

第二章 传输层

授课教师：李彤

中国大学



本章目标



- 掌握传输层服务原理
 - 传输层复用和分用
 - 可靠数据传输
 - 流量控制
 - 拥塞控制
- 掌握因特网传输层协议：
 - UDP协议
 - TCP协议



本章内容



中國人民大學
RENMIN UNIVERSITY OF CHINA

6.1 概述和传输层服务

6.2 无连接传输：UDP

6.3 面向连接的传输：TCP

6.4 理解网络拥塞

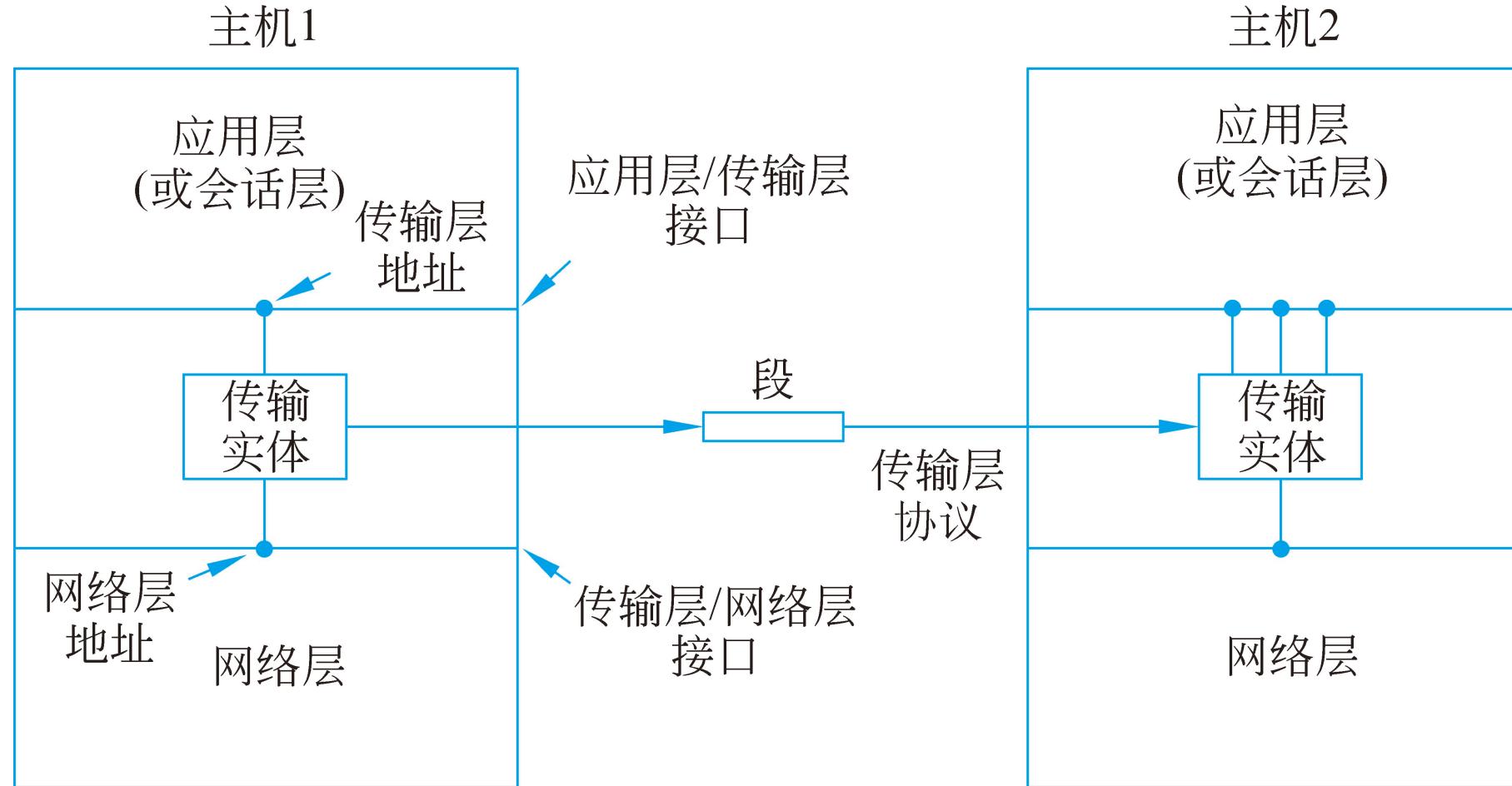
6.5 TCP拥塞控制

6.6 拥塞控制的发展

6.7 传输层协议的发展

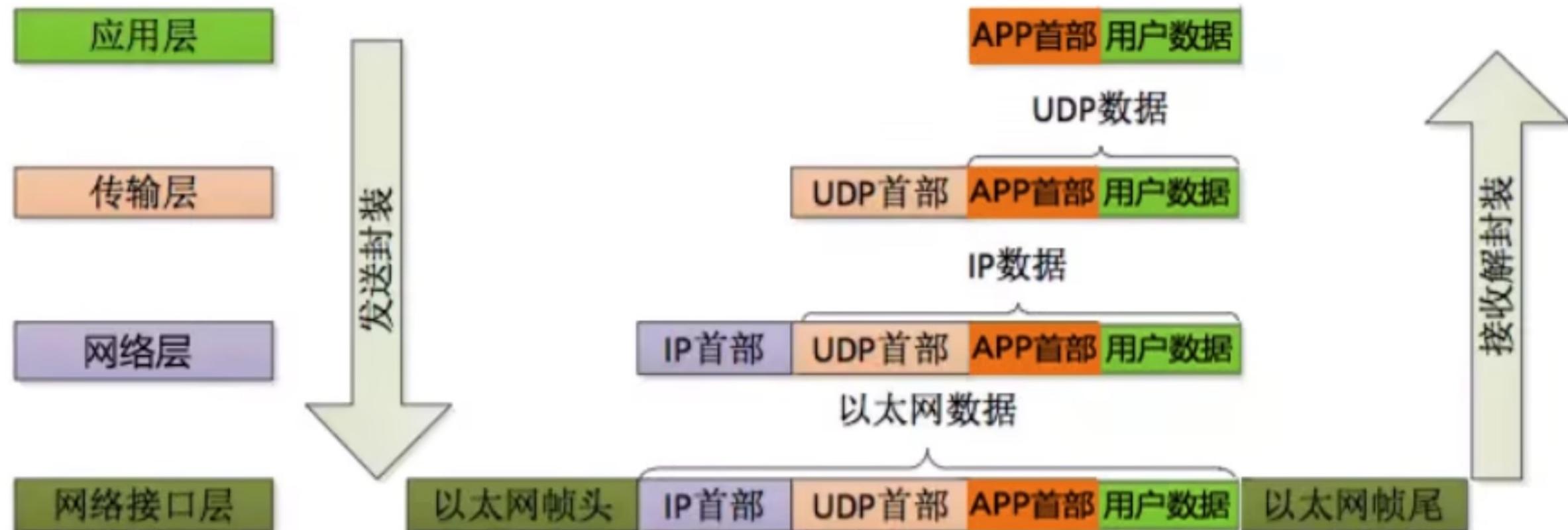


传输层的位置





传输层的位置

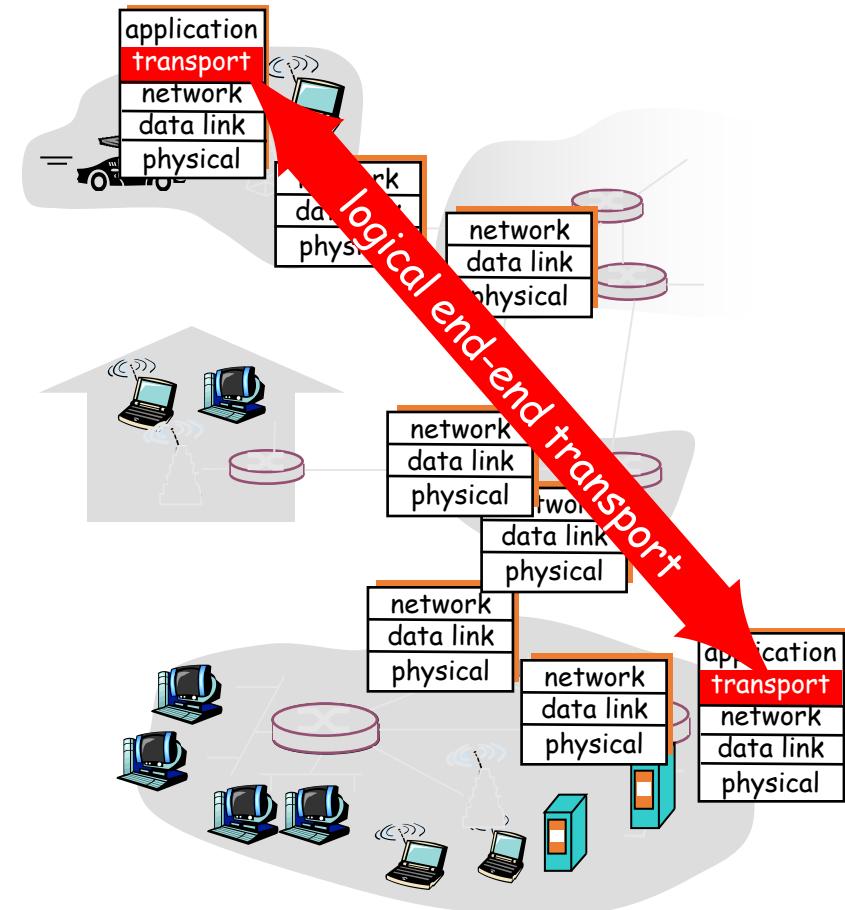




传输层提供什么服务？



- 因特网的网络层提供“**尽力而为**”的服务：
 - 网络层尽最大努力在终端间交付分组，但不提供任何承诺
 - 具体来说，不保证交付，不保证按序交付，不保证数据完整，不保证延迟，不保证带宽等
- 传输层的**有所为、有所不为**：
 - 传输层可以通过差错恢复、重排序等手段提供可靠、按序的交付服务
 - 但传输层无法提供延迟保证、带宽保证等服务





本章内容



中國人民大學
RENMIN UNIVERSITY OF CHINA

6.1 概述和传输层服务

6.2 无连接传输 : UDP

6.3 面向连接的传输 : TCP

6.4 理解网络拥塞

6.5 TCP拥塞控制

6.6 拥塞控制的发展

6.7 传输层协议的发展

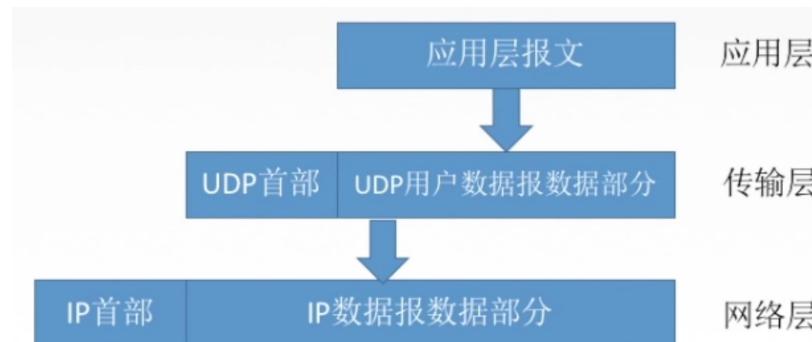


- 网络层提供的服务 (best-effort service)：
 - 尽最大努力将数据包交付到目的主机
 - 不保证投递的可靠性和顺序
 - 不保证带宽及延迟要求
- 用户数据报协议 UDP (User Datagram Protocol) 提供的服务：
 - 进程到进程之间的报文交付
- UDP只在IP数据报服务之上增加了很少的功能：
 - 复用和分用
 - 报文差错检测



➤ UDP的主要特点

- UDP是无连接的，减少开销和发送数据之前的时延
- UDP使用最大努力交付，不保证可靠交付
- UDP是面向报文的，适合一次性传输少量数据的网络应用
- UDP无拥塞控制，适合实时应用
- UDP首部开销小，8字节（TCP占用20字节）



SOCK_DGRAM :
应用层给UDP多长的报文，UDP都照样发送，即一次发一个完整报文

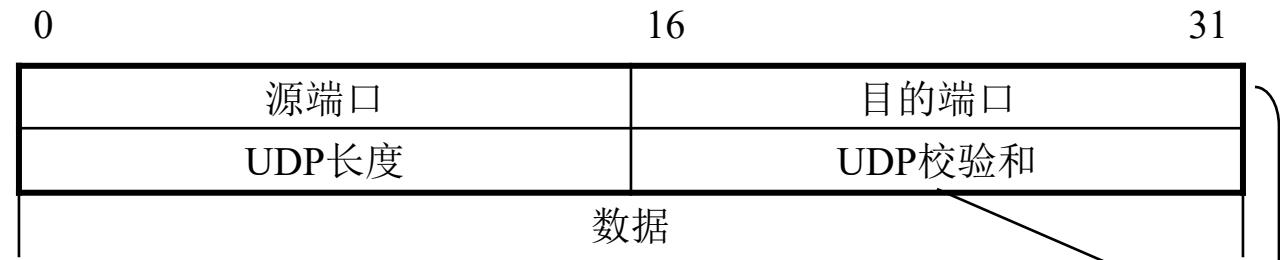


UDP报文段结构

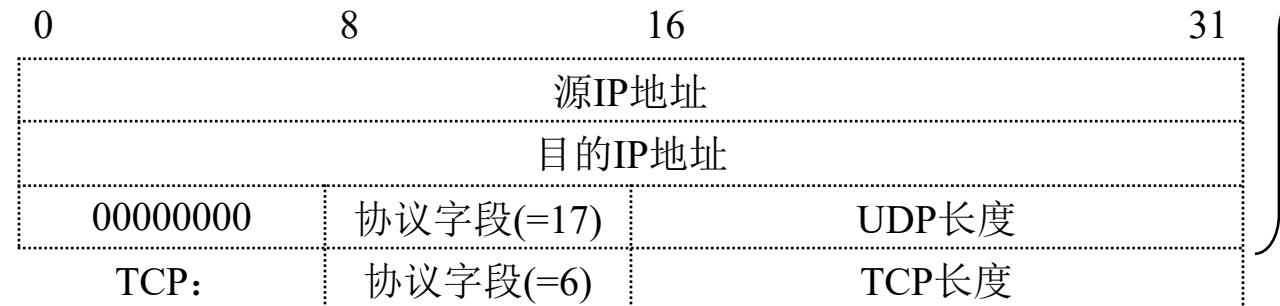


中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ UDP首部



➤ IPv4伪首部



- 1、伪首部只在计算校验和时才出现，不向下传送也不向上递交
- 2、UDP长度：UDP首部8B+数据部分长度（不包括伪首部）



为什么需要UDP？

为什么需要UDP？

- 应用可以尽可能快地发送报文：
 - 无建立连接的延迟
 - 不限制发送速率（不进行拥塞控制和流量控制）
- 报头开销小
- 协议处理简单

UDP适合哪些应用？

- 容忍丢包但对延迟敏感的应用：
 - 如流媒体
- 以单次请求/响应为主的应用：
 - 如DNS
- 若应用要求基于UDP进行可靠传输：
 - 由应用层实现可靠性



中國人民大學
RENMIN UNIVERSITY OF CHINA

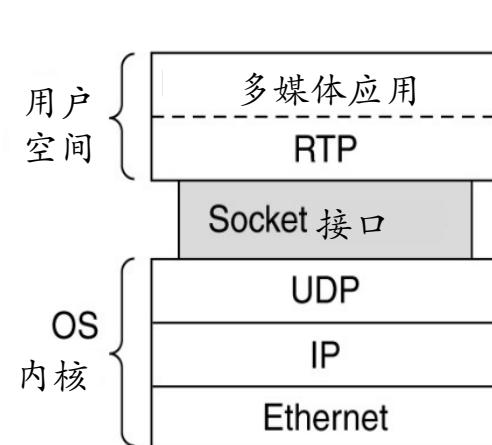


基于UDP的传输协议举例: RTP

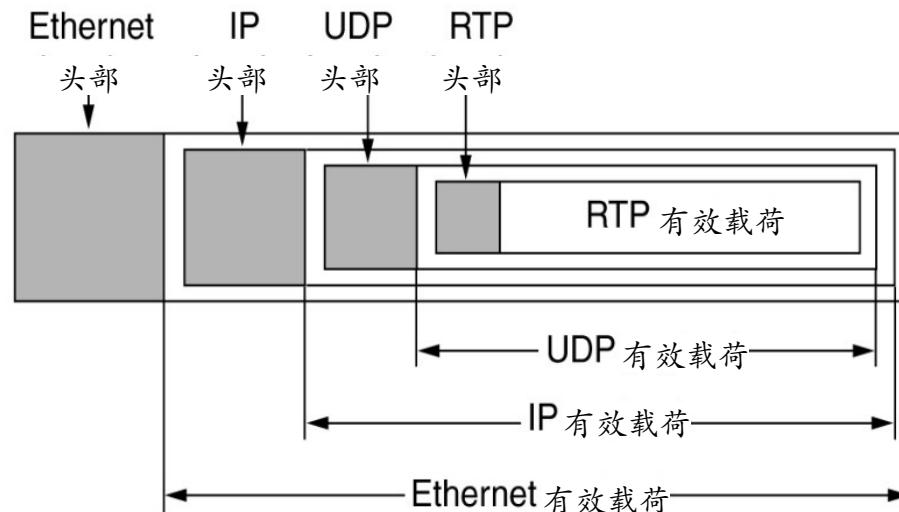


中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ 实时传输协议RTP (RFC3550)



(a)



(b)



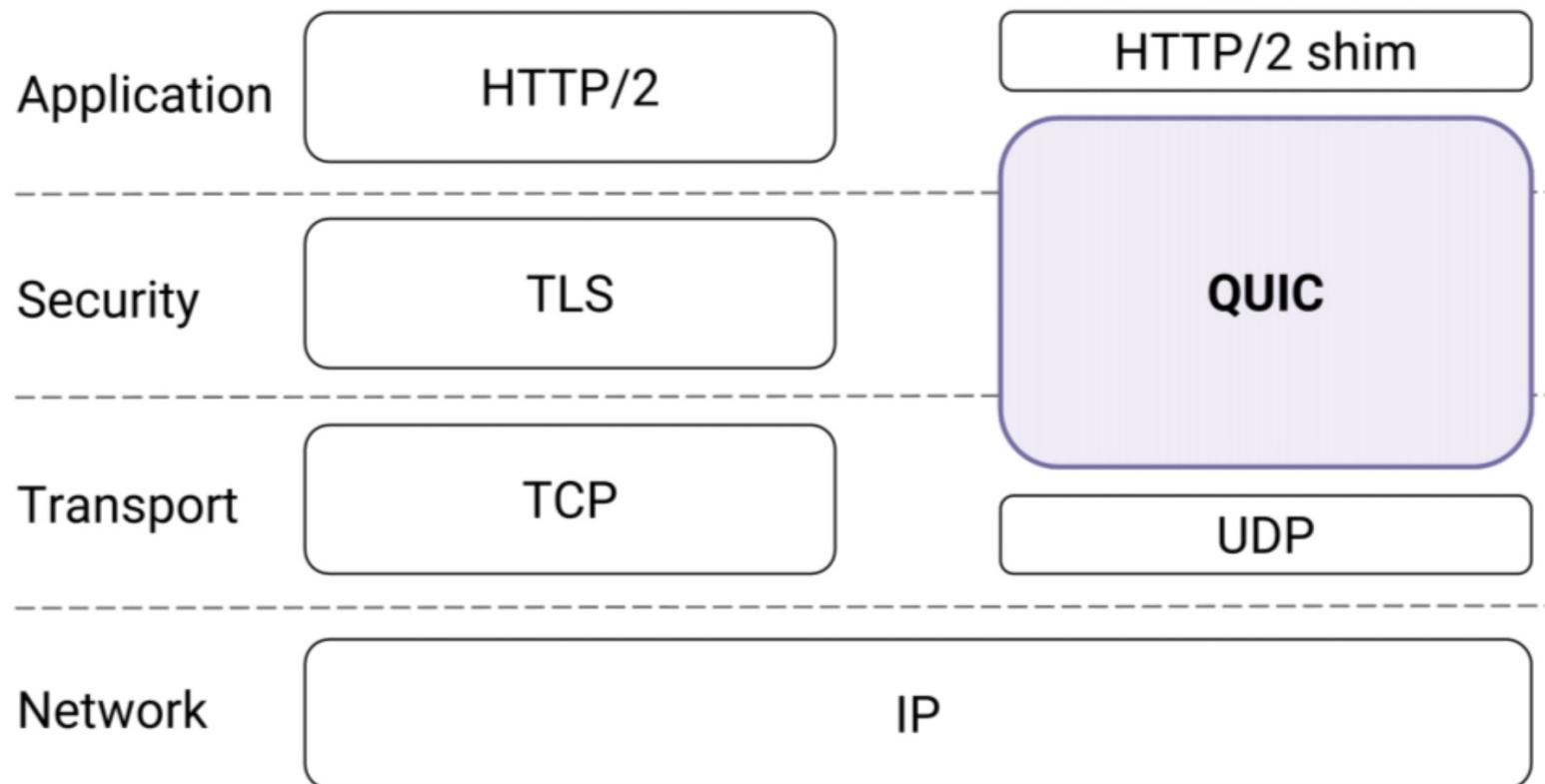
基于UDP的传输协议举例: QUIC



中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ 传输协议QUIC (RFC9000)

QUIC : Quick UDP Internet Connection

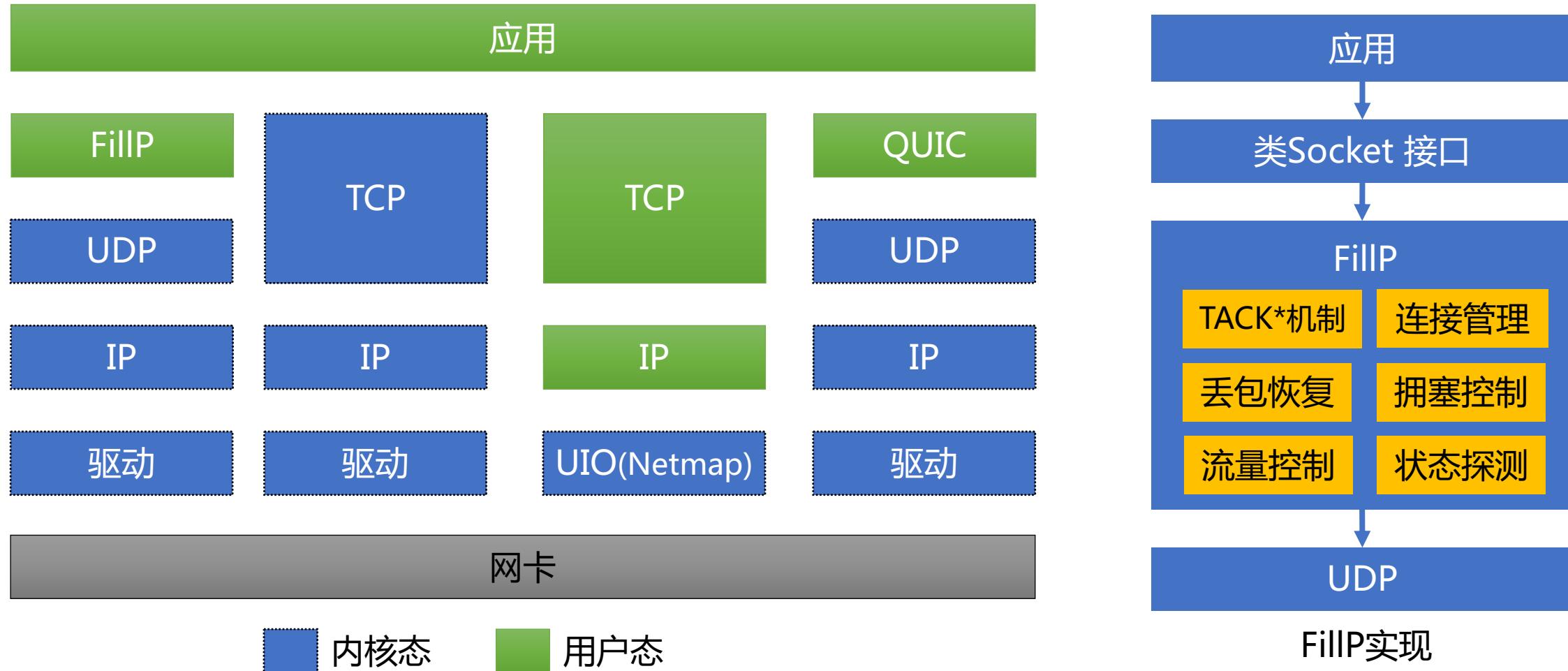




基于UDP的传输协议举例: FillP



中國人民大學
RENMIN UNIVERSITY OF CHINA



*Tong Li et al., TACK: Improving Wireless Transport Performance by Taming Acknowledgments. ACM SIGCOMM, 2020

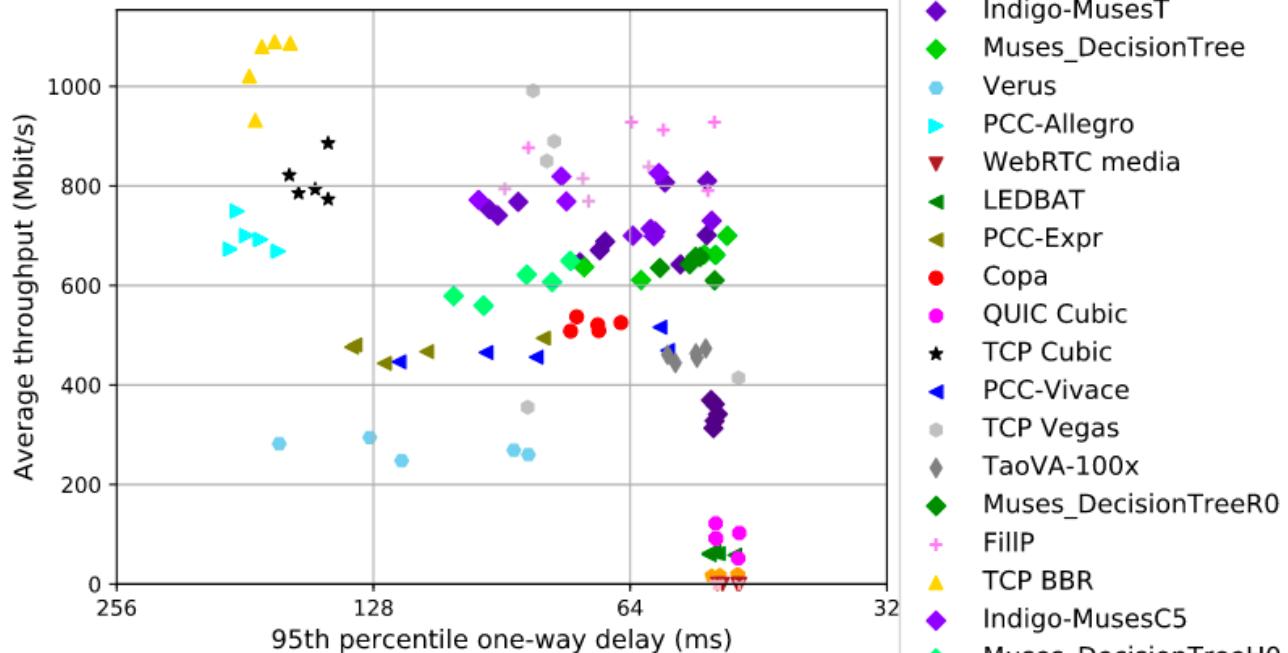


基于UDP的传输协议举例: FillP



中國人民大學
RENMIN UNIVERSITY OF CHINA

test from GCE London to GCE Iowa, 5 runs of 30s each per scheme
3 flows with 10s interval between flows



数据来源：<https://pantheon.stanford.edu/>

Best **Worst**
□ represents no data available, either if the scheme did not run, or failed during tests. This summary includes only single-flow experiments; for individual results (including multi-flow experiments) please see the [Find Results](#) tab.

Date (UTC)	Description	TCP BBR	Copa	TCP Cubic	FillP	FillP-Sheep	Indigo	LEDBAT	PCC-Allegro	PCC-Expr	QUIC Cubic	SCReAM	Sprout	TaoVA-100x	TCP Vegas	Verus	PCC-Vivace	WebRTC media
04/17/2020	GCE London to GCE Iowa, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/17/2020	GCE Sydney to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/17/2020	GCE London to GCE Iowa, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/17/2020	GCE Sydney to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Sydney to GCE London, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Iowa to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Iowa to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Sydney to GCE London, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Tokyo to GCE London, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Sydney to GCE Iowa, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Tokyo to GCE London, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
04/16/2020	GCE Sydney to GCE Iowa, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/19/2020	GCE London to GCE Iowa, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/19/2020	GCE Sydney to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/18/2020	GCE London to GCE Iowa, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/18/2020	GCE Sydney to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/18/2020	GCE Iowa to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/18/2020	GCE Sydney to GCE London, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/18/2020	GCE Tokyo to GCE London, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/18/2020	GCE Sydney to GCE Iowa, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
02/18/2020	GCE Iowa to GCE Tokyo, Ethernet	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■



本章内容

6.1 概述和传输层服务

6.2 无连接传输：UDP

6.3 面向连接的传输：TCP

6.4 理解网络拥塞

6.5 TCP拥塞控制

6.6 拥塞控制的发展

6.7 传输层协议的发展

1. TCP概述
2. TCP报文段结构
3. TCP可靠数据传输
4. TCP流量控制
5. TCP连接管理





传输控制协议TCP (Transmission Control Protocol) 概述

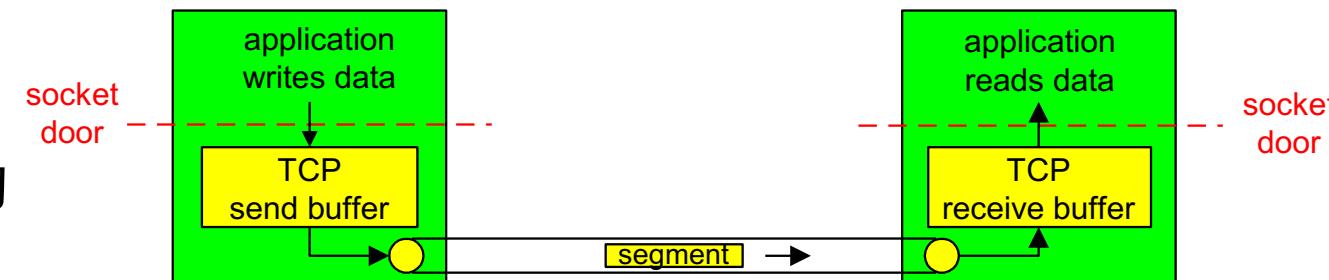


中國人民大學
RENMIN UNIVERSITY OF CHINA

- TCP是面向连接的传输层协议：
 - 为进程之间提供一条字节流管道
- 点到点通信：
 - 每条TCP连接只能有两个端点
- **保流、保序、可靠传输：**
 - 将应用数据看作一连串无结构的字节流
 - 无差错、不丢失、不重复、有序提交
- 全双工通信：
 - 发送缓存：准备发送的数据&已发送但未收到确认的数据
 - 接收缓存：按序到达但未被应用读取的数据&乱序到达的数据

TCP的核心功能模块：

- 建立连接：
 - 通信双方为本次通信建立数据传输所需的状态（套接字、缓存、变量等）
- 可靠数据传输：
 - 流水线式发送，报文段检错，丢失重传
- 流量控制：
 - 发送方不会令接收方缓存溢出





TCP字节流的含义（补充说明）



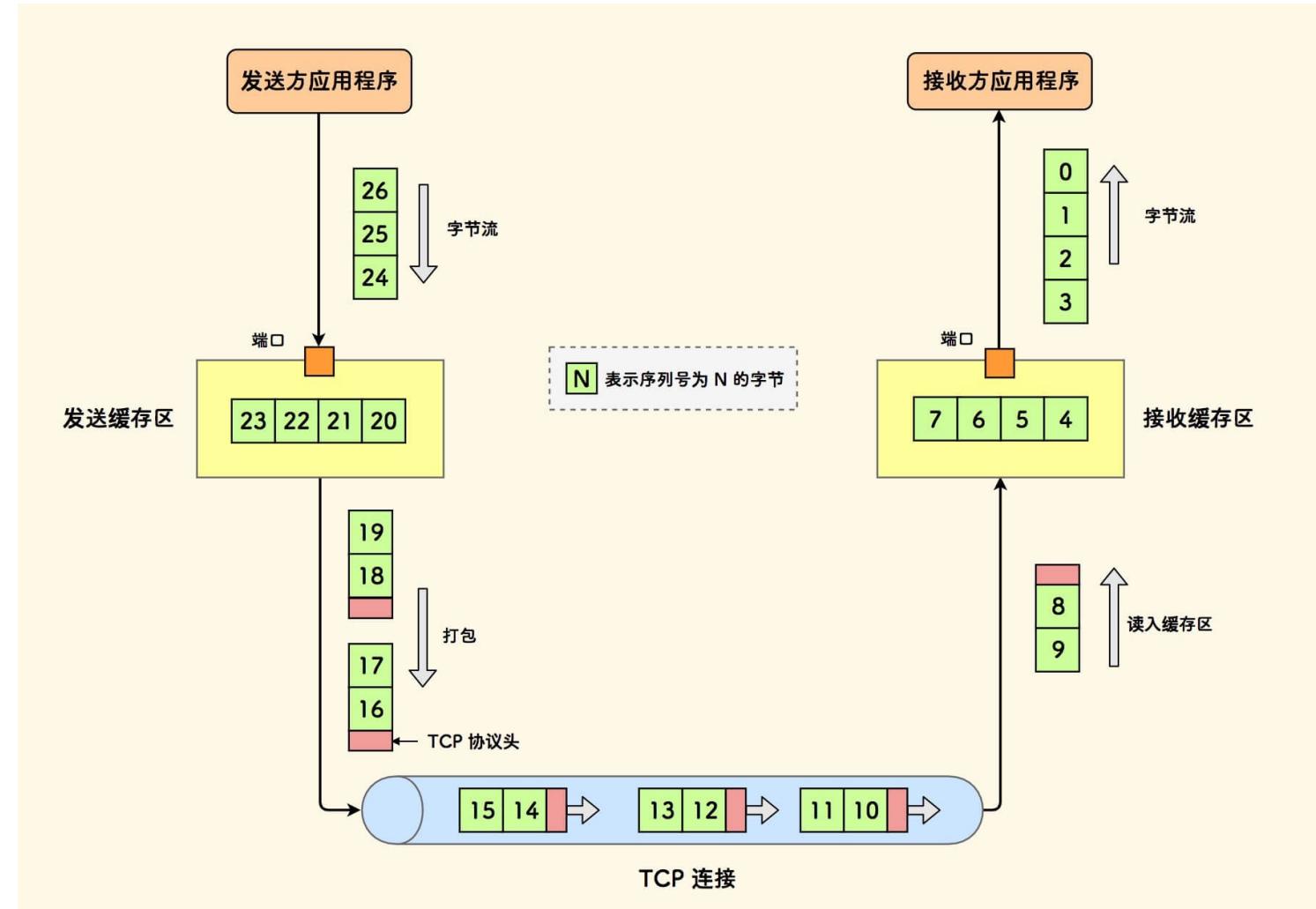
中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ UDP保留报文边界：

- 应用层给UDP多长的报文，UDP都照样发送，即一次发一个完整报文。所以，UDP是保留报文（**应用程序的输出**）边界的

➤ TCP不保留报文边界：

- 发送方TCP通常会缓存应用程序交给它的数据，应用程序的多次输入被不加区分地放在一起。因此，应用程序不同时候的输出可能会被封装在同一个报文段中传输，在TCP看来，它交付的内容是没有结构的字节流，这些字节流的语义由应用程序解释

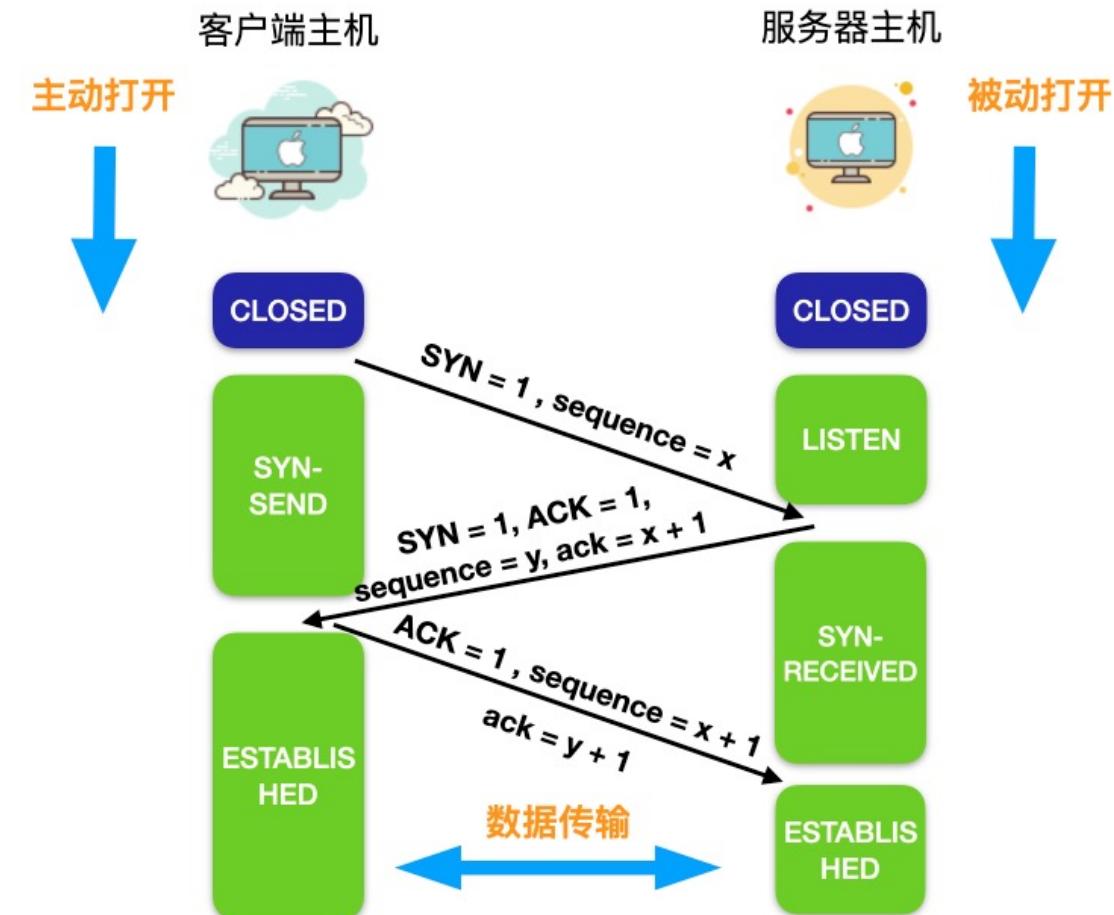




建立连接的本质（补充说明）



- 建立连接：通信的双方为本次通信建立好传输数据所需要的状态，包括套接字、缓冲区和相关变量
- 关闭连接：释放相关的资源，包括关闭套接字、释放缓冲区和变量所占用的内存





本章内容

- 6.1 概述和传输层服务
- 6.2 套接字编程
- 6.3 传输层复用和分用
- 6.4 无连接传输：UDP
- 6.5 面向连接的传输：TCP
- 6.6 理解网络拥塞
- 6.7 TCP拥塞控制
- 6.8 拥塞控制的发展
- 6.9 传输层协议的发展

- 1. TCP概述
- 2. TCP报文段结构
- 3. TCP可靠数据传输
- 4. TCP流量控制
- 5. TCP连接管理



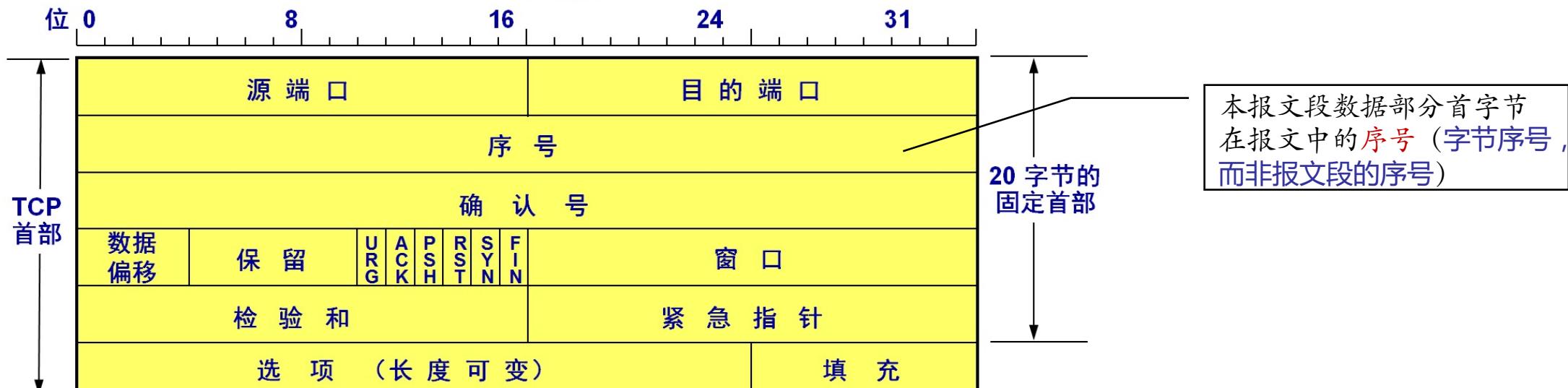


TCP报文段结构



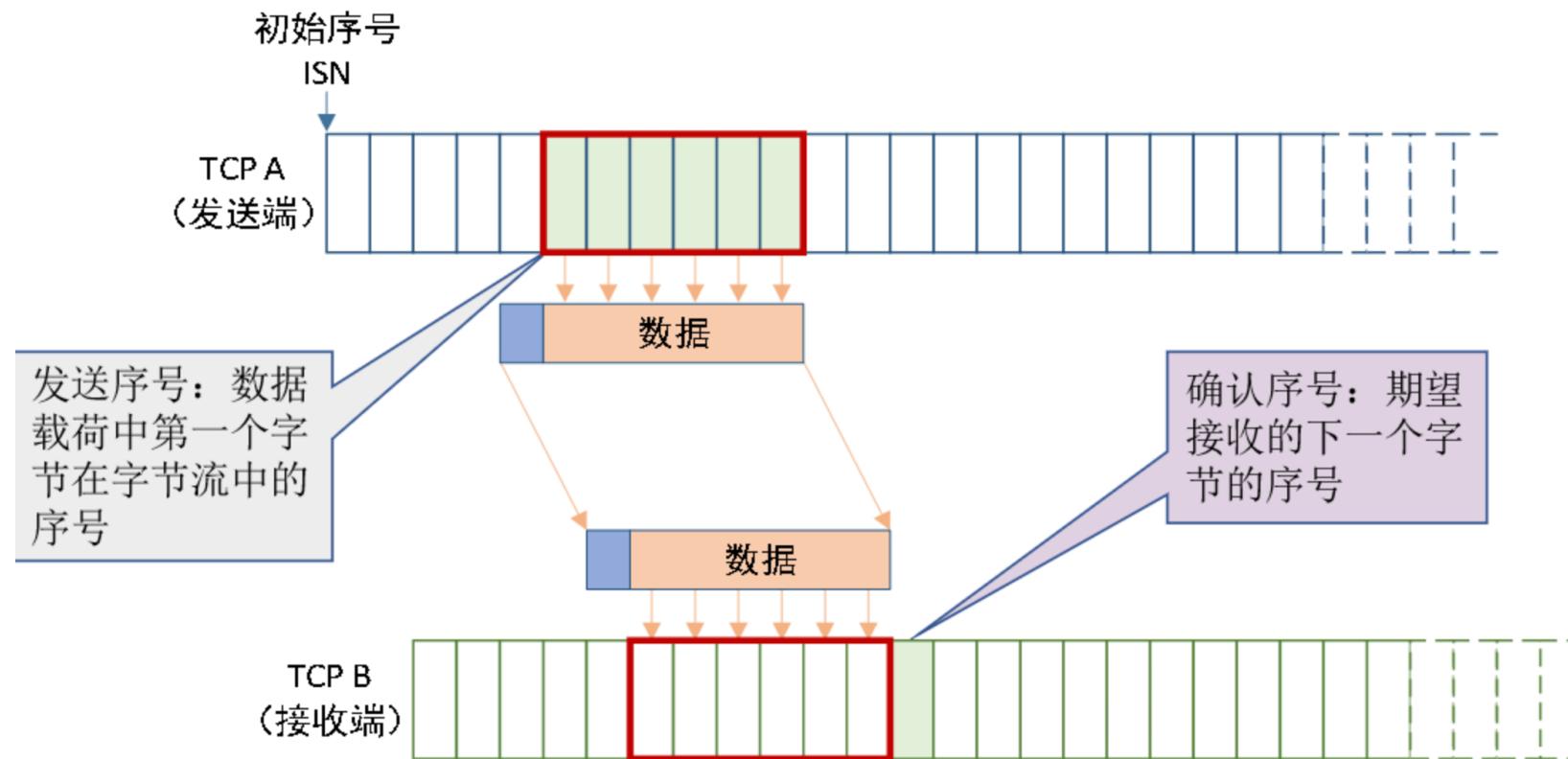
➤ TCP段的结构

- 数据偏移占4位，则头部最大长度：60字节，固定部分：20字节，选项字段不超过40字节





发送序号和确认序号的含义



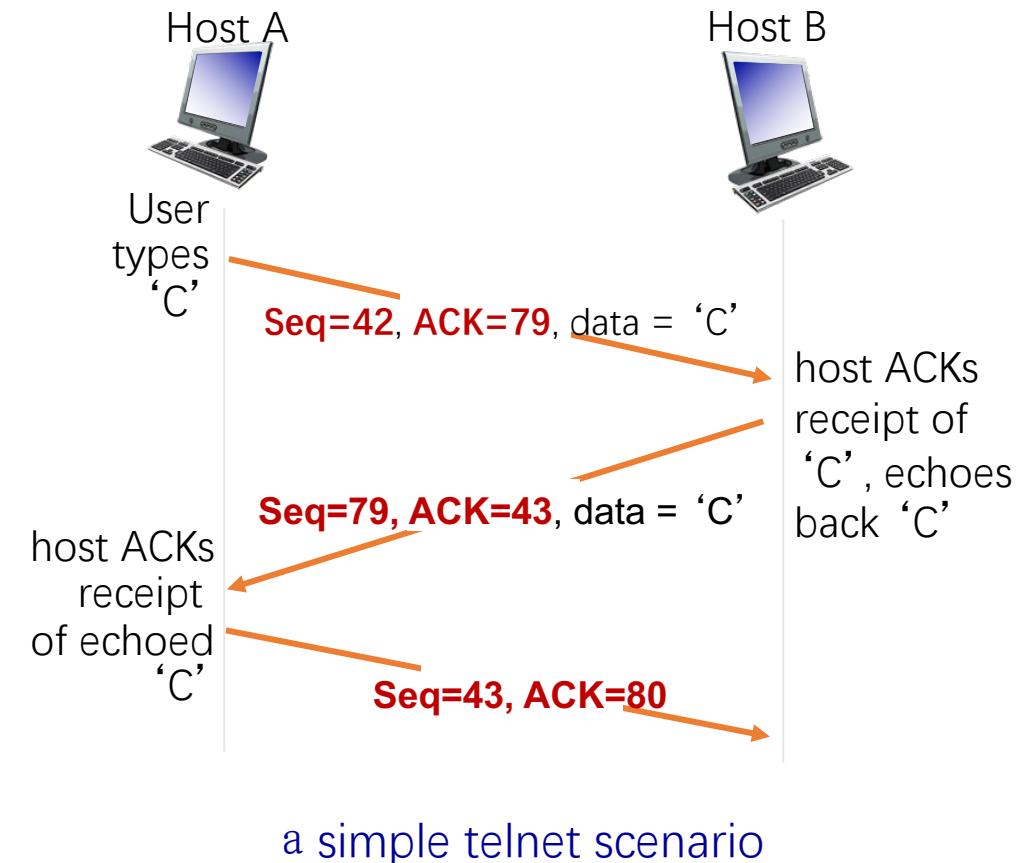
初始序号的选取: 每个TCP实体维护一个32位计数器，该计数器每4微秒增1，建立连接时从中读取计数器当前值



发送序号和确认序号：使用举例



- 主机A向主机B发送仅包含一个字符‘C’的报文段：
 - 发送序号为42
 - 确认序号为79（对前一次数据的确认）
- 主机B将字符‘C’回送给主机A：
 - 发送序号为79
 - 确认序号为43（对收到‘C’的确认）
- 主机A向主机B发送确认报文段（不包含数据）：
 - 确认序号为80（对收到‘C’的确认）



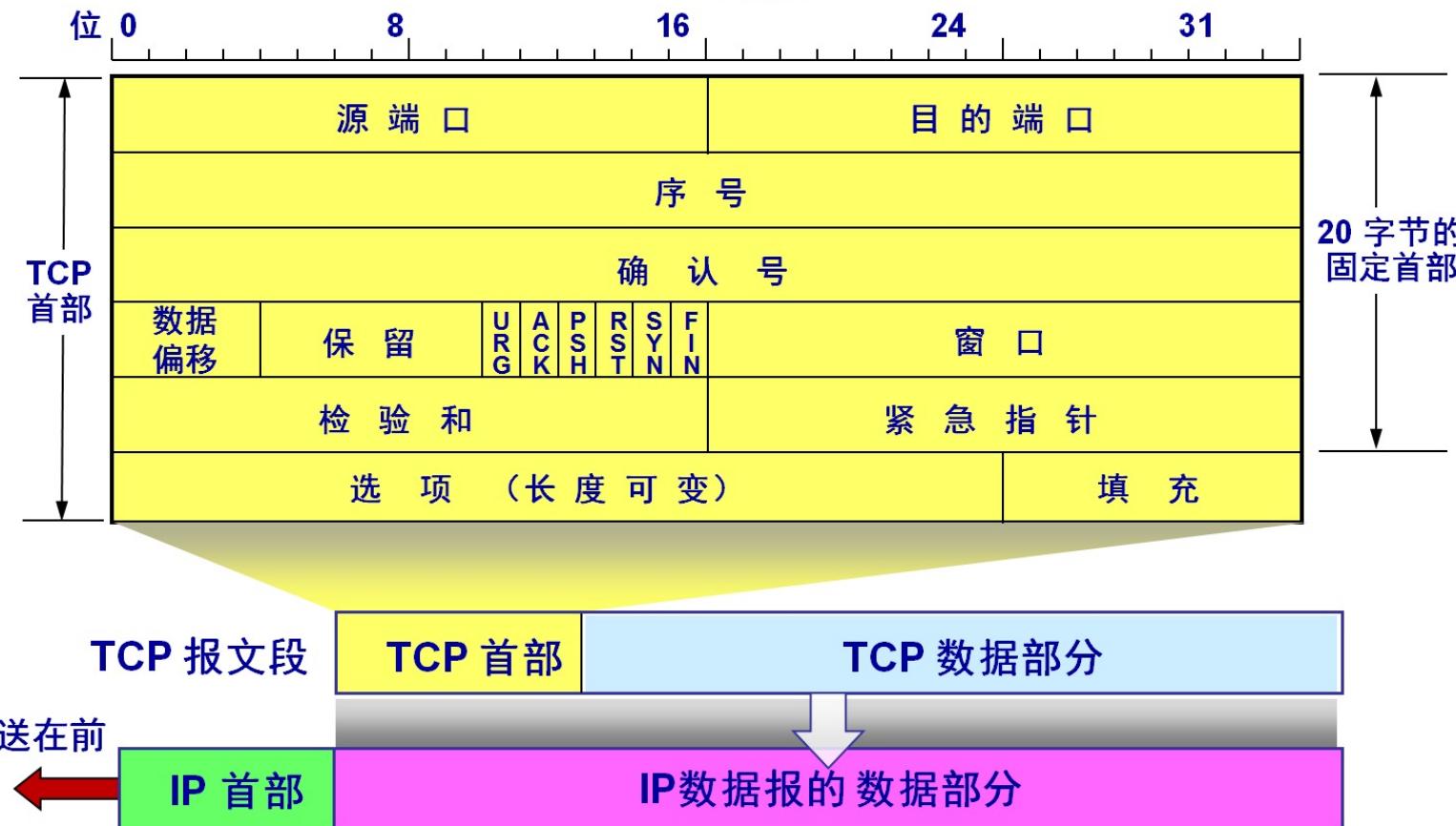


TCP报文段结构



➤ TCP首部中的控制位ACK

- 确认 ACK (ACKnowledgment) 仅当 $ACK = 1$ 时确认号字段才有效。当 $ACK = 0$ 时，确认号无效。TCP 规定，在连接建立后所有传送的报文段都必须把 ACK 置 1



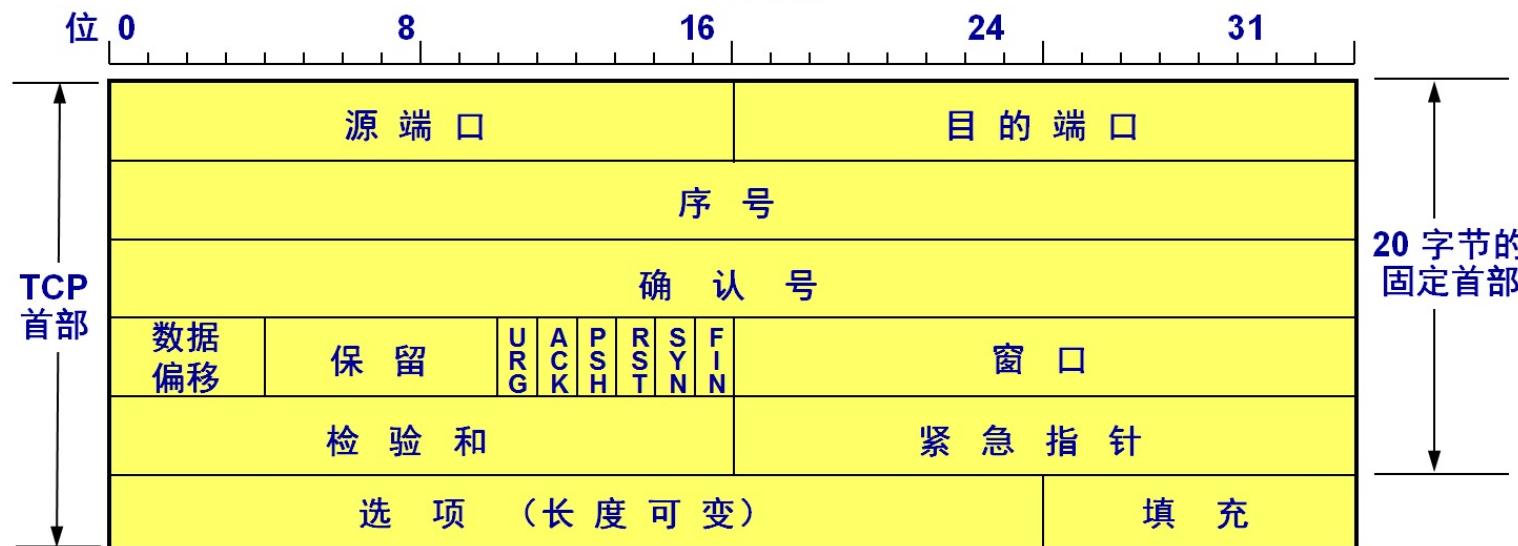


TCP报文段结构



➤ TCP首部中的控制位SYN

- 同步 SYN (SYNchronization) 在连接建立时用来同步序号。 SYN 置为 1 表示这是一个连接请求或连接接受报文





TCP报文段结构



➤ TCP首部中的控制位FIN

- 终止 FIN (FINish) 用来释放一个连接。当 $\text{FIN} = 1$ 时，表明此报文段的发送方的数据已发送完毕，并要求释放运输连接



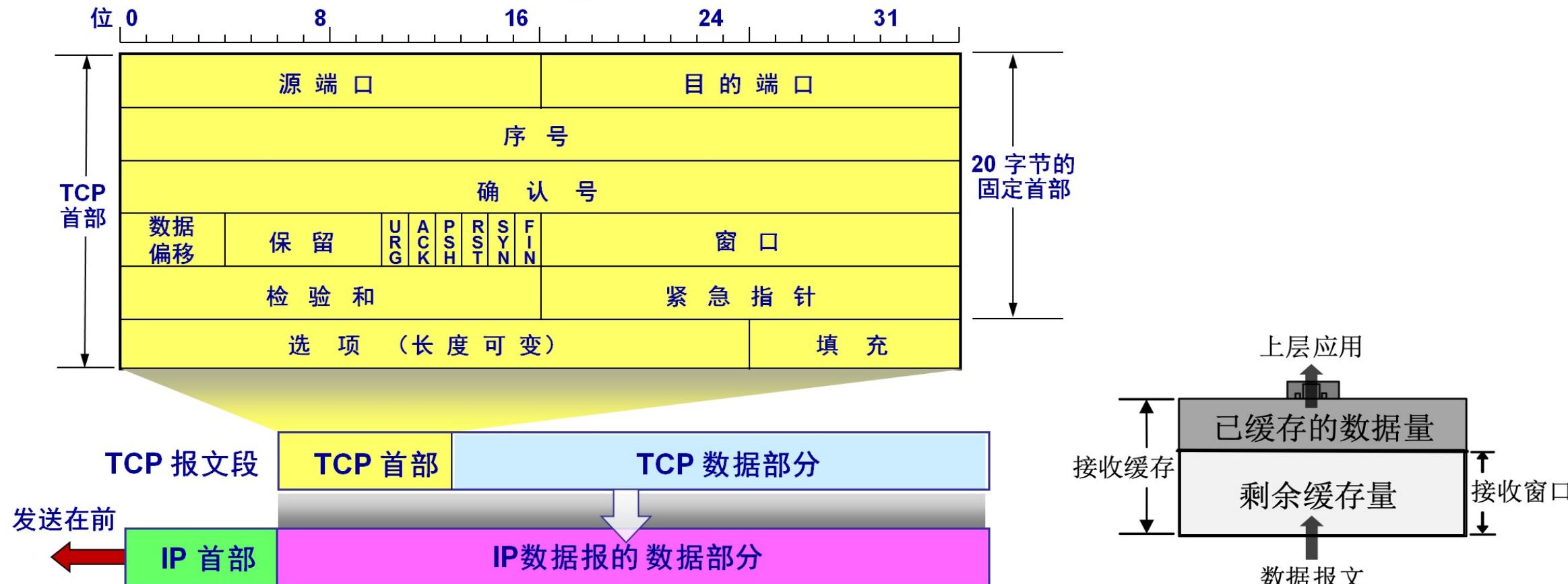


TCP报文段结构



➤ TCP首部中的窗口

- 窗口指的是发送本报文段的一方的接收窗口（而不是自己的发送窗口）。窗口值告诉对方：从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量（以字节为单位）





TCP选项 (Option) 字段



中國人民大學
RENMIN UNIVERSITY OF CHINA

- TCP选项区域可包含多个选项 (Option) 字段
- 采用经典的TLV (Type-Length-Value) 结构来表示

Kind / Type (1 Byte)	Length (1 Byte)	Value
------------------------	-------------------	-------

- TCP定义了丰富的选项字段
 - Window Scale (WSCALE或WSopt)选项
 - Maximum Segment Size(MSS)选项
 - Timestamp选项
 - SACK选项 (Selective ACK)

Kind (Type)	Length	Name	Reference	描述 & 用途
0	1	EOL	RFC 793	选项列表结束
1	1	NOP	RFC 793	无操作 (用于补位填充)
2	4	MSS	RFC 793	最大Segment长度
3	3	WSOPT	RFC 1323	窗口扩大系数 (Window Scaling Factor)
4	2	SACK-Premitted	RFC 2018	表明支持SACK
5	可变	SACK	RFC 2018	SACK Block (收到乱序数据)
8	10	TSPOT	RFC 1323	Timestamps
19	18	TCP-MD5	RFC 2385	MD5认证
28	4	UTO	RFC 5482	User Timeout (超过一定闲置时间后拆除连接)
29	可变	TCP-AO	RFC 5925	认证 (可选用各种算法)
253/254	可变	Experimental	RFC 4727	保留, 用于科研实验



重要的TCP选项（1）



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 最大段长度（MSS）：
 - TCP段中可以携带的最大数据字节数
 - 建立连接时，每个主机可声明自己能够接受的MSS，缺省为536字节
- 窗口比例因子（Window Scale）：
 - 建立连接时，双方可以协商一个窗口比例因子
 - 实际接收窗口大小 = window size * $2^{\text{window scale}}$



➤ 时间戳选项结构

- 10个字节 = type(1字节) + length(1字节) + value (8字节)

Kind / Type (1 Byte)	Length (1 Byte)	Value
------------------------	-------------------	-------

- 其中，type=8，length=10，value由发送方时间戳 (Timestamp Value (TSval))、回显时间戳 (Timestamp Echo Reply (TSecr)) 两个值组成，各4个字节的长度

➤ 时间戳选项的功能

- 两端往返时延测量 (RTT)
- 序列号回绕 (PAWS)

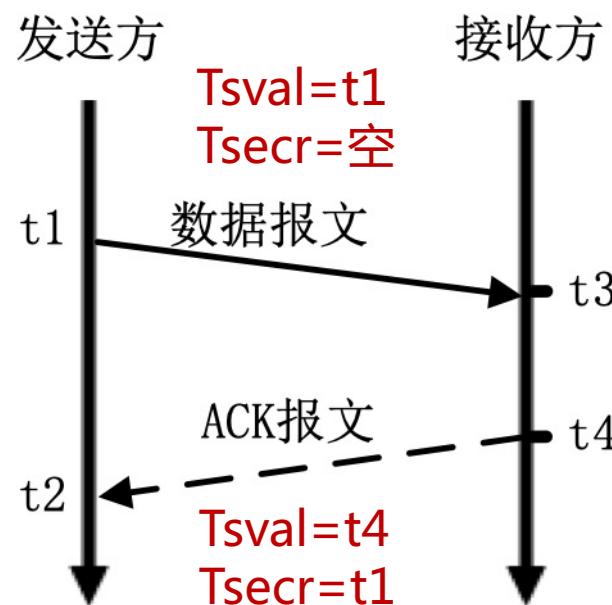


Timestamp选项



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 时间戳选项中的发送方时间戳 (TSval) 和回显时间戳 (TSecr)



$$RTT = t_1 - t_2 - (t_4 - t_3)$$

TCP认为 : t_4 等于 t_3

```
-----  
▲ TCP Option - Timestamps: TSval 149055455, TSecr 5207020  
  Kind: Time Stamp Option (8)  
  Length: 10  
  Timestamp value: 149055455  
  Timestamp echo reply: 5207020  
▲ TCP Option - No-Operation (NOP)  
  Kind: No-Operation (1)  
▲ TCP Option - Window scale: 7 (multiply by 128)  
  Kind: Window Scale (3)  
  Length: 3  
  Shift count: 7  
  [Multiplier: 128]  
▷ [SEQ/ACK analysis]
```

0000	45 00 00 3c ba 5c 00 00	39 06 0c 3d d8 3a d0 2e	E..<.\... 9..=.:...
0010	0a 08 08 b2 01 bb 95 67	46 b9 2a 5f e2 ca 9f a3g F.*_....
0020	a0 12 a6 2c 77 86 00 00	02 04 05 28 04 02 08 0a	...w....(....
0030	08 e2 67 df 00 4f 73 ec	01 03 03 07	..g..0s.

不需要时钟同步 , 不需要保存状态



➤ SACK选项结构

- 10个字节 = type(1字节) + length(1字节) + range (8字节)

Kind / Type (1 Byte)	Length (1 Byte)	Block
------------------------	-------------------	-------

- 其中，type=5，length=10，range由**左边界** (left edge of block) 、**右边界** (right edge of block) 两个值组成，各4个字节的长度
- 每新增加一个Block，将占用8字节.

➤ SACK选项的功能

- 用于接收方告诉发送方，本端已经**收到并缓存**的**不连续**的数据块
- 避免不必要的重传



SACK选项

Kind=5	Length
Left Edge of 1st Block	
Right Edge of 1st Block	
...	
Left Edge of nth Block	
Right Edge of nth Block	



SACK选项最多
携带4组Block



本章内容

6.1 概述和传输层服务

6.2 套接字编程

6.3 传输层复用和分用

6.4 无连接传输：UDP

6.5 面向连接的传输：TCP

6.6 理解网络拥塞

6.7 TCP拥塞控制

6.8 拥塞控制的发展

6.9 传输层协议的发展

1. TCP概述
2. TCP报文段结构
3. TCP可靠数据传输
4. TCP流量控制
5. TCP连接管理





- TCP 在不可靠的IP服务上建立可靠的数据传输
- 基本机制
 - 发送端：流水线式发送数据、等待确认、超时重传
 - 接收端：进行差错检测，采用累积确认机制
- 乱序段处理：
 - 接收端缓存失序的报文：效率高，但处理复杂



高度简化的TCP协议：仅考虑可靠传输机制，且数据仅在一个方向上传输

- 接收方：
 - 确认方式：采用累积确认，仅在正确、按序收到报文段后，更新确认序号；其余情况，重复前一次的确认序号（与GBN类似）
 - 失序报文段处理：缓存失序的报文段（与SR类似）
- 发送方：
 - 发送策略：流水线式发送报文段
 - 定时器的使用：仅对最早未确认的报文段使用一个重传定时器（与GBN类似）
 - 重发策略：仅在超时后重发最早未确认的报文段（与SR类似，因为接收端缓存了失序的报文段）



TCP发送方要处理的事件

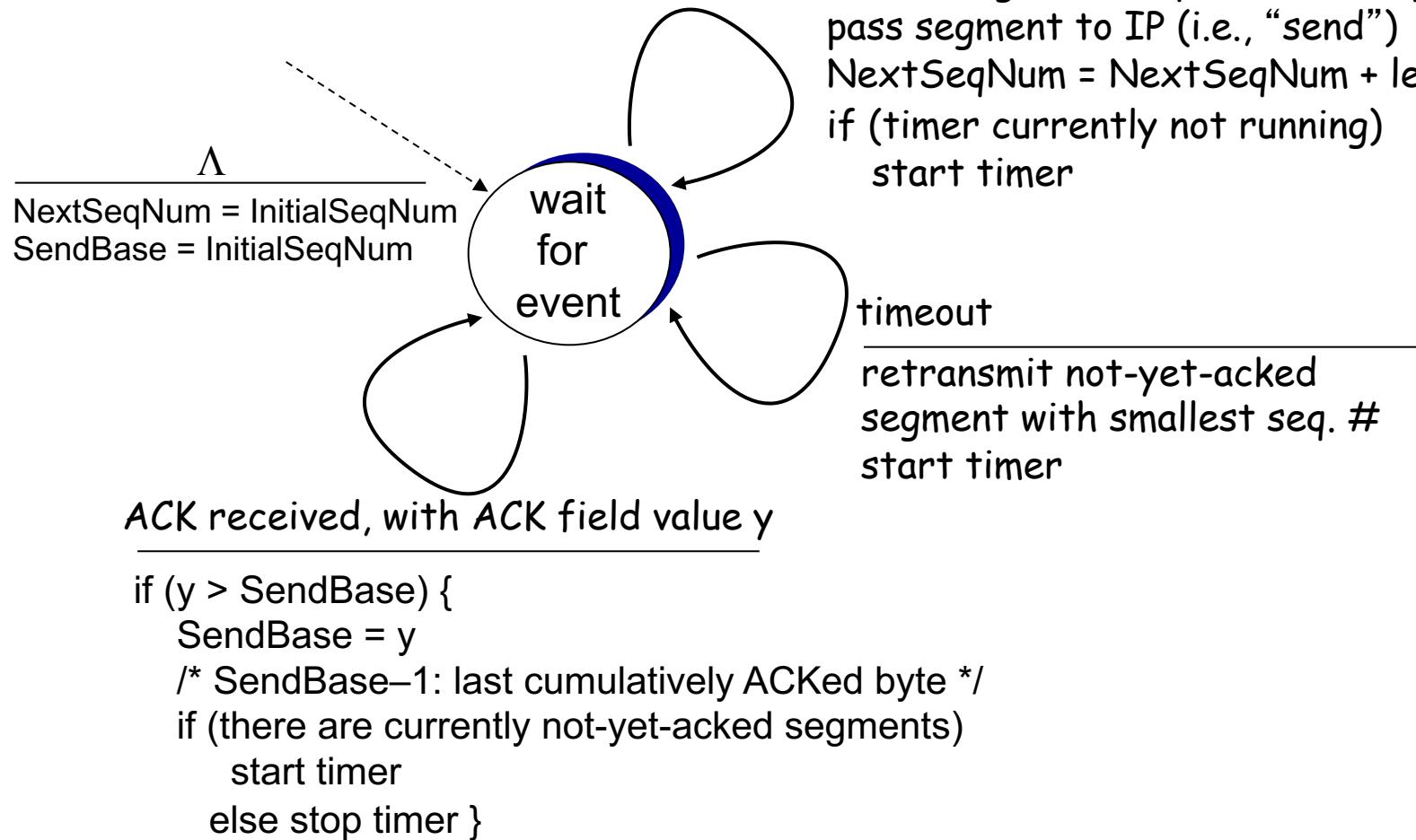


中國人民大學
RENMIN UNIVERSITY OF CHINA

- 收到应用数据：
 - 创建并发送TCP报文段
 - 若当前没有定时器在运行（没有已发送、未确认的报文段），启动定时器
- 超时：
 - 重传包含最小序号的、未确认的报文段
 - 重启定时器
- 收到ACK：
 - 如果确认序号大于基序号（已发送未确认的最小序号）：
 - 推进发送窗口（更新基序号）
 - 如果发送窗口中还有未确认的报文段，启动定时器，否则终止定时器



TCP发送端状态机（简化版）



data received from application above

create segment, seq. #: NextSeqNum

pass segment to IP (i.e., “send”)

NextSeqNum = NextSeqNum + length(data)

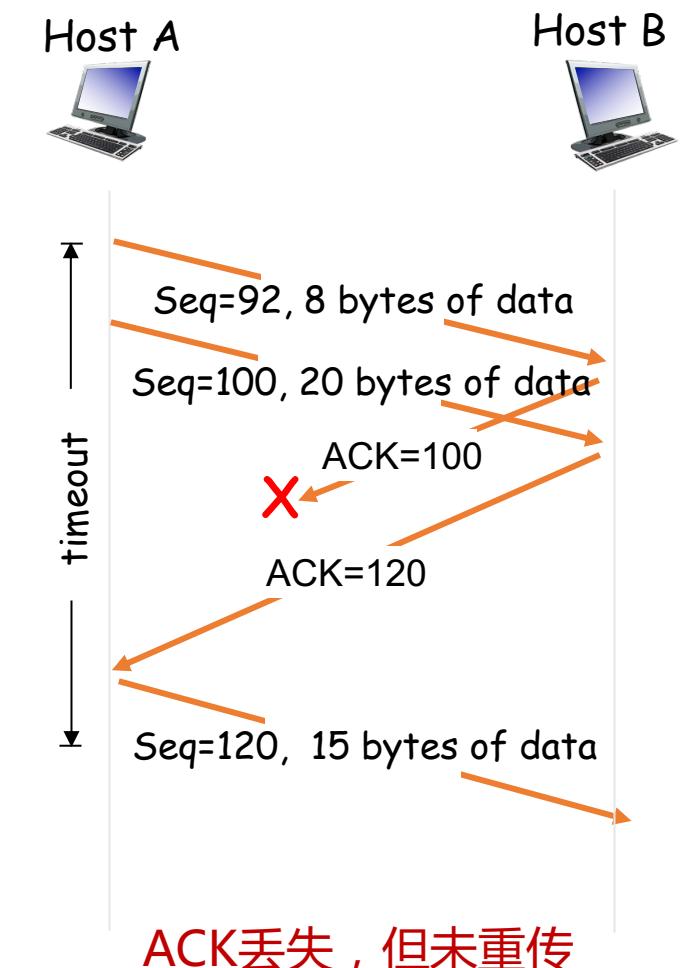
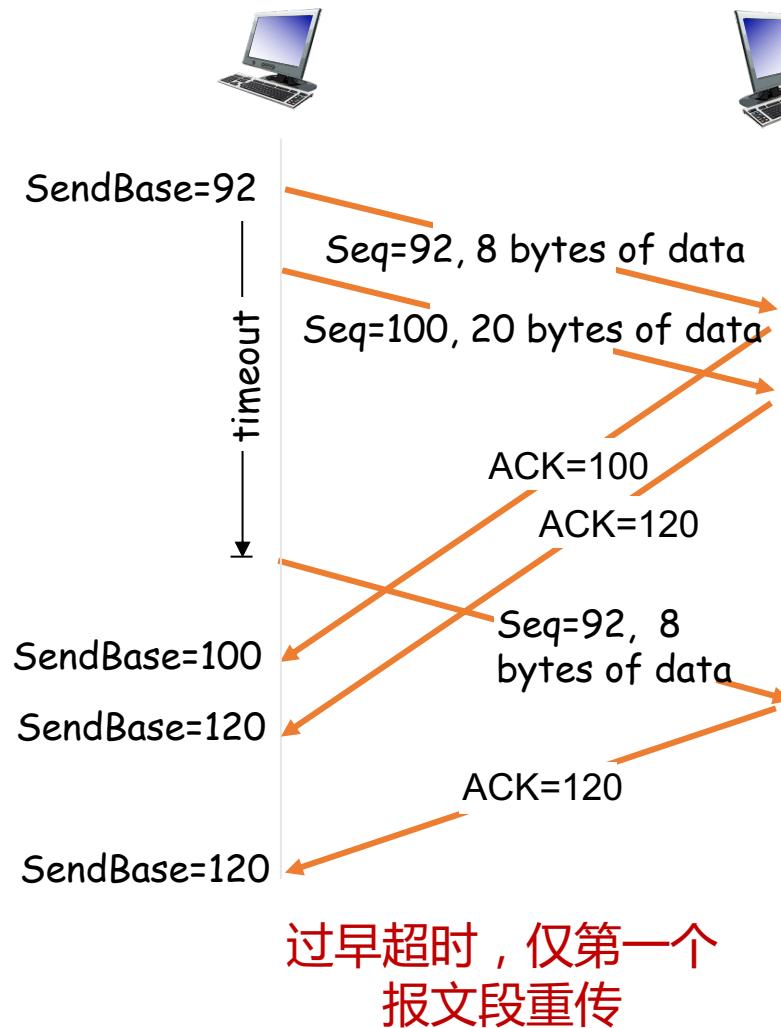
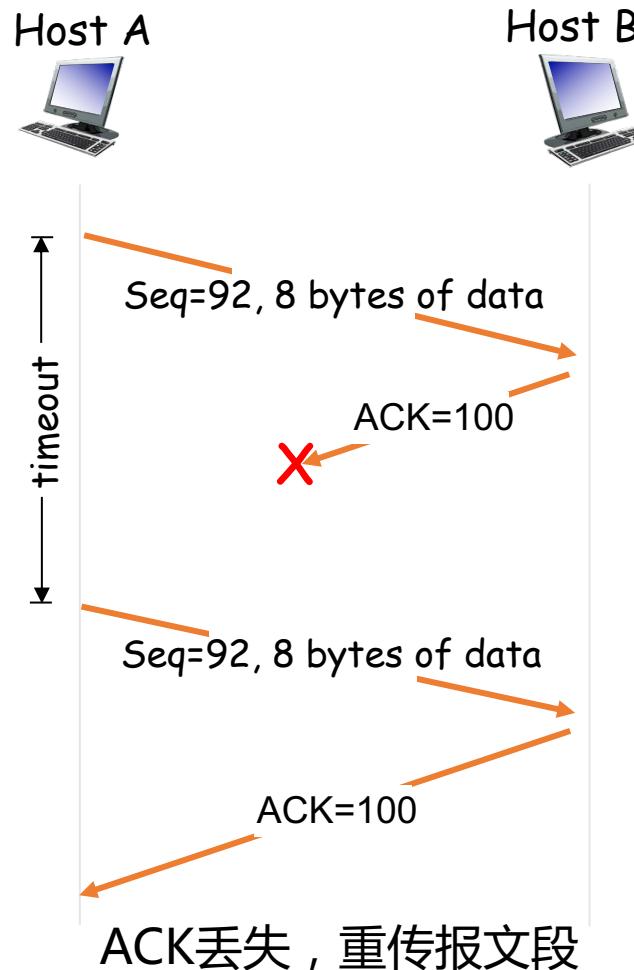
if (timer currently not running)
 start timer

timeout

retransmit not-yet-acked
segment with smallest seq. #
start timer



TCP可能的重传场景





➤ 思考以下问题：

- 第二种情形，如果TCP，像SR一样每个报文段使用一个定时器，会怎么样？
- 第三种情形，采用流水式发送和累积确认，可以避免重发哪些报文段？

➤ 可见，TCP通过采用以下机制减少了不必要的重传：

- 只使用一个定时器，避免了超时设置过小时重发大量报文段
- 利用流水式发送和累积确认，可以避免重发某些丢失了ACK的报文段

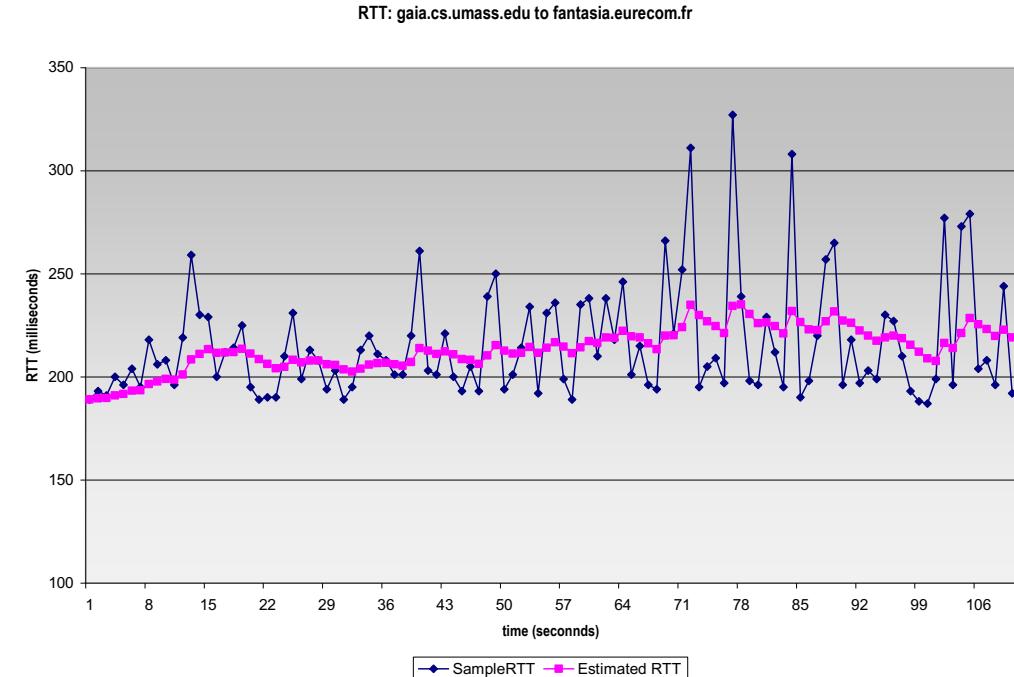
➤ 想一想：超时值设置多大比较合适呢？



如何设置超时值



- 为什么设置合理的超时值很重要：
 - 若超时值太小，容易产生不必要的重传
 - 若超时值太大，则丢包恢复的时间太长
- 直观上，超时值应大于RTT，但RTT是变化的
- 如何估计RTT：
 - RTT是变化的，需要实时测量从发出某个报文段到收到其确认报文段之间经过的时间（称 **SampleRTT**）
 - 由于SampleRTT波动很大，更有意义的是计算其平均值（称**EstimatedRTT**）
- 平均RTT的估算方法（指数加权移动平均）：
 - $\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$
 - 典型地， $\alpha = 0.125$



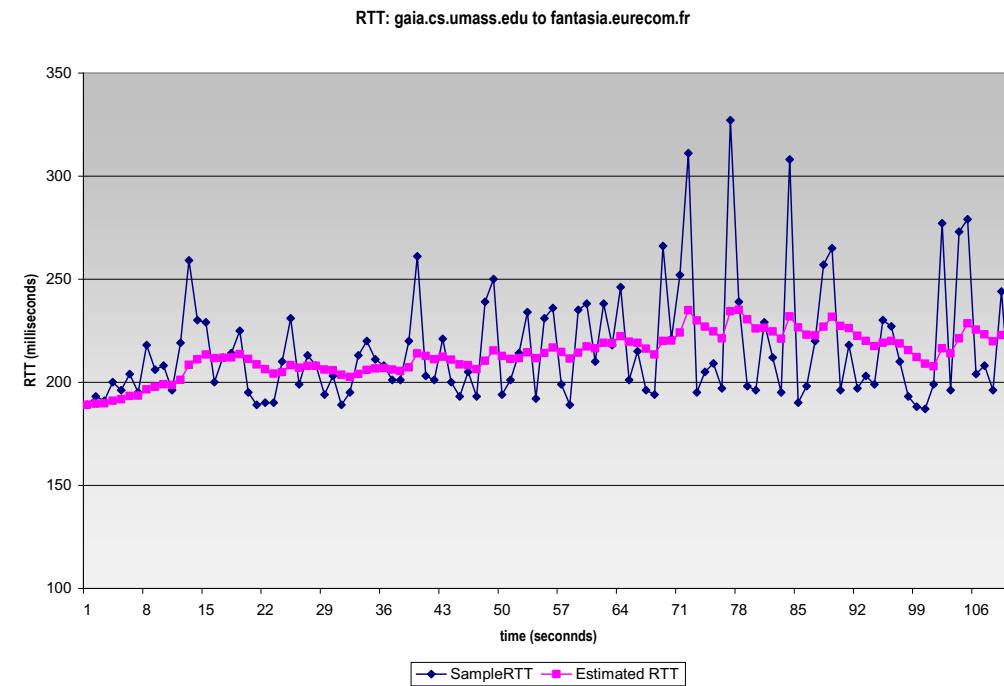


如何设置超时值



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 瞬时RTT和平均RTT有很大的偏差：
 - 需要在EstimatedRTT 上加一个“安全距离”，作为超时值
 - 安全距离的大小与RTT的波动幅度有关
- 估算SampleRTT 与 EstimatedRTT的偏差（称 DevRTT）：
 - $\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$
 - 典型地， $\beta = 0.25$
- 设置重传定时器的超时值：
 - $\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$





TCP确认的二义性



➤ TCP确认的二义性问题：

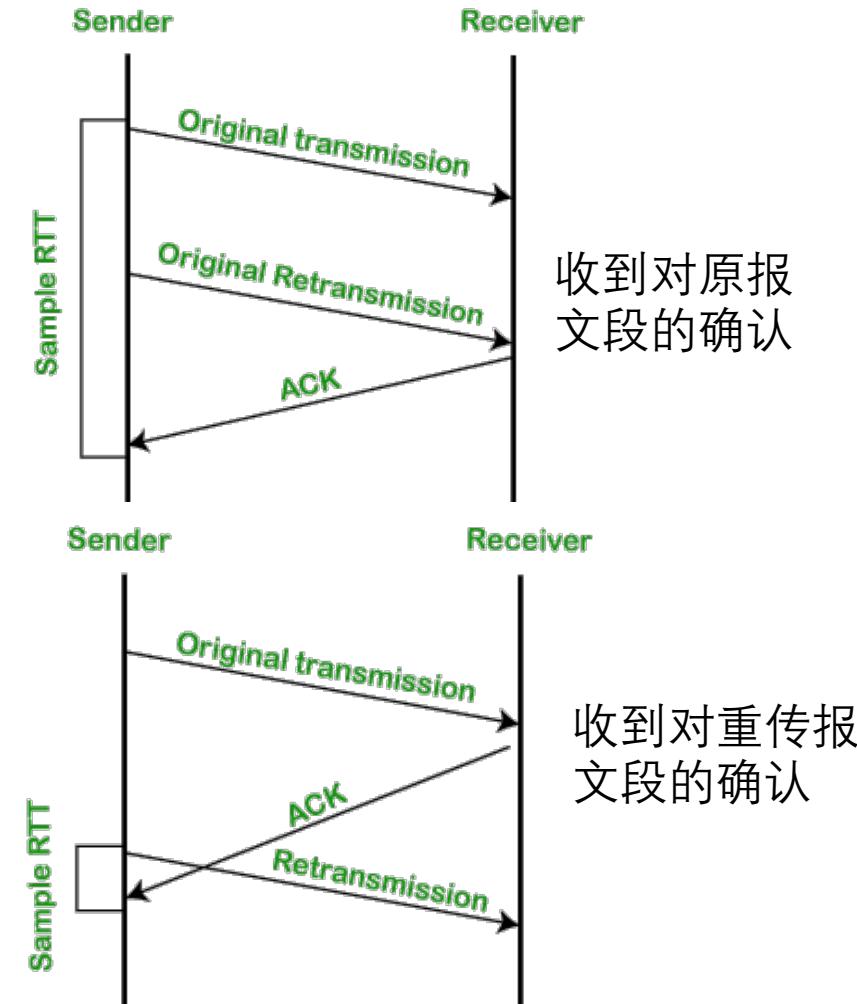
- 重传的TCP报文段使用与原报文段相同的序号
- 发送端收到确认后，无法得知是对哪个报文段进行的确认

➤ 二义性确认带来的问题：

- 对重传报文段测量的SampleRTT，可能不准确

➤ 解决方法：

- 忽略有二义性的确认，只对一次发送成功的报文段测量SampleRTT，并更新EstimatedRTT
- 当TCP重传一个段时，停止测量SampleRTT





➤ 简单忽略重传报文段的问题：

- 重传意味着超时值可能偏小了，需要增大
- 若简单忽略重传报文段（不更新EstimatedRTT），则超时值也不会更新，超时设置过小的问题没有解决

➤ 解决方法：

- 采用**定时器补偿策略**，发送方每重传一个报文段，就直接将**超时值增大一倍**（不依赖于RTT的更新）
- 若连续发生超时事件，超时值呈**指数增长**（至一个设定的上限值）



- 理论上，接收端只需区分两种情况：
 - 收到期待的报文段：发送更新的确认序号
 - 其它情况：重复当前的确认序号
- 推迟确认带来的问题：
 - 若延迟太大，会导致**不必要的重传**
 - 推迟确认造成**RTT估计不准确**
- 为减小通信量，TCP允许接收端**延迟确认（Delayed ACK）**：
 - 接收端可以在收到**若干个**报文段后，发送一个累积确认的报文段
- TCP协议规定：
 - 推迟确认的时间最多为**500ms (不同的操作系统参数不同)**
 - 接收方至少**每隔一个(每2个)**报文段使用正常方式进行确认



TCP接收端的事件和处理



中國人民大學
RENMIN UNIVERSITY OF CHINA

接收端事件	接收端动作
收到一个期待的报文段，且之前的报文段均已发送过确认	推迟发送确认，在500ms时间内若无下一个报文段到来，发送确认
收到一个期待的报文段，且前一个报文段被推迟确认	立即发送确认（估计RTT的需要）
收到一个失序的报文段（序号大于期待的序号），检测到序号间隙	立即发送确认（快速重传的需要），重复当前的确认序号
收到部分或全部填充间隙的报文段	若报文段始于间隙的低端，立即发送确认（推进发送窗口），更新确认序号



快速重传

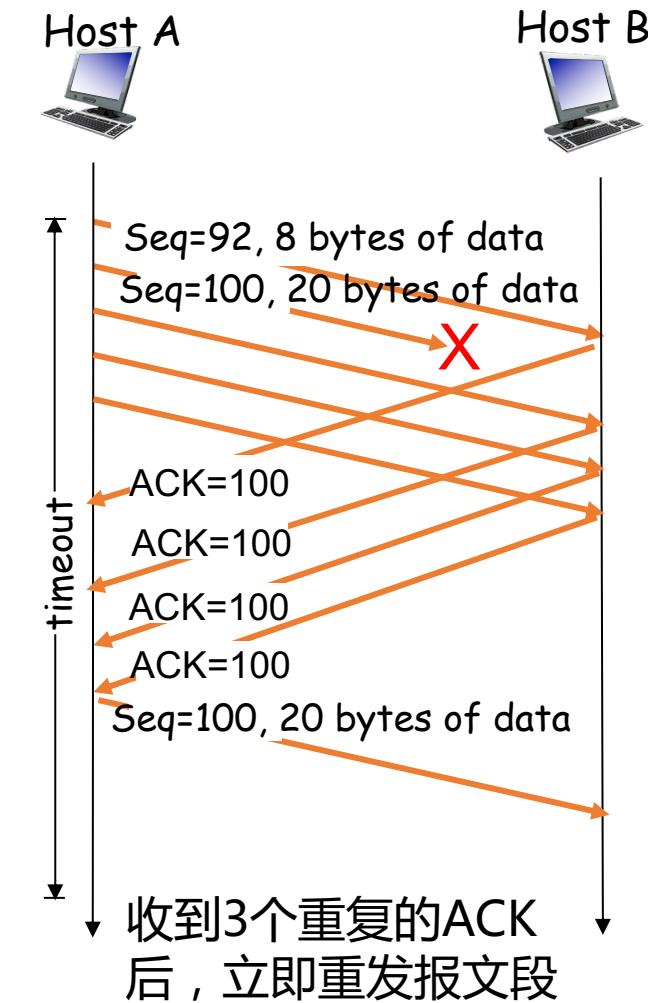
- 仅靠超时重发丢失的报文段，恢复太慢！
- **发送方可利用重复ACK检测报文段丢失：**
 - 发送方通常连续发送许多报文段
 - 若仅有个别报文段丢失，发送方将收到多个重复序号的ACK
 - 多数情况下IP按序交付分组，重复ACK极有可能因丢包产生

➤ TCP协议规定：

- 当发送方收到对同一序号的**3次重复确认**（**Duplicate ACK**）时，立即重发包含该序号的报文段

➤ 快速重传：

- 所谓快速重传，就是在定时器到期前重发丢失的报文段





小结



中國人民大學
RENMIN UNIVERSITY OF CHINA

- TCP可靠传输的设计要点：
 - 流水式发送报文段
 - 缓存失序的报文段
 - 采用累积确认
 - 只对最早未确认的报文段使用一个重传定时器
 - 超时后只重传包含最小序号的、未确认的报文段
- 以上措施可大量减少因ACK丢失、定时器过早超时引起的重传

- 超时值的确定：
 - 基于RTT估计超时值 + 定时器补偿策略
- 测量RTT：
 - 不对重传的报文段测量RTT
 - 确认延迟的时间不能过长
- 快速重传：
 - 收到3次重复确认，重发报文段



本章内容

6.1 概述和传输层服务

6.2 套接字编程

6.3 传输层复用和分用

6.4 无连接传输：UDP

6.5 面向连接的传输：TCP

6.6 理解网络拥塞

6.7 TCP拥塞控制

6.8 拥塞控制的发展

6.9 传输层协议的发展

1. TCP概述
2. TCP报文段结构
3. TCP可靠数据传输
4. **TCP流量控制**
5. TCP连接管理



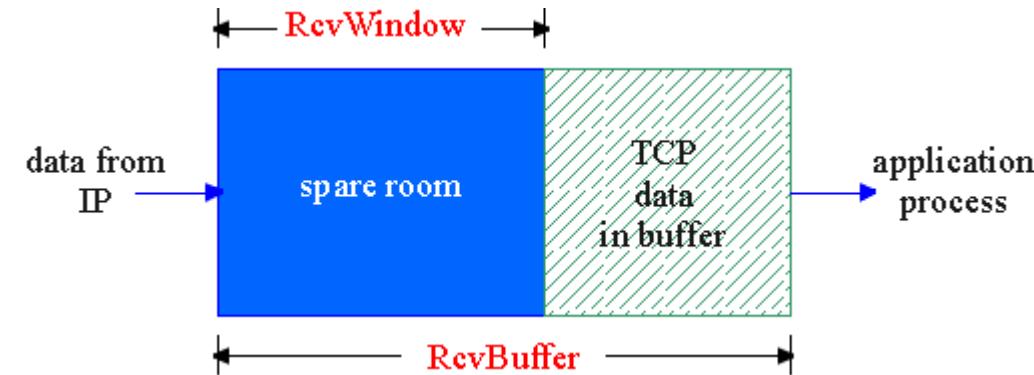


TCP的接收端：接收缓存



中國人民大學
RENMIN UNIVERSITY OF CHINA

- TCP接收端有一个接收缓存：
 - 接收端TCP将收到的数据放入接收缓存
 - 应用进程从接收缓存中读数据
 - **进入接收缓存的数据不一定被立即取走、取完**
 - 如果接收缓存中的数据未及时取走，后续到达的数据可能会因缓存溢出而丢失



- 流量控制：
 - 发送端TCP通过调节发送速率，不使接收端缓存溢出



TCP如何进行流量控制

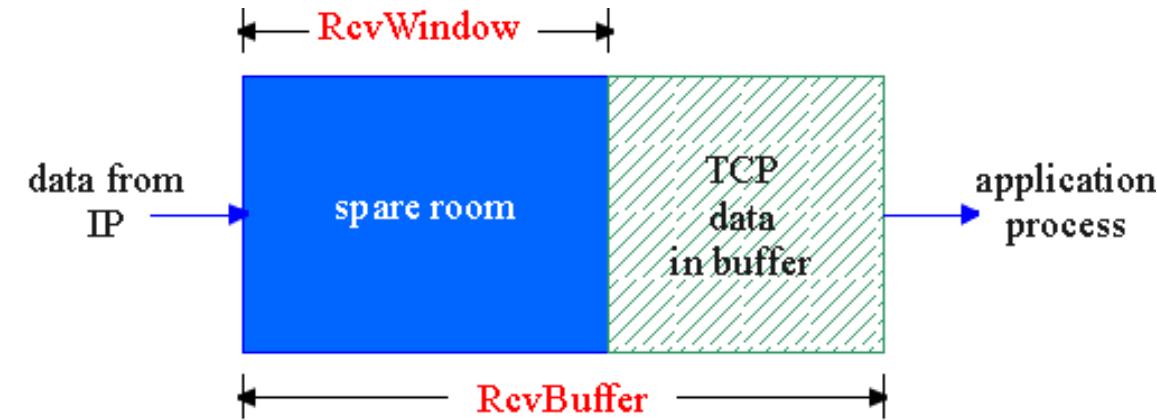


- 接收缓存中的可用空间称为**接收窗口**：

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

- 接收方将RcvWindow放在**TCP首部的窗口字段**中，向发送方通告接收缓存的可用空间
- 发送方限制**未确认的字节数 (bytes-in-flight) 不超过接收窗口的大小**，即：

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{RcvWindow}$$



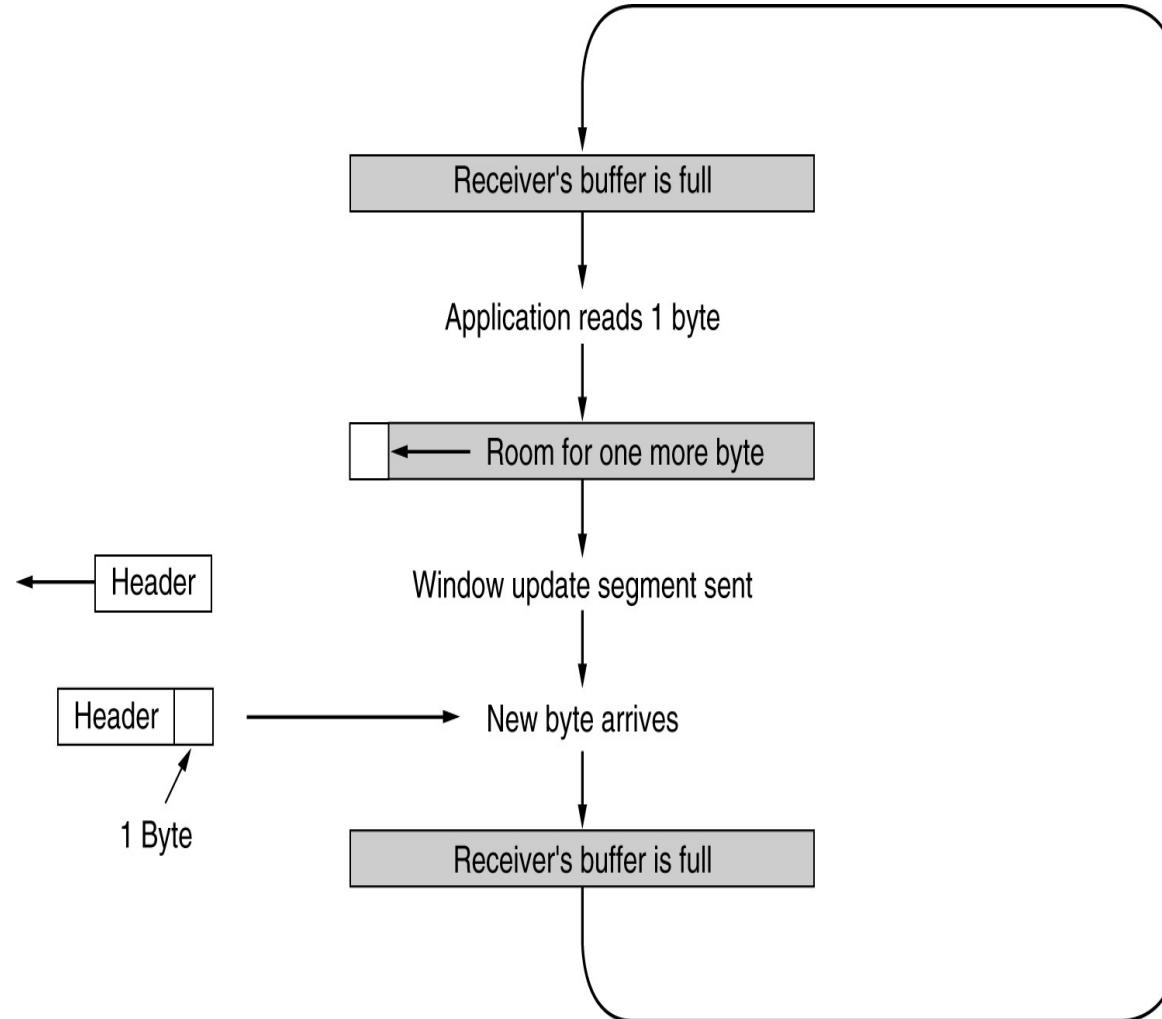
- 特别是，当接收方通告接收窗口为0时（**零窗口通告**），发送方必须停止发送



- 发送方/接收方对零窗口的处理：
 - 当接收窗口为0时，发送方必须停止发送
 - 当接收窗口变为**非0**时，接收方应通告增大的接收窗口
- 在TCP协议中，触发一次TCP传输需要满足以下三个条件之一：
 - 应用程序调用
 - 超时
 - 收到数据/确认
- 对于单向传输中的接收方，只有**第三个**条件能触发传输
- **当发送方停止发送后**，接收方不再收到数据，**如何触发**接收端发送“非零窗口通告”呢？
- TCP协议规定：
 - **发送方**收到“零窗口通告”后，可以发送“零窗口探测”报文段
 - 从而接收方可以发送包含接收窗口的响应报文段



- 当数据的发送速度很快、而消费速度很慢时，零窗口探测的简单实现带来以下问题：
 - 接收方不断发送微小窗口通告
 - 发送方不断发送很小的数据分组
 - 小包问题，大量带宽被浪费（报文头开销），传输效率低下
- 解决方案：
 - 接收方启发式策略
 - 发送方启发式策略





- 接收端避免糊涂窗口综合症的策略：
 - 通告零窗口之后，仅当窗口大小显著增加之后才发送更新的窗口通告
 - 什么是显著增加：窗口大小达到缓存空间的一半或者一个MSS，取两者的较小值
- TCP执行该策略的做法（双管齐下）：
 - 当窗口大小满足以上策略时，通告新的窗口大小
 - 当窗口大小不满足以上策略时，推迟发送确认，即Delayed ACK（但最多推迟500ms，且至少每隔一个报文段使用正常方式进行确认），寄希望于推迟间隔内有更多数据被消费



- 发送方避免糊涂窗口综合症的策略：
 - 发送方应**积聚足够多**的数据再发送，以防止发送太短的报文段
- 问题：发送方应等待多长时间？
 - 若等待时间不够，报文段会太短
 - 若等待时间过久，应用程序的时延会太长
 - 更重要的是，**TCP不知道**应用程序会不会在最近的将来生成更多的数据



发送方启发式策略：Nagle算法



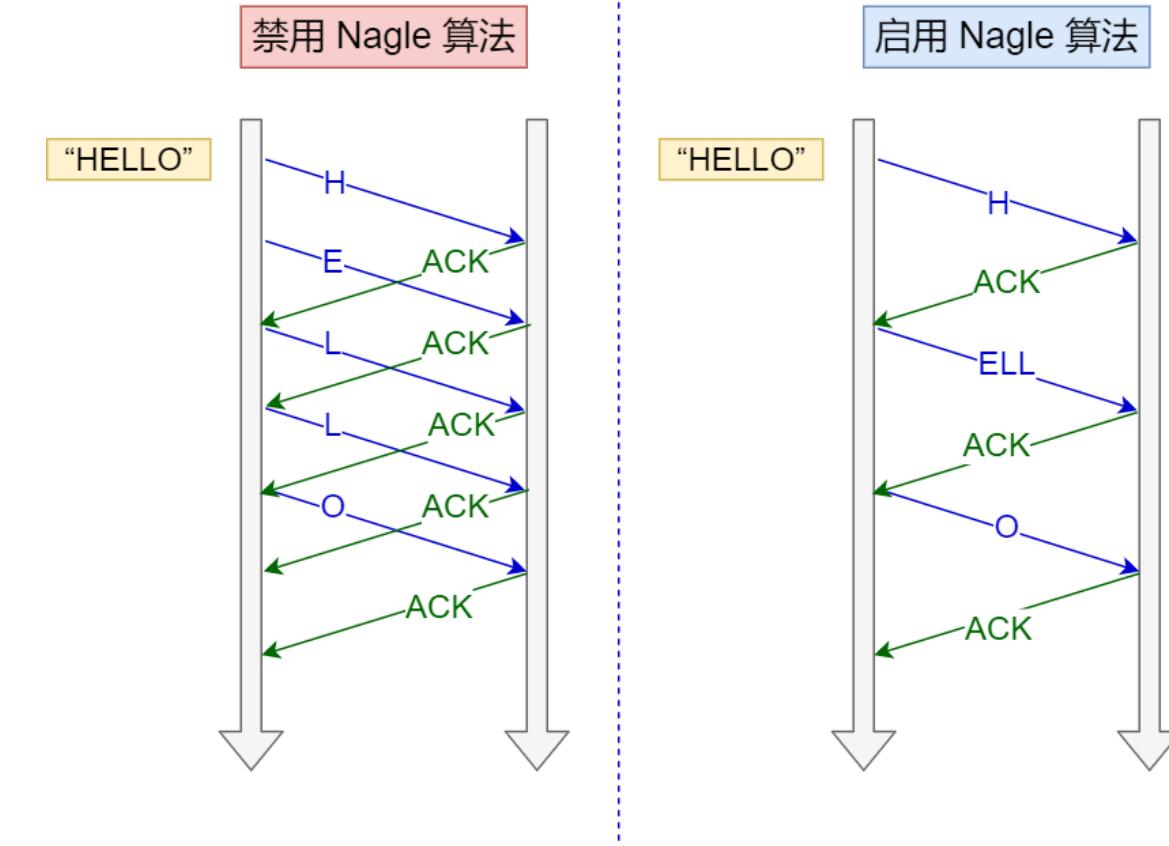
中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ Nagle算法的解决方法：

- 在新建连接上，当应用数据到来时，组成一个TCP段发送（哪怕只有一个字节）
- 在收到确认之前，后续到来的数据放在发送缓存中
- 当数据量达到一个MSS或上一次传输的确认到来（取两者的较小时间），用一个TCP段将缓存的字节全部发走

➤ Nagle算法的优点：

- 适应网络延时、MSS长度及应用速度的各种组合
- 常规情况下不会降低网络的吞吐量





Nagle算法 vs Delayed ACK

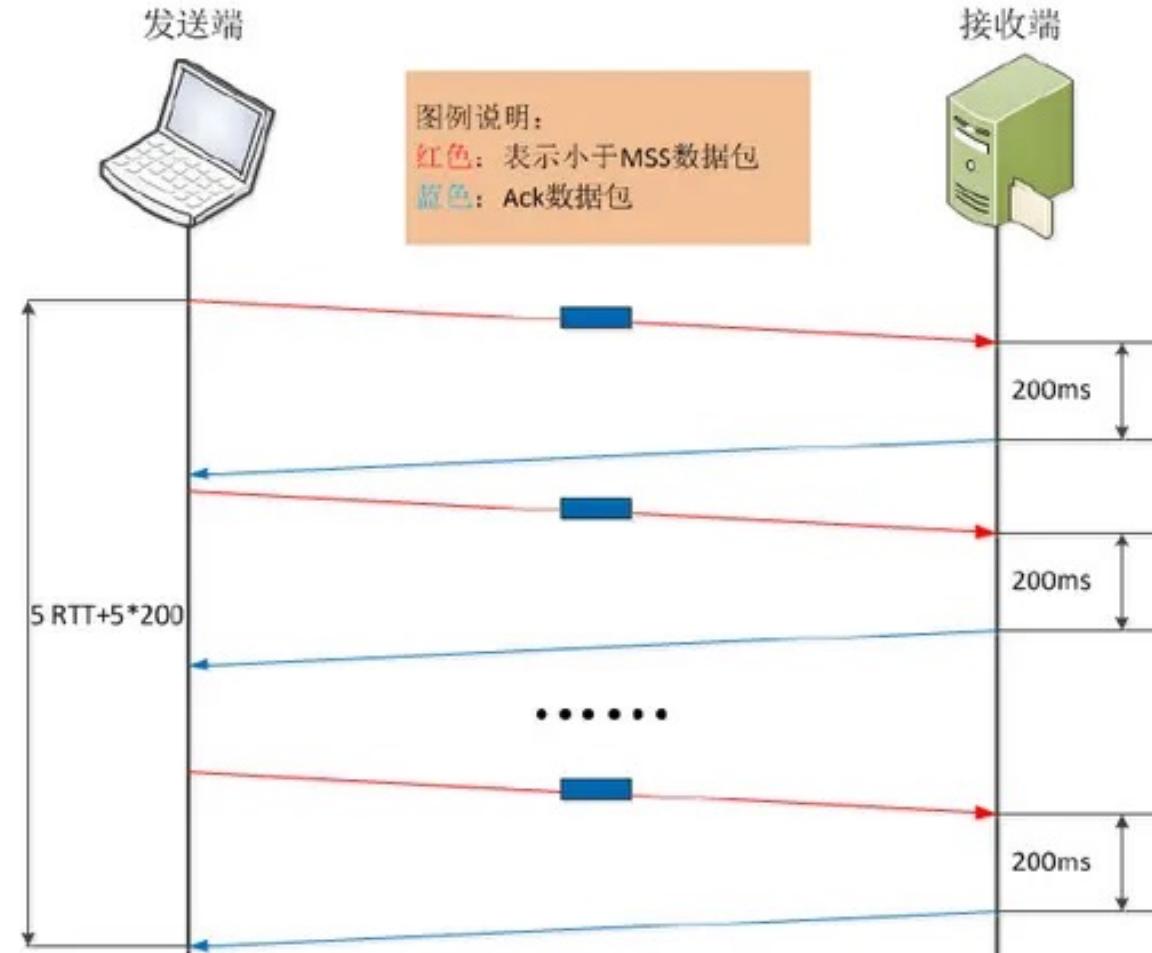


中國人民大學
RENMIN UNIVERSITY OF CHINA

- 同时打开Nagle算法和Delayed ACK，会发生什么？ $1+1>2$ ？

举例：发送端需要连续发送5个写操作（应用程序将数据写入到缓冲池的动作）的小报文

灾难发生了！

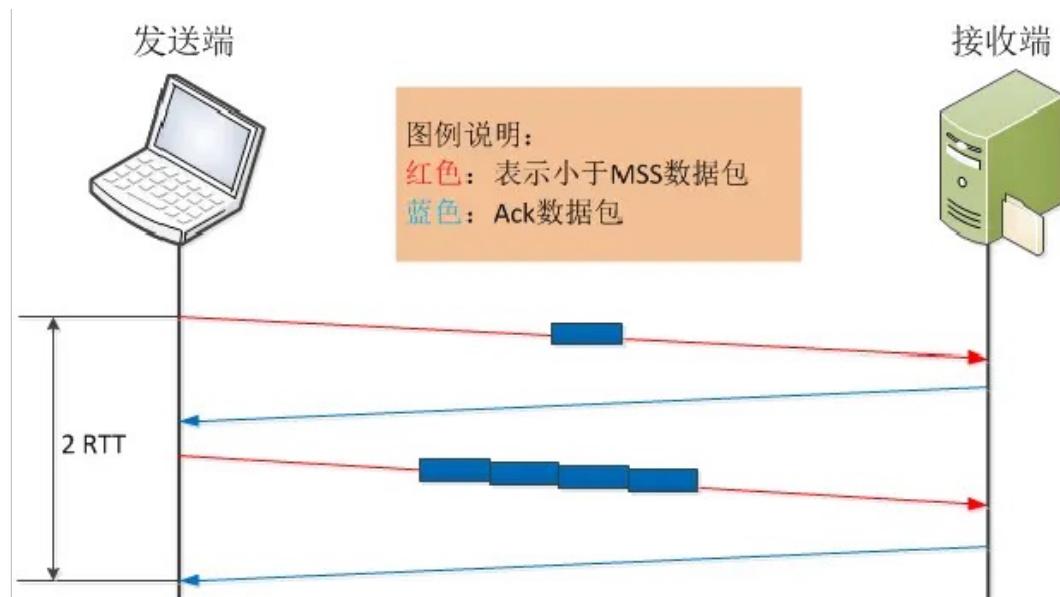




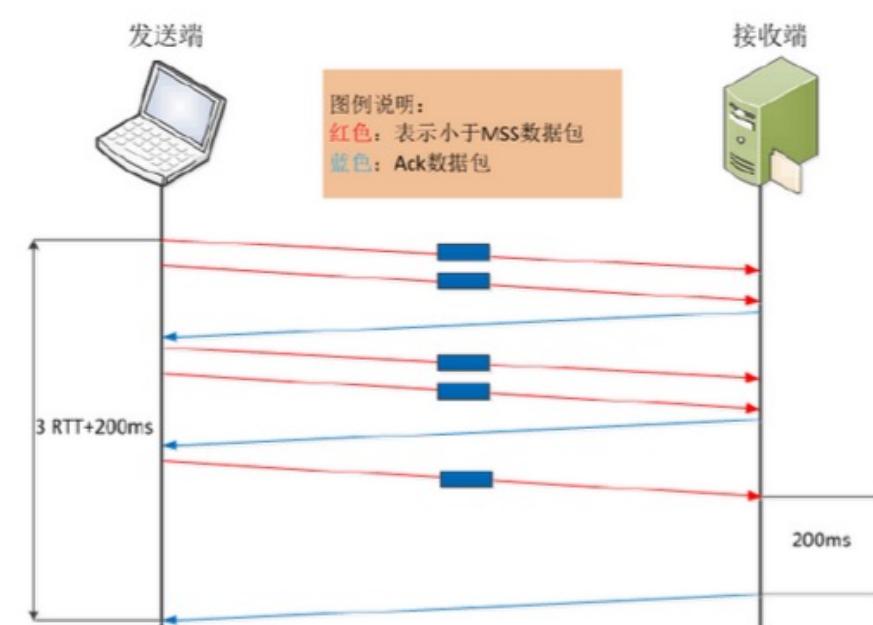
Nagle算法 vs Delayed ACK



- 关闭Delayed ACK: 对应的socket设置为TCP_QUICKACK
 - 例如 `setsockopt(fd, IPPROTO_TCP, TCP_QUICKACK, (int[]){1}, sizeof(int))`
- 关闭Nagle算法: 对应的socket设置为TCP_NODELAY
 - 例如 `setsockopt(fd, IPPROTO_TCP, TCP_NODELAY, (char*)&flag, sizeof(flag));`



打开Nagle算法，关闭Delayed ACK



关闭Nagle算法，打开Delayed ACK



➤ TCP接收端

- 使用显式的窗口通告，告知发送方可用的缓存空间大小
- 在接收窗口较小时，延迟发送确认
- 仅当接收窗口显著增加时，通告新的窗口大小

➤ TCP发送端

- 使用Nagle算法确定发送时机
- 使用接收窗口限制发送的数据量， $\text{bytes-in-flight} < \text{RcvWindow}$



本章内容

6.1 概述和传输层服务

6.2 套接字编程

6.3 传输层复用和分用

6.4 无连接传输：UDP

6.5 面向连接的传输：TCP

6.6 理解网络拥塞

6.7 TCP拥塞控制

6.8 拥塞控制的发展

6.9 传输层协议的发展

1. TCP概述
2. TCP报文段结构
3. TCP可靠数据传输
4. TCP流量控制
5. TCP连接管理

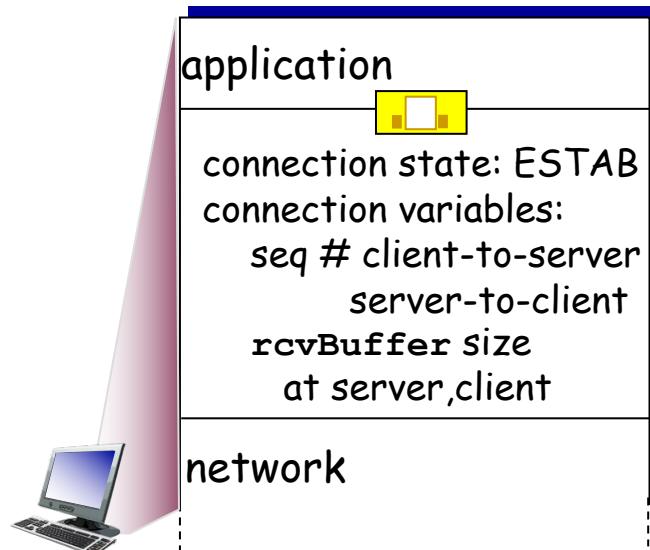




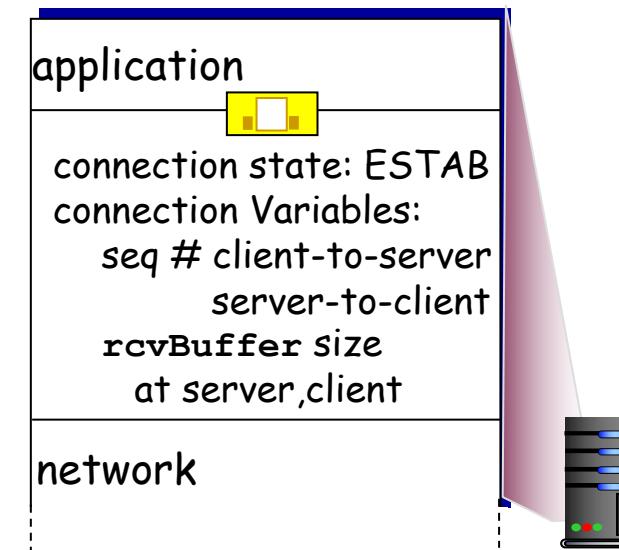
建立TCP连接



- 建立一条TCP连接需要确定两件事：
 - 双方都同意建立连接（知晓另一方想建立连接）
 - 初始化连接参数（序号，MSS等）



```
Socket clientSocket =  
    newSocket("hostname", "port  
    number");
```



```
Socket connectionSocket =  
    welcomeSocket.accept();
```

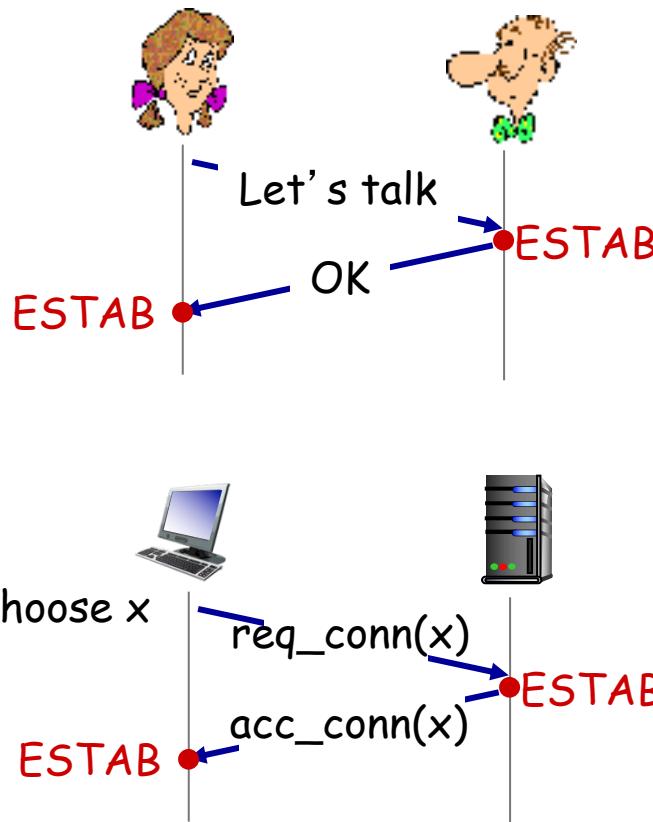


两次握手建立连接



中國人民大學
RENMIN UNIVERSITY OF CHINA

两次握手建立连接的类比



➤ 问题:

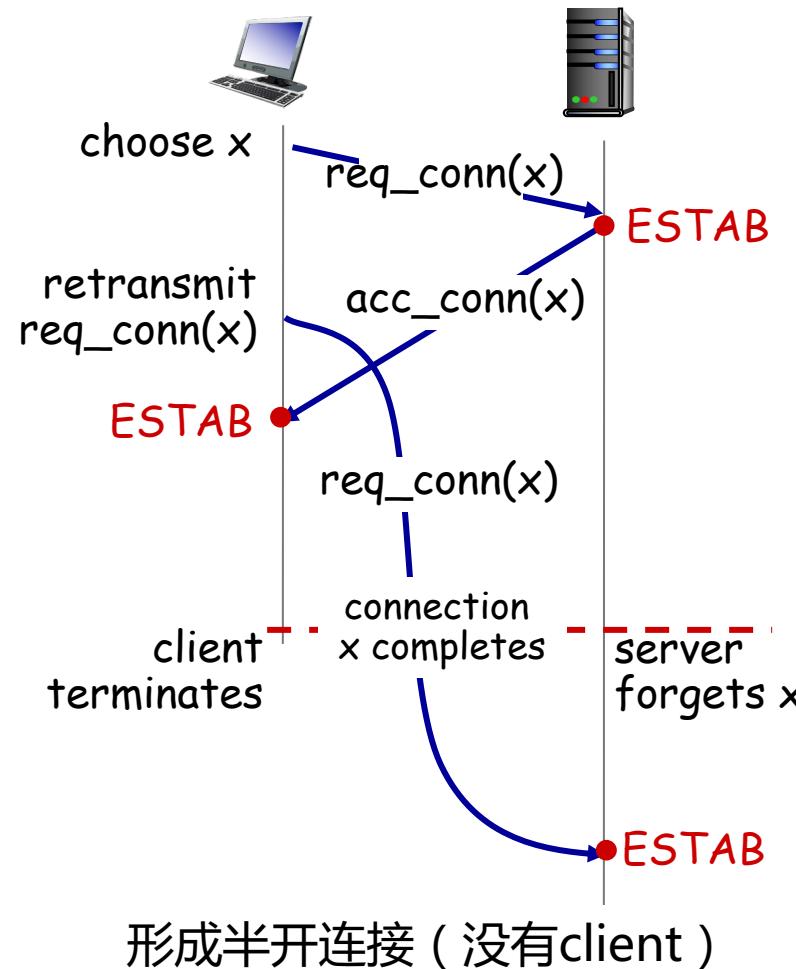
- 在网络中，2次握手总是可行的吗？

➤ 在一个不可靠的网络中，总会有一些意外发生：

- 包传输延迟变化很大
- 存在重传的报文段
- 存在报文重排序



两次握手失败的情形：客户端没了？！





TCP三次握手建立连接



中國人民大學
RENMIN UNIVERSITY OF CHINA

客户状态

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

LISTEN

```
clientSocket.connect((serverName,serverPort))
```

SYNSENT

choose init seq num, x
send TCP SYN msg



ESTAB

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

SYNbit=1, Seq=x

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

ACKbit=1, ACKnum=y+1

服务器状态

```
serverSocket = socket(AF_INET,SOCK_STREAM)  
serverSocket.bind((),serverPort)  
serverSocket.listen(1)  
connectionSocket, addr = serverSocket.accept()
```

LISTEN

SYN RCVD

choose init seq num, y
send TCP SYNACK
msg, acking SYN

received ACK(y)
indicates client is live

ESTAB



TCP三次握手建立连接



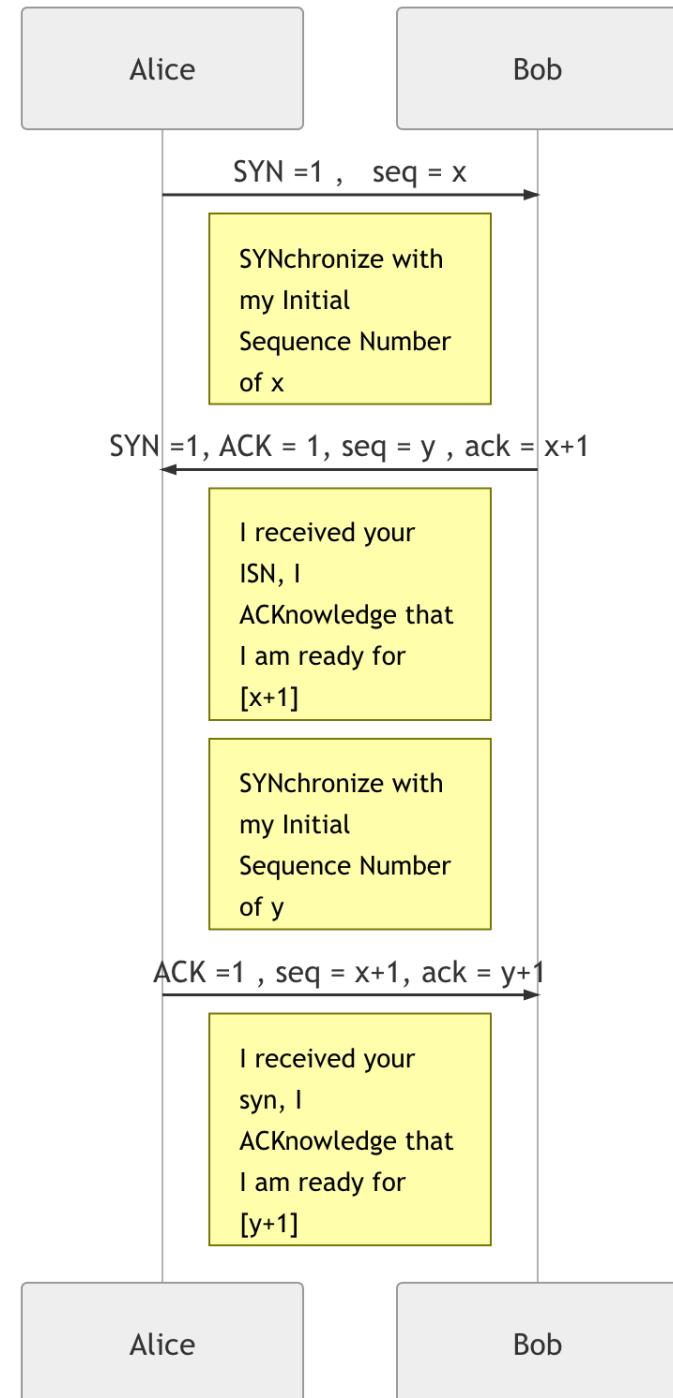
中國人民大學
RENMIN UNIVERSITY OF CHINA

1. 客户TCP发送SYN 报文段 (SYN=1, ACK=0)
 - 给出客户选择的起始序号 (Initial Sequence Number, ISN)
 - 不包含数据
2. 服务器TCP发送SYNACK报文段 (SYN=ACK=1) (服务器端分配缓存和变量)
 - 给出服务器选择的起始序号
 - 确认客户的起始序号
 - 不包含数据
3. 客户发送ACK报文段 (SYN=0 , ACK=1) (客户端分配缓存和变量)
 - 确认服务器的起始序号
 - 可能包含数据



重新思考：为什么三次握手？

- 三次握手的过程即是通信双方相互告知序列号起始值，并确认对方已经收到了序列号起始值的必经步骤
- 如果只是两次握手，至多只有连接发起方的起始序列号能被确认，另一方选择的序列号则得不到确认

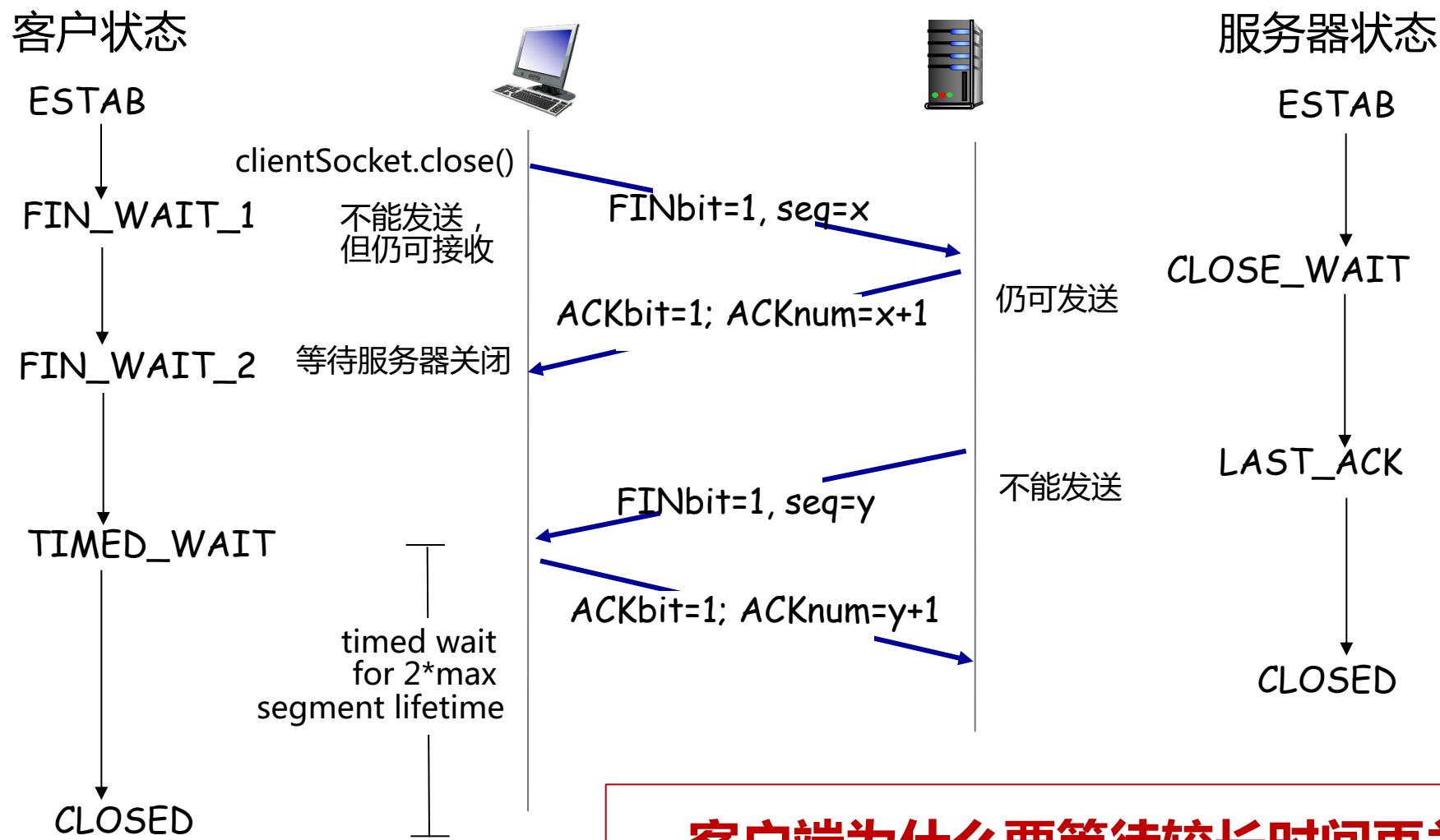




关闭TCP连接



中國人民大學
RENMIN UNIVERSITY OF CHINA

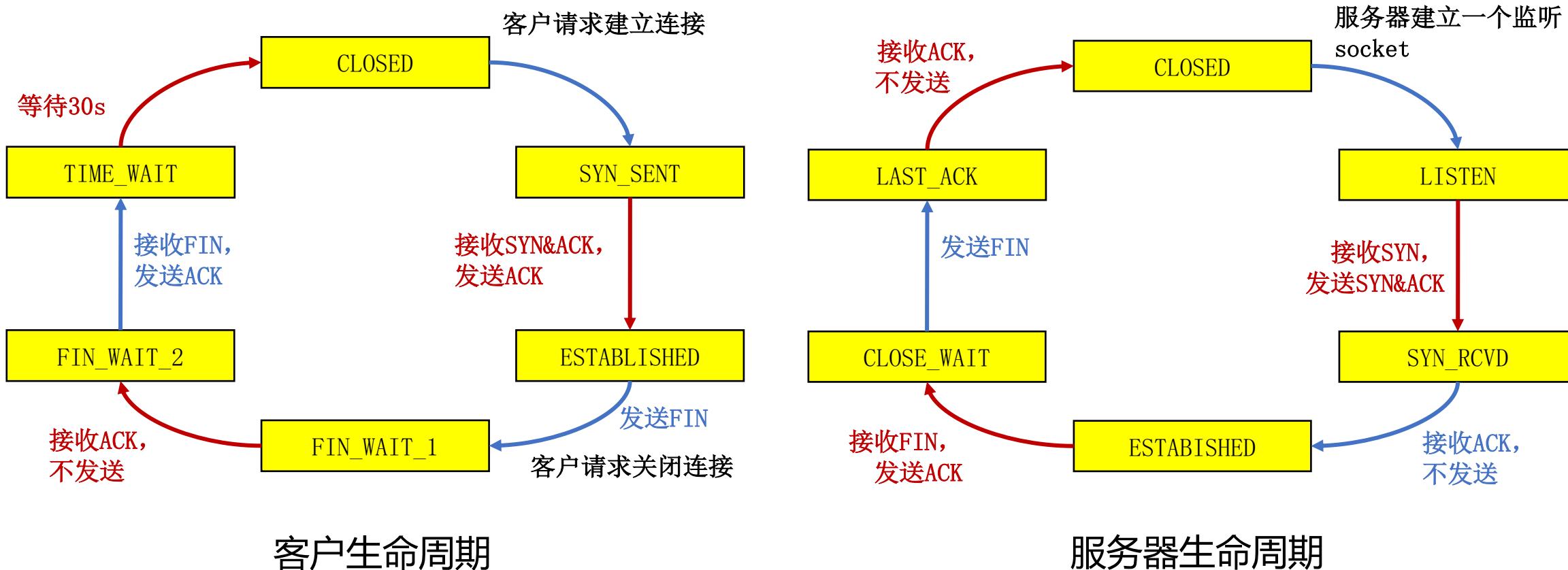




客户/服务器经历的TCP状态序列



中國人民大學
RENMIN UNIVERSITY OF CHINA

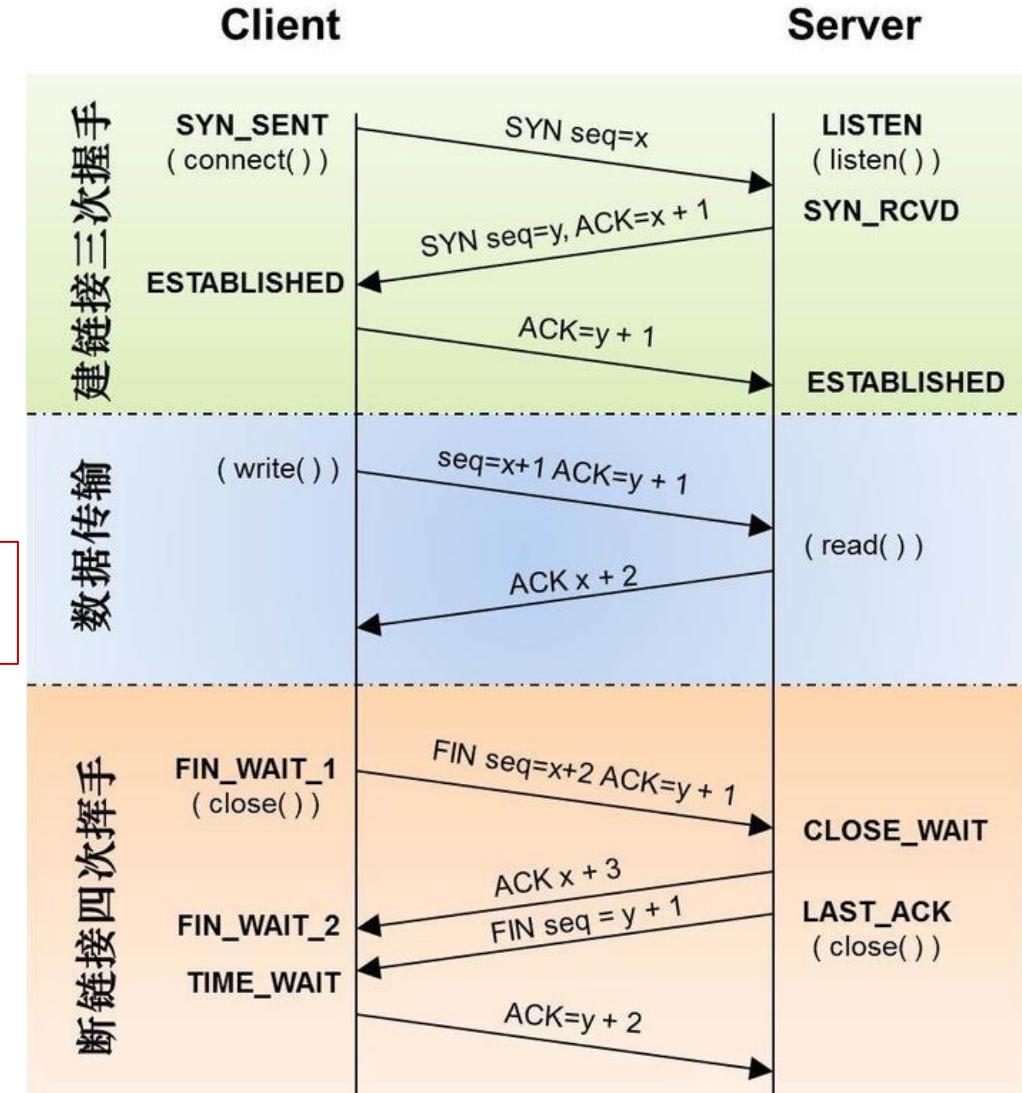




思考：客户端又没了？！

- 如果意外断开，客户端并没有正常关闭socket，怎么办？

KeepAlive机制





- Server端检测到超过一定时间（默认是 7,200,000 milliseconds，也就是2个小时）没有数据传输，会向Client端发送一个keep-alive packet，此时client端应该为以下三种情况之一：
 - Client端仍然存在，网络连接状况良好。此时Client端会返回一个ACK。server端接收到ACK后重置计时器（复位存活定时器），
 - 客户端异常关闭，或是网络断开。Server周期性（系统默认为1000 ms）发送keep-alive packet，并且重复发送一定次数
 - 客户端曾经崩溃，但已经重启。这种情况下，服务器将会收到对其存活探测的响应，但该响应是一个复位，从而引起服务器对连接的终止



KeepAlive机制



中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ 对于应用程序来说，2小时的空闲时间太长。因此，我们需要手工开启Keepalive功能并设置合理的Keepalive参数：

- 全局设置可更改/etc/sysctl.conf,加上:

net.ipv4.tcp_keepalive_intvl = 20

net.ipv4.tcp_keepalive_probes = 3

net.ipv4.tcp_keepalive_time = 60

- 在程序中设置如下:

```
int keepAlive = 1; // 开启keepalive属性
int keepIdle = 60; // 如该连接在60秒内没有任何数据往来，则进行探测
int keepInterval = 5; // 探测时发包的时间间隔为5 秒
int keepCount = 3; // 探测尝试的次数.如果第1次探测包就收到响应了,则后2次的不再发.

setsockopt(rs, SOL_SOCKET, SO_KEEPALIVE, (void *)&keepAlive, sizeof(keepAlive));
setsockopt(rs, SOL_TCP, TCP_KEEPIDLE, (void*)&keepIdle, sizeof(keepIdle));
setsockopt(rs, SOL_TCP, TCP_KEEPINTVL, (void *)&keepInterval, sizeof(keepInterval));
setsockopt(rs, SOL_TCP, TCP_KEEPCNT, (void *)&keepCount, sizeof(keepCount));
```

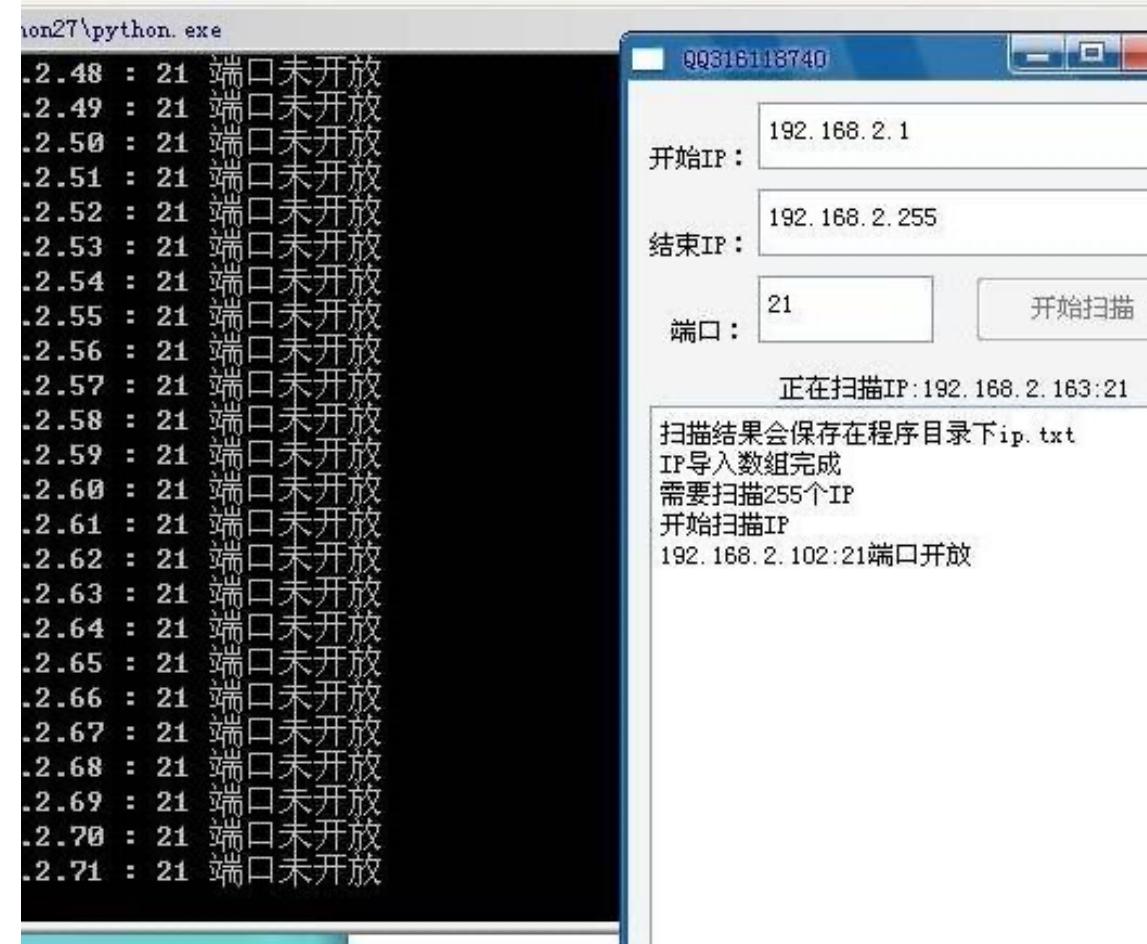
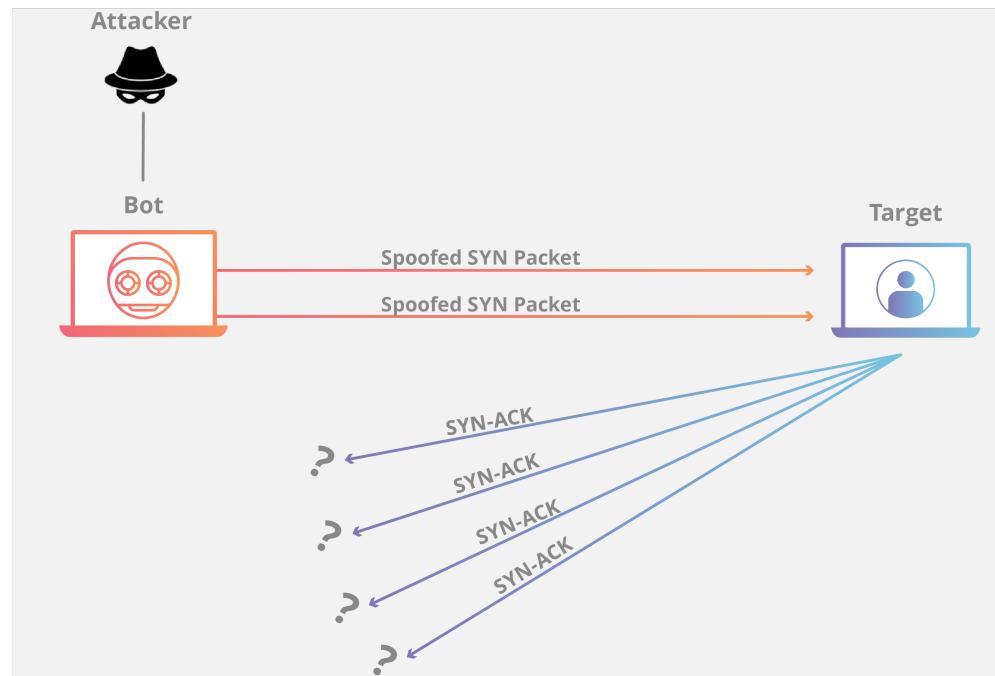


TCP连接过程中的安全性问题



中國人民大學
RENMIN UNIVERSITY OF CHINA

- SYN洪泛攻击
- 端口扫描





➤ TCP实现的问题：

- 服务器在收到SYN段后，发送SYNACK段，分配资源
- 若未收到ACK段，服务器超时后重发SYNACK段
- 服务器等待一段时间（称SYN超时）后丢弃未完成的连接，**SYN超时的典型值为30秒~120秒**

➤ SYN洪泛攻击：

- 攻击者采用伪造的源IP地址，向服务器发送大量的SYN段，却不发送ACK段
- 服务器为维护一个巨大的半连接表耗尽资源，导致无法处理正常客户的连接请求，表现为服务器停止服务



- TCP端口扫描的原理：
 - 扫描程序依次与目标机器的各个端口建立TCP连接
 - 根据获得的响应来收集目标机器信息
- 在典型的TCP端口扫描过程中，发送端向目标端口发送SYN报文段：
 - 若收到SYNACK段，表明目标端口上有服务在运行
 - 若收到RST段，表明目标端口上没有服务在运行
 - 若什么也没收到，表明路径上有防火墙，有些防火墙会丢弃来自外网的SYN报文段



本章内容



中國人民大學
RENMIN UNIVERSITY OF CHINA

6.1 概述和传输层服务

6.2 无连接传输 : UDP

6.3 面向连接的传输 : TCP

6.4 理解网络拥塞

6.5 TCP拥塞控制

6.6 拥塞控制的发展

6.7 传输层协议的发展



交通拥堵：

- 起因：大量汽车短时间内进入路网，超出路网的承载能力
- 表现：道路通行能力下降，车速变慢，甚至完全停滞
- 措施：减少车辆进入路网（交通管制）

网络拥塞：

- 起因：大量分组短时间内进入网络，超出网络的处理能力
- 表现：分组延迟增大，网络吞吐量下降，甚至降为0
- 措施：减少分组进入网络（拥塞控制）

流量控制与拥塞控制的异同：

- 流量控制：限制发送速度，使不超过接收端的处理能力
- 拥塞控制：限制发送速度，使不超过网络的处理能力



➤ 网络拥塞造成：

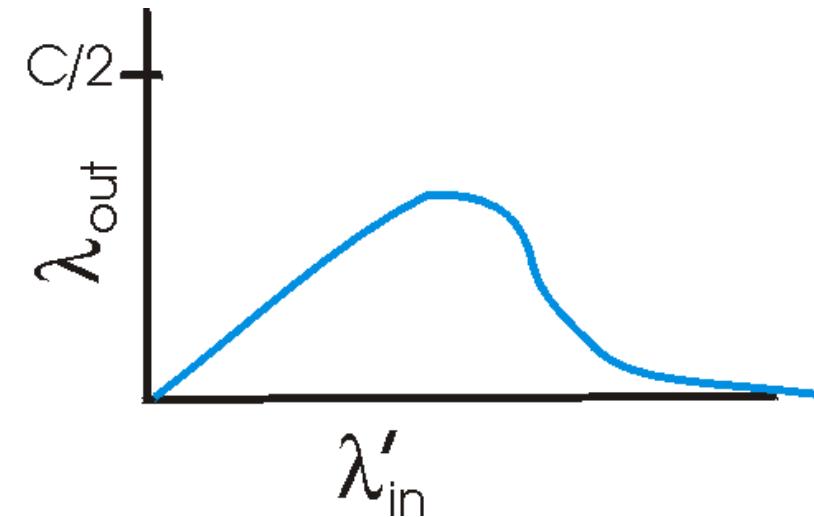
- 丢包：由路由器缓存溢出造成
- 分组延迟增大：链路接近满载造成

➤ 大量网络资源用于：

- 重传丢失的分组
- (不必要地) 重传延迟过大的分组
- 转发最终被丢弃的分组

➤ 结果：

- 进入网络的负载很重，网络吞吐量却很低





➤ 网络辅助的拥塞控制

- 路由交换设备向端系统提供显式的反馈，例如：
 - 设置拥塞指示比特
 - 给出发送速率指示
- ATM、X.25、数据中心采用此类方法

➤ 端到端拥塞控制

- 网络层不向端系统提供反馈
- 端系统通过观察丢包和延迟，自行推断拥塞的发生
- TCP采用此类方法



本章内容



中國人民大學
RENMIN UNIVERSITY OF CHINA

6.1 概述和传输层服务

6.2 无连接传输 : UDP

6.3 面向连接的传输 : TCP

6.4 理解网络拥塞

6.5 TCP拥塞控制

6.6 拥塞控制的发展

6.7 传输层协议的发展



TCP拥塞控制要解决的问题



中國人民大學
RENMIN UNIVERSITY OF CHINA

- TCP使用端到端拥塞控制机制：
 - 发送方根据自己感知的网络拥塞程度，限制其发送速率
- 需要回答三个问题：
 - 发送方如何感知网络拥塞？
 - 发送方采用什么机制来限制发送速率？
 - 发送方感知到网络拥塞后，采取什么策略调节发送速率？



发送方如何感知拥塞？

- 发送方利用丢包事件感知拥塞：
 - 拥塞造成丢包和分组延迟增大
 - 无论是丢包还是分组延迟过大，对于发送端来说都是**丢包了**
- 丢包事件包括：
 - 重传定时器超时
 - 发送端收到3个重复的ACK

发送方采用什么机制限制发送速率？

- 发送方使用**拥塞窗口**cwnd限制已发送未确认的数据量：

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

$$\text{rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ Bytes/sec}$$

- cwnd随发送方感知的网络拥塞程度而变化

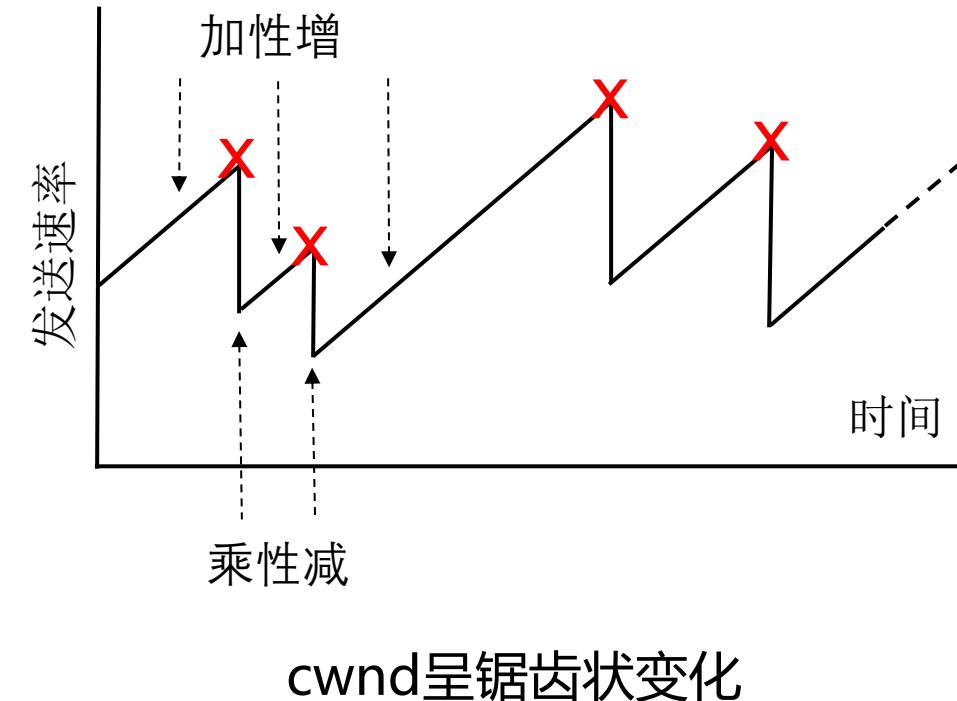


➤ 乘性减 (Multiplicative Decrease)

- 发送方检测到丢包后，将cwnd的大小减半（但不能小于一个MSS）
- 目的：迅速减小发送速率，缓解拥塞

➤ 加性增 (Additive Increase)

- 若无丢包，每经过一个RTT，将cwnd增大一个MSS，直到检测到丢包
- 目的：缓慢增大发送速率，避免振荡





TCP慢启动



中國人民大學
RENMIN UNIVERSITY OF CHINA

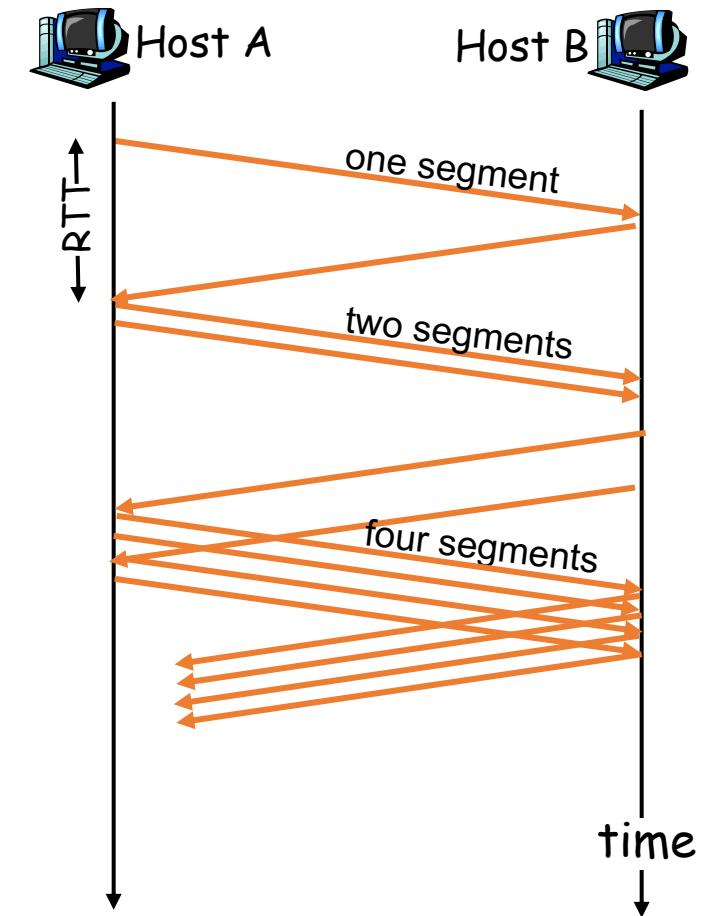
- 在新建的连接（或沉寂了一段时间的连接）上，以什么速率发送数据（此时接收窗口达最大值）？
- 早期的TCP协议：
 - 发送端仅以接收窗口大小限制发送速率，网络经常因为拥塞而崩溃！
- 采用“加性增”增大发送窗口，太慢！
 - 在新建连接上，令 $cwnd = 1 \text{ MSS}$ ，起始速度= MSS/RTT
 - 然而，网络中的可用带宽可能远大于 MSS/RTT
- 慢启动的基本思想：
 - 在新建连接上指数增大 $cwnd$ ，直至检测到丢包（此时终止慢启动）
 - 希望迅速增大 $cwnd$ 至可用的发送速度



慢启动的实施



- 慢启动的策略：
 - 每经过一个RTT，将cwnd加倍
- 慢启动的具体实施：
 - **每收到一个ACK段，cwnd增加一个MSS**
 - 只要发送窗口允许，发送端可以立即发送下一个报文段
- 特点：
 - 以一个很低的速率开始，按指数增大发送速率
- 慢启动比谁“慢”？
 - 与早期TCP按接收窗口发送数据的策略相比，采用慢启动后发送速率的增长较慢





区分不同的丢包事件



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 超时和收到3个重复的ACK，它们反映出来的网络拥塞程度是一样的吗？**当然不一样！**
 - **超时**：说明网络交付能力很差
 - **收到3个重复的ACK**：说明网络仍有一定的交付能力
 - 目前的TCP实现区分上述两种不同的丢包事件
-
- 收到3个重复的ACK：
 - 将cwnd降至一半
 - 使用AIMD调节cwnd
 - 超时：
 - 设置门限 =cwnd/2
 - cwnd=1MSS
 - 使用慢启动增大cwnd至门限
 - 使用AIMD调节cwnd

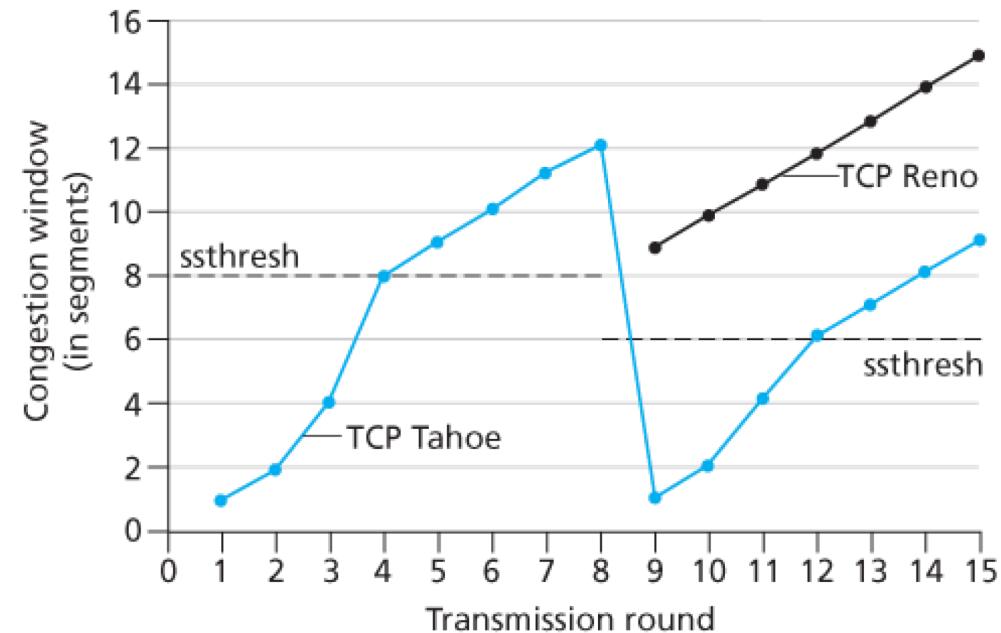


TCP拥塞控制的实现



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 发送方维护变量ssthresh
- 发生丢包时， $ssthresh = cwnd/2$
- **ssthresh是从慢启动转为拥塞避免的分水岭：**
 - $cwnd$ 低于门限时，执行慢启动
 - $cwnd$ 高于门限：执行拥塞避免
- 拥塞避免阶段，拥塞窗口线性增长：
 - 每当收到ACK， $cwnd = cwnd + MSS * (MSS/cwnd)$



- 检测到3个重复的ACK后：
 - TCP Reno实现： $cwnd = ssthresh + 3 (MSS)$ ，线性增长
 - TCP Tahoe实现： $cwnd = 1 MSS$ ，慢启动



TCP Reno发送端的事件与动作



中國人民大學
RENMIN UNIVERSITY OF CHINA

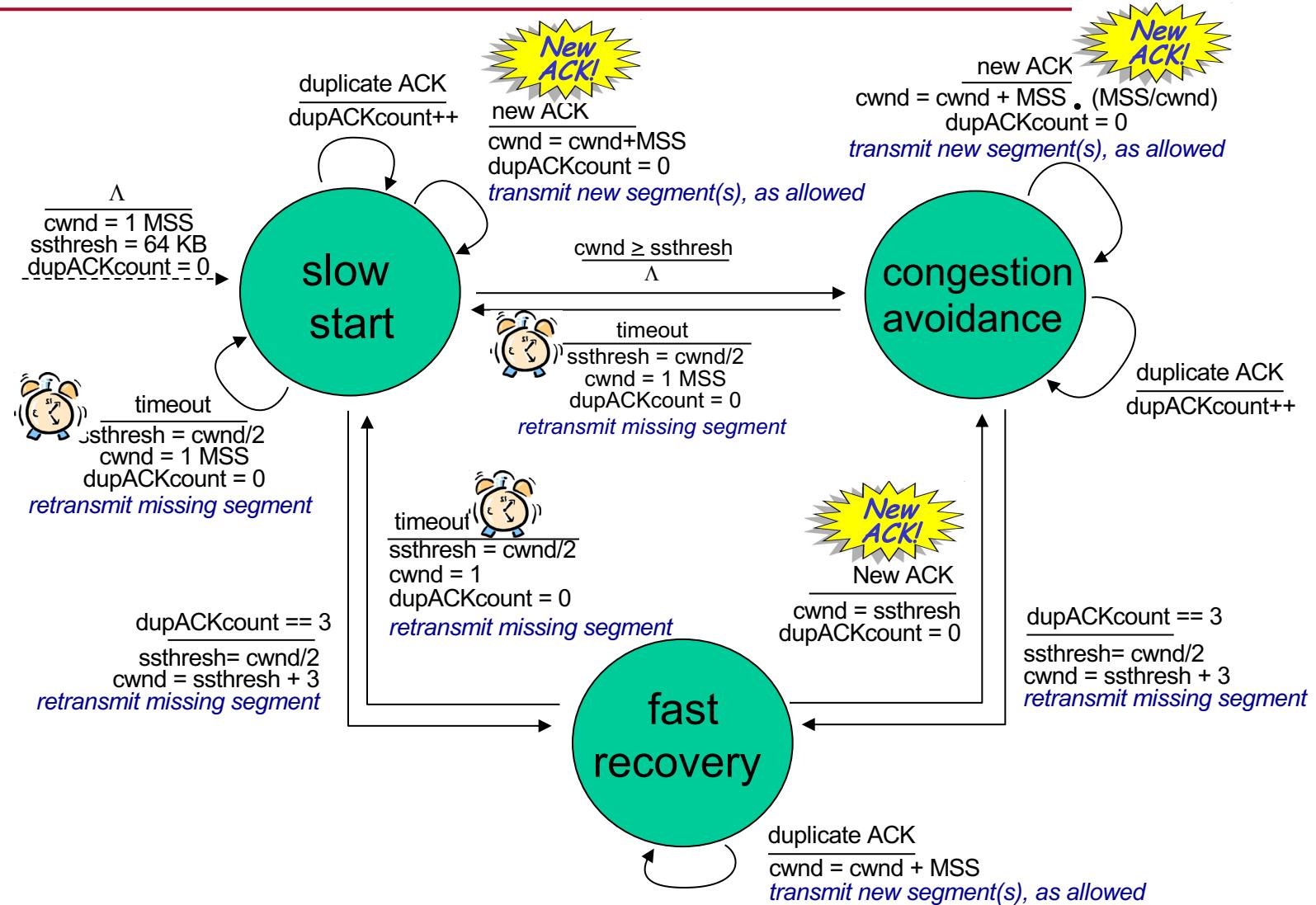
State	Event	TCP Sender Action	Commentary
慢启动 (SS)	收到新的确认	$cwnd = cwnd + MSS,$ If ($cwnd > ssthresh$) set state to "Congestion Avoidance"	每经过一个RTT , cwnd 加倍
拥塞避免 (CA)	收到新的确认	$cwnd = cwnd + MSS * (MSS/cwnd)$	每经过一个RTT , cwnd 增加一个MSS
SS or CA	收到3个重复的确认	$ssthresh = cwnd/2,$ $cwnd = Threshold+3 (MSS),$ Set state to "Congestion Avoidance"	cwnd减半，然后线性增长
SS or CA	超时	$ssthresh = cwnd/2,$ $cwnd = 1 MSS,$ Set state to "Slow Start"	cwnd降为一个MSS，进入慢启动
SS or CA	收到一个重复的确认	统计收到的重复确认数	cwnd 和 ssthresh 都不变



TCP拥塞控制状态机



中國人民大學
RENMIN UNIVERSITY OF CHINA



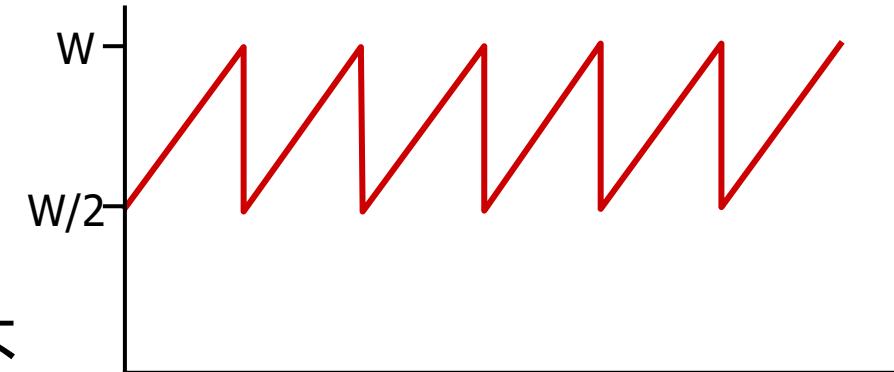


TCP连接的吞吐量



一个长期存活的TCP连接，平均吞吐量是多少？

- 忽略慢启动阶段（该阶段时间很短），只考虑拥塞避免阶段
- 令 W =发生丢包时的拥塞窗口，此时有：
$$\text{throughput} = W/\text{RTT}$$
- 发生丢包后调整 $cwnd=W/2$ ，此时有：
$$\text{throughput}=W/2\text{RTT}$$
- 假设在TCP连接的生命期内，RTT 和 W 几乎不变，有：
$$\text{Average throughout}=0.75 W/\text{RTT}$$





关于TCP和UDP的一些思考



中國人民大學
RENMIN UNIVERSITY OF CHINA

1. TCP提供可靠传输、流量控制、拥塞控制，而UDP都没有，能否说TCP服务优于UDP服务？
2. 多媒体应用希望的传输层服务是：带宽有保证、延迟有保证、顺序有保证、但能忍受一些丢包，TCP或UDP能够满足多媒体应用的需求吗？
3. 现实中的多媒体应用，有的使用TCP、有的使用UDP，它们分别出于什么考虑选择采用TCP或UDP？
4. 如何阻止UDP流量压制TCP流量？



本章内容



中國人民大學
RENMIN UNIVERSITY OF CHINA

6.1 概述和传输层服务

6.2 套接字编程

6.3 传输层复用和分用

6.4 无连接传输：UDP

6.5 面向连接的传输：TCP

6.6 理解网络拥塞

6.7 TCP拥塞控制

6.8 拥塞控制的发展

6.9 传输层协议的发展

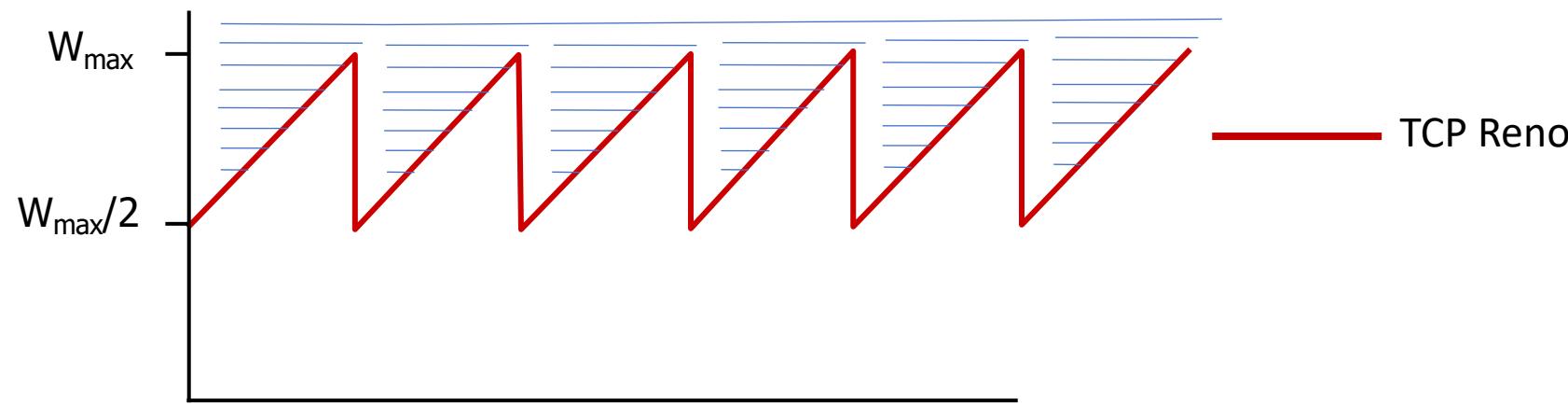
1. TCP CUBIC
2. Google BBR
3. Data Center TCP



经典TCP拥塞控制的性能问题



- 核心问题：在探测满载窗口的过程中，如何增加拥塞窗口以尽可能利用网络带宽？
- TCP Reno线性增大拥塞窗口，探测当前可用网络带宽，即每经过一个RTT，拥塞窗口增加一个MSS；当端到端时延带宽乘积（BDP）较大时，拥塞窗口增长过慢，导致信道无法满载



阴影部分表示信道带宽未被充分利用



- BIC算法对满载窗口进行二分查找：
 - 如发生丢包时窗口大小是 W_1 ，为保持满载而不丢包，满载窗口应小于 W_1
 - 如检测到丢包并将窗口乘性减小为 W_2 ，则满载窗口应大于 W_2
- 窗口更新受ACK时钟驱动，即以RTT为更新间隔时间
- 二分查找：
 - 在ACK时钟的驱动下，将拥塞窗口置为 $(W_1 + W_2)/2$ (新的 W_2 值)，不断逼近满载窗口
- 最大探查：
 - 如窗口再次达到 W_1 而没有丢包，说明满载窗口大于 W_1 ，则以逼近 W_1 的镜像过程增大拥塞窗口

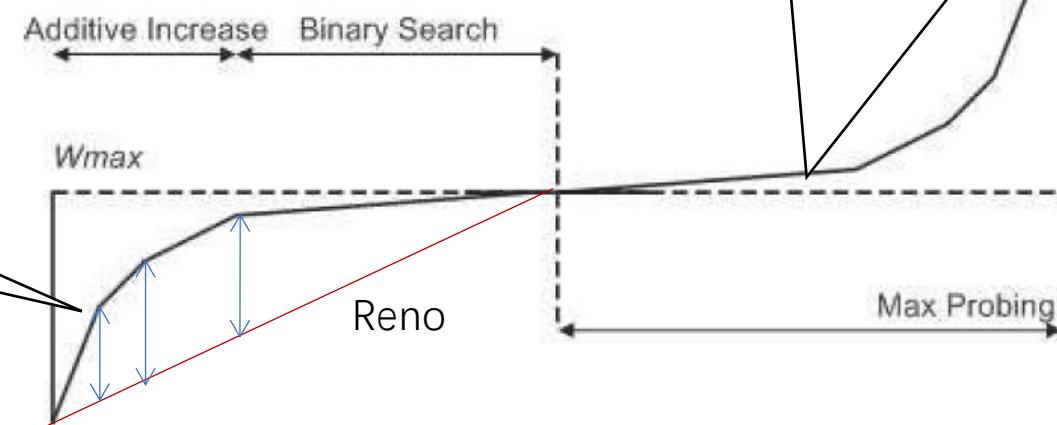


TCP-BIC 图解



中國人民大學
RENMIN UNIVERSITY OF CHINA

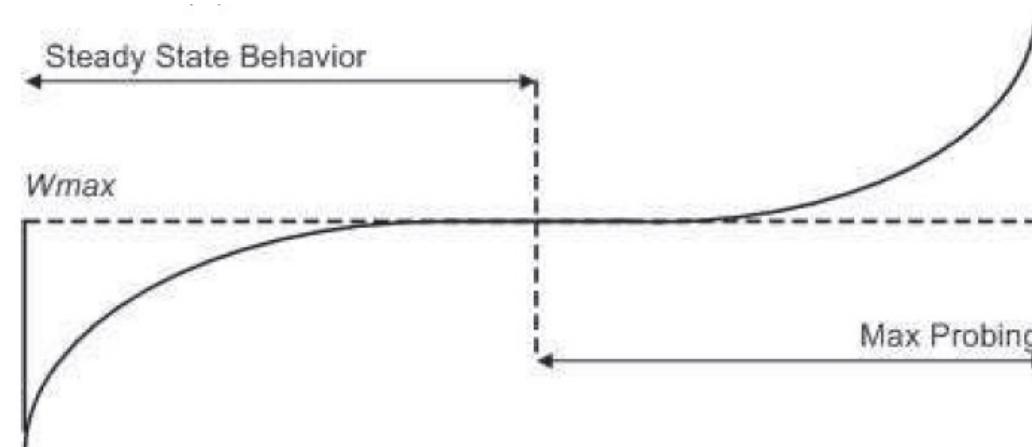
BIC将线性增大探查
 W_{max} 的过程转变为折半查找，窗口增长速度大大快于线性查找



当 W_{max} 发生更新时以先慢后快的方式探测 W_{max} ,
保证全过程拥塞窗口尽可能接近 W_{max}



- CUBIC将BIC算法连续化，用三次函数拟合BIC算法曲线
- 拥塞窗口成为距上次丢包的时间 t 的函数， t 取值位于两次丢包之间
- 三次函数增长分为两个阶段：
 - Steady State Behavior阶段：以凹函数增长逼近最近一次丢包时窗口；
 - Max probing阶段：以凸函数增长探测当前满载窗口
- 拥塞窗口在绝大多数时间内接近 W_{max} ，保持了较高的发送效率
- CUBIC已实现在Linux 2.6.18中





本章内容

6.1 概述和传输层服务

6.2 套接字编程

6.3 传输层复用和分用

6.4 无连接传输：UDP

6.5 面向连接的传输：TCP

6.6 理解网络拥塞

6.7 TCP拥塞控制

6.8 拥塞控制的发展

6.9 传输层协议的发展

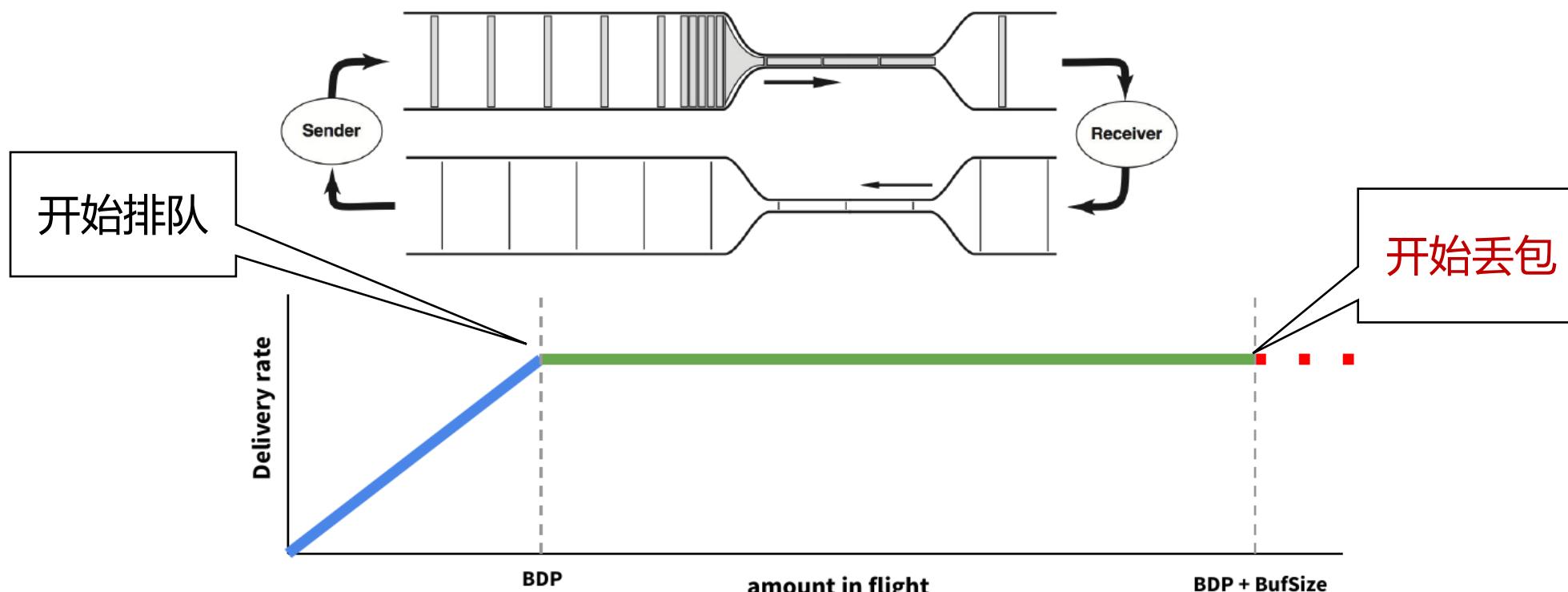


1. TCP CUBIC
2. Google BBR
3. Data Center TCP



➤ 拥塞与瓶颈链路带宽：

- 瓶颈链路带宽BtlBw，决定了端到端路径上的最大数据投递速率
- 拥塞窗口大于BtlBw时，瓶颈链路处会形成排队，导致RTT延长(直至超时)

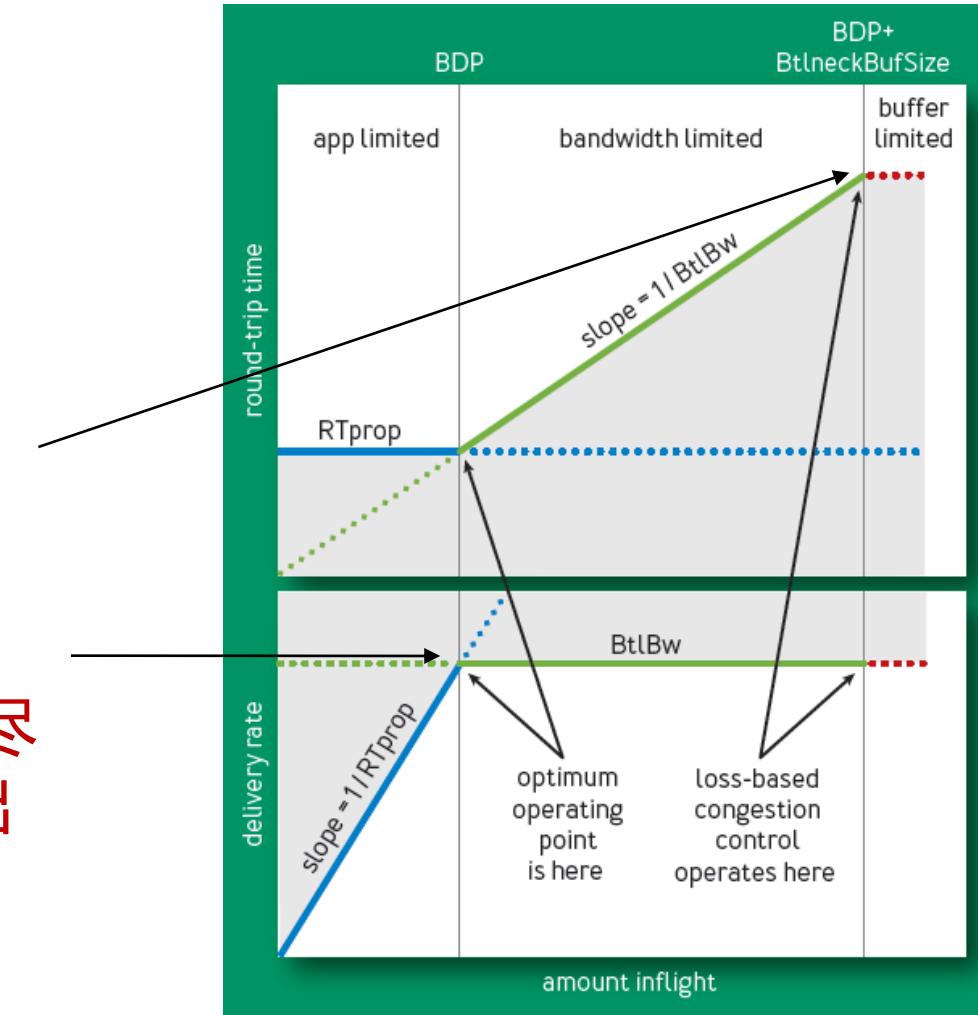




BBR: Bottleneck Bandwidth and Round-trip propagation time



- 瓶颈链路带宽BtlBw：不会引起路由器缓存排队的最大发送速率
- RTprop：往返时间
- $BDP = BtlBw * RTprop$
- 经典拥塞控制状态机以丢包事件为驱动，探测阶段将瓶颈链路上的缓冲区填满直至丢包，并以此为依据判断是否进行被动的乘性减小
- BBR试图测量图中左侧优化点的BtlBw，尽量将cwnd收敛到实际BtlBw，从而避免出现丢包，属于主动探测
- 以图中BDP竖线为分界点，右侧测得RTprop会因瓶颈链路发生排队而逐渐增长

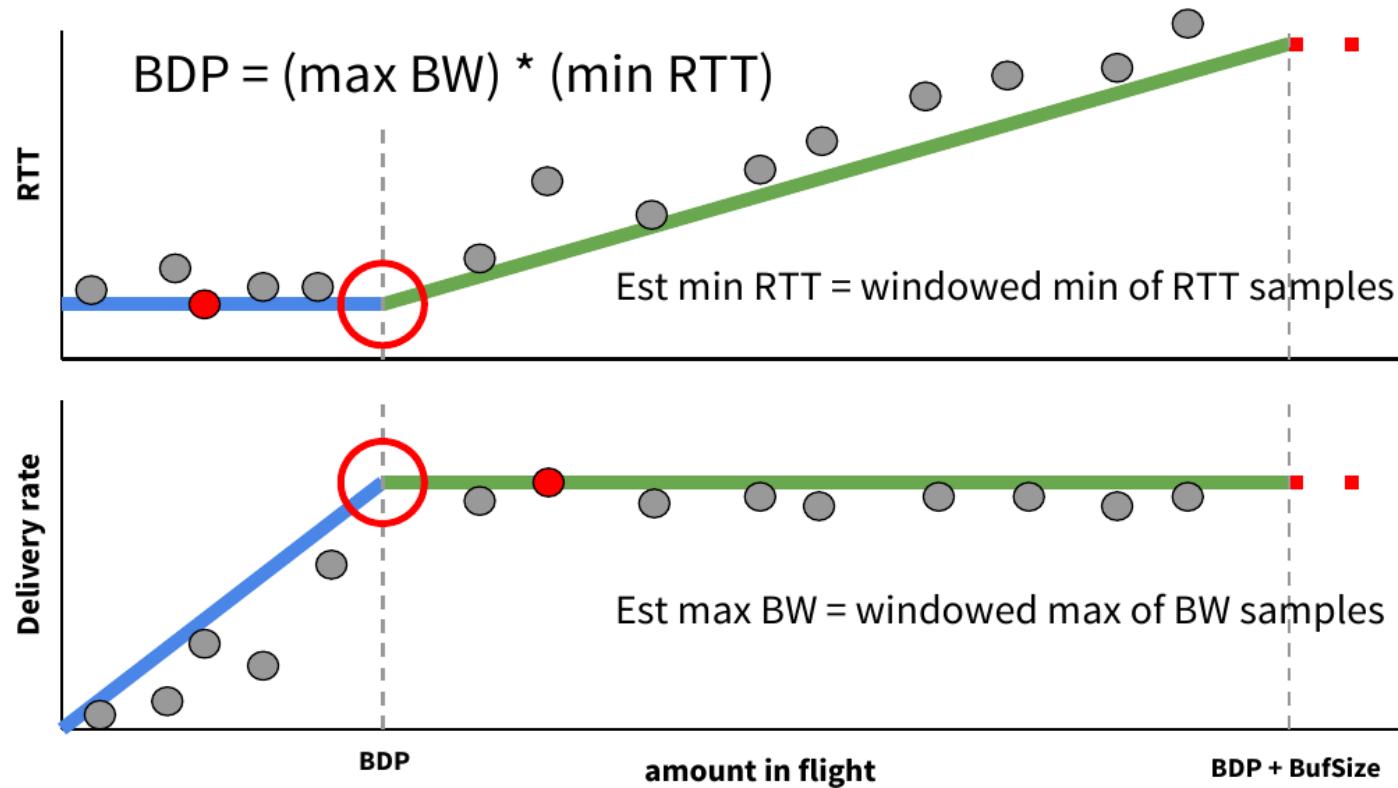




优化点的近似观测



用过去10秒内的最小RTT (min RTT) 和最大投递率 (max BW) , 分别近似RTprop和BtlBw , 并依据这两个值估算当前BDP



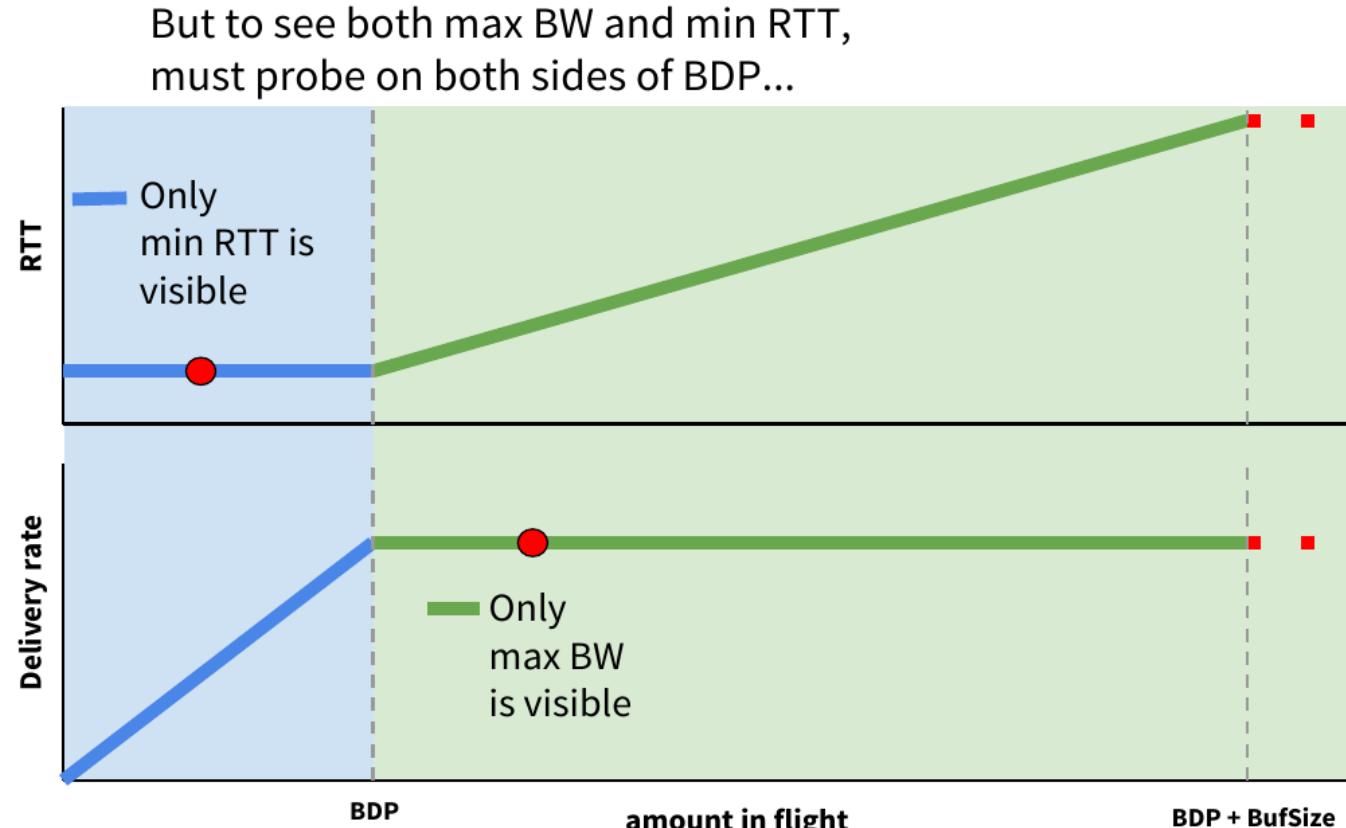


Max BW和min RTT不能同时被测得



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 要测量最大带宽，就要把瓶颈链路填满，此时buffer中存在排队分组，延迟较高
- 要测量最低延迟，就要保证链路队列为空，网络中分组越少越好，cwnd较小



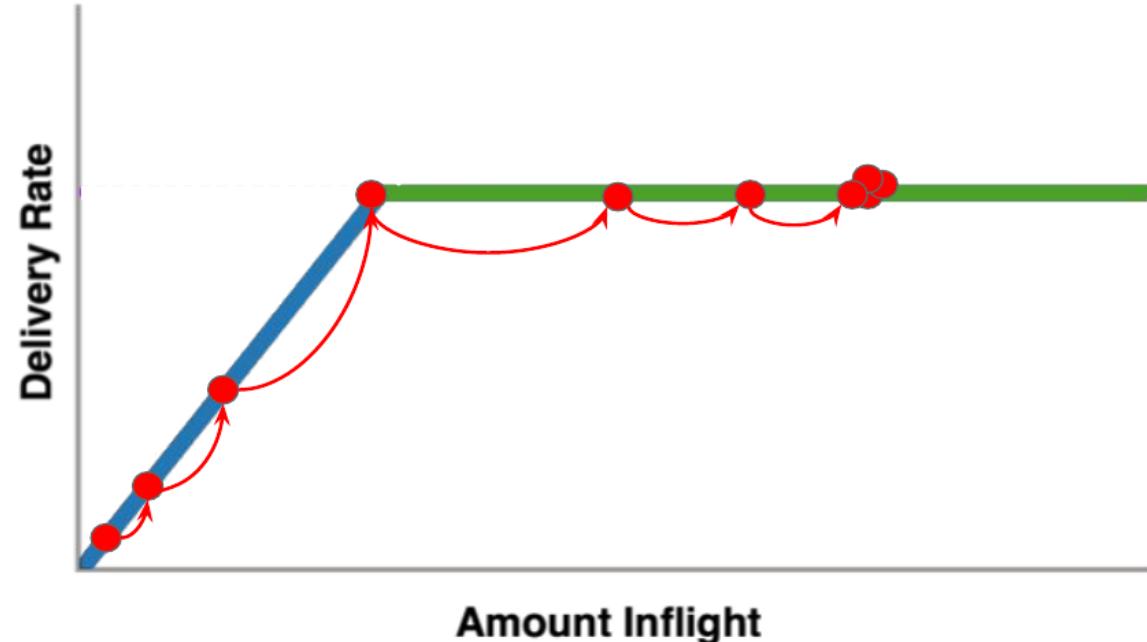


BDP检测：启动阶段 (START_UP)



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 当连接建立时，类似TCP的慢启动，指数增加发送速率，尽可能快地占满管道
- 若经过三次发现投递率不再增长，说明已达到 Bw/Bw ，瓶颈链路处分组已开始排队（事实上此时占的是三倍BDP）



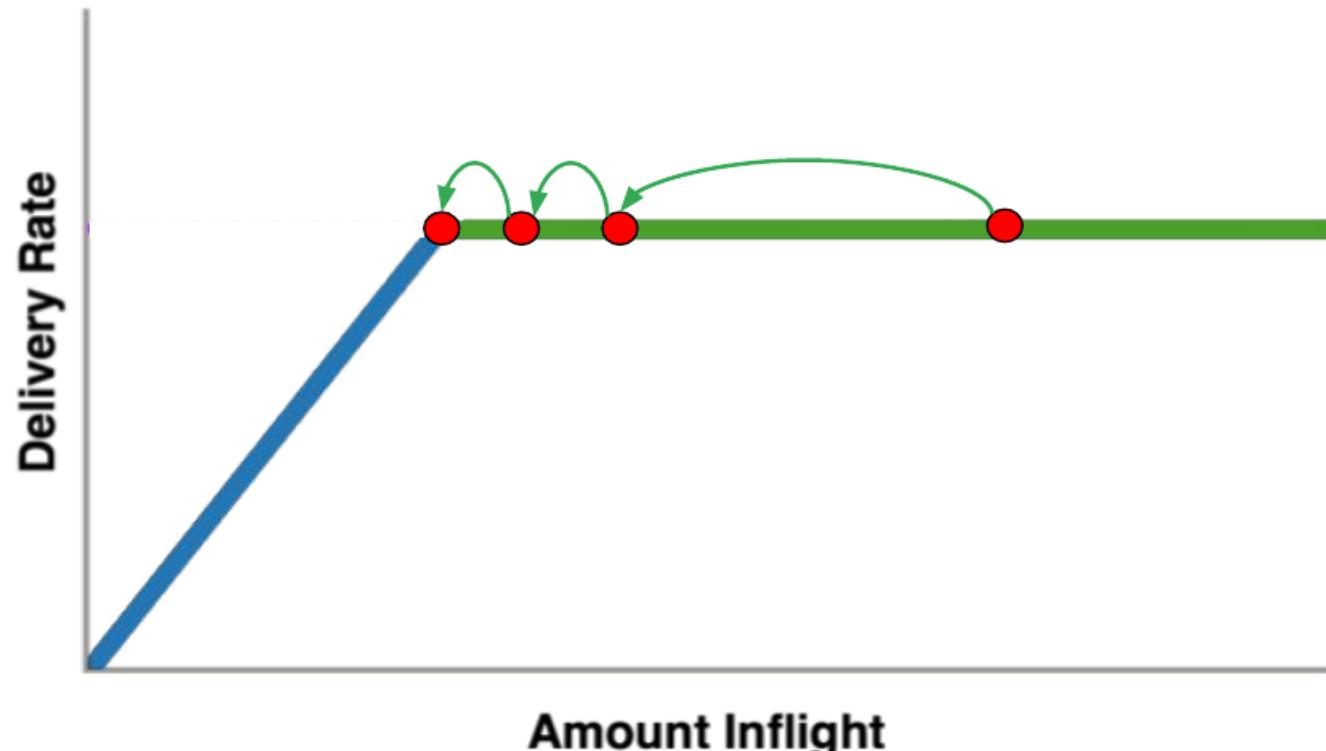


BDP检测：排空阶段（DRAIN）



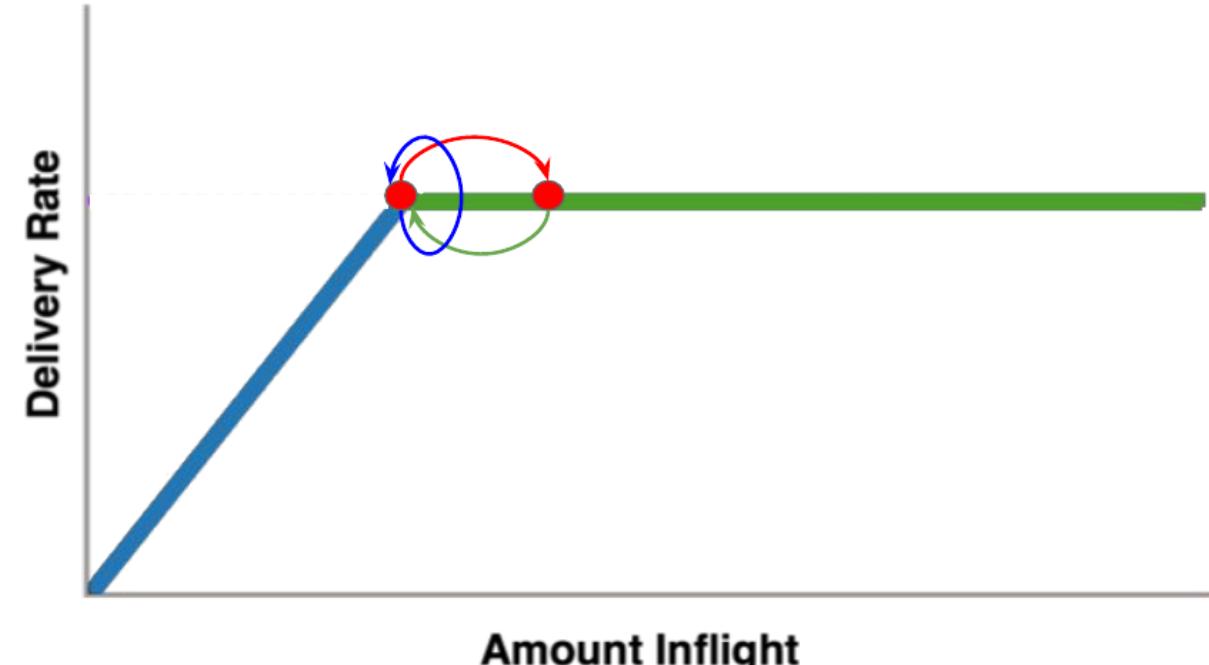
中國人民大學
RENMIN UNIVERSITY OF CHINA

指数降低发送速率（相当于是startup的逆过程），将多占的两倍
buffer慢慢排空



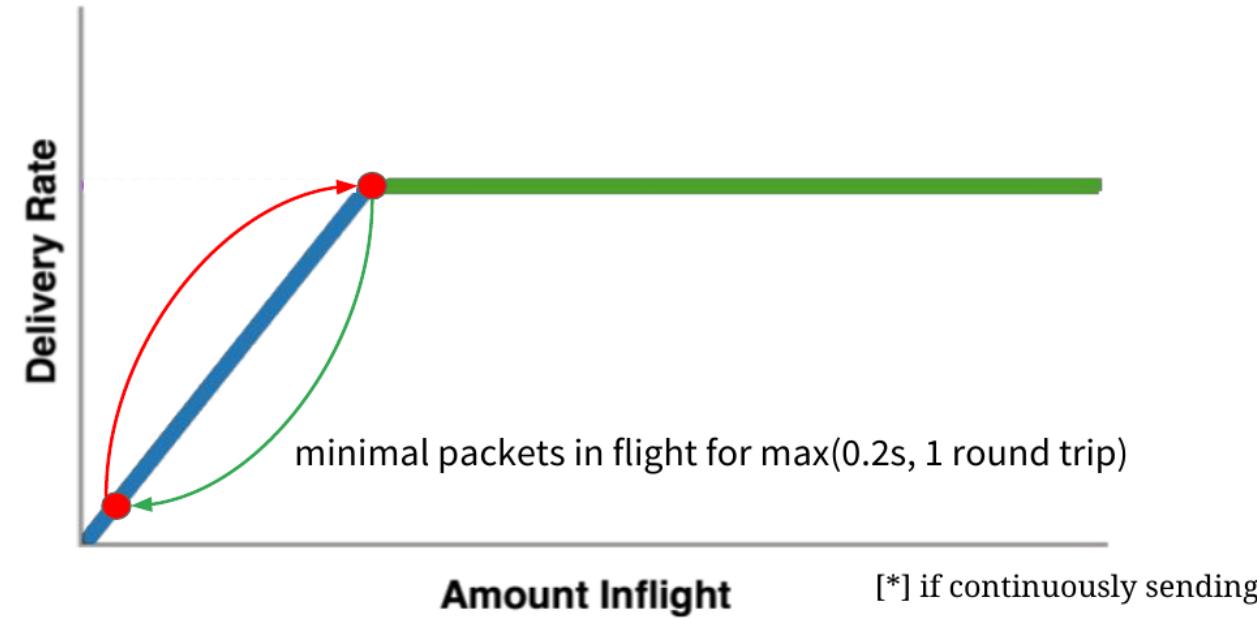


- 进入稳定状态后, 先在一个RTT内增加发送速率, 探测最大带宽
- 如果RTT没有变化, 再减小发送速率, 排空前一个RTT多发出来的包
- 后面6个周期使用更新后的估计带宽发送





- 每过10秒，如果估计的RTprop不变，就进入RTprop探测阶段
- 在这段占全过程2%的时间内，cwnd固定为4个包
- 测得的RTprop作为基准，用以判断带宽检测阶段瓶颈链路中是否发生排队
- 为抵消此阶段牺牲的发送速率，结合pacing_gain数组在后期短时间增加发送速率，保证对BtlBw的及时观测
- pacing_gain数组用来微调每个状态下不同时段的拥塞窗口大小

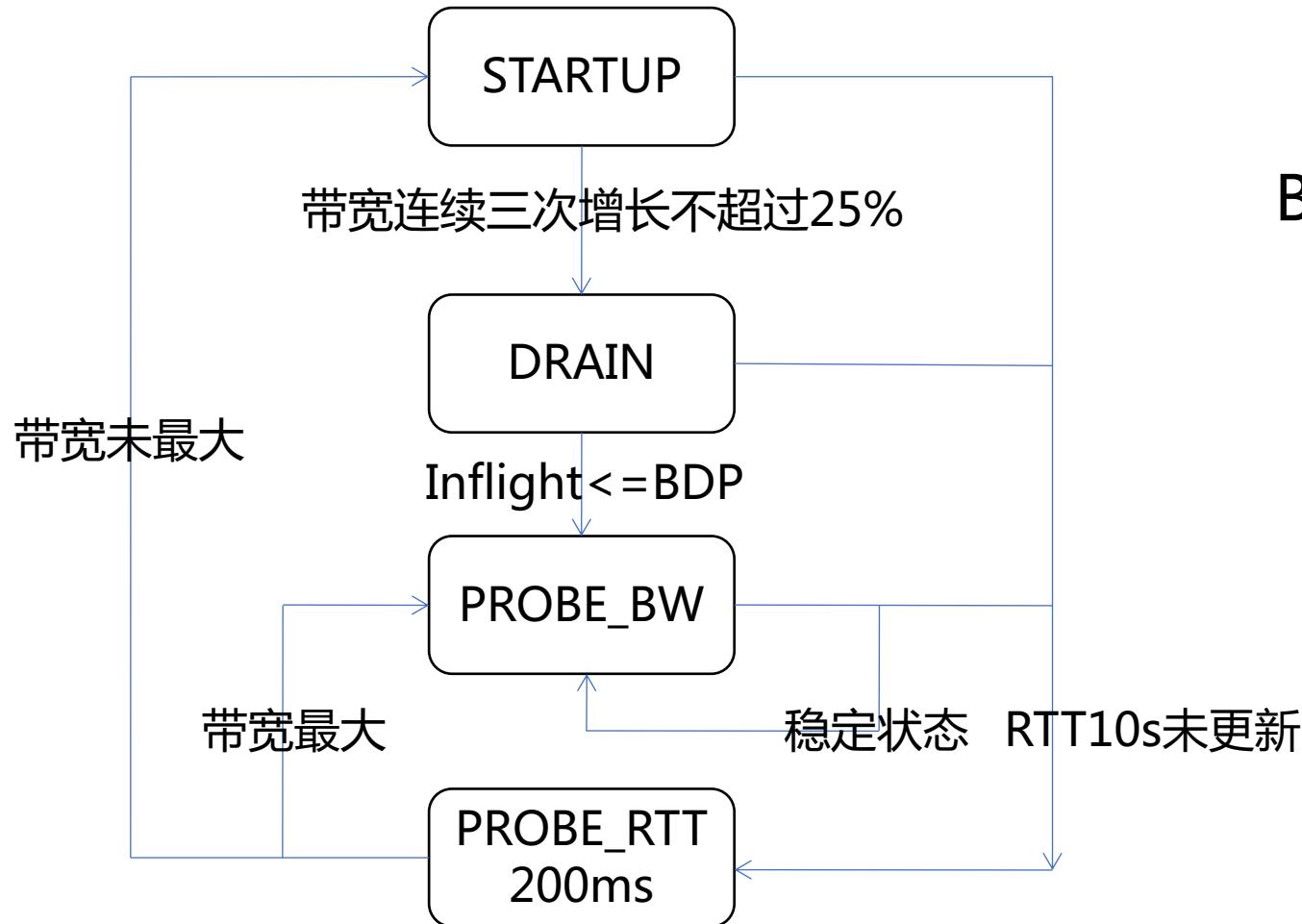




BBR状态机



中國人民大學
RENMIN UNIVERSITY OF CHINA



BBR已实现在Linux 4.9中



本章内容

6.1 概述和传输层服务

6.2 套接字编程

6.3 传输层复用和分用

6.4 无连接传输：UDP

6.5 面向连接的传输：TCP

6.6 理解网络拥塞

6.7 TCP拥塞控制

6.8 拥塞控制的发展

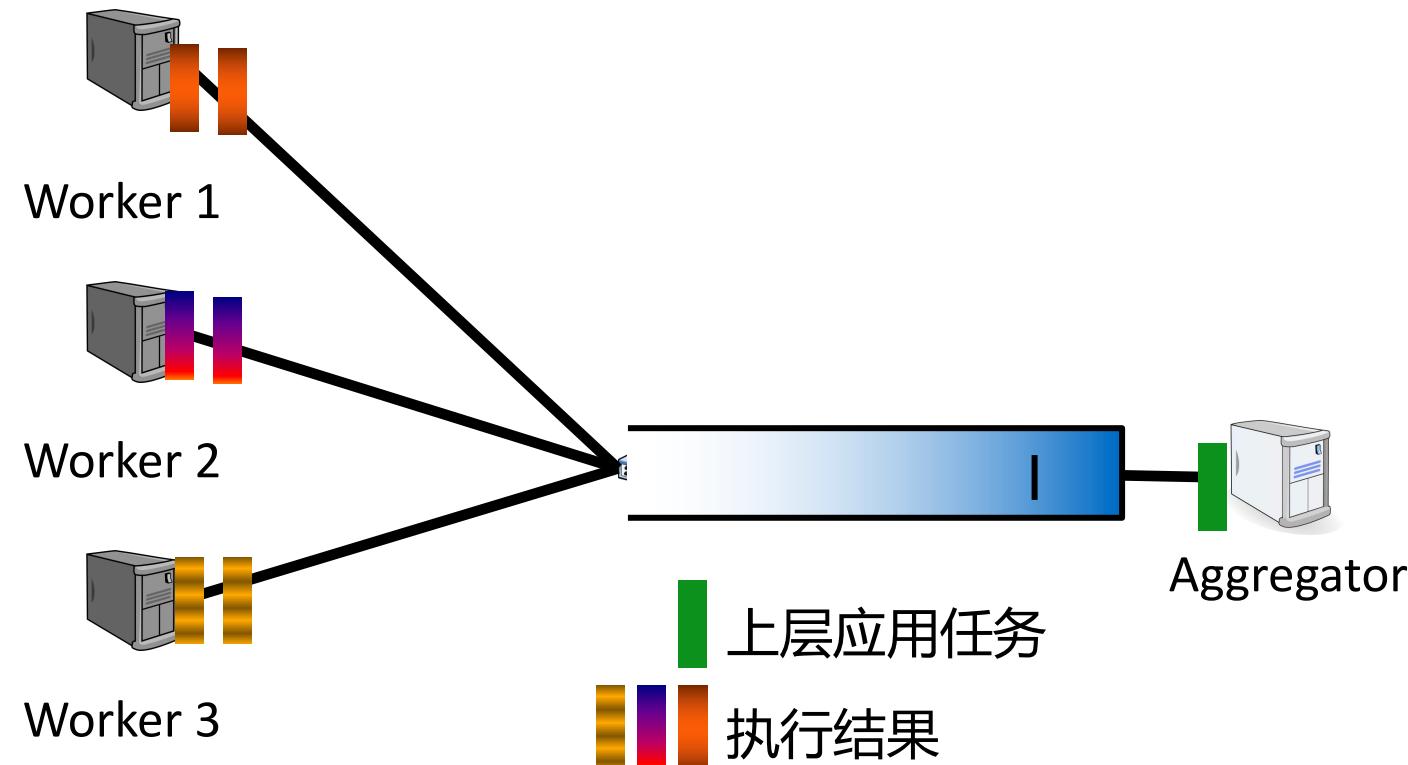
6.9 传输层协议的发展



1. TCP CUBIC
2. Google BBR
3. Data Center TCP



- 数据中心内部有一类经典的通信模式-Partition/Aggregate
 - Aggregator将上层应用任务划分给下层的worker执行





- 90%+的流量基于TCP连接，根据流量大小可以分为：
 - 时延敏感的短流（50KB-1MB）
 - 吞吐敏感的长流（1MB-50MB）
- 此外，还存在大量突发流量
 - 时间<1s
 - 大小<2KB

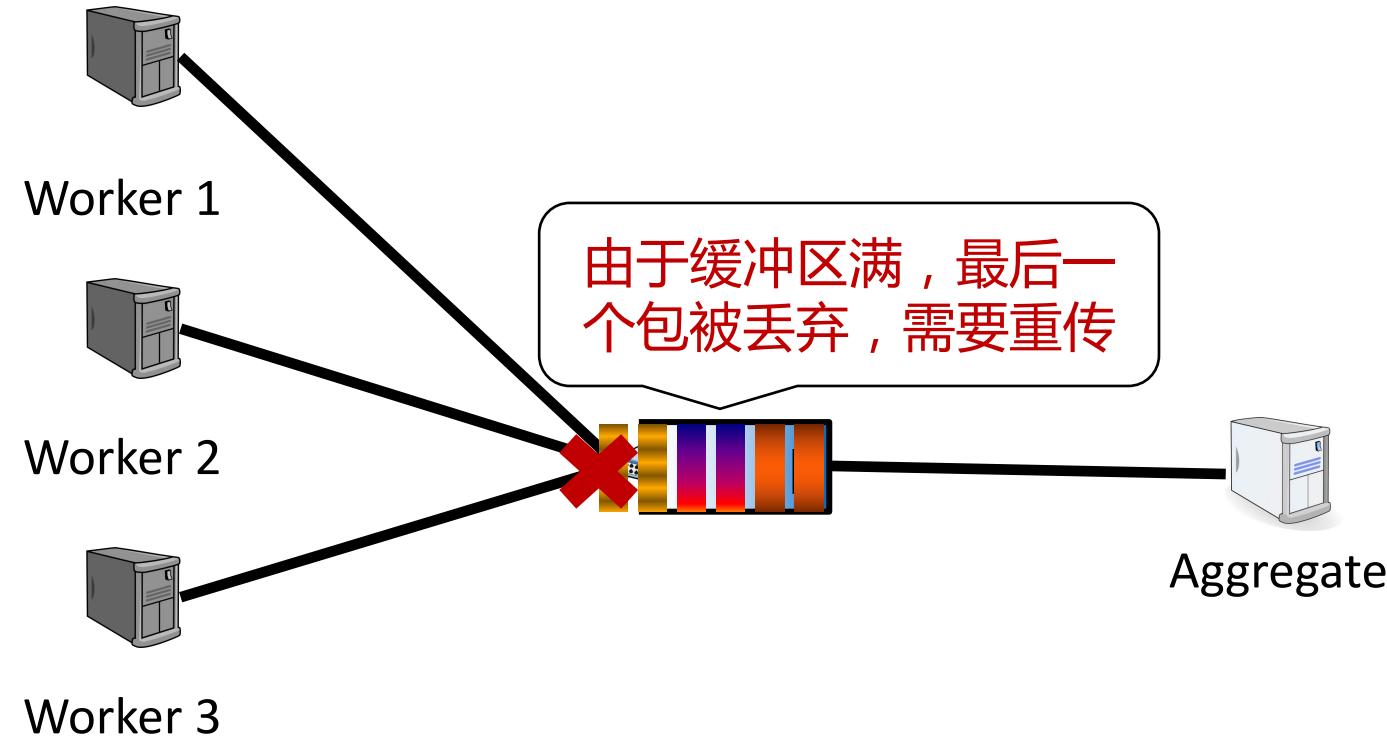
以上数据来源于：

[1] Alizadeh M, Greenberg A, Maltz D A, et al. Data center tcp (dctcp)[C]//Proceedings of the ACM SIGCOMM 2010 Conference. 2010: 63-74.



1. Incast

- 在一个很短的时间内，**大量流量同时到达交换机的一个端口**，导致缓冲区被占满，最终导致丢包
- 在并发流量很大的情况下，即使每条流的包很小，也会产生Incast问题



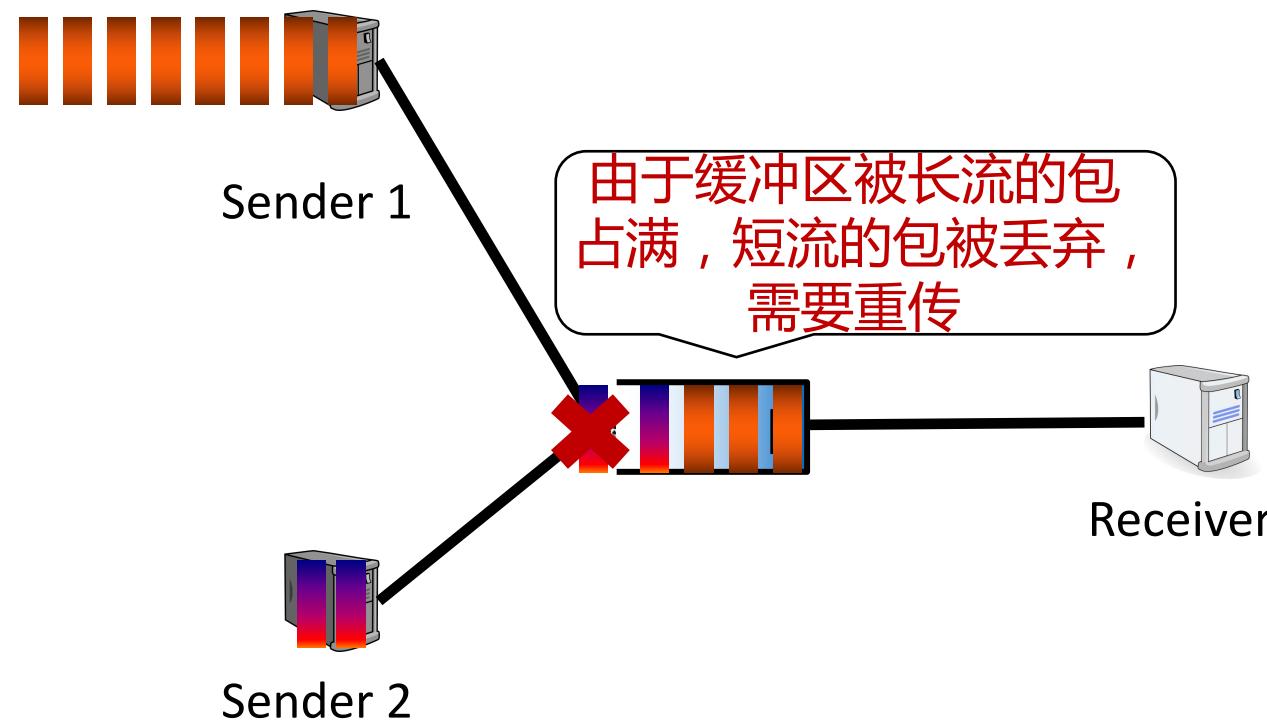


2. Queue Buildup

- 长流和短流同时通过交换机的同一个端口时，由于长流占用较多的缓冲区空间，导致短流延迟增大，甚至丢包

3. Buffer Pressure

- 交换机的不同端口通常共享同一块缓冲区，即使长流和短流通过不同的端口，短流通过的端口也会出现缓冲区不足的问题





➤ 容忍高突发流量

- 在Partition/Aggregate通信模式中，所有Worker几乎会在同一时间向Aggregator返回执行结果，产生很高的突发流量

➤ 低时延

- 数据中心有大量时延敏感的短流，如网页搜索等

➤ 高吞吐

- 数据中心有大量吞吐敏感的长流，如文件传输、分布式机器学习中神经网络模型参数的传输等



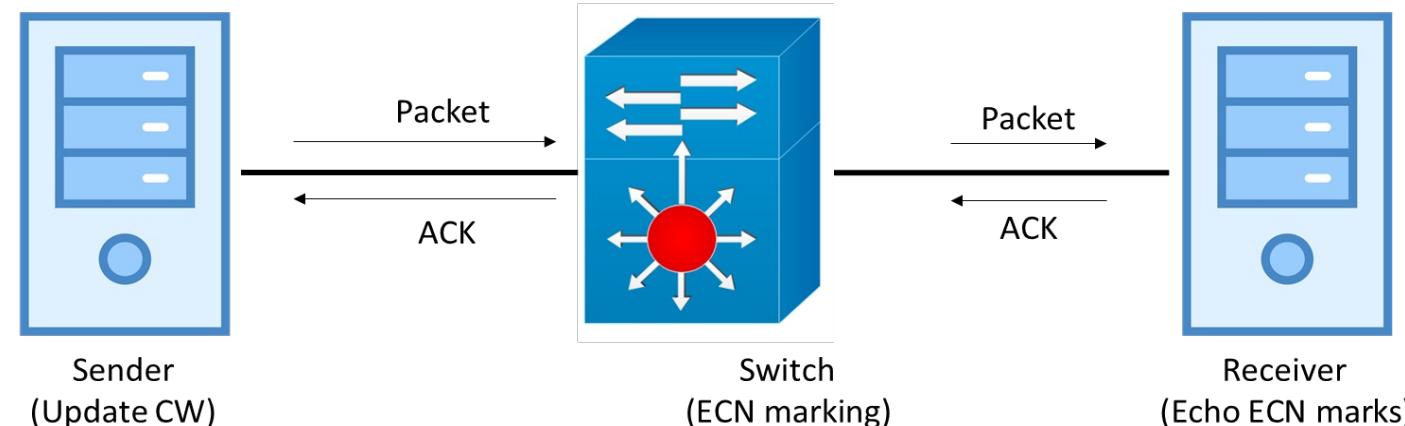
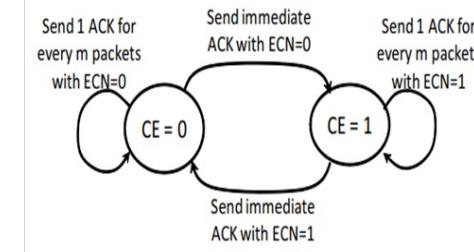
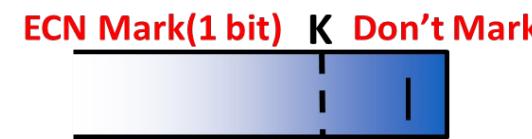
DCTCP核心思想



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 根据网络拥塞程度精细地减小发送窗口：
 - 一旦发现拥塞，发送窗口减至原窗口的 $(1 - \alpha/2)$ ， α 反映了拥塞程度，而TCP中 α 总为1
- 根据交换机队列的瞬时长度标记ECN (explicit congestion notification)
 - 使用显式的拥塞反馈能够更好地控制突发流量

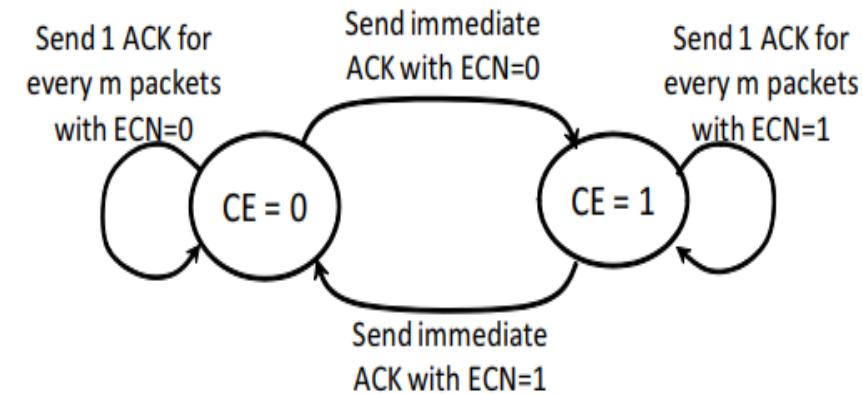
$$cwnd = \begin{cases} \left(1 - \frac{\alpha}{2}\right) \times cwnd, & \text{when congestion happens} \\ cwnd + 1, & \text{when there is no congestion} \end{cases}$$





- 交换机
 - 当队列长度超过K时，给随后到来的包标记ECN
- 接收端
 - 仅当被标记的报文出现或消失时才立即发送ACK，否则采取Delay ACK的策略
- 发送端
 - 每个RTT更新一次发送窗口
 - $\alpha \leftarrow (1 - g)\alpha + \alpha F$ ，这个值反应了拥塞程度
 - 其中 $F = \frac{\# \text{ of marked ACKs}}{\text{Total } \# \text{ of ACKs}}$
 - $Cwnd \leftarrow \left(1 - \frac{\alpha}{2}\right) * Cwnd$

ECN Mark(1 bit) K Don't Mark



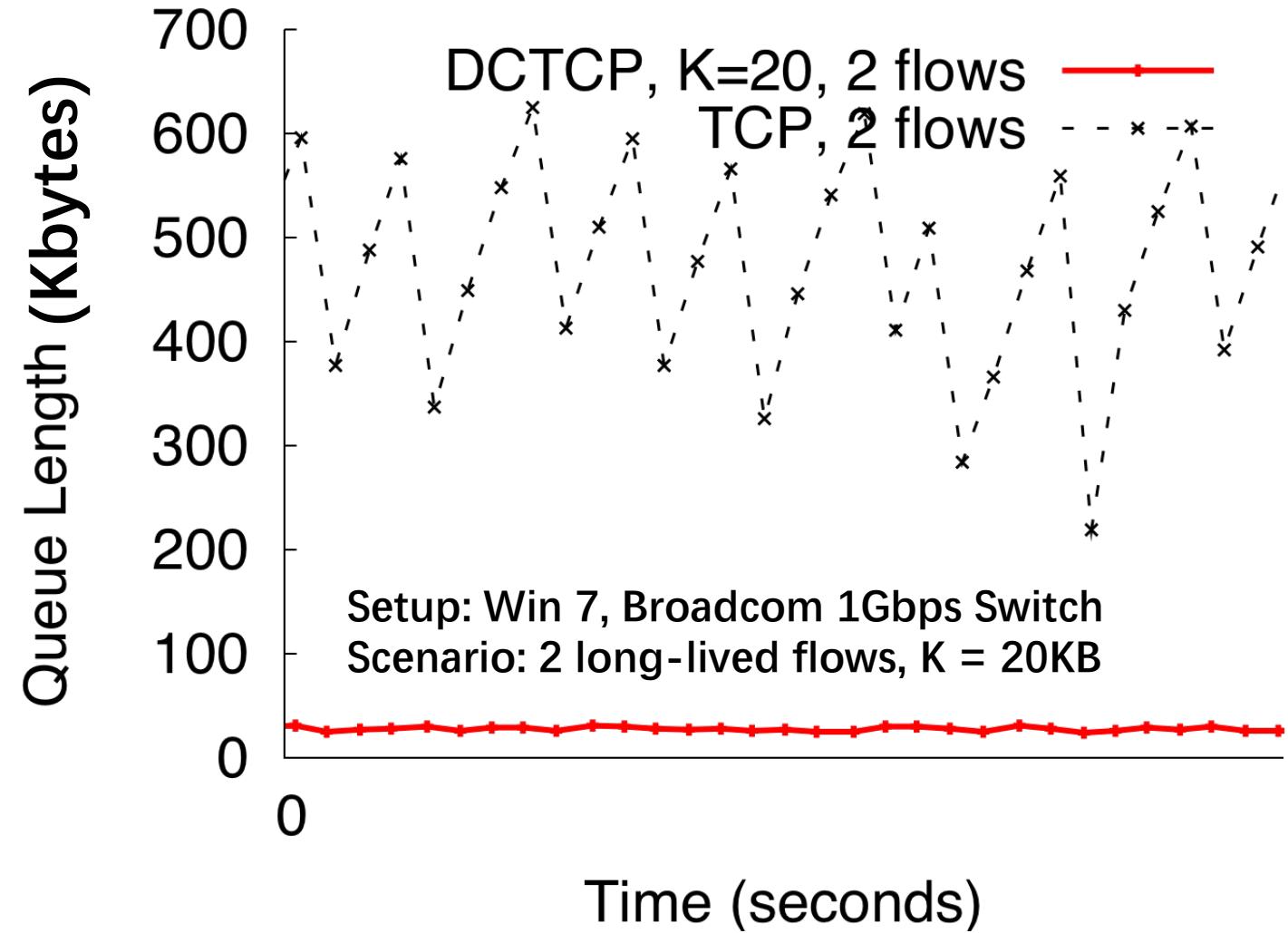
➤ DCTCP的实现：

- 只需30行左右的代码改动
- 在交换机上设置参数K



DCTCP和TCP的比较

- 交换机中的队列长度：
 - DCTCP能将队列长度稳定地维持在一个很低的水平
 - 而TCP的队列长度不仅高，而且波动很大
- 因此，DCTCP更适用于现代数据中心





为什么DCTCP能够奏效



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 容忍高突发流量
 - DCTCP维持了较低的队列长度，可以留出较大的缓冲区给突发流量
 - 采用激进的标记策略，使得发送端在丢包之前就感知到拥塞
- 低时延
 - 由于队列长度较短，也减少了包在队列中的排队时延
- 高吞吐
 - DCTCP根据拥塞程度精确调节窗口，使得发送窗口的变化比较平滑，不会出现吞吐量骤降的情形



本章内容



- 6.1 概述和传输层服务
- 6.2 套接字编程
- 6.3 传输层复用和分用
- 6.4 无连接传输：UDP
- 6.5 面向连接的传输：TCP
- 6.6 理解网络拥塞
- 6.7 TCP拥塞控制
- 6.8 拥塞控制的发展
- 6.9 传输层协议的发展

1. MPTCP
2. QUIC

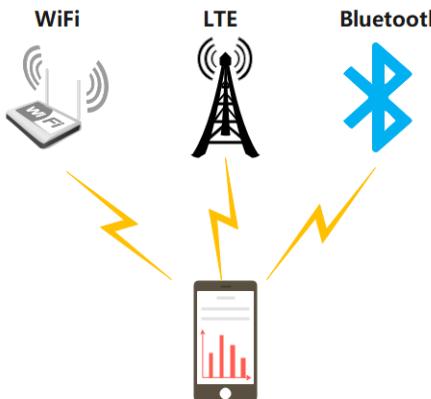


传统TCP协议的不足与改进方案

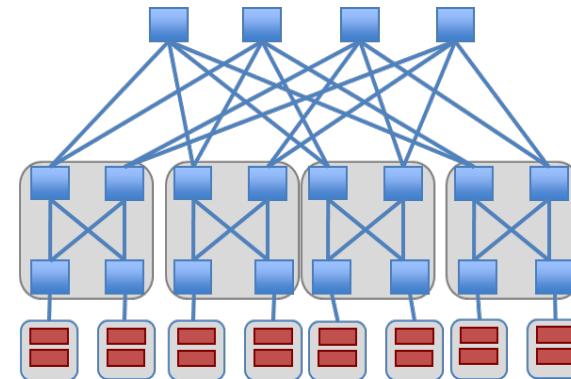


中國人民大學
RENMIN UNIVERSITY OF CHINA

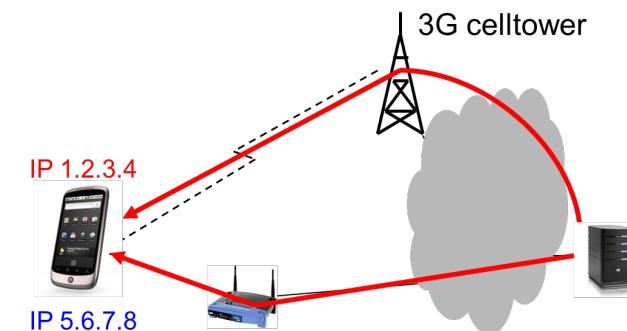
- 传统TCP协议仅支持单路径传输，即只能利用终端主机上的一个网络接口传输数据
- 随着接入技术的发展，同时具备多个网络接口的网络设备已经越来越普及，多路径传输更适合当前的网络环境



移动网络



数据中心移动网络



多路径传输示意图

- 移动网络和数据中心网络中的网络设备天然拥有多个网络接口，已具备实现多路径传输的物理基础
- 多路径TCP协议（MPTCP）应运而生，它可将单一数据流切分为若干子流，同时利用多条路径进行传输



MPTCP的优势



中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ 多径带宽聚合

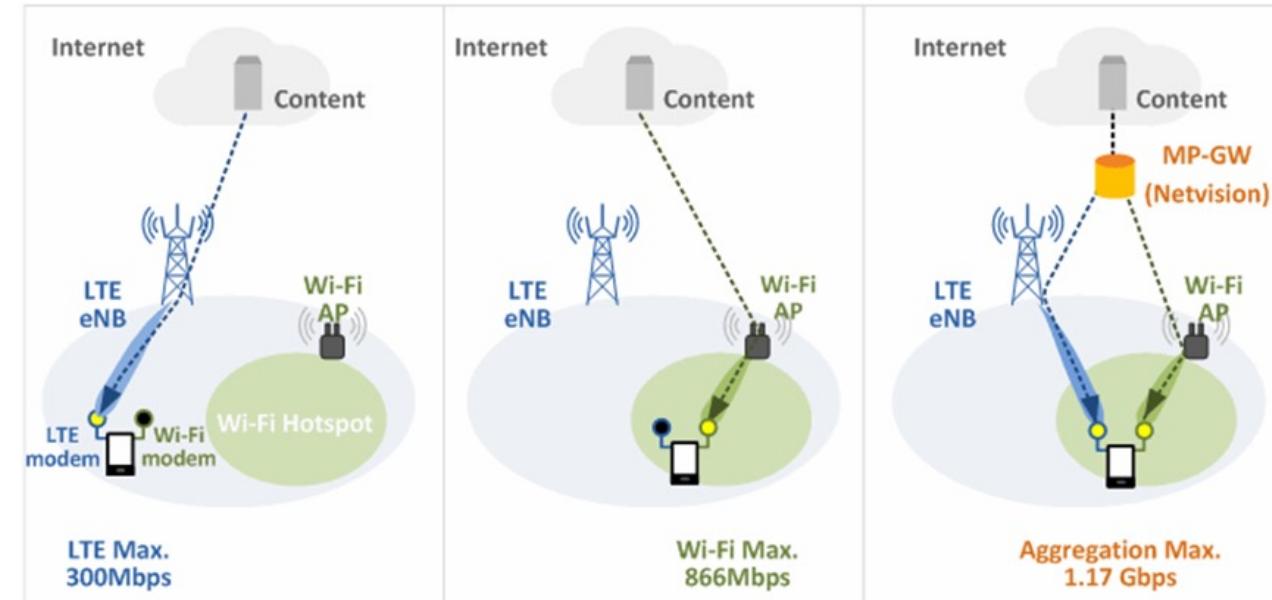
- 终端设备可以聚合不同路径上的可用带宽，以获得更高的网络带宽

➤ 提升传输的可靠性

- 使用多条路径传输数据，可以有效避免因单条路径性能恶化或中断导致的应用连接中断

➤ 支持链路的平滑切换

- 多路径传输方式允许终端在不同接入网络间快速、平滑地切换，选取当前链路质量最好的路径传输数据





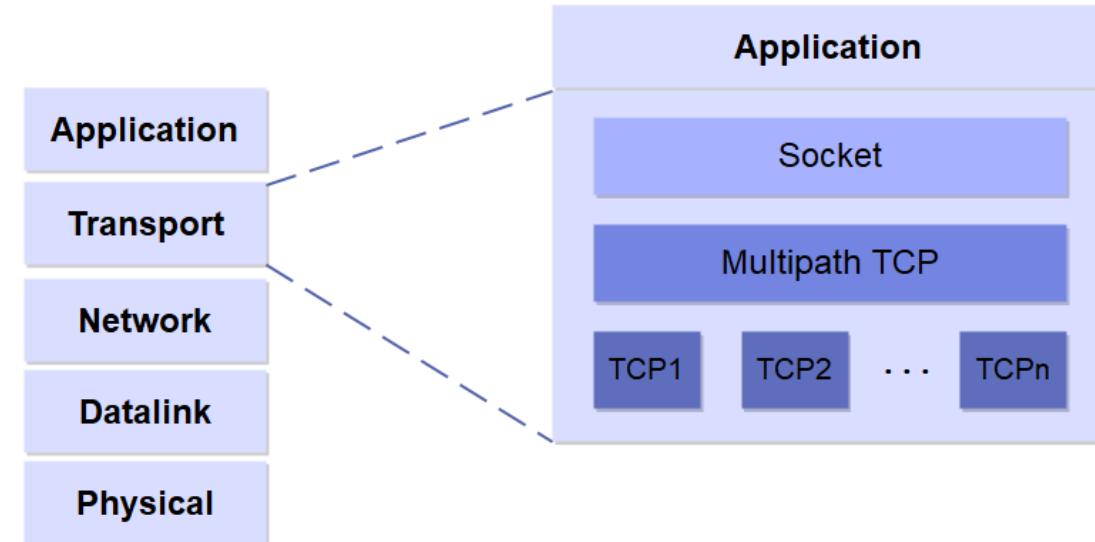
MPTCP在网络体系结构中的位置



中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ MPTCP位于套接字和TCP之间：

- 应用程序通过套接字调用MPTCP，MPTCP向应用程序提供单条连接的抽象，因而对应用层透明
- MPTCP可在源主机和目的主机的多对网络接口间分别建立TCP连接，并将数据流分配到多条TCP连接上传输
- MPTCP兼容并扩展了TCP协议：TCP基本头不变，只定义了新的选项，从而对网络层也是透明的



➤ MPTCP连接是一个或多个子流的集合：

- 路径：本地主机与远程主机之间可用于建立连接的一个链路序列称为路径，使用四元组<本地IP地址，本地端口，远程IP地址，远程端口>表示
- 子流：在单个路径上运行的TCP流称为子流，是MPTCP连接的组成部分

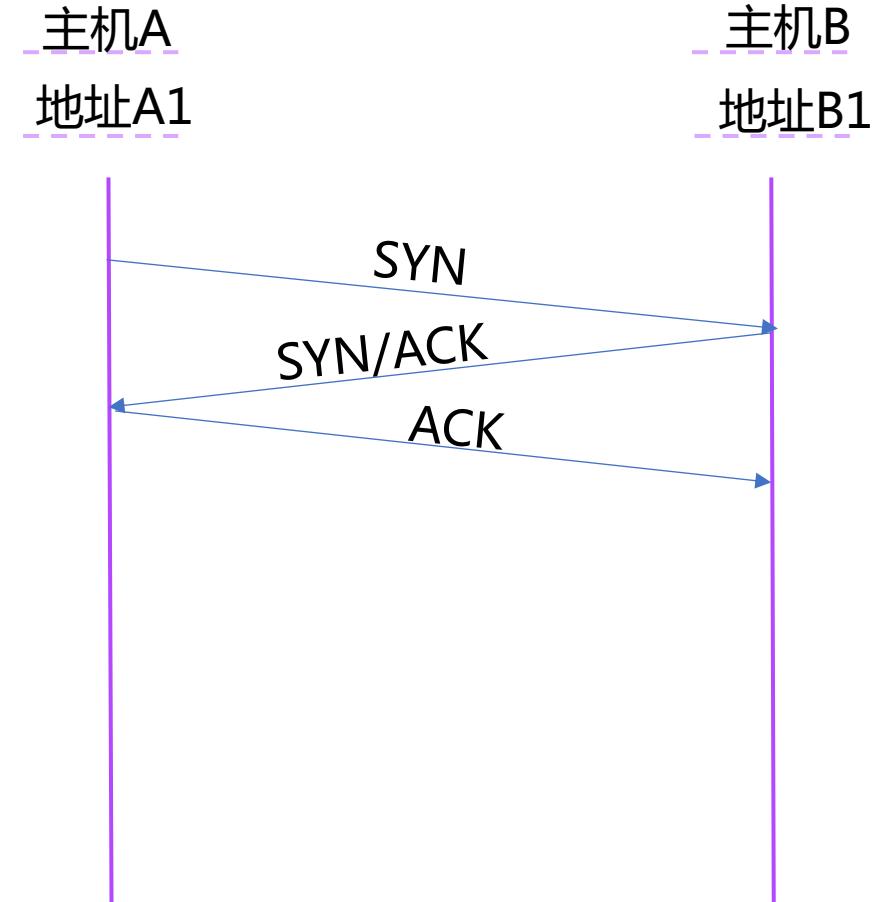


MPTCP连接管理



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 用于MPTCP连接管理的新字段：
 - MP_CAPABLE：建立MPTCP连接
 - MP_JOIN：附加新的子流到已有连接
 - ADD_ADDR：新增可用路径
 - REMOVE_ADDR：删除路径
 - MP_FASTCLOSE：关闭所有子流
- 如何建立MPTCP连接：
 - 首先初始化一条MPTCP连接（与建立常规TCP连接的过程相似）
 - 然后将MPTCP的其它子流附加到已经存在的MPTCP连接上



常规TCP连接的建立过程-三次握手

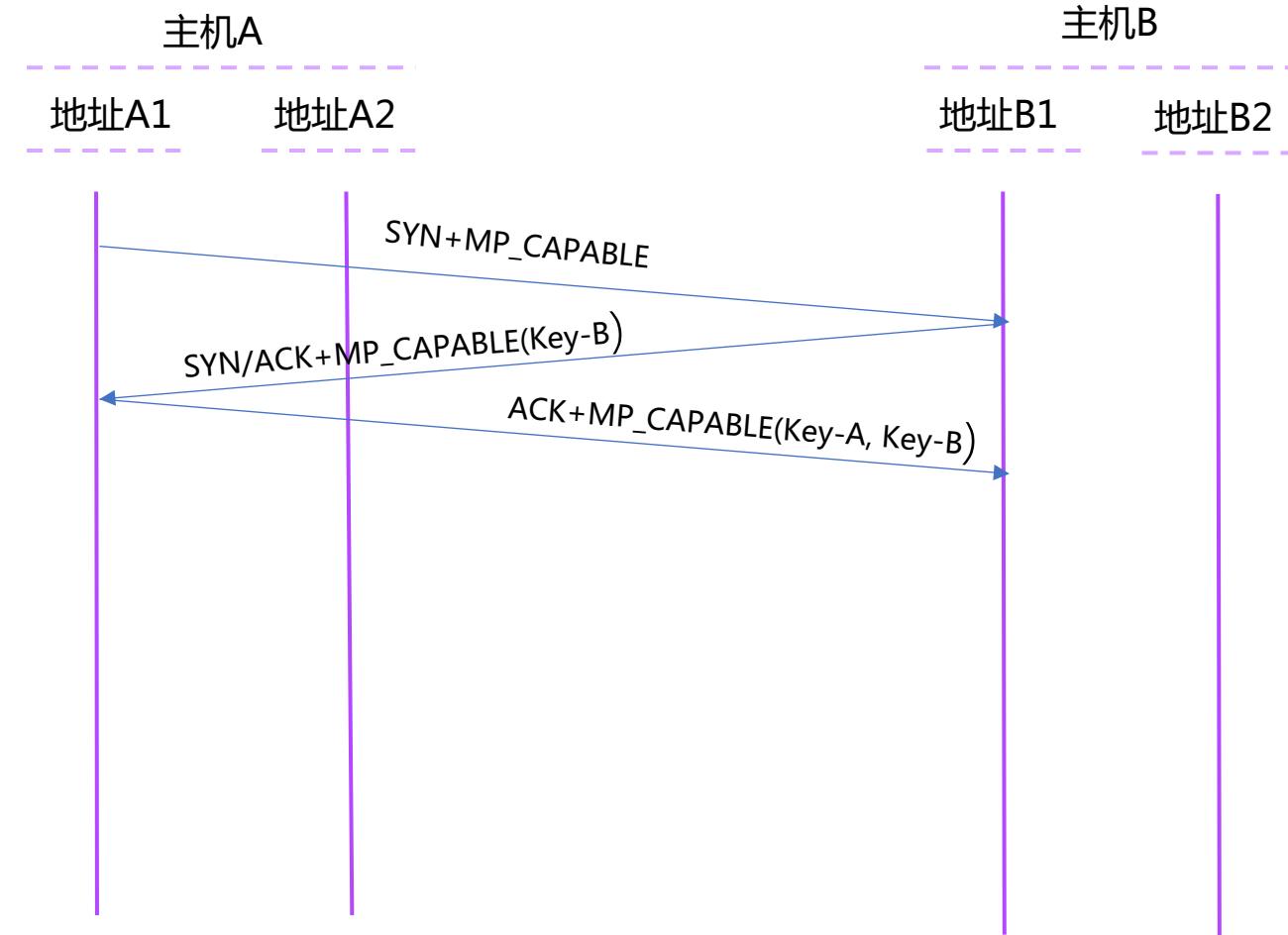


初始化MPTCP连接



➤ 初始化一条MPTCP连接 (A1->B1)

- 建立MPTCP连接的过程与建立常规TCP连接相似，不同之处在于启用了MP_CAPABLE字段，并交换了用于身份认证和建立子流的密钥信息
- 主机A发出SYN报文段，其中启用了MP_CAPABLE字段，用以询问主机B是否支持MPTCP
- 若主机B支持MPTCP，则响应SYN/ACK报文段，其中启用了MP_CAPABLE字段并附上自己的密钥
- 主机A向主机B发送ACK报文段，并附上自己和主机B的密钥，至此MPTCP的连接初始化完成



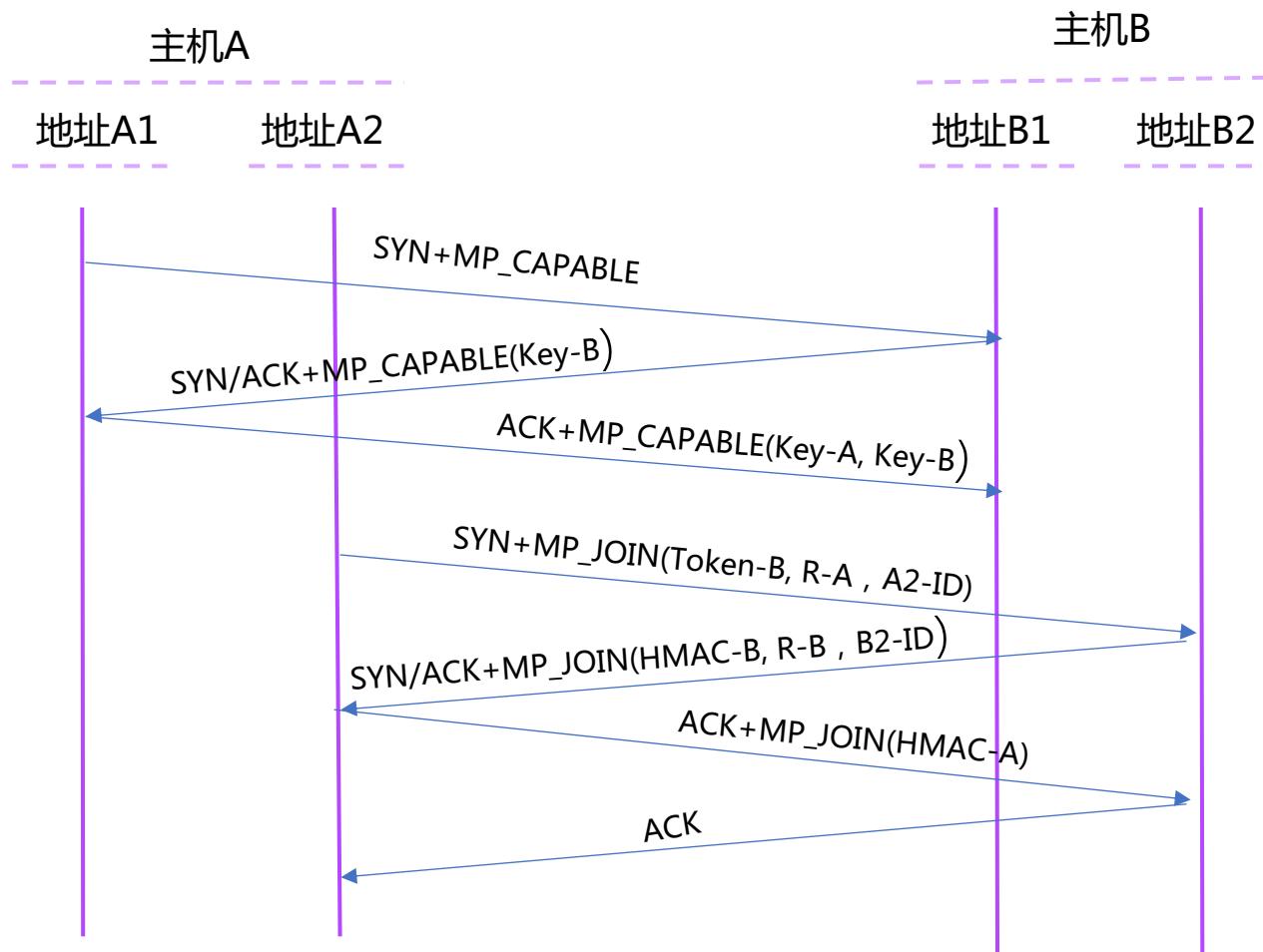


附加子流到MPTCP连接上



- 将新的子流 (A2->B2) 附加到已经存在的MPTCP连接

- 附加子流需要启用MP_JOIN字段
- 主机A根据Key-B生成主机B的令牌 (Token-B) , 该令牌指定当前子流需要附加到哪个MPTCP连接上，并将自身产生的随机数 (R-A) 和当前子流的地址编号 (A2-ID) 添加到SYN报文段中
- 主机B根据前面MP_CAPABLE字段交换的密钥和随机数 (R-A) 产生认证信息 (HMAC-B) , 与自身产生的随机数 (R-B) 和当前子流的地址编号 (B2-ID)一同写入SYN/ACK报文段中
- 主机A验证HMAC-B , 无误后向主机B发送ACK报文段，并附上自己的认证信息 (HMAC-A)
- 主机B验证HMAC-A , 无误后向主机A发送ACK报文段，至此子流建立完毕





➤ 新增路径

- 隐式新增：使用MP_JOIN方式新增子流
- 显式新增：仅新增路径（并不启用新的子流），使用ADD_ADDR字段
 - 主机A发送启用了ADD_ADDR字段的报文段，包含新的IP地址/端口对 (IP#-A3)和对应的IP地址编号(IP#A3-ID)、认证信息 (HMAC-A3)，ECHO指示当前报文段是“发出”
 - 主机B验证HMAC-A3，无误后将路径信息记录下来，向主机A发送响应报文段，ECHO标注为“响应”，其它信息原样发回

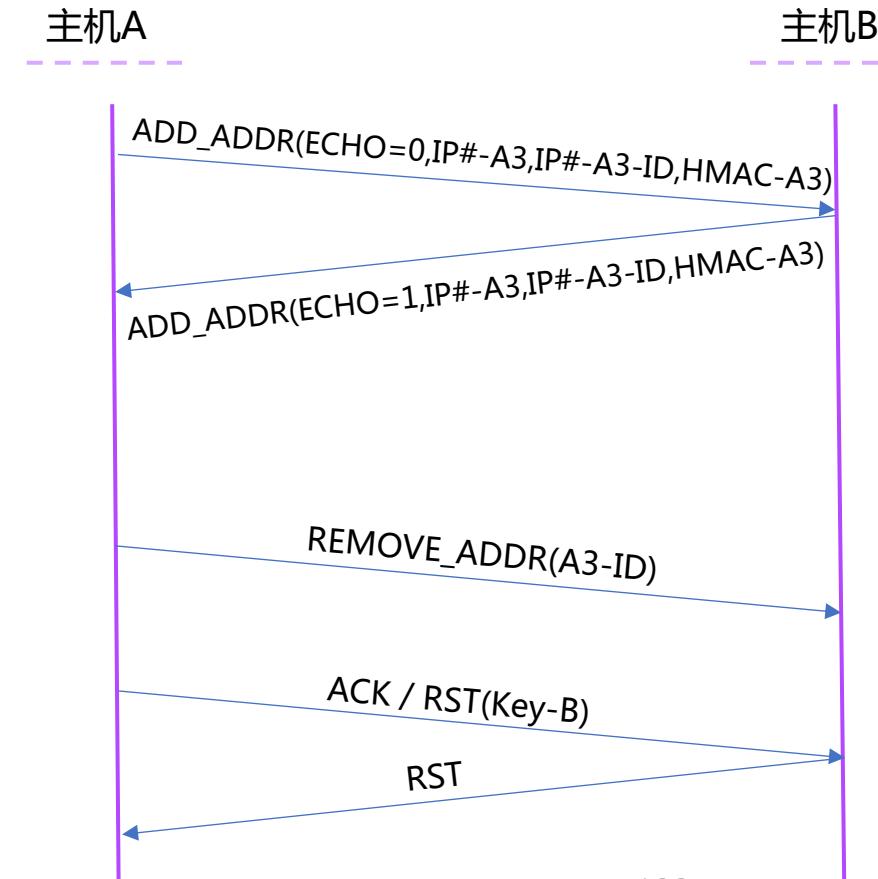
➤ 删 除 路 径

- 主机A发送启用了REMOVE_ADDR字段的报文段，指定要删除的地址编号

➤ 关闭单个子流：与关闭常规TCP连接一致

➤ 关闭所有子流：

- 主机A发送启用了MP_FASTCLOSE字段的报文段，其中附加了主机B的密钥用以身份认证
- 主机B收到后，选择恰当时机关闭所有连接，并向主机A发出确认

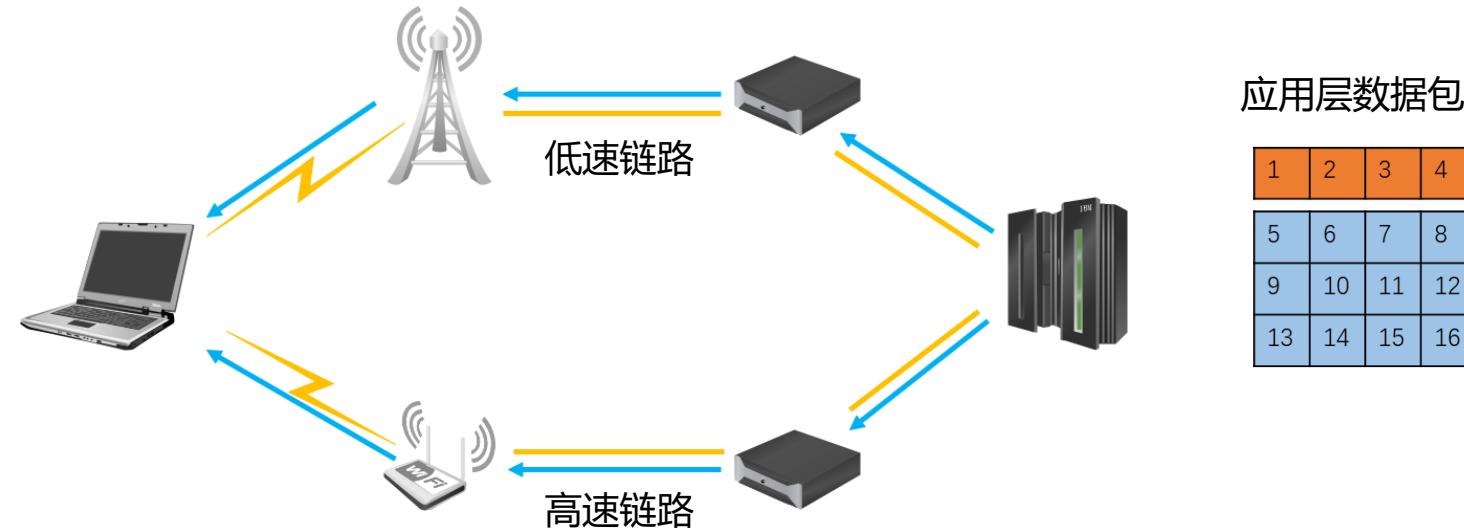




MPTCP的数据调度



- 在多路径传输中，发送端将属于同一个数据流的数据包调度到不同的路径上传输，由于不同路径的差异，这些数据包往往无法按照发送顺序到达接收端



- 乱序到达的数据包需暂存在接收缓存中，直到接收缓中的数据包能够按序交付给上层应用，这既影响了数据传输的实时性，又影响了网络的吞吐量
- MPTCP根据拥塞窗口大小及路径延迟，将数据按比例分配给各个子流，尽力保证数据包按序到达接收端，降低数据乱序到达对网络性能产生的不利影响



MPTCP的拥塞控制



中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ IETF MPTCP工作组提出的拥塞控制的设计目标

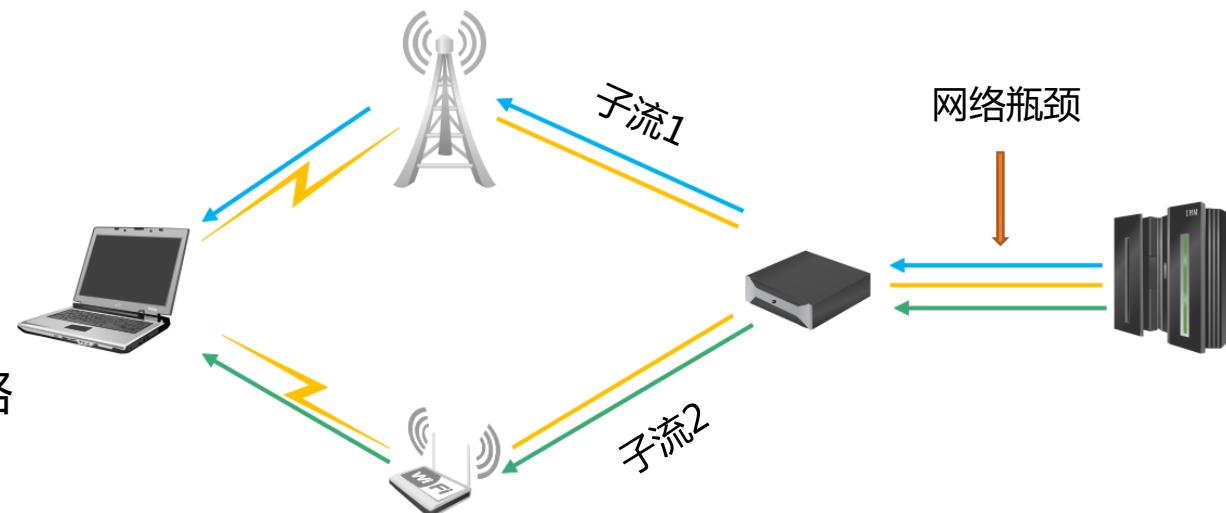
- 提升吞吐量：多路径TCP连接的各个子流获得的总吞吐量，不应低于其最优路径上单路径TCP连接的吞吐量
- 公平性：多路径TCP连接在同一网络瓶颈处的多个子流，不能过多地侵占其它单路径TCP连接的带宽
- 均衡拥塞：多路径TCP连接应能在满足前两个准则的情况下，实现各个子流之间的负载均衡，尽可能将拥塞路径上的流量迁移到较好的路径上

➤ 公平性是关注的重点，MPTCP采用的是网络公平性原则

- 举例：假设MPTCP的两个子流与一个常规TCP流共享一个网络瓶颈，则MPTCP两个子流获得的带宽应为瓶颈带宽的 $1/2$ ，而不是 $2/3$ ，这样才能保证MPTCP连接对常规TCP连接的友好性

➤ MPTCP实现网络公平性准则的方法：

- 限制各个子流窗口增长速度的总和，不超过单路径TCP连接的窗口增长速度
- 从而，MPTCP连接通过各子流获得的总吞吐量和单路径TCP连接获得的吞吐量相当



高速移动场景下基于多运营商的MPTCP 性能测量与分析

A Measurement Study on Multi-path TCP
with Multiple Cellular Carriers on High Speed Rails



高铁(HSR)成为人们出行的首选

38,000
km

Length

66%

China

310
km/h

Speed

1.7
billion

Passenger

30%

Growing

30,000
km

2020



High speed
mobility



高速的网络技术在高速的列车上，却提供的是低速的网络连接！

不同运营商的基站分布具有异质性



移动



联通



电信



联通



电信



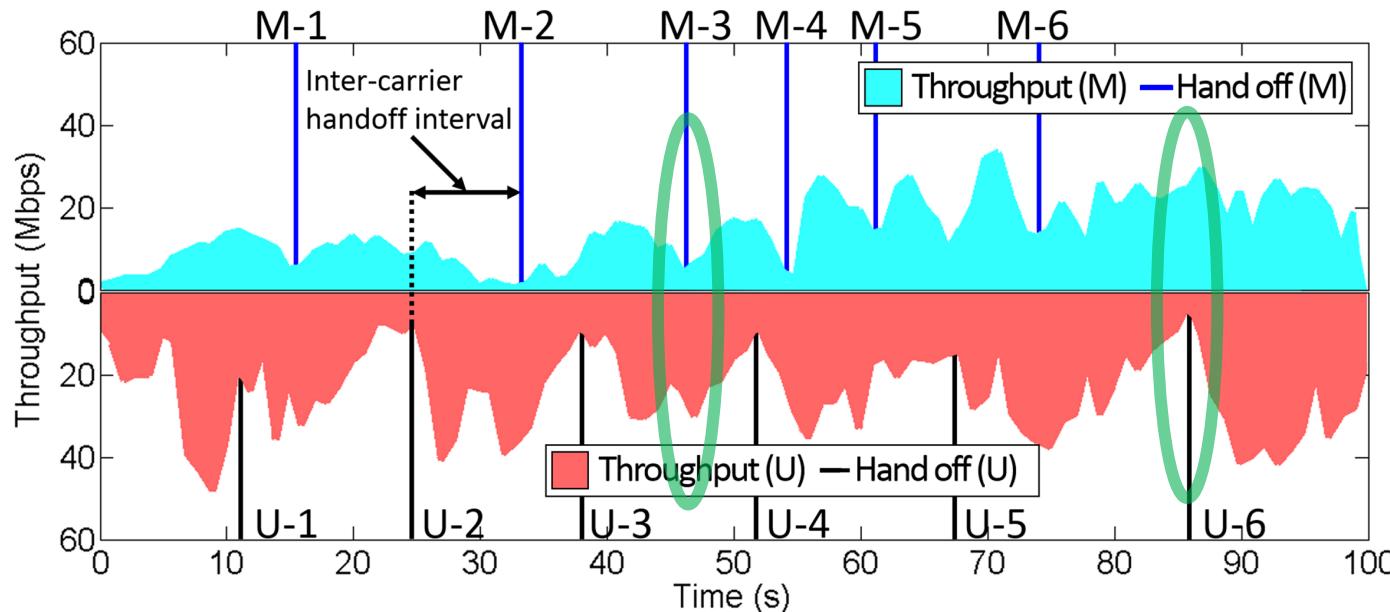
移动



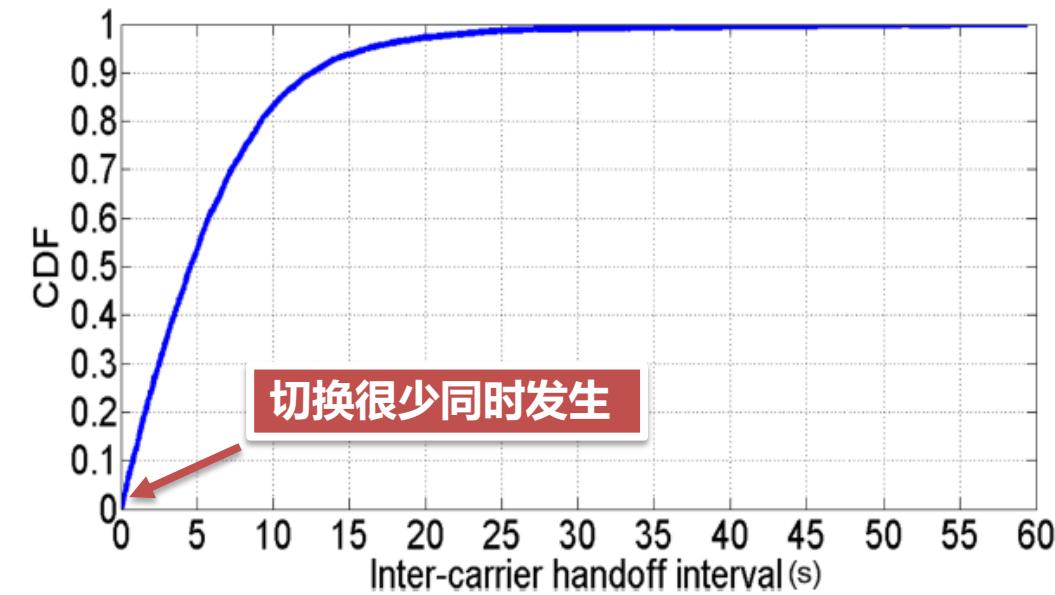
联通

能否利用运营商之间的互补性？

互补性：不同运营商的切换时机不同



举例：同一个终端在高铁上使用两个不同的运营商

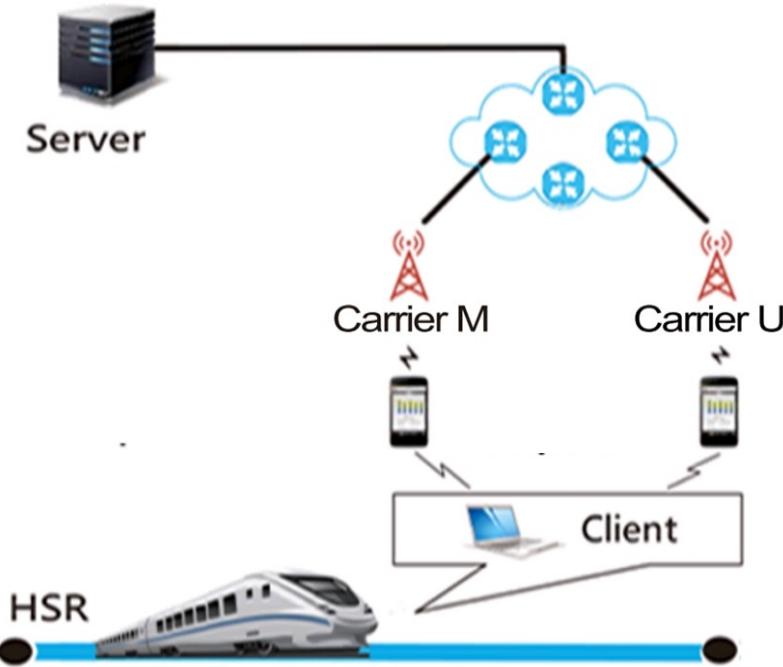


运营商切换间隔

科学问题：探究使用多路径TCP（MPTCP）的潜在收益

测量和数据收集方法

测试平台



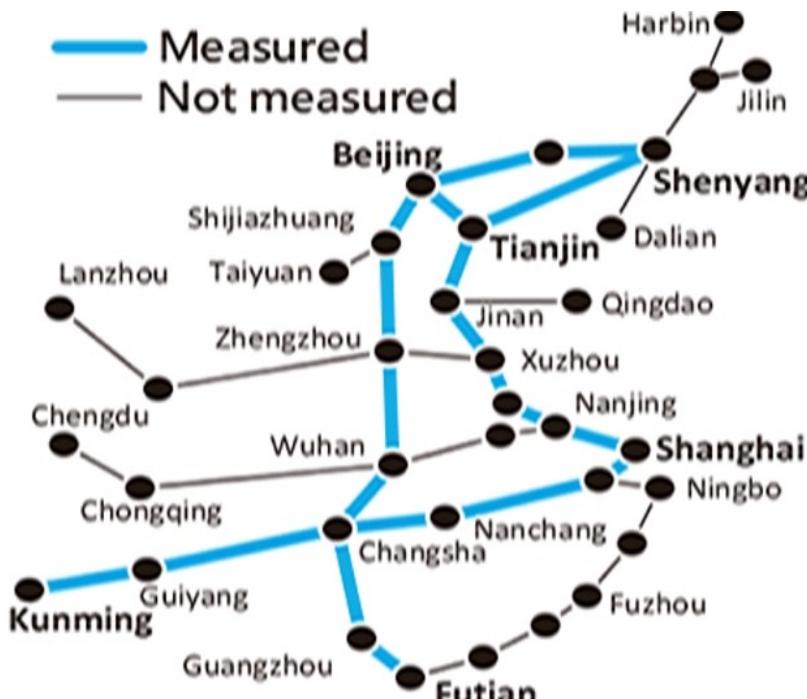
笔记本同时连接
多个手机的蜂窝网络（USB或热点共享）

MobiNet安卓应用

The figure shows the MobiNet application interface and its listing on the Google Play Store. The app screen displays a TCP Downlink Test configuration with Server IP: 123.57.73.173, Duration: 100 min, Packets: 10, and a note about data connection and GPS. It also shows signal strength (2G:99 4G:-1), parameters (RSRP:7 RSRQ:99 SNR: 2147483647), and current location information. The Google Play Store listing shows a 5-star rating, 5655 downloads, and a green '安全下载' button. A link to the app's page is provided: <http://www.wandoujia.com/apps/thu.kejifan.mobinet>.

开发安卓测试工具，实时记录地理位置、车速、
网络类型和切换等

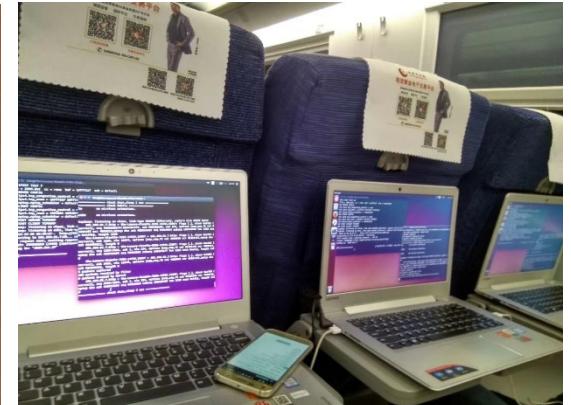
历时22个月，覆盖中国6条主要高铁线路



历时22个月
覆盖6条主要线路

PS: 如果要计算里程的话，总里程可以绕地球赤道2圈

某一天测试所用高铁票



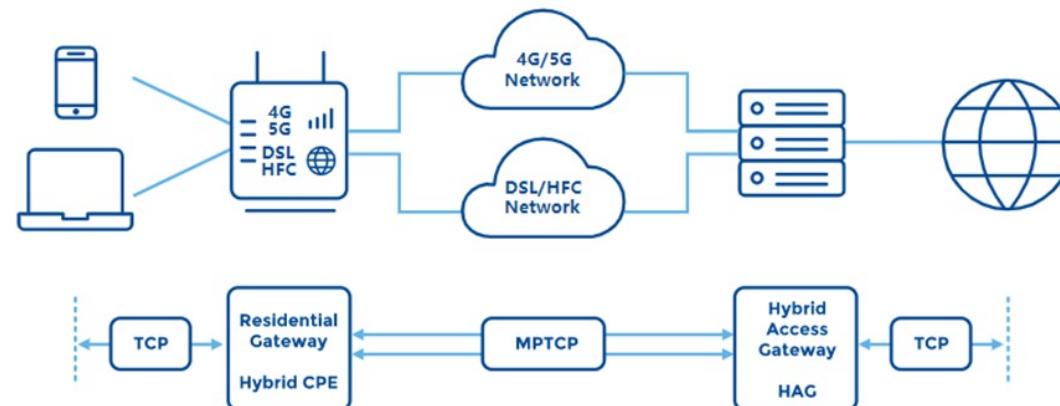
部分SIM卡



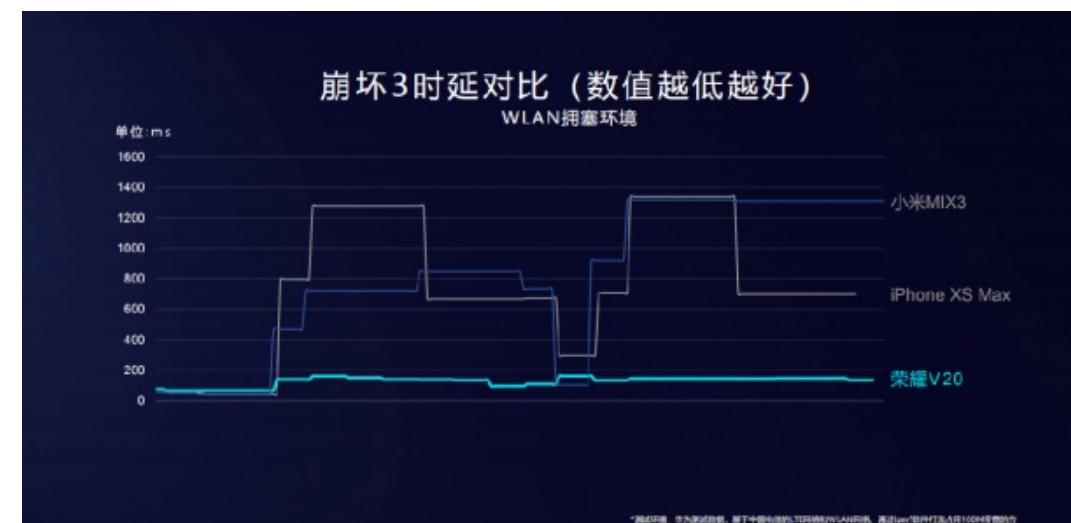
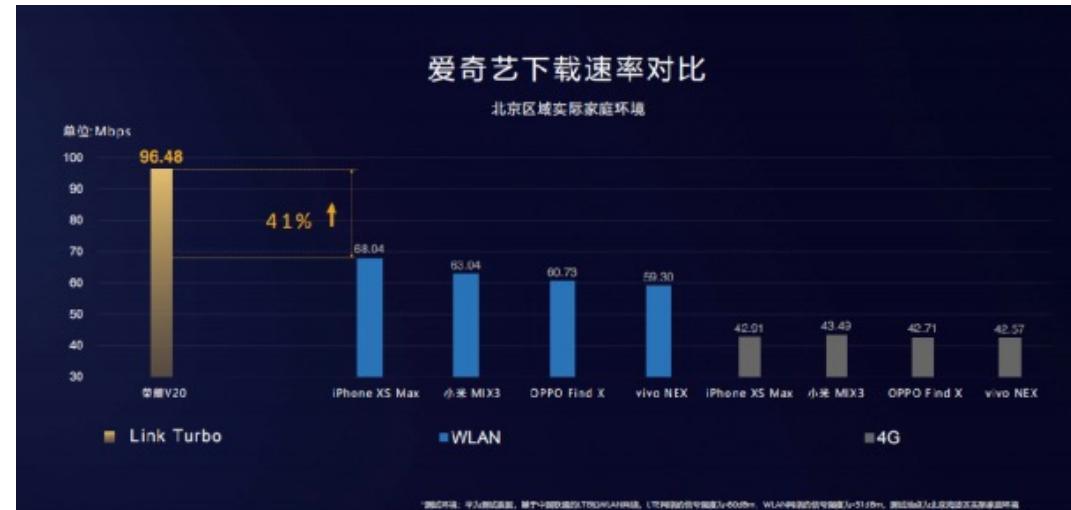
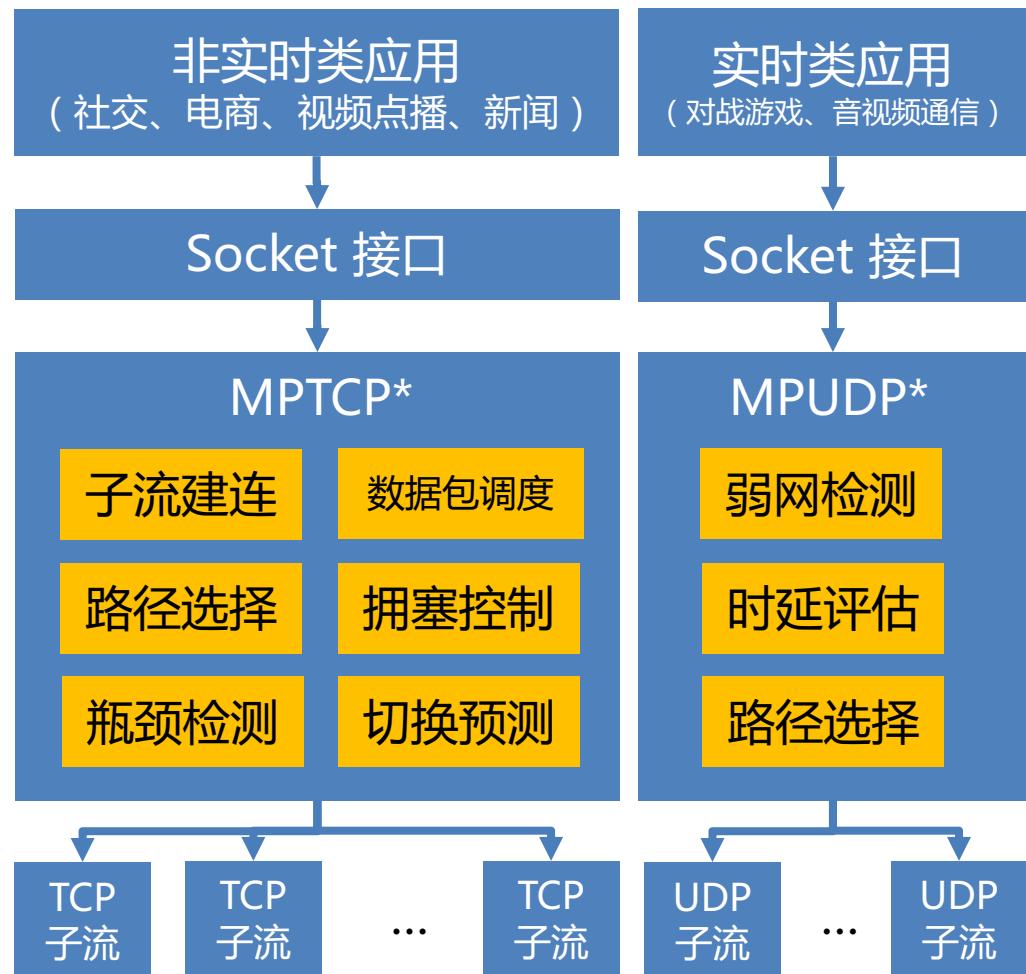
MPTCP的应用现状



- Linux内核已经支持MPTCP协议，当前最新版本为v0.95（<http://www.multipath-tcp.org/>）
- iPhone和iPad上的Multipath TCP（<https://support.apple.com/zh-cn/HT201373>）
 - 在同时支持蜂窝移动数据网络和Wi-Fi网络的iOS设备中，Siri 尝试通过 Wi-Fi 建立 MPTCP 连接；如果连接成功，Siri 通过蜂窝移动数据建立备用连接；如果 Wi-Fi 不可用或不可靠，MPTCP 立即在后台切换到蜂窝移动数据网络
- 三星的手机设备 Galaxy S6 和 S6 Edge 已经在韩国KT运营网络中支持MPTCP协议
- Citrix公司的Web应用交付方案Netscaler 和F5公司的 BIG-IP 目前也同样支持使用MPTCP协议提升网络性能
- Tessares公司推出的商业化MPTCP解决方案可以在用户端和服务器端快速部署MPTCP服务



华为：MPTCP深度优化+首创MPUDP聚合



2018年，华为发布全网络聚合加速技术Link Turbo，成为荣耀、华为系列旗舰手机核心卖点

LINK TURBO

首次实现移动数据通信和固网数据通信技术的融合演进

HONOR



荣耀 V20

科技标杆 见所未见

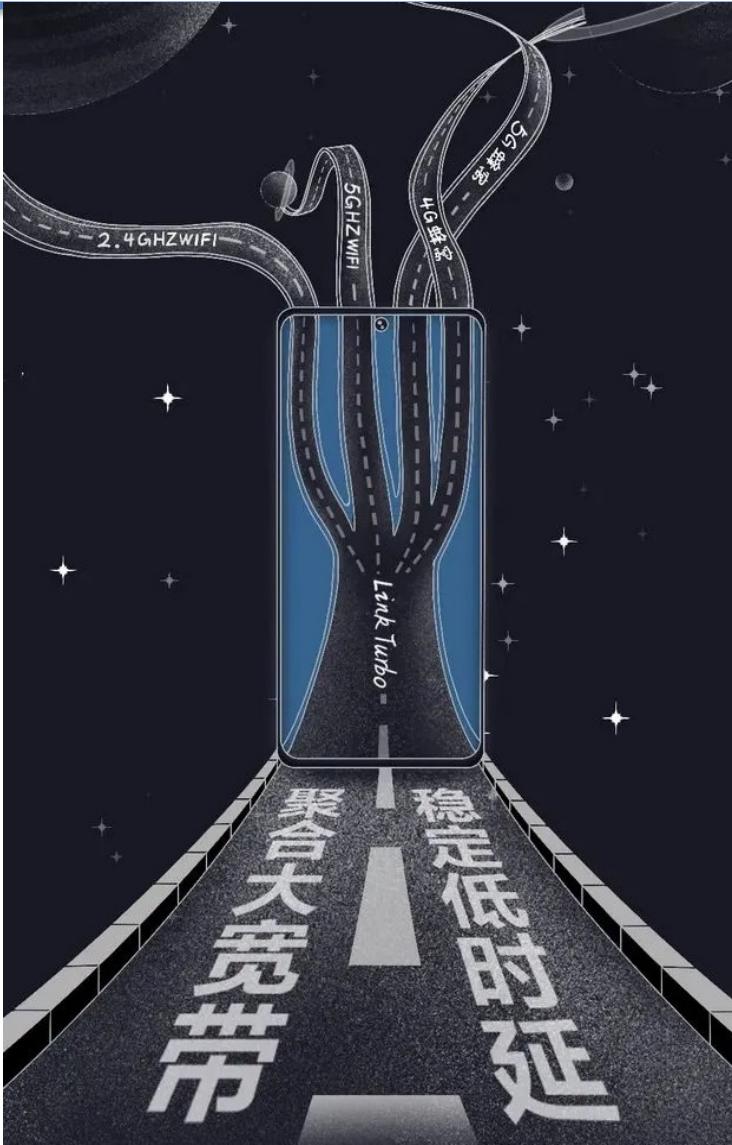
Link Turbo技术获华为十大发明奖

构建万物互联的智能世界

HUAWEI

十破天惊 万象更新

2022年6月8日，华为公司公布了在其第四届“十大发明”评选活动中获奖的重大发明。一系列有潜力开创新的产品系列、成为产品重要商业特性，并为行业带来巨大商业价值的发明脱颖而出。

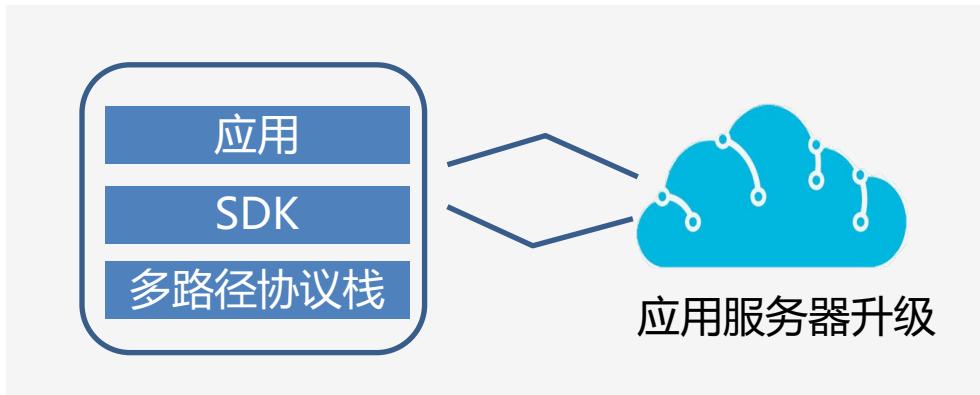


鸿蒙网络聚合加速 与内存扩展

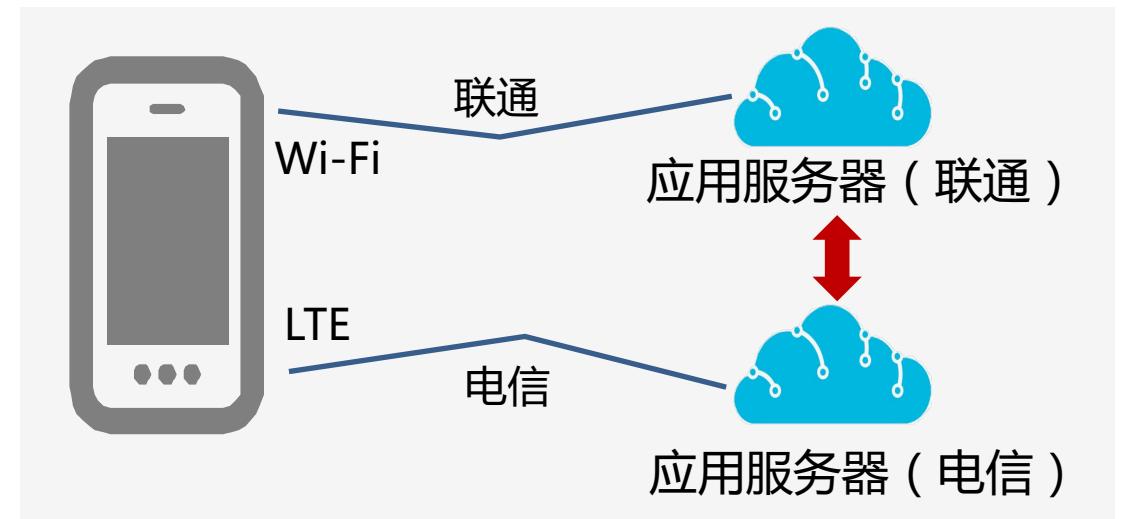
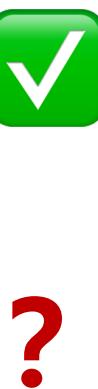
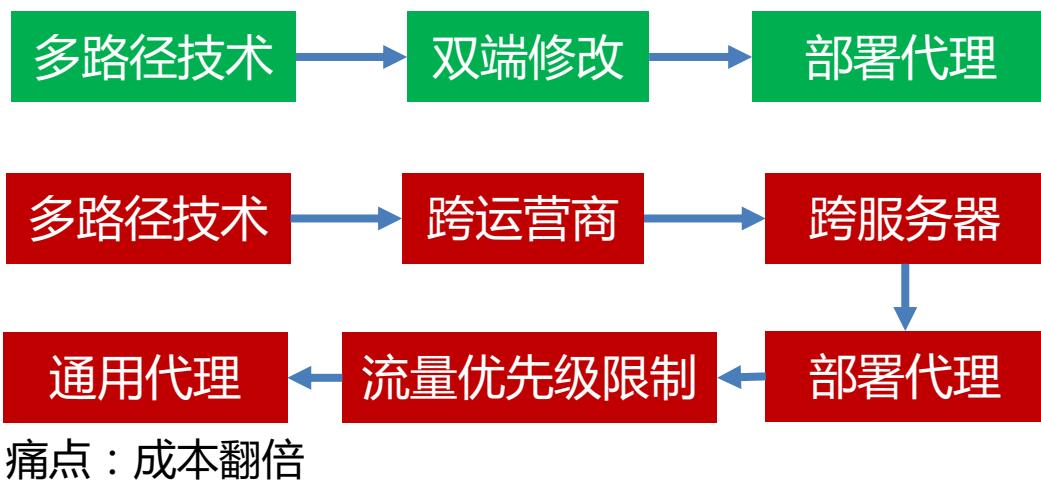
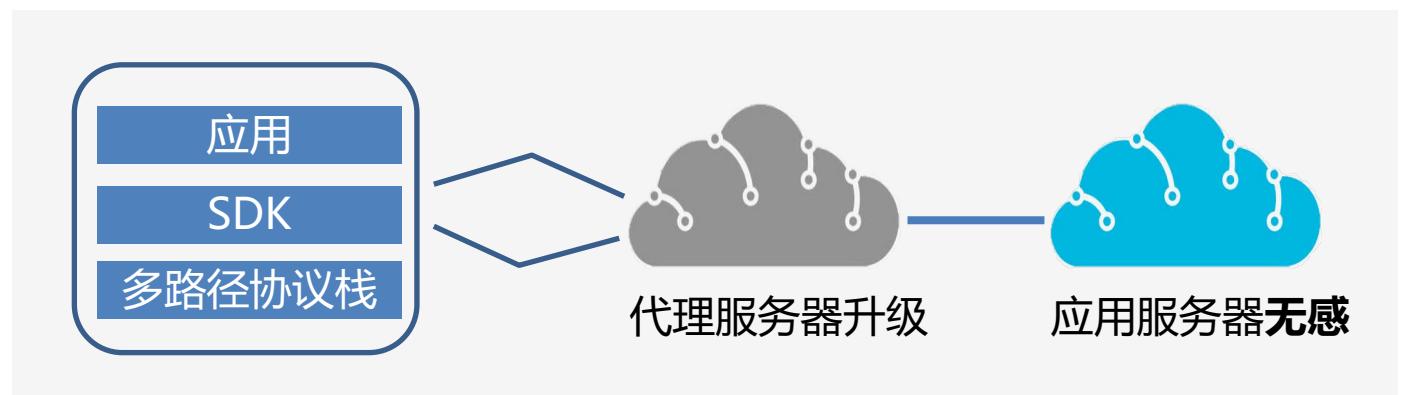
该项专利包揭示了在华为终端产品上广泛使用的网络与内存革新技术——包括终端产品上Wi-Fi、蜂窝的多网融合LinkTurbo技术以及内存动态扩展Hyperhold技术。LinkTurbo以纯软方案将多个网络模组的能力堆叠融合，在网络受限、抖动以及关键业务场景下，并发下载速率提升83%，游戏时延降低69%，视频起播时延下降87%，多网协同能力互补，“拳头攥在一起力量更强”；Hyperhold大幅扩展可用内存，减少低内存频繁换入、换出引发的卡顿，提升基础读写性能，为用户带来极致流畅体验。

开放问题：规模部署是多路径技术的核心难点

模型一：直接改应用服务器（如：爱奇艺）



模型二：部署代理服务器（如：崩坏3）





本章内容



- 6.1 概述和传输层服务
- 6.2 套接字编程
- 6.3 传输层复用和分用
- 6.4 无连接传输：UDP
- 6.5 面向连接的传输：TCP
- 6.6 理解网络拥塞
- 6.7 TCP拥塞控制
- 6.8 拥塞控制的发展
- 6.9 传输层协议的发展

1. MPTCP
2. QUIC

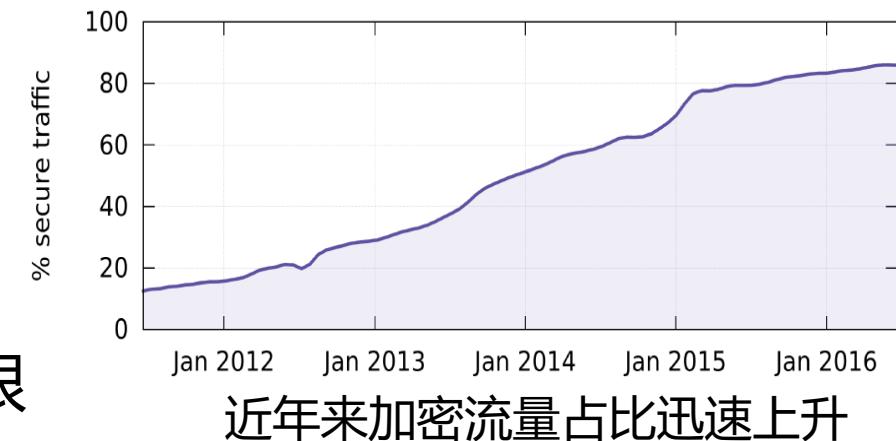


TCP存在的问题



中國人民大學
RENMIN UNIVERSITY OF CHINA

- TCP 实现在操作系统内核中
 - 作为传输优化的最终受益者，**应用无法对TCP进行修改**
 - 操作系统的更新往往跟不上应用的需求和节奏
- TCP体系握手时延大
 - 互联网上的大趋势：**低时延需求越来越强烈**；加密流量占比越来越大
 - TLS(传输层安全性协议)+TCP的体系握手时延很大，传输前需要3个RTT进行握手





TCP存在的问题



中國人民大學
RENMIN UNIVERSITY OF CHINA

- TCP多流复用存在队头阻塞问题
 - 当前的应用普遍比较复杂，常需要同时传输多个元素
 - 例如：网页传输中，每个单独的图片即为一个数据流，不同数据流之间相互独立。为每个数据建立一个TCP连接很高效（尤其对于小流，单独建立连接成本高昂），因此出现了**多流复用**
 - TCP传输需要保持**有序性**
 - 出现丢包时，后面的数据需要等丢失的包重传完成才能使用，这就导致了队头阻塞



网页中的多个图片可以同时传输



上图：每个数据流使用单独的TCP连接
下图：**多个数据流复用一个TCP连接**



QUIC在网络体系结构中的位置



中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ 传统的传输架构

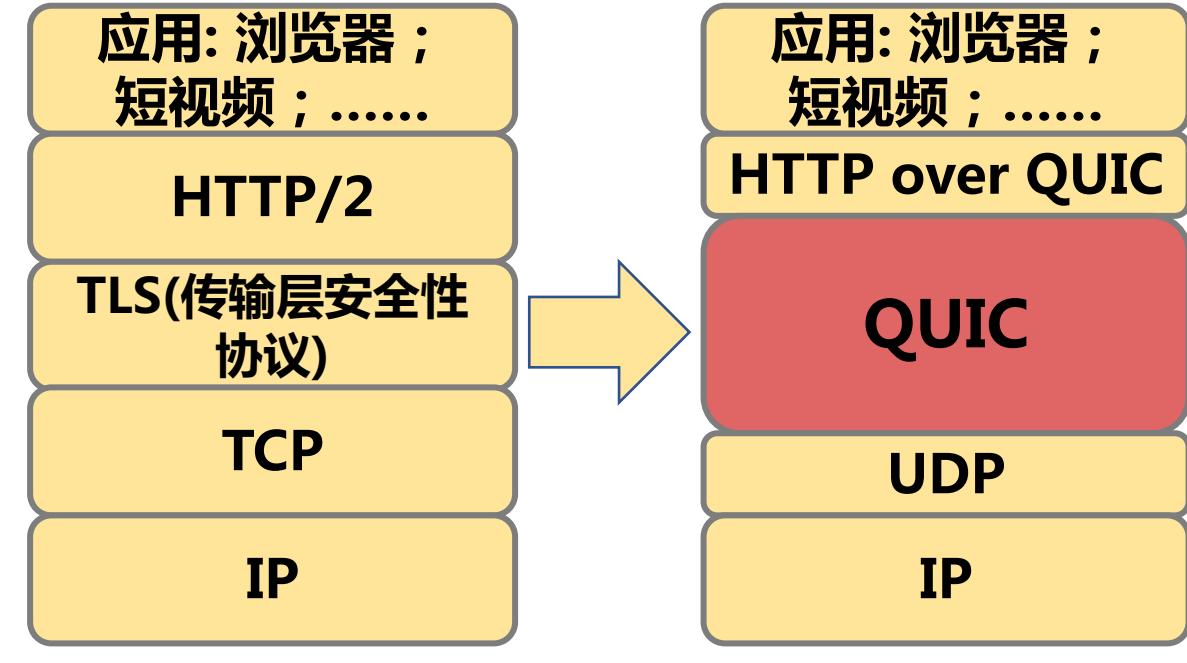
- TCP提供数据传输服务
- TLS（传输层安全性协议）对数据进行加密
- HTTP协议定义如何发起请求-响应请求
- 应用在HTTP之上实现

➤ 基于QUIC的传输架构

- QUIC替代TCP、TLS和部分HTTP的功能

➤ QUIC实现在用户态中

- 底层基于UDP实现
- 拥塞控制是模块化的，可以方便地使用各种TCP拥塞控制算法，如Cubic等





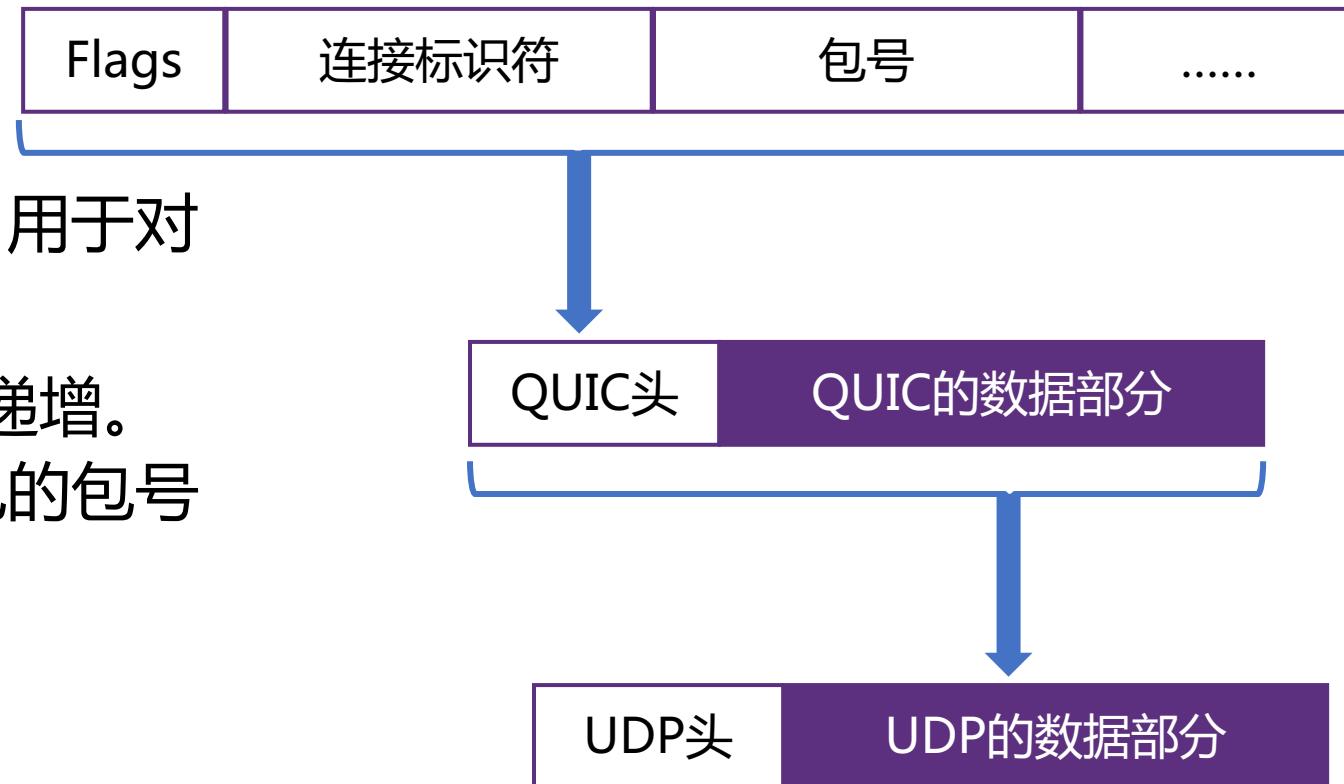
QUIC包格式介绍（简化）



中國人民大學
RENMIN UNIVERSITY OF CHINA

➤ 部分相关字段：

- 连接标识符(Connection ID)：用于对连接进行表示和识别
- 包号(Packet Number)：单调递增。即同一个连接中，每个QUIC包的包号都不一样



➤ QUIC底层使用UDP进传输

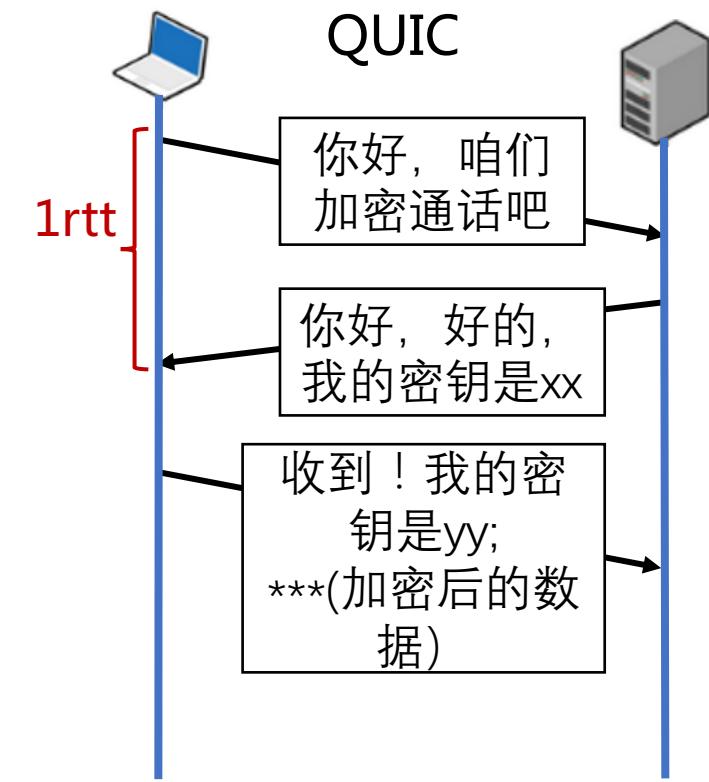
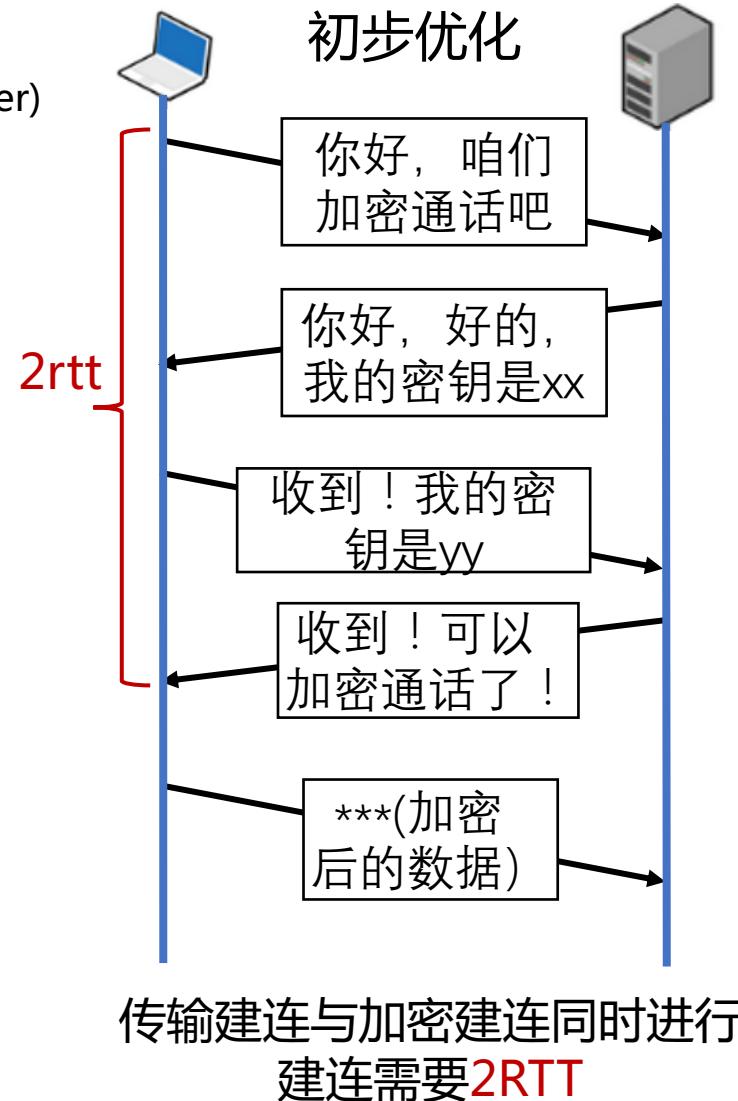
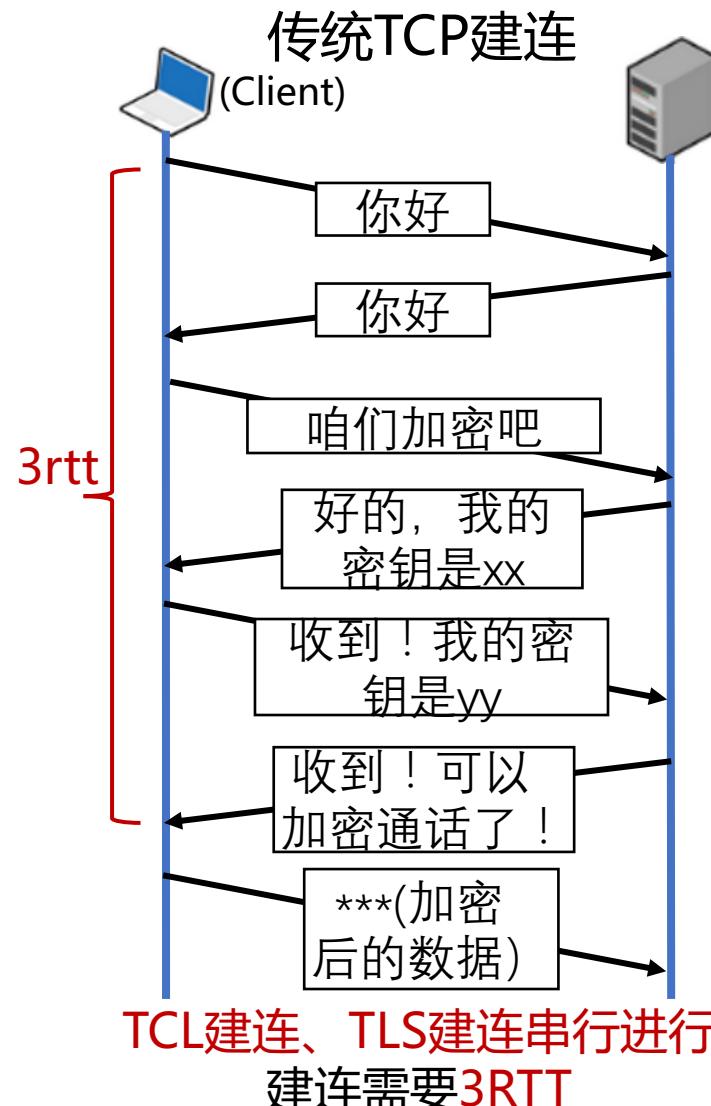
- QUIC包作为UDP的数据载荷
- IANA (互联网数字分配机构) 建议 QUIC使用UDP的443端口



连接建立时延优化：简化演示



中國人民大學
RENMIN UNIVERSITY OF CHINA





➤ TCP重传歧义的问题：

- TCP的重传包使用和原包相同的序号，因此可能某一序号被用了不止一次
- TCP收到这一序号的ACK时，无法判断是针对哪个包的ACK，从而影响后续操作，如测量RTT的大小

➤ QUIC解决重传歧义的方法：

- QUIC的packet number单调递增，对于重传包也会递增packet number
- 每个packet number只会出现一次，ACK没有歧义！
- QUIC接收端记录收到包与发出ACK之间的时延，并反馈给发送端，方便发送端更准确地测量RTT

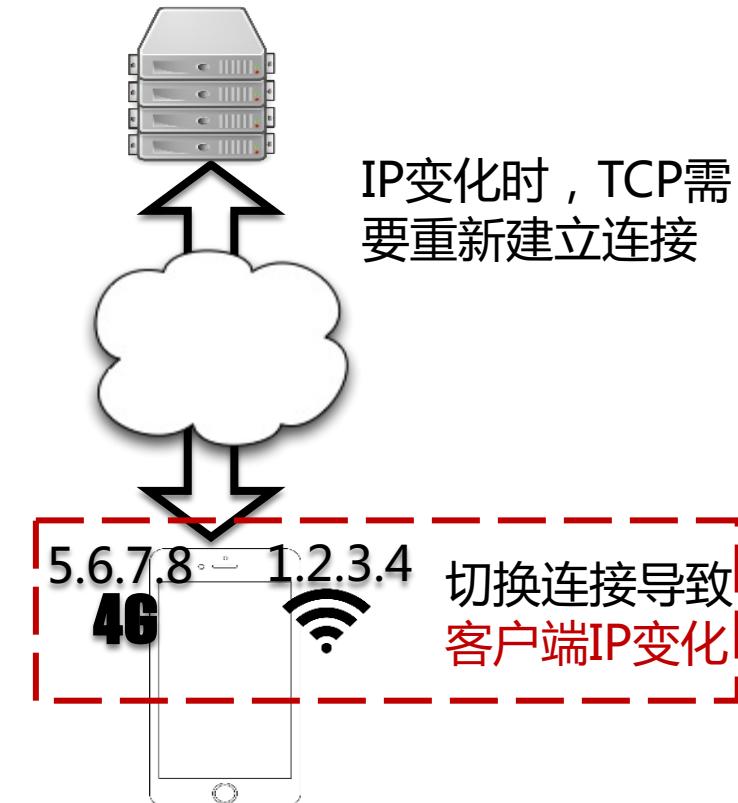


IP地址/端口切换无需重新建立连接



中國人民大學
RENMIN UNIVERSITY OF CHINA

- IP地址/端口发生变化时，TCP连接会断开
 - 例如手机WIFI断开时，常常自动转而使用移动信号
 - 此时，TCP会断连，需要应用进行处理
- QUIC支持IP/端口切换
 - QUIC使用Connection ID来表示每个连接
 - IP地址或端口的变化不影响对原有连接的识别
 - 客户IP地址或端口发生变化时，QUIC可以快速恢复
- 由传输层对连接的切换进行管理
 - 更符合互联网体系结构
 - 不再需要应用重复造轮子





无队头阻塞的多流复用



中國人民大學
RENMIN UNIVERSITY OF CHINA

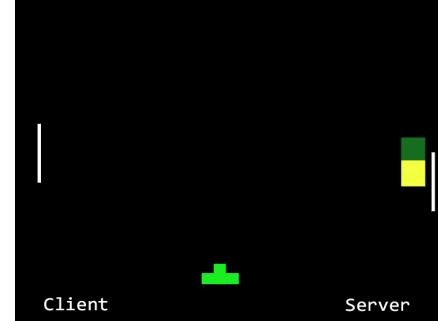
➤ 多流复用时的队头阻塞问题

- TCP为保持数据的**有序性**，出现丢包时，会等待该数据到达后，再提交给上层应用
- **多流复用**时，某个数据流的数据包丢失，会使得TCP连接上所有数据流都需要等待

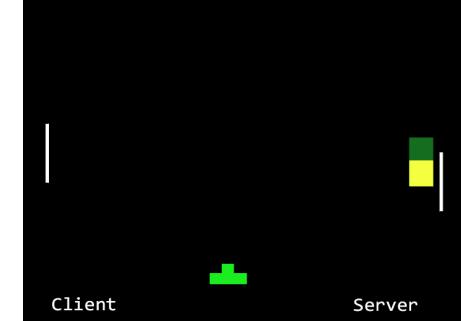
➤ QUIC对队头阻塞问题的解决

- 在单个连接中，建立相互独立的多个QUIC流，某个流的数据包丢失不影响其它流的数据交付
- 分析：TCP保证了整个连接的数据交付的**有序性**（**有序**）；QUIC利用了各个流相互独立的特性，仅保持了流内部数据的**有序性**（**部分有序**），减少了不必要的等待

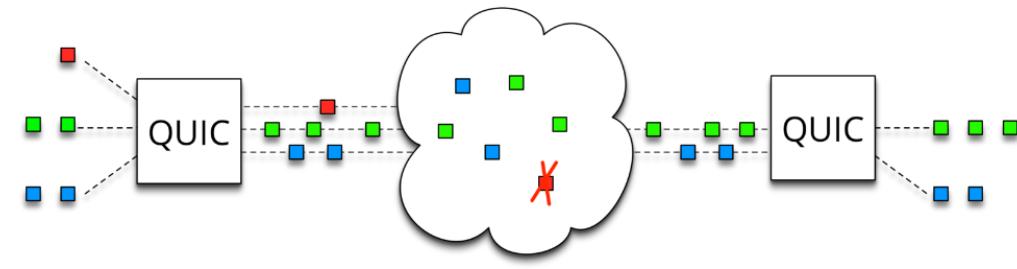
动画：同时发送两个流，黄颜色的流上有丢包



TCP丢包会阻塞所有后续的流



QUIC丢包只会影响相关的流



红色流的包丢失，不会阻塞绿色和蓝色的流

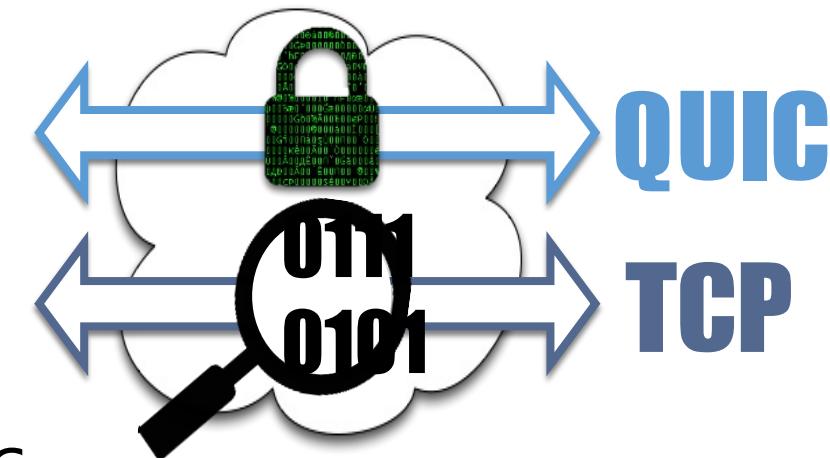


QUIC易于部署和更新



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 整个QUIC包被加密传输
 - 保护用户数据隐私
 - 避免被中间设备识别和修改
- QUIC在用户态实现
 - 与操作系统解耦，从而能和应用一同快速迭代
- 版本协商机制
 - 由于QUIC的快速迭代特性，会同时存在众多QUIC版本
 - 客户需要和服务器进行版本协商（不引入额外时延的协商机制）



中间设备无法识别和修改
QUIC header中的信息



QUIC的发展状态



- 截止2017年，互联网上7%的数据使用QUIC进行传送
- 2018年，IETF宣布，HTTP/3将弃用TCP协议，改为使用QUIC协议实现
- 2020年，华为在最新的HMS core网络加速套件中推出hQUIC
- 2020年，Facebook宣布其超过75%的网络流量使用QUIC；.....

正在推进的新特性（截止2020年底）

- 加密算法由谷歌定义的算法，换成更通用的TLS
- QUIC不可靠传输的扩展
- 可路由的Connection ID
- 截止时间可感知的QUIC扩展
-



本章总结



中國人民大學
RENMIN UNIVERSITY OF CHINA

- 传输服务原理：
 - 可靠数据传输
 - 流量控制
 - 连接管理
 - 拥塞控制
- Internet中的传输服务：
 - UDP
 - TCP
 - MPTCP
 - QUIC