实验二 传输层 TCP 协议实验

1. 实验目的

（1）理解和掌握 TCP 连接管理中“三次握手”建立连接和拆除连接的过程；
（2）理解和掌握 TCP 可靠数据传输的实现原理和方法；
（3）理解和掌握 TCP 流量控制的实现原理和方法；
（4）理解和掌握 TCP 拥塞控制的实现原理和方法；
（5）学习和掌握通过编程和抓包分析工具验证和分析协议运行过程。

2. 实验内容

（1）理解实验原理，熟悉 Mininet 网络仿真工具和 Wireshark 抓包分析工具；
（2）阅读和运行实验代码，理解和掌握创建网络拓扑、建立 TCP 连接的过程，并抓取 TCP 连接过程中的数据包；
（3）基于网络仿真工具和抓包分析工具，查找 TCP 连接管理、可靠传输相关的数据包，结合 TCP 协议原理进行验证和分析；
（4）基于网络仿真工具和抓包分析工具，测量 TCP 流量控制和拥塞控制中关键性能指标的变化情况，结合 TCP 协议原理进行验证和分析。

3. 实验原理

TCP 协议是传输层面向连接的可靠传输协议，其主要功能包括连接管理、可靠传输、流量控制和拥塞控制四部分。具体而言：

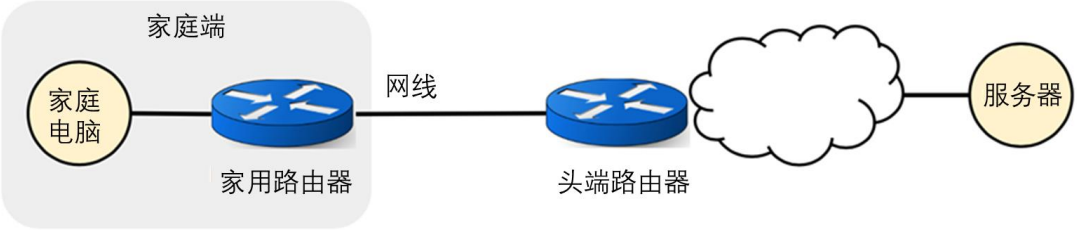
针对连接管理，包括（1）通过三次握手建立连接和设置必要参数的 TCP 连接建立过程；（2）通过四次关闭连接和确认的 TCP 连接终止过程。

针对可靠传输，发送端基于往返时间 RTT，通过（1）重传、快速重传和（2）差错恢复来处理报文段丢失的情况。

针对流量控制，接收端通过设置 TCP 分组报头中接收窗口（rwnd）参数告知发送端其缓存器的空闲空间大小，进而控制发送端的发送速率以避免接收端的缓存器的溢出，实现流量控制。

针对拥塞控制，通过拥塞窗口（cwnd）参数控制发送端向网络中发送流量的发送速率，以避免网络拥塞。TCP 处理拥塞的一般策略基于三个算法：慢启动、以加性增为规则的拥塞避免，以乘性减为规则的快速恢复。其中，慢启动措施是在 TCP 连接建立开始以慢速率发送，但以指数速度快速增加发送速率，直至丢包事件发生进入加性增乘性减阶段；加性增措施是在确认网络通畅时，缓慢/加性增加拥塞窗口大小（每次增加 1）保证发送方的可用带宽；乘性减措施是在检测到网络拥塞时，快速/乘性减小拥塞窗口大小（每次减少为之前的一半），降低发送速率，导致 TCP 连接的拥塞窗口大小随时间呈现锯齿状。

本次实验将以下面一个简单的“家庭电脑-路由器-互联网服务器”网络为实例，通过编程模拟 TCP 连接建立过程，针对以上四方面的 TCP 实现原理，结合抓取的数据包报文和性能指标变化情况进行协议验证和分析。



具体而言：

- 家庭电脑连接到家用路由器，一般具有较高的网络带宽；
- 家用路由器连接到由互联网服务提供商（ISP）运行的头端路由器，进一步连接到互联网服务器，一般具有较低的网络带宽；
- 通过调用网络仿真软件 Mininet 的 Python API，可以模拟搭建上述网络；
- 通过 Linux 系统自带的网络性能测试命令和工具（如 ping、iperf、tcpdump、tcpprobe）启动 TCP 流量并抓取传输的数据包；
- 通过 Wireshark 软件解析相关数据报文，观察网络性能指标变化，理解和验证 TCP 协议中连接管理、可靠传输、流量控制和拥塞控制四部分主要功能的实现。

相关测试命令在本实验中的作用和使用方法示例如下：

命令	作用	原理	使用方法示例
ping[1]	测试特定主机间的 IP 层连接	向目标主机传出一个 ICMP 的请求数据包，并等待接收回应数据包。按时间和成功响应的次数估算丢失数据包率和数据包往返时间	ping <IP address>
iperf[2]	测量 TCP 网络带宽	产生了两个主机间的 TCP 流量，主动测量 IP 网络上最大可实现带宽	iperf -c <server IP address> -t <time>
tcpdump[3]	抓包分析	根据需求对网络上的数据包进行截获，将网络中传送的数据包包头截获下来提供分析	tcp host <host IP address>
tcpprobe[4]	监测 TCP 流量	基于 TCP 实现机制，监听特定端口号捕捉 cwnd 和序列号等信息	cat /proc/net/tcpprobe

4. 实验环境和使用工具

4.1 实验环境

（1） 为方便实验操作，本次实验需要预先安装虚拟机软件（Windows：VMware Workstation；Mac：VMware Fusion），载入课程提供的环境配置完毕的虚拟机配置文件（.vmx 文件），在虚拟机中完成实验。

- VMware Workstation 下载链接：https://www.vmware.com/products/workstation-pro/workstation-pro-

[1] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/ping
[2] https://iperf.fr/iperf-doc.php#3doc
[3] https://www.cnblogs.com/ggjucheng/archive/2012/01/14/2322659.html
[4] https://wiki.linuxfoundation.org/networking/tcpprobe

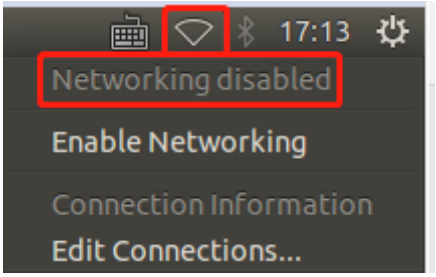
evaluation.html

- VMware Fusion 下载链接：
  https://cloud.tsinghua.edu.cn/f/b0fb45ffd8724c1ba6a0/?dl=1
  https://www.vmware.com/products/fusion/fusion-evaluation.html
- **课程实验提供的虚拟机配置文件下载链接：**
  https://cloud.tsinghua.edu.cn/f/e65f283ed8b84936a6f3/

（2） 下载虚拟机配置文件 Exp2_NEW.zip 解压后，在 VMware 主界面中点击左上角“文件(F)”，在弹出下拉框中点击“打开(O)”，选择并打开下载的虚拟机配置文件（.vmx 文件），之后根据提示选择.vmdk 文件，点击“开启虚拟机”，选择“我已复制该虚拟机”，完成仿真环境搭建，若弹出“该虚拟机似乎正在使用中”提示，选择获取所有权。注意：使用 VMware Fusion 不要根据界面提示选择虚拟机，而需要直接双击.vmx 文件打开。

（3） 启动虚拟机后，将进入 Ubuntu 系统。用户名：cn，密码：12345678，桌面存放本次实验相关文件。

（4） 为保证本次实验顺利进行，需要切断虚拟机的网络连接，如下图所示：



（5） 本次实验的关键文件和代码及作用参见下表。

表1 实验主要文件及功能

文件名	功能及说明
Exp2_TCP.ipynb	核心代码，位于虚拟机内，实验将在其中调用其他文件函数完成。
plot_xx.py	用于读取.txt 文件作图分析的 python 代码，位于虚拟机内，可直接在 ipynb 文件中调用。xx 包含 cwnd、ping，对应网络性能指标。

4.2 Mininet 网络仿真工具介绍

Mininet 是一个轻量级网络配置和测试平台。它采用轻量级的虚拟化技术使一个单一的系统看起来像一个完整的网络，其中包含内核系统和用户代码，可以模拟一个完整的网络，主机、链接和交换机在同一台计算机上且有助于互动开发、测试和演示。其所具有的特点包括：

- 支持系统级的还原测试，支持复杂拓扑，自定义拓扑等；
- 提供 Python API，方便多人协作开发；

- 很好的硬件移植性与高扩展性；
- 支持数千台主机的网络结构。

本次实验将基于 Mininet 的 Python API 搭建基本的实验网络仿真平台，基本使用方法可参考查阅 Mininet 用户指南[Working with Mininet]提供的相关内容。

4.3 Wireshark 抓包分析工具介绍

Wireshark 是一款广泛使用的、功能强大的开源抓包软件，常用来检测网络问题和分析网络通信机制，具有友好的图形化交互界面。其所具有的特点包括：

- 支持超过上千种协议数据包采集和分析；
- 用户友好度较高的图形化交互界面；
- 支持包括 Windows、Mac OSX 以及基于 Linux 的操作系统。

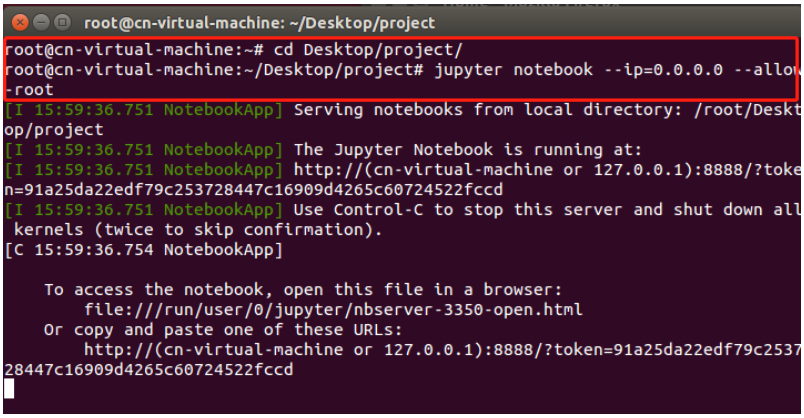
本次实验我们已经安装 Wireshark，位于桌面工具栏。我们首先通过 Linux 系统自带的命令行抓包分析工具 tcpdump 进行抓包，之后利用 Wireshark 图形化界面对数据包进行分析，完成实验内容。详细使用方法可参考查阅 Wireshark 用户指南[Wireshark Users' Guide]和提供的使用教程。

5. 实验流程

5.1 网络仿真环境运行和实验网络搭建

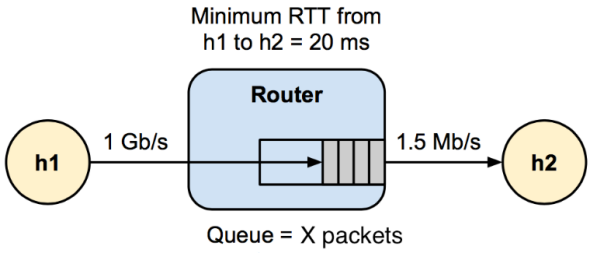
（1） **启动 jupyter notebook**：打开 Terminal，进入 project 文件夹，并以 root 权限启动 jupyter notebook，具体命令为：

jupyter notebook --ip=0.0.0.0 --allow-root



浏览器弹出界面如下，打开 Exp2_TCP.ipynb，运行网络仿真环境，**附录提供了 Jupyter Notebook 运行的基本命令和操作，供参考。**

（2） **搭建实验网络：**通过 Mininet 的 Python API 创建如下的网络，用以模拟典型家庭网络，这里为了简化网络我们将头部路由器省去，直接用“家庭电脑-家庭路由器-服务器主机”的网络结构模拟。这里 h1 是家庭电脑，它可以通过较快速度（1Gb/s）连接到家庭路由器。家庭路由器有一个缓慢的上行连接（1.5Mb/s）到互联网服务器主机 h2。h1 和 h2 之间的往返传播延迟（最小 RTT）是 20ms。



我们通过如下代码，实现搭建网络拓扑和 Mininet 实例化的功能。具体地，我们通过 addLink 函数，建立 h1 至路由器、路由器至互联网服务器的简单拓扑。addLink 函数设置的参数包括带宽 bw，延迟 delay 和最大队列长度 max_queue_size，**本部分实验我们将最大队列长度设置为 10**：

def mininet.topo.Topo.addLink(self, node1, node2, bw, delay, max_queue_size)

5.1 网络仿真环境运行和实验网络搭建

```
from mininet.topo import Topo
from mininet.node import CPULimitedHost, OVSController
from mininet.link import TCLink
from mininet.net import Mininet
from mininet.log import lg, info
from mininet.util import dumpNodeConnections

class BBTopo(Topo):
    "Simple topology for bufferbloat experiment."

    def __init__(self, queue_size):
        super(BBTopo, self).__init__()

        # Create router s0 (这里不区分交换机和路由器,统一用addSwitch命令添加)
        s0 = self.addSwitch('s0')

        # Create two hosts with names 'h1' and 'h2'
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')

        # Add links with appropriate bandwidth, delay, and queue size parameters.
        # Set the router queue size using the queue_size argument
        # Set bandwidths/latencies using the bandwidths and minimum RTT given in the network diagram above
        self.addLink(h1, s0, bw=1000, delay='10ms', max_queue_size=queue_size)
        self.addLink(h2, s0, bw=1.5, delay='10ms', max_queue_size=queue_size)

        return

import os
# Set the cwnd control algorithm to "reno"
os.system("sysctl -w net.ipv4.tcp_congestion_control=reno")
# create the topology with queue_size=10
topo = BBTopo(queue_size=10)
```

5.2 TCP 流量产生和数据包抓取

（1） **利用 iperf 工具产生两个主机 h1 和 h2 之间的 TCP 流量：**我们通过 start_iperf 函数建立了一个长期的 TCP 连接：使用 iperf 从 h1 向 h2 发送数据。iperf 是“一种主动测量 IP 网络上最大可实现带宽的工具”。具体地，我们通过如下代码实现：

```
def start_iperf(net, experiment_time):
    # Start a TCP server on host 'h2' using perf.
    # The -s parameter specifies server mode
    # The -w 16m parameter ensures that the TCP flow is not receiver window limited (not necessary for client)
    print("Starting iperf server")
    h2 = net.get('h2')
    server = h2.popen("iperf -s -w 16m", shell=True)

    print("Starting iperf client")
    h1 = net.get('h1')
    # Start an TCP client on host 'h1' using iperf. Ensure that the client runs for experiment_time seconds
    client = h1.popen("iperf -c {0} -t {1}".format(h2.IP(), experiment_time), shell=True)
```

该函数 start_iperf 接收一个名为 “net” 的参数，它是一个 Mininet 的实例，具有我们上面创建拓扑结构。具体而言，可通过下述命令指定发送流量对象和传输时，实现在一段时间内客户端运行 iperf，iperf 命令同时需要包括 h2 的 IP 地址，可通过 h2.IP()方法访问，具体命令形式为：

iperf -c <server address> -t <time>

具体参数含义可参阅 https://iperf.fr/iperf-doc.php#3doc

进一步，需要使用 popen 函数来模拟在 Mininet 主机上运行 shell 命令，popen 的第一个参数是一个字符串命令，同在 shell 中运行一致，第二个参数是 shell=True：

def mininet.node.Node.popen(self, *args, shell)

（2） **利用 Linux 自带的命令行抓包分析工具 tcpdump 进行抓包：**我们通过 tcpdump 进行抓包，并写入到 test_10_tcpdumper.pcap 文件中，以用于后续实验 Wireshark 载入该文件，进行解析数据包。具体地，我们通过如下代码实现：

```
# Start capturing packets
from subprocess import Popen
experiment_name = 'test_10'    # set experiment name
tcpdumper = Popen("tcpdump -S -w ./{}_tcpdumper.pcap".format(experiment_name), shell=True)
```

类似于 popen 函数，Popen 函数模拟在系统中启动 tcpdump 抓包。在 tcpdump 命令中，-S 表示保存数据包的原始序列号，-w 表示将包数据写入文件。

（3） **利用 tcpprobe 工具监测 TCP 流量的相关性能指标：**tcpprobe 可以记录 RTT 和拥塞窗口等指标，我们通过 “sudo cat /proc/net/tcpprobe > test_10_cwnd.txt” 命令[5]将监测数据写入到文件中，该文件中每一行的数据字段含义如下：

[时间戳] [源 IP 及端口] [目的 IP 及端口] [数据包大小]
[下一个带发送数据包序列号] [待确认数据包序列号] [拥塞窗口大小]
[慢启动阈值] [发送窗口大小] [估计 RTT] [接收窗口大小]

具体地，我们如下代码实现：

```
import os
def start_tcpprobe(outfile="cwnd.txt"):
    Popen("sudo modprobe tcp_probe", shell=True)
    Popen("sudo cat /proc/net/tcpprobe > " + outfile, shell=True)

def stop_tcpprobe():
    Popen("killall -9 cat", shell=True).wait()
```

其中，“sudo modprobe tcp_probe” 是用 modprobe 加载 tcpprobe 以监

[5] “>” 是输出重定向操作，将命令输出重定向到文件。

听当前的 TCP 流；加载 tcpprobe 模块后，会新增一个/proc/net/tcpprobe 的接口，可以通过这个接口获取 tcpprobe 捕捉的信息。

（4） **利用 ping 命令实现主机 h1 向主机 h2 发送请求：**我们通过 start_ping 函数实现从 h1 到 h2 的 ping 请求，并测量样本 RTT，将输出写入到 test_10_pings.txt 文件中。具体地，我们通过如下代码实现：

```
def start_ping(net, outfile="pings.txt"):
    # Start a ping train from h1 to h2 with 0.1 seconds between pings, redirecting stdout to outfile
    print("Starting ping...")
    h1 = net.get('h1')
    h2 = net.get('h2')
    h1.popen("ping -i 0.1 {0} > {1}".format(h2.IP(),outfile), shell=True)
```

其中，“h2.IP()”命令获取 h2 的 IP。

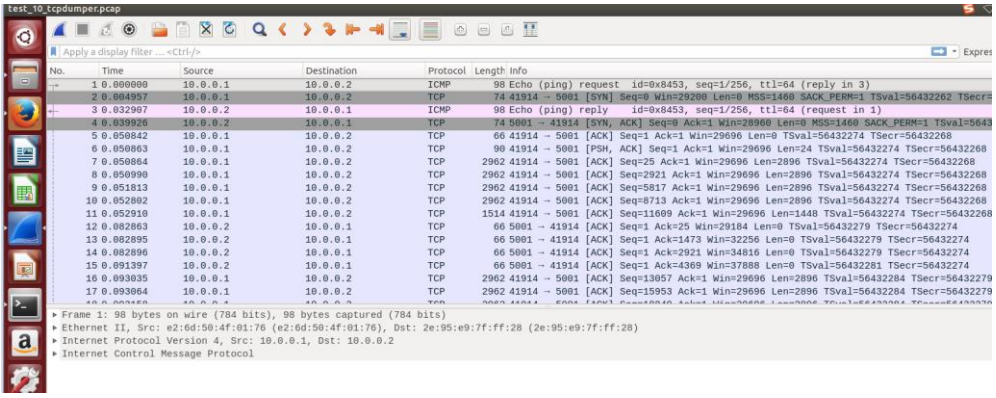
（5） **在规定实验时间内关闭网络仿真环境：**我们通过 stop_tcpprobe、kill 等函数关闭网络仿真环境。

通过运行 Jupyter Notebook 中的代码块，本部分实验获得的输出文件包括：

文件	用途
test_10_tcpdumper.pcap	用于后续 Wireshark 解析数据包
test_10_cwnd.txt：	用于估计 RTT 获取
test_10_pings.txt	用于样本 RTT 获取

5.3 TCP 连接管理

（1） 将上小节实验输出文件 test_10_tcpdumper.pcap 用 Wireshark 打开，如下图所示。



（2） 根据课堂讲授知识，找到从 h1 到 h2 的 TCP 连接建立过程中三次握手发送的数据包，补全下表中空白部分：

- 第一步，h1 发送的报文段：

源端口号：	目的端口号：
序号：	
确认号：	

首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN	

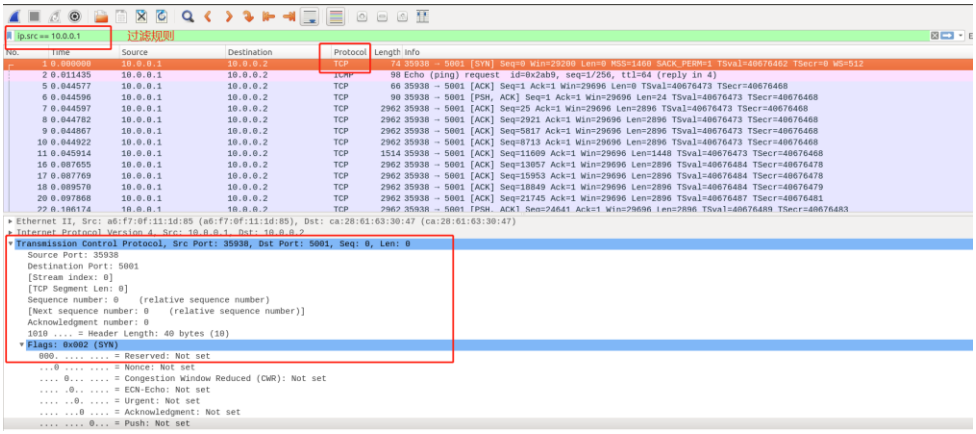
- 第二步，h2 发送的报文段：

源端口号：								目的端口号：
序号：								
确认号：								
首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN	

- 第三步，h1 发送的报文段

源端口号：								目的端口号：
序号：								
确认号：								
首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN	

提示：可根据 TCP 连接建立数据包的特点如（SYN、ACK）进行过滤（例如，tcp.flags.syn==1 过滤出 SYN=1 的数据包），例如下面通过 ip.src==10.0.0.1 过滤出所有从 h1 发送的数据包。



（3） 根据理论课讲授知识，找到从 h1 到 h2 的 TCP 连接终止过程中发送四个数据包，补全下表中空白部分：

- 第一步，h1 发送的报文段：

源端口号：	目的端口号：
序号：	
确认号：	

首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN	

- 第二步，h2 发送的报文段：

源端口号：								目的端口号：
序号：								
确认号：								
首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN	

- 第三步，h2 发送的报文段

源端口号：								目的端口号：
序号：								
确认号：								
首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN	

- 第四步，h1 发送的报文段

源端口号：								目的端口号：
序号：								
确认号：								
首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN	

5.4 TCP 可靠传输

（1） 根据课堂讲授知识，利用 Wireshark 针对 test_10_tcpdumper.pcap 文件，找到为保证可靠数据传输，h2 发送给 h1 的重复 ACK 包，记录数据包信息，并分析重传原因。

（2） 运行 Exp2_TCP.ipynb 中 5.4 提供的代码，观察基于 ping 的采样 RTT 和基于 tcpprobe 的估计 RTT 之间的差别，并分析解释可能的原因。

5.4 TCP可靠传输

根据实验指导书要求，利用Wireshark完成

运行下放代码，观察基于ping的采样RTT和基于tcpprobe的估计RTT之间的差别

In [7]:
```
from plot_ertt import plot_srtt_ertt
plot_srtt_ertt('test_10')
```

（3） 【选做】根据课堂讲授知识，基于采样 RTT 根据相关公式计算 估计 RTT，并将计算出的估计 RTT 曲线与基于 tcpprobe 的汇报的估计 RTT 曲线对比分析。提示：需要读取 test10_pings.txt 存储的采样 RTT 信息和 test_10_cwnd.txt 存储的基于 tcpprobe 的估计 RTT 信息。

5.5 TCP 流量控制

（1） 利用 Wireshark 针对 test_10_tcpdumper.pcap 文件，记录在一段连续时间内 h2 的接收窗口变化并分析变化原因。

时间																		
接收窗口																		

5.6 TCP 拥塞控制

（1） 运行 Exp2_TCP.ipynb 中 5.6 提供的代码[6]，绘制拥塞窗口变化曲线，从中区分慢启动、拥塞避免和快速恢复阶段。

5.6 TCP拥塞控制

```
%matplotlib inline
from plot_cwnd import plot_congestion_window
plot_congestion_window('test_10_cwnd.txt', histogram=False)
```

（2） 根据 Exp2_TCP.ipynb 中提供的代码，通过在网络构建时设置的 queue_size 值，绘制路由器缓存大小（即最大队列长度）为 10、50 和 100 时的拥塞窗口变化曲线，记录并标出对应的慢启动、拥塞避免和快速恢复阶段，进行对比分析。

```
import os
# Set the cwnd control algorithm to "reno"
os.system("sysctl -w net.ipv4.tcp_congestion_control=reno")
# create the topology with queue_size=10
topo = BBTopo(queue_size=10)
```

（3） 【选做】增大路由器缓存大小为 500、1000，当缓存达到某一数值时，会出现缓存溢出（bufferbloat）现象[7]，产生拥塞。请根据实验记录的拥塞窗口和 RTT 变化，比较缓存大小为 10/50/100 和 500/1000 情况下拥塞窗口和 RTT 的绝对大小，分析缓存大小为 500/1000 时 RTT 显著增大的原因。

5. 实验考核

（1） 完成上述实验必做内容，理解 TCP 在连接管理、可靠传输、流量控制和拥塞控制四方面的实现原理；

[6] 这里我们将 TCP 拥塞控制算法设置为最经典的 Reno 算法，后续提出了很多改进的算法，同学们可以自行研究。

[7] 网络路由器的缓存过大，也可能会出现数据包过度缓存而导致网络延迟大的现象，即 Bufferbloat（缓存膨胀）现象。https://en.wikipedia.org/wiki/Bufferbloat

（2） 通过实验记录，结合相关分析，对课堂讲授知识和实验现象进行验证和解释；
（3） 掌握利用 Wireshark 软件解析数据包报文的能力。
（4） 网络学堂中提交实验报告，报告内容合乎逻辑，表达清晰，有实验过程记录、截图及思考题回答。提交截止时间为 2022 年 10 月 14 日。

6. 实验思考题

（1） 本实验中，基于 Wireshark 和抓到的数据包，分析 h1 的接收窗口变化情况，请解释产生这样现象的原因。
（2） 本实验利用 Wireshark 分析抓包过程中，除了本实验重点分析的 TCP 协议数据包，还存在哪些其他类型数据包？通过进一步 Baidu 或查阅资料确定这些数据包对应于网络哪一层。

7. 实验参考资料

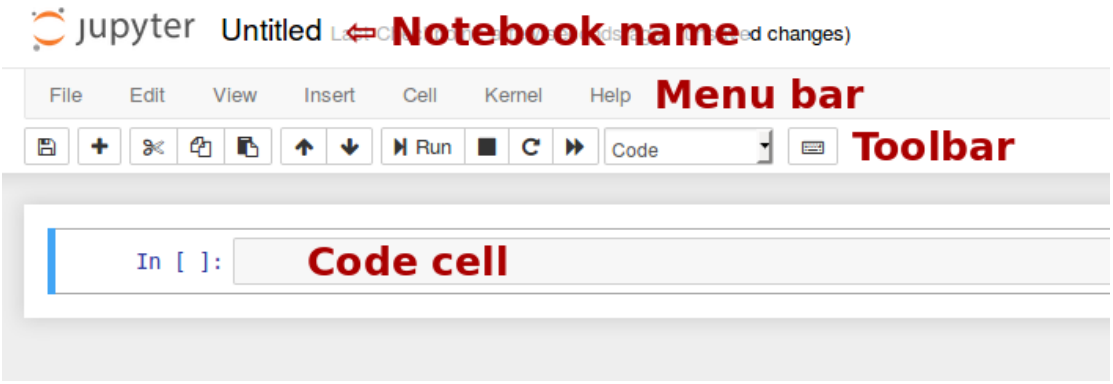
（1） Python 教程：https://www.liaoxuefeng.com/wiki/1016959663602400
（2） Iperf 文档：https://iperf.fr/iperf-doc.php#3doc
（3） TCP Probe：https://wiki.linuxfoundation.org/networking/tcpprobe
（4） Python Mininet API：https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#working
（5） Jupyter Notebook 文档：https://docs.jupyter.org/en/latest/start/index.html
（6） Wireshark 使用：https://blog.csdn.net/qq_44204058/article/details/123014013
（7） tcpdump 使用：https://www.cnblogs.com/ggjucheng/archive/2012/01/14/2322659.html

8. 附录

8.1 Jupyter Notebook 基本操作

Jupyter Notebook 是基于网页的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。官方文档：https://jupyter-notebook.readthedocs.io/en/stable/notebook.html。简而言之，Jupyter Notebook 是以网页的形式打开，可以在网页页面中直接编写代码和运行代码，代码的运行结果也会直接在代码块下显示的程序。如在编程过程中需要编写说明文档，可在同一个页面中直接编写，便于作及时的说明和解释。

Notebook 用户界面：当创建/打开新的 Notebook 文档时，将看到 Notebook 名称，菜单栏，工具栏和空的编码单元。



Notebook 名称：显示在页面顶部的 Jupyter 徽标旁边的名称反映了.ipynb 文件。单击 Notebook 名称会弹出一个对话框，允许重命名它。

菜单栏：菜单栏显示不同的选项，可用于操作 Notebook 的功能方式。

工具栏：通过单击图标，工具栏可以快速执行 Notebook 中最常用的操作。

编码单元：单元格的默认类型；可以修改为 Markdown cell，常用于编辑文字。

特别地，单元格有两种模式，命令模式和编辑模式。命令模式对应蓝色左边框，编辑模式对应绿色左边框。

通过 Cell 栏可以使用鼠标操作运行单元格。

通过键盘操作运行，常用快捷键：

- shift+回车：运行单元格并跳转到下一个单元格
- ctrl+回车：运行单元格但不跳转到下一个单元格
- 选中单元格，按下 M 键，变成 Markdown 单元格
- 选中单元格，按下 Y 键，变成代码单元格
- 选中单元格，在命令模式下，
  - 按下 A 键在该单元格上方添加单元格
  - 按下 B 键在该单元格在下方添加单元格
  - 按下 D 键删除当前单元格

8.2 VMware Fusion 加载虚拟机

VMware Fusion 加载虚拟机.vmx 文件可能遇到 bug 的解决方案：


1
2
3
4
5
6
7
8
无法启动虚拟机。
好
是否要升级此虚拟机?
升级
不升级
不再询问
您正在运行的此虚拟机已启用侧通道缓解。
好
不再显示此对话框