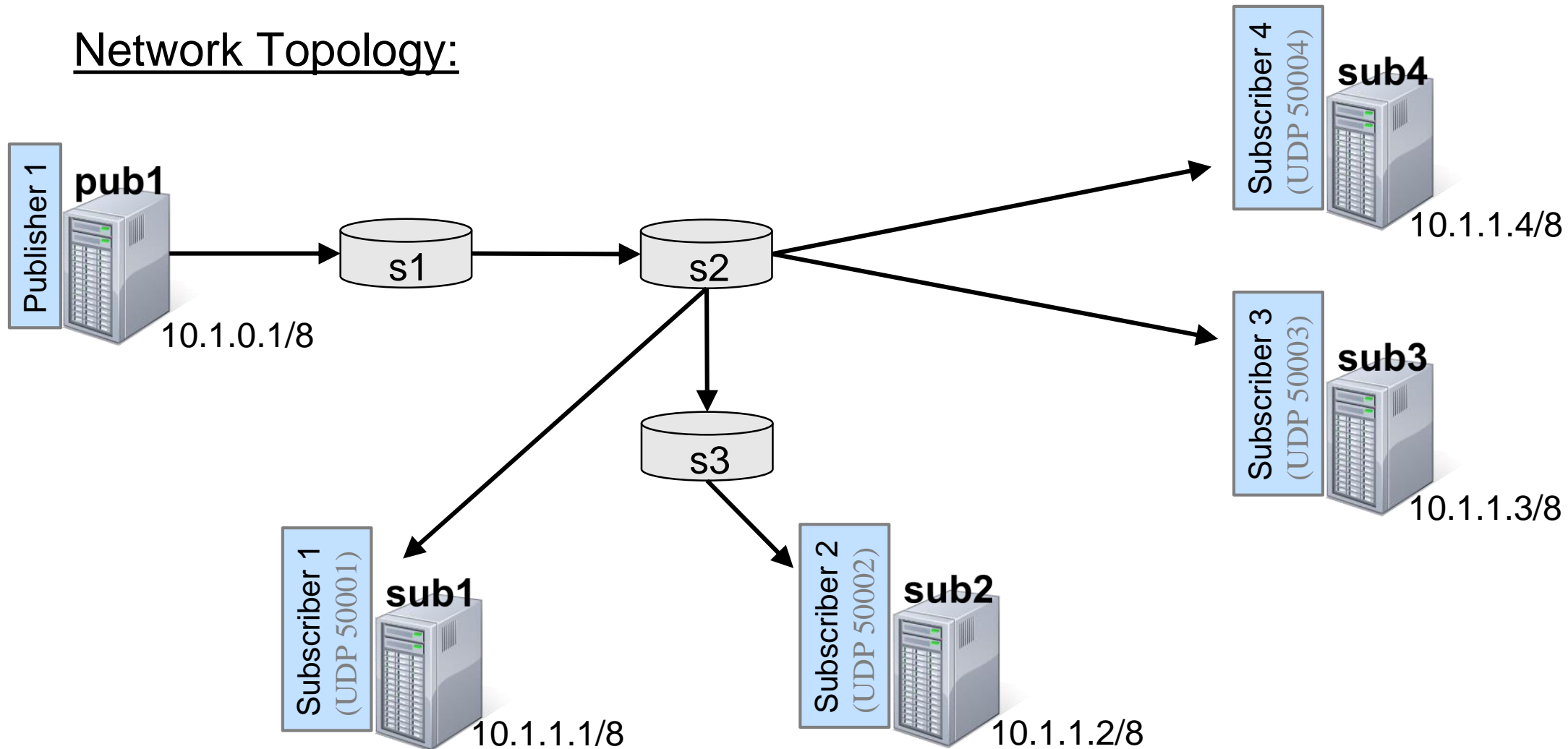
Task 4.1 – Pub-Sub Routing

- Measure residential energy consumption
 - Households equipped with "smart plugs"
 - Periodically "publish" energy and power measurements
 - "Subscribers" filter these measurements (e.g. by value, source, time, ...) and perform some data analysis
 - *Naive networking*: forward all measurements to all subscribers



Task 4.1 – Pub-Sub Routing

Network Topology:



IPVS

Research Group

“Distributed Systems”

Task 4.1 – Pub-Sub Routing

- First, write a "subscriber" application (pref. Python or Java) which receives UDP datagrams containing measurement data and possibly filters them by measurement type (0 for energy and 1 for power) and value (greater, less or equal to a certain reference value). Write to **stdout**:
 - matching measurements as they arrive
 - a report of min/max values of all **received** measurements and min/max values of all **matching** measurements at exit
- Your application should take parameters to define
 - the listening UDP port
 - the type (0/1)
 - the reference value
 - whether to filter at all
 - the comparison operator ($>$ or \leq)



Task 4.1 – Pub-Sub Routing

- Last, run two experiments with `~/ex4/mininet4.py`, which implements the previously shown topology, and `/opt/floodlight/floodlight-noforwarding.sh`
 1. All datagrams are forwarded to the subscribers and printed out, unfiltered by the application
 2. All datagrams are forwarded to the subscribers and filtered by the application
- To publish measurements, call `~/ex4/publish` on node `pub1`



Task 4.1 – Pub-Sub Routing

Run in total four subscribers:

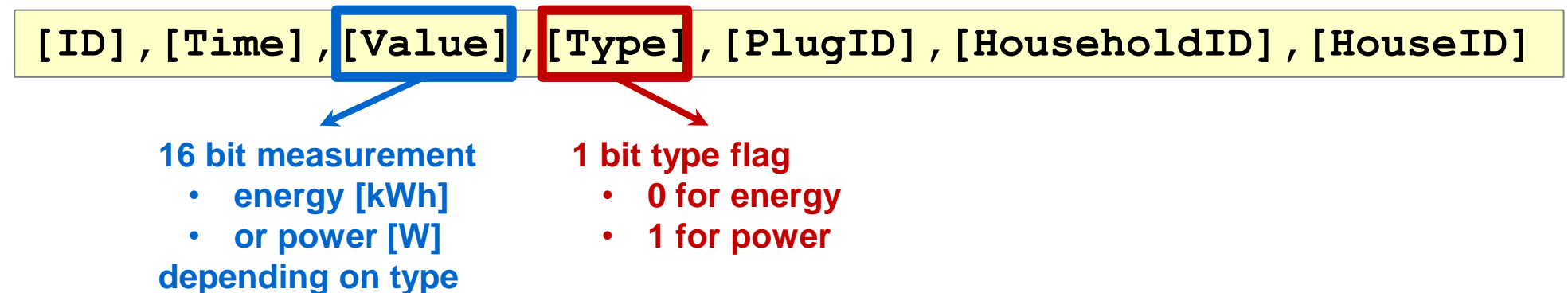
1. On host sub1, collect power (type 1) values > 500 W (on port 50001)
2. On host sub2, collect power (type 1) values > 100 W (on port 50002)
3. On host sub3, collect energy (type 0) values ≤ 30 kWh (on port 50003)
4. On host sub4, collect energy (type 0) values > 136 kWh (on port 50004)



Task 4.1 – Pub-Sub Routing

We will use real data from the "DEBS Grand Challenge 2014"

- Measurement format (UDP **payload string**):



- Write a script `~/ex4/task41.sh` which install flows for ...
 - ... ARP broadcast and IP forwarding in the 10.0.0.0/8 network
 - ... forwarding of measurements (matched by the multicast IP 230.0.0.0/8) to all subscribers
- Compare the size of the unfiltered vs. filtered output



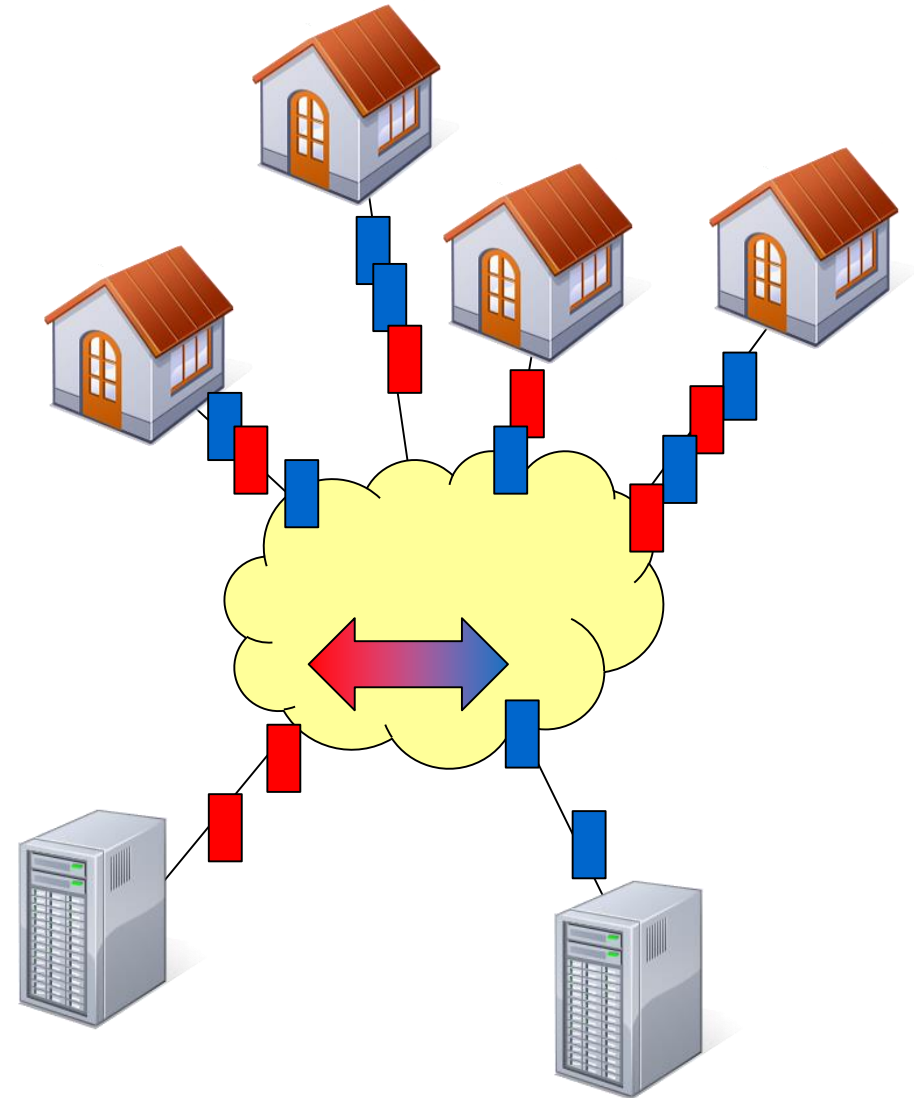
Overview

- **Task 4**
 - Goals of this task
 - 4.1 – Pub-Sub Routing
 - **4.2 – Content-based Routing**
 - 4.3 – REST Interface for Content-based Routing
- Deadline and Submission



Task 4.2 – Content-based Routing

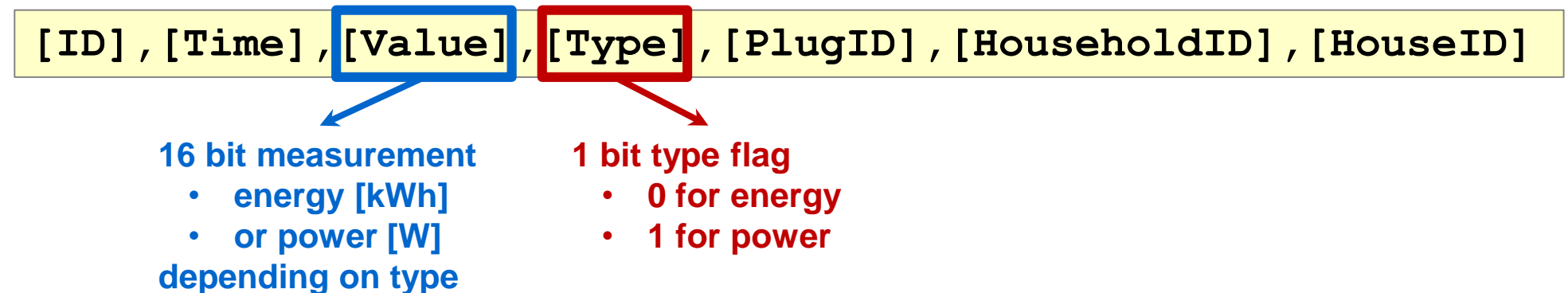
- Better:
 - (Pre-) filtering already in network
 - Reduce bandwidth in network and processing overhead at subscribers
 - **But:** measurements must be filtered *by content!*
- Solution: encode (interesting parts of) content into IP address
- Express filters as OF matching rules



Task 4.2 – Content-based Routing

We will use real data from the "DEBS Grand Challenge 2014"

- Measurement format (UDP payload string):



We encode the type (<T>) and value (<V>) fields of the payload into the destination IP address: 230.<T>.<V-MSB>.<V-LSB>

- Example: Payload: 3496,1377986403,285,1,0,6,0
Dst-IP: 230.1.1.29



Task 4.2 – Content-based Routing

- Make a copy of your script from Task 4.1 (`~/ex4/task42.sh`) and add appropriate flow entries for content-based routing
- Implement content-based routing by forwarding only datagrams with values greater/less than the next lower/larger power of two
- On the data plane, comparisons have to be emulated using prefix matching. Simple example:
 - `type=1 and value ≤ 3` \longleftrightarrow `"eth_type=0x0800,ipv4_dst=230.1.0.0/30"`
(i.e. the network prefix [230], the type [1] and the 14 most significant bits of the value [0] are fixed, while the 2 least significant bits are wildcarded)
- Execute your modified script to install all necessary flow entries
- Exemplarily implement the rules from Slide 9



Task 4.2 – Content-based Routing

Important hints:

- Goal prioritization of the Content-based Routing approach
 1. Minimize total network traffic
 - Eliminate all unnecessary traffic (with a value filter granularity of power of twos)
 2. Reduce subscriber filtering effort
 3. *Minimize flow table consumption*
- Consider generic system properties (like scalability).



Overview

- **Task 4**
 - Goals of this task
 - 4.1 – Pub-Sub Routing
 - 4.2 – Content-based Routing
 - **4.3 – REST Interface for Content-based Routing**
- Deadline and Submission



Task 4.3 – REST Interface

- Create a package `net.sdnlab.ex4.task43` for a Floodlight Module providing the following REST interface:
 - **POST** `http://<controller>:8080/subscriptions/{name}/json` to add a new subscription with `{name}` (define necessary filtering and subscriber parameters in the HTTP payload!)
 - **DELETE** `http://<controller>:8080/subscriptions/{name}/json` to delete the subscription with `{name}` again
 - **GET** `http://<controller>:8080/subscriptions/json` to return a list of all subscriptions in JSON format
- You can find a module skeleton in `~/ex4/java/task43`

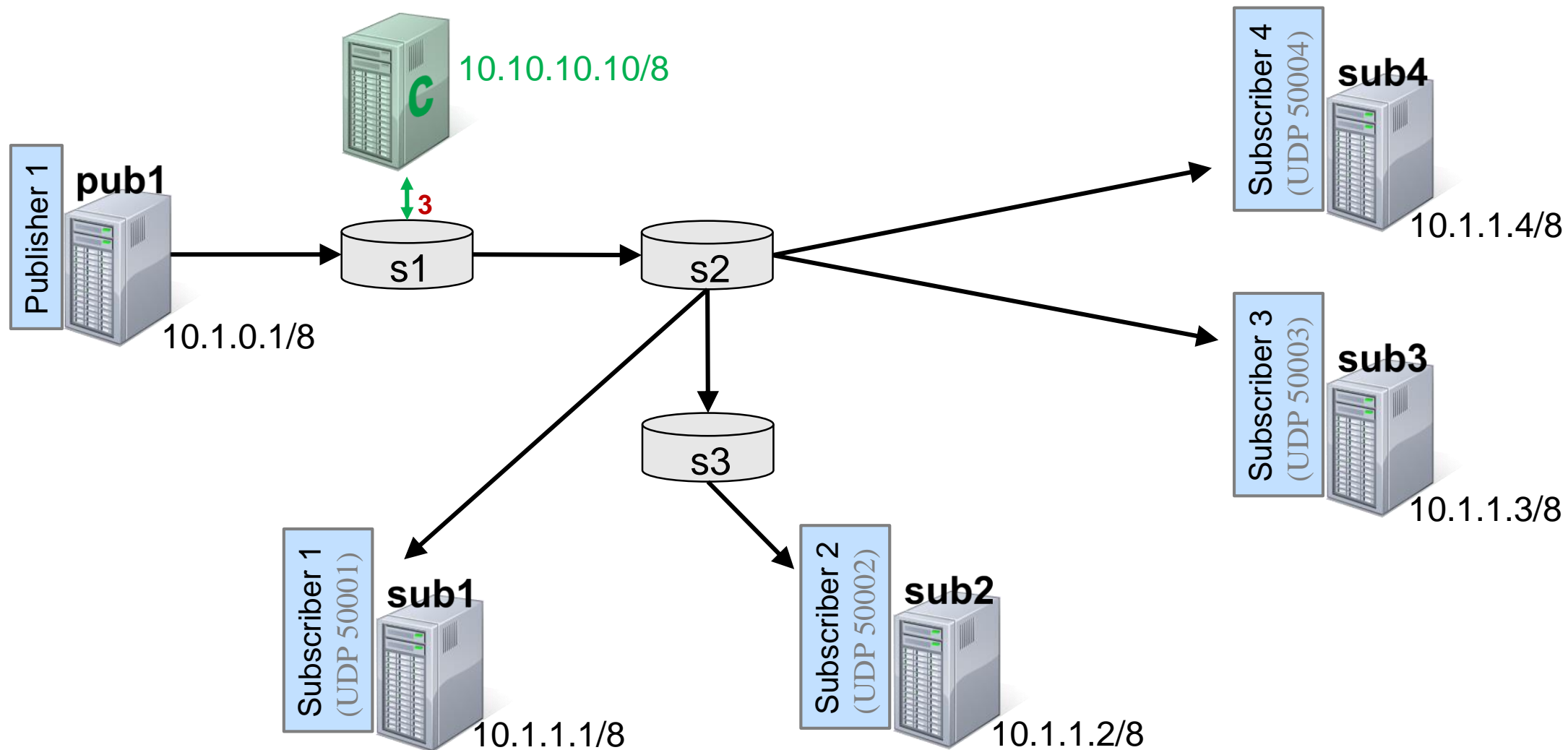


Task 4.3 – REST Interface

- Your module should
 - Implement the REST-interface described on the previous slide
 - Determine and install or delete the necessary flow entries upon handling incoming requests
- Also update your subscriber application to use this REST-interface to register subscriptions!
- Mininet hosts can reach the Floodlight controller's HTTP interface at <http://10.10.10.10:8080/...>
 - *Hint:* Make sure flows for communication with **c** are set up properly! (see next slide)

Task 4.3 – Pub-Sub Routing

Network Topology:



IPVS

Research Group

“Distributed Systems”

19

Universität Stuttgart

IPVS

Task 4.3 – REST Interface

- Module skeleton in `~/ex4/java/task43`
 - `ITask43Service.java` – IFloodlightService interface for your module
 - `Task43.java` – your module
 - `Task43WebRoutable.java` – HTTP router
 - `ListSubscriptionsResource.java` – resource for **GET** requests
 - `SubscriptionResource.java` – resource for **POST/DELETE** requests
- You may of course add classes, e.g. to represent a subscription
- Read the tutorial "[How to add a REST API to a Module](#)" to understand project architecture (cf. ILIAS Wiki)



Overview

- Task 4
- **Deadline and Submission**



Deadline and Submission

- When (submission deadline): January 26th 2021 at 08:00
- 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. Your scripts
 3. Archive of source package net.sdnlab.ex4
- Be prepared to show a live demo to the supervisor during the next meeting

