Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Lab-course / Fachpraktikum

Computer Communication: Software-defined Networking

Summer Term 2020

Assignment 2

Consistent Network Updates

May 29th, 2020

Sukanya Bhowmik, David Hellmanns

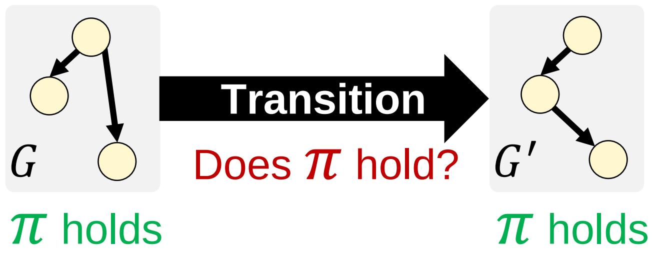
Overview

- **Task 2**
  - **Goals of this task**
  - 2.1 – Micro-Loops **[3 points]**
  - 2.2 – A Monitoring Scenario **[6 points]**
  - 2.3 – Controller-Phased Update **[6 points]**
- Deadline and Submission

Introduction – Update Consistency

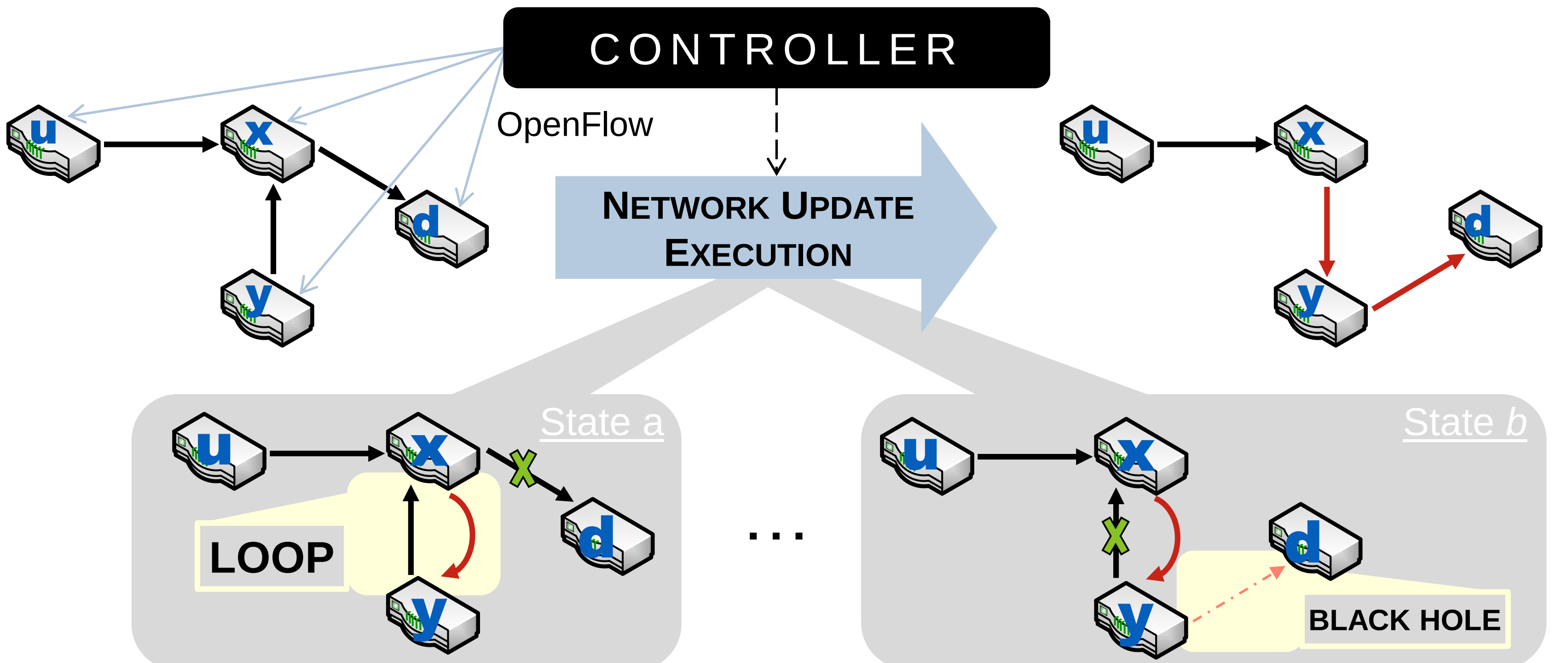
- Network state transition $G \rightarrow G'$
  - **I Planning**: determination of necessary flow updates
  - **II Execution**: execution of flow updates

- *Update Consistency*:
  Maintaining certain properties (invariants) during execution
  - E.g., path properties: *loop-freeness*, *drop-freeness*
  - Per-packet consistency:
    During an update, a packet is thoroughly processed according to the old or the new state (but no mixture of both states).



Introduction – Update Consistency – Example

- **Problem:** Concurrent updates not possible
  → Intermediate states, potentially violate invariants



Goals of this Task

要找到consistent的update順序

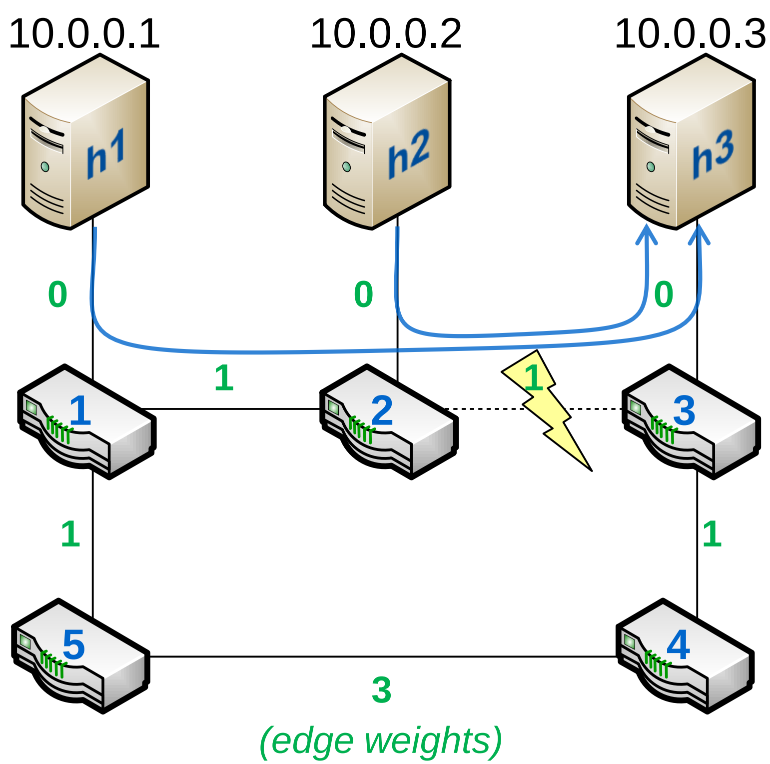
Perform Consistent Updates for OpenFlow Networks to avoid Black Holes and Micro-Loops:

- Ordered Updates
- 2-Phase Updates
  - Controller intermediary
  - Packet Marking (VLAN tagging)
    Cf. "*Consistent Updates for Software-Defined Networks: Change You Can Believe In!*", section 3.1
    http://frenetic-lang.org/publications/consistent-updates-hotnets11.pdf

Overview

- **Task 2**
  - Goals of this task
  - **2.1 – Micro-Loops**
  - 2.2 – A Monitoring Scenario
  - 2.3 – Controller-Phased Update
- Deadline and Submission

Task 2.1 – Micro-Loops



Check port numbers via net-cmd

- **Shortest path (min. weight) routing** from all (*s1–s5*) to *h3*
- *h1* & *h2* send packets to *h3*
- Update tables due to link congestion* (*s2*↔*s3*):

1. Naive ordered update *s1,s2,…,s5*: *What happens?*
2. Is there a better update order which avoids this problem?

*assume that the link weight between s2 and s3 suddenly increases from 1 to 100

Task 2.1 – Micro-Loops

- Write a bash script task21-init.sh, which pushes the initial flows using curl and the StaticEntryPusher REST API
- Complete scripts task21-naive-update.sh, to perform the naive ordered update, and task21-better-update.sh with your improved solution
  - In both of your update scripts, add a line with

    `sleep 1` simulate latency from real network

    after **each** curl request. This is to make the effects on non-predictable flow installation times (and the possible problems resulting therefrom) more visible.

Task 2.1 – Micro-Loops

- Start the Controller with

  `/opt/floodlight/floodlight-noforwarding.sh`

- Bring up Mininet with the given topology and xterms for all hosts:

```
~$ cd ex2
ex2$ sudo mn --custom microloop-topo.py
     --topo microloop --controller remote,port=6653
     --mac --arp
mininet> xterm h1 h2 h3
```

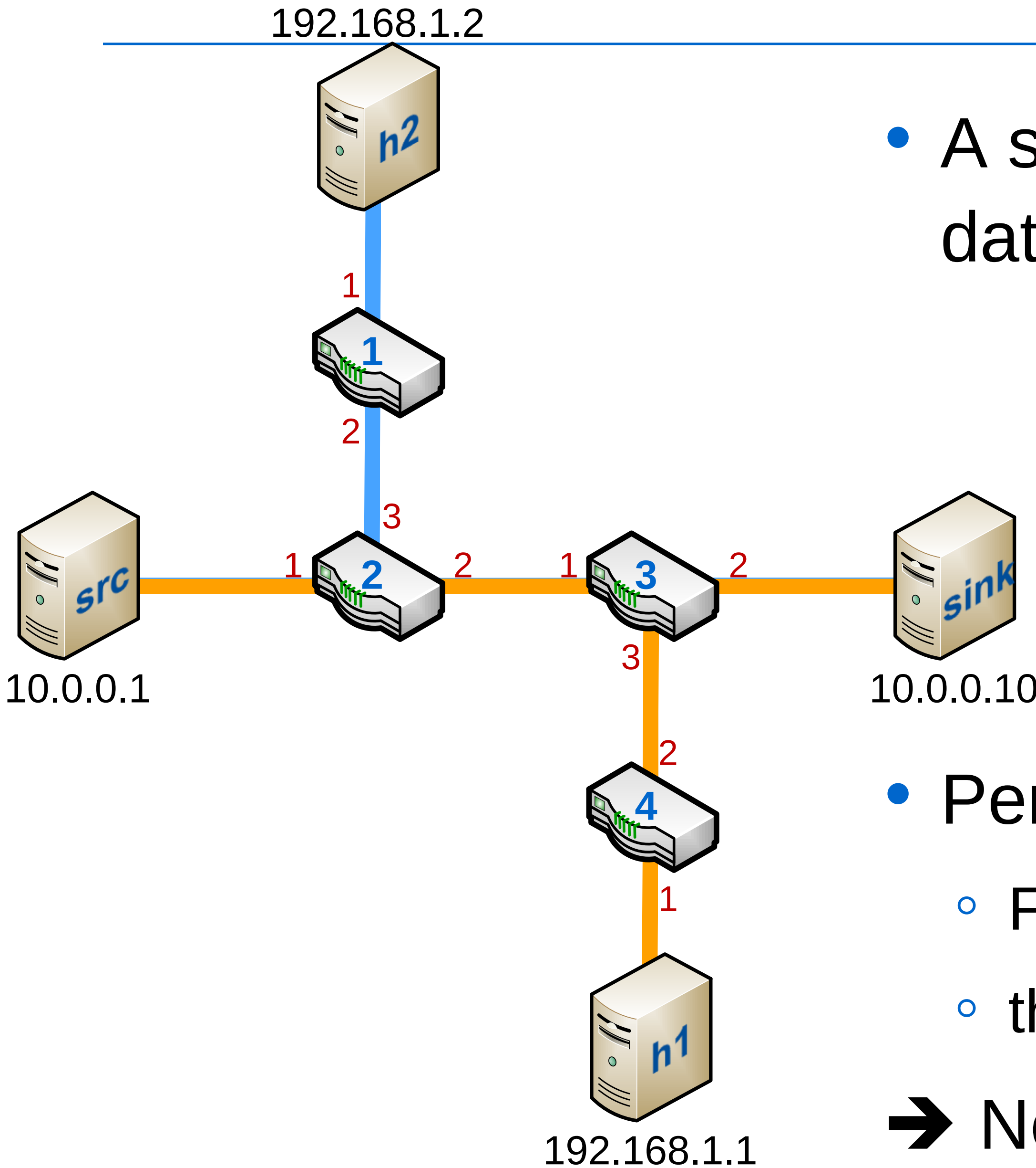
Task 2.1 – Micro-Loops

- Naïve update experiment:
  - Push your initial entries (shortest-path routes from all switches to *h3*) using task21-init.sh
  - On *h3*, run **`./udpreceiver 4000`** (observe the output in the terminal during your update!)
  - On *h1* and *h2*, run **`./udpsender 10.0.0.3 4000 600`**, i.e. send datagrams to 10.0.0.3:4000 for 60 seconds (or longer)
  - Perform the update using task21-naive-update.sh, and observe what happens in *h3*'s xterm
- Repeat the experiment with your task21-better-update.sh instead and compare what happens now to the naive update

Overview

- **Task 2**
  - Goals of this task
  - 2.1 – Micro-Loops
  - **2.2 – A Monitoring Scenario**
  - 2.3 – Controller-Phased Update
- Deadline and Submission

Task 2.2 – A Monitoring Scenario



- A source *(src)* periodically sends datagrams to a *sink*.
  - Each datagram should be received by *sink* **and either** *h1* or *h2*, but **not both**! (Think of two monitors …)
- Perform consistent update!
  - First, *sink & h1* are recipients,
  - then *sink & h2*.

➔ No Multicast Groups!

Task 2.2 – A Monitoring Scenario

- Write a bash script task22-init.sh, which pushes the initial flows.
- 用vlan tag
  Write a script task22-update.sh, which performs the consistent update (**packet marking**), both using the REST API
  - Like in the previous task, add a line with

    ```
    sleep 1
    ```

    after **each** curl request in your task22-update.sh script.

Task 2.2 – A Monitoring Scenario

- Start the Controller with

  `/opt/floodlight/floodlight-noforwarding.sh`

- Bring up Mininet with the given topology and xterms for all hosts:

```
~$ cd ex2
ex2$ sudo mn --custom monitors-topo.py
     --topo monitors --controller remote,port=6653
     --mac --arp
mininet> xterm h1 h2 src sink
```

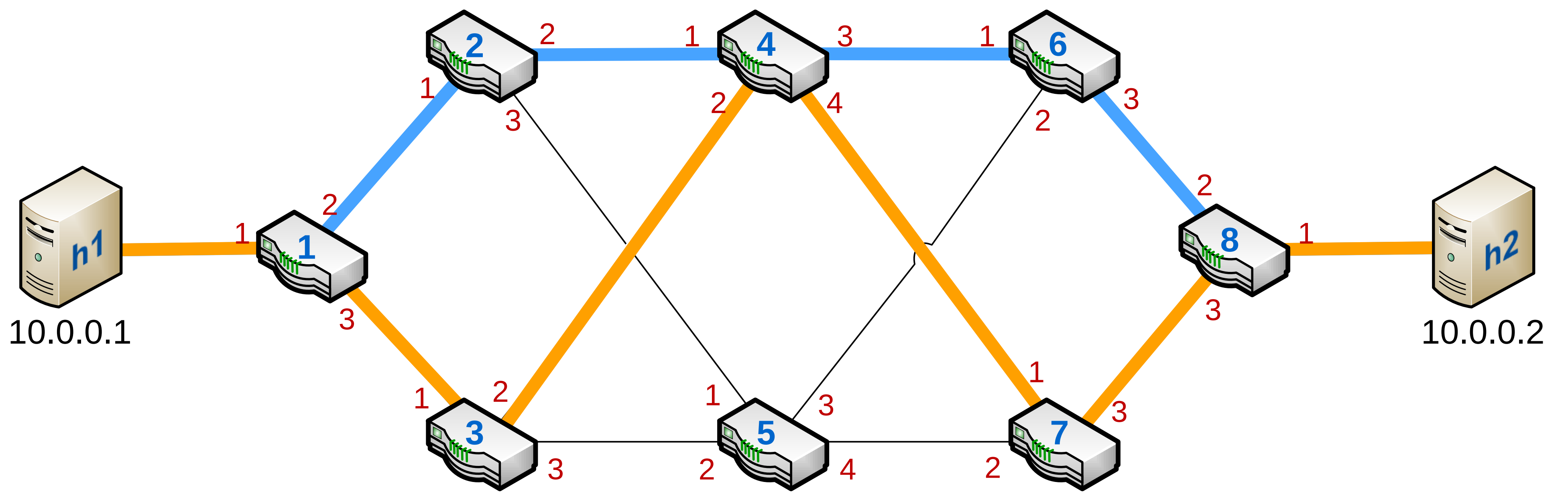
Task 2.2 – A Monitoring Scenario

- Push your initial entries (towards *h1* and *sink*) using task22-init.sh
- On *h1*, *h2*, and *sink*, run `./udpreceiver 4000` (possibly redirect the outputs into files h1.log, h2.log and srv.log for later inspection, but you can also look at the output in the terminals)
- On *src*, run `./udpsender 10.0.0.10 4000 600`, i.e. send datagrams to 10.0.0.10:4000 for 60 seconds (or longer if you like)
- Perform your consistent update using task22-update.sh
- Check in the udpreceiver-outputs that
  - …all datagrams (consecutive numbers) were received by *sink*
  - …each datagram received by *sink* was also received by *h1* or *h2*
  - …no datagram was received by both *h1* and *h2*

Overview

- **Task 2**
  - Goals of this task
  - 2.1 – Micro-Loops
  - 2.2 – A Monitoring Scenario
  - **2.3 – Controller-Phased Update**
- Deadline and Submission

Task 2.3 – Controller-Phased Update



- Perform consistent update *(orange → blue)* in two phases!
- Route intermediate packets **over controller**
- Packets **must not** be lost *(drop-freeness)*
- Packets **must not** be routed along s1 → s2 → s4 → s7 → s8 *or* s1 → s3 → s4 → s6 → s8! *(per-packet consistency)*

Task 2.3 – Controller-Phased Update

- Bring up Mininet with the given topology and xterms for h1 & h2:

```
~$ cd ex2
ex2$ sudo mn --custom regular-topo.py
     --topo regular --controller remote --arp
mininet> xterm h1 h2
```

important!

- On h2, run `./udpreceiver 4000` (receive on port 4000)
- On h1, run `./udpsender 10.0.0.2 4000 600`, this will send 600 datagrams to h2:4000 (at a rate of 10 datagrams per second)
- Make sure your controller performs the update while datagrams are being transmitted
- Check in h2's xterm that datagrams are still received **during** and **after** the route update

Task 2.3 – Controller-Phased Update

- Implement your solution in a Floodlight module net.sdnlab.ex2.Task23 (more details on following slide)
- At the beginning, route the flows over the orange path
  - You may use Floodlight topology services or simply hard-code topology knowledge
- Update Flow Tables so that
  - Intermediate packets are routed over the controller (no tagging)
  - Finally the flow is forwarded completely over the blue path
- Suggestion: set a **hard timeout** on the initial flow entry on s1. That way, you will receive a packet_in event at the controller when it is time to perform the update…

Task 2.3 – Controller-Phased Update

- Create a Floodlight Module stub class in Eclipse as follows:
  - Start under workspace /home/student/eclipse-workspace
  - Open project floodlight and right-click project folder
  - New->Class (src folder should be floodlight/src/main/java)
  - Package: net.sdnlab.ex2
  - Name: Task23
  - Add the following Interfaces (using search function under Add…):
    - IFloodlightModule
    - IOFMessageListener
  - Finish

Task 2.3 – Controller-Phased Update

- To register your module, add a line
  net.sdnlab.ex2.Task23
  to the file
  src/main/resources/META-INF/services/
  net.floodlightcontroller.core.module.IFloodlightModule
- Create a dedicated task23.properties file modelled on src/main/resources/floodlightnoforwarding.properties, where you add your module to the floodlight.modules field to have it loaded
- Create a launch configuration Task23.launch referencing this task23.properties file (e.g., modelled on Floodlight-Quatum-Conf.launch), so that you can start a Floodlight instance that loads your module

Overview

- Task 2
- **Deadline and Submission**

Deadline and Submission

- When (submission deadline): June 12th 2020, 9:45
- 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 bash scripts
  3. Archive of source package net.sdnlab.ex2 (e.g. Eclipse export…)
- Be prepared to show a live demo to the supervisor during the next meeting