Mininet(2): SDN-based Routing

Week 7

Contents

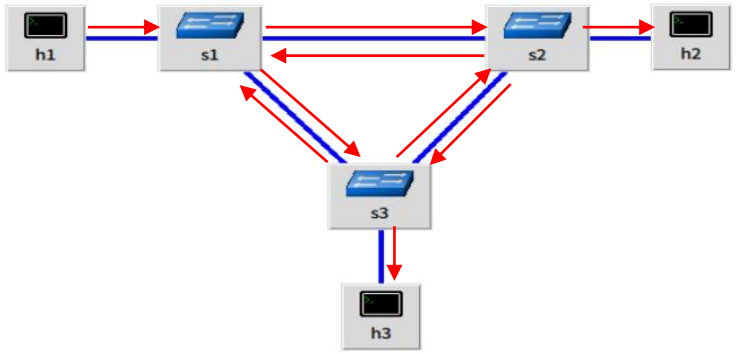
Getting started

Getting started

■ Switching Loop and STP

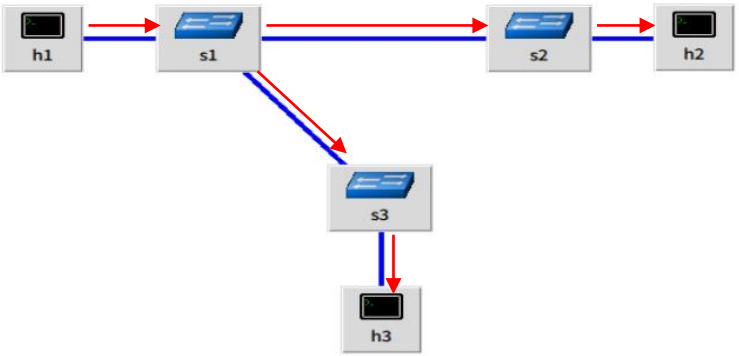
– Switching Loop

- Loop 형태의 Topology에서 Frame이 끝없이 순환하여 일어나는 현상
- 네트워크의 마비를 일으킨다
- Switching Loop에 의한 현상
  - Broadcast Strom
    - » Broadcast되는 Frame이 Loop topology에서 끊임없이 돌고 도는 현상
  - Multi Fame Copies
    - » 시계방향으로 도는 Frame과 반대방향으로 도는 Frame이 모두 중복되어 수신된다.



– STP

- Switching Loop를 해결하기 위한 프로토콜
- 네트워크의 특정 포트를 차단시켜 Tree topology 구성
  - Tree 구조를 만듦으로써 사용하지 않게 되는 port가 생기게 된다.
- 특정 포트가 Down되면 다시 새로운 Spanning Tree를 구성한다.



Getting started

■ Static Routing and Dynamic Routing

– Static Routing

- 관리자가 각 라우터의 라우팅 테이블에 경로를 수동으로 추가해주는 라우팅
  - 장점: 라우팅 경로 계산을 위한 리소스가 들지 않는다
  - 단점 : 특정 경로가 다운되거나 해서 경로를 바꾸려면
    관리자가 각 라우터의 라우팅 테이블을 일일이 수정해줘야 한다.

– Dynamic Routing

- 라우터가 최적의 경로를 직접 계산해서 자동으로 라우팅 테이블을 업데이트하는 라우팅
  - 장점: 특정 링크가 다운되었을 때 같은 링크 상태 변화에 대응할 수 있다.
  - 단점: 라우팅 경로 계산을 위한 리소스가 많이 든다.
- 주로 Shortest Path Algorithm을 활용한 Routing으로 구현된다.
  - Shortest Path Algorithm
    - » Floyd-Warshall Algorithm : 모든 노드 사이의 최단 거리를 구하는 알고리즘
    - » Dijkstra Algorithm : 한 노드에서 다른 노드로 가는 최단 거리를 구하는 알고리즘

Getting started

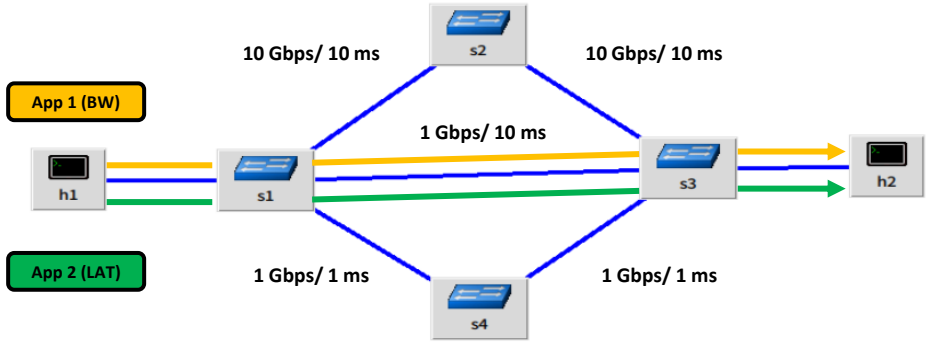
■ Application-aware routing

– **Application의 특성을 고려하여 routing 경로를 설정해주는 방식**
  - Application별 Requirement를 충족시켜줄 수 있는 최적의 경로 설정

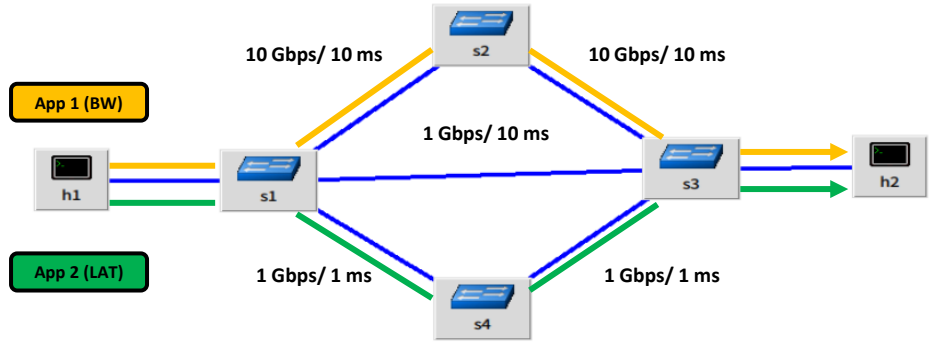
– 과거
  - Application 종류가 다양하지 않아 중요성↓
  - Port number를 활용하여 간단하게 Application 식별 가능

– 현재
  - Application 종류가 다양해지면서 중요성↑
  - 최근 Application들은 대부분 HTTP/HTTPS 프로토콜 기반
    – Port number를 통한 식별 불가능 (HTTP: 80, HTTPS: 443)
  - 추가적인 Application 식별 매커니즘이 필요
    – Ex) AI를 활용한 stream 패턴 인식

Ex)
App 1 : 큰 대역폭을 요구하는 Application
(4k화질의 동영상, …)
App 2: 낮은 지연시간을 요구하는 Application
(실시간 스트리밍 영상, …)



Shortest path routing



Application aware routing

Getting started

■ Application-aware routing

- **SDN을 활용하면 Application-aware routing 효율적 구현 가능**

- Legacy Network
  - 각 라우터마다 Application 식별 매커니즘 적용
    - 고성능의 라우터 요구
    - 라우터가 라우팅에 집중 x
    - 네트워크 상황을 고려한 유동적인 경로 제어 x

- SDN Network
  - 고성능의 Controller가 Application 식별 후 최적의 경로 설정하여 각 OpenFlow Switch들에게 전달
    - 고성능의 OpenFlow Switch 필요 x
    - OpenFlow Switch는 Forwarding에만 집중 가능
    - 네트워크 상황을 고려한 유동적인 경로 제어 o

Experiment

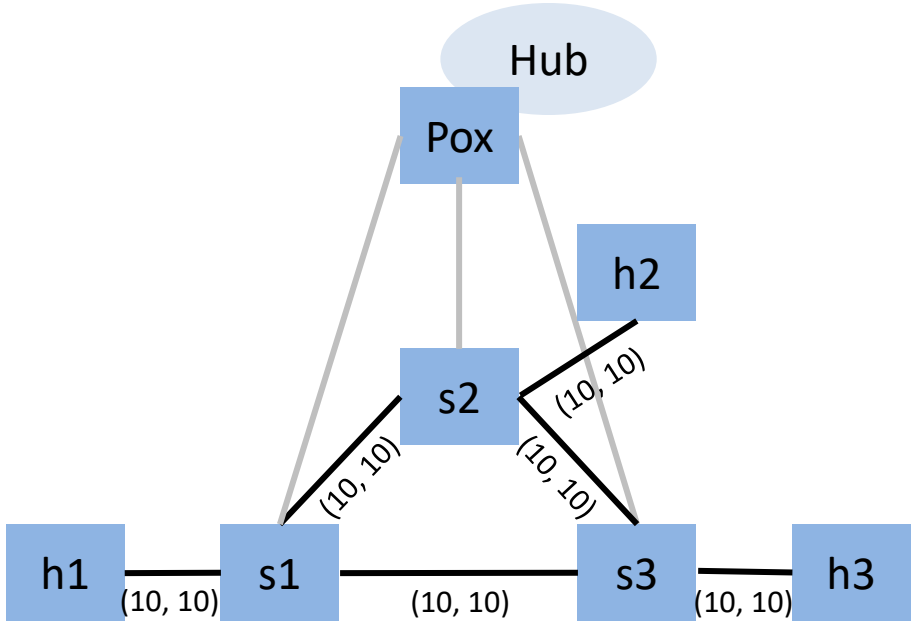
Experiment

■ **Experiment 1: SDN-based Hub with STP**

(각 링크의 BW(Mbps), 각 링크의 delay(ms))

– 실험 1-1: Switching Loop 현상 확인하기

- Pox controller에서 hub.py 실행
  - » ./pox/pox.py forwarding.hub
- Mininet을 활용하여 Ring topology 구성
  - – 3 Switch, 3 Host
    - » sudo mn --custom ~/mininet/custom/Ring.py --topo=mytopo,3 --mac --switch=ovsk --controller=remote --link tc,bw=10,delay=10ms
- h1과 h3 사이 Ping Test 후 결과 분석
  - » h1 ping h3



Experiment

- **Experiment 1: SDN-based Hub with STP**
  - 실험 1-1: Switching Loop 현상 확인하기
    - Pox controller에서 hub.py 실행
      - » ./pox/pox.py forwarding.hub

```
ubuntu@ubuntu-VirtualBox:~$ ./pox/pox.py forwarding.hub
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al
.
INFO:forwarding.hub:Proactive hub running.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-02
```

Experiment

- **Experiment 1: SDN-based Hub with STP**
  - 실험 1-1: Switching Loop 현상 확인하기
    - Mininet을 활용하여 Ring topology 구성
      - 3 Switch, 3 Host
        - » sudo mn --custom ~/mininet/custom/Ring.py --topo=mytopo,3 --mac --switch=ovsk --controller=remote --link tc,bw=10,delay=10ms

```
ubuntu@ubuntu-VirtualBox:~$ sudo mn --custom ~/mininet/custom/Ring.py
 --topo=mytopo,3 --mac --switch=ovsk --controller=remote --link tc,bw
=10,delay=10ms
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (h1, s1) (10.00Mbit 10m
s delay) (10.00Mbit 10ms delay) (h2, s2) (10.00Mbit 10ms delay) (10.0
0Mbit 10ms delay) (h3, s3) (10.00Mbit 10ms delay) (10.00Mbit 10ms del
ay) (s1, s2) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s1, s3) (
10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s2, s3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay)
*** Starting CLI:
mininet>
```

Experiment

- **Experiment 1: SDN-based Hub with STP**
  - 실험 1-1: Switching Loop 현상 확인하기
    - h1과 h3 사이 Ping Test 후 결과 분석
      - » h1 ping h3

    - Switching Loop에 의해 Packet이 중복 수신 되는 것을 확인할 수 있다.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1061 ms
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1166 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1237 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1276 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1278 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1342 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1354 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=407 ms
```

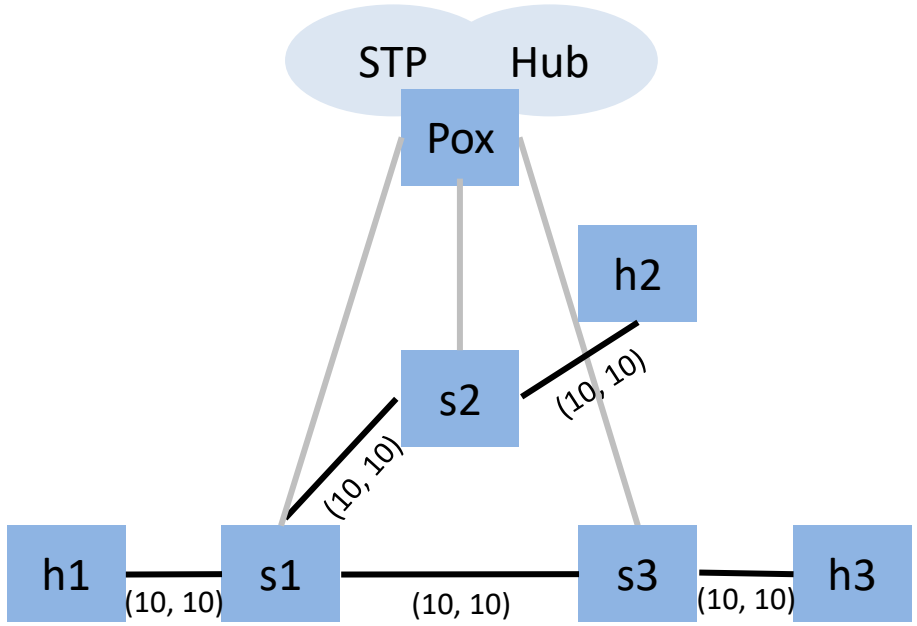
Experiment

■ Experiment 1: SDN-based Hub with STP

(각 링크의 BW(Mbps), 각 링크의 delay(ms))

– 실험 1-2: Pox controller 활용하여 Spanning Tree 구현하기

- Pox controller에서 hub.py, discovery.py, spanning_tree.py 실행
  - » ./pox/pox.py forwarding.hub openflow.discovery openflow.spanning_tree
- Mininet을 활용하여 Ring topology 구성
  - – 3 Switch, 3 Host
    - » sudo mn --custom ~/mininet/custom/Ring.py --topo=mytopo,3 --mac --switch=ovsk --controller=remote --link tc,bw=10,delay=10ms
- 명령어를 통해 Tree 구성 확인
  - » dpctl dump-ports-desc
- h1 과 h3 사이 ping test를 통해 Latency 측정
  - – 0.01초의 인터벌로 1000개 보내기
    - » h1 ping -c1000 -i0.01 h3
- h2과 h3 사이 ping test를 통해 Latency 측정
  - – 0.01초의 인터벌로 1000개 보내기
    - » h2 ping -c1000 -i0.01 h3



Experiment

■ Experiment 1: SDN-based Hub with STP

– 실험 1-2: Pox controller 활용하여 Spanning Tree 구현하기

- Pox controller에서 hub.py, discovery.py, spanning_tree.py 실행
  - » ./pox/pox.py forwarding.hub openflow.discovery openflow.spanning_tree

- discovery.py : Controller가 전체 topology를 파악할 수 있게 해준다. (LLDP protocol 이용)
- spanning_tree.py : Spanning tree를 구성한다.

```
ubuntu@ubuntu-VirtualBox:~$ ./pox/pox.py forwarding.hub open
flow.discovery openflow.spanning_tree
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al
.
INFO:forwarding.hub:Proactive hub running.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-02
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -
> 00-00-00-00-00-01.2
```

Experiment

- **Experiment 1: SDN-based Hub with STP**
  - 실험 1-2: Pox controller 활용하여 Spanning Tree 구현하기
    - Mininet을 활용하여 Ring topology 구성
      - 3 Switch, 3 Host
        - » sudo mn --custom ~/mininet/custom/Ring.py --topo=mytopo,3 --mac --switch=ovsk --controller=remote --link tc,bw=10,delay=10ms

```
ubuntu@ubuntu-VirtualBox:~$ sudo mn --custom ~/mininet/custom/Ring.py
 --topo=mytopo,3 --mac --switch=ovsk --controller=remote --link tc,bw
=10,delay=10ms
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (h1, s1) (10.00Mbit 10m
s delay) (10.00Mbit 10ms delay) (h2, s2) (10.00Mbit 10ms delay) (10.0
0Mbit 10ms delay) (h3, s3) (10.00Mbit 10ms delay) (10.00Mbit 10ms del
ay) (s1, s2) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s1, s3) (
10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s2, s3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay)
*** Starting CLI:
mininet>
```

Experiment

- **Experiment 1: SDN-based Hub with STP**
  - 실험 1-2: Pox controller 활용하여 Spanning Tree 구현하기
    - 명령어를 통해 Tree 구성 확인
      - » dpctl dump-ports-desc
    - s2-eth3, s3-eth2가 NO-FLOOD 상태로 변경되어 Spanning Tree가 구현된 것을 확인할 수 있다.

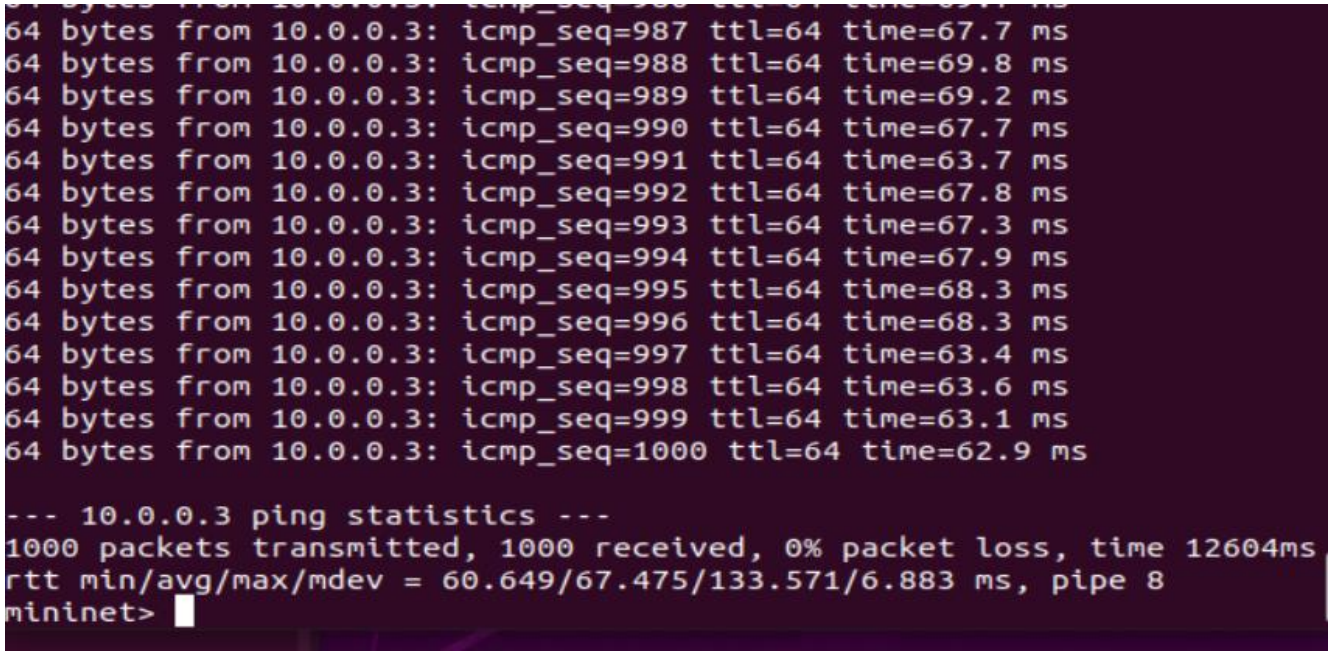
```
mininet> dpctl dump-ports-desc
*** s1 ----------------------------------
----------
OFPST_PORT_DESC reply (xid=0x2):
 1(s1-eth1): addr:b6:fe:4b:57:95:ef
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:aa:34:58:6f:61:c2
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:ba:17:85:9e:85:24
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:22:63:2f:88:56:44
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
```

```
*** s2 ----------------------------------
----------
OFPST_PORT_DESC reply (xid=0x2):
 1(s2-eth1): addr:be:51:86:39:dc:a6
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s2-eth2): addr:3e:5a:9f:f7:2d:14
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s2-eth3): addr:be:b1:f1:90:43:ed
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s2): addr:22:4b:a4:4d:3d:43
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
*** s3 ----------------------------------
```

```
*** s3 ----------------------------------
----------
OFPST_PORT_DESC reply (xid=0x2):
 1(s3-eth1): addr:46:f3:9f:f8:24:76
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s3-eth2): addr:56:76:7d:e4:ef:66
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s3-eth3): addr:d2:81:9f:30:09:e6
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s3): addr:fa:27:b5:14:1b:46
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
```

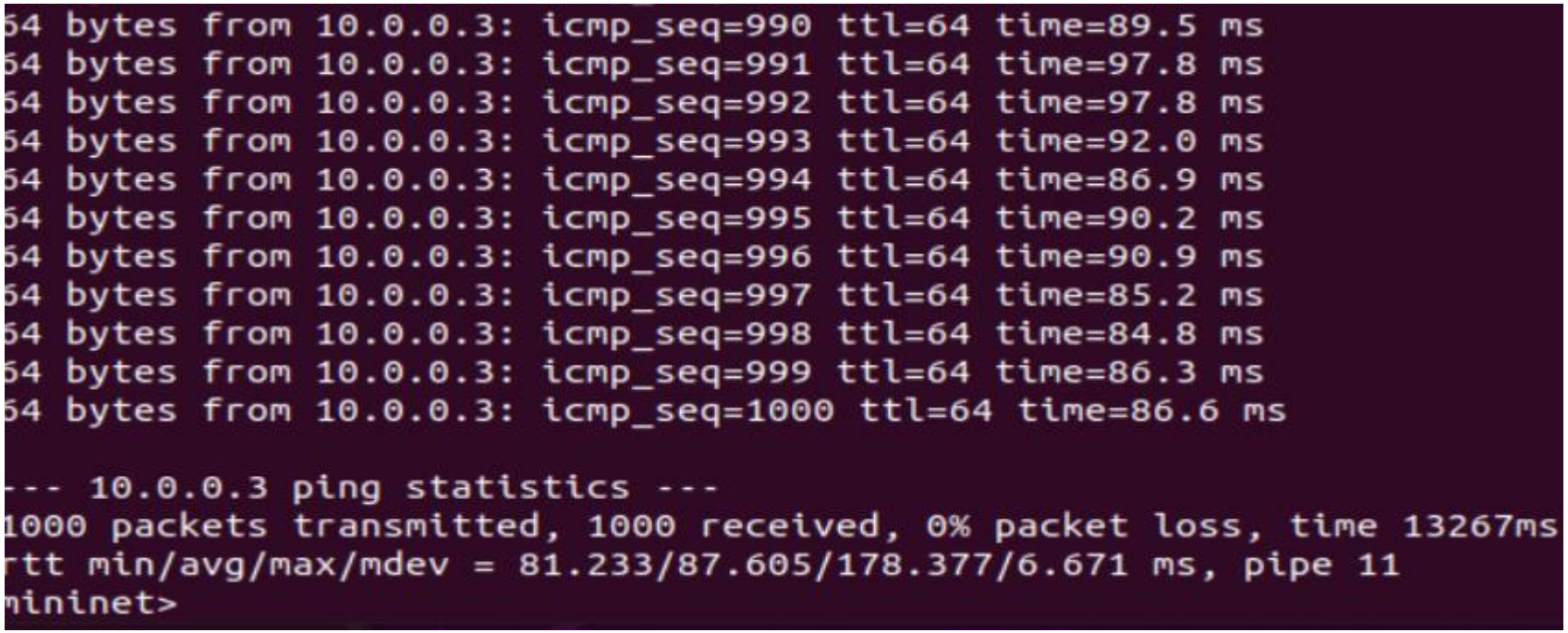
Experiment

- **Experiment 1: SDN-based Hub with STP**
  - 실험 1-2: Pox controller 활용하여 Spanning Tree 구현하기
    - h1 과 h3 사이 ping test를 통해 Latency 측정
      - 0.01초의 인터벌로 1000개 보내기
        - » h1 ping -c1000 -i0.01 h3
    - 실험 1-1 과 달리 ping이 잘 전달되는 것을 확인할 수 있다
    - Delay 10ms의 link를 6번 거치므로 Average RTT $\approx 60 + \alpha$ 인 것을 확인할 수 있다



Experiment

- **Experiment 1: SDN-based Hub with STP**
  - 실험 1-2: Pox controller 활용하여 Spanning Tree 구현하기
    - h2과 h3 사이 ping test를 통해 Latency 측정
      - 0.01초의 인터벌로 1000개 보내기
        - » h2 ping -c1000 -i0.01 h3
    - Delay 10ms의 link를 8번 거치므로 Average RTT ≈ $80 + \alpha$ 인 것을 확인할 수 있다



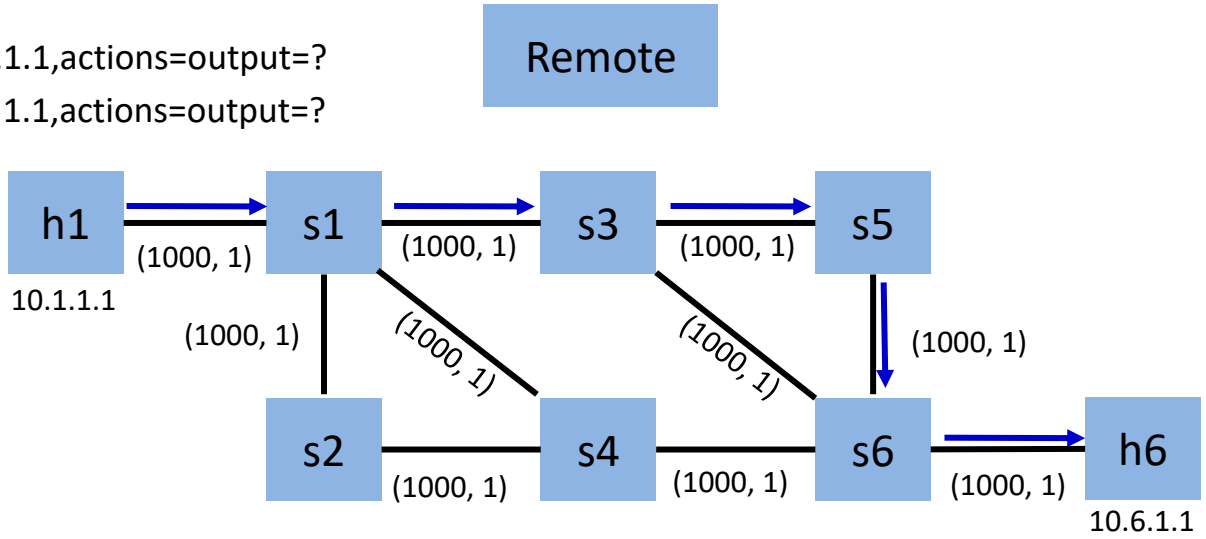
Experiment

■ Experiment 2: SDN-based Static Routing

(각 링크의 BW(Mbps), 각 링크의 delay(ms))

– 실험 2-1 : Static Routing 구현하기
h1 → s1 → s3 → s5 → s6 → h6 경로의 static routing 구현

- Mininet을 활용하여 Week7 topology 구성
  - » sudo mn --custom ~/mininet/custom/Week7.py --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc
- 현재 Switch들의 포트 연결 상태 확인
  - » net
- 다음 경로를 생각하여 Flow Table에 Flow 추가하기
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?
- Flow Table을 확인하여 추가된 경로 검토하기
  - » dpctl dump-flows
- h1과 h6 사이 ping test를 통해 Latency 측정
  - – 0.01초의 인터벌로 1000개 보내기
    - » h1 ping -c1000 -i0.01 h6



Experiment

■ Experiment 2: SDN-based Static Routing

– 실험 2-1 : Static Routing 구현하기
h1 → s1 → s3 → s5 → s6 → h6 경로의 static routing 구현

- Mininet을 활용하여 Week7 topology 구성
  - » sudo mn --custom ~/mininet/custom/Week7.py --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc

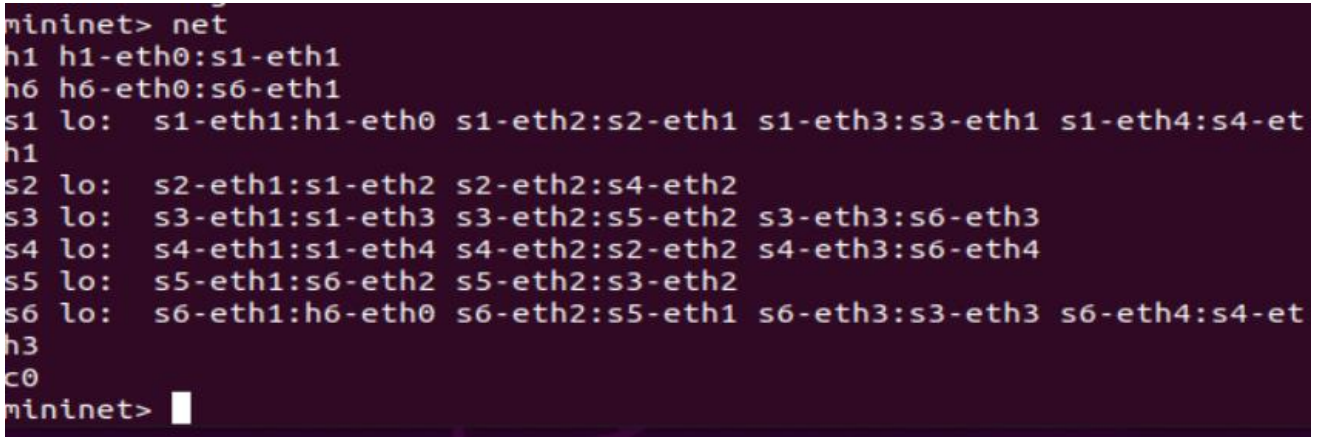
```
ubuntu@ubuntu-VirtualBox:~$ sudo mn --custom ~/mininet/custom/Week7.p
y --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc
[sudo] ubuntu의 암호:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h6
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, h1) (1000.00Mbit
 1ms delay) (1000.00Mbit 1ms delay) (s1, s2) (1000.00Mbit 1ms delay)
(1000.00Mbit 1ms delay) (s1, s3) (1000.00Mbit 1ms delay) (1000.00Mbit
 1ms delay) (s1, s4) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay)
(s2, s4) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s3, s5) (10
00.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s3, s6) (1000.00Mbit 1m
s delay) (1000.00Mbit 1ms delay) (s4, s6) (1000.00Mbit 1ms delay) (10
00.00Mbit 1ms delay) (s5, s6) (1000.00Mbit 1ms delay) (1000.00Mbit 1m
s delay) (s6, h6)
*** Configuring hosts
h1 h6
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay)
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms dela
y) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms d
```

Experiment

■ Experiment 2: SDN-based Static Routing

– 실험 2-1 : Static Routing 구현하기
h1 → s1 → s3 → s5 → s6 → h6 경로의 static routing 구현

- 현재 Switch들의 포트 연결 상태 확인
  - » net
- Ex) s1의 eth2 포트는 s2의 eth1 포트에 연결되어 있다



Experiment

- **Experiment 2: SDN-based Static Routing**
  - 실험 2-1 : Static Routing 구현하기
    h1 → s1 → s3 → s5 → s6 → h6 경로의 static routing 구현
    - 다음 경로를 생각하여 Flow Table에 Flow 추가하기
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?

```
mininet> sh ovs-ofctl s1 add-flow in_port=1,nw_dst=10.6.1.1,actions=output=3
ovs-ofctl: unknown command 's1'; use --help for help
mininet> sh ovs-ofctl add-flow s1 in_port=1,nw_dst=10.6.1.1,actions=output=3
2021-08-26T14:25:03Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:25:03Z|00002|ofp_match|INFO| pre: in_port=1,nw_dst=10.6.1.1
2021-08-26T14:25:03Z|00003|ofp_match|INFO|post: in_port=1
mininet> sh ovs-ofctl add-flow s3 in_port=1,nw_dst=10.6.1.1,actions=output=2
2021-08-26T14:26:27Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:26:27Z|00002|ofp_match|INFO| pre: in_port=1,nw_dst=10.6.1.1
2021-08-26T14:26:27Z|00003|ofp_match|INFO|post: in_port=1
mininet> sh ovs-ofctl add-flow s5 in_port=2,nw_dst=10.6.1.1,actions=output=1
2021-08-26T14:26:53Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:26:53Z|00002|ofp_match|INFO| pre: in_port=2,nw_dst=10.6.1.1
2021-08-26T14:26:53Z|00003|ofp_match|INFO|post: in_port=2
mininet> sh ovs-ofctl add-flow s6 in_port=2,nw_dst=10.6.1.1,actions=output=1
2021-08-26T14:27:37Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:27:37Z|00002|ofp_match|INFO| pre: in_port=2,nw_dst=10.6.1.1
2021-08-26T14:27:37Z|00003|ofp_match|INFO|post: in_port=2
mininet> sh ovs-ofctl add-flow s6 in_port=1,nw_dst=10.1.1.1,actions=output=2
2021-08-26T14:27:47Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
```

```
mininet> sh ovs-ofctl add-flow s6 in_port=1,nw_dst=10.1.1.1,actions=output=2
2021-08-26T14:27:47Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:27:47Z|00002|ofp_match|INFO| pre: in_port=1,nw_dst=10.1.1.1
2021-08-26T14:27:47Z|00003|ofp_match|INFO|post: in_port=1
mininet> sh ovs-ofctl add-flow s5 in_port=1,nw_dst=10.1.1.1,actions=output=2
2021-08-26T14:27:57Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:27:57Z|00002|ofp_match|INFO| pre: in_port=1,nw_dst=10.1.1.1
2021-08-26T14:27:57Z|00003|ofp_match|INFO|post: in_port=1
mininet> sh ovs-ofctl add-flow s3 in_port=2,nw_dst=10.1.1.1,actions=output=1
2021-08-26T14:28:08Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:28:08Z|00002|ofp_match|INFO| pre: in_port=2,nw_dst=10.1.1.1
2021-08-26T14:28:08Z|00003|ofp_match|INFO|post: in_port=2
mininet> sh ovs-ofctl add-flow s1 in_port=3,nw_dst=10.1.1.1,actions=output=1
2021-08-26T14:28:21Z|00001|ofp_match|INFO|normalization changed ofp_match, de
s:
2021-08-26T14:28:21Z|00002|ofp_match|INFO| pre: in_port=3,nw_dst=10.1.1.1
2021-08-26T14:28:21Z|00003|ofp_match|INFO|post: in_port=3
```

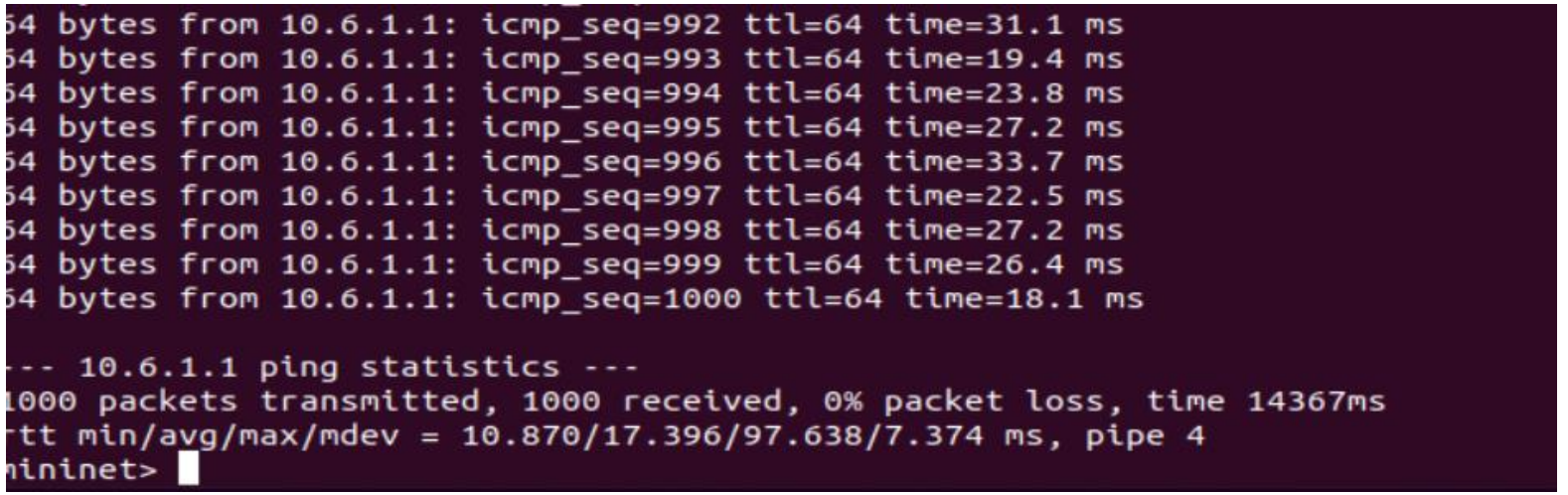
Experiment

- **Experiment 2: SDN-based Static Routing**
  - 실험 2-1 : Static Routing 구현하기
    h1 → s1 → s3 → s5 → s6 → h6 경로의 static routing 구현
    - Flow Table을 확인하여 추가된 경로 검토하기
      - » dpctl dump-flows

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=213.180s, table=0, n_packets=6, n_bytes=448, in_port="s1-et
h1" actions=output:"s1-eth3"
 cookie=0x0, duration=14.774s, table=0, n_packets=5, n_bytes=378, in_port="s1-eth
3" actions=output:"s1-eth1"
*** s2 ------------------------------------------------------------------------
*** s3 ------------------------------------------------------------------------
 cookie=0x0, duration=129.341s, table=0, n_packets=6, n_bytes=448, in_port="s3-et
h1" actions=output:"s3-eth2"
 cookie=0x0, duration=27.678s, table=0, n_packets=5, n_bytes=378, in_port="s3-eth
2" actions=output:"s3-eth1"
*** s4 ------------------------------------------------------------------------
*** s5 ------------------------------------------------------------------------
 cookie=0x0, duration=103.438s, table=0, n_packets=6, n_bytes=448, in_port="s5-et
h2" actions=output:"s5-eth1"
 cookie=0x0, duration=39.343s, table=0, n_packets=5, n_bytes=378, in_port="s5-eth
1" actions=output:"s5-eth2"
*** s6 ------------------------------------------------------------------------
 cookie=0x0, duration=59.284s, table=0, n_packets=6, n_bytes=448, in_port="s6-eth
2" actions=output:"s6-eth1"
 cookie=0x0, duration=48.971s, table=0, n_packets=5, n_bytes=378, in_port="s6-eth
1" actions=output:"s6-eth2"
```

Experiment

- **Experiment 2: SDN-based Static Routing**
  - 실험 2-1 : Static Routing 구현하기
    h1 → s1 → s3 → s5 → s6 → h6 경로의 static routing 구현
    - h1과 h6 사이 ping test를 통해 Latency 측정
      - 0.01초의 인터벌로 1000개 보내기
        - » h1 ping -c1000 -i0.01 h6



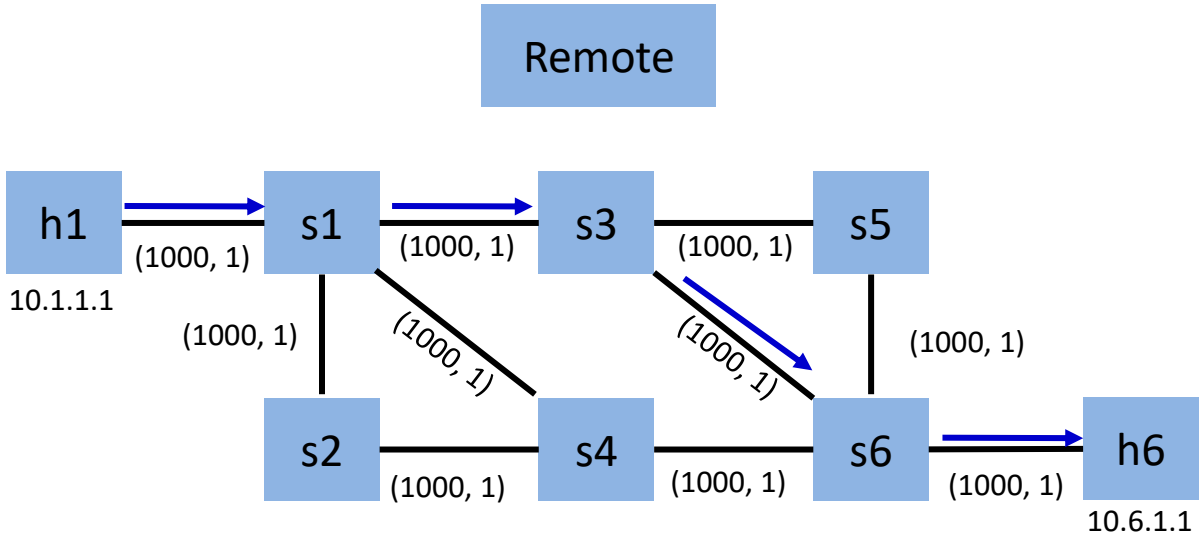
Experiment

■ Experiment 2: SDN-based Static Routing

(각 링크의 BW(Mbps), 각 링크의 delay(ms))

– 실험 2-2 : Routing 경로 변경하기
h1 → s1 → s3 → s6 → h6 경로의 static routing 구현

- 실험 2-1 Flow Table들의 Flow 수정하기
  - » sh ovs-ofctl del-flow sN
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?
- Flow Table을 확인하여 추가된 경로 검토하기
  - » dpctl dump-flows
- h1과 h6 사이 ping test를 통해 Latency 측정
  - – 0.01초의 인터벌로 1000개 보내기
    - » h1 ping -c1000 -i0.01 h6
  - – 실험 2-1과 Latency 비교해보기



Experiment

- **Experiment 2: SDN-based Static Routing**
  - 실험 2-2 : Routing 경로 변경하기
    h1 → s1 → s3 → s6 → h6 경로의 static routing 구현
    - 실험 2-1 Flow Table들의 Flow 수정하기
      - » sh ovs-ofctl del-flow sN
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?

Experiment

- **Experiment 2: SDN-based Static Routing**
  - 실험 2-2 : Routing 경로 변경하기
    h1 → s1 → s3 → s6 → h6 경로의 static routing 구현
    - Flow Table을 확인하여 추가된 경로 검토하기
      - » dpctl dump-flows

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=172.062s, table=0, n_packets=4, n_bytes=280, in_port="s1-et
h1" actions=output:"s1-eth3"
 cookie=0x0, duration=35.990s, table=0, n_packets=4, n_bytes=280, in_port="s1-eth
3" actions=output:"s1-eth1"
*** s2 ------------------------------------------------------------------------
*** s3 ------------------------------------------------------------------------
 cookie=0x0, duration=139.265s, table=0, n_packets=4, n_bytes=280, in_port="s3-et
h1" actions=output:"s3-eth3"
 cookie=0x0, duration=51.517s, table=0, n_packets=4, n_bytes=280, in_port="s3-eth
3" actions=output:"s3-eth1"
*** s4 ------------------------------------------------------------------------
*** s5 ------------------------------------------------------------------------
*** s6 ------------------------------------------------------------------------
 cookie=0x0, duration=110.933s, table=0, n_packets=4, n_bytes=280, in_port="s6-et
h3" actions=output:"s6-eth1"
 cookie=0x0, duration=99.269s, table=0, n_packets=5, n_bytes=350, in_port="s6-eth
1" actions=output:"s6-eth3"
mininet>
```

Experiment

- **Experiment 2: SDN-based Static Routing**
  - 실험 2-2 : Routing 경로 변경하기
    h1 → s1 → s3 → s6 → h6 경로의 static routing 구현
    - h1과 h6 사이 ping test를 통해 Latency 측정
      - 0.01초의 인터벌로 1000개 보내기
        - » h1 ping -c1000 -i0.01 h6

    - 실험 2-1의 경로에 비해 짧은 경로이므로 RTT의 값이 줄어든 것을 확인할 수 있다

```
64 bytes from 10.6.1.1: icmp_seq=989 ttl=64 time=17.6 ms
64 bytes from 10.6.1.1: icmp_seq=990 ttl=64 time=9.56 ms
64 bytes from 10.6.1.1: icmp_seq=991 ttl=64 time=21.1 ms
64 bytes from 10.6.1.1: icmp_seq=992 ttl=64 time=11.4 ms
64 bytes from 10.6.1.1: icmp_seq=993 ttl=64 time=13.9 ms
64 bytes from 10.6.1.1: icmp_seq=994 ttl=64 time=9.83 ms
64 bytes from 10.6.1.1: icmp_seq=995 ttl=64 time=9.09 ms
64 bytes from 10.6.1.1: icmp_seq=996 ttl=64 time=12.1 ms
64 bytes from 10.6.1.1: icmp_seq=997 ttl=64 time=9.21 ms
64 bytes from 10.6.1.1: icmp_seq=998 ttl=64 time=12.7 ms
64 bytes from 10.6.1.1: icmp_seq=999 ttl=64 time=10.9 ms
64 bytes from 10.6.1.1: icmp_seq=1000 ttl=64 time=11.6 ms

--- 10.6.1.1 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time 12382ms
rtt min/avg/max/mdev = 8.496/13.001/159.862/6.352 ms, pipe 3
mininet>
```

Experiment

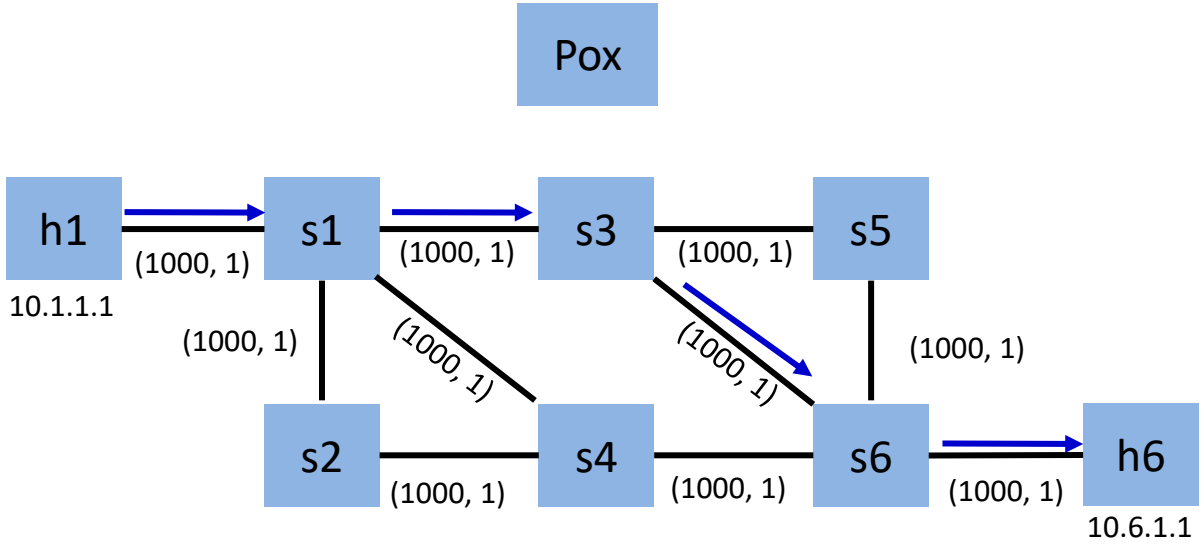
(각 링크의 BW(Mbps), 각 링크의 delay(ms))

Proactive란?
OpenFlow Switch에 Packet 도착 전에 SDN controller 가 미리 해당 Packet의 Flow를 정해 전달해주는 방식

■ Experiment 3: SDN-based Dynamic Routing

– 실험 3 : Proactive 방식의 Dynamic Routing 구현하기

Proactive
- Pox controller에서 discovery.py, topo_proactive.py 실행
  - discovery.py를 활용해 주기적으로 현재 topology와 Node들의 정보 확인 가능
  - topo_proactive.py를 활용해 Floyd-warshall 알고리즘 기반의 가장 간단한 Shortest path routing 구현 가능
    - » ./pox/pox.py forwarding.topo_proactive openflow.discovery
- Mininet을 활용하여 Week7 topology 구성
  - » sudo mn --custom ~/mininet/custom/Week7.py --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc
- 각 OpenFlow Switch 들의 Flow Table분석해보기
  - » dpctl dump-flows
- h1 과 h6 사이 ping test를 통해 Latency 측정
  - 0.01초의 인터벌로 1000개 보내기
    - » h1 ping -c1000 -i0.01 h6



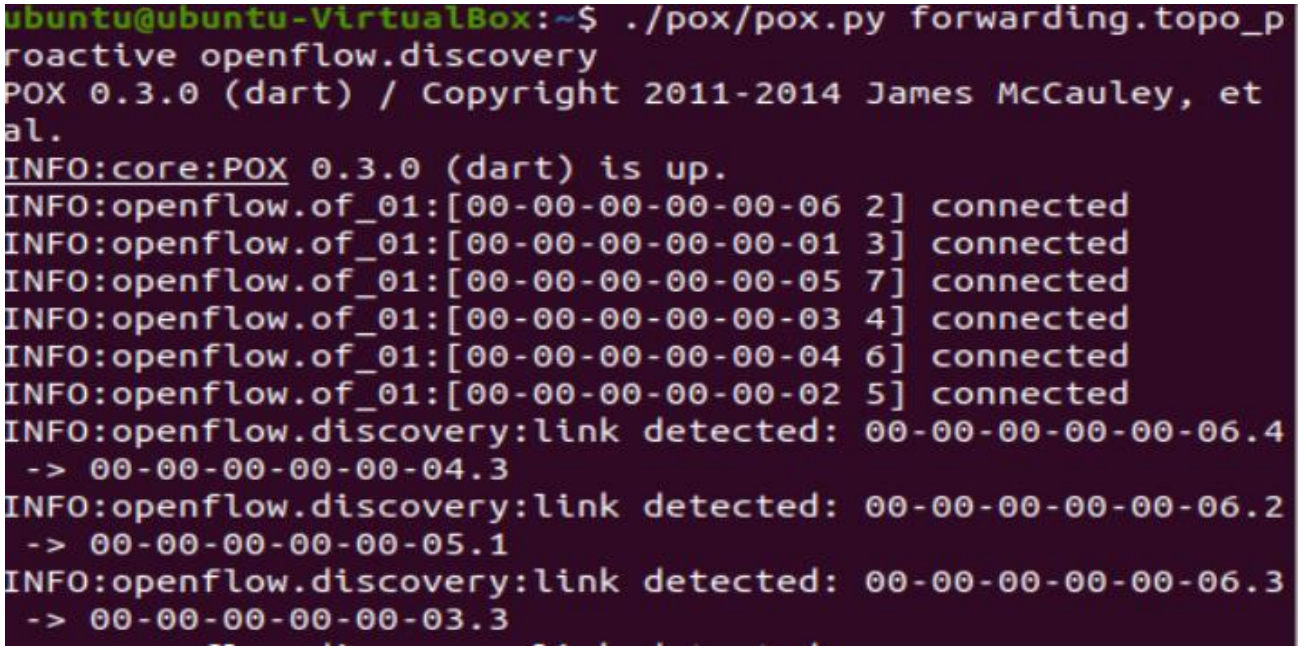
Experiment

■ Experiment 3: SDN-based Dynamic Routing

– 실험 3 : Proactive 방식의 Dynamic Routing 구현하기

- Pox controller에서 discovery.py, topo_proactive.py 실행

Proactive
- – discovery.py를 활용해 주기적으로 현재 topology와 Node들의 정보 확인 가능
- – topo_proactive.py를 활용해 Floyd-warshall 알고리즘 기반의 가장 간단한 Shortest path routing 구현 가능



Experiment

- **Experiment 3: SDN-based Dynamic Routing**
  - 실험 3 : Proactive 방식의 Dynamic Routing 구현하기
    - Mininet을 활용하여 Week7 topology 구성
      - » sudo mn --custom ~/mininet/custom/Week7.py --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc
    - ~/mininet/custom 폴더에 Week7.py 파일을 추가해야 한다

```
ubuntu@ubuntu-VirtualBox:~$ sudo mn --custom ~/mininet/c
ustom/Week7.py --topo=mytopo,6 --mac --switch=ovsk --con
troller=remote --link tc
[sudo] ubuntu의 암호:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:665
3
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h6
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, h1)
 (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, s2
) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, s
3) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1,
s4) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s2,
 s4) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s3
, s5) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s
3, s6) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (
s4, s6) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay)
(s5, s6) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay)
 (s6, h6)
```

Experiment

- **Experiment 3: SDN-based Dynamic Routing**
  - 실험 3 : Proactive 방식의 Dynamic Routing 구현하기
    - 각 OpenFlow Switch 들의 Flow Table분석해보기

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------
----------------------
 cookie=0x0, duration=32.090s, table=0, n_packets=17, n_
bytes=697, priority=65000,dl_dst=01:23:20:00:00:01,dl_ty
pe=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=32.103s, table=0, n_packets=0, n_b
ytes=0, udp,tp_src=68,tp_dst=67 actions=CONTROLLER:65535
 cookie=0x0, duration=32.090s, table=0, n_packets=0, n_b
ytes=0, ip,nw_dst=10.2.0.0/16 actions=output:"s1-eth2"
 cookie=0x0, duration=32.090s, table=0, n_packets=0, n_b
ytes=0, ip,nw_dst=10.3.0.0/16 actions=output:"s1-eth3"
 cookie=0x0, duration=32.090s, table=0, n_packets=0, n_b
ytes=0, ip,nw_dst=10.4.0.0/16 actions=output:"s1-eth4"
 cookie=0x0, duration=32.090s, table=0, n_packets=0, n_b
ytes=0, ip,nw_dst=10.5.0.0/16 actions=output:"s1-eth3"
 cookie=0x0, duration=32.090s, table=0, n_packets=0, n_b
ytes=0, ip,nw_dst=10.6.0.0/16 actions=output:"s1-eth3"
 cookie=0x0, duration=32.073s, table=0, n_packets=0, n_b
ytes=0, priority=32767,ip,nw_dst=10.1.4.0/24 actions=CON
TROLLER:65535
 cookie=0x0, duration=32.046s, table=0, n_packets=0, n_b
ytes=0, priority=32767,ip,nw_dst=10.1.1.0/24 actions=CON
TROLLER:65535
 cookie=0x0, duration=32.046s, table=0, n_packets=0, n_b
ytes=0, priority=32767,ip,nw_dst=10.1.2.0/24 actions=CON
TROLLER:65535
 cookie=0x0, duration=32.046s, table=0, n_packets=0, n_b
ytes=0, priority=32767,ip,nw_dst=10.1.3.0/24 actions=CON
TROLLER:65535
```

Experiment

- **Experiment 3: SDN-based Dynamic Routing**
  - 실험 3 : Proactive 방식의 Dynamic Routing 구현하기
    - h1과 h6 사이 ping test를 통해 Latency 측정
      - 0.01초의 인터벌로 1000개 보내기
        - » h1 ping -c1000 -i0.01 h6
    - 실험 2-2의 Routing 경로와 같아 비슷한 RTT 값을 갖는다.

```
mininet> h1 ping -c1000 -i0.01 h6
PING 10.6.1.1 (10.6.1.1) 56(84) bytes of data.
64 bytes from 10.6.1.1: icmp_seq=4 ttl=64 time=45.0 ms
64 bytes from 10.6.1.1: icmp_seq=5 ttl=64 time=27.3 ms
64 bytes from 10.6.1.1: icmp_seq=6 ttl=64 time=14.6 ms
64 bytes from 10.6.1.1: icmp_seq=7 ttl=64 time=15.8 ms
64 bytes from 10.6.1.1: icmp_seq=8 ttl=64 time=13.8 ms
64 bytes from 10.6.1.1: icmp_seq=9 ttl=64 time=10.1 ms
64 bytes from 10.6.1.1: icmp_seq=10 ttl=64 time=12.0 ms
64 bytes from 10.6.1.1: icmp_seq=11 ttl=64 time=15.6 ms
64 bytes from 10.6.1.1: icmp_seq=12 ttl=64 time=10.4 ms
64 bytes from 10.6.1.1: icmp_seq=13 ttl=64 time=10.8 ms
64 bytes from 10.6.1.1: icmp_seq=14 ttl=64 time=10.5 ms
64 bytes from 10.6.1.1: icmp_seq=15 ttl=64 time=12.5 ms
64 bytes from 10.6.1.1: icmp_seq=16 ttl=64 time=10.7 ms
64 bytes from 10.6.1.1: icmp_seq=17 ttl=64 time=11.6 ms
64 bytes from 10.6.1.1: icmp_seq=18 ttl=64 time=11.7 ms
64 bytes from 10.6.1.1: icmp_seq=19 ttl=64 time=10.8 ms
64 bytes from 10.6.1.1: icmp_seq=20 ttl=64 time=11.6 ms
64 bytes from 10.6.1.1: icmp_seq=21 ttl=64 time=11.2 ms
64 bytes from 10.6.1.1: icmp_seq=22 ttl=64 time=11.7 ms
64 bytes from 10.6.1.1: icmp_seq=23 ttl=64 time=10.5 ms
64 bytes from 10.6.1.1: icmp_seq=24 ttl=64 time=13.5 ms
64 bytes from 10.6.1.1: icmp_seq=25 ttl=64 time=10.9 ms
```

Extra Experiment

Extra Experiment

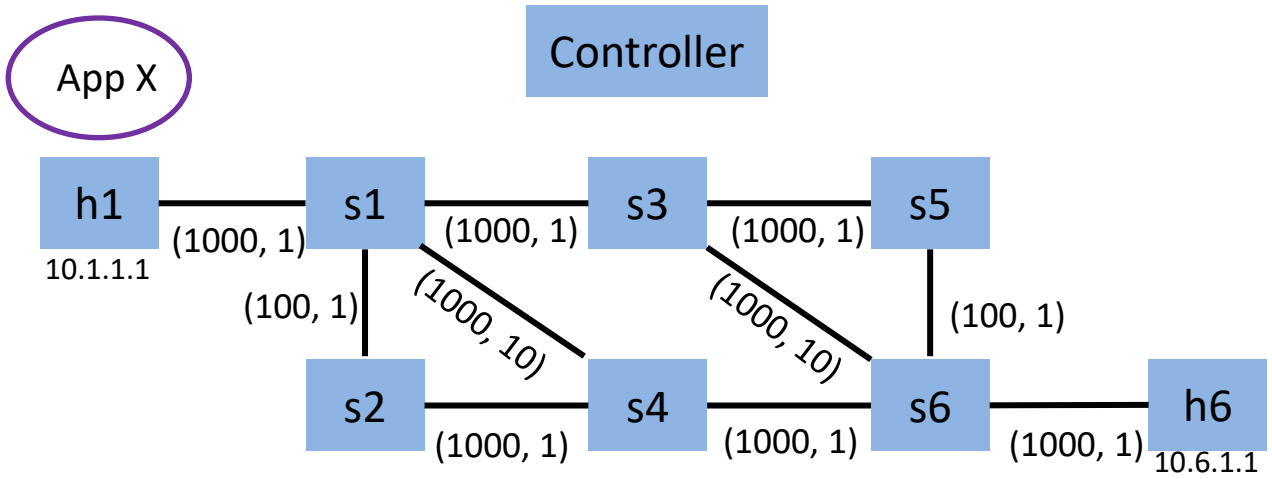
■ **Extra Experiment: SDN-based Application-aware Routing**

(각 링크의 BW(Mbps), 각 링크의 delay(ms))

App X Requirement
Latency (< 12ms)
Bandwidth (>10Mbps)

– 추가 실험 1: Application X의 Requirement에 기반한 Static routing 구현하기

- Mininet을 활용하여 Week8 토폴로지 구성
  - » sudo mn --custom ~/mininet/custom/Week8.py --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc
- App X의 Requirement를 만족시킬 수 있는 경로 생각해보기
- 해당 경로로 실험 2의 Static routing을 활용한 경로 설정해보기
  - » sh ovs-ofctl del-flow sN
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?
- Ping을 통해 Latency requirement 만족하는지 확인
  - Latency ≈ 0.5 × Round Trip Time 으로 가정
  - Propagation delay 이외의 delay들의 합 ≈ 0.4ms
  - 0.01초의 인터벌로 1000개 보내기
    - » h1 ping -c1000 -i0.01 h6
- iperf 통해 BW requirement 만족하는지 확인
  - BW ≈ 0.5 × Bottleneck Link Bandwidth
    - » xterm h1 h6
  - "Node: h1"
    - » iperf –s
  - "Node: h6"
    - » iperf –c 10.1.1.1



Extra Experiment

- **Extra Experiment: SDN-based Application-aware Routing**
  - 추가 실험 1: Application X의 Requirement에 기반한 Static routing 구현하기
    - Mininet을 활용하여 Week8 토폴로지 구성
      - » sudo mn --custom Week8.py --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc

```
ubuntu@ubuntu-VirtualBox:~$ sudo mn --custom ~/mininet/custom/Week8.py --topo=mytopo,6
sk --controller=remote --link tc
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h6
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, h1) (100.00Mbit 1ms delay) (100.0
s1, s2) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, s3) (1000.00Mbit 10ms dela
0ms delay) (s1, s4) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s2, s4) (1000.00M
00.00Mbit 1ms delay) (s3, s5) (1000.00Mbit 10ms delay) (1000.00Mbit 10ms delay) (s3, s
s delay) (1000.00Mbit 1ms delay) (s4, s6) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay)
0Mbit 1ms delay) (1000.00Mbit 1ms delay) (s6, h6)
*** Configuring hosts
h1 h6
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...(1000.00Mbit 1ms delay) (100.00Mbit 1ms delay) (1000.00Mbit 1ms de
 10ms delay) (100.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (10
ay) (1000.00Mbit 10ms delay) (1000.00Mbit 10ms delay) (1000.00Mbit 1ms delay) (1000.00M
000.00Mbit 1ms delay) (100.00Mbit 1ms delay) (1000.00Mbit 10ms delay) (1000.00Mbit 1ms
t 1ms delay) (1000.00Mbit 1ms delay)
*** Starting CLI:
```

Extra Experiment

- **Extra Experiment: SDN-based Application-aware Routing**
  - 추가 실험 1: Application X의 Requirement에 기반한 Static routing 구현하기
    - App X의 Requirement를 만족시킬 수 있는 경로 생각해보기

Extra Experiment

- **Extra Experiment: SDN-based Application-aware Routing**
  - 추가 실험 1: Application X의 Requirement에 기반한 Static routing 구현하기
    - 해당 경로로 실험 2의 Static routing을 활용한 경로 설정해보기
      - » sh ovs-ofctl del-flow sN
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?

Extra Experiment

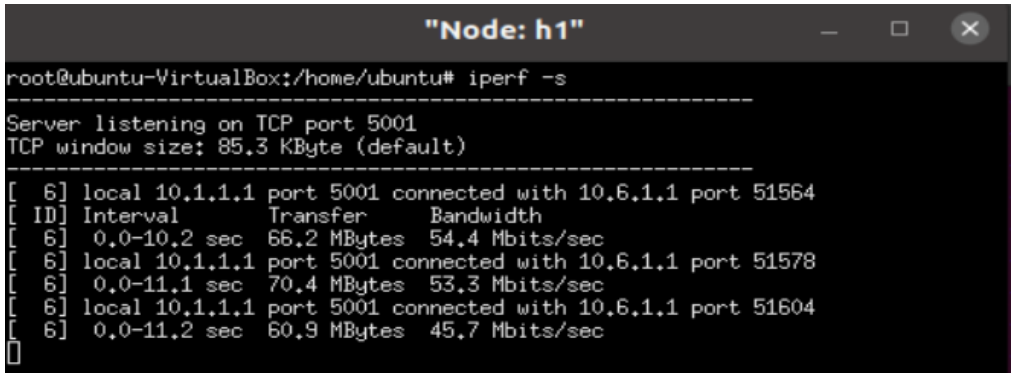
- **Extra Experiment: SDN-based Application-aware Routing**
  - 추가 실험 1: Application X의 Requirement에 기반한 Static routing 구현하기
    - Ping을 통해 Latency requirement 만족하는지 확인
      - Latency ≈ 0.5 × Round Trip Time 으로 가정
      - Propagation delay 이외의 delay들의 합 ≈ 0.4ms
      - 0.01초의 인터벌로 1000개 보내기
        - » h1 ping -c1000 -i0.01 h6

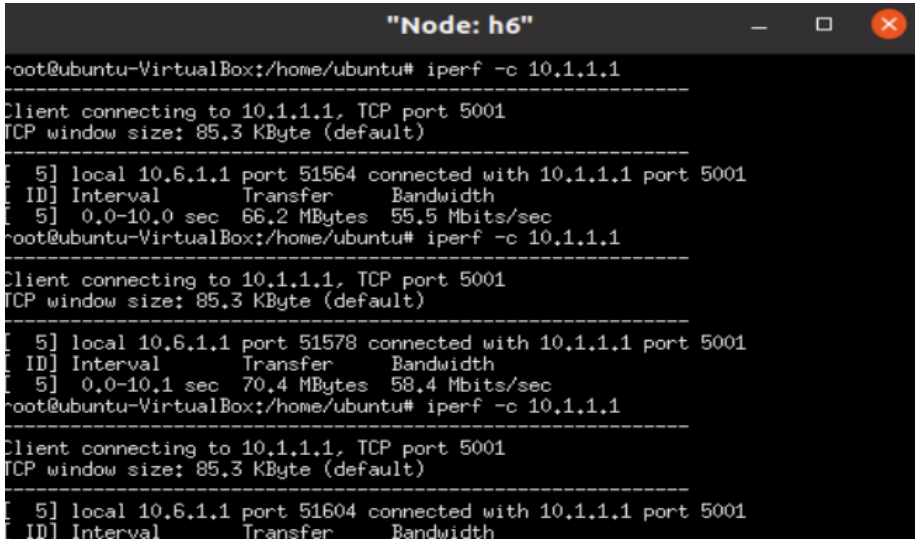
```
64 bytes from 10.6.1.1: icmp_seq=985 ttl=64 time=13.9 ms
64 bytes from 10.6.1.1: icmp_seq=986 ttl=64 time=14.4 ms
64 bytes from 10.6.1.1: icmp_seq=987 ttl=64 time=13.0 ms
64 bytes from 10.6.1.1: icmp_seq=988 ttl=64 time=13.5 ms
64 bytes from 10.6.1.1: icmp_seq=989 ttl=64 time=19.3 ms
64 bytes from 10.6.1.1: icmp_seq=990 ttl=64 time=15.0 ms
64 bytes from 10.6.1.1: icmp_seq=991 ttl=64 time=13.6 ms
64 bytes from 10.6.1.1: icmp_seq=992 ttl=64 time=11.0 ms
64 bytes from 10.6.1.1: icmp_seq=993 ttl=64 time=15.5 ms
64 bytes from 10.6.1.1: icmp_seq=994 ttl=64 time=12.4 ms
64 bytes from 10.6.1.1: icmp_seq=995 ttl=64 time=18.5 ms
64 bytes from 10.6.1.1: icmp_seq=996 ttl=64 time=14.4 ms
64 bytes from 10.6.1.1: icmp_seq=997 ttl=64 time=13.6 ms
64 bytes from 10.6.1.1: icmp_seq=998 ttl=64 time=12.4 ms
64 bytes from 10.6.1.1: icmp_seq=999 ttl=64 time=16.0 ms
64 bytes from 10.6.1.1: icmp_seq=1000 ttl=64 time=12.7 ms

--- 10.6.1.1 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time 13928ms
rtt min/avg/max/mdev = 10.689/15.834/48.830/4.805 ms, pipe 3
mininet>
```

Extra Experiment

- **Extra Experiment: SDN-based Application-aware Routing**
  - 추가 실험 1: Application X의 Requirement에 기반한 Static routing 구현하기
    - iperf 통해 BW requirement 만족하는지 확인
      - BW ≈ 0.5 × Bottleneck Link Bandwitdh
        - » xterm h1 h6
      - "Node: h1"
        - » iperf –s
      - "Node: h6"
        - » iperf –c 10.1.1.1





Extra Experiment

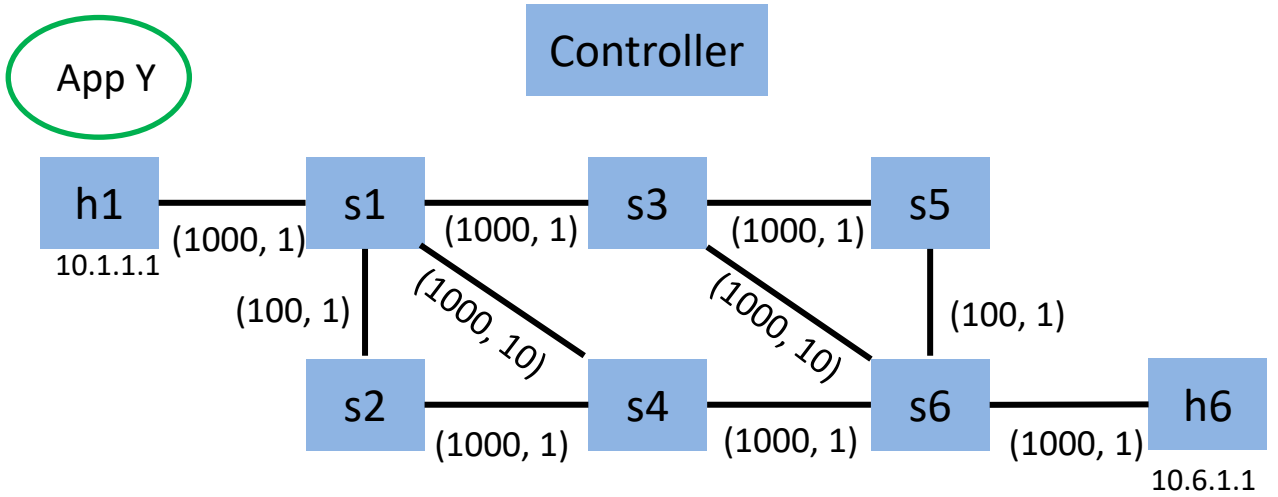
■ **Extra Experiment : SDN-based Application-aware Routing**

– 추가 실험 2 : Application Y의 Requirement에 기반한 Static routing 구현하기

(각 링크의 BW(Mbps), 각 링크의 delay(ms))

App Y Requirement
Latency (< 100ms)
Bandwidth (>200Mbps)

- Mininet을 활용하여 Week8 토폴로지 구성
  - » sudo mn --custom Week8.py --topo=mytopo,6 --mac --switch=ovsk --controller=remote --link tc
- App X의 Requirement를 만족시킬 수 있는 경로 생각해보기
- 해당 경로로 실험 2의 Static routing을 활용한 경로 설정해보기
  - » sh ovs-ofctl del-flow sN
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
  - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?
- Ping을 통해 Latency requirement 만족하는지 확인
  - Latency ≈ 0.5 × Round Trip Time 으로 가정
  - Propagation delay 이외의 delay들의 합 ≈ 0.4ms
  - 0.01초의 인터벌로 1000개 보내기
    - » h1 ping -c1000 -i0.01 h6
- iperf 통해 BW requirement 만족하는지 확인
  - BW ≈ 0.5 × Bottleneck Link Bandwidth
    - » xterm h1 h6
  - "Node: h1"
    - » iperf –s
  - "Node: h6"
    - » iperf –c 10.1.1.1



Extra Experiment

- **Extra Experiment : SDN-based Application-aware Routing**
  - 추가 실험 2 : Application Y의 Requirement에 기반한 Static routing 구현하기
    - Mininet을 활용하여 Week8 토폴로지 구성

```
ubuntu@ubuntu-VirtualBox:~$ sudo mn --custom ~/mininet/custom/Week8.py --topo=mytopo,6
sk --controller=remote --link tc
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h6
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, h1) (100.00Mbit 1ms delay) (100.00
s1, s2) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s1, s3) (1000.00Mbit 10ms dela
0ms delay) (s1, s4) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (s2, s4) (1000.00Mb
00.00Mbit 1ms delay) (s3, s5) (1000.00Mbit 10ms delay) (1000.00Mbit 10ms delay) (s3, s6
s delay) (1000.00Mbit 1ms delay) (s4, s6) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay)
0Mbit 1ms delay) (1000.00Mbit 1ms delay) (s6, h6)
*** Configuring hosts
h1 h6
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...(1000.00Mbit 1ms delay) (100.00Mbit 1ms delay) (1000.00Mbit 1ms de
 10ms delay) (100.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (10
ay) (1000.00Mbit 10ms delay) (1000.00Mbit 10ms delay) (1000.00Mbit 1ms delay) (1000.00M
000.00Mbit 1ms delay) (100.00Mbit 1ms delay) (1000.00Mbit 10ms delay) (1000.00Mbit 1ms
t 1ms delay) (1000.00Mbit 1ms delay)
*** Starting CLI:
```

Extra Experiment

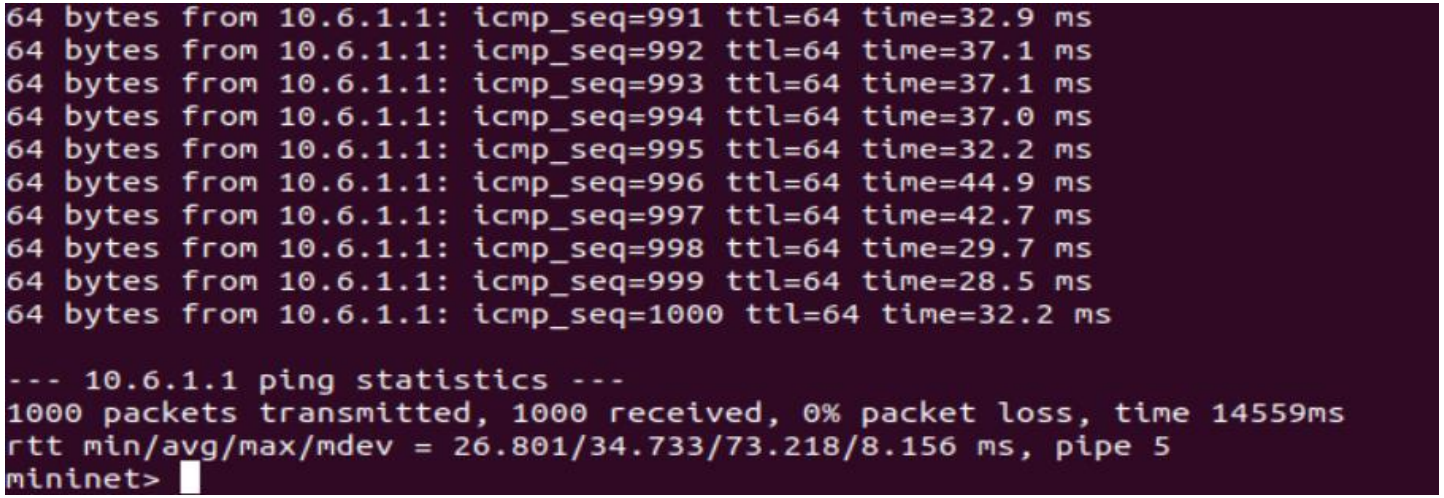
- **Extra Experiment : SDN-based Application-aware Routing**
  - 추가 실험 2 : Application Y의 Requirement에 기반한 Static routing 구현하기
    - App Y의 Requirement를 만족시킬 수 있는 경로 생각해보기

Extra Experiment

- **Extra Experiment : SDN-based Application-aware Routing**
  - 추가 실험 2 : Application Y의 Requirement에 기반한 Static routing 구현하기
    - 해당 경로로 실험 2의 Static routing을 활용한 경로 설정해보기
      - » sh ovs-ofctl del-flow sN
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.6.1.1,actions=output=?
      - » sh ovs-ofctl add-flow sN in_port=?,nw_dst=10.1.1.1,actions=output=?

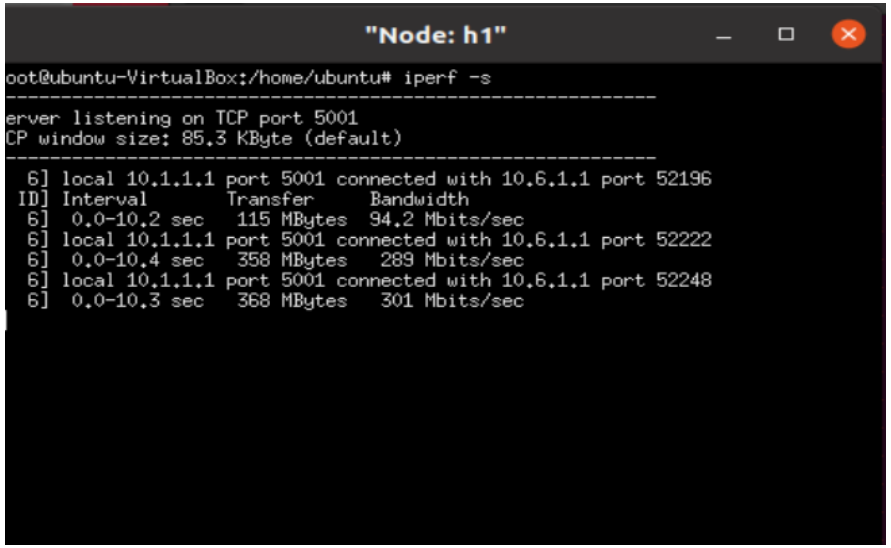
Extra Experiment

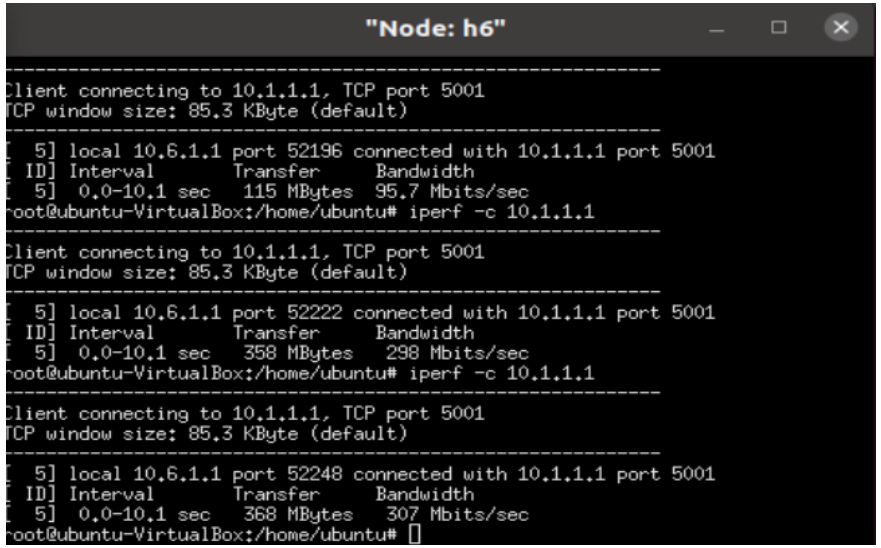
- **Extra Experiment : SDN-based Application-aware Routing**
  - 추가 실험 2 : Application Y의 Requirement에 기반한 Static routing 구현하기
    - Ping을 통해 Latency requirement 만족하는지 확인
      - Latency ≈ 0.5 × Round Trip Time 으로 가정
      - Propagation delay 이외의 delay들의 합 ≈ 0.4ms
      - 0.01초의 인터벌로 1000개 보내기
        - » h1 ping -c1000 -i0.01 h6



Extra Experiment

- **Extra Experiment : SDN-based Application-aware Routing**
  - 추가 실험 2 : Application Y의 Requirement에 기반한 Static routing 구현하기
    - iperf 통해 BW requirement 만족하는지 확인
      - BW ≈ 0.5 × Bottleneck Link Bandwitdh
        - » xterm h1 h6
      - "Node: h1"
        - » iperf –s
      - "Node: h6"
        - » iperf –c 10.1.1.1





Result Report

Result Report

- **Experiment 1**
  - 실험 1-1 : Ping Test 화면을 캡처해 결과보고서에 첨부
  - 실험 1-2 : dump-ports-desc 결과, Ping Test 화면을 캡처해 결과보고서에 첨부

- **Experiment 2**
  - 실험 2-1 : Flow Table, Ping Test 화면을 캡처해 결과보고서에 첨부
  - 실험 2-2 : Flow Table, Ping Test 화면을 캡처해 결과보고서에 첨부

- **Experiment 3**
  - 실험 3 : Flow Table, Ping Test 화면을 캡처해 결과보고서에 첨부

- **Extra Experiment**
  - 추가 실험 1: Flow Table, Ping Test, iperf Test 화면을 캡처해 결과보고서에 첨부
  - 추가 실험 2: Flow Table, Ping Test, iperf Test 화면을 캡처해 결과보고서에 첨부