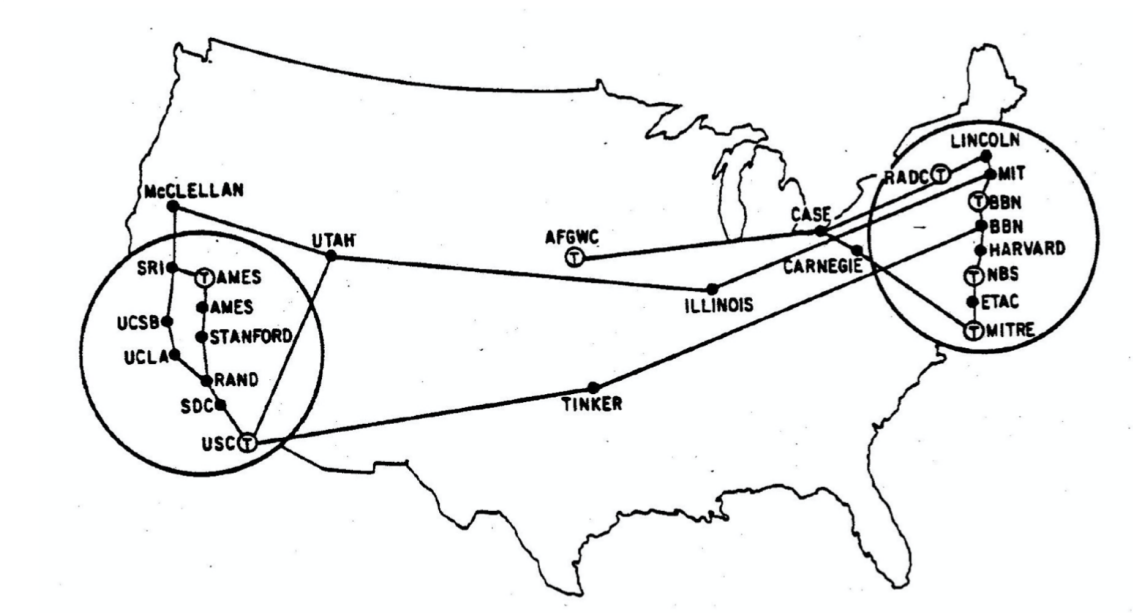
实验三指导书

实验目的

通过本次实验，希望大家掌握以下内容：

- 学习利用`OpenFlow Message`请求交换机端口状态，并计算端口**工作负载。**
- 理解瓶颈链路的概念，利用`networkx`函数库或自行实现其他算法，计算**带宽最佳路径。**
- 理解并实现网络安全需求中的强制路径点(`Waypoint-enforcement`)策略。

问题背景



在过去的工作中，你已经学会利用`Ryu app`设计自学习交换机并克服环路广播问题，实现网络节点之间初步的连通性；也掌握了如何在链路可能发生故障的情况下计算最小时延路径。

时间来到1972年，`ARPANET`的规模达到25个结点，用户数量和用户在网络中传输的数据量激增，作为`ARPANET`建设者的你面临的挑战主要来自于如何提升服务质量和客户满意度。用户的流量具有突发性和不确定性，如何规划网络流量使得网络中各方都能最大可能的利用网络现有资源，是你在本次实验中的任务。

在实验任务一中，你将学习计算交换机端口的工作负载。在实验任务二中，你将实现最佳带宽路径的计算。在附加题中，你可以尝试实现网络中重要的`Waypoint-enforcement`策略，以满足网络安全需求。

实验任务一：测量端口负载

本任务中，利用交换机端口被占用的带宽（即用于转发数据的带宽）表征端口工作负载。请根据以下提示，利用`Ryu`控制器周期性地测量并输出**各交换机端口的当前工作负载**，拓扑请采用`topo_1972.py`，启动拓扑如果报错，可以尝试`mn -c`。

1.端口负载测量原理

- 请求端口状态

控制器向交换机发送`OFPMP_PORT_STATS`消息，请求端口状态信息。

```
req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
datapath.send_msg(req)
```

- 捕捉`Reply`报文

如果交换机对`OFPMP_PORT_STATS`消息进行回复，控制器可以对`Reply`报文进行捕捉。

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
```

- 计算端口负载

交换机的`Reply`报文中包含该交换机每个端口的状态，主要关注每个`port`的以下4个属性：`tx_bytes`、`rx_bytes`、`duration_sec`、`duration_nsec`。

`tx_bytes`和`rx_bytes`分别是当前`port`总共发出和接收的字节数，`duration`是一个计时器，由`sec`和`nsec`组成，`nsec`向`sec`进位。

```
uint64_t rx_bytes;       /* Number of received bytes. */
uint64_t tx_bytes;       /* Number of transmitted bytes. */

uint32_t duration_sec;   /* Time port has been alive in seconds. */
uint32_t duration_nsec;  /* Time port has been alive in nanoseconds beyond
                            duration_sec. */
```

`tx_bytes`和`rx_bytes`之和是端口总共收发的字节数。

对比同一交换机两次相邻的`Reply`报文，端口总共收发的字节数之差是两次`Reply`报文之间端口传输的总数据量，`duration`之差是时间间隔，二者相除就是端口正在用于传输的带宽，即端口负载。

2.实现周期性测量

开启一个线程专门负责测量工作负载，利用`sleep`函数让其每隔一段时间进行一次测量。

```
self.workload_thread = hub.spawn(self._count_workload)
```

3.代码框架

- 代码框架参考附件`work_load.py`，已经实现了周期性地向每个交换机发送`OFPMP_PORT_STATS`消息的代码：

```
def _count_workload(self):
    while True:
        for dp in self.datapaths.values():
            self._send_request(dp)
        self.get_topology(None)
        hub.sleep(4)

def _send_request(self,datapath):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
    datapath.send_msg(req)
```

- 你的主要任务是在`_port_stats_reply_handler`函数中填充代码，可以根据需要自行创建新函数。去掉以下的注释，可以打印`Reply`报文的内容，这将对你有所帮助。

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
    body = ev.msg.body
    dpid = ev.msg.datapath.id
    self.workload.setdefault(dpid, {})
    #you need to code here to finish mission1
    #of course, you can define new function as you wish
    #for stat in sorted(body, key=attrgetter('port_no')):
        #port_no = stat.port_no
        #if port_no != ofproto_v1_3.OFPP_LOCAL:
            #key = (dpid, port_no)
            #value = (stat.tx_bytes, stat.rx_bytes,
                     #stat.duration_sec, stat.duration_nsec)
            #print(key,end=':')
            #print(value)
```

- 别忘记编写函数输出你测量到的端口负载（单位：`Mbit/s`）。
- 本任务利用`iperf`向网络施加负载，需要首先实现各结点之间的联通，框架中已经实现了感知拓扑、下发(hop)最短路的功能，你只需要将实验一任务二中处理环路的代码填充到以下位置即可。

```
##############################deal with loop#############################
    def handle_arp(self,msg):
    #just your code in exp1 mission2
```

4.测量结果验证

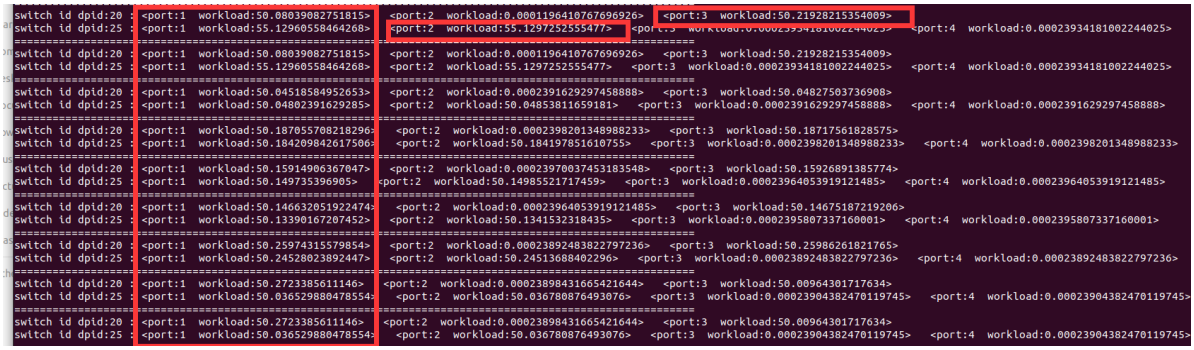
- 初始情况

不向网络下达任何指令时，每个端口的负载非常小，近乎为0`Mbit/s`。

- 施加工作负载

`iperf`工具用于测量两点之间的带宽，它会发送大量的数据，增加链路工作负载。下图中，`iperf`占用的带宽与`Ryu app`测量出的链路负载基本吻合。

如下图的示例是在`MIT`（dpid 25）和`UCLA`（dpid 20）之间施加大小为50`Mbit/s`的流量，为了方便观察，只输出`MIT`和`UCLA`的工作负载，可以观察到测量结果和发送的流量大小基本一致。



观察上述现象的命令为：

```
mininet>xterm MIT
iperf-s

mininet>xterm UCLA
iperf -c 10.0.0.12 -b 50m -t 300
```

上述代码表示，将`MIT`作为`iperf`的服务端打开。在`UCLA`端，向`10.0.0.12(MIT)`发送`50Mbps`，持续300s的`tcp`报文。

为什么要持续300s？因为`iperf`需要一小段时间才能达到稳定状态，将持续时间设长更容易测量到稳定的负载。

- 别忘记`--observe-links`！！！

实验任务二：最佳带宽路径

用户在网络中传输数据时，希望获得较高的网速。传输数据包时所选路径的可用带宽，很大程度上决定了用户获得的网速，影响着用户的使用体验。从源主机到目的主机之间经常会有很多条可选路径，请设计方案计算并下发一条**瓶颈链路可用带宽最大的路径**。

1.计算链路可用带宽

链路可用带宽 = 链路总带宽 - 链路负载。

- 链路总带宽

为了降低实验难度，交换机之间链路的总带宽都预设为`1000Mbps`，假设主机和交换机之间链路带宽无限大。

- 用端口负载估算链路负载

交换机和交换机之间的链路关联着两个交换机端口，取两个端口负载最大者为链路负载。

主机和交换机之间的链路关联一个主机端口和一个交换机端口，以交换机端口的负载为链路负载。

2.带宽最佳路径

- 瓶颈链路

一条**路径**包含多条**链路**，路径中可用带宽最小的链路称为瓶颈链路，限制着整条路径的传输能力。

- 最佳带宽路径

本次任务中，选择源主机和目的主机之间的所有路径中，瓶颈链路可用带宽最高的路径作为最佳带宽路径。

- 获取两点之间所有路径

可以利用`networkx`的`shortest_simple_paths`函数计算，也可以自行实现。

- 流表软超时

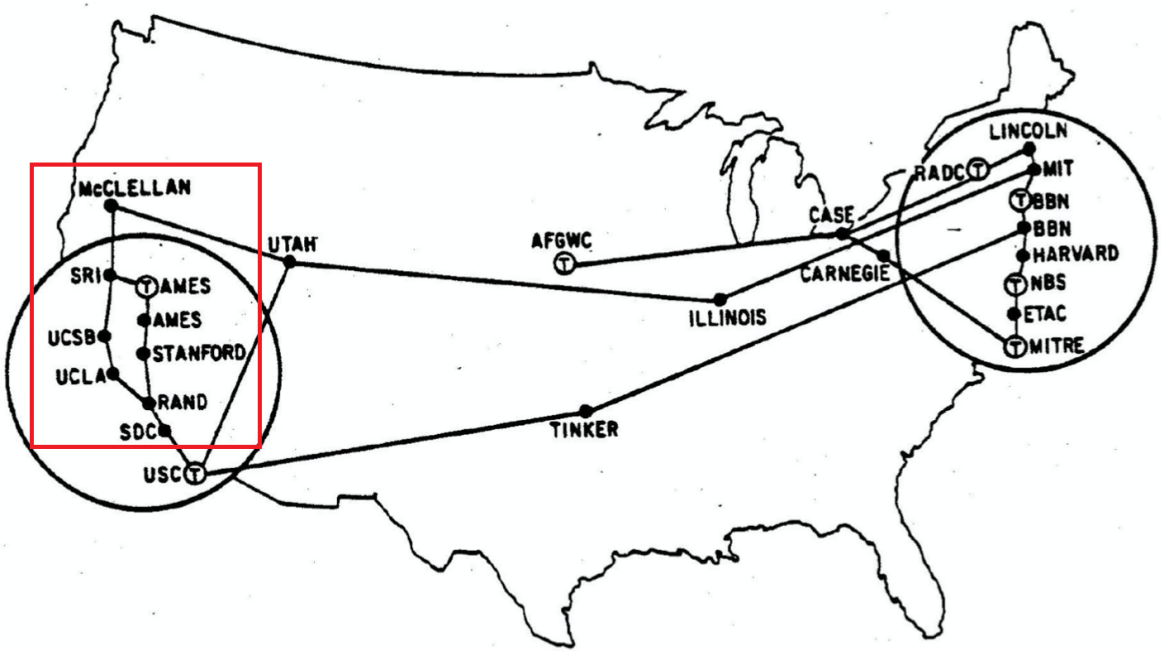
为了使同一个数据流在传输时保持同一条路径，不同数据流之间根据可用带宽灵活选择最佳路径，下发流表时注意合理运用流表软超时。

3.代码框架

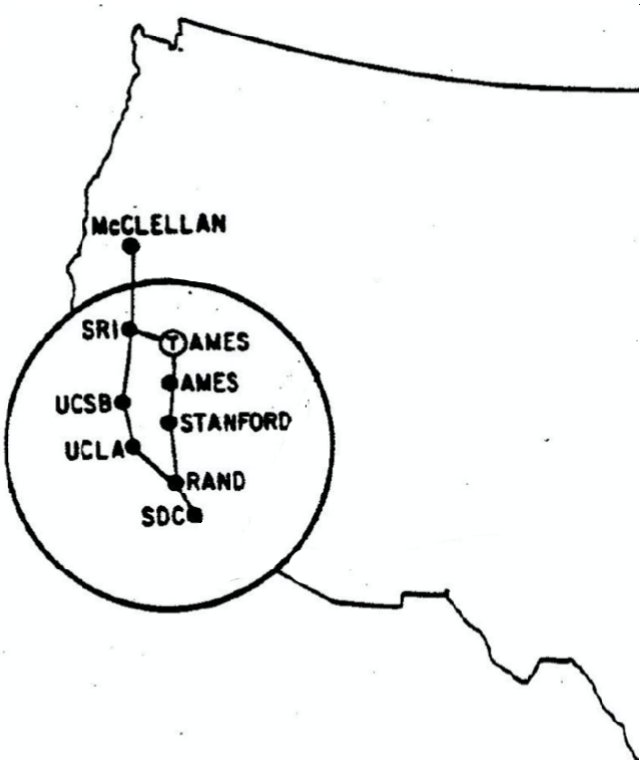
- 直接在任务一的基础上完成即可。
- 当然，任务二将实现新的路由策略，你需要将原本框架中下发最小跳数路径的代码替换或加以修改，实现你新设计的策略。

4.结果示例

本任务主要关注红色方框中的结点，如下图：



拓扑文件为`topo_1972_bandwidth.py`，拓扑图如下：



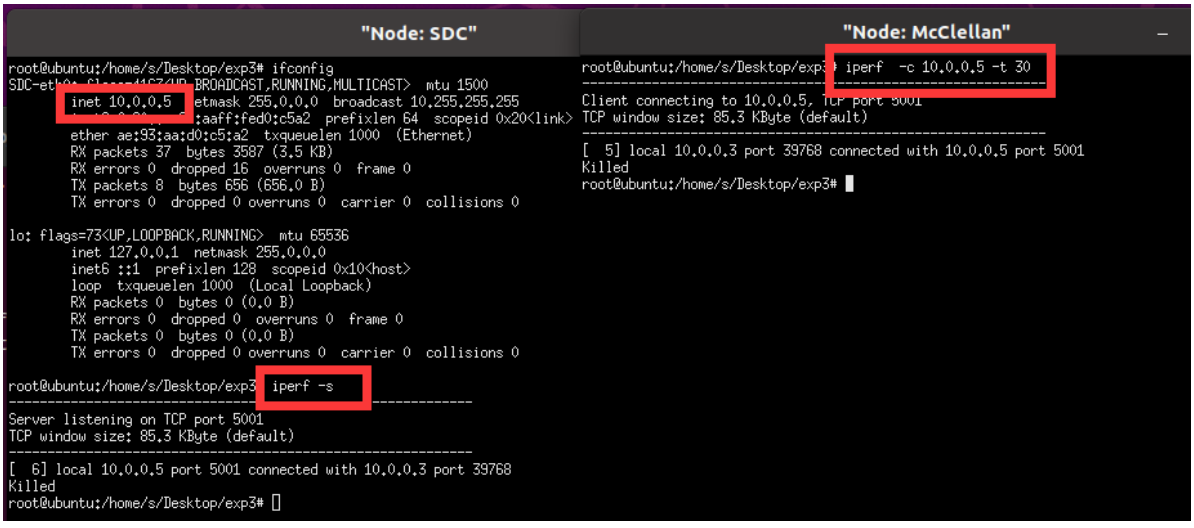
一个简单的例子：

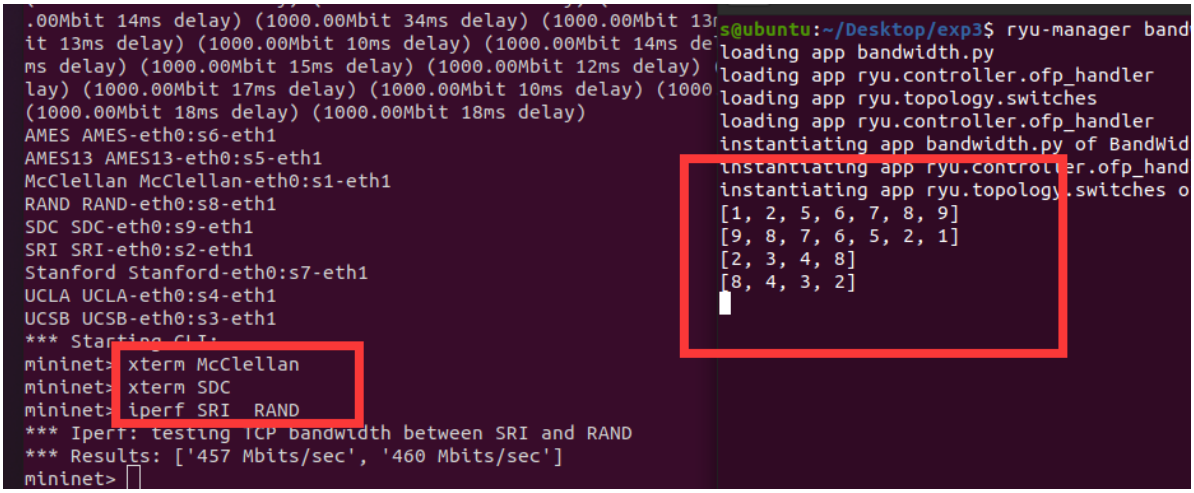
在`xterm`中令`SDC`对`McClellan`进行`iperf`，控制器选择了`SDC RAND Stanford AMES AMES13 SRI McClellan`这条路径。

在`mininet`中`iperf SRI RAND`时，由于另一条路正承载着`SDC`到`McClellan`之间的工作负载，控制器下发了`SRI UCSB UCLA RAND`这条当前可用带宽最高的路。

也可以利用`iperf`占用某条路的带宽，然后用`ping`检查控制器是否会选择到另一条路。

你的运行结果不必与这个例子完全一致，合理即可。





附加题

国防部对`ARPANET`网络流量的安全性非常看重，军方在`TINKER`设立了流量检查点用于分析经过的数据，要求`ILLINOIS`到`UTAH`之间的所有流量都必须经过`TINKER`。请你下发满足以下条件的路径：

- 满足路径点(`Waypoint-enforcement`)策略，`ILLINOIS`到`UTAH`之间的所有流量都必须经过`TINKER`。
- 一端主机发出的数据包，在到达`Waypoint`之前不能途径另一端主机直连的交换机，例如`ILLINOIS UTAH USC TINKER USC UTAH`。
- 同时满足上述两个条件的路径中跳数最少的一条。