《通信与网络》实验二

传输层TCP协议

清华大学 电子工程系

通信与网络课程组

2022年10月

目录

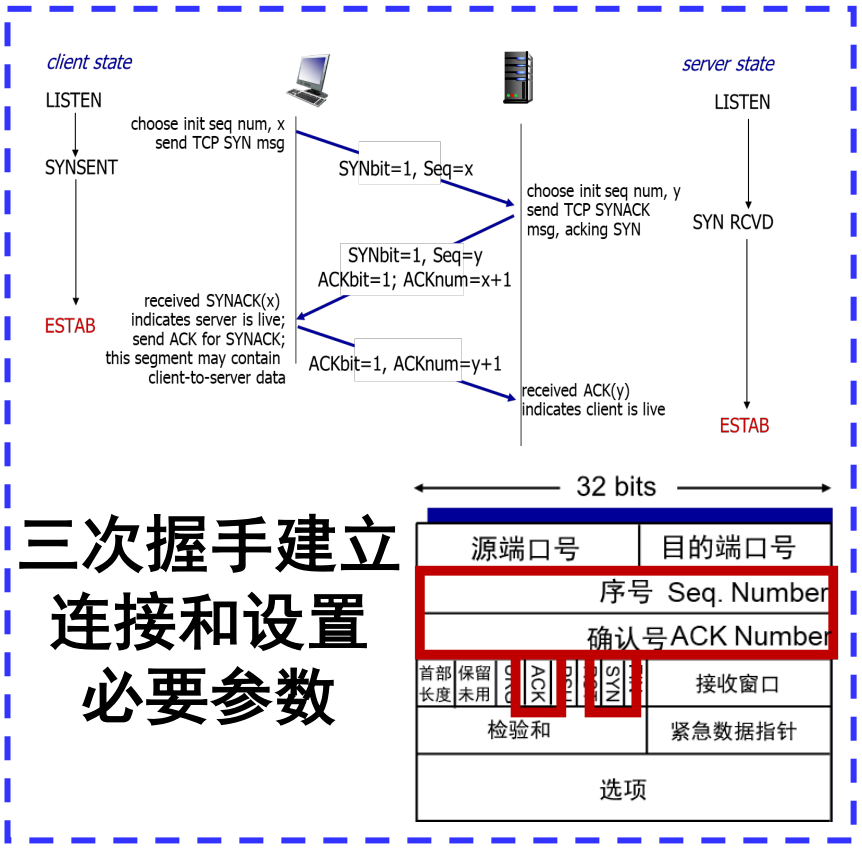
- 实验原理：TCP主要功能回顾
- 实验环境和工具：Mininet & Wireshark
- 实验内容和流程

一、实验原理：

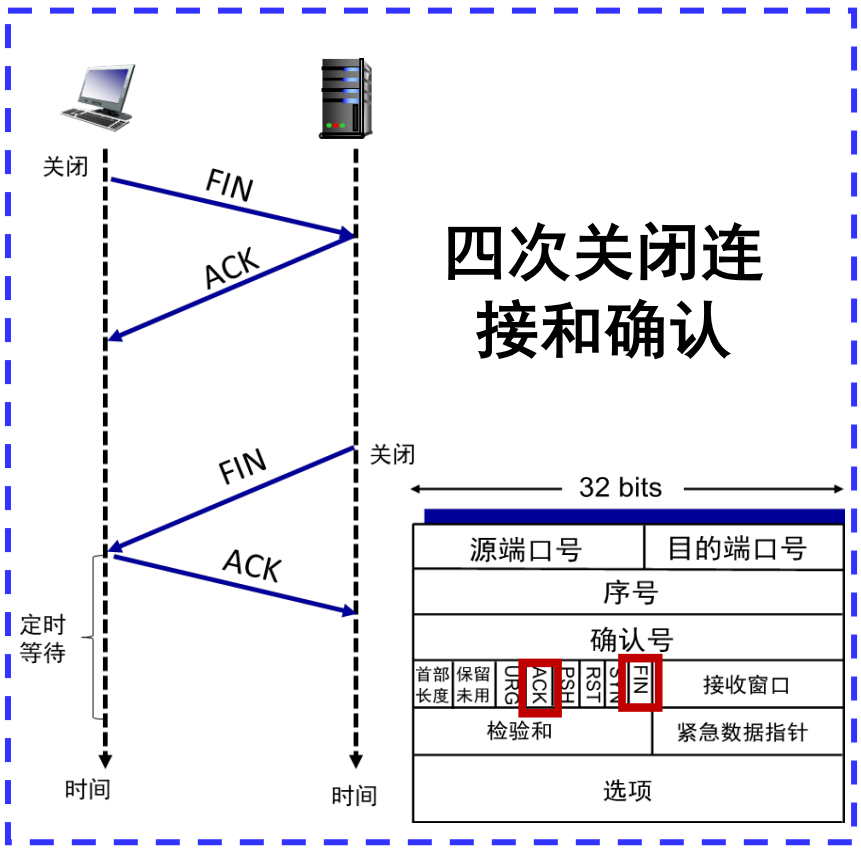
TCP主要功能回顾

TCP主要功能：连接管理

- TCP连接建立



- TCP连接终止

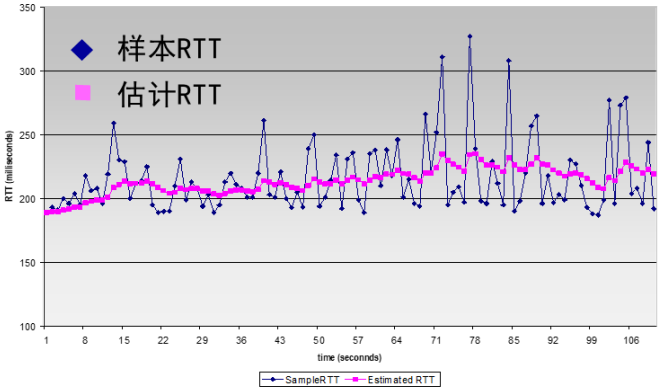


TCP主要功能：可靠数据传输

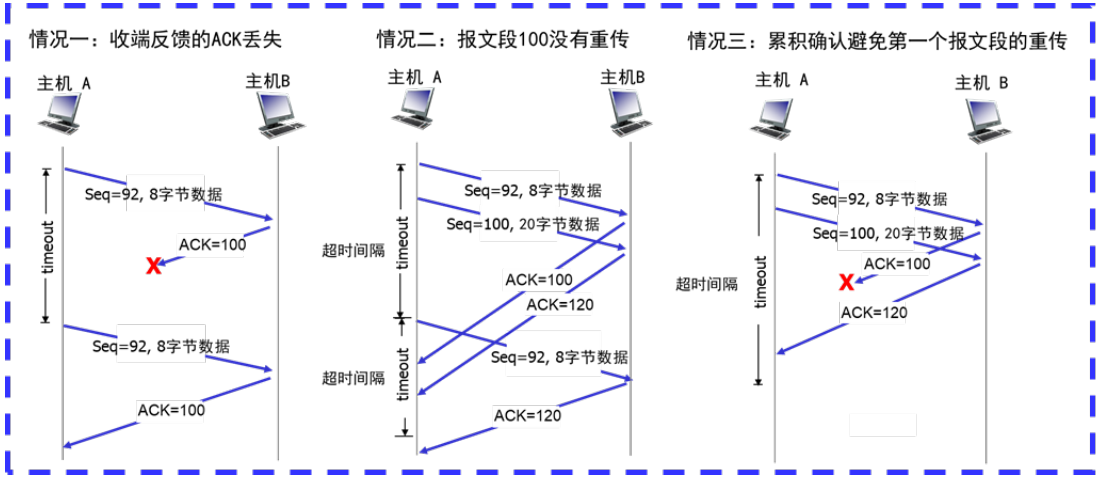
- 往返时间RTT估计

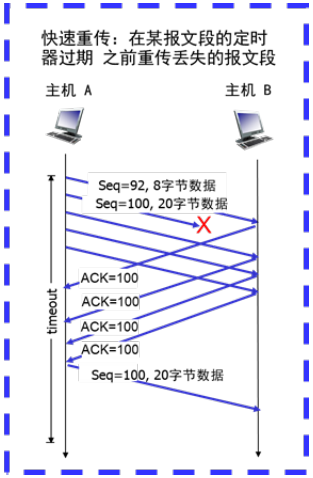
$$\underline{\texttt{EstimatedRTT}} = (1-\alpha)*\underline{\texttt{EstimatedRTT}} + \alpha*\underline{\texttt{SampleRTT}}$$

新的RTT估计值　　之前的RTT估计值　　新的RTT样本值



- 重传、快速重传



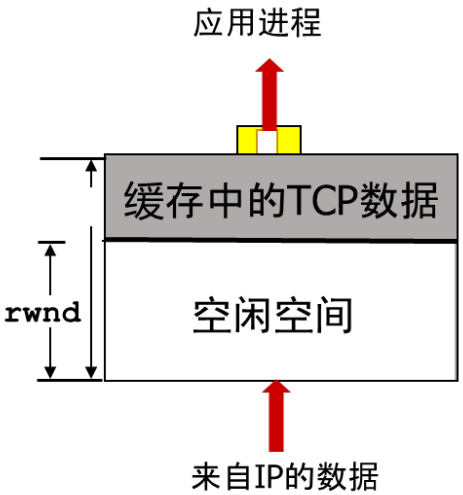


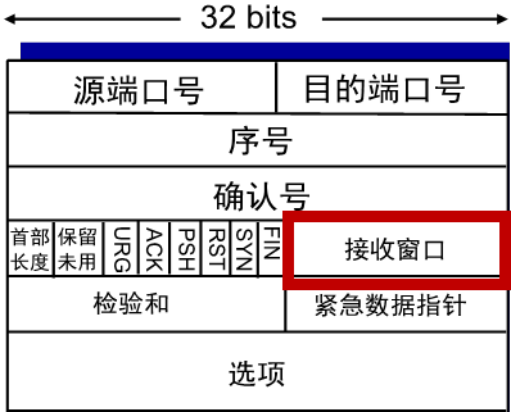
- 差错恢复

TCP主要功能：流量控制

- **接收窗口（cwnd）：接收端通过设置 TCP 分组报头中rwnd值告知发送端其缓存器的空闲空间大小**
  - RcvBuffer:接收端缓存器空间大小
  - LastByteRead：接应用进程从缓存读出的数据流最后一个字节的编号
  - LastByteRcvd：从网络放入接收缓存中的数据流最后一个字节的编号

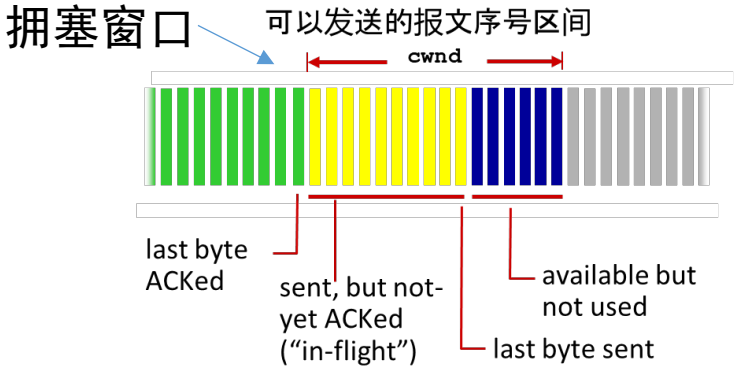
$$\textbf{rwnd} = \text{RcvBuffer} - [\ \text{LastByteRcvd} - \text{LastByteRead}\ ]$$

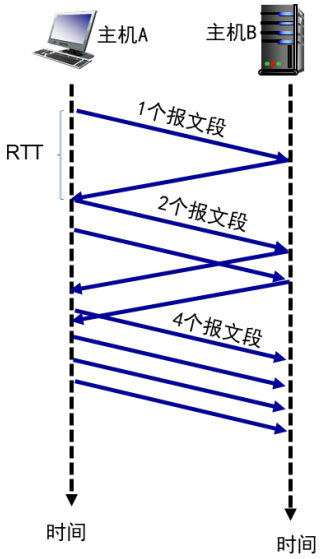




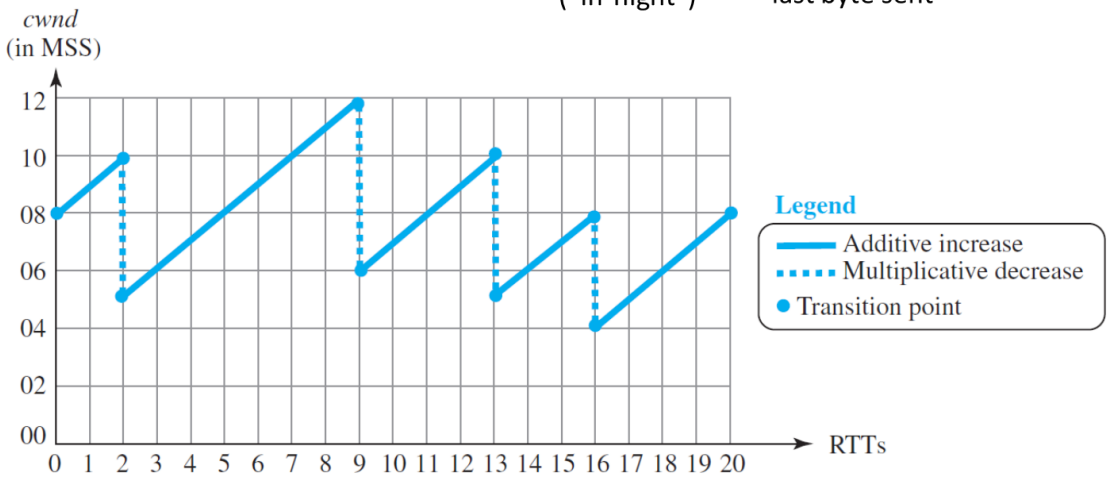
TCP主要功能：拥塞控制

- **拥塞控制算法（TCP congestion control algorithm）**
  - 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





TCP慢启动



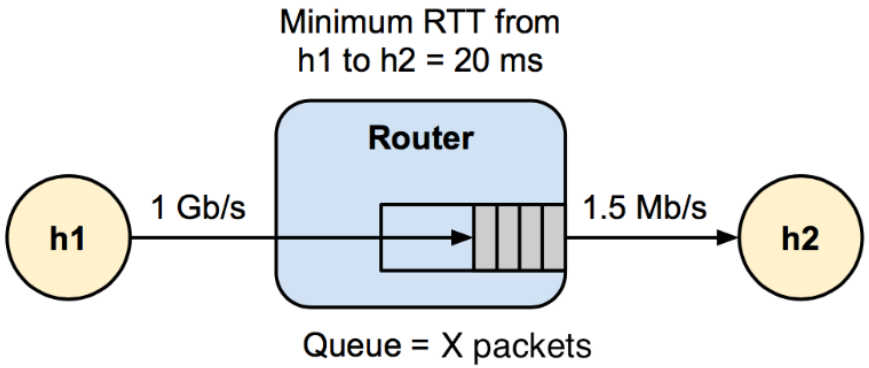
加性增、乘性减的拥塞控制

二、实验环境和工具

Mininet & Wireshark

实验环境

- **网络拓扑：家庭电脑-家庭路由器-服务器主机**



  - h1是家庭电脑，它通过快速链路连接（1Gb/s）到路由器s0
  - 路由器s0通过一个上行链路（1.5Mb/s）连接到互联网服务器h2
  - h1和h2之间的往返传播延迟（最小RTT）是20ms，默认h1和s0之间的往返传播延迟、h2和s0之间的往返传播延迟相等
  - 路由器的缓存大小（最大队列长度）将是模拟中的重要参数变量

实验工具

- **仿真工具：Mininet**
  - 轻量级网络配置和测试平台

Mininet
An Instant Virtual Network on your Laptop (or other PC)

- **抓包工具：Wireshark**
  - 一款广泛使用的、功能强大的开源抓包分析软件

WIRESHARK

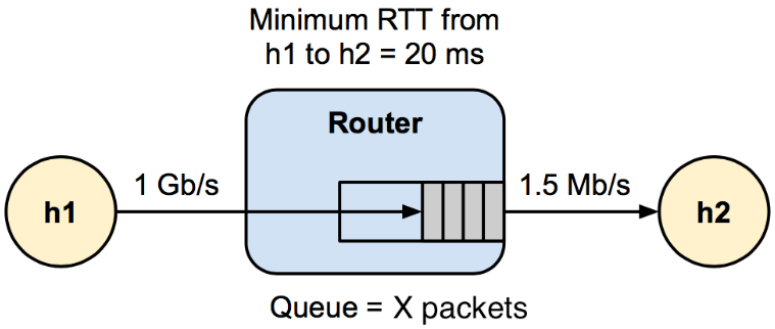
- **Linux自带网络仿真和性能监测模块**

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

三、实验内容和流程

1. 网络仿真环境运行和实验网络搭建

- 基于课程提供的虚拟机实验
  - 在虚拟机中启动Jupyter Notebook，建立网络



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)
```

2. TCP流量产生和数据包抓取

- 利用iperf产生TCP流量
- 利用tcpdump抓包
- 利用tcpprobe监测TCP流量性能指标
- 利用ping发送请求

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

字段含义

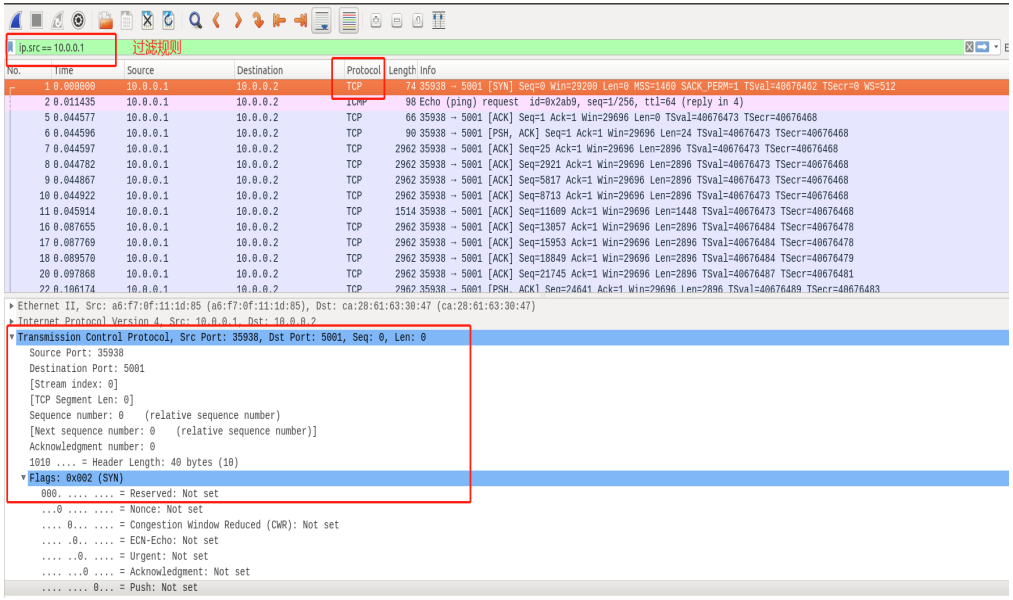
[时间戳] [源 IP 及端口] [目的 IP 及端口] [数据包大小]
[下一个带发送数据包序列号] [待确认数据包序列号] [拥塞窗口大小]
[慢启动阈值] [发送窗口大小] [估计 RTT] [接收窗口大小]

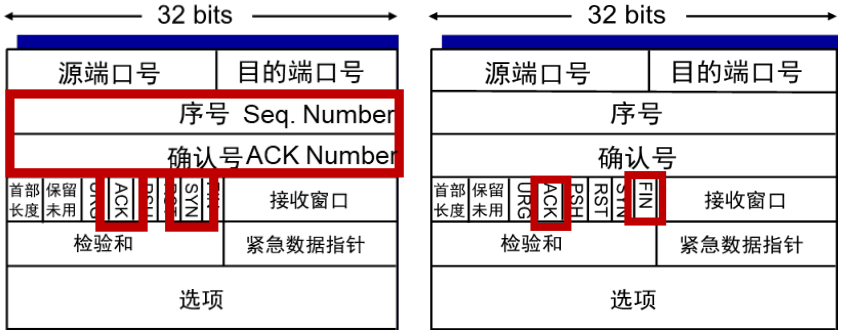
5.2 TCP连接建立和数据包抓取

```
# Start capturing packets
from subprocess import Popen
experiment_name = 'test_10'    # set experiment name
tcpdumper = Popen("tcpdump -s 0 -w ./{}_tcpdumper.pcap".format(experiment_name), shell=True)

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()

# Start monitoring TCP cwnd size
experiment_name = 'test_30'    # set experiment name
outfile = "{}_cwnd.txt".format(experiment_name)
start_tcpprobe(outfile)


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)

# Start the long lived TCP connections with start_iperf
experiment_time = 30
start_iperf(net, experiment_time)


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...")
```

3. TCP连接管理实验

- 记录TCP连接建立和连接终止的报文
  - 基于Wireshark规则过滤数据包
    - 例如，过滤SYN=1的数据包：tcp.flags.syn==1





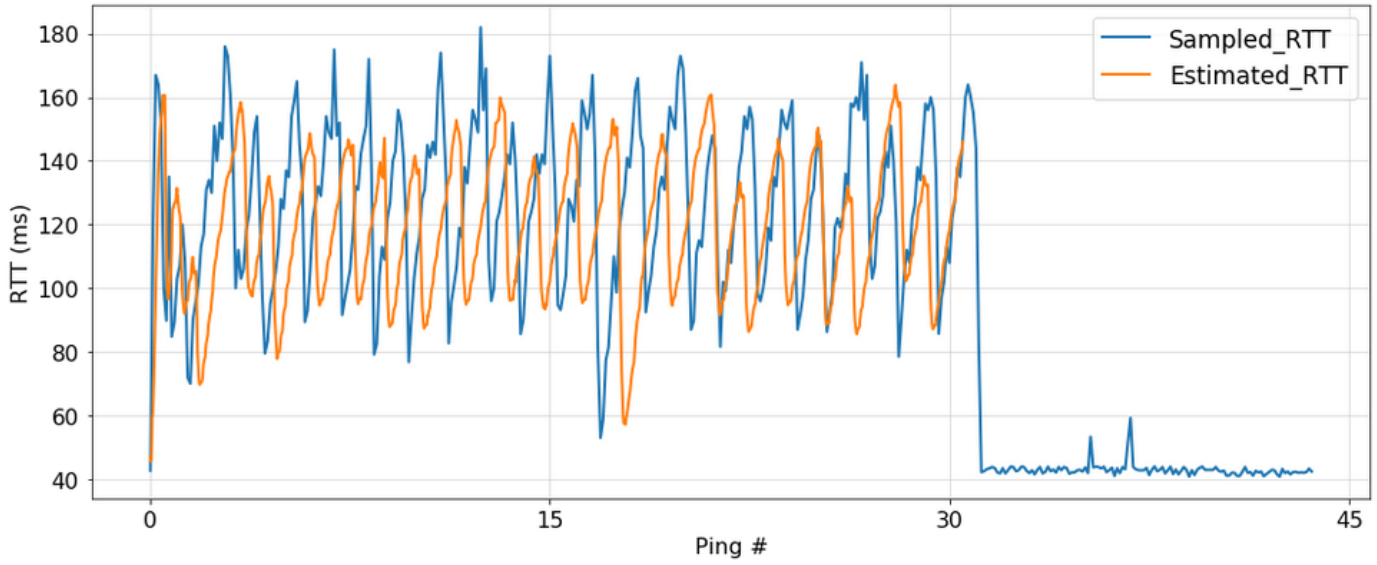
源端口号：							目的端口号：
序号：							
确认号：							
首部长度	保留位用	URG	ACK	PSH	RST	SYN	FIN

4. TCP可靠传输

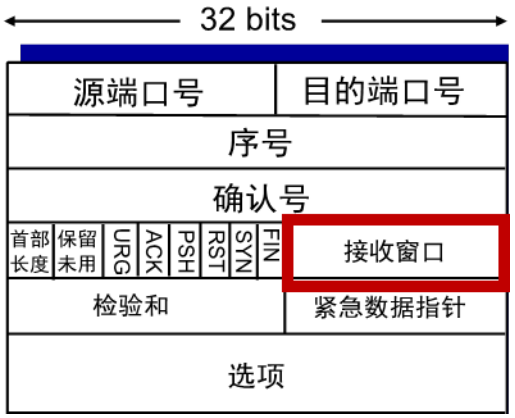
- 记录数据包重传现象
- 对比分析采样RTT v.s. 估计RTT

$$\underline{\texttt{EstimatedRTT}} = (1-\alpha)*\underline{\texttt{EstimatedRTT}} + \alpha*\underline{\texttt{SampleRTT}}$$

新的RTT估计值　　之前的RTT估计值　　新的RTT样本值

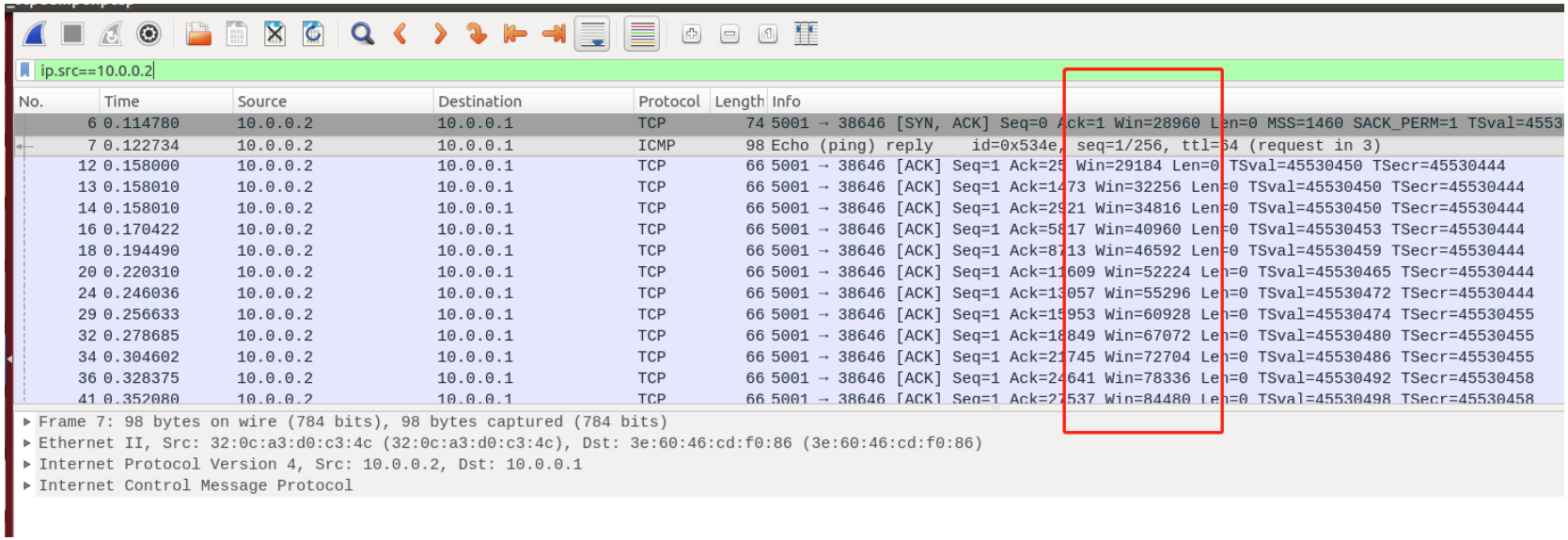


5. TCP流量控制



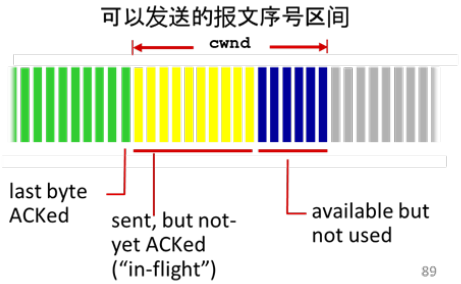
- 记录接收窗口变化情况

时间																	
接收窗口																	

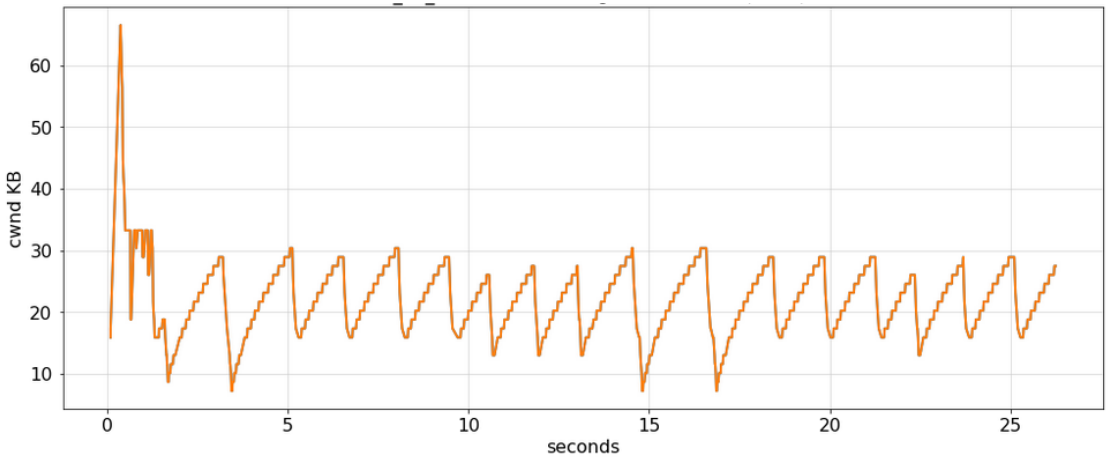


6. TCP拥塞控制

- 拥塞控制算法现象观察和验证
  - 慢启动、拥塞避免和快速恢复
  - 不同缓存大小的拥塞窗口和RTT变化
  - 选做：bufferbloat现象观察分析和解释



拥塞窗口



7. 注意事项

- **编程语言和环境**
  - 提供Python代码和虚拟机环境
  - 同时安装Wireshark软件
- **实验考核**
  - 提交实验报告至网络学堂，截止时间为2022年10月14日
  - 实验报告需包括实验中的重要现象、思考题回答