互联网体系结构初识

武庆华

wuqinghua@ict.ac.cn

提纲

- 课程说明
- 互联网体系结构初识
  - 互联网体系结构
  - 互联网性能
- 实验环境简介
- 两个简单实验
  - 互联网协议分析实验
  - 流完成时间实验

课程基本信息

- 计算机网络（研讨课）
  - 是计算机网络课程的扩展和具化
  - 通过实验等手段，认识：
    - 什么是计算机网络（互联网）
    - 如何构建一个完整的计算机网络系统

- 课程设置
  - 2次讲解（5课时）+ 11次实验（31课时）+ 2次学生报告（6课时）
- 考核方式
  - 实验代码+实验报告

课程设置

周次	内容	类别	难易程度	与本讲的关系	网络体系结构层次
5	互联网系统初识	讲解+实验	简单	/	
6	Socket网络编程实验	实验	简单	P16-P17	网络应用
7	网络转发实验	实验	简单	P20	组网
8	生成树网络实验	实验	简单	P20	
9	网络设备缓冲区实验	实验	简单	P35-P36	
10	网络转发+路由实验	实验	较难	P21	网络互联
11	高效IP查找实验	实验	简单	P21	
12	NAT实验	实验	简单	-	
13	网络层实验总结	报告	/	/	
14	网络传输实验	实验	较难	P22-P23	网络传输
15	拥塞控制发展历史	讲解	/	P9	
16	拥塞控制实验	实验	较难	P38	
17	用户态Socket机制实验	实验	简单	P24	
18	传输层实验总结	报告	/	/	

实验作业提交说明

- 每次实验满分计10分
  - 实验总成绩：$\sum_{i=1}^{N} S_i/\mathrm{N} * 10$
  - 机动加分：课堂表现
- 实验报告提交时间
  - 实验代码截止时间：（简单：一周；较难：两周）（OJ网站，下次课讲）
  - 实验报告截止时间：代码截止时间的第二天23:55（SEP网站）
    - 节假日调课时会调整提交截止时间
  - 如遇特殊情况（生病/会议等），可邮件补交实验报告
    - 发送邮件给zhaoyuankang@ict.ac.cn（赵员康）和 liuting19g@ict.ac.cn （刘婷），抄送给 wuqinghua@ict.ac.cn
    - 邮件标题为：补交第xx周实验报告 姓名 学号
    - 如果无充分理由，则拒绝该次补交作业

实验工具与参考文献

- 实验环境
  - 环境：VirtualBox + Ubuntu 18.04 ~ 21.04
  - 工具： MiniNet、Wireshark
  - 编程、脚本语言： C、Python 、 Linux Shell

- 学术论文
  - ACM SIGCOMM、 ACM HotNets 、ACM IMC 、 USENIX NSDI、ACM CoNEXT、IEEE ICNP

什么是网络？

- 网络是计算的“管道”？
- 几乎所有与计算相关的领域都以网络为基础
  - 边缘计算
  - 分布式计算
  - 云计算
  - 大数据
  - 智慧城市
  - 。。。
- 网络是计算的核心支撑！

网络 -> 万物互联 -> 智慧万物

Smart Health

Smart Watch

Smart Car

Smart City

Smart TV

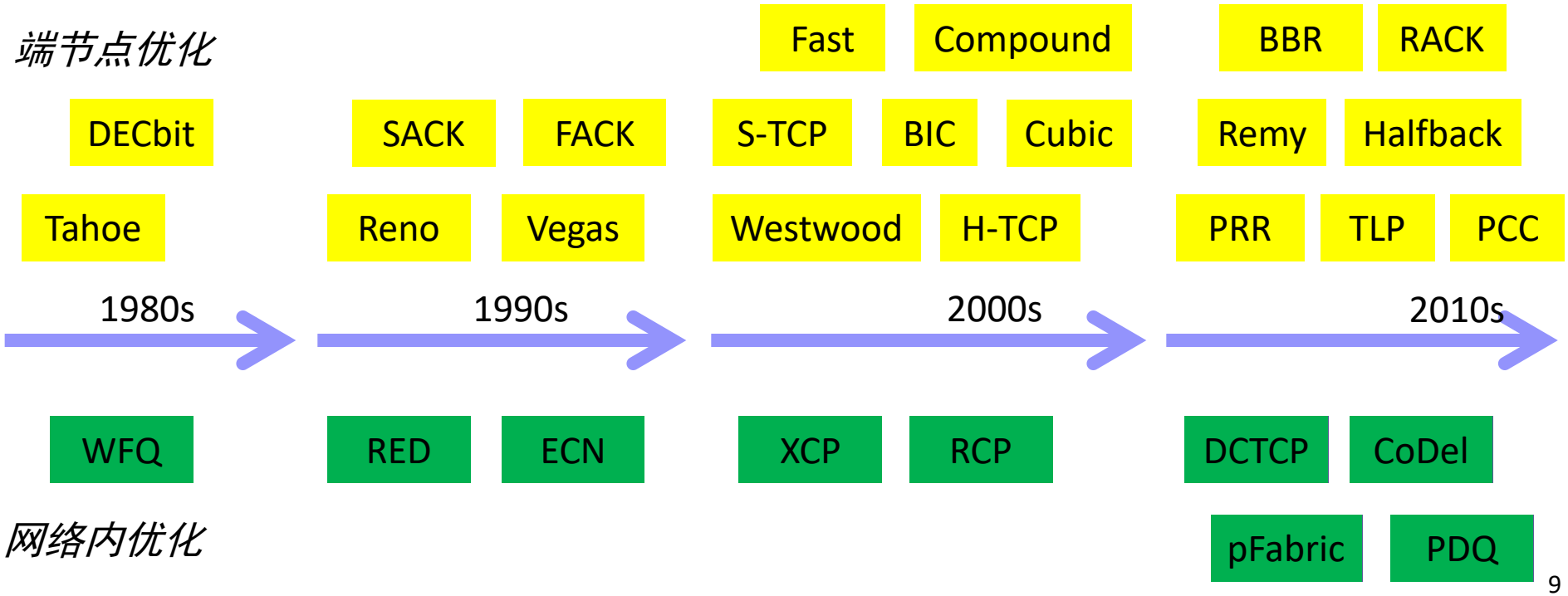
Smart Home

Smart Industry

网络不是已经很成熟了么？

- 网络一直在演进中
  - 底层基础设施越来越多样化
  - 上层应用越来越丰富
  - 对网络的要求越来越高

以网络传输机制为例



计算机网络领域的研究方法

- 构建系统
  - 先系统，后理论
  - 例子：ARPAnet、TCP性能理论模型 [van Jacobson 1988]
- 网络测量
  - 互联网是人造的、分布式的、异构的
  - 网络特征难以直接刻画，需要通过网络测量等手段认知
  - 例子：互联网流量的自相似特征 [Leland 1993]

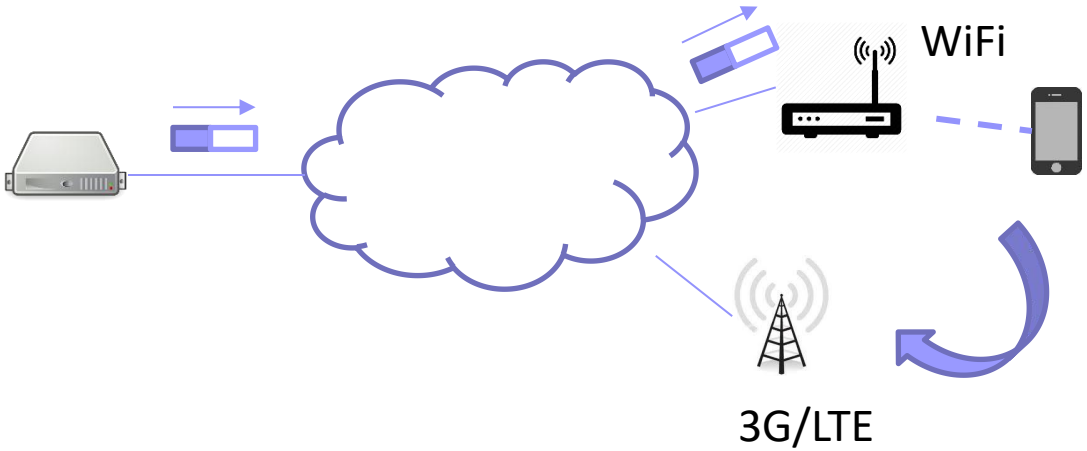
互联网体系结构

- 标识空间
  - 如何标识一个网络通信节点？
- 系统设计
  - 互联网系统是如何设计的？
- 协议层次
  - 互联网协议，协议间的关系

标识空间

- 如何标识一个网络通信节点？
  - 标识是固定的，还是变动的？
    - （固定：www.baidu.com；变动：159.226.39.22）
  - 标识是扁平化的，还是层次化的？
    - （扁平：02:42:6b:a7:7c:ef；层次：www.baidu.com ）
  - 标识是全局的，还是局部的？
    - （全局：159.226.39.22；局部：10.0.0.2）

固定/变动标识的例子



- 一个主机的标识是固定的（名字）还是变动的（地址）？
  - 如使用固定标识，如何告知对方将数据发送到自己所在的位置
  - 如使用变动标识，在移动到新的位置后，如何证明你是原来通信的节点？

互联网系统的标识空间

- 互联网有三个不同层次的标识空间

层次化的域名空间

举例：www.ict.ac.cn，具有易读性，层次化命名，递归解析，端节点可见

DNS解析，将域名映射成IP地址

层次化的IP地址空间

举例：159.226.97.84，层次化编址，固定长度，计算机易处理，互联网可见

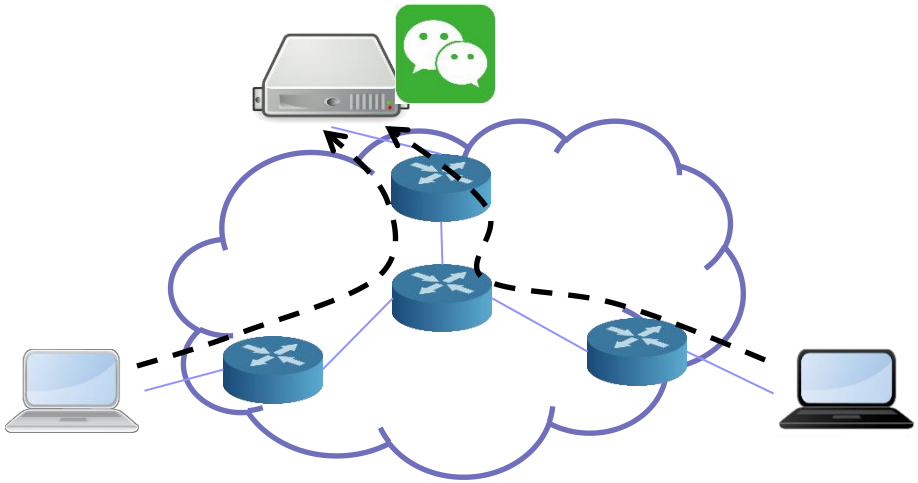
ARP解析，查找IP地址对应的下一跳MAC地址

扁平化的MAC地址空间

举例：08:00:27:b4:1c:a7，扁平化编址，固定长度，在局域网可见

互联网系统设计

- 互联网的核心是消息分发和信息共享



- 三个核心部件 [Koponen 2011]：
  - 网络编程接口：对上层应用提供统一的调用接口
  - 报文传递：将报文从一个端节点传送到另一个端节点
  - 网络安全：网络体系结构内在安全

网络程序接口

- BSD Socket API
  - 不是为每个应用程序定义接口，而是提供最基本的通信功能
  - 对上层提供统一的调用接口，支持丰富的上层应用开发

BSD Socket API

socket(domain, type, proto);
close(sockfd);
bind(sockfd, addr, addrlen);

Datagram
sendto(sockfd, buf, len, flags, dest_addr, addrlen); recvfrom(sockfd, &buf, len, flags, src_addr, &addrlen);
Stream

accept(sockfd, addr, &addrlen);
connect(sockfd, addr, addrlen);
send(sockfd, buf, len, flags);
recv(sockfd, &buf, len, flags);

编程示例

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
```

```
int sockfd, n;
char buffer[1024];
struct hostent *server;
struct sockaddr_in serv_addr;

const char *req = "GET / HTTP/1.0\r\nHost: www.ict.ac.cn\r\n\r\n";

server = gethostbyname("www.ict.ac.cn");
serv_addr.sin_family = AF_INET;
bcopy(server->h_addr, &serv_addr.sin_addr.s_addr, server->h_length);
serv_addr.sin_port = htons(80);

sockfd = socket(AF_INET, SOCK_STREAM, 0);

connect(sockfd,(struct sockaddr *)&serv_addr, sizeof(serv_addr));
send(sockfd, req, strlen(req), 0);
while ((n = recv(sockfd, buffer, sizeof(buffer), 0)) > 0)
        fwrite(buffer, 1, n, stdout);

close(sockfd);
```

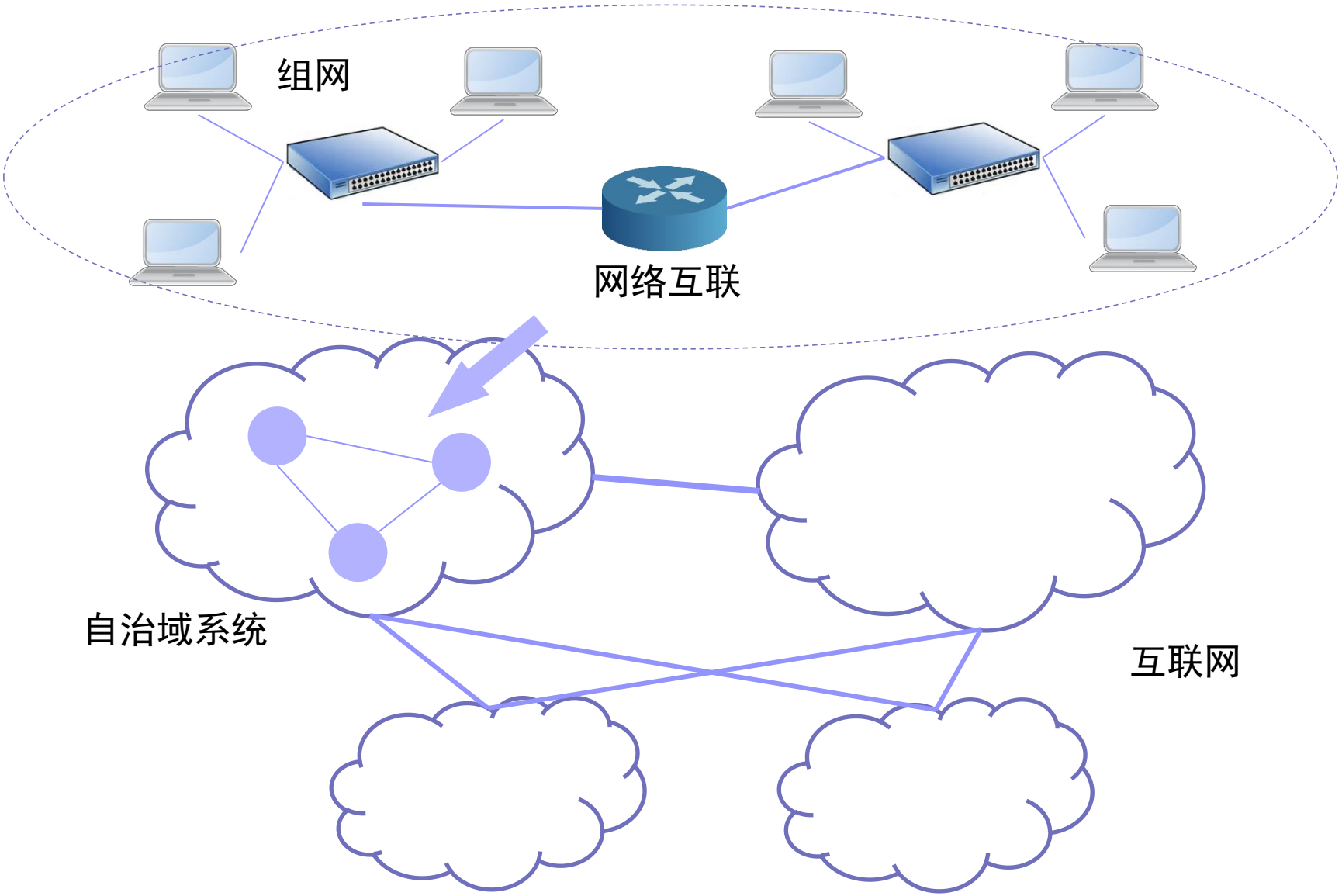
报文传递

- 报文传递：将报文从一个端节点传送到另一个端节点

报文传递
- 确定端节点到端节点的传输路径 —— 组网与网络互联
- 将报文按照确定好的路径进行传送 —— 数据传输

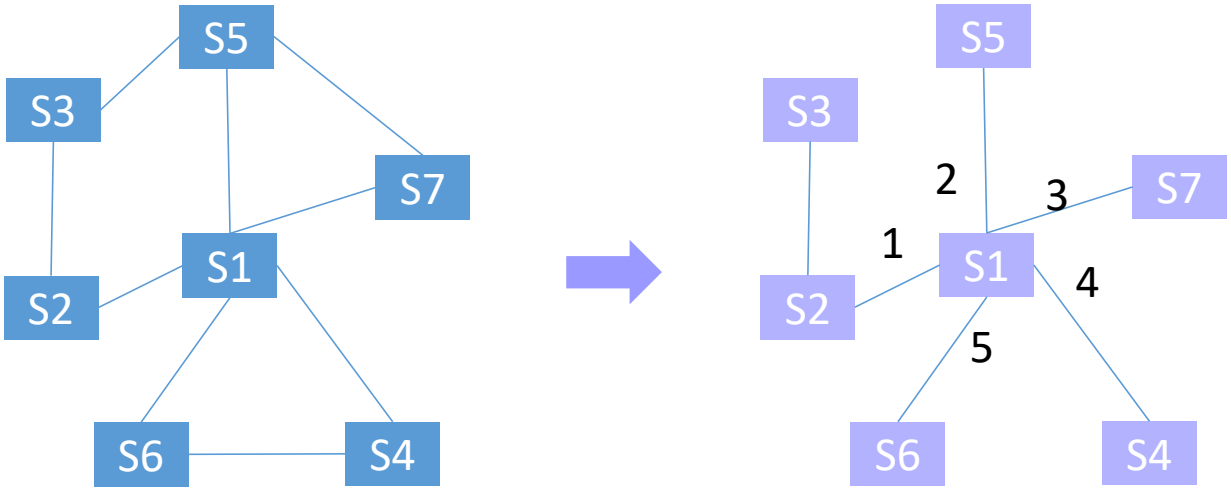
- 组网与网络互联
  - 动态的维护网络互联
  - 保证端到端数据可达
- 数据传输
  - 连接管理
  - 可靠传输
  - 流量控制
  - 拥塞控制

组网与网络互联



组网

- 网络中通常包含冗余链路，形成图状网络，提升健壮性
  - 如果直接转发，会形成环路
  - 使用最小生成树(Minimum Spanning Tree)算法，计算生成对应的最小代价的树状转发拓扑
  - 每个交换机节点保存到所有其它节点(MAC地址)的转出端口映射关系

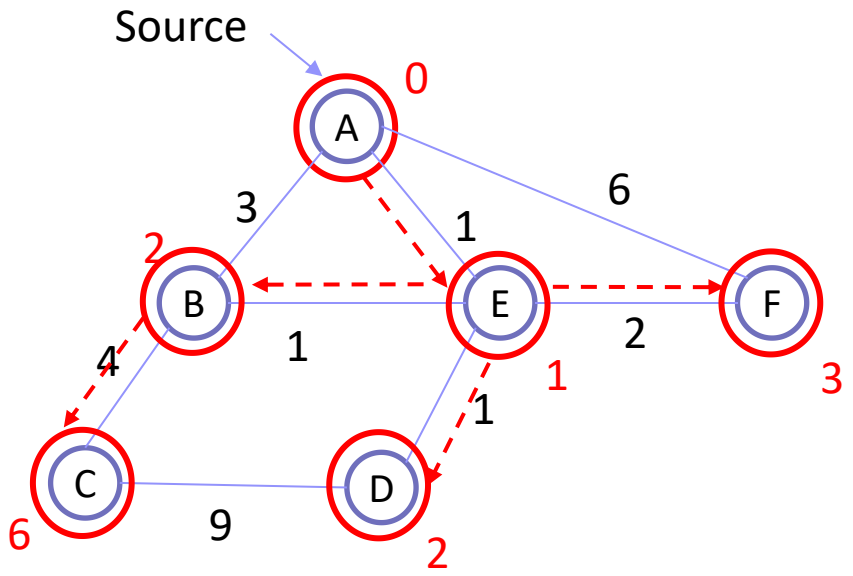


S1的转发数据库 (示例)

Dest Addr	Port
S2	1
S3	1
S4	4
S5	2
S6	5
S7	3

网络互联

- 网络路由用于连接不同的网络
  - 网络路由算法确定一个从源网络到目的网络的路径
  - 相比于交换机组网中的按MAC地址查询转发，基于IP地址的路由/转发机制具有更好聚合性，能够适用于互联网规模

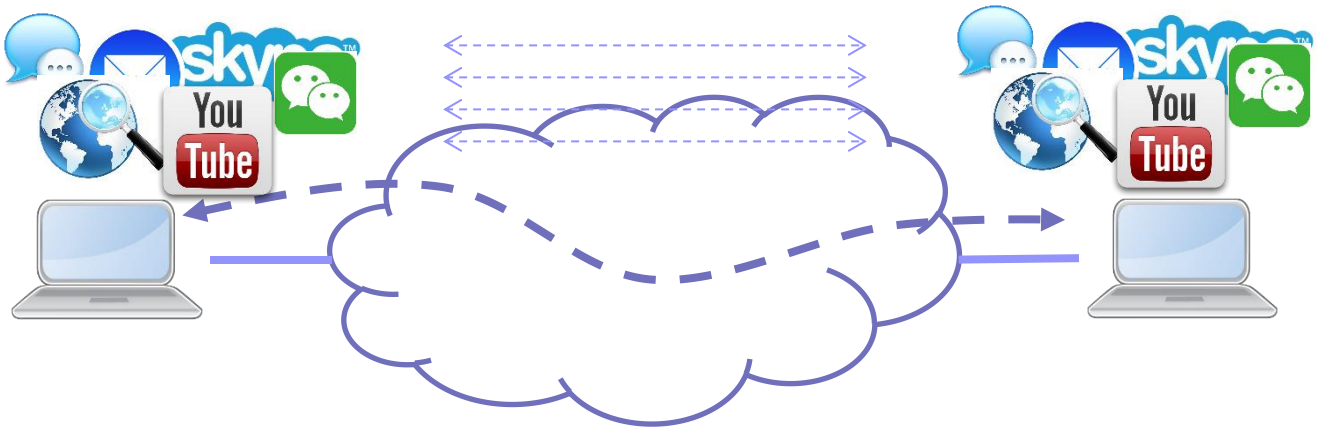


A的转发表

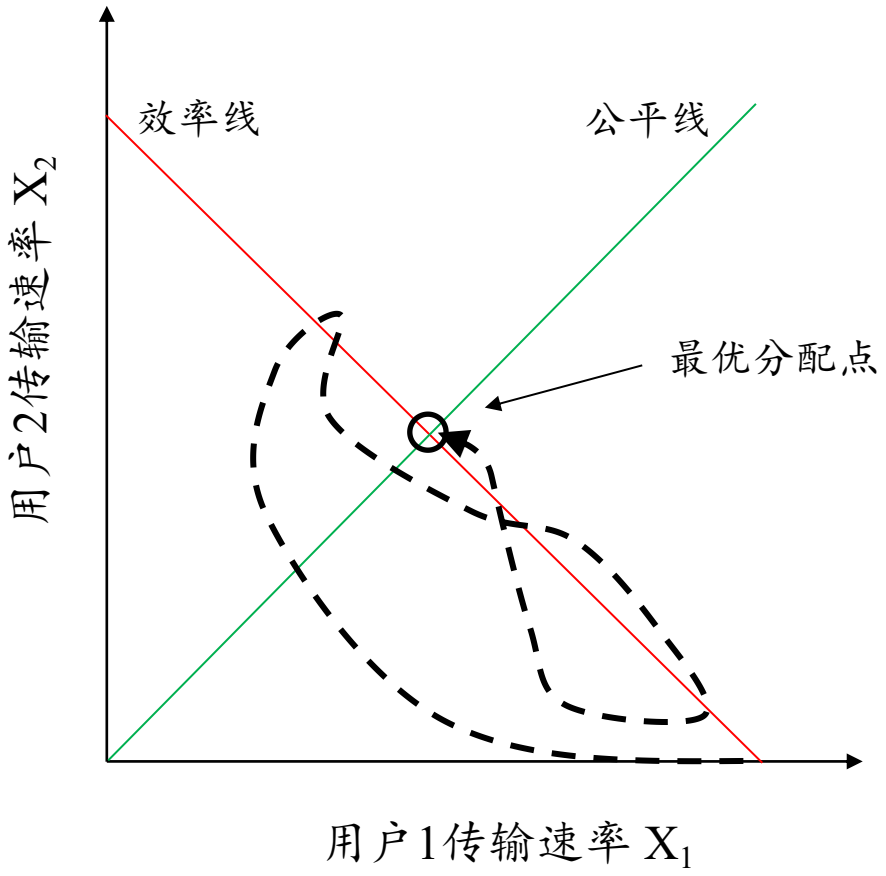
Dest Net	Cost	Next Hop
A	0	A
B	2	E
C	6	E
D	2	E
E	1	E
F	3	E

数据传输

- 网络互联提供了端到端的数据通路
  - 无连接的、尽最大努力交付（best-effort delivery）的数据报服务
- 数据传输
  - 传输控制机制：可靠、高效、公平的将数据沿路径从一端传到另一端



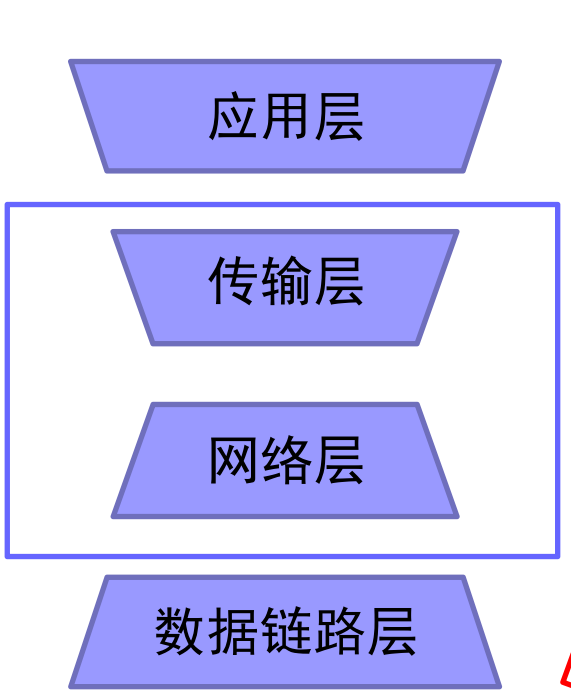
传输控制机制



- 传输控制机制的设计目标：
  1. 高效利用网络资源
     - $\max \sum X_i$
  2. 节点间公平分享网络资源
     - $\max (\sum X_i)^2/(n \sum X_i^2)$
  3. 收敛速度快

- 设计一个互联网规模的分布式传输机制是非常难的
  - 幸运的是，TCP基本实现上述目标，且稳定运行了40年

互联网体系结构层次模型



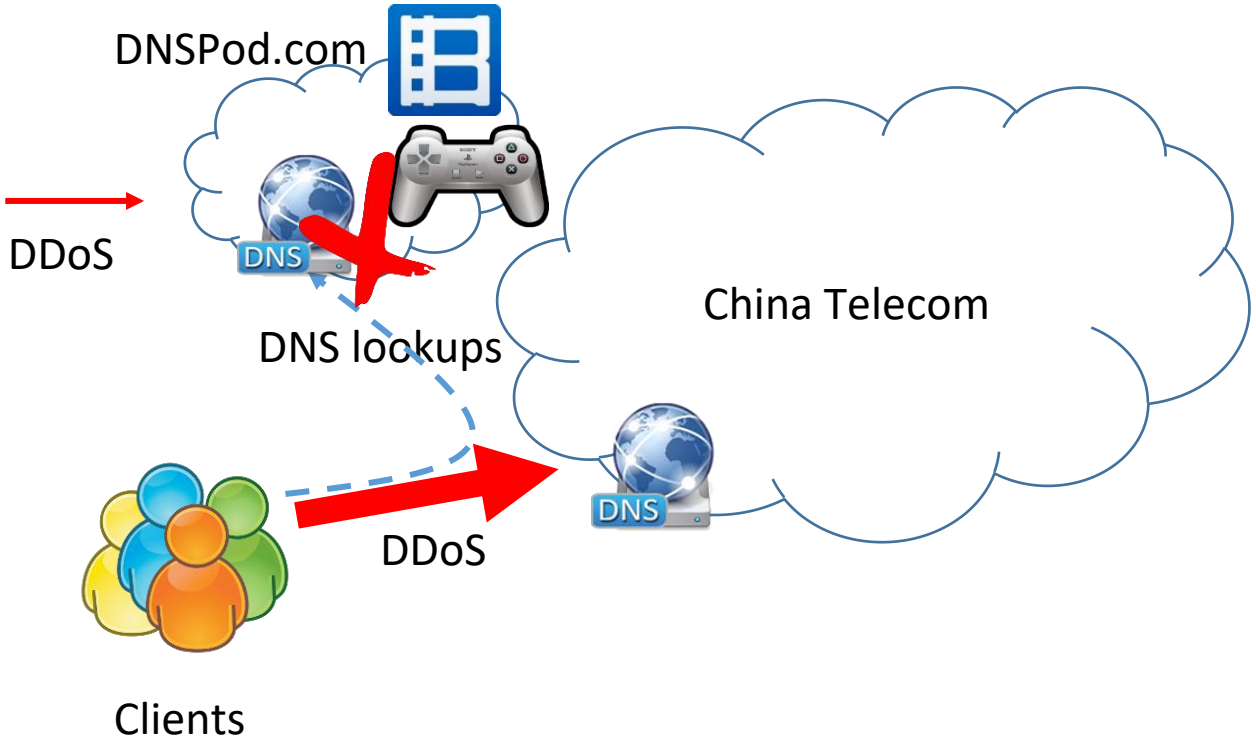
- 成千上万种应用协议
  - 大都基于Socket API开发，只需端节点支持
  - 将TCP/IP作为数据载体，在之上定义自己的通信格式
- TCP/IP
  - 占据了互联网流量的90%以上
  - 最初作为一个协议设计，后分成两个[RFC791 & RFC793]
  - 互联网初期还有其他竞争者
- 多种数据链路层协议
  - Ethernet、WiFi、Cellular Network
  - 只需要直连网络内部实现，与网络外部无关

网络安全

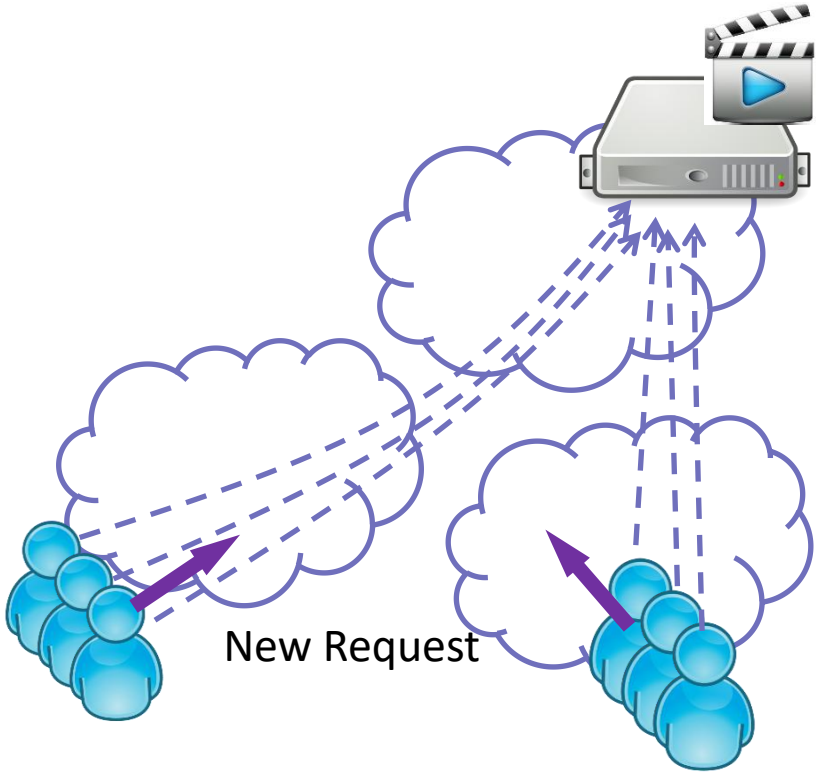
- 网络安全
  - 网络协议的安全性
    - 指具体某个网络协议设计中存在的安全问题
    - 例如，SSL/TLS再协商问题
  - 系统实现的安全性
    - 指网络系统、协议栈实现中存在的漏洞等安全问题
    - 例如，缓冲区溢出
  - 体系结构的安全性
    - 指现有网络体系结构中广泛存在的分布式拒绝服务攻击 (DDoS, Distributed Denial-of-Service)
    - DDoS攻击是现有网络体系结构与生俱来的安全问题

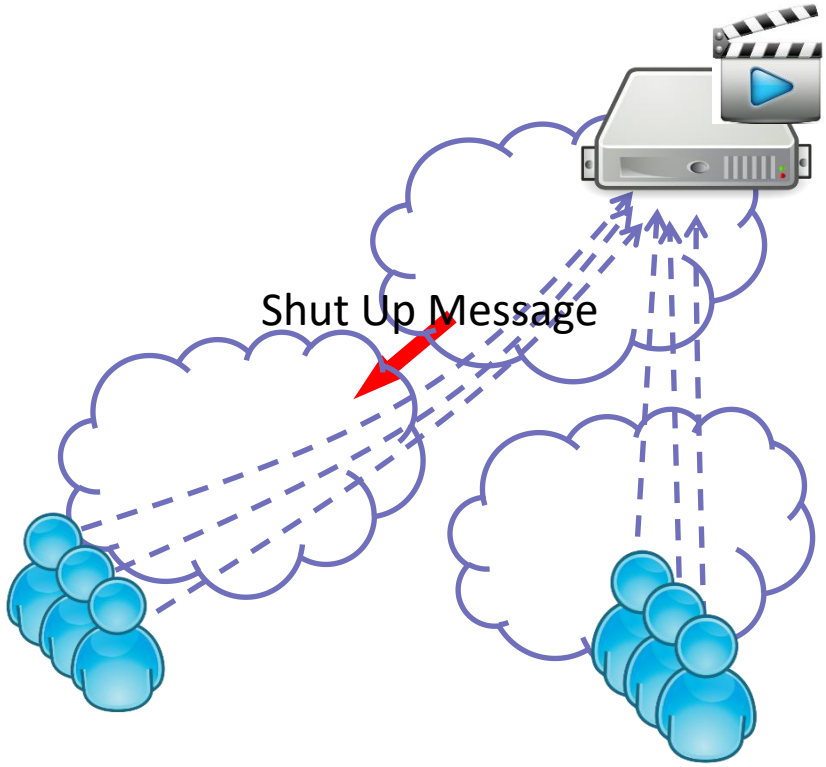
网络安全示例

- DNSPod服务器宕机导致的大规模DDoS攻击



如何缓解DDoS问题？





- 利用缓存等技术，对服务请求进行聚合，拉近用户与资源的距离 [Zhang 2010]
- 域间实现Shut Up Message机制，被攻击者发送该消息，通知对方停止发送服务请求 [Koponen 2011]

互联网传输性能

- 传输性能是互联网最核心的指标
  - 有时网络系统设计为了性能会忽略安全等其他指标

- 传输性能，既决定了用户体验，同时也影响了企业营收

	延迟	结果
Amazon	+100ms	-1.0% 营收额
Bing	+500ms	-1.2% 营收额
Google	+400ms	-0.7% 搜索量

互联网性能指标

- 不同应用有不同的性能指标



。。。

缓冲、码率、卡顿　流完成时间　吞吐率　延迟　吞吐率/延迟

- 不同层关注不同的性能指标

应用层

传输层

网络层

数据链路层

QoE (Quality of Experience): 如上，与用户感受相关

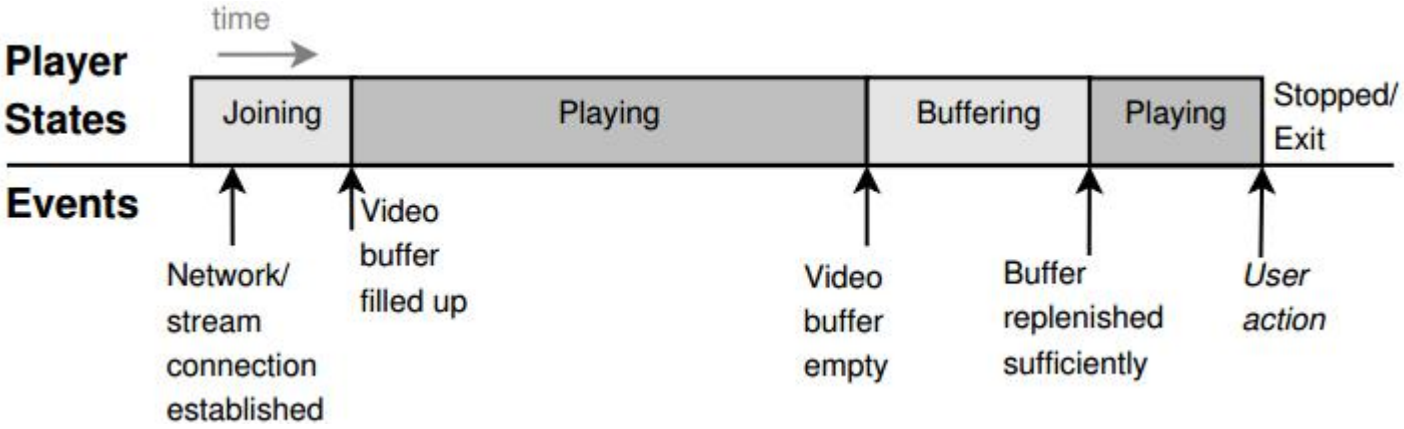
≠

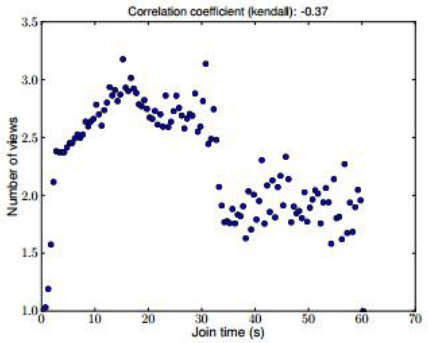
吞吐率、延迟、完成时间，。。

QoS (Quality of Service): 网络延迟、丢包、延迟抖动，。。。

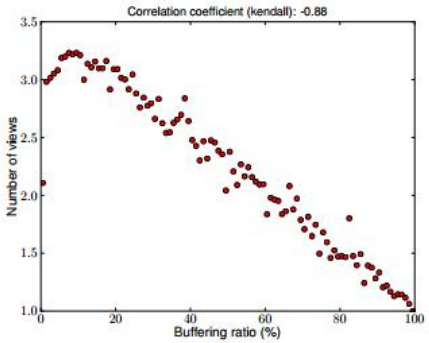
传输性能与用户参与度

- 视频传输性能对用户观看的影响 [Dobrian 2011]

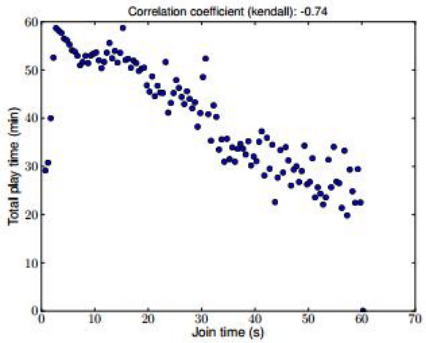




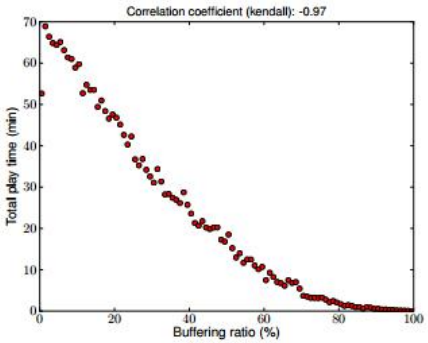
(a) Join time, # views



(b) Buffering ratio, # views



(c) Join time, Play time



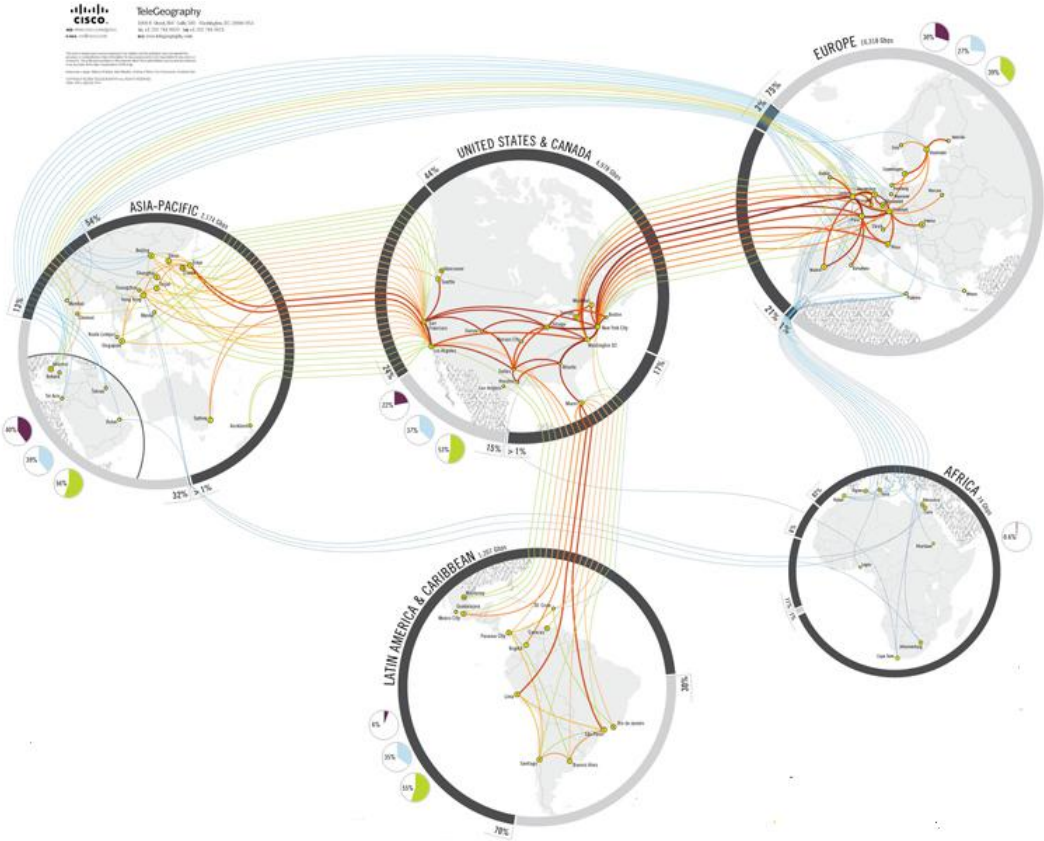
(d) Buffering ratio, Play time

理想性能模型

- 光速网络(Networking at the Speed of Light)性能模型

- 文件大小: K
- 连接距离：D
- 单位时间内的传输量： I
- 光纤传播时延: L = D / (0.6*C)

- 理想性能
  - T = K / I * L
- 实际性能
  - T’ = (10 ~ 100) * T



[www.telegeography.com]

传输性能的例子

- ping www.thesaurus.com
  - IP地址： 184.50.87.107，RTT：40毫秒
- wget http://www.thesaurus.com/
  - 文件大小：52KB，用时：0.6秒

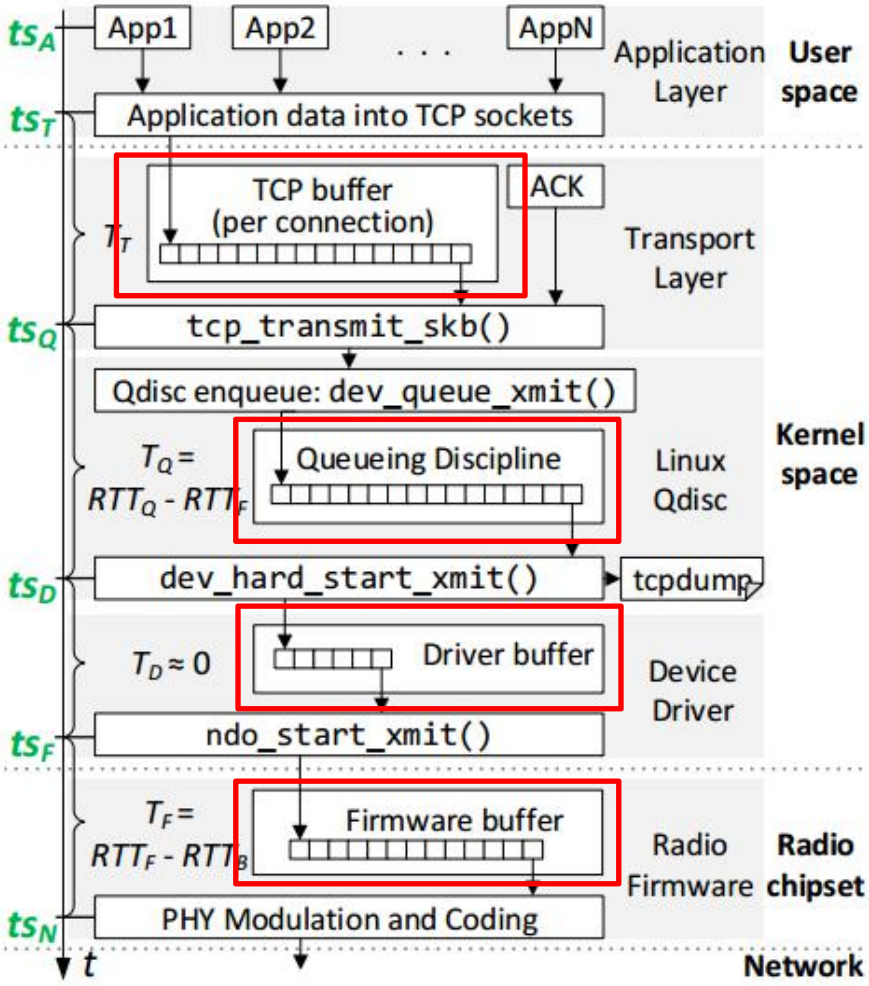
- 性能比较
  - 源、目的节点间的距离不超过100KM，直线光纤的往返时延不超过1毫秒
  - 按光速网络模型计算，时延性能相差40倍，传输性能相差600倍

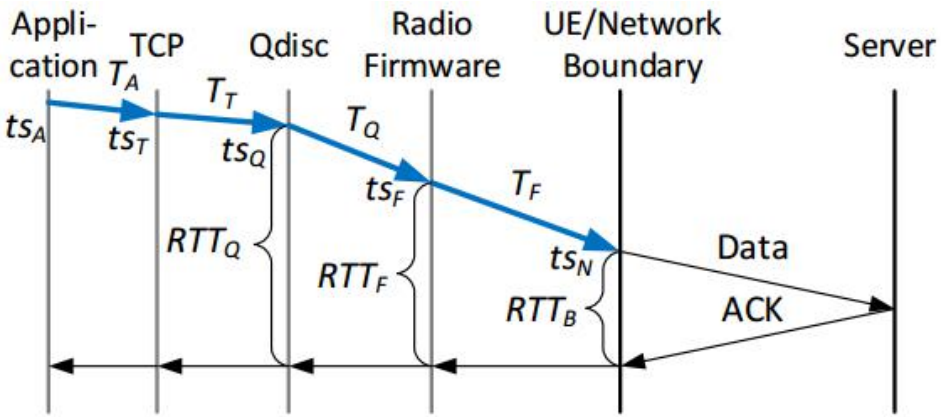
网络延迟

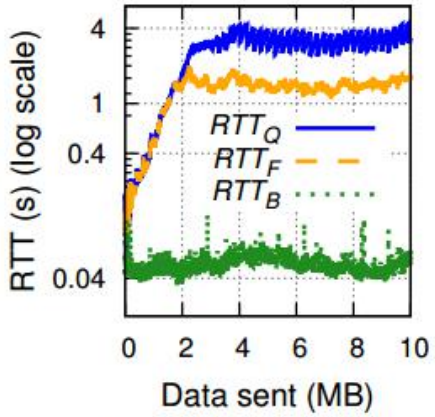
- 网络延迟 = 传播时延 + 发送延迟 + 队列延迟

- 传播时延 := 传播距离 / (0.6 x 光速)
  - 传播距离通常大于地理距离
- 发送延迟 := 数据从端节点进入到传输介质所需要的时间
  - 发送端设备的发送延迟最多可达4秒 [Guo 2016]
- 队列延迟 := 队列长度 / 处理速度
  - 网络中间设备为了减少丢包而增大Buffer，造成了队列延迟增大，该问题叫做BufferBloat [Gettys 2011]

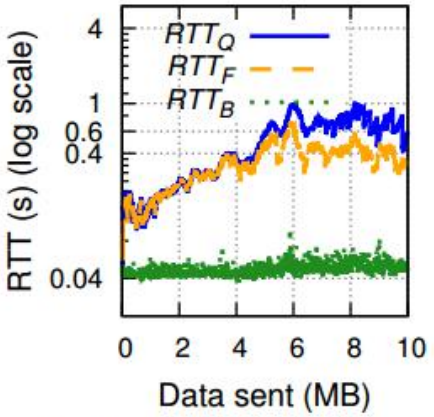
Android/Linux设备的发送延迟







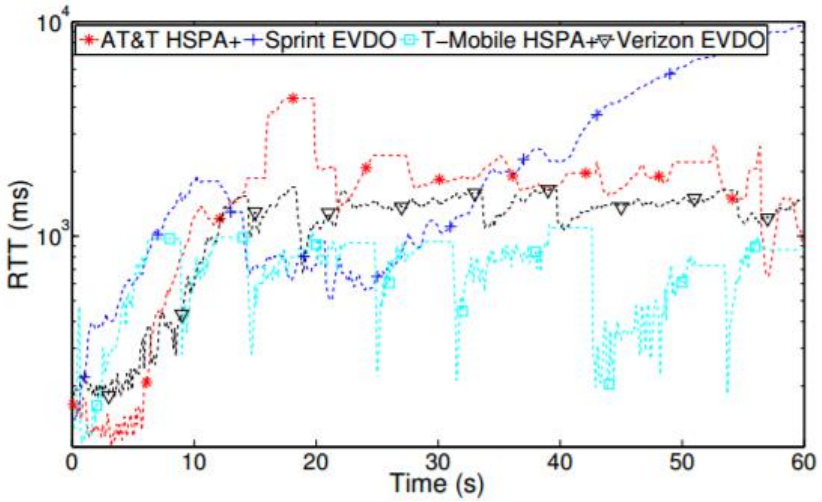
(a) Uplink BW 2Mbps



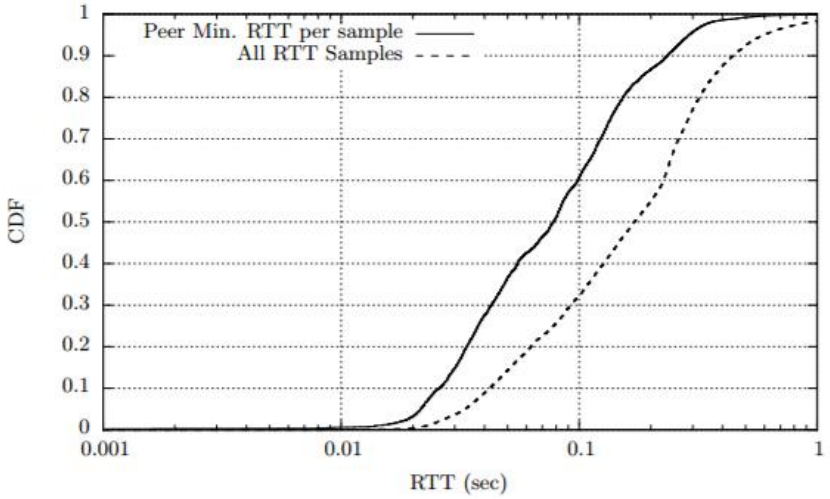
(b) Uplink BW 8Mbps

[Guo 2016]

队列延迟引起的BufferBloat问题



移动数据网络中的BufferBloat问题 [Jiang 2012]

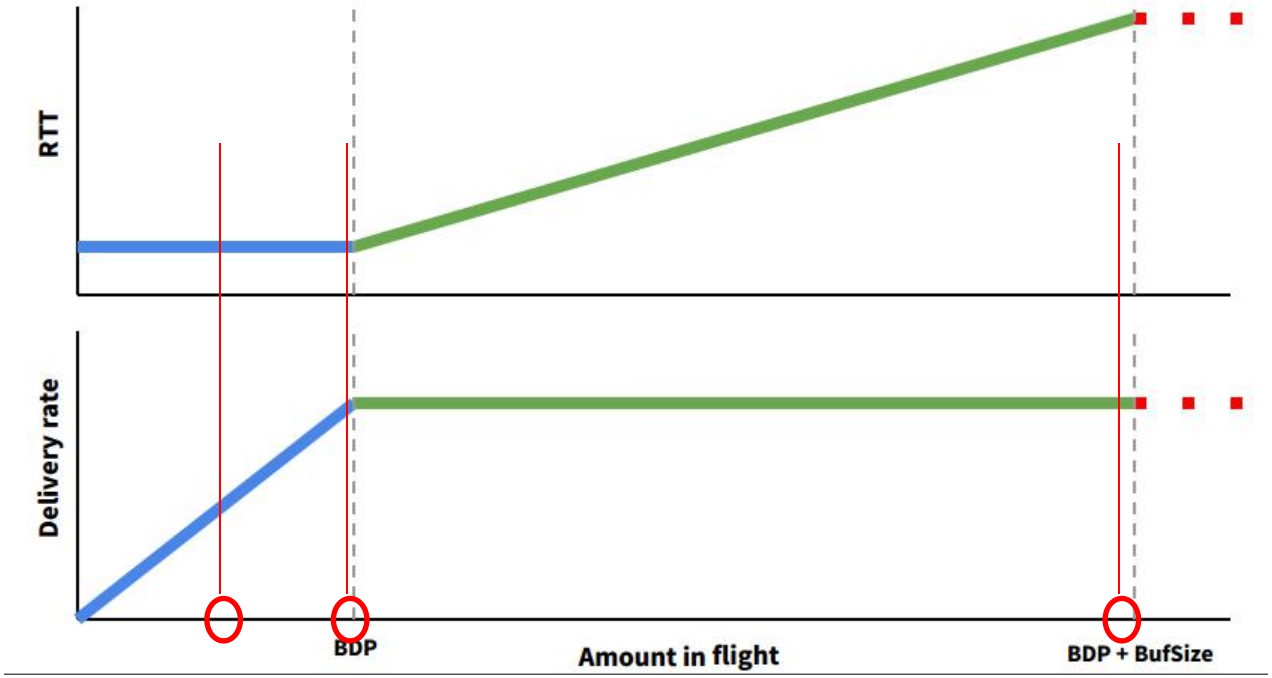


Internet边缘网络的RTT分布 [Allman 2013]

- 移动网络的BufferBloat问题比固网中更严重
  - 移动网络为了达到QoS要求，在基站部署了大量的buffer
- BufferBloat问题具有瞬时性
  - 不会一直存在；一旦发生，对网络传输性能影响非常大
  - BufferBloat通常对延迟敏感的流（例如，Web搜索）影响非常大，对以吞吐率为目标的流（例如，大文件下载）影响不大

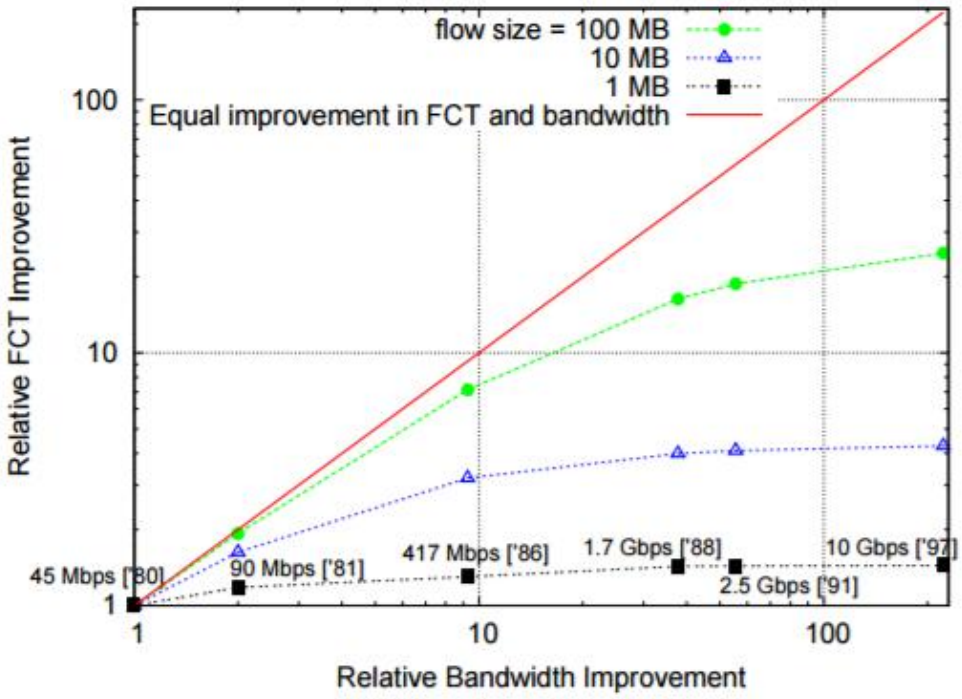
Buffer与传输性能

- 给定带宽和延迟，端节点单位时间内可发送的数据是固定的
  - BDP = Bw * Delay（Bw-Delay Product）
  - 发送方单位时间内发送的数据叫做在途数据量（inflight）
  - Inflight值在合理范围内，才能同时获得高吞吐率与低延迟 [Cardwell 2016]



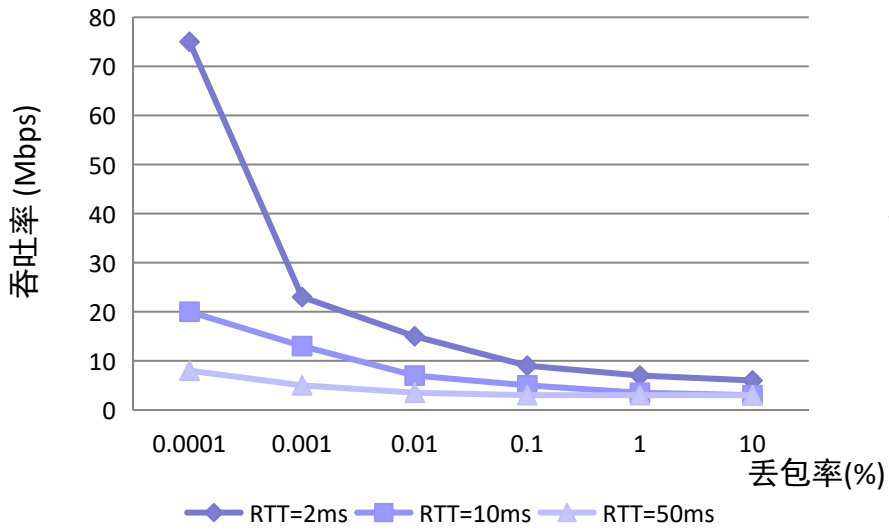
吞吐率与流完成时间

- 为什么吞吐率不能刻画所有应用的性能？[Dukkipati 2006]
  - 传输控制协议TCP使用慢启动机制探测可用网络带宽
  - 对于较小的传输文件，在未探测到可用带宽上限时传输已经结束
  - Log2(K / 3pkts) * RTT vs K/B * RTT

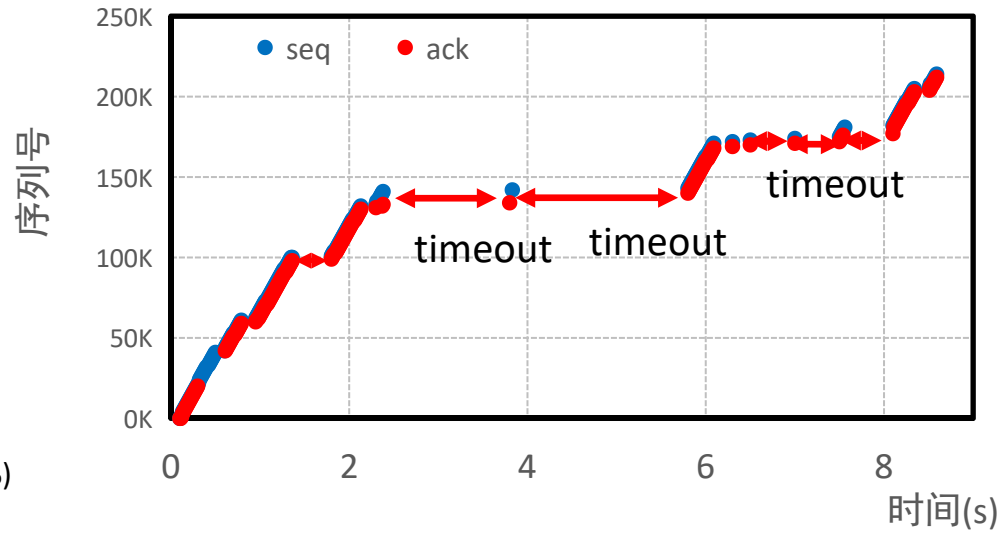


TCP传输的性能问题

- 现有TCP的传输控制策略过于保守
  - 拥塞避免：判断网络拥塞后，发送速率减半
  - 快速重传：后发的数据包先收到确认，判断网络丢包



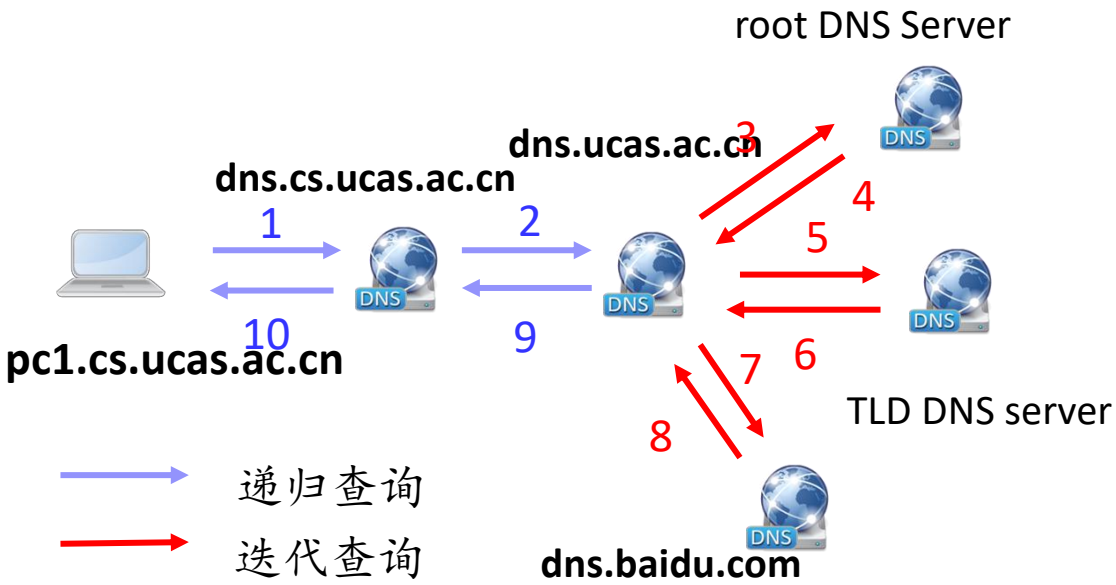
TCP在不同网络环境的吞吐率

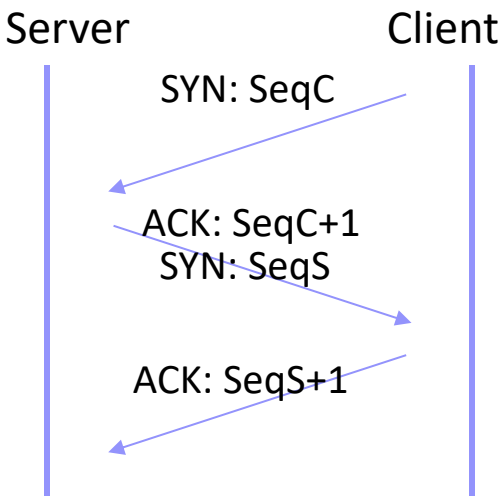


TCP丢包恢复效率的例子

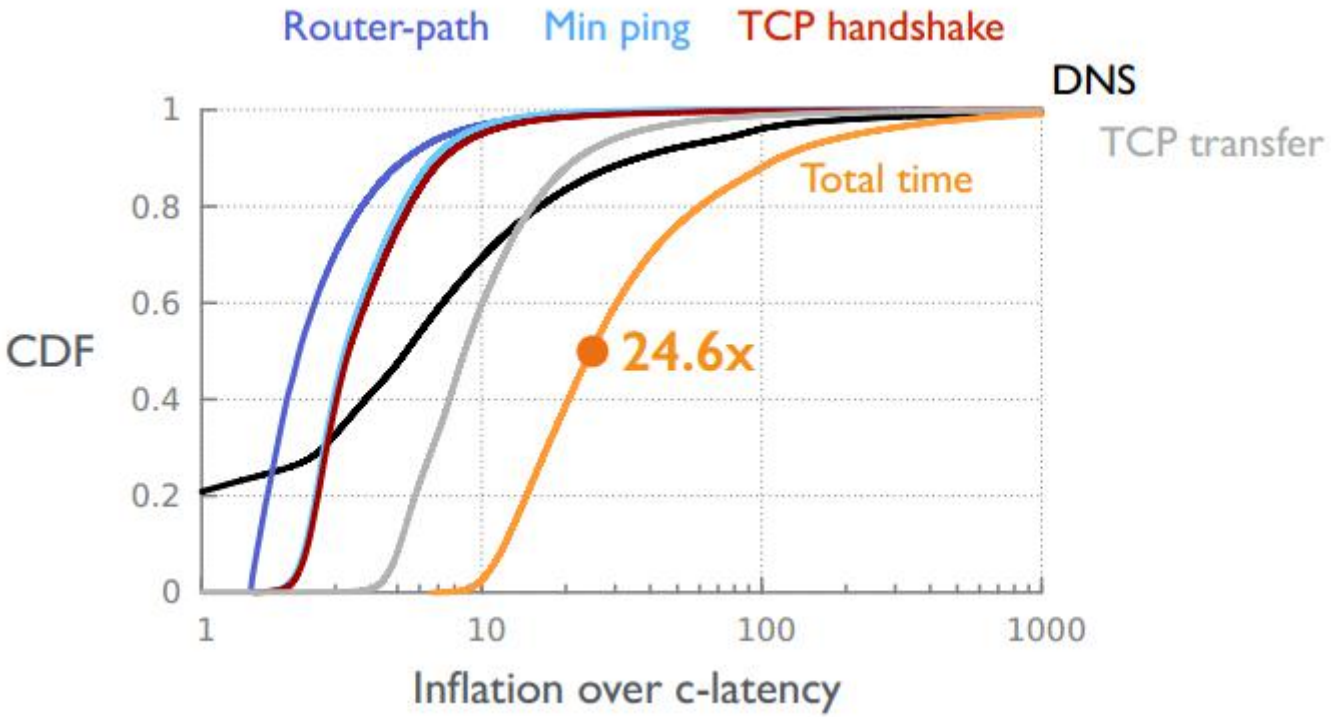
传输中的其它开销

- 数据传输前的流程
  - DNS解析：将域名解析到特定的IP地址，通常花费1个RTT
  - TCP三次握手：建立连接，需要花费1个RTT





现实网络中的传输性能



Total time (24.6x) = DNS resolution (5.4x) + TCP handshake (3.2x) + TCP transfer (8.7x)

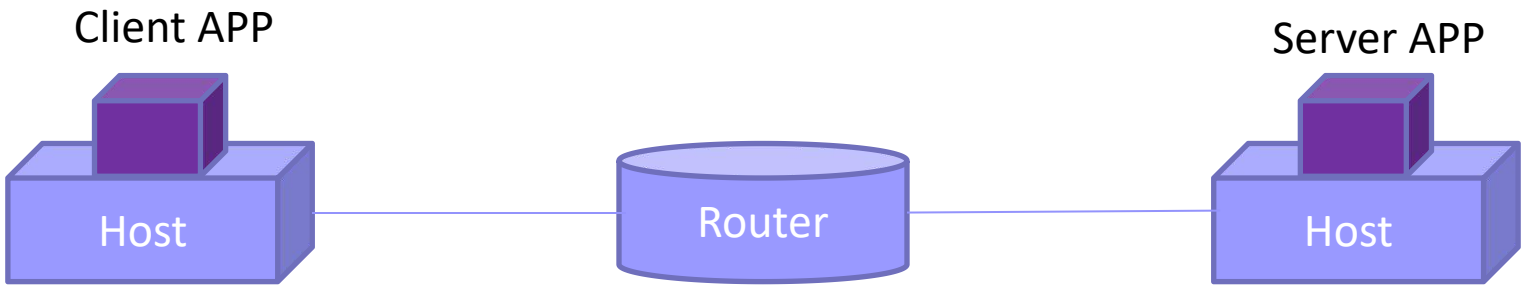
[Singla 2014]

小结

- 互联网是一个分层设计的分布式网络系统
  - 现有体系结构模型不是一蹴而就，而是逐渐演化来的
- 网络测量，是互联网研究领域的重要工作
  - 是认识互联网的有效途径
  - 是性能分析、诊断、资源调度、网络规划的重要前提
- 互联网传输性能至关重要
  - 决定了用户体验，影响服务营收
  - 受很多因素影响，有较大的改进空间

网络实验：搭建一个简单的网络

- 通过路由器（Router）将两台主机互连，每台主机上运行相应的网络程序



网络实验平台

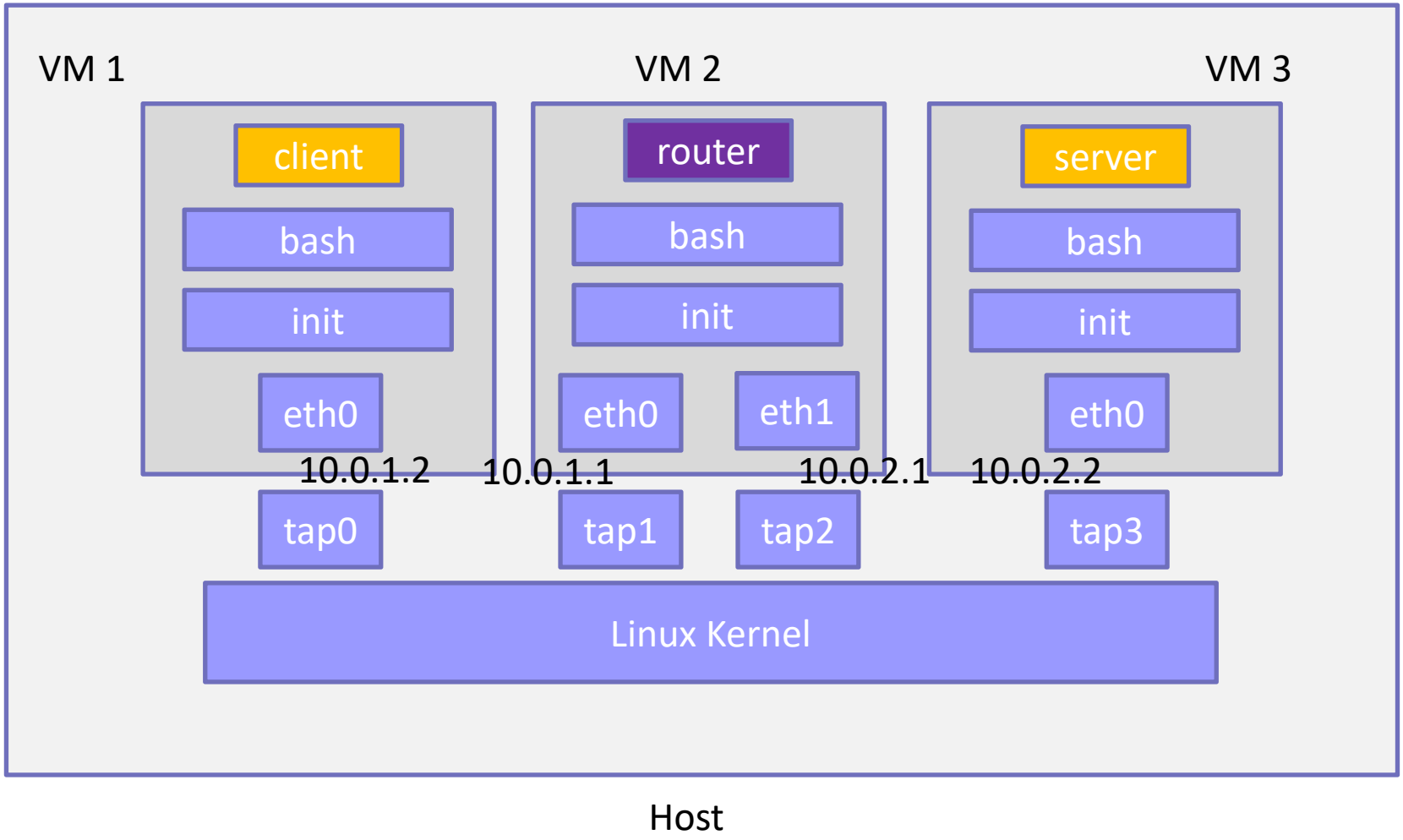
平台	优点	缺点
硬件网络平台	真实；高效	价格昂贵；不易配置；不易扩展；
模拟平台（Simulator）	软件实现；快速实验；容易扩展；	模拟结果可能和实际结果差别较大
仿真平台（Emulator）	软件实现；容易扩展	比硬件平台速度稍慢

网络仿真平台

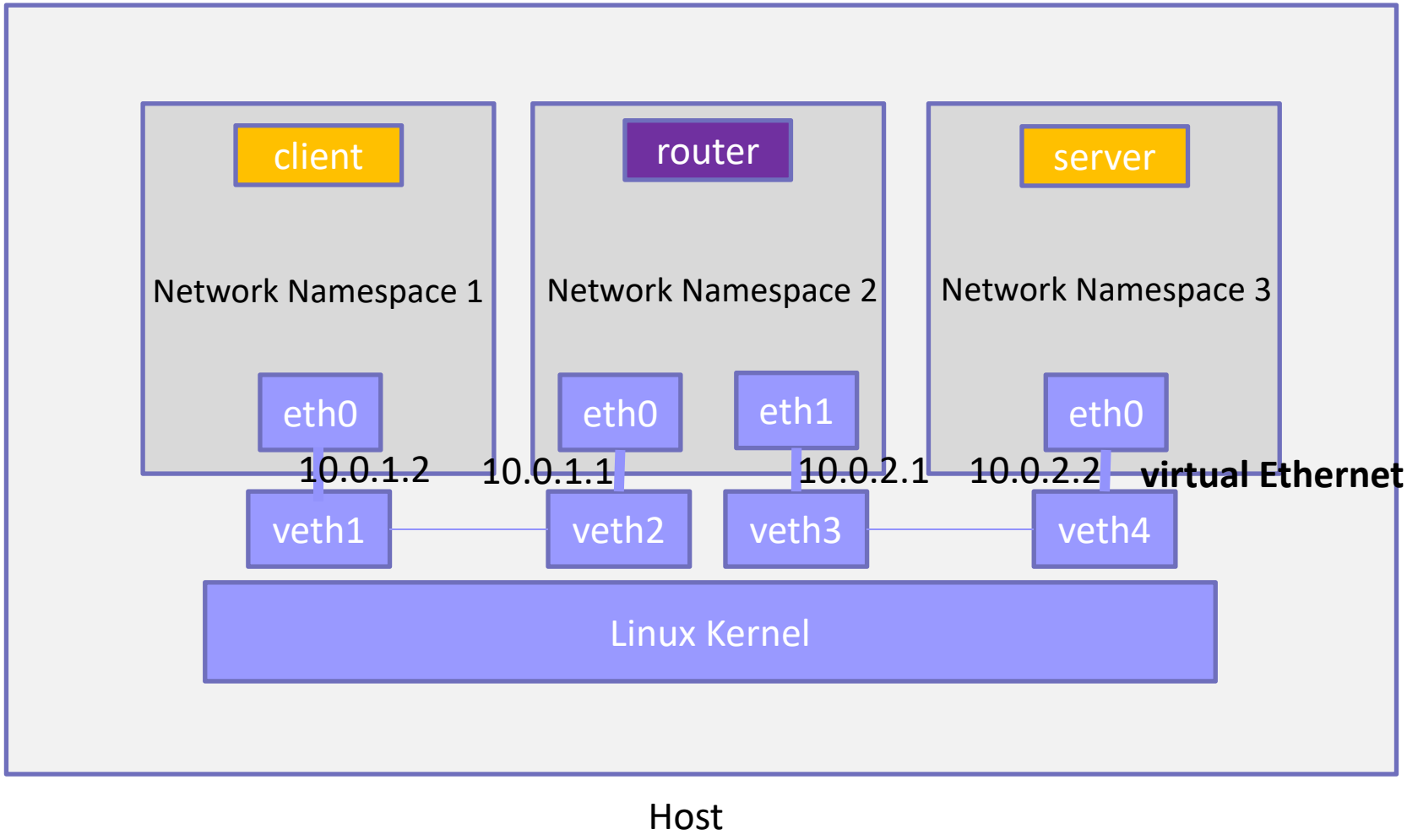
- 借助虚拟化技术，在物理机器上虚拟出多个节点，不同节点间通过虚拟链路（例如 open vswitch）互连

- 相比于硬件网络平台：
  - 成本低、部署快、可扩展
- 相比于网络模拟器：
  - 更接近真实网络结果

全虚拟化技术

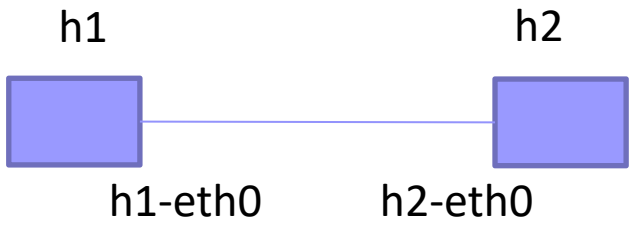

VM 1
VM 2
VM 3
client
router
server
bash
bash
bash
init
init
init
eth0
eth0
eth1
eth0
10.0.1.2
10.0.1.1
10.0.2.1
10.0.2.2
tap0
tap1
tap2
tap3
Linux Kernel
Host

网络命名空间(Network Namespace)技术



基于网络命名空间的网络环境搭建

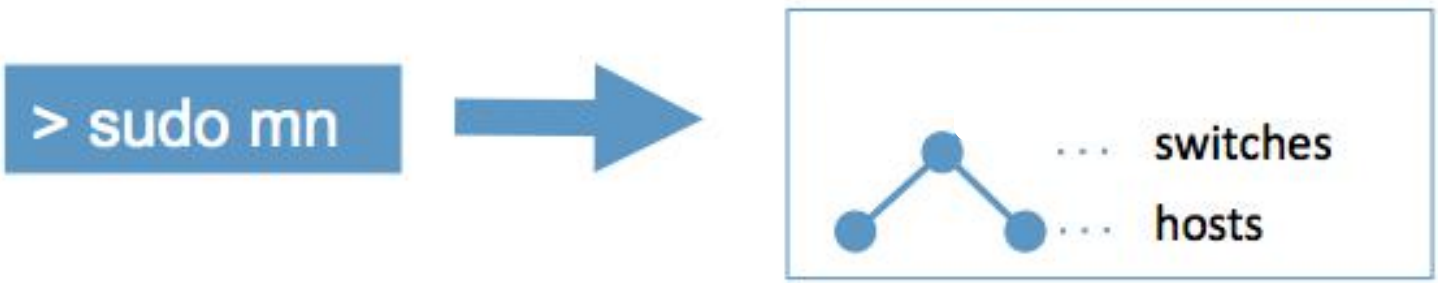
```
# Create link
sudo ip link add name h1-eth0 type veth peer name h2-eth0 netns 1

# Create host namespaces
sudo ip netns add h1
sudo ip netns add h2

# Move host ports into namespaces
sudo ip link set h1-eth0 netns h1
sudo ip link set h2-eth0 netns h2
sudo ip netns exec h1 ip link show
sudo ip netns exec h2 ip link show

# Configure ports
sudo ip netns exec h1 ifconfig h1-eth0 10.0.0.1
sudo ip netns exec h2 ifconfig h2-eth0 10.0.0.2

# Test connectivity
sudo ip netns exec h1 ping 10.0.0.2
```



Mininet环境

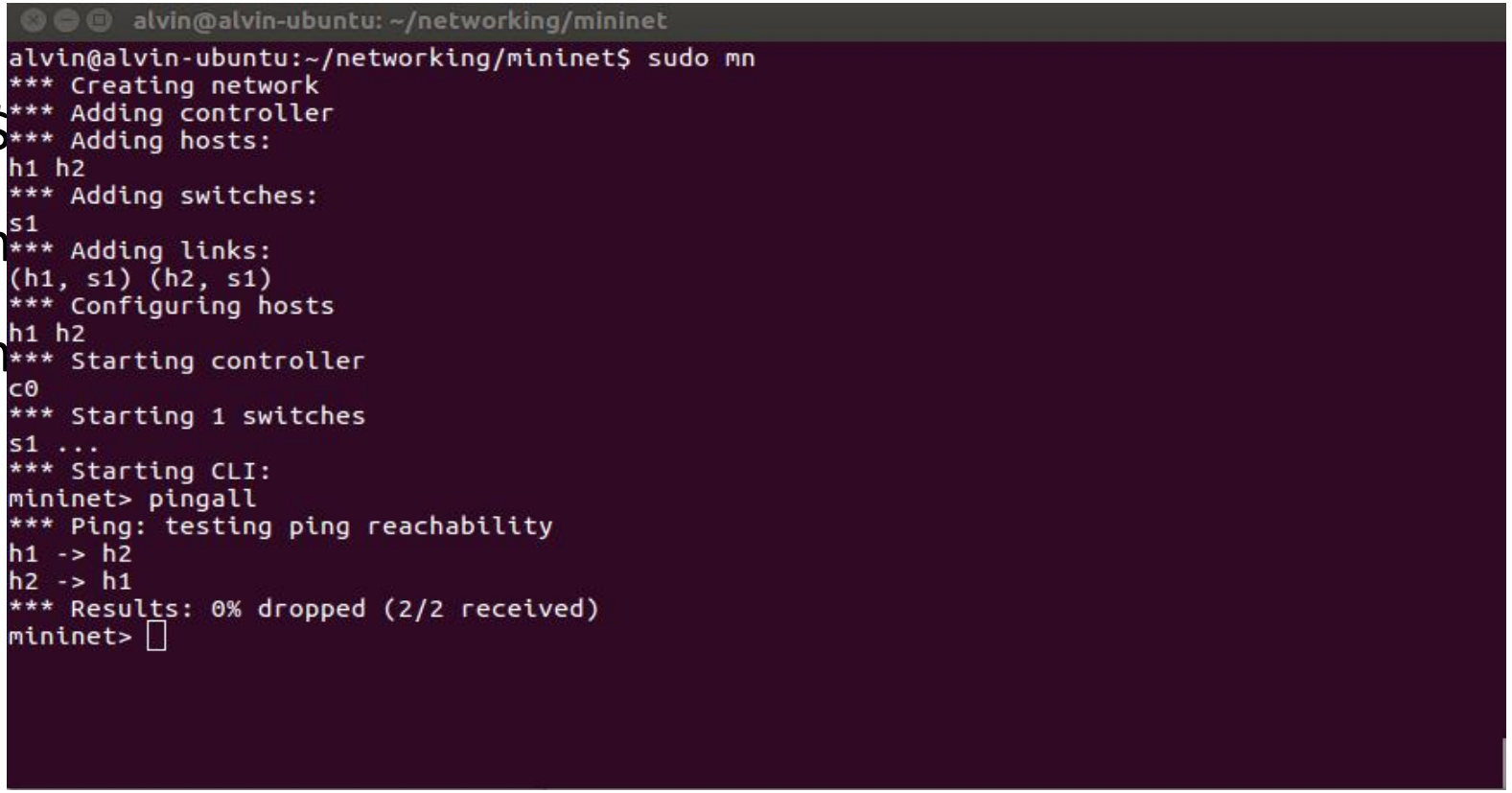
- Mininet是基于Linux Network Namespace的python封装

- 支持python API以及命令行工具（CLI）
- 可以很方便的创建拓扑、设置网络条件、运行网络程序
- 支持不同层次的抽象和语法



Mininet安装

- $ sudo apt install mininet
- $
- m
- m



Mininet API举例

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.cli import CLI
from time import sleep

net = Mininet()
h1 = net.addHost('h1')
h2 = net.addHost('h2')
net.addLink(h1, h2)
net.start()
h2.cmd('python -m SimpleHTTPServer 80 &')
sleep(2)
h1.cmd('wget %s -O result.txt' % (h2.IP()))
net.stop()
```

Mininet支持设置性能参数

```
from mininet.node import CPULimitedHost
from mininet.link import TCLink

# Use performance-modeling link and host
net = Mininet(link=TCLink, host=CPULimitedHost)

# Limit CPU
net.addHost('h1', cpu=0.2)

# Set link bandwidth, delay and loss rate
net.addLink(h2, s1, bw=10, delay='50ms', loss=2)
```

Mininet CLI举例

```
$ sudo mn
mininet> xterm h1 h2
h2# python -m SimpleHTTPServer 80
h1# wget 10.0.0.2

$ sudo mn --topo tree,depth=3,fanout=3 --link=tc,bw=10

$ sudo mn --topo linear,20

# to test this, you need to implement custom.py
$ sudo mn --custom custom.py --topo mytopo
```

Mininet自定义网络拓扑

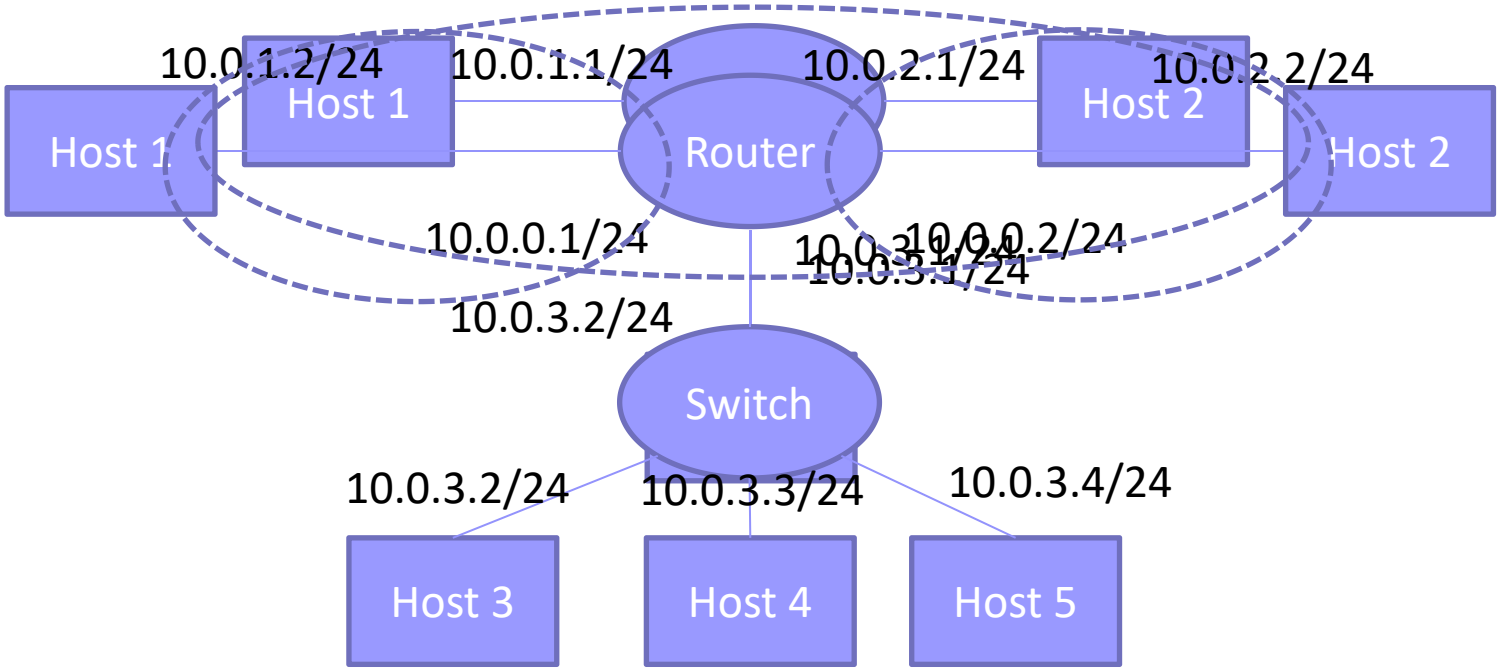
$ cat custom.py

```
---------------------------------------------------------------
from mininet.topo import Topo

class StarTopo(Topo):
    “Star Topology"
    def build(self, count=10):
        hosts = [ self.addHost('h%d' % i) for i in range(1, count + 1) ]
        s1 = self.addHost('s1')
        for h in hosts:
            self.addLink(h, s1)

topos = { 'mytopo': StarTopo }
---------------------------------------------------------------
```

$ sudo mn --custom custom.py --topo mytopo, 20

网络环境搭建

- 网络由网络前缀(e.g. 192.168.0.0/24)来表示
- 交换机(Switch)用于组网
- 路由器(Router)连接不同网络



网络管理工具

- # ifconfig eth0 10.0.0.1/24 # set ip address & netmask
- # route add default gw 10.0.0.2 # set default gateway
- # route add 10.0.1.0/24 gw 10.0.3.1 dev h1-eth0 # set gateway
- # arp -n # show ip->mac mapping
- # arp -d 10.0.0.1 # delete the entry of 10.0.0.1
- # nslookup www.baidu.com # dns lookup

网络测量工具

- # ping 10.0.2.2 # connectivity & RTT
- # traceroute 10.0.2.2 # hops to the destination
- # iperf # bandwidth measurement
  - 10.0.0.1 # iperf -s
  - 10.0.0.2 # iperf -c 10.0.0.1

互联网协议实验

- 搭建实验环境
  - $ sudo apt install wireshark
  - $ sudo mn --nat # allows hosts to connect with the Internet
  - mininet> xterm h1
  - h1 # echo "nameserver 1.2.4.8" > /etc/resolv.conf
  - h1 # wireshark &
- 实验步骤
  - h1 # wget www.ucas.ac.cn

互联网协议实验

- 观察wireshark输出 （一）， 以www.baidu.com页面为例
  - ARP协议: IP地址->MAC地址映射
  - DNS协议: 域名->IP地址映射
  - TCP协议: 数据传输

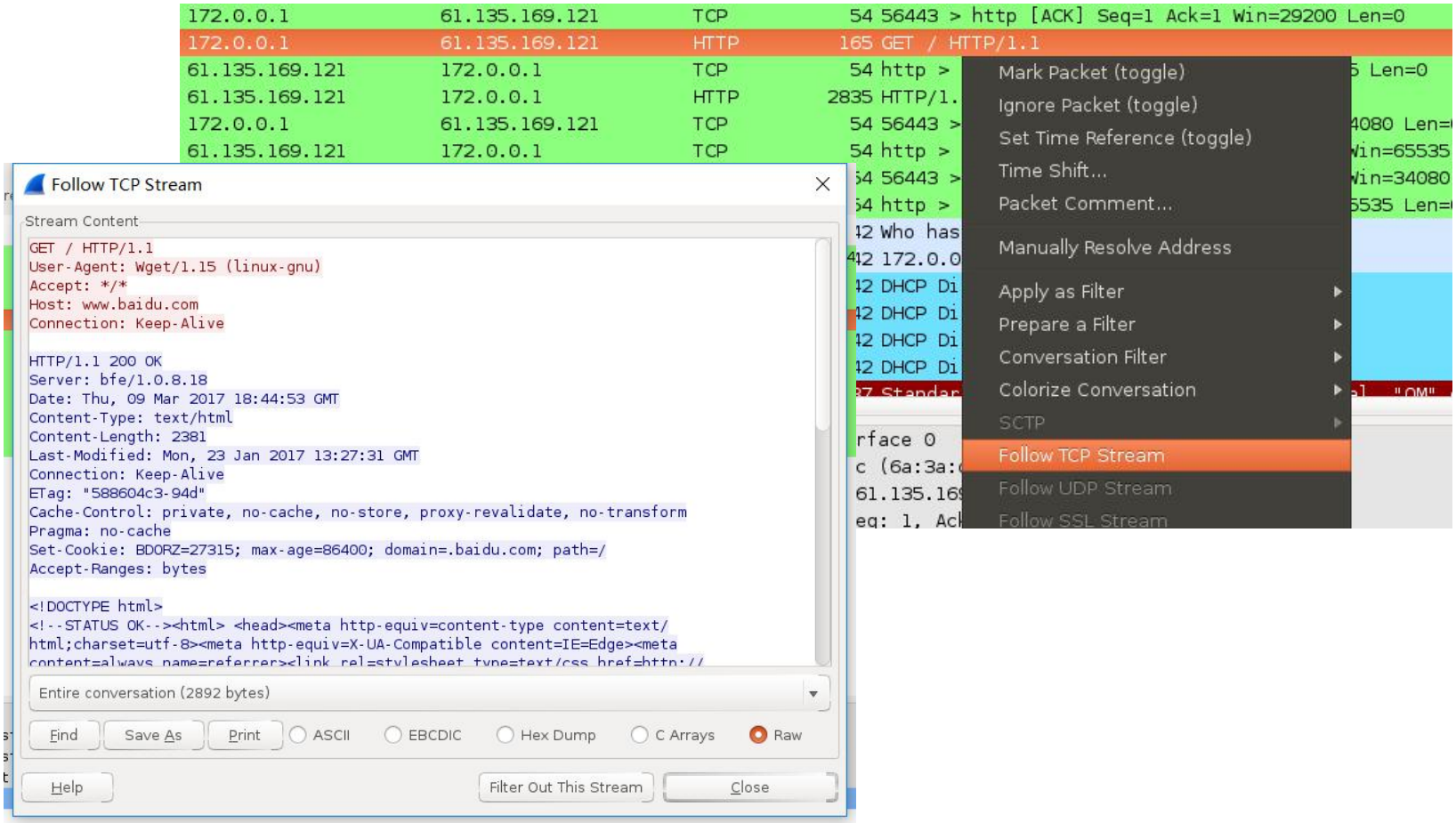
No.	Time	Source	Destination	Protocol	Length	Info
4	9.256069038	82:24:2a:96:c2:c8	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.1
5	9.259799814	72:e0:11:58:ce:4f	82:24:2a:96:c2:c8	ARP	42	10.0.0.3 is at 72:e0:11:58:ce:4f
6	9.259812291	10.0.0.1	159.226.39.1	DNS	73	Standard query 0x0393 A www.baidu.com
7	9.259815360	10.0.0.1	159.226.39.1	DNS	73	Standard query 0xd6fe AAAA www.baidu.com
8	9.308805004	159.226.39.1	10.0.0.1	DNS	132	Standard query response 0x0393 A www.baidu.com CNAME w
9	9.318736091	159.226.39.1	10.0.0.1	DNS	157	Standard query response 0xd6fe AAAA www.baidu.com CNAM
10	9.319227342	10.0.0.1	119.75.216.20	TCP	74	48488 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_P
11	9.338055235	119.75.216.20	10.0.0.1	TCP	58	80 → 48488 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
12	9.338087433	10.0.0.1	119.75.216.20	TCP	54	48488 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
13	9.341933425	10.0.0.1	119.75.216.20	HTTP	194	GET / HTTP/1.1
14	9.342532422	119.75.216.20	10.0.0.1	TCP	54	80 → 48488 [ACK] Seq=1 Ack=141 Win=65535 Len=0
15	9.353627861	119.75.216.20	10.0.0.1	TCP	454	[TCP segment of a reassembled PDU]
16	9.353647250	10.0.0.1	119.75.216.20	TCP	54	48488 → 80 [ACK] Seq=141 Ack=401 Win=30016 Len=0
17	9.362959249	119.75.216.20	10.0.0.1	TCP	1474	[TCP segment of a reassembled PDU]
18	9.362983464	10.0.0.1	119.75.216.20	TCP	54	48488 → 80 [ACK] Seq=141 Ack=1821 Win=32660 Len=0
19	9.363141969	119.75.216.20	10.0.0.1	HTTP	1015	HTTP/1.1 200 OK (text/html)
20	9.363161257	10.0.0.1	119.75.216.20	TCP	54	48488 → 80 [ACK] Seq=141 Ack=2782 Win=35500 Len=0
21	9.363337625	119.75.216.20	10.0.0.1	TCP	54	80 → 48488 [FIN, ACK] Seq=2782 Ack=141 Win=65535 Len=0
22	9.366398955	10.0.0.1	119.75.216.20	TCP	54	48488 → 80 [FIN, ACK] Seq=141 Ack=2783 Win=35500 Len=0
23	9.367574703	119.75.216.20	10.0.0.1	TCP	54	80 → 48488 [ACK] Seq=2783 Ack=142 Win=65535 Len=0

互联网协议实验

- 观察wireshark输出 （二）
  - 不同层次的协议封装
- Ethernet < IP < UDP < DNS
- Ethernet < IP < TCP < HTTP

互联网协议实验

- 观察wireshark输出（三）
  - TCP承载HTTP协议



流完成时间实验

- 搭建实验环境：
  - $ sudo python fct_exp.py
  - mininet> xterm h1 h2
- 实验步骤：
  - h2 # dd if=/dev/zero of=1MB.dat bs=1M count=1
  - h1 # wget http://10.0.0.2/1MB.dat

流完成时间实验

- fct_exp.py脚本

```
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.node import OVSBridge

class MyTopo(Topo):
    def build(self):
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        self.addLink(h1, h2, bw=10, delay='10ms')

topo = MyTopo()
net = Mininet(topo = topo, switch = OVSBridge, link = TCLink, controller=None)

net.start()
h2 = net.get('h2')
h2.cmd('python -m SimpleHTTPServer 80 &')
CLI(net)
h2.cmd('kill %python')
net.stop()
```

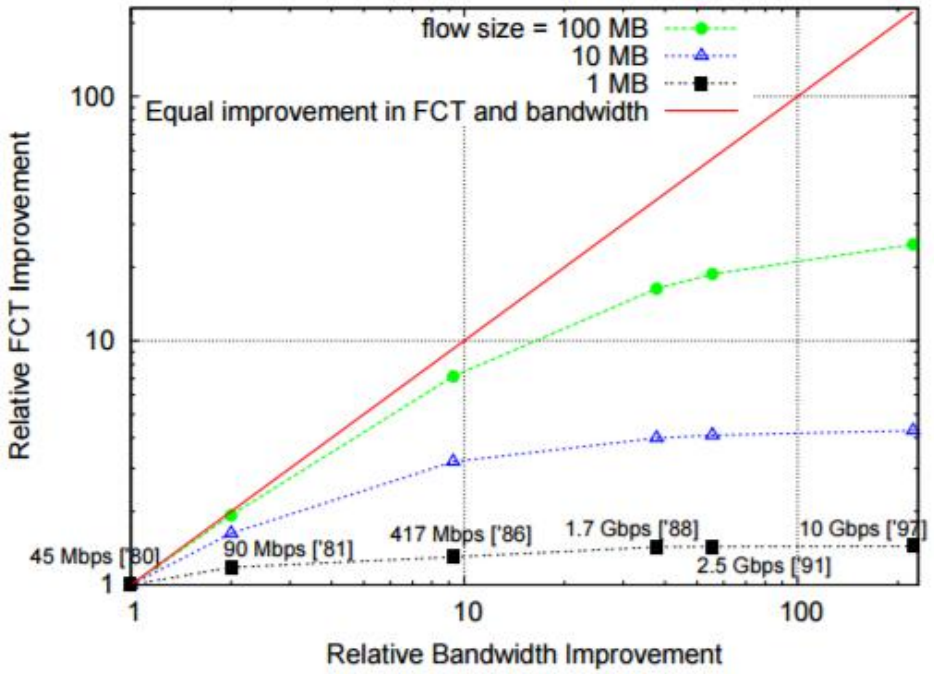
可调节参数，bw单位为Mbps

流完成时间实验

- 观察实验结果
  - 在给定带宽、延迟和文件大小前提下，查看流完成时间
  - 变化文件大小(10MB, 100MB)、带宽(10Mbps, 100Mbps, 1Gbps)、延迟(10ms, 100ms)，查看不同条件下的流完成时间

流完成时间实验

- 利用fct_exp.py脚本，重现下图中的实验结果
- 文件大小: 1MB, 10MB, 100MB
- 带宽：10Mbps, 50Mbps, 100Mbps, 500Mbps, 1Gbps
- 延迟：100ms

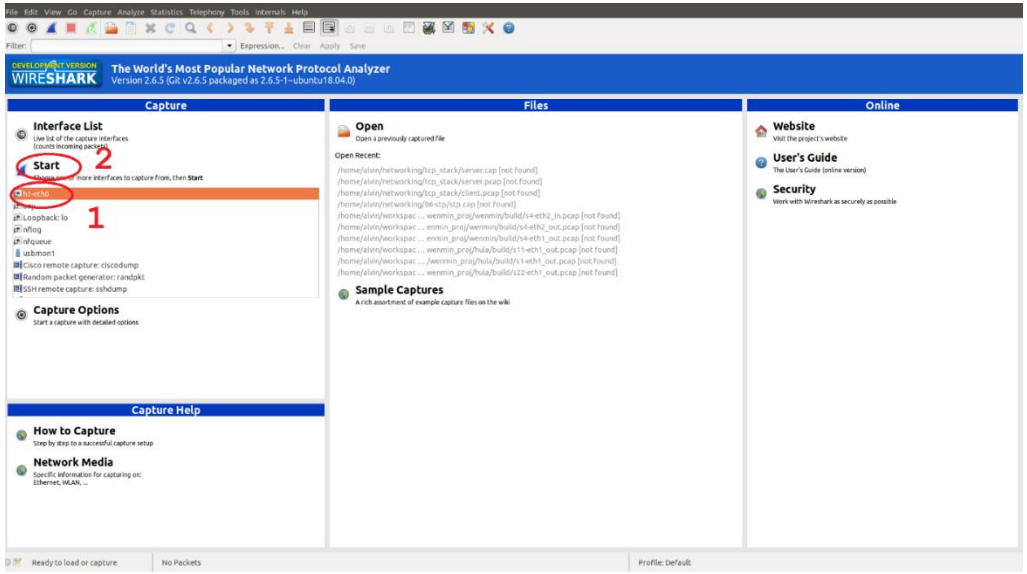


实验内容

- 互联网协议实验
  - 在节点h1上开启wireshark抓包，用wget下载www.ucas.ac.cn页面
  - 调研说明wireshark抓到的几种协议
    - ARP, DNS, TCP, HTTP, HTTPS
  - 调研解释h1下载ucas页面的整个过程
    - 几种协议的运行机制
- 流完成时间实验
  - 利用fct_exp.py脚本复现上页幻灯片中的图
    - 每个数据点做5次实验，取均值
  - 调研解释图中的现象
    - 提示：TCP传输、慢启动机制

实验注意事项

- Ubuntu发行版中默认不包括xterm，需要单独安装
  - sudo apt install xterm
- wireshark启动时提示init.lua脚本错误
  - 可将配置文件中相应行注释掉，或直接忽略
- 抓包时，先选中相应网口（例如,h1-eth0），再启动/Start



实验报告

- 提交内容
  - 实验代码（本次实验不需要代码），提交到OJ网站，下次课讲
  - 实验报告：以作业形式提交到SEP网站，模板不限，内容包括但不限于实验题目、实验内容、实验流程、实验结果及分析

小结

- Mininet是一种基于Linux容器/虚拟化技术的网络仿真器

- 相比于硬件网络平台和网络模拟器，Mininet的优点
  - 对硬件要求较低、速度较快、可以支持较大规模拓扑
  - 具有与硬件网络平台类似的精确度

  - 支持命令行工具和Python API
  - 可实现从L2到L7的不同层次的网络系统

参考文献

- [Allman 2013] M. Allman, “Comments on Bufferbloat”, ACM SIGCOMM CCR 2013
- [Cardwell 2016] N. Cardwell et al., “BBR: Congestion-Based Congestion Control Measuring bottleneck bandwidth and round-trip propagation time”, ACM Queue 2016
- [Dobrian 2011] F. Dobrian et al., “Understanding the Impact of Video Quality on User Engagement”, ACM SIGCOMM 2011
- [Dukkipati 2006] N. Dukkipati et al., “Why Flow-Completion Time is the Right Metric for Congestion Control”, ACM SIGCOMM CCR, 2006
- [Gettys 2011] J. Gettys et al., “Bufferbloat: Dark Buffers in the Internet”, ACM Queue 2011
- [Guo 2016] Y. Guo et al., “Understanding On-device Bufferbloat For Cellular Upload”, ACM IMC 2016
- [Jiang 2012] H. Jiang et al., “Tackling Bufferbloat in 3G/4G Networks”, ACM IMC 2012
- [Koponen 2011] T. Koponen et al., “Architecting for Innovation”, ACM SIGCOMM CCR 2011
- [Leland 1993] W. Leland et al., “On the self-similar nature of Ethernet traffic”, ACM SIGCOMM 1993
- [Singla 2014] A. Singla et al., “The internet at the speed of light”, ACM HotNets 2014
- [van Jacobson 1988] van Jacobson, “Congestion Avoidance and Control”, ACM SIGCOMM 1988
- [Zhang 2010] L. Zhang et al., “Named Data Networking”, Tech Report, 2010

课程（微信）群

计算机网络（研讨课）讨论群

该二维码7天内(3月30日前)有效，重新进入将更新