Lab 4

Group / Meter / Intent

助教: 劉承遠

sdnta@win.cs.nycu.edu.tw

Deadline: 2024/11/6

Outline

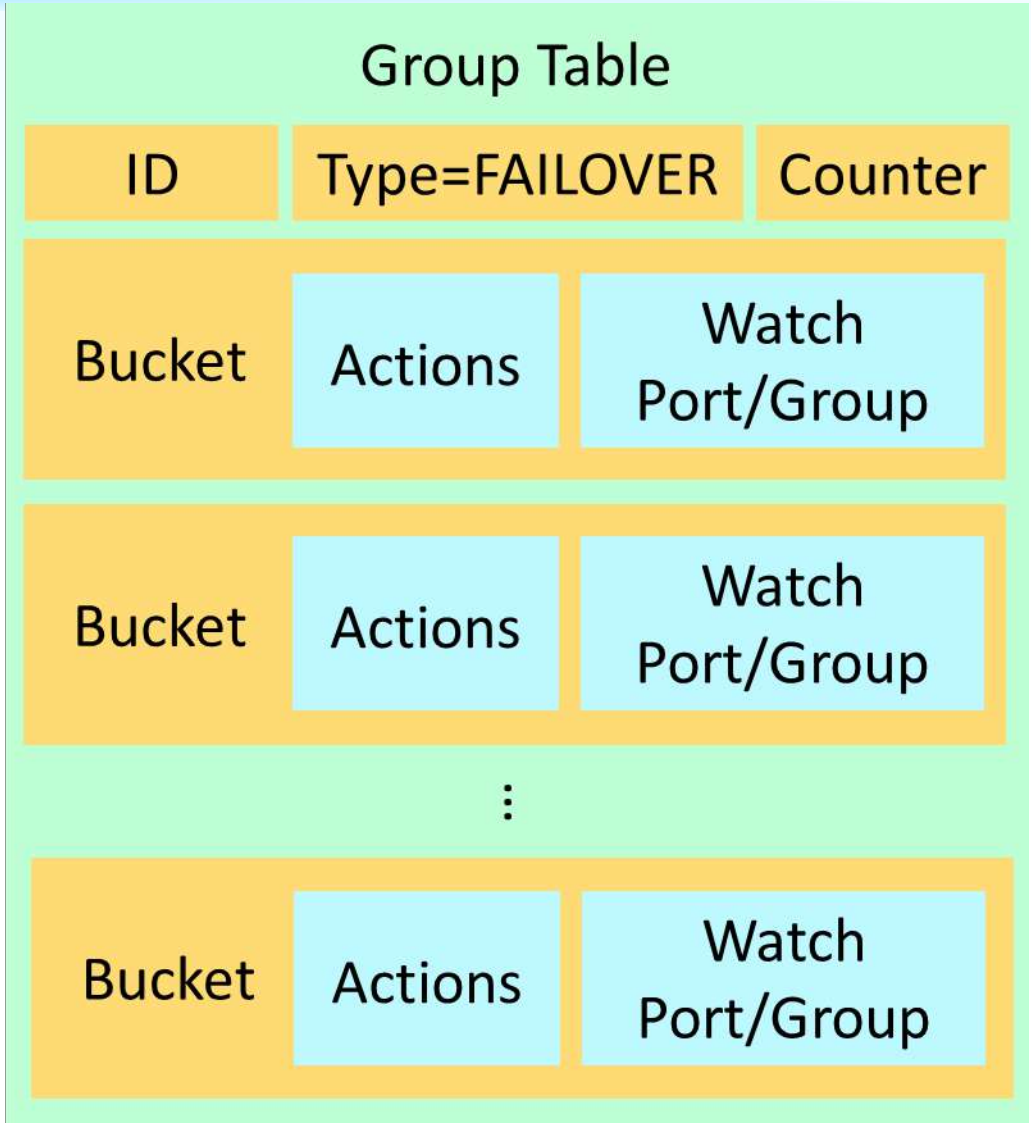
- Introduction to Group Table
  - Group Overview
  - Failover Group Workflow
- Introduction to Meter Table
- Introduction to Intent Service
- Introduction to Network Configuration Service
- Project 4 Overview
- Scoring Criteria, Submission, and Demo
- Reference

Group Overview

- Group provides more complex and specialized packet operations than flow table
  - Each group receives packets as input and performs actions on these packets
- Four group types:
  - All
  - Indirect
  - Select
  - Failover
- We only use **FAILOVER** in this project

Failover Group

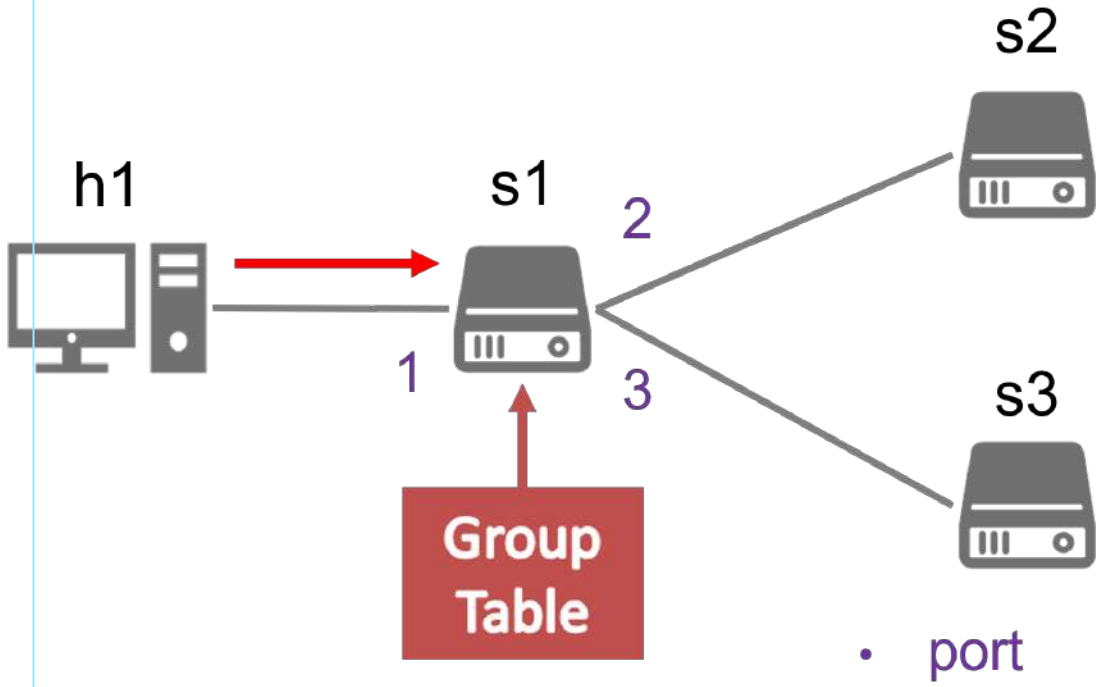
- Executes the first **LIVE** action bucket in the group
  - Use **WatchPort** or **WatchGroup (cascade)** for liveness detection
  - Only one bucket can be used at a time
- When the currently used bucket's Watch Port transition from up to down:
  - Select the next bucket in the bucket list with a Watch Port that is up



Failover Group Workflow (1/5)

1. h1 sends packets to s1
2. s1 selects bucket 1 and sends packets to s2
   - Since port 2 is up
3. Turn down s1 – s2 link
4. h1 sends packets to s1
5. s1 select bucket 2 and sends packets to s3
   - Since port 2 is down, port 3 is up

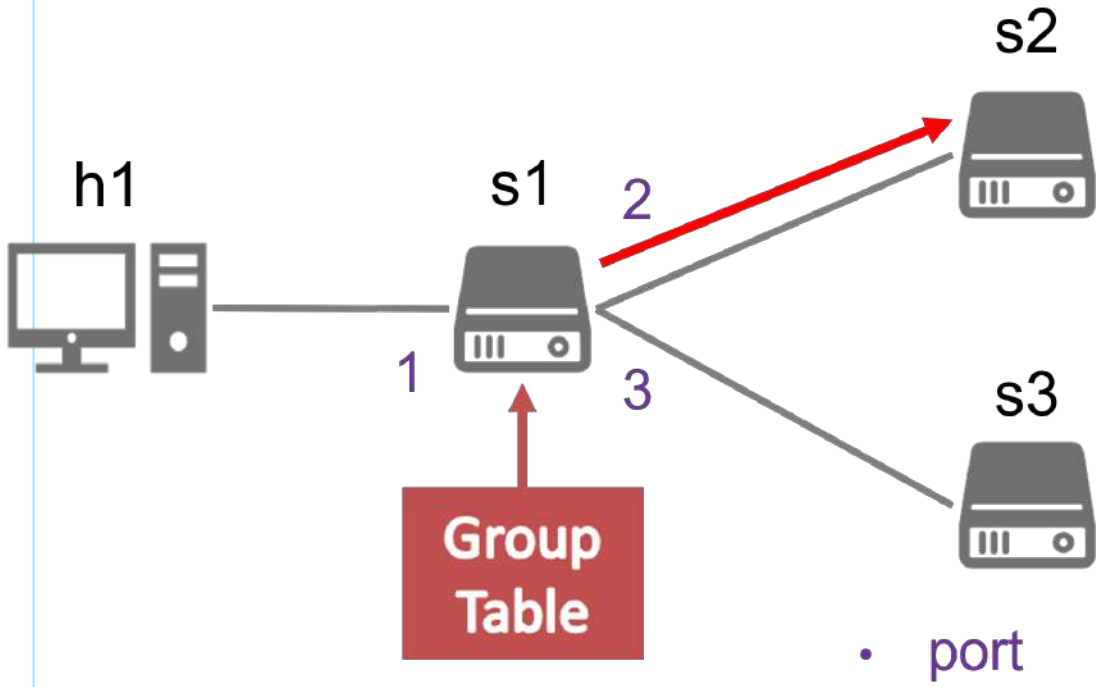
	Output Port	Watch port
Bucket 1	2	2
Bucket 2	3	3



Failover Group Workflow (2/5)

1. h1 sends packets to s1
2. s1 selects bucket 1 and sends packets to s2
   - Since port 2 is up
3. Turn down s1 – s2 link
4. h1 sends packets to s1
5. s1 select bucket 2 and sends packets to s3
   - Since port 2 is down, port 3 is up

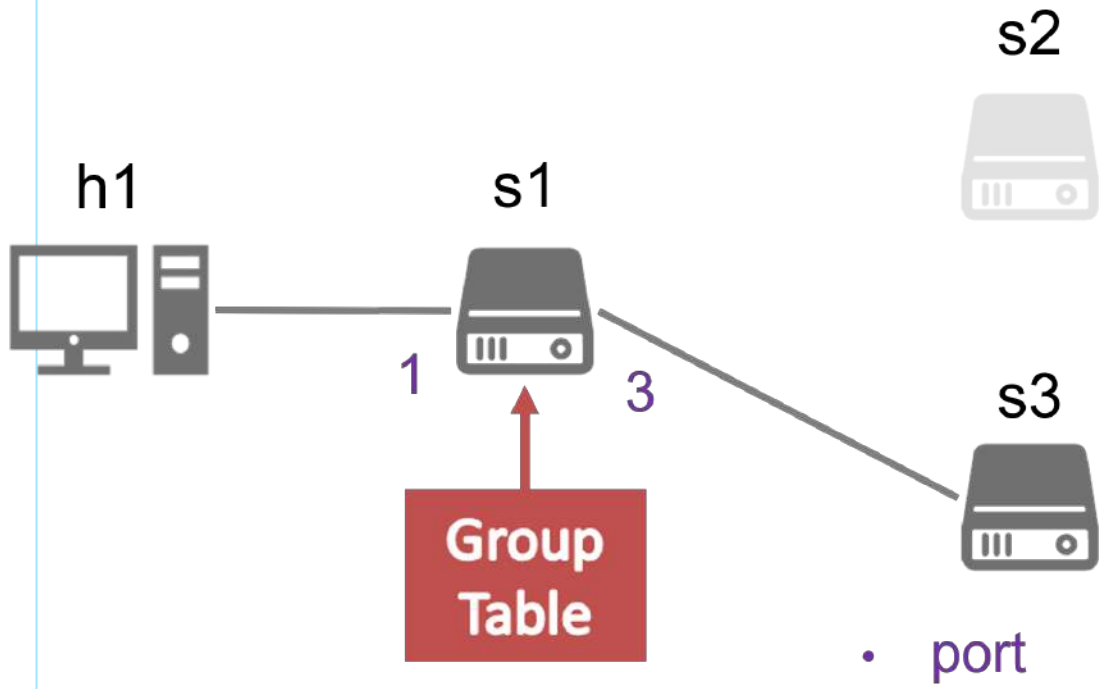
	Output Port	Watch port
Bucket 1	2	2
Bucket 2	3	3



Failover Group Workflow (3/5)

1. h1 sends packets to s1
2. s1 selects bucket 1 and sends packets to s2
   - Since port 2 is up
3. Turn down s1 – s2 link
4. h1 sends packets to s1
5. s1 select bucket 2 and sends packets to s3
   - Since port 2 is down, port 3 is up

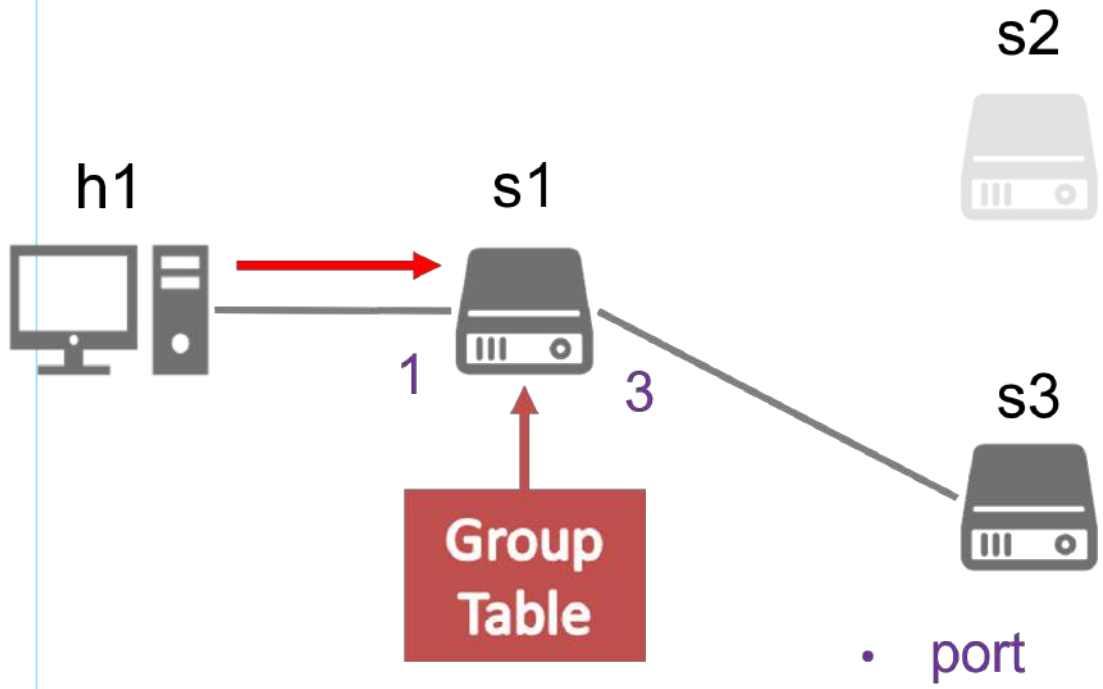
	Output Port	Watch port
Bucket 1	2	2
Bucket 2	3	3



Failover Group Workflow (4/5)

1. h1 sends packets to s1
2. s1 selects bucket 1 and sends packets to s2
   - Since port 2 is up
3. Turn down s1 – s2 link
4. h1 sends packets to s1
5. s1 select bucket 2 and sends packets to s3
   - Since port 2 is down, port 3 is up

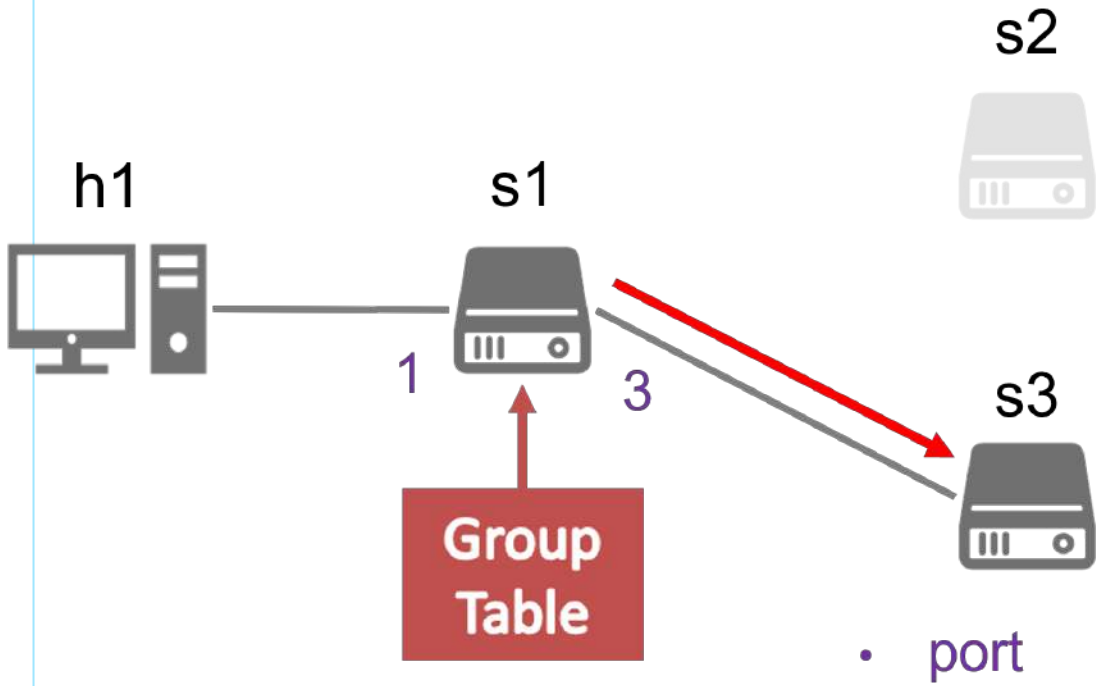
	Output Port	Watch port
Bucket 1	2	2
Bucket 2	3	3



Failover Group Workflow (5/5)

1. h1 sends packets to s1
2. s1 selects bucket 1 and sends packets to s2
   - Since port 2 is up
3. Turn down s1 – s2 link
4. h1 sends packets to s1
5. s1 select bucket 2 and sends packets to s3
   - Since port 2 is down, port 3 is up

	Output Port	Watch port
Bucket 1	2	2
Bucket 2	3	3



Outline

- Introduction to Group Table
- Introduction to Meter Table
  - Meter Overview
  - Drop Meter Workflow
- Introduction to Intent Service
- Introduction to Network Configuration Service
- Project 4 Overview
- Scoring Criteria, Submission, and Demo
- Reference

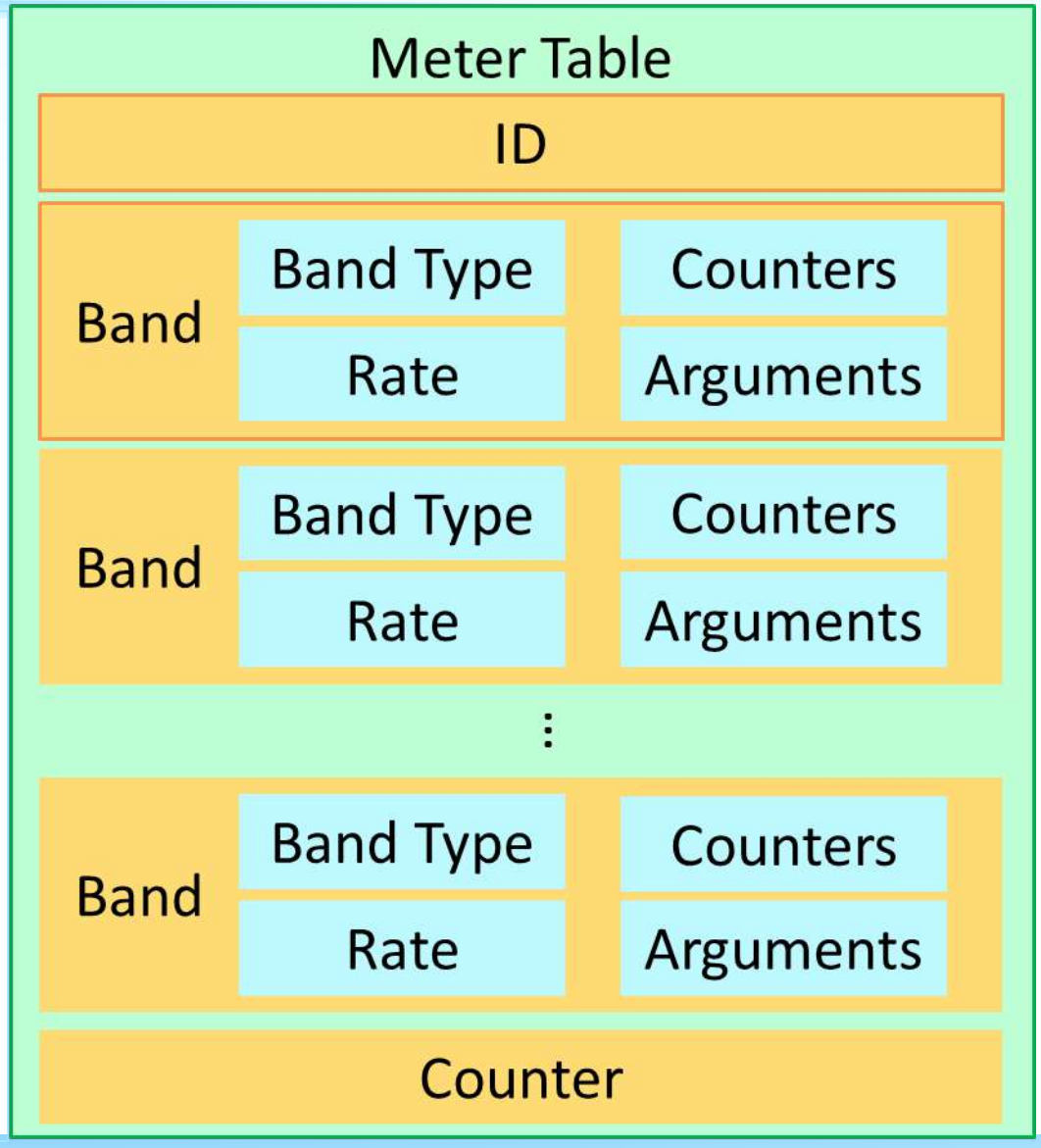
Meter Overview

- Meter provides a mechanism to control and manage traffic flow
  - Measure traffic flow rate and compare it against predefined thresholds
  - When measured rate exceeds a threshold
    - Dropping packets, changing packet priority, re-routing, …
  - Useful when network resources are limited, or certain traffic classes need to be prioritized
- Two meter types:
  - DROP
  - DSCP Remark (for QoS enforcement)
- We only use **DROP** in this project

DSCP: Differentiated Services Code Point

Meter Band

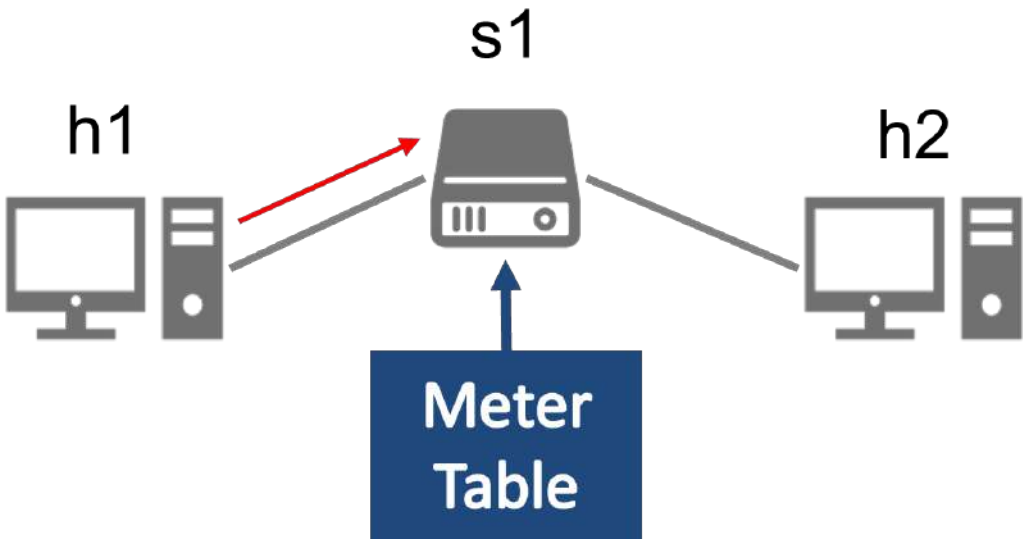
- Band represents how to handle packets after reaching a target lowest rate (threshold)
- Each packet is only processed by a single meter band, which
  - Meter Measured Rate > Band Rate
  - Band rate is the largest



Example Meter Workflow (1/3)

1. h1 sends packets to h2
2. s1 monitors traffic
   - rate under 100 KBps: pass through
   - rate over 100 KBps: drop packets

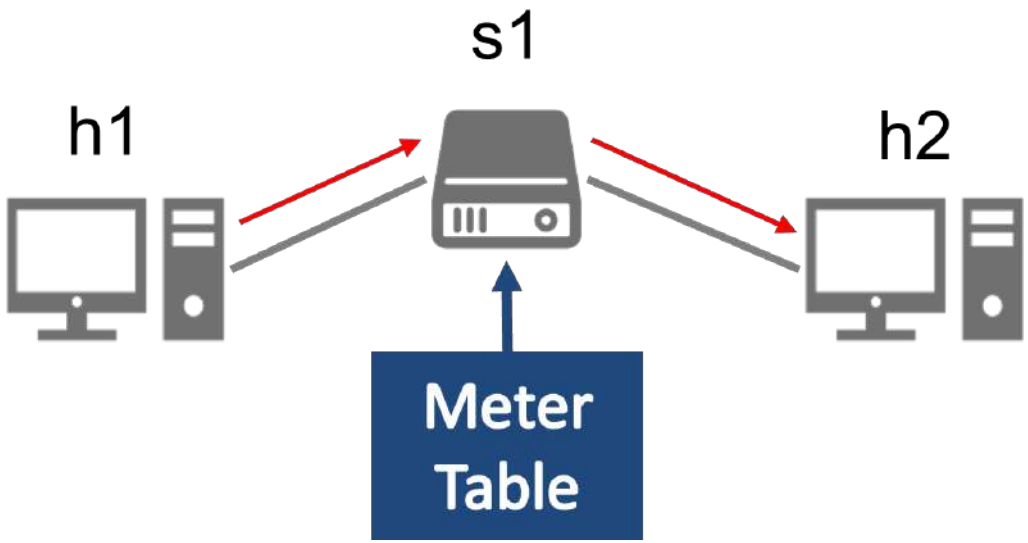
Rate	Type
0 KBps	NONE
100 KBps	DROP



Example Meter Workflow (2/3)

1. h1 sends packets to h2
2. s1 monitors traffic
   - rate under 100 KBps: pass through
   - rate over 100 KBps: drop packets

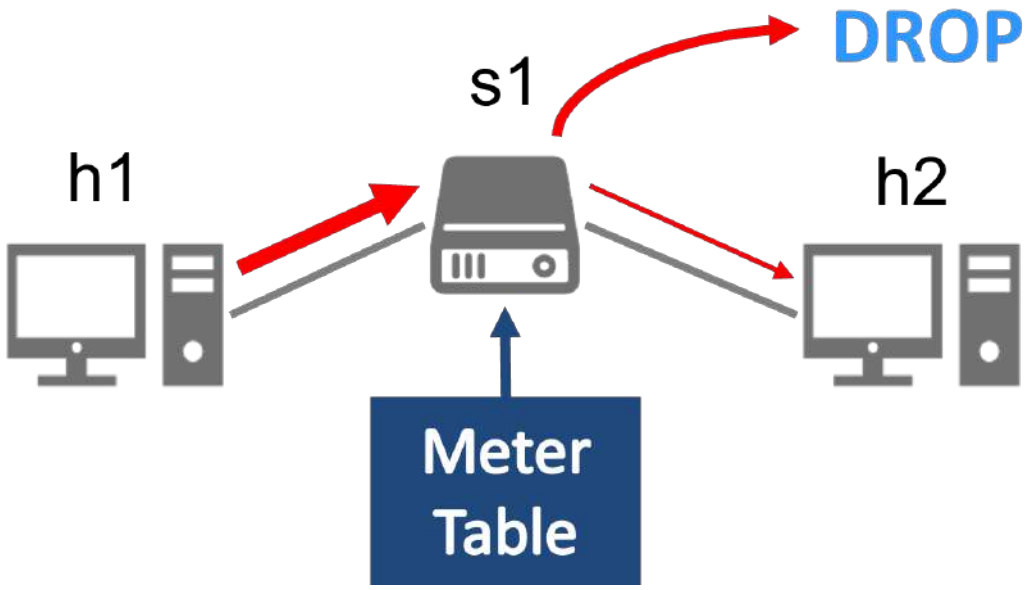
Rate	Type
0 KBps	NONE
100 KBps	DROP



Example Meter Workflow (3/3)

1. h1 sends packets to h2
2. s1 monitors traffic
   - rate under 100 KBps: pass through
   - rate over 100 KBps: drop packets

Rate	Type
0 KBps	NONE
100 KBps	DROP

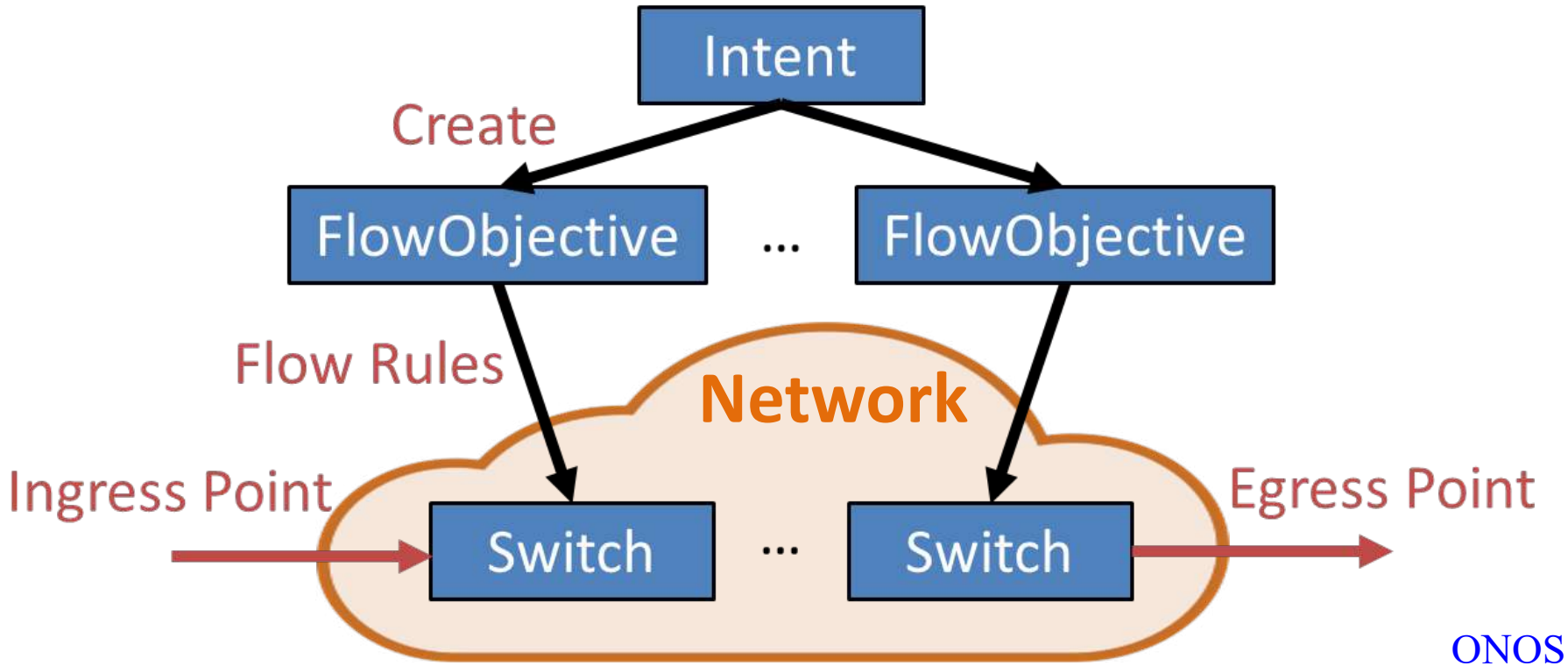


Outline

- Introduction to Group Table
- Introduction to Meter Table
- Introduction to Intent Service
- Introduction to Network Configuration Service
- Project 4 Overview
- Scoring Criteria, Submission, and Demo
- Reference

Introduction to Intent Service (1/2)

- Provide a high-level, network-centric abstraction
  - Focuses on **what** should be done
  - Rather than **how** it is specifically programmed



ONOS Reference

Introduction to Intent Service (2/2)

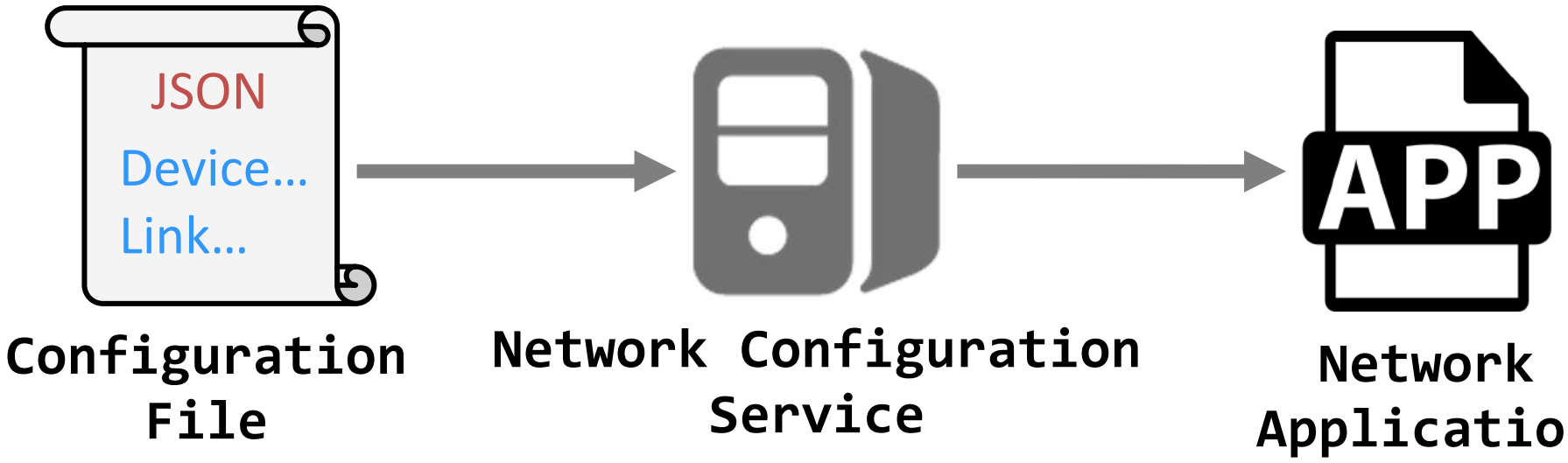
- For each intent, we need to define
  - **Ingress point:** ConnectPoint where packets enter the SDN network
    - One: Single-point to single-point intents
    - Multiple: Multi-point to single-point intents
  - **Egress point:** ConnectPoint where packets leave the SDN network
    - One: Single-point to single-point intents
    - Multiple: Single-point to multi-point intents
  - **Traffic selector:** Define what kind of packet this intent processes
  - **Traffic Treatment:** Define how to modify the packet
  - **Priority:** Priority for every flow rule this intent creates

Outline

- Introduction to Group Table
- Introduction to Meter Table
- Introduction to Intent Service
- Introduction to Network Configuration Service
  - Network Configuration Service Overview
  - Example ONOS Application
- Project 4 Overview
- Scoring Criteria, Submission, and Demo
- Reference

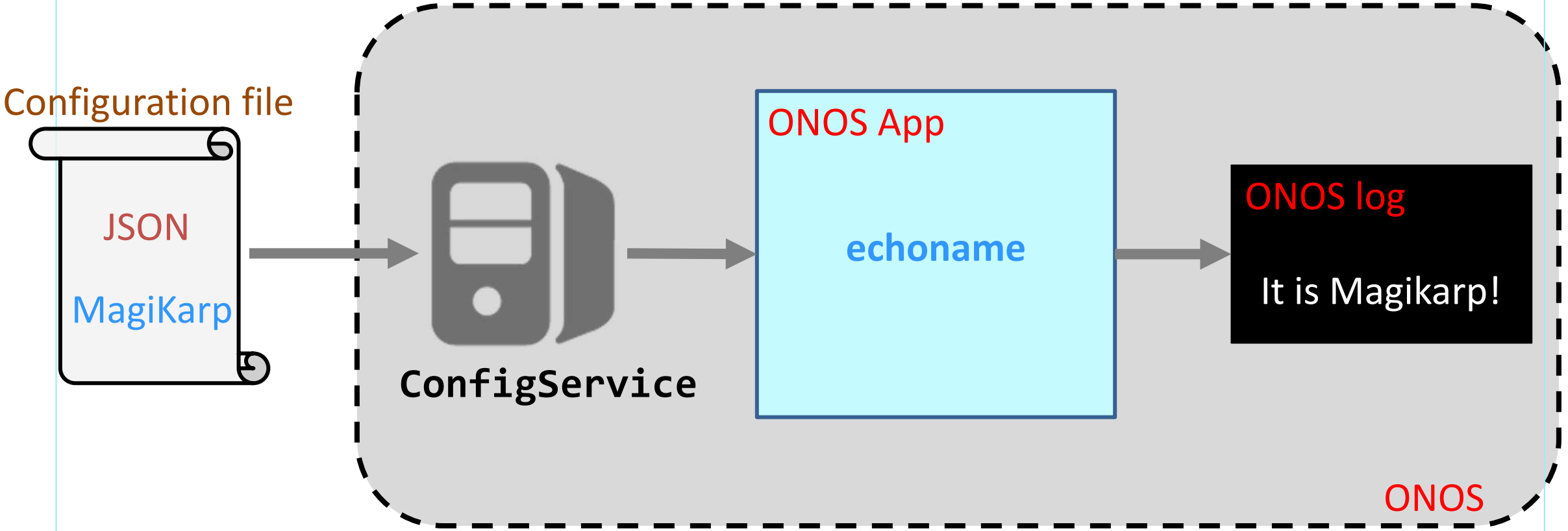
ONOS Network Configuration Service

- Purpose
  - Configure ONOS Apps that provide network services
  - Add information about devices, links configuration into ONOS's network view
- Functionality
  - Provide an extendable configuration database
  - Provide a restful API endpoint for configuration upload



Example Application – echoname

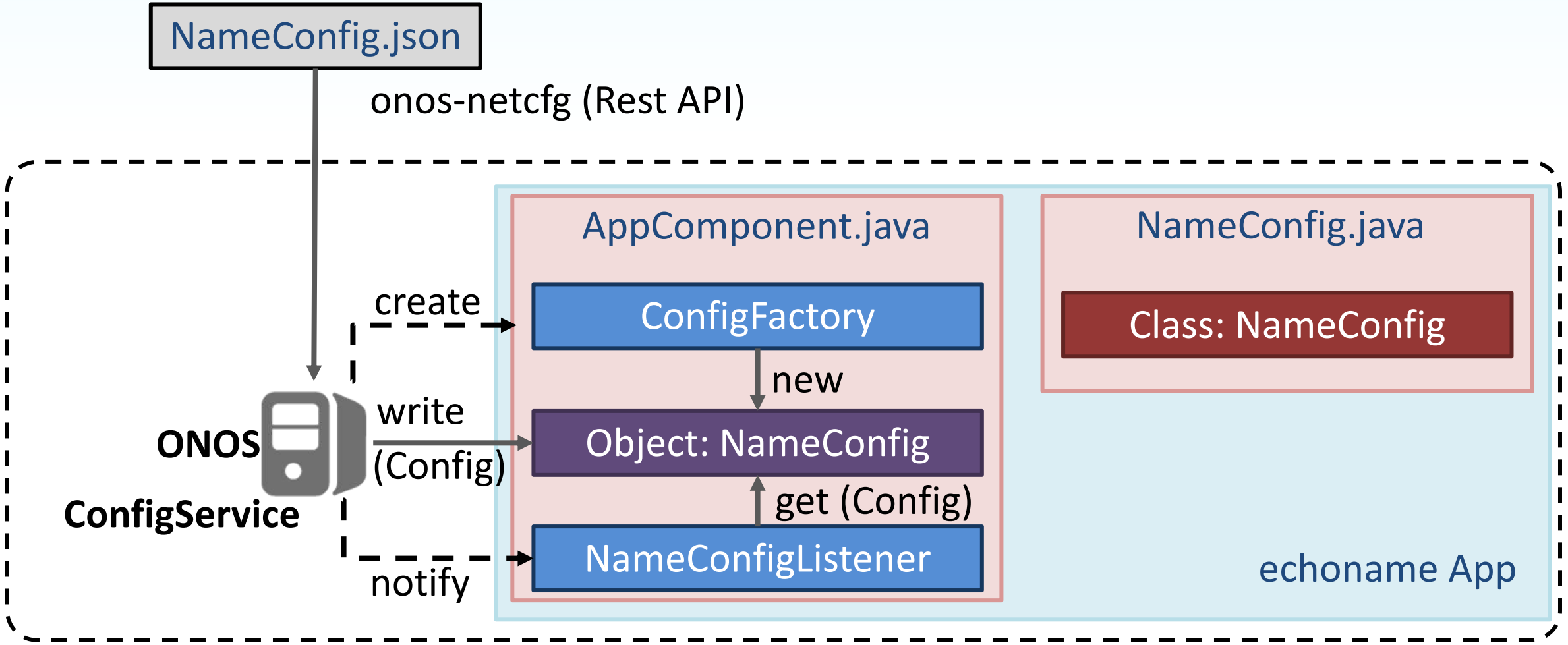
- **echoname** App
  - Utilize ONOS **Configuration Service** to receive a configuration specifying a **name**
  - Retrieve and print out the **name** specified in the configuration file



Example to Illustrate Network Configuration Service

- **echoname**: an application used to illustrate ONOS Network Configuration Service
- Components of **echoname**
  - NameConfig.json
    - Name configuration file (for **echoname**)
    - Provides **name** value for **echoname App** to print out (echo)
  - NameConfig.java
    - Validate and retrieve **name value** from **NameConfig.json**
  - AppComponent.java (main program of echoname App)
    - Listens to configuration file uploaded event
    - Prints value of name specified in the configuration file
    - Instantiates a **NameConfig** object

echoname APP and Configuration Uploading



NameConfig.java and NameConfig.json

- Define NameConfig Class
- Provide function to validate **NameConfig.json**
  - Check presence of “name” field
- Provide function to retrieve “name” from **NameConfig.json**

```java
public class NameConfig extends Config<ApplicationId> {
  public static final String NAME = "name";

  @Override
  public boolean isValid() {
    return hasOnlyFields(NAME);
  }

  public String name() {
    return get(NAME, null);
  }
}
```

Define **NameConfig** Class

NameConfig.java

```json
{
  "apps": {
    "nycu.winlab.echoname": {
      "whoami": {
        "name": "Magikarp"
      }
    }
  }
}
```

ApplicationId

NameConfig.json

Config

AppComponent.java Overview

- **ConfigFactory** Instantiation
- **NameConfigListener** Class Implementation
- **NameConfigListener** object Instantiation
- **NameConfigListener** and **ConfigFactory** Registration

```java
@Component(immediate = true)
public class AppComponent {
  private final Logger log = LoggerFactory.getLogger(getClass());
  private final NameConfigListener cfgListener = new NameConfigListener();

  private final ConfigFactory<ApplicationId, NameConfig> factory = new ConfigFactory<ApplicationId, NameConfig>(
      APP_SUBJECT_FACTORY, NameConfig.class, "whoami") {
    @Override
    public NameConfig createConfig() {
      return new NameConfig();
    }
  };

  private ApplicationId appId;

  @Reference(cardinality = ReferenceCardinality.MANDATORY)
  protected NetworkConfigRegistry cfgService;

  @Reference(cardinality = ReferenceCardinality.MANDATORY)
  protected CoreService coreService;

  @Activate
  protected void activate() {
    appId = coreService.registerApplication("nycu.winlab.echoname");
    cfgService.addListener(cfgListener);
    cfgService.registerConfigFactory(factory);
    log.info("Started");
  }

  @Deactivate
  protected void deactivate() {
    cfgService.removeListener(cfgListener);
    cfgService.unregisterConfigFactory(factory);
    log.info("Stopped");
  }

  private class NameConfigListener implements NetworkConfigListener {
    @Override
    public void event(NetworkConfigEvent event) {
      if ((event.type() == CONFIG_ADDED || event.type() == CONFIG_UPDATED)
          && event.configClass().equals(NameConfig.class)) {
        NameConfig config = cfgService.getConfig(appId, NameConfig.class);
        if (config != null) {
          log.info("It is {}!", config.name());
        }
      }
    }
  }
}
```

2. Instantiate **NameConfigListener** object

1. Instantiate **ConfigFactory** object (for creating **NameConfig** object)

3. Register **NameConfigListener** and **ConfigFactory** with Configuration Service

2. Implement **NameConfigListener**

ConfigFactory Instantiation

- Instantiate a **ConfigFactory** object
  - For creating **NameConfig** object

```
{
  "apps": {
    "nycu.winlab.echoname": {
      "whoami": {
        "name": "Magikarp"
      }
    }
  }
}
```

NameConfig.json

```
private final ConfigFactory<ApplicationId, NameConfig> factory = new ConfigFactory<ApplicationId, NameConfig>(
    APP_SUBJECT_FACTORY, NameConfig.class, "whoami") {
  @Override
  public NameConfig createConfig() {
    return new NameConfig();
  }
};
```

ConfigFactory instantiation

NameConfig object creation

AppComponent.java

ConfigFactory

NameConfigListener Class Implementation and Instantiation

- Implement **NameConfigListener** Class

```
private class NameConfigListener implements NetworkConfigListener {
  @Override
  public void event(NetworkConfigEvent event) {
    if ((event.type() == CONFIG_ADDED || event.type() == CONFIG_UPDATED)
        && event.configClass().equals(NameConfig.class)) {
      NameConfig config = cfgService.getConfig(appId, NameConfig.class);
      if (config != null) {
        log.info("It is {}!", config.name());
      }
    }
  }
}
```

Listen to network configuration event

AppComponent.java

- Instantiate the **NameConfigListener** object

```
private final NameConfigListener cfgListener = new NameConfigListener();
```

Instantiate **NameConfigListener**

AppComponent.java

NameConfigListener and ConfigFactory Registration

- Register **NameConfigListener** and **ConfigFactory** with ONOS Configuration Service

```
@Activate
protected void activate() {
  appId = coreService.registerApplication("nycu.winlab.echoname");
  cfgService.addListener(cfgListener);
  cfgService.registerConfigFactory(factory);
  log.info("Started");
}
```

Register with ONOS Configuration Service

AppComponent.java

cfgService: an object of NetworkConfigRegistry class

echoname Demonstration

1. Build, install, and activate **echoname** App
2. Upload **NameConfig.json**

```
bash$ onos-netcfg localhost NameConfig.json
```

3. Observe ONOS log

```
{
  "apps": {
    "nycu.winlab.echoname": {
      "whoami": {
        "name": "Magikarp"
      }
    }
  }
}
```

NameConfig.json

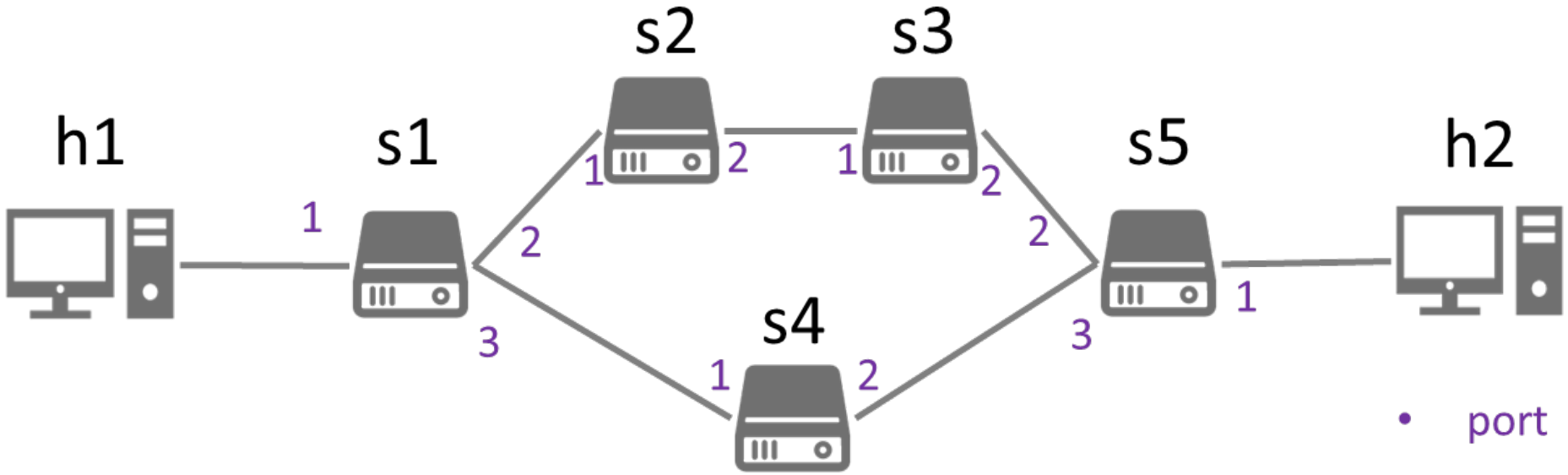
```
18:07:46.678 INFO [ApplicationManager] Application nycu.winlab.echoname has been installed
18:07:46.681 INFO [FeaturesServiceImpl] Adding features: echoname/[1.0.0.SNAPSHOT,1.0.0.SNAPSHOT]
18:07:46.952 INFO [FeaturesServiceImpl] Changes to perform:
18:07:46.954 INFO [FeaturesServiceImpl]   Region: root
18:07:46.954 INFO [FeaturesServiceImpl]     Bundles to install:
18:07:46.954 INFO [FeaturesServiceImpl]       mvn:nycu.winlab/echoname/1.0-SNAPSHOT
18:07:46.955 INFO [FeaturesServiceImpl] Installing bundles:
18:07:46.955 INFO [FeaturesServiceImpl]   mvn:nycu.winlab/echoname/1.0-SNAPSHOT
18:07:46.961 INFO [FeaturesServiceImpl] Starting bundles:
18:07:46.963 INFO [FeaturesServiceImpl]   nycu.winlab.echoname/1.0.0.SNAPSHOT
18:07:46.967 INFO [AppComponent] Started
18:07:46.968 INFO [FeaturesServiceImpl] Done.
18:07:46.979 INFO [ApplicationManager] Application nycu.winlab.echoname has been activated
18:08:40.914 INFO [AppComponent] It is Magikarp!
```

Outline

- Introduction to Group Table
- Introduction to Meter Table
- Introduction to Intent Service
- Introduction to Network Configuration Service
- Project 4 Overview
  - Project Overview and Workflow
  - How to Test Your App
- Scoring Criteria, Submission, and Demo
- Reference

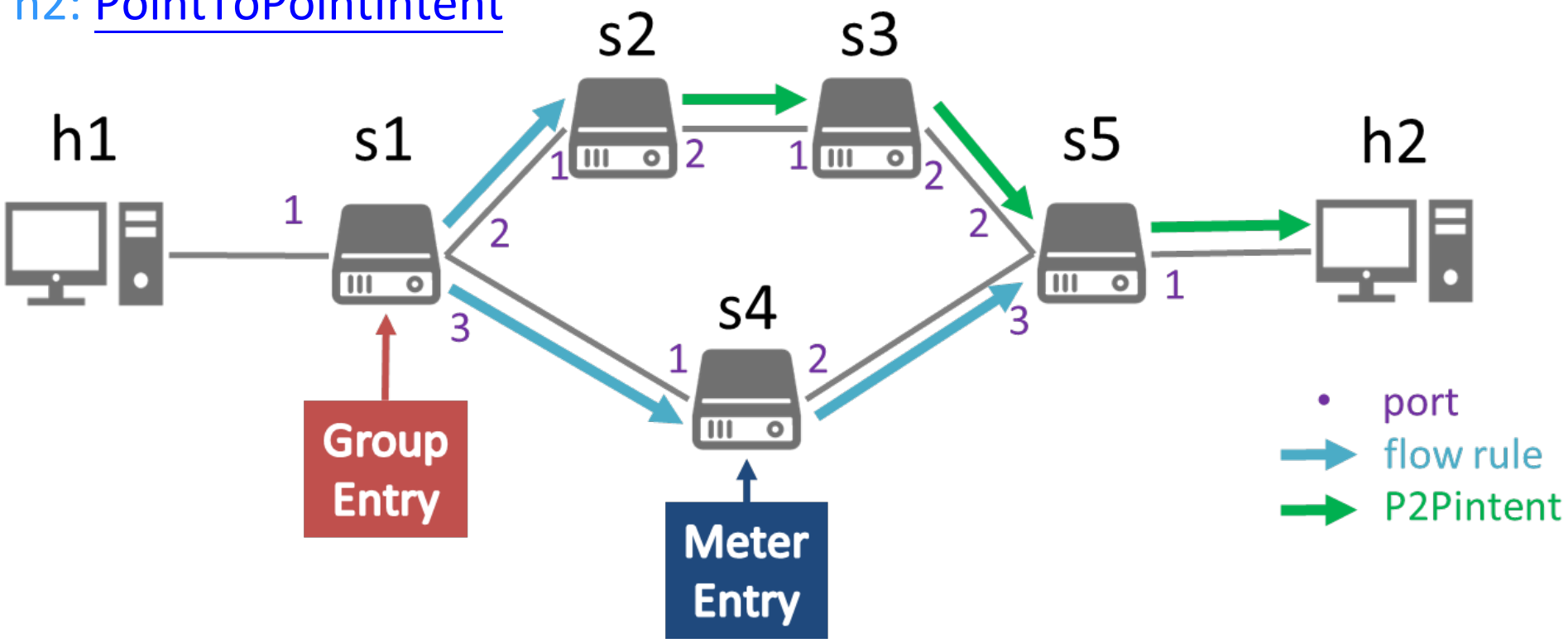
Project 4 Overview

- h1 (client) uses **iperf** to send UDP packets to h2 (server)
  - h1 to h2
    - **s1-s2 link up**: through **s2, s3**
    - **s1-s2 link down** : through **s4**, limit the **traffic rate**
  - h2 to h1
    - Use **IntentService** to set up flow rules



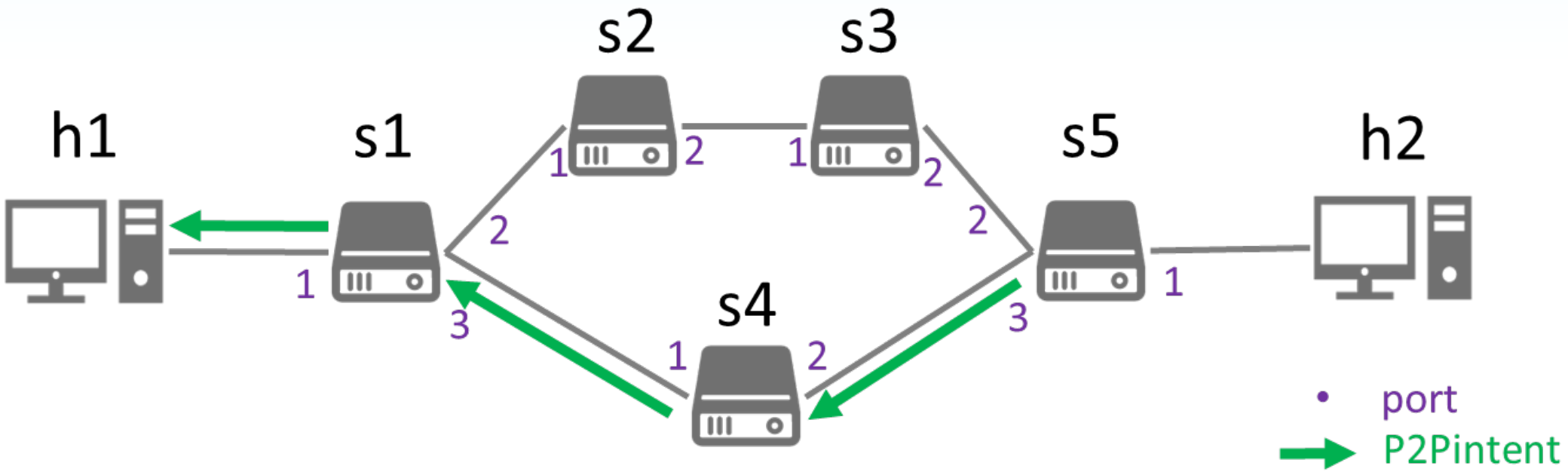
h1 to h2 Path Setup

- h1 to s2/s4: flow rule + Failover group table entry
- s2 to h2: PointToPointIntent
- s4 to s5: flow rule + meter table entry
- s5 to h2: PointToPointIntent



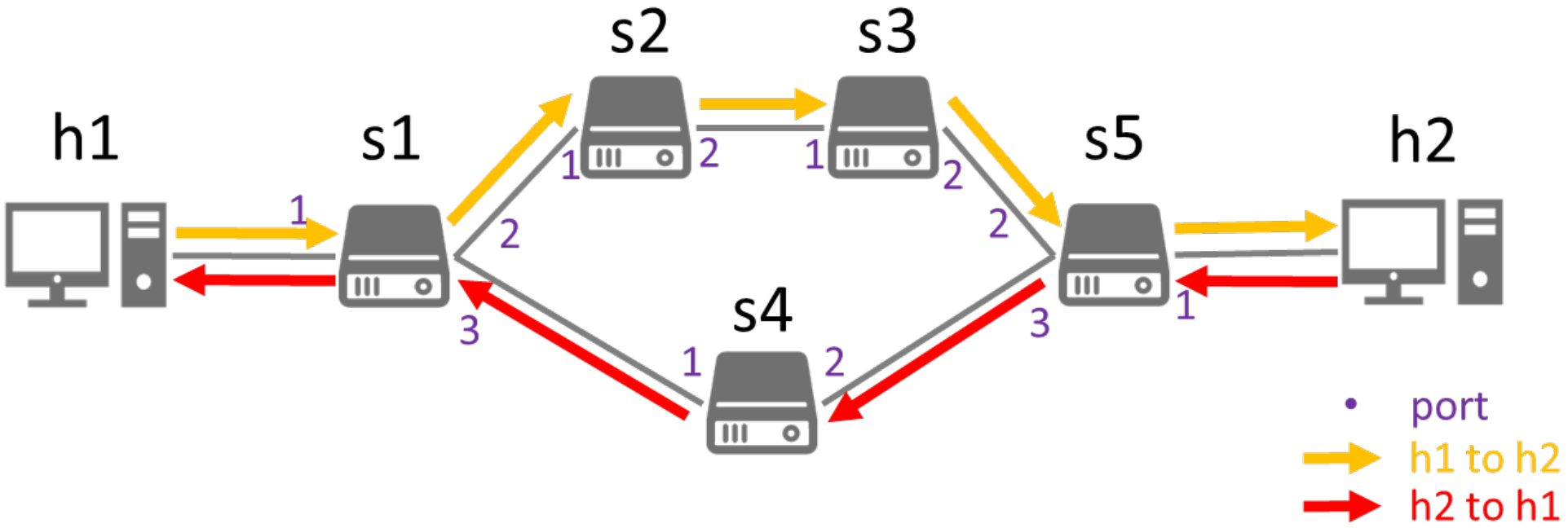
h2 to h1 Path Setup

- h2 to h1: PointToPointIntent



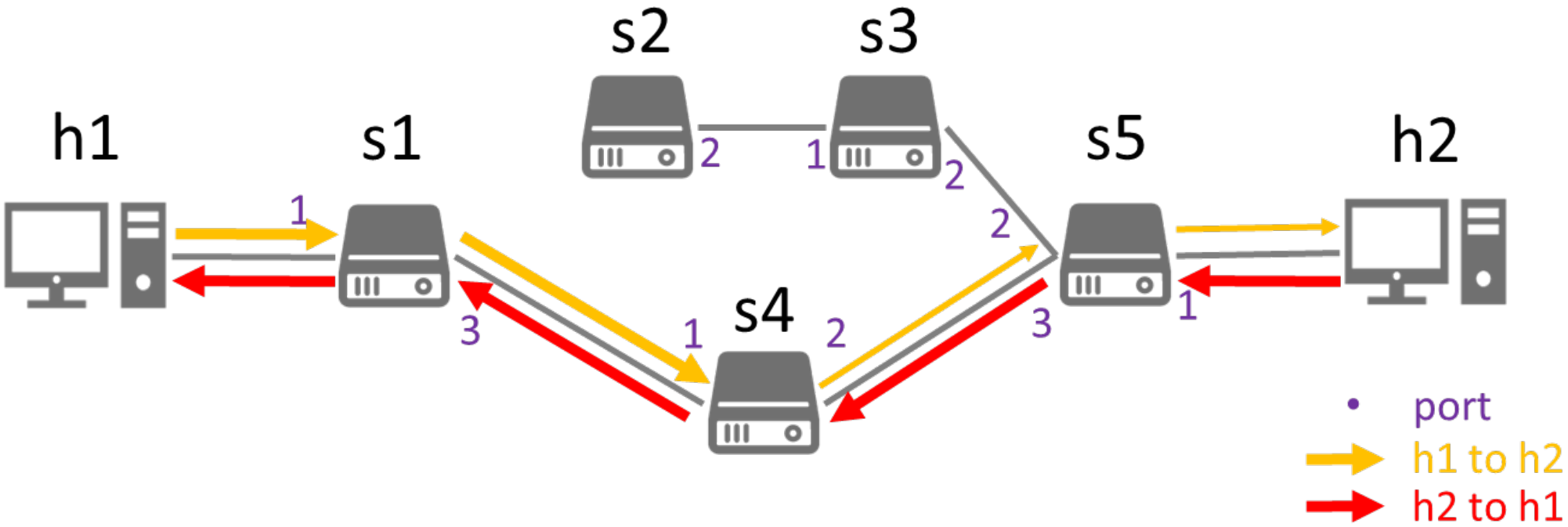
Workflow – s1-s2 Linkup

- Traffic from h1 to h2 through **s2**, **s3**
- Traffic from h2 to h1 determined by **IntentService**



Workflow – s1-s2 Linkdown

- Traffic from h1 to h2 through **s4** with **Meter**
- Traffic from h2 to h1 determined by **IntentService**



Host ConnectPoint and IP/Mac Configuration

- Prepare **hostconfig.json**, containing
  - **ConnectPoint**, **MacAddress**, and **IpAddress** of hosts
- Upload **hostconfig.json** to ONOS ConfigurationService via REST API

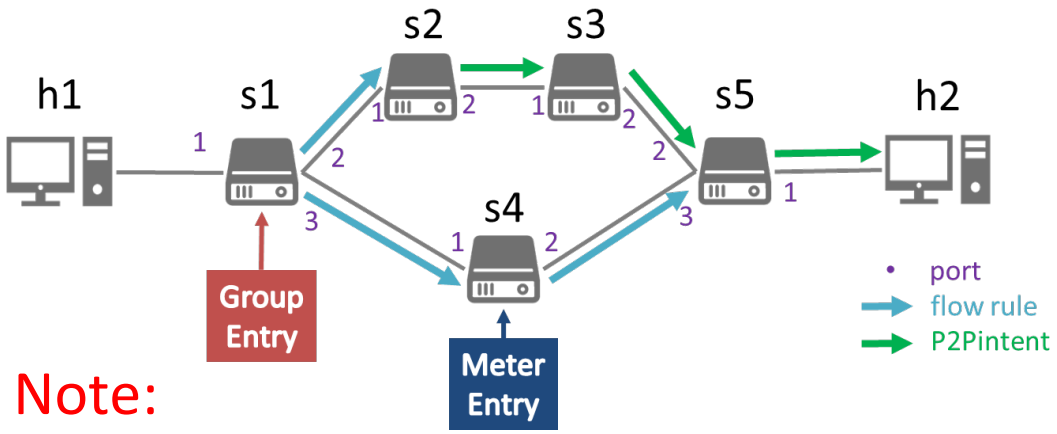
```
bash$ onos-netcfg localhost hostconfig.json
```

- Print configuration information with ONOS log after uploaded
  - Configuration information:
    - “ConnectPoint_h1: {h1 connect point}, ConnectPoint_h2: {h2 connect point}”
    - “MacAddress_h1: {h1 mac}, MacAddress _h2: {h2 mac}”
    - “IpAddress_h1: {h1 ip}, IpAddress_h2: {h2 ip}”

```
ConnectPoint_h1: of:0000000000000001/1, ConnectPoint_h2: of:0000000000000005/1
MacAddress_h1: 00:00:00:00:00:01, MacAddress_h2: 00:00:00:00:00:02
IpAddress_h1: 10.6.1.1, IpAddress_h2: 10.6.1.2
```

Flow Rule and Group Entry

- Flow Rule
  - Selector:
    - Input Port: 1
    - IPv4: 0x0080
  - Treatment
    - GROUP_ID

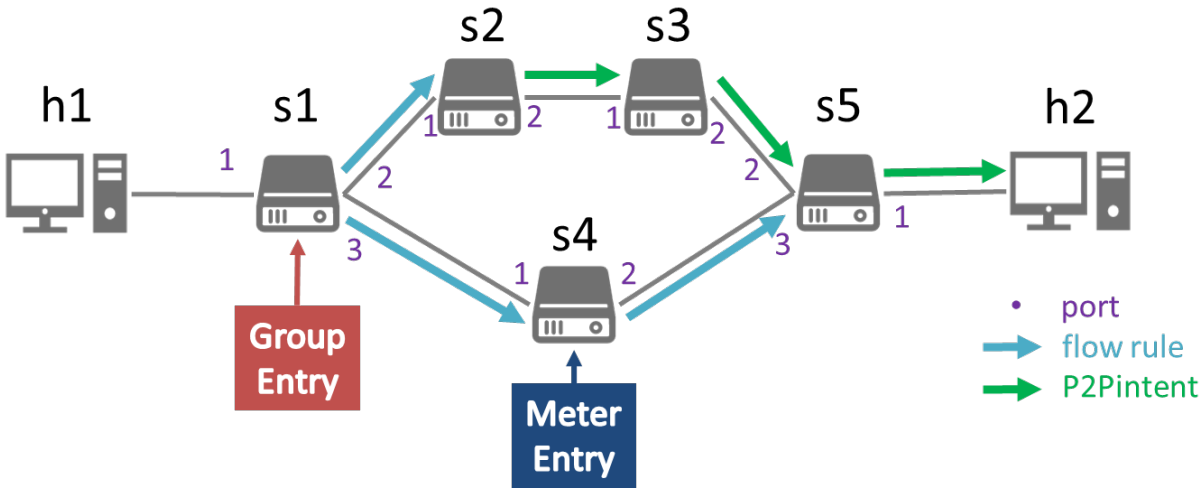


- Note:
  - Install Group entry
  - Find GROUP_ID and install flow rule

- Group Entry
  - Device: s1
  - Type: FAILOVER
  - Buckets:
    - Bucket 1:
      - TrafficTreatment: output, 2
      - WatchPort: 2
      - WatchGroup: any
    - Bucket 2:
      - TrafficTreatment: output, 3
      - WatchPort: 3
      - WatchGroup: any

Flow Rule and Meter Entry

- Flow Rule
  - Selector:
    - Source Mac: h1 Mac
  - Treatment
    - Output port: 2
    - METER_ID

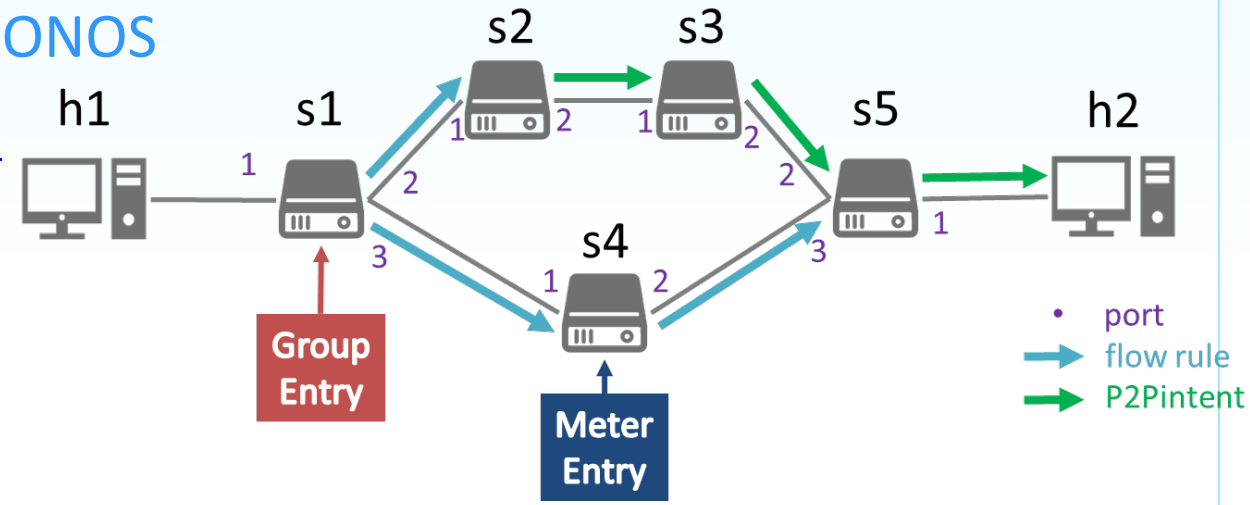


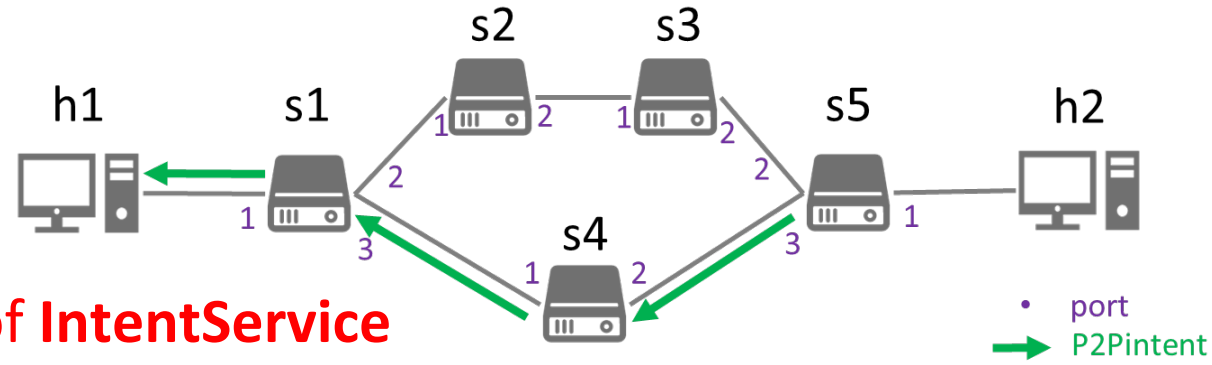
- Meter Entry
  - Device: s4
  - Burst: true
  - Unit: KB_PER_SEC
  - Band:
    - Type: DROP
    - BurstSize: 1024
    - withRate: 512

- Note:
  - Install Meter entry
  - Find METER_ID and install flow rule

Flow Rules Installation with IntentService

- Use IntentService to submit intents to ONOS
  - s2(s5) to h2: Use PointToPointIntent
    - Ingress: from packet-in packet
    - Egress: from **hostconfig.json**
    - TrafficSelector: h2 **Mac Address**
  - h2 to h1: Use PointToPointIntent
    - Ingress: from packet-in packet
    - Egress: from **hostconfig.json**
    - TrafficSelector: h1 **Mac Address**





- Print intent information in ONOS log of **IntentService**
  - Intent information:

    "Intent `{ingress device ID}`, port `{ingress port}` => `{egress device ID}`, port `{egress port}` is submitted."

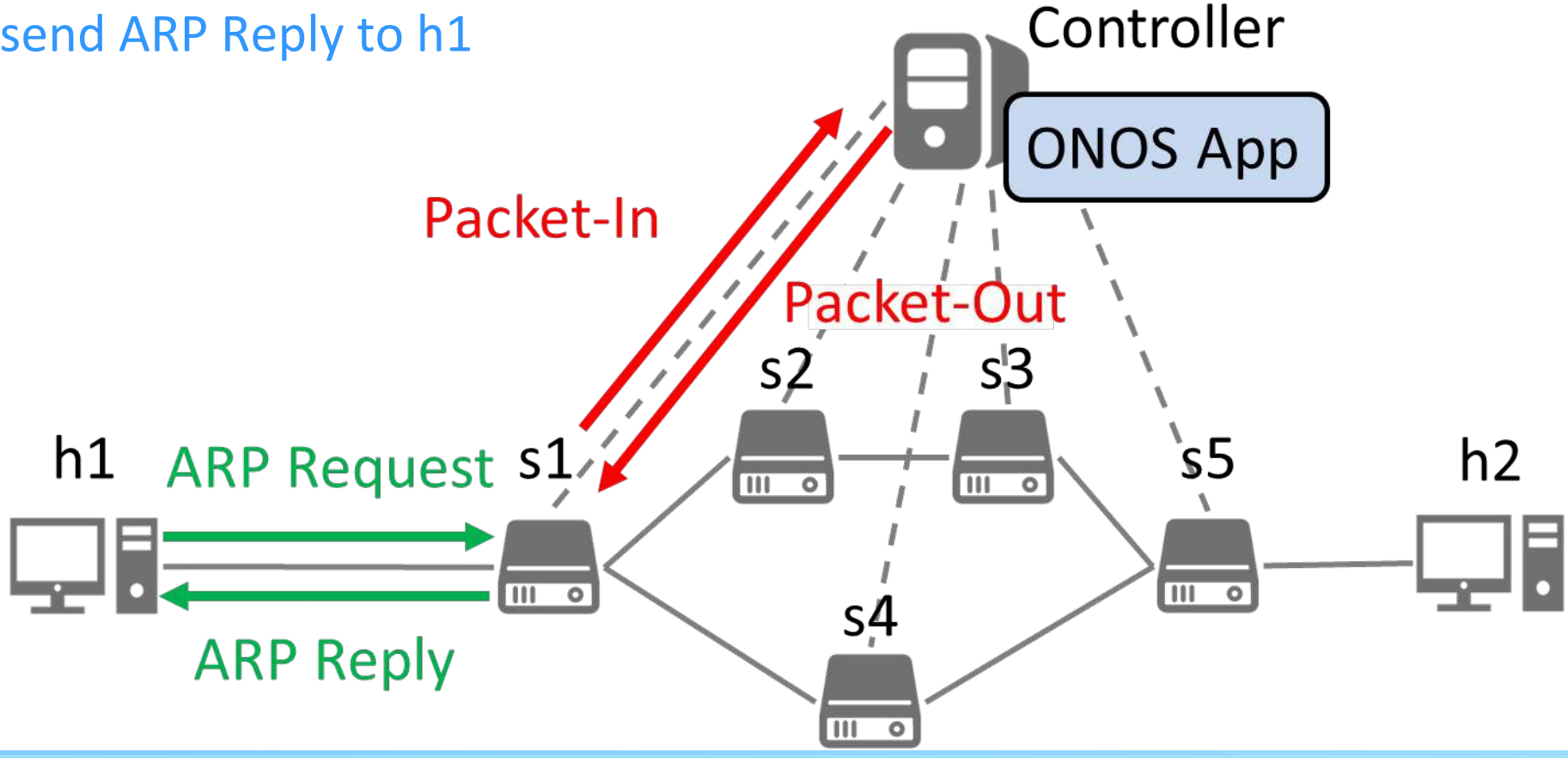
```
Intent `of:0000000000000002`, port `1` => `of:0000000000000005`, port `1` is submitted.
```

Proxy ARP

- Initially, h1 does not know h2's MacAddress
  - h1 sends ARP Request before sending first iperf packet
    - Destination MacAddress = **FF:FF:FF:FF:FF:FF (broadcast)**
- Need ProxyArp
  - Sends an ARP Reply packet with target MacAddress

Reply the first ARP Request

1. h1 send ARP Request to h2
2. s1 Packet-In to controller
3. Controller Packet-Out ARP Reply to s1 (with target MacAddress)
4. s1 send ARP Reply to h1



How to Test Your App (1/2)

1. Run **ring_topo.py** to build the topology

```
bash$ sudo mn --custom=ring_topo.py --topo=mytopo
--controller=remote,ip=127.0.0.1,port=6653
--switch=ovs,protocols=OpenFlow14
```

2. Upload config file to ONOS

```
bash$ onos-netcfg localhost hostconfig.json
```

3. Build, install, and activate your App
4. Use h1 as **iperf UDP** client and h2 as **iperf UDP** server to test your traffic
5. Monitor s1 and s4 interface
   - Check if traffic from h1 to h2 through s2 and s3
   - Check the path from h2 to h1 through which switch

```
Mininet> sh ovs-ofctl dump-ports –O OpenFlow14 s1(s4)
```

- Take screenshots and explain results

How to Test Your App (2/2)

6. Turn down s1–s2 link

```
Mininet> link s1 s2 down
```

7. Run **iperf UDP** on h1 to h2
8. Monitor s1 and s4 interface
   - Check if both traffic go through s4
   - Check if the iperf traffic rate is limited
   - Take screenshots and explain results

- Note:
  - The first few packets may drop when the topology change
    - Since ONOS takes some time to add flow rules on switches

Outline

- Introduction to Group Table
- Introduction to Meter Table
- Introduction to Intent Service
- Introduction to Network Configuration Service
- Project 4 Overview
- Scoring Criteria, Submission, and Demo
- Reference

Scoring Criteria

- **Score = Project (60%) + Demo (40%)**
- **(5%)** Project naming convention
  - <groupId>: nycu.winlab
  - <artifactId>: groupmeter
  - <version>: <use default> (1.0-SNAPSHOT)
  - <package>: nycu.winlab.groupmeter
- **(10%)** Acquire ConnectPoint, MacAddress, and IpAddress from the configuration file
  - You must use classes under org.onosproject.net.config
    - Otherwise, ZERO point is given in this part
  - Print the correct ConnectPoint, MacAddress, and IpAddress in ONOS log
    - Wrong format or no output is a 5-point deduction

Scoring Criteria

- **(15%)** Group and Meter entry installation
  - You must use GroupDescription, GroupService, MeterRequest, MeterService
  - Please set the correct parameters which provided in p.37, p.38
    - If any of the rules are violated, ZERO points are given in this part
- **(20%)** Intents installation
  - You must NOT use any class under **org.onosproject.net.flowobjective**
  - You must use PointToPointIntent and IntentService
    - If any of the rules are violated, ZERO points are given in this part
  - Print intent information in ONOS log
    - Wrong format or no output is a 10-point deduction

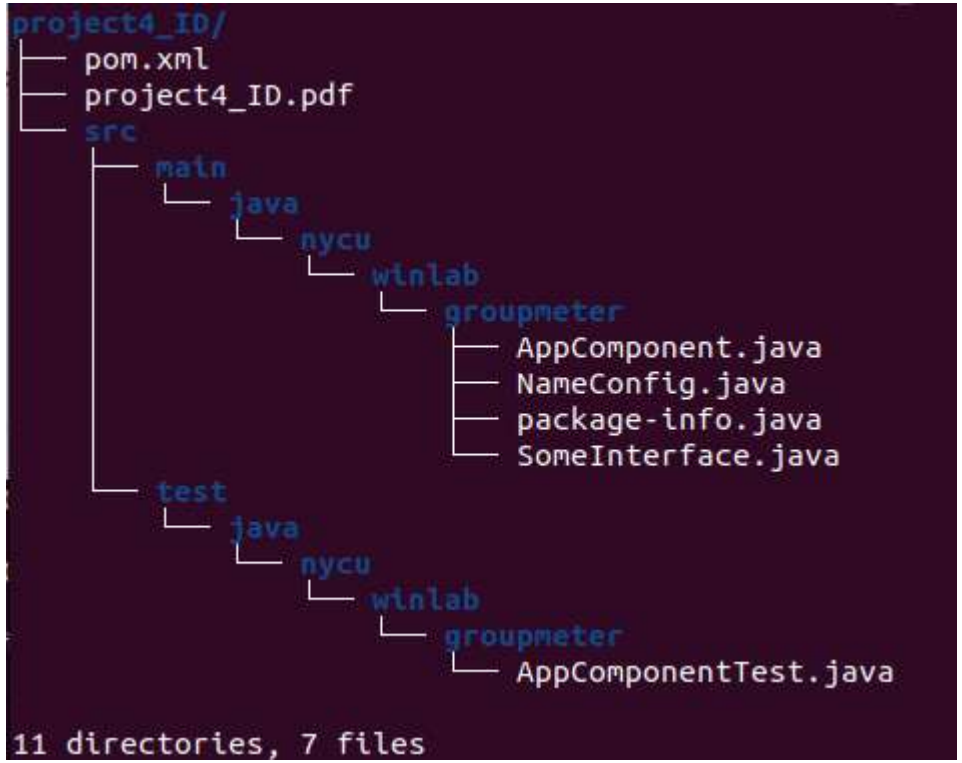
Scoring Criteria

- **(10%)** Report
  - Should contain screenshots mentioned in p.42, p.43
  - Briefly explain the result
- Reminder
  - You should not activate **fwd** application
  - We will **not** activate fwd before testing your App

```
andy@root > apps -s -a                                                    17:16:16
*   4 org.onosproject.optical-model          2.7.0    Optical Network Model
*  17 org.onosproject.drivers                2.7.0    Default Drivers
*  49 org.onosproject.hostprovider           2.7.0    Host Location Provider
*  50 org.onosproject.lldpprovider           2.7.0    LLDP Link Provider
*  51 org.onosproject.openflow-base          2.7.0    OpenFlow Base Provider
*  52 org.onosproject.openflow               2.7.0    OpenFlow Provider Suite
* 139 org.onosproject.gui2                   2.7.0    ONOS GUI2
* 178 nycu.winlab.groupmeter                1.0.SNAPSHOT Group, Meter, and Intent
```

Submission Naming Convention

- Rename your App directory as **project4_<student ID>**
- Rename your report as **project4_<student ID>.pdf**
- Compress the directory into a zip file named **project4_<student ID>**.zip
- Upload your zip file to E3.
- Wrong file name or format will result in 10-points deduction



Lab 4 Demo

- The demo dates will be in the week after lab 4 deadline.
- Demo questions will be shown when the demo starts.
- The questions involve modification of the code and the content related to the lecture and the lab

About help!

- For lab problem, ask at e3 forum
  - Ask at the e3 forum
  - TAs will help to clarify Lab contents instead of giving answers!
  - Please describe your questions with sufficient context,
    - e.g. Environment setup, Input/Output, Screenshots, …
- For personal problem mail to **sdnta@win.cs.nycu.edu.tw**
  - You have special problem and you can't meet the deadline
  - You got weird score with lab
- No Fixed TA hours

Outline

- Introduction to Group Table
- Introduction to Meter Table
- Introduction to Intent Service
- Introduction to Network Configuration Service
- Project 4 Overview
- Scoring Criteria, Submission, and Demo
- Reference

Reference

- org.onosproject.net.group
  - https://api.onosproject.org/2.4.0/apidocs/org/onosproject/net/group/package-summary.html
- org.onosproject.net.meter
  - https://api.onosproject.org/2.5.1/apidocs/org/onosproject/net/meter/package-summary.html
- org.onosproject.net.intent
  - https://api.onosproject.org/2.3.0/apidocs/org/onosproject/net/intent/package-summary.html

Q & A