第七章 传输层进阶

崔 勇

清华大学

计 算 机 网 络
教 案 社 区

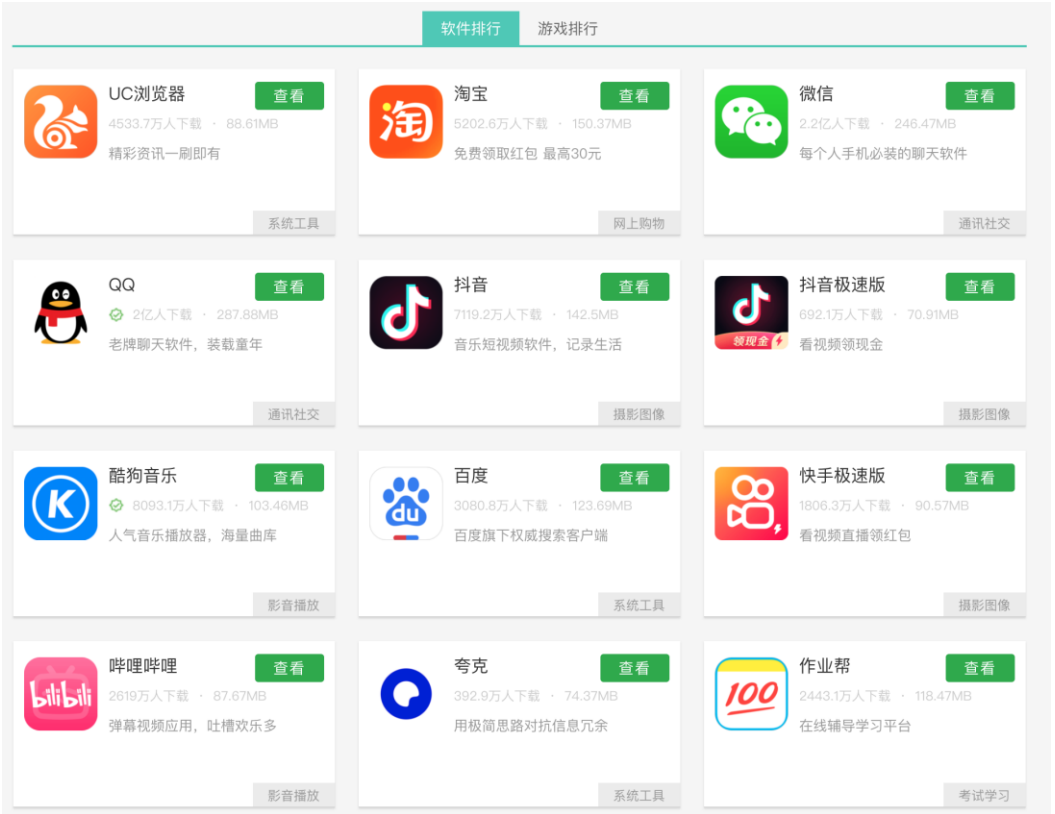
致谢社区成员	
人民大学 何军	清华大学 崔勇
中国科技大学 华蓓	安徽工业大学 李沁

回顾与展望

- UDP的局限性
  - 回忆：平时用的互联网应用，可靠的多还是不可靠的多？
  - 微信发送消息，可靠与实时性哪个更重要？邮件呢，尽力而为可以吗？
  - 看视频：4k但不断跳帧 vs 1080P完整观看，选哪个？画面前后颠倒？

可靠的TCP是核心

- TCP的优化和升级
  - 丢失真要降速吗？RTO队头阻塞？
  - 多接口多路径如何同时使用？
  - 在什么场景下更能实现端网协同？



新时代的新需求：云计算？

本节目标

1. 掌握广泛使用的拥塞控制算法Cubic和BBR
2. 了解新型传输协议QUIC，掌握其基本原理
3. 了解多路径传输的难点和MPTCP协议设计思路
4. 掌握数据中心网络及其传输协议设计思路

本节内容

7.6 拥塞控制的发展

- 7.6.1 经典拥塞控制的性能问题
- 7.6.2 CUBIC 算法
- 7.6.3 BBR 算法

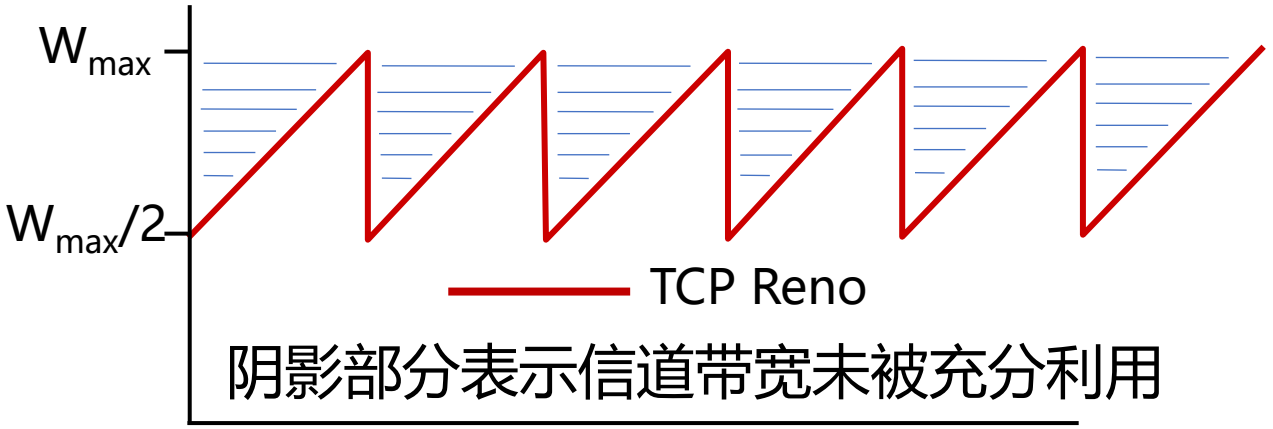
7.7 新型传输层协议QUIC

7.8 多路径传输协议MPTCP

7.9 数据中心网络传输协议

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的工作方式
  - 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（Maximum Segment Size，最大报文长度）
  - 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



维持在满载窗口?
网络动态性?
更快达到满载窗口?

TCP-BIC: Binary Increase Congestion

- 查找满载窗口的约束条件
  - 发生丢包时的窗口上限 W_1：为保持满载而不丢包，满载窗口应小于 W_1
  - 窗口下限 W_2：检测到丢包并将窗口乘性减小为 W_2，满载窗口应大于 W_2
  - 窗口更新频率：受ACK时钟驱动，即以RTT为更新间隔时间
- BIC算法对满载窗口进行二分查找
  - 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

满载窗口的变化？向下、向上

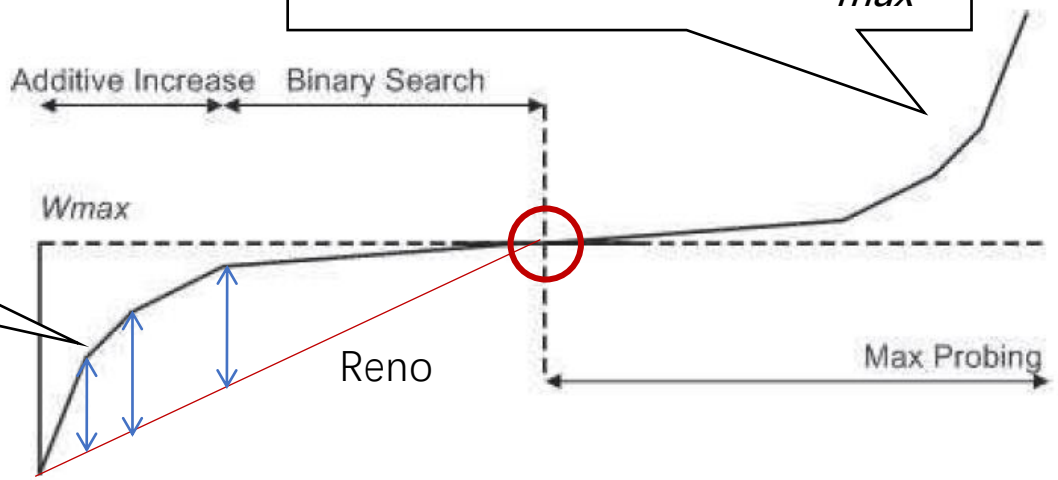
- 最大探查
  - 如窗口再次达到 W_1而没有丢包，说明满载窗口大于 W_1，则以逼近 W_1的镜像过程增大拥塞窗口

TCP-BIC 图解

满载窗口的变化？
向下、向上

当W_{max}发生更新时，以先慢后快方式探测W_{max}，全过程尽可能接近W_{max}

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

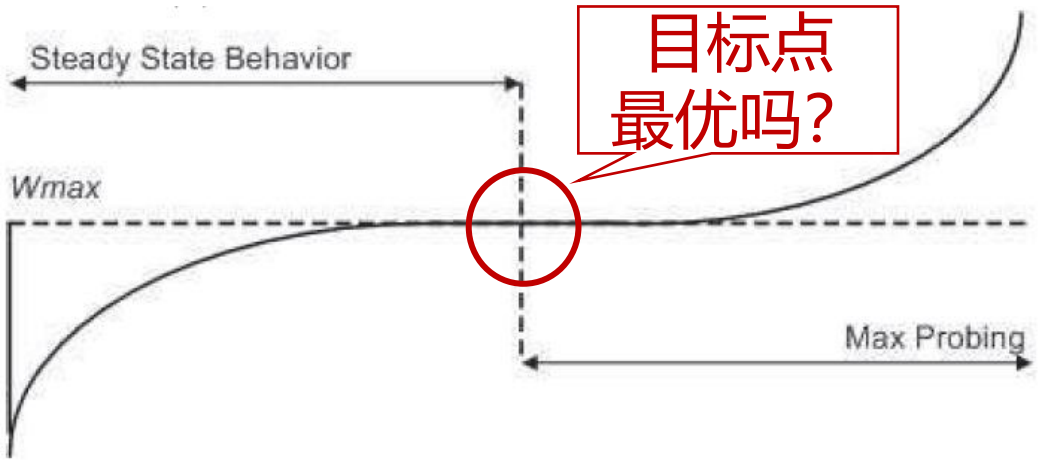


- BIC存在带宽不公平性问题
  - BIC以ACK时钟驱动拥塞窗口的更新，RTT较短的连接会更快到达满载窗口，占据更多的带宽，产生不公平性问题（RTT-fairness）

TCP CUBIC

- CUBIC基于BIC的算法优化
  - CUBIC将BIC算法连续化，用三次函数拟合BIC算法曲线
  - 不再根据RTT间隔来确定调整窗口的时机，避免RTT不公平问题：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：以上凸函数增长逼近最近一次丢包时窗口
  - Max probing阶段：以下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
- CUBIC的缺点分析?

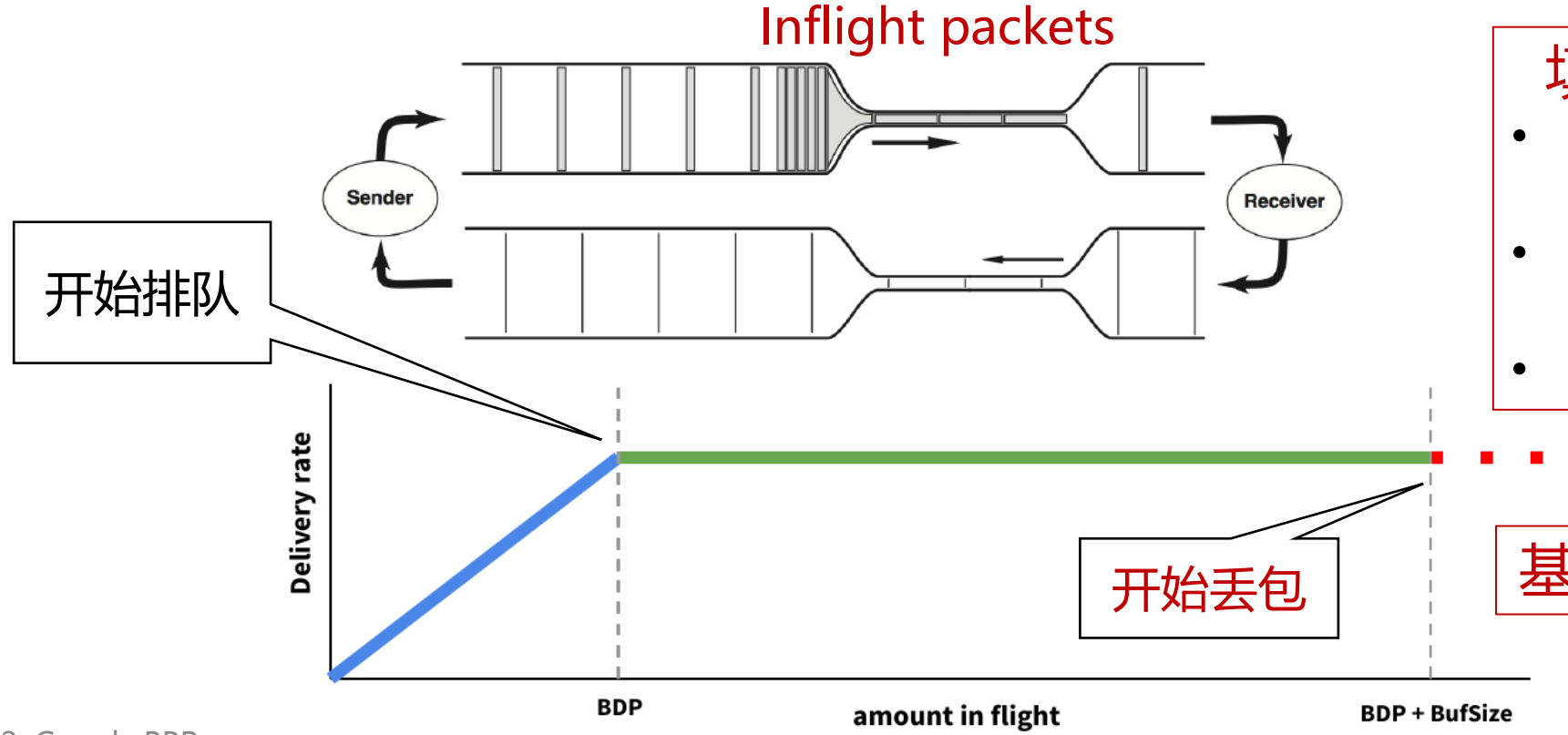
如何应对无线网络随机丢失?



拥塞与瓶颈链路带宽

➢拥塞与瓶颈链路带宽

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



填满缓冲区直至丢包

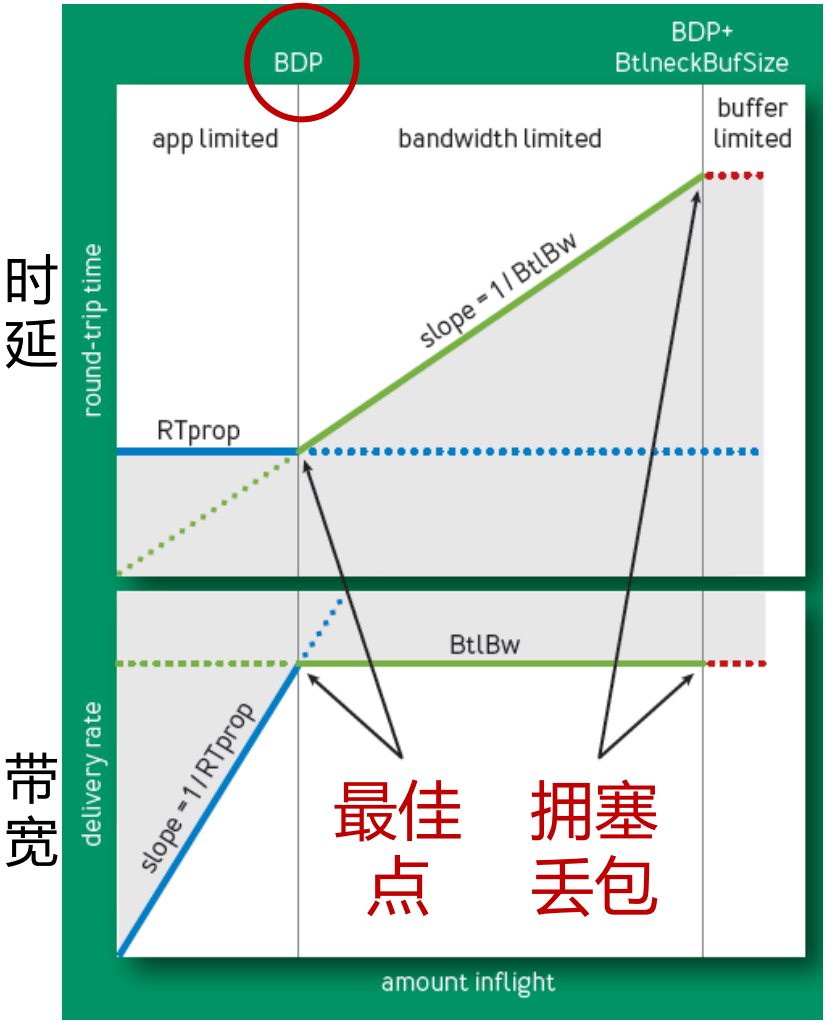
- 经典拥塞控制状态机以丢包事件为驱动
- 探测阶段填满瓶颈链路缓冲区，直至丢包
- 以此为依据乘性减小

基于丢包判断合适吗？

拥塞与瓶颈链路带宽

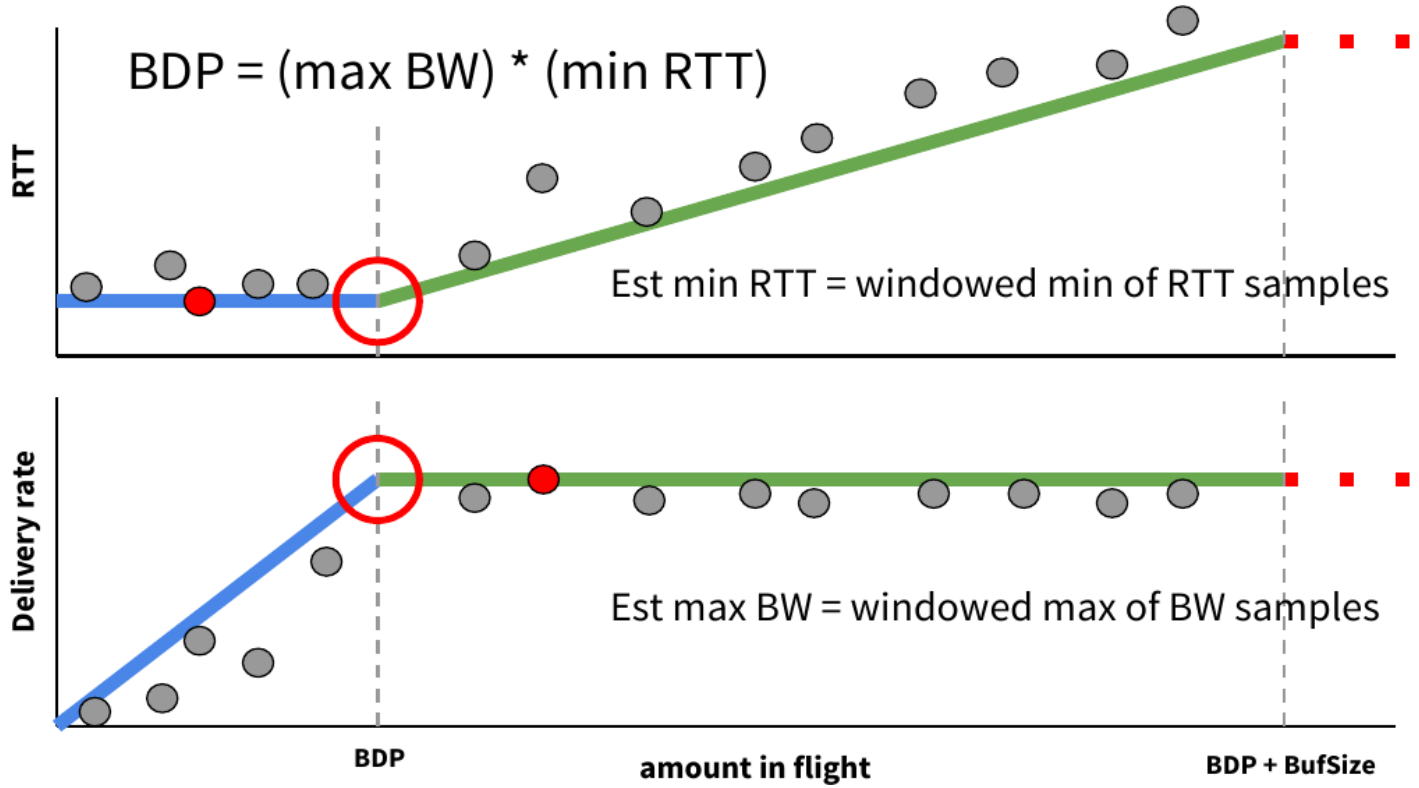
- 瓶颈链路带宽BtlBw
  - 不会引起路由器缓存排队的最大发送速率
- RTprop：往返时间（传播+队列？）
- 带宽时延积：BDP=BtlBw*RTprop
- BBR的优化思路
  - 试图测量图中左侧优化点的BtlBw
  - 尽量将cwnd收敛到实际BtlBw
  - 从而避免出现拥塞丢包，属于主动探测
- 以图中BDP竖线为分界点
  - BDP竖线右侧，Rtprop因瓶颈链路发生排队而逐渐增长

如何找到最佳工作点?



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

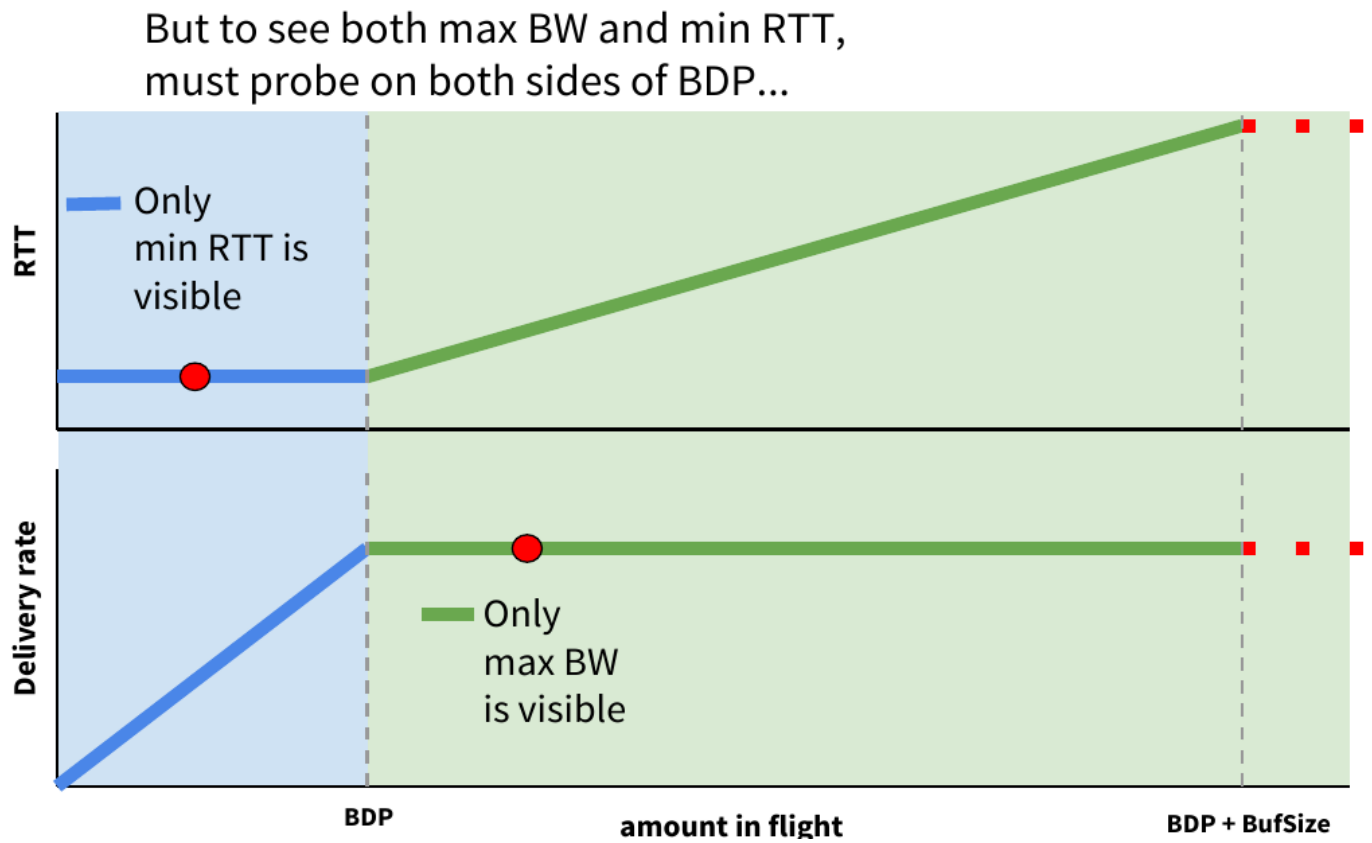


如何应对随机丢失?

如何精确测量?

Max BW和min RTT不能同时被测得

- 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
- 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

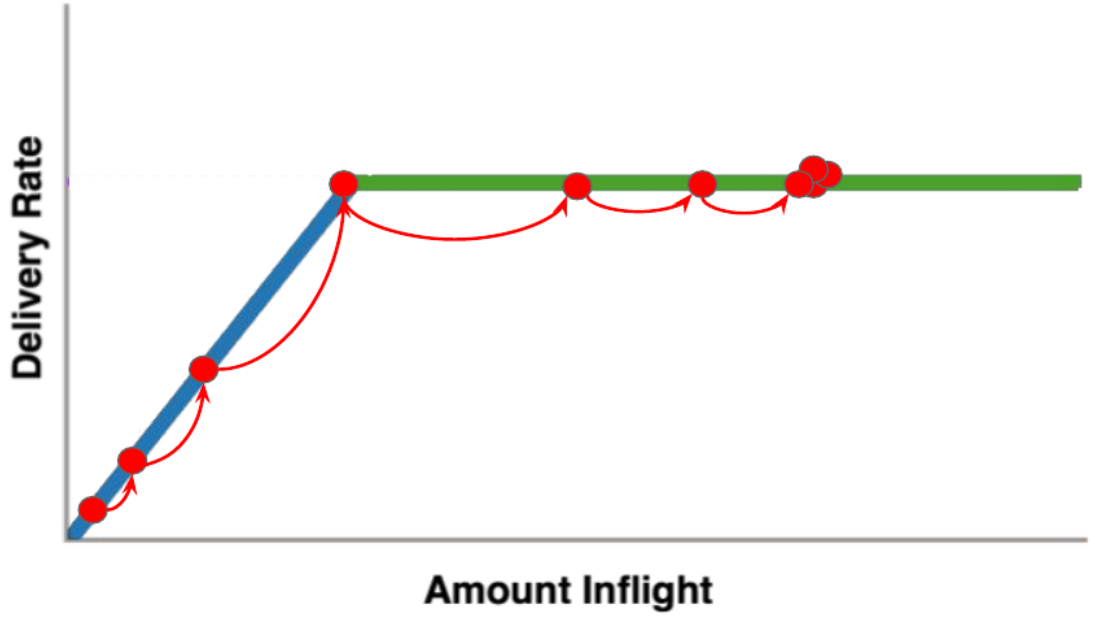


精确测量怎么办？

BBR: Bottleneck Bandwidth and Round-trip propagation time

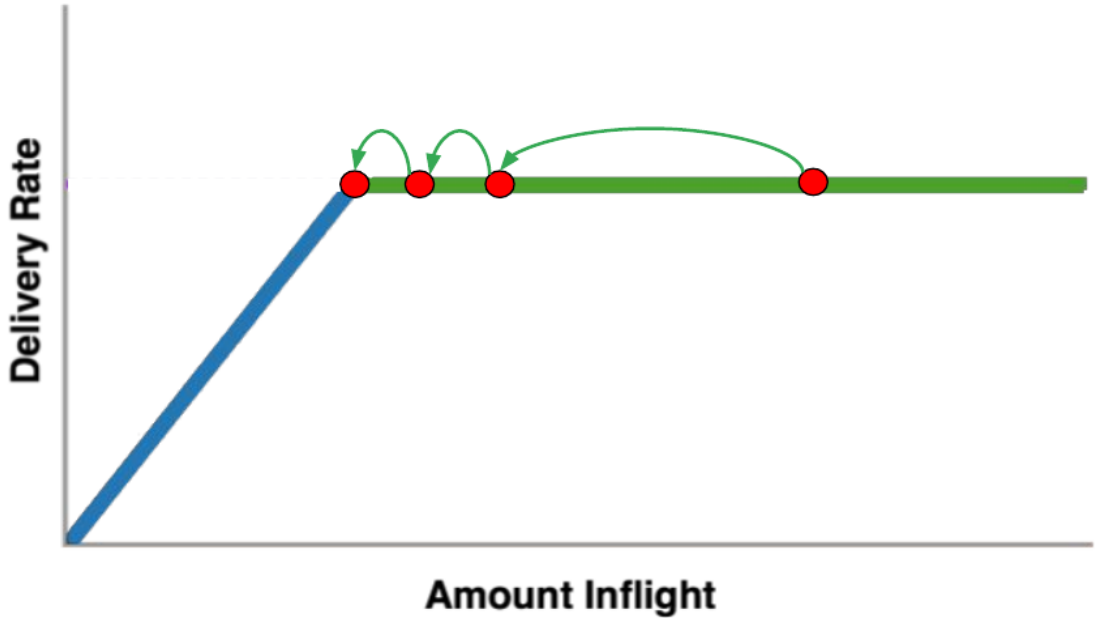
BDP检测：启动阶段（START_UP）

- 慢启动
  - 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
- 最大带宽发现
  - 若经过三次窗口增长，发现投递率不再增长，说明已达到最大带宽BtlBw
  - 瓶颈链路处分组已开始排队（事实上此时占的是三倍BDP）



BBR: Bottleneck Bandwidth and Round-trip propagation time

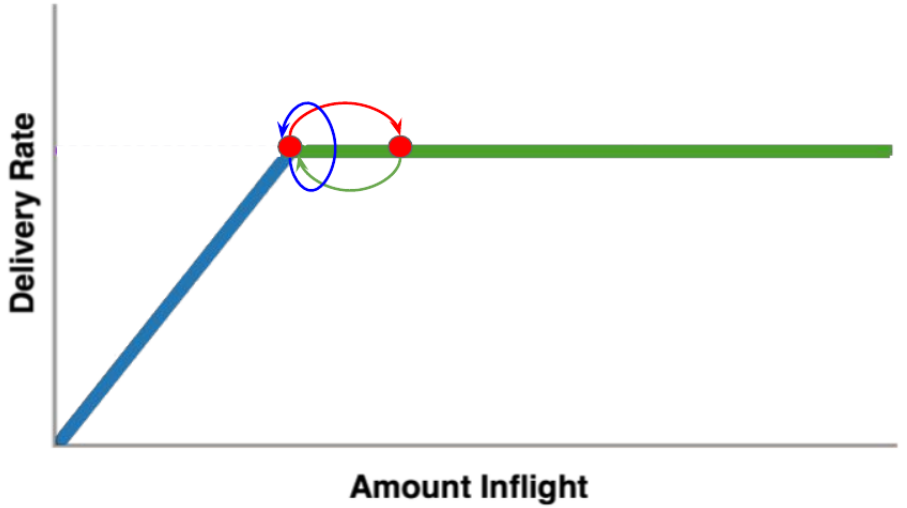
- BDP检测：排空阶段（DRAIN）
  - 指数降低发送速率（相当于是startup的逆过程），将多占的两倍buffer慢慢排空

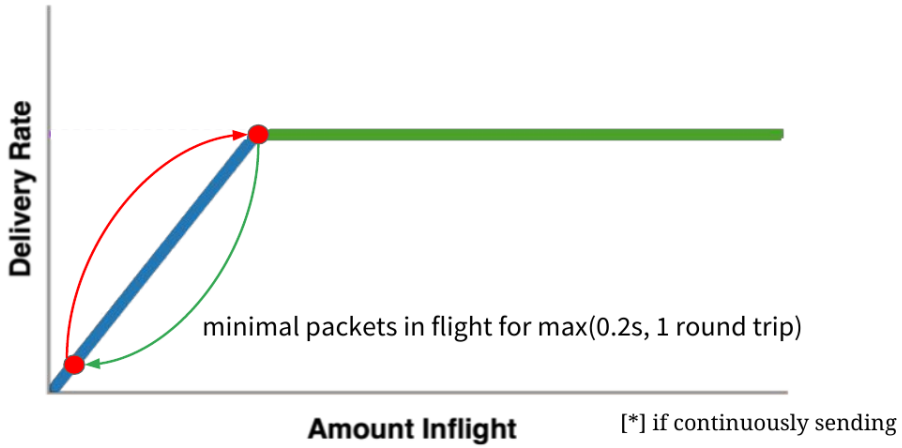


BBR BDP检测：不断探测

- 瓶颈带宽探测（PROBE_BW）
  - 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
  - 减小发送速率，排空前一个RTT多发的包
  - 后面6个周期使用更新后的估计带宽发送
- 时延探测（PROBE_RTT）
  - 每过10秒，如果估计的RTprop不变，就进入RTprop探测阶段
  - 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

minRTT和可用带宽是核心！不响应丢包！





本节内容

7.6 拥塞控制的发展

7.7 新型传输层协议QUIC

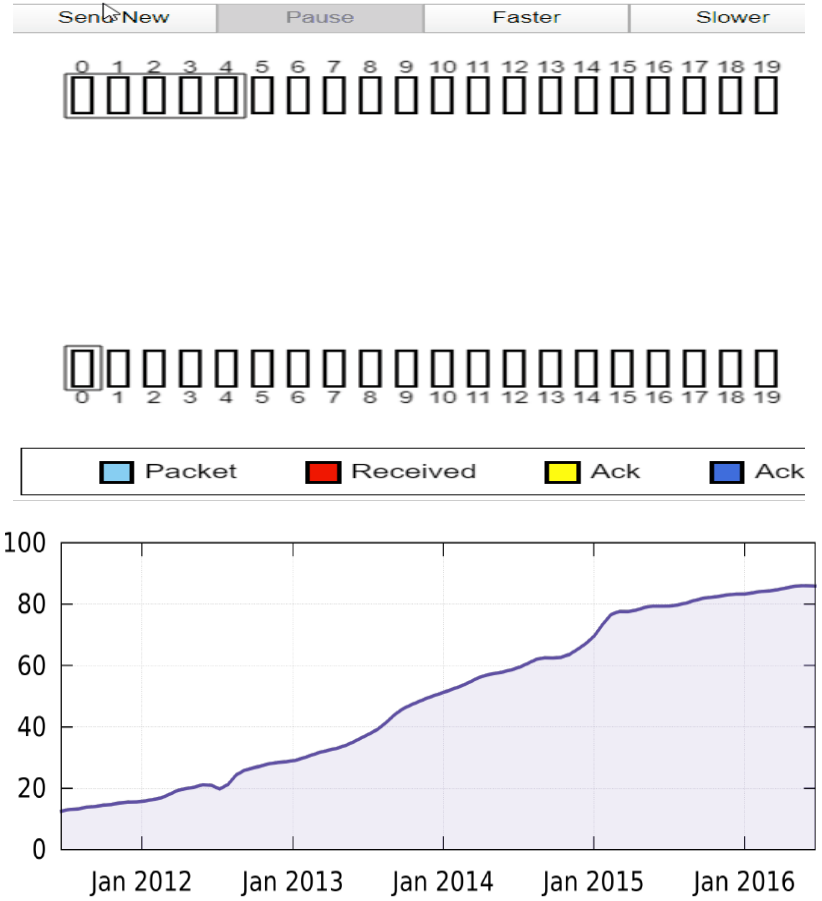
7.8 多路径传输协议MPTCP

7.9 数据中心网络传输协议

- TCP存在的问题分析
- QUIC在体系结构中的位置
- QUIC的主要优化思路
- QUIC包格式介绍
- QUIC的发展状态

TCP存在的问题

- RTO造成队头阻塞，严重降低传输性能☹
- TCP实现在操作系统内核中
  - 作为传输优化的最终受益者，应用难以对TCP进行优化和调整
  - 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

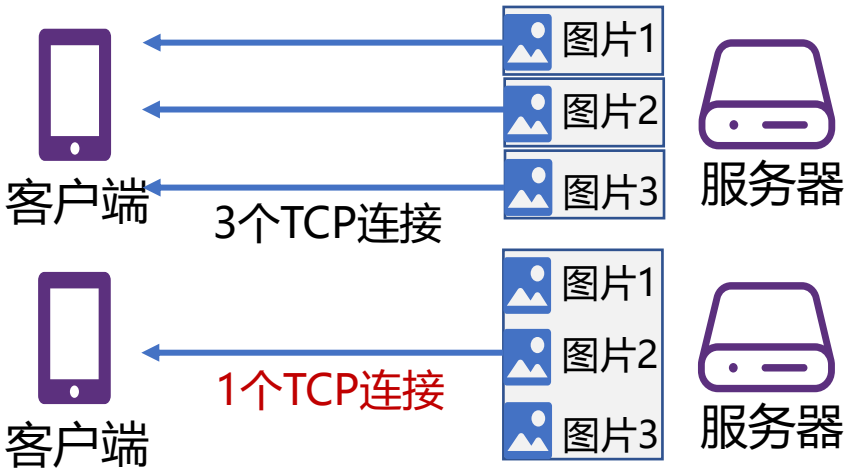


近年来加密流量占比迅速上升

TCP存在的问题

- TCP多流复用加剧了队头阻塞
  - 当前应用常需要同时传输多个元素
    - 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连接很低效
    - 因此出现了多流复用
  - TCP传输多流复用队头阻塞问题严重
    - TCP不区分多流，即多流之间存在队头阻塞
    - 任意一流被阻塞，会导致所有流都被阻塞

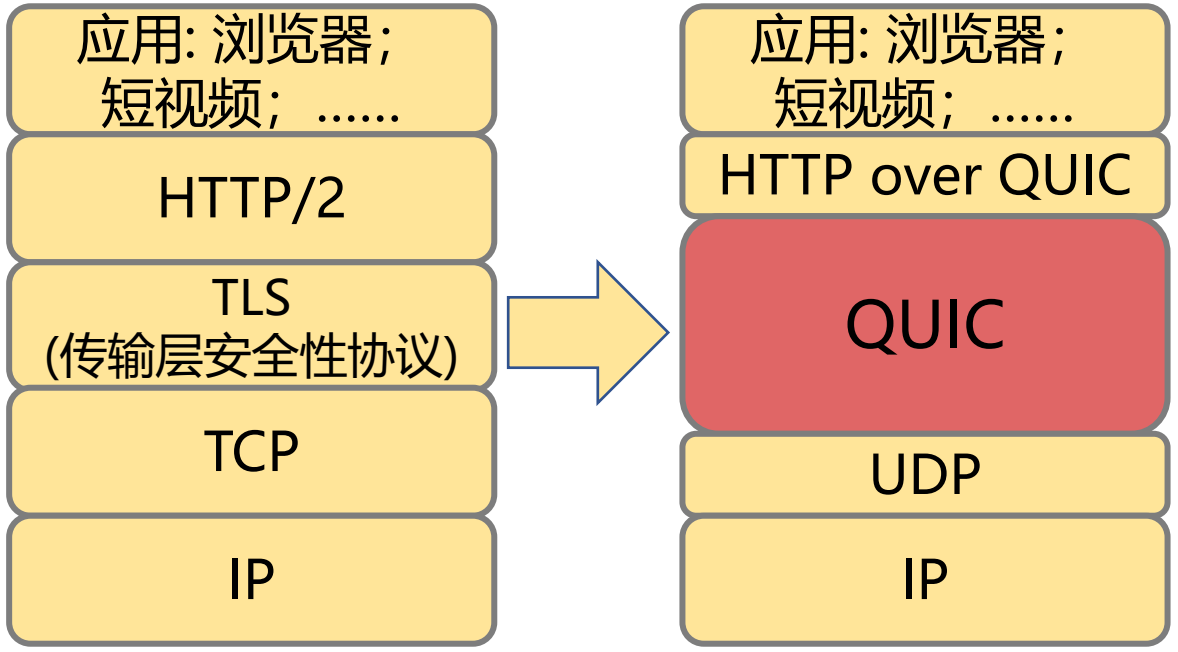
网页中的多个图片可以同时传输



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

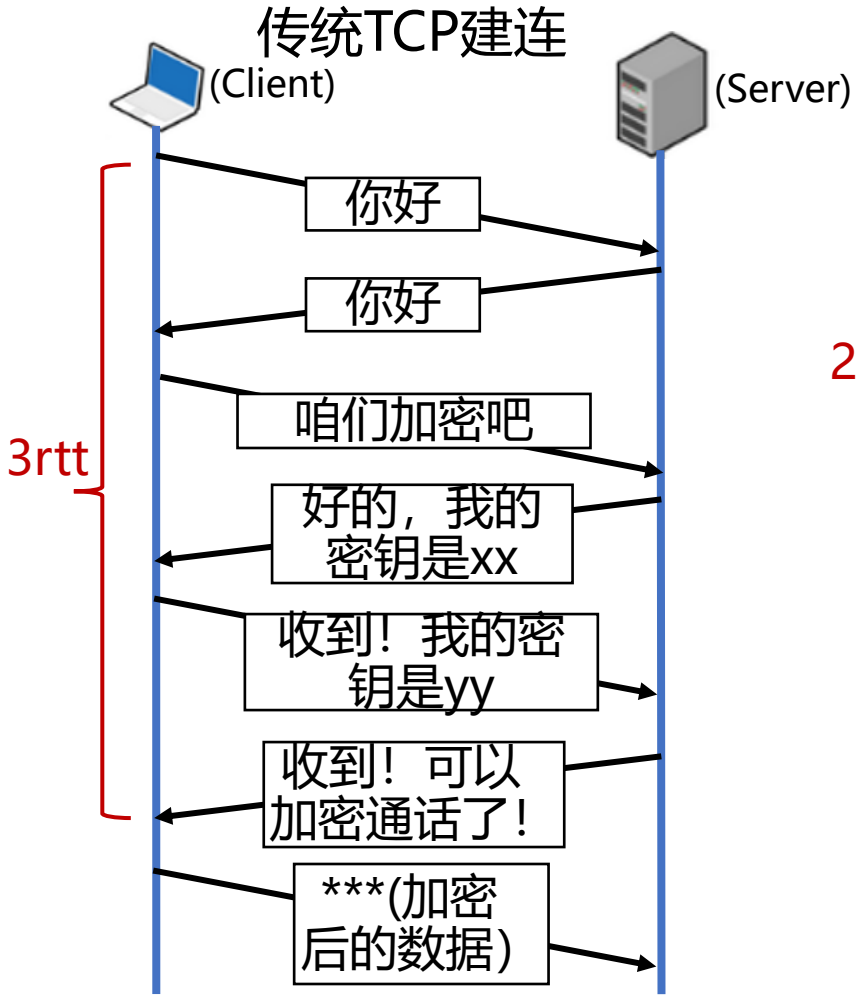
- 传统的传输架构
  - TCP提供数据传输服务
  - TLS（传输层安全性协议）对数据进行加密
  - HTTP协议定义如何发起请求-响应请求
  - 应用在HTTP之上实现
- 基于QUIC的传输架构
  - QUIC替代TCP、TLS和部分HTTP的功能
- QUIC实现在用户态中
  - 底层基于UDP实现
  - 拥塞控制是模块化的，可以方便地使用各种TCP拥塞控制算法，如Cubic等



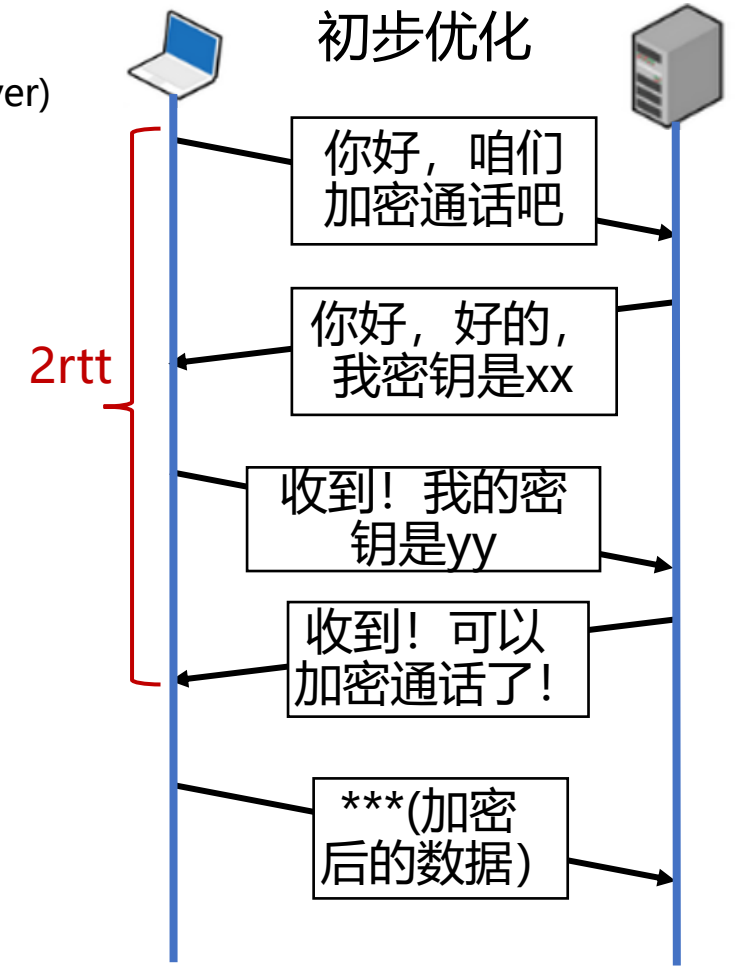
基于TCP的传输架构　　基于QUIC的传输架构

- 用户态实现
- 加密建连握手优化
- 精准RTT测量
- 移动切换连接管理
- 独立子流减轻队头阻塞

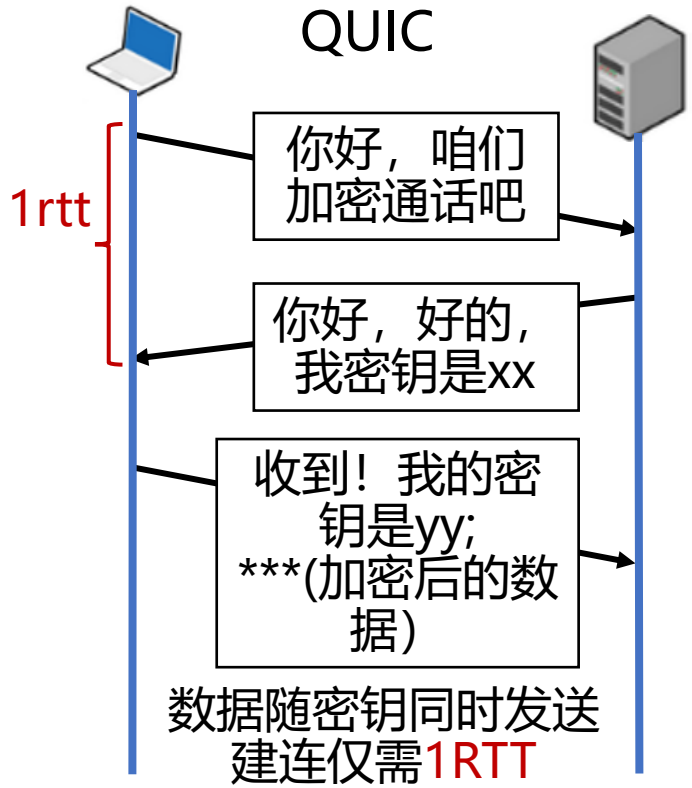
连接建立时延优化：简化演示



TCL建连、TLS建连串行进行
建连需要3RTT



传输建连与加密建连同时进行
建连需要2RTT



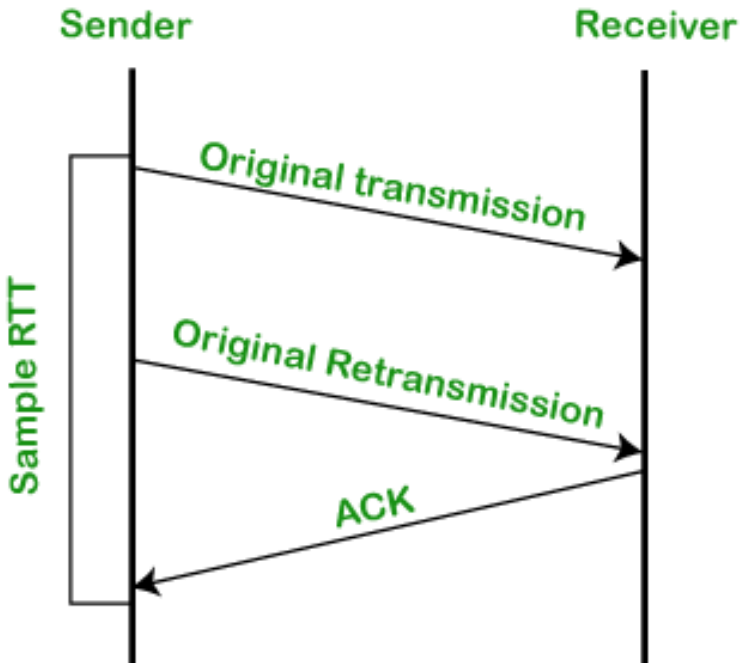
数据随密钥同时发送
建连仅需1RTT

- 用户态实现
- 加密建连握手优化
- 精准RTT测量
- 移动切换连接管理
- 独立子流减轻队头阻塞

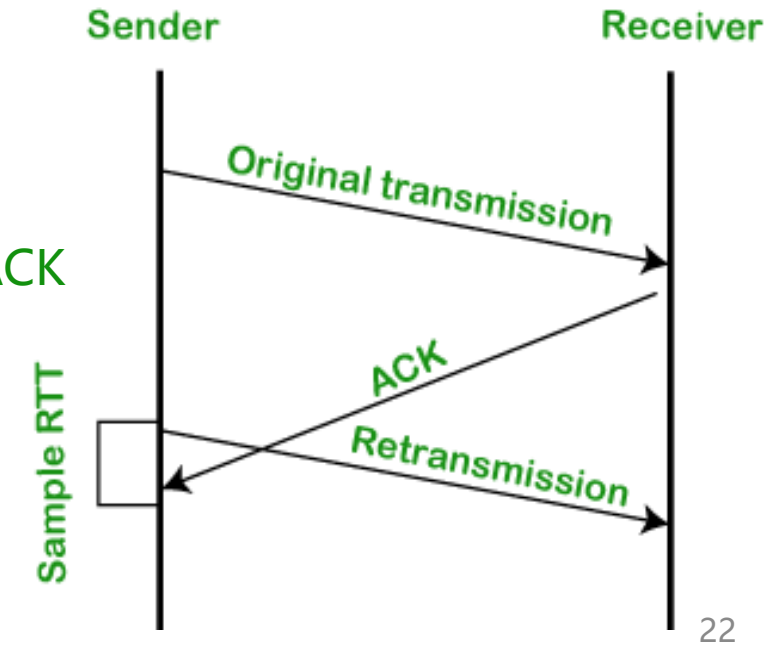
明确的包序号和更精确的RTT

- 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

可能性1：
是原包的ACK



可能性2：
是重传包的ACK



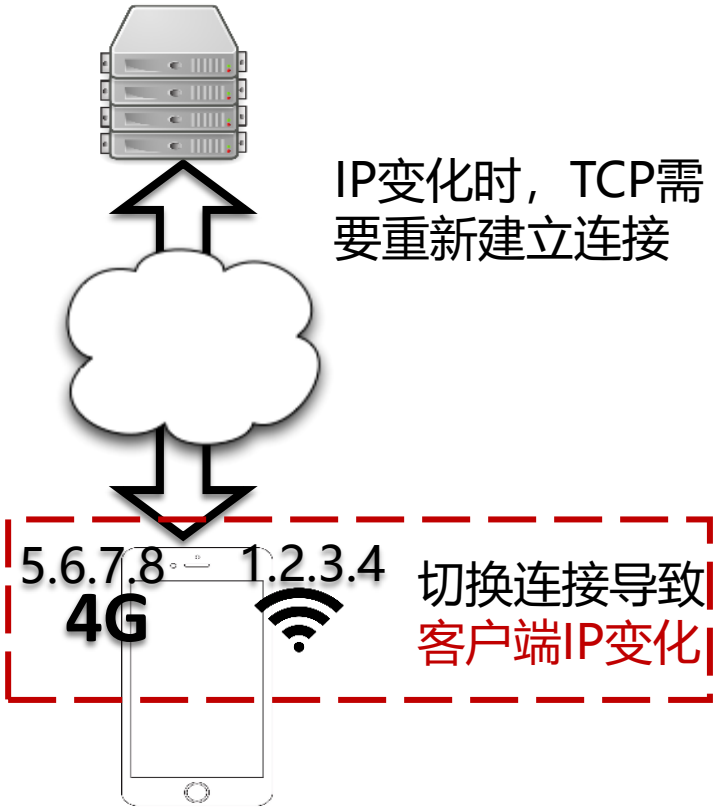
明确的包序号和更精确的RTT

- 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，并发馈给发送端，方便发送端更准确地测量RTT

- 用户态实现
- 加密建连握手优化
- 精准RTT测量
- 移动切换连接管理
- 独立子流减轻队头阻塞

IP地址/端口切换无需重新建立连接

- TCP连接基于IP地址/端口
  - 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



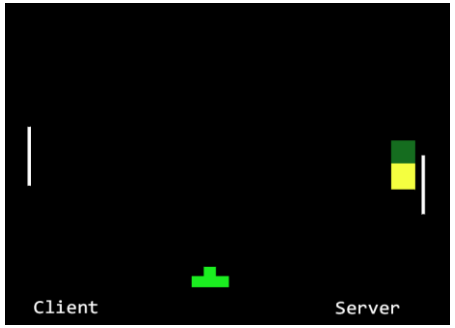
- 用户态实现
- 加密建连握手优化
- 精准RTT测量
- 移动切换连接管理
- 独立子流减轻队头阻塞

无队头阻塞的多流复用

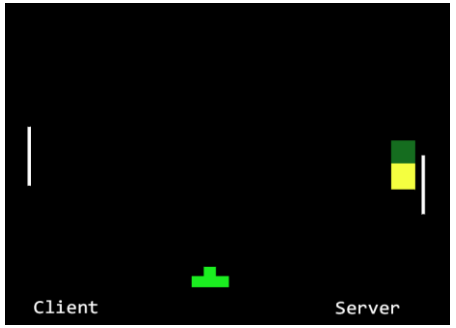
- 多流复用时的队头阻塞问题
  - TCP为保持数据的有序性，出现丢包时，会等待该数据到达后，再提交给上层应用
  - 多流复用时，某个数据流的数据包丢失，会使得TCP连接上所有数据流都需要等待
- QUIC对队头阻塞问题的解决
  - 在QUIC连接中，建立相互独立的多个子流，某子流数据包丢失不影响其它子流数据交付

- 用户态实现
- 加密建连握手优化
- 精准RTT测量
- 移动切换连接管理
- 独立子流减轻队头阻塞

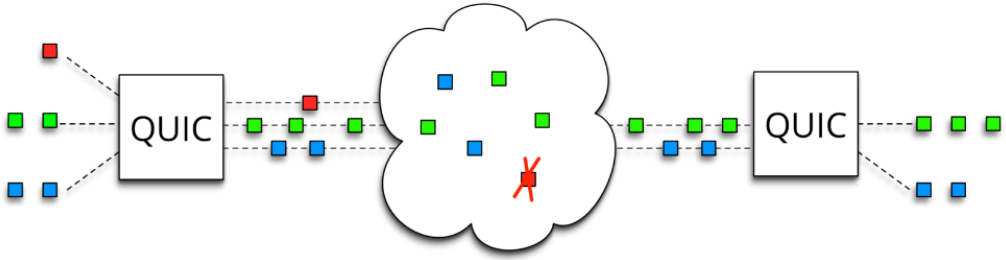
同时发送两个流，黄颜色的流上有丢包



TCP丢包会阻塞
所有后续的流



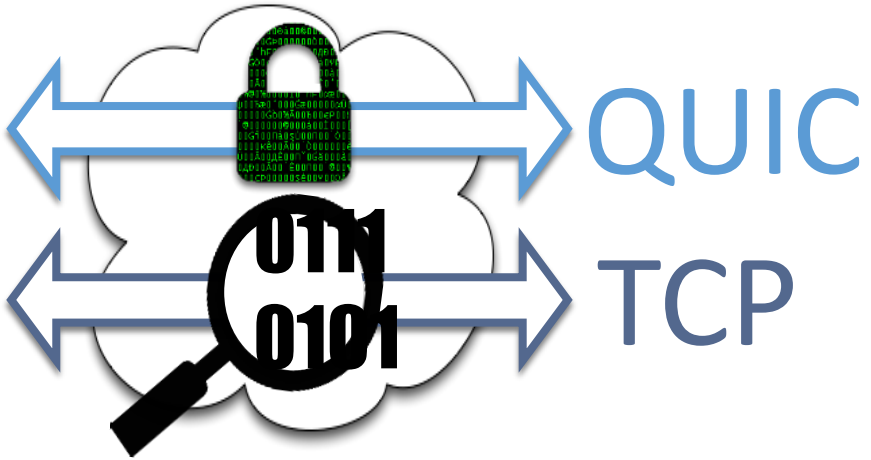
QUIC丢包只会
影响相关的流



红色流的包丢失，不会
阻塞绿色和蓝色的流

QUIC协议的其他优势

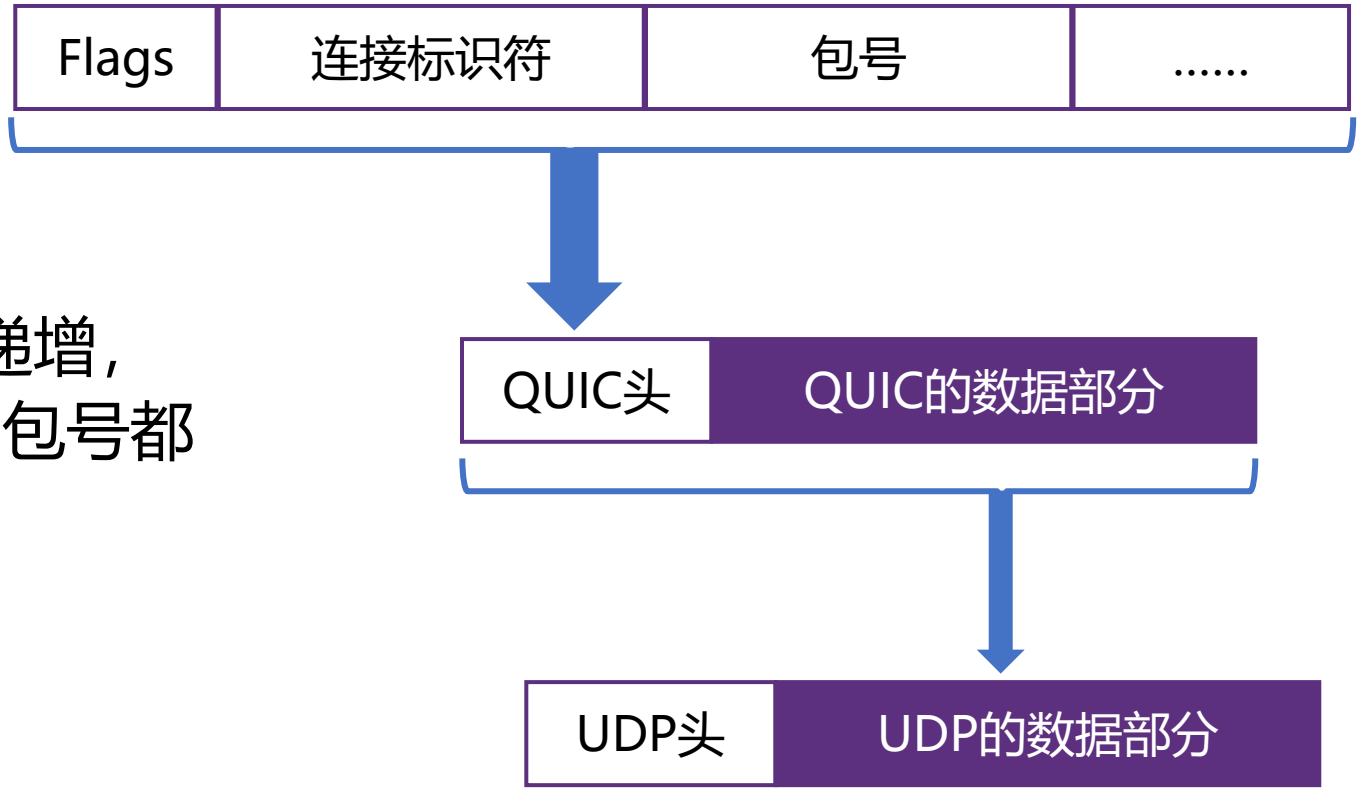
- 整个QUIC包被加密传输
  - 保护用户数据隐私
  - 避免被中间设备识别和修改
- QUIC在用户态实现
  - 与操作系统解耦，从而能和应用一同快速迭代
- 版本协商机制易于更新迭代
  - 由于QUIC的快速迭代特性，会同时存在众多QUIC版本
  - 客户需要和服务器进行版本协商（不引入额外时延的协商机制）



中间设备无法识别和修改
QUIC header中的信息

QUIC包格式介绍（简化）

- 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



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（谷歌&苹果）
- 可路由的Connection ID
- 截止时间可感知的QUIC扩展（呼唤愿为浪潮之巅者）
- ……

传输协议QUIC-小结

- 为什么需要QUIC：TCP存在的问题
  - RTO队头阻塞问题，多流复用加剧
  - TCP握手过程引入很大的时延
  - 移动用户底层短连接，难以支撑上层业务的长连接
  - 应用优化TCP的难度高
- QUIC协议
  - 基于UDP在用户态实现
  - 优化了TLS加密建连的握手过程
  - 包序号持续递增，无重传歧义问题，实现精准RTT测量
  - 以Connection ID识别连接，即使IP地址/端口发生变化也无需重连
  - 各个流的传输相互独立，消除了队头阻塞问题

本节内容

7.6 拥塞控制的发展

7.7 新型传输层协议QUIC

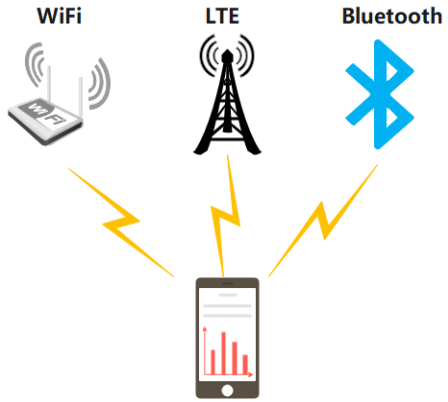
7.8 多路径传输协议MPTCP

7.9 数据中心网络传输协议

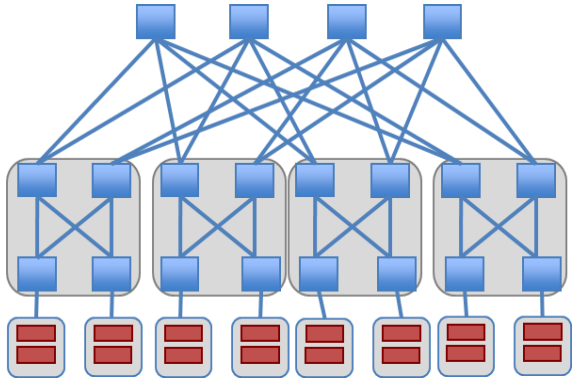
- 多径传输的场景分析和目标
- MPTCP的设计思路
- MPTCP的应用现状

现代设备常有多个网络接口

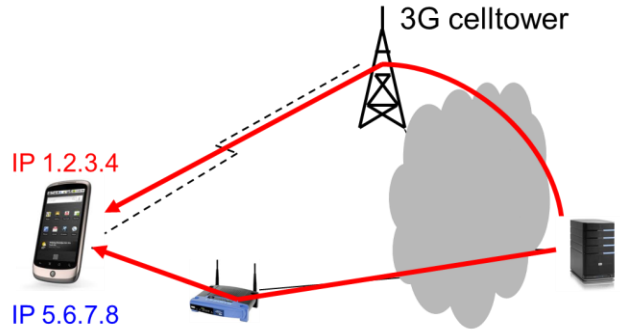
- 传统TCP协议仅支持单路径传输，即只能利用终端主机上的一个网络接口传输数据
- 随着接入技术的发展，同时具备多个网络接口的网络设备已经越来越普及，多路径传输更适合当前的网络环境



移动网络



数据中心网络

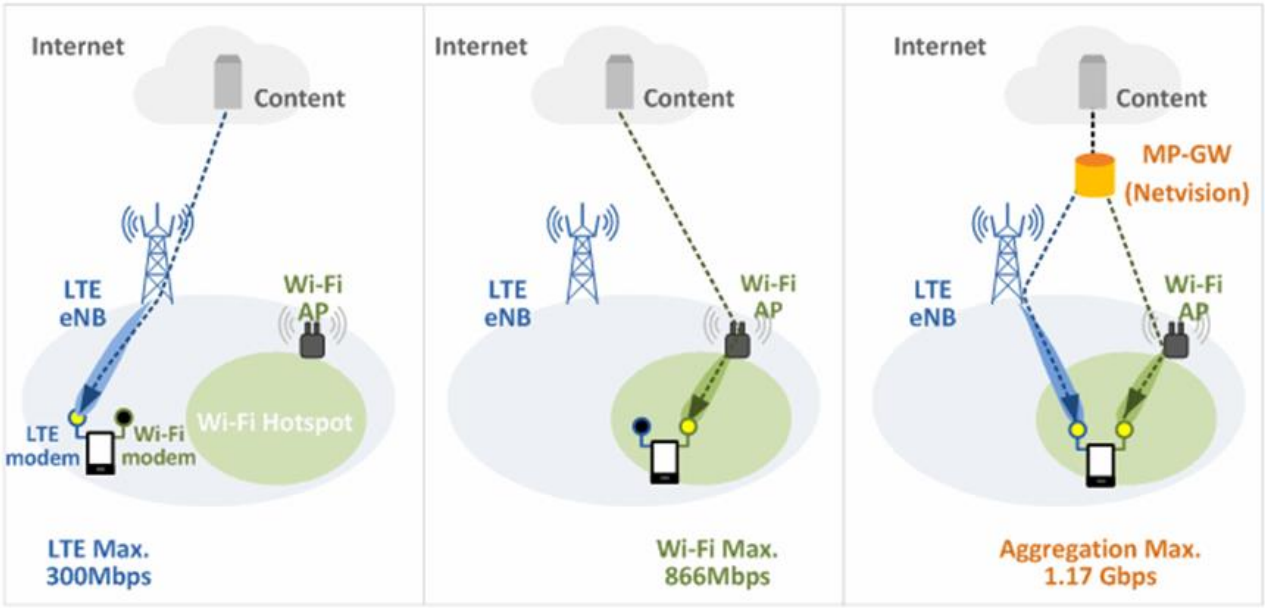


多路径传输示意图

- 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的设计目标

- 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

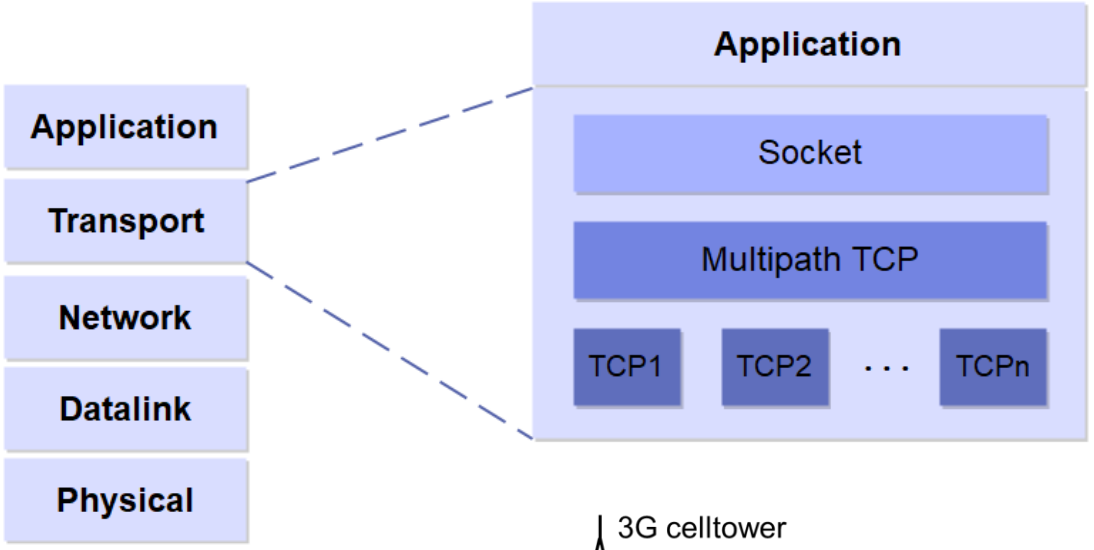


设计思路：应用是否需要修改?

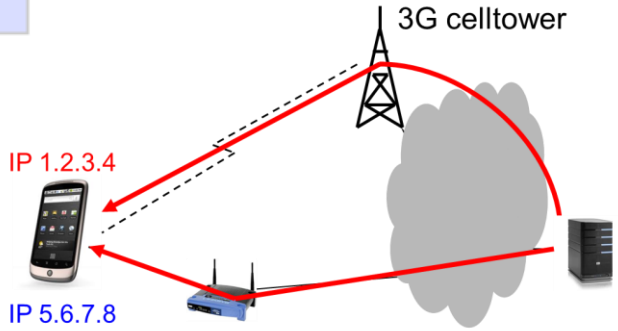
图片来源：https://www.cnblogs.com/yhp-smarthome/p/7440855.html

MPTCP在网络体系结构中的位置

- 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
  - 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

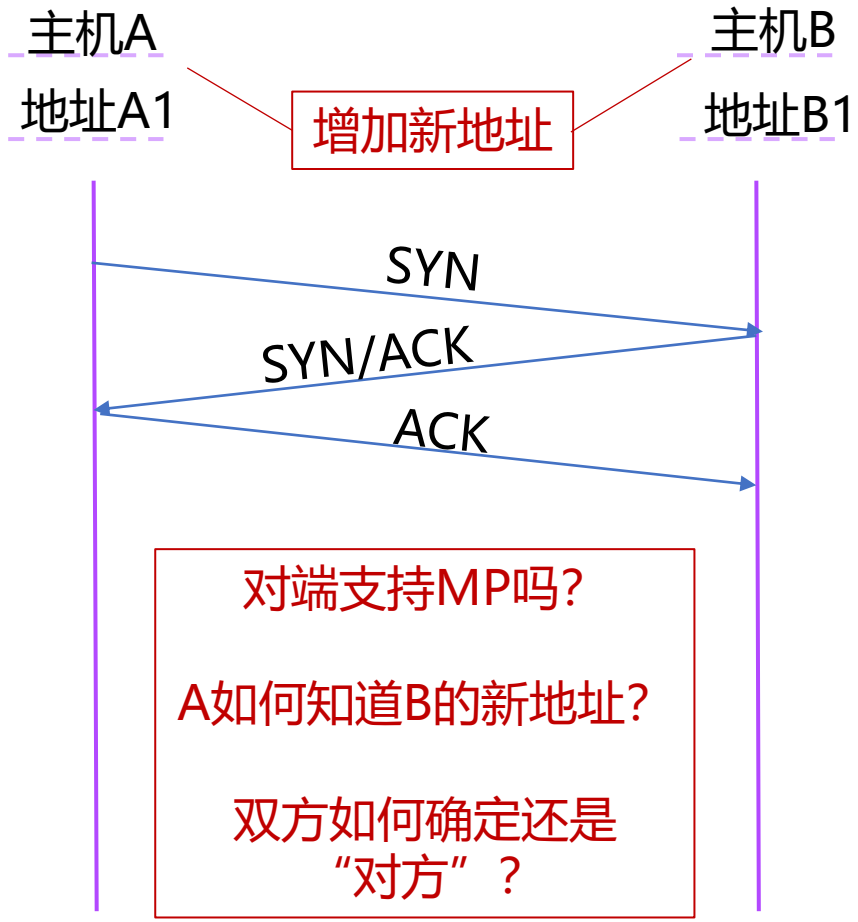


- MPTCP连接是一个或多个子流的集合
  - 路径：主机之间的路径使用四元组表示
    <本地IP地址，本地端口，远程IP地址，远程端口>
  - 子流：在单个路径上运行的 TCP 流称为子流，是MPTCP连接的组成部分



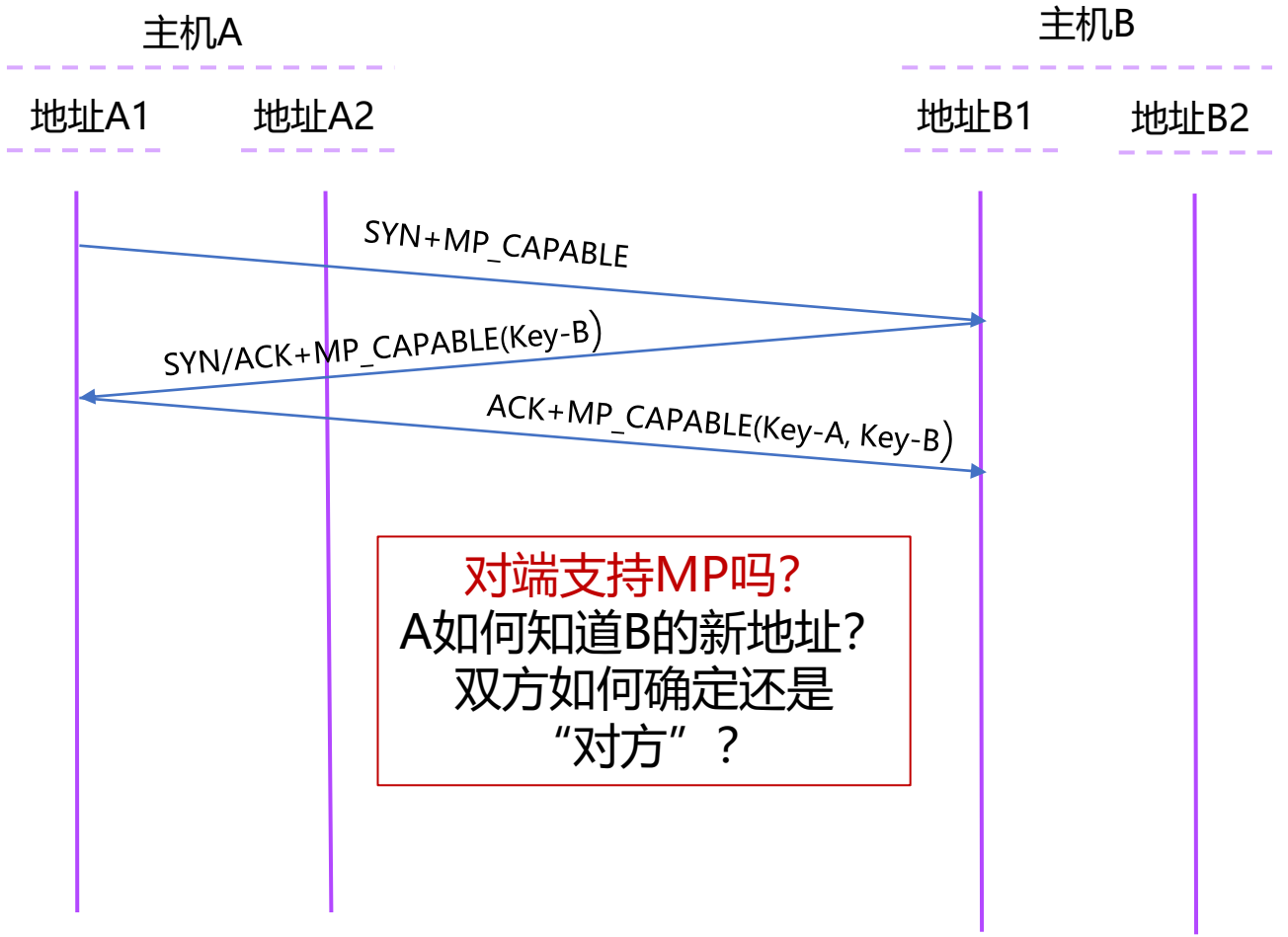
MPTCP连接管理

- 如何建立MPTCP连接
  - 通信双方增加新地址啦！？
  - 初始化一条MPTCP连接（与建立常规TCP连接的过程相似）
  - 将其它子流/子路径附加到已经存在的MPTCP连接上
- 用于MPTCP连接管理的新字段
  - MP_CAPABLE：建立MPTCP连接
  - ADD_ADDR：新增可用路径
  - REMOVE_ADDR：删除路径
  - MP_FASTCLOSE：关闭所有子流
  - MP_JOIN：附加新的子流到已有连接



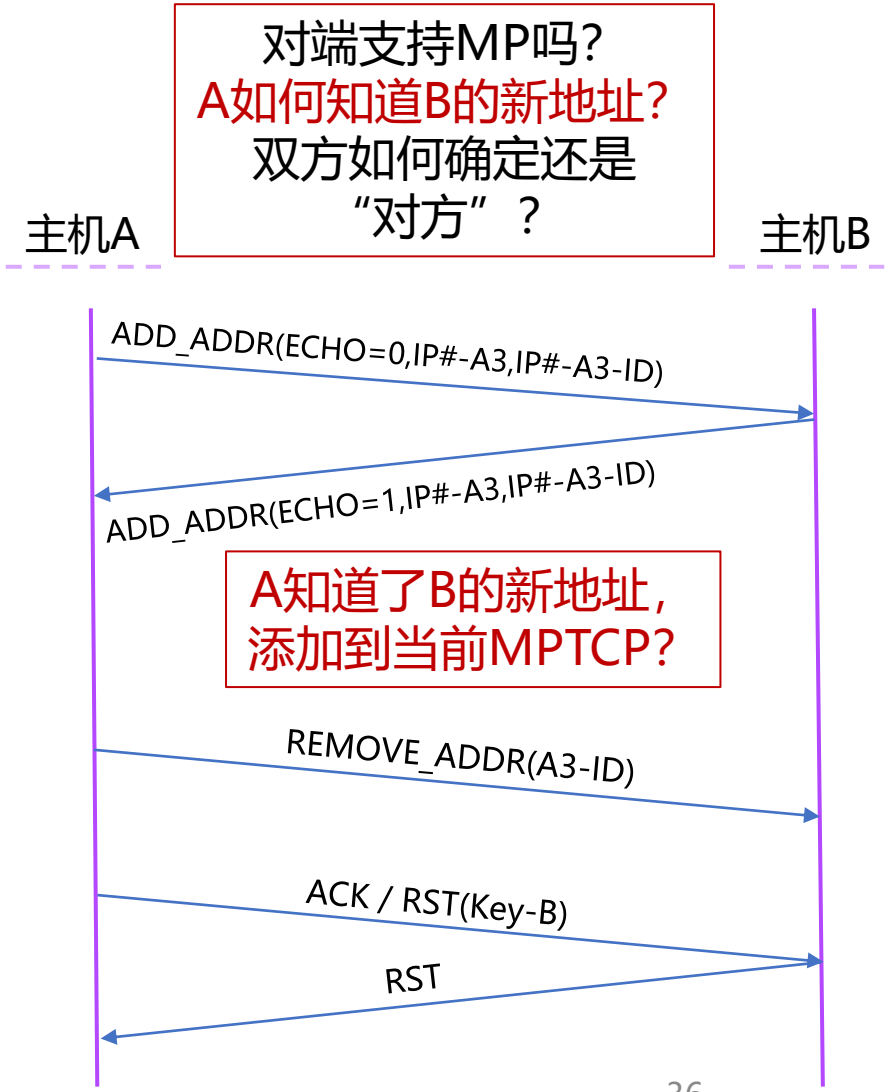
初始化MPTCP连接

- 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_CAPBLE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_CAPBLE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_CAPBLE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



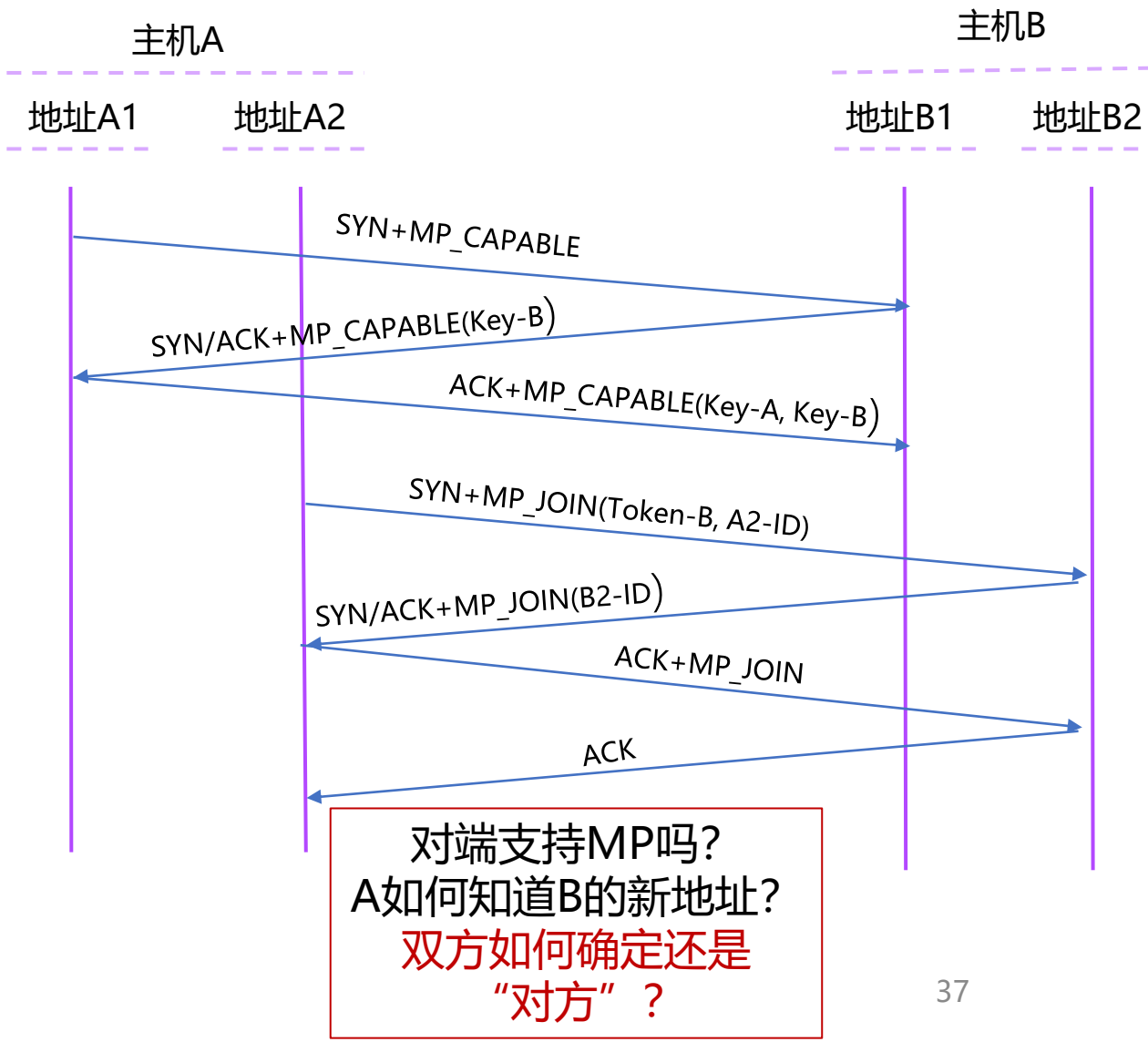
MPTCP路径管理和关闭连接

- 新增路径
  - 使用ADD_ADDR字段：新增路径（并不启用新的子流）
  - 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)，ECHO指示当前报文段是“发出”
  - 主机B验证无误后记录路径信息，向主机A发送ECHO响应报文段
- 删除路径
  - 主机A发送REMOVE_ADDR报文，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



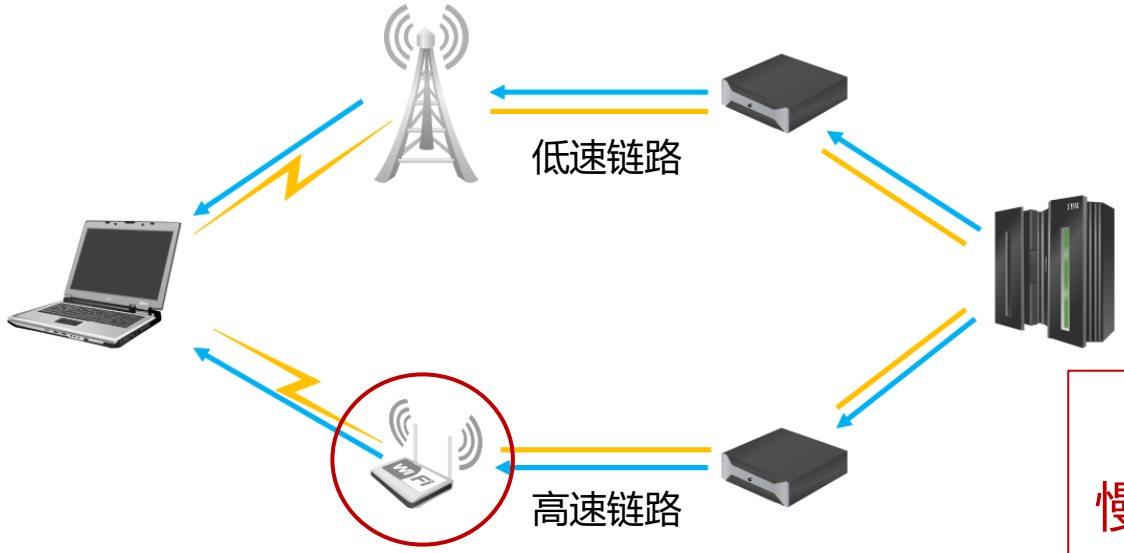
附加子流到MPTCP连接上

- 启用新的子流（A2->B2）
  - 使用MP_JOIN，附加到已存在的MPTCP连接
  - 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_CAPBLE字段交换的密钥等信息，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验证无误后向主机B发送ACK报文段，并附上自己的认证信息
  - 主机B验证无误后向主机A发送ACK报文段，至此子流建立完毕



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



应用层数据包

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

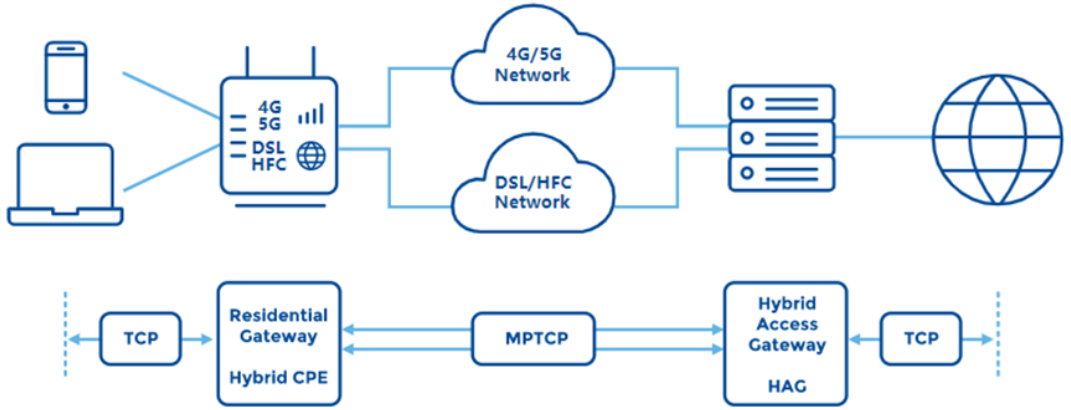
五星博士

慢路径造成的快路径突发，导致交换机丢包?

- 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/）
- 手机的广泛应用
  - iPhone和iPad上的Multipath TCP（https://support.apple.com/zh-cn/HT201373）
  - 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



多路径传输协议MPTCP-小结

- 动机：现代网络设备通常有多个接口
- 优势：带宽聚合、可靠性、平滑切换链路
- MPTCP协议
  - 基于TCP实现，单个路径上运行一个TCP流
  - 未改变TCP头的基础定义，仅添加新的选项
  - 连接过程：初始化、附加子流、路径管理、关闭连接
- MPTCP的数据调度
  - 将属于同一个数据流的数据包调度到不同的路径上传输
  - 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
  - 目标：吞吐量、公平性、均衡拥塞

进一步研究

如何避免多径耦合造成的速率下降

高带宽还是低时延?

FMTCP: 基于喷泉编码的MPTCP

本节内容

7.6 因特网传输协议TCP

7.7 新型传输层协议QUIC

7.8 多路径传输协议MPTCP

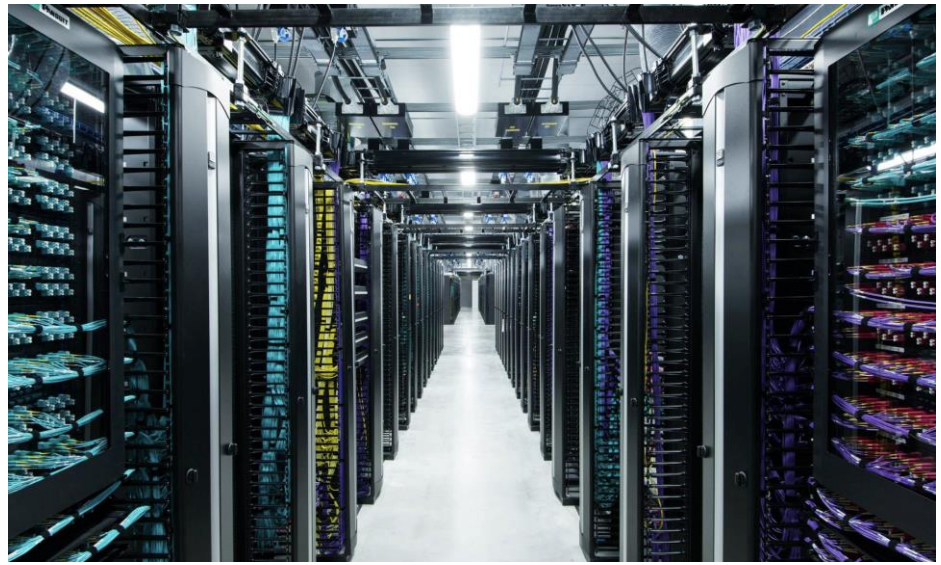
7.9 数据中心网络传输协议

- 数据中心网络及其特点
- DCTCP核心思想
- RDMA数据中心简介

数据中心网络

一台服务器

机柜: 十几台服务器

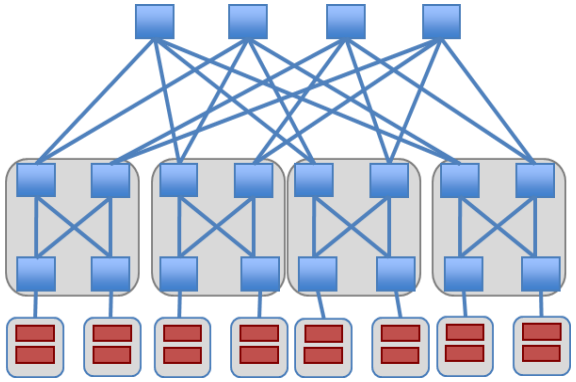


数据中心: 几千台服务器

成千上万台服务器之间如何互联?

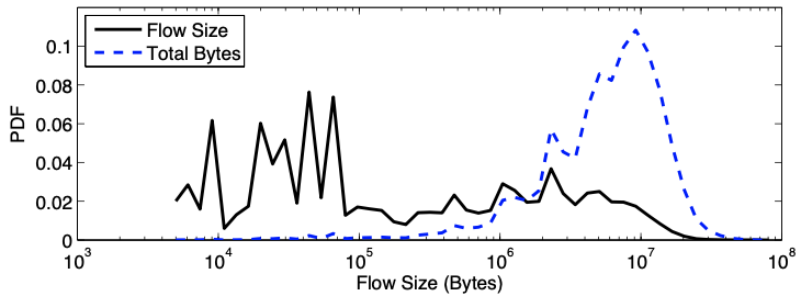
数据中心网络的特点

网络拓扑



- 高度对称的拓扑结构
- 高带宽: 40G/100G/400G
- 低时延: ~us

流量特点



- 99%+的流量基于TCP连接
- 多种流量混合
- 查询流量(2KB-20KB)
- 时延敏感的短流(100KB-1MB)
- 吞吐敏感的长流(1MB-100MB)

细致研究

带宽/时延

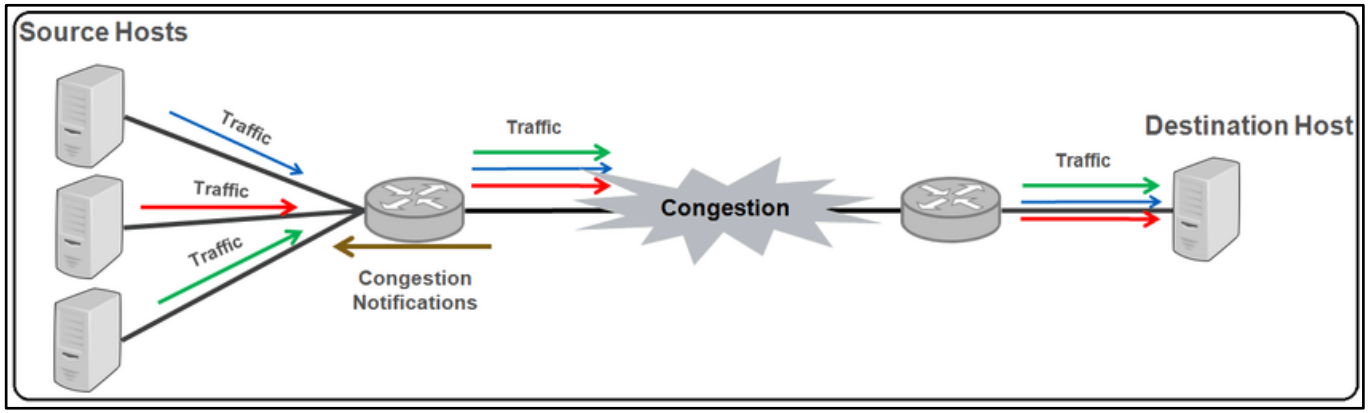
v.s.

流完成时间FCT

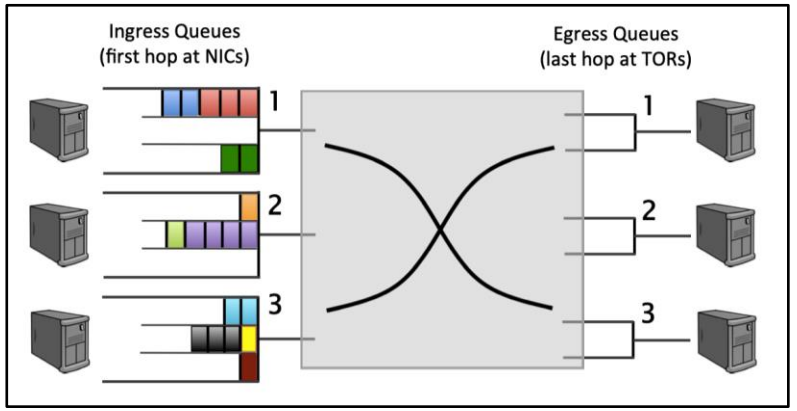
大象流/长流关心什么?

老鼠流/短流关心什么?

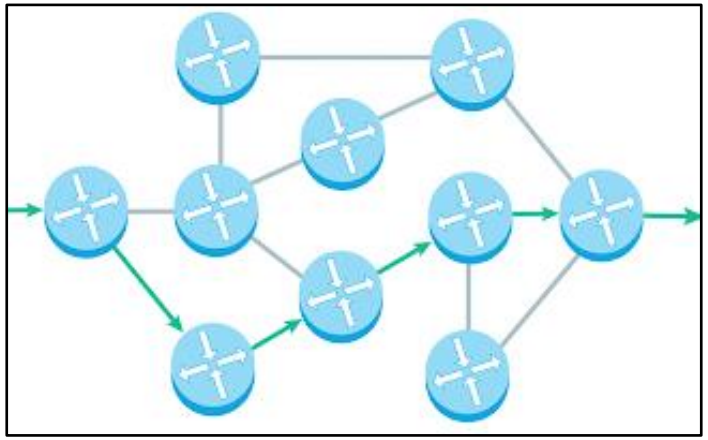
数据中心网络的关键问题



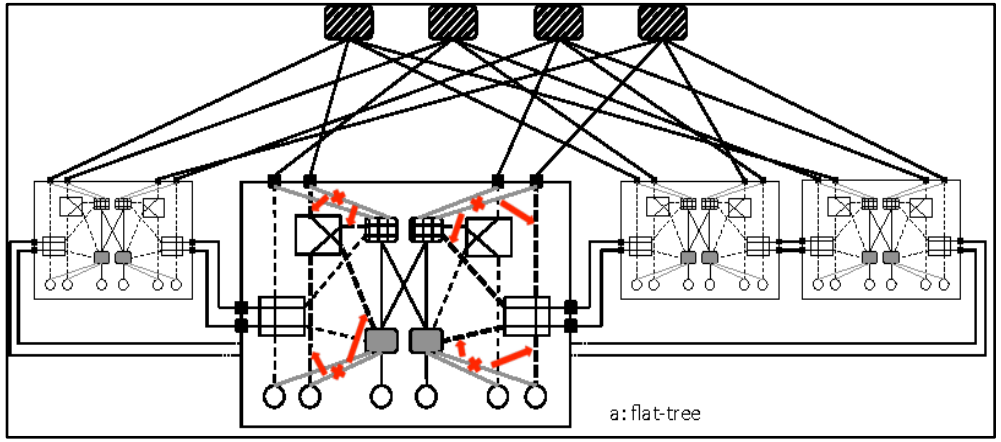
拥塞控制



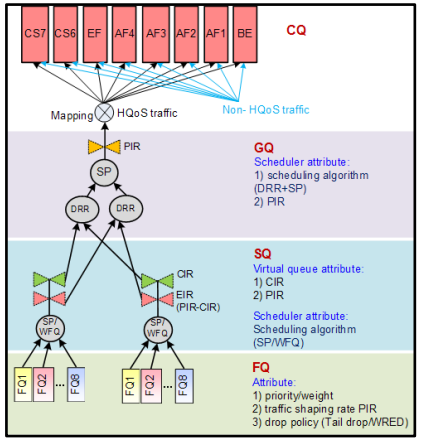
流量调度



路由策略与负载均衡

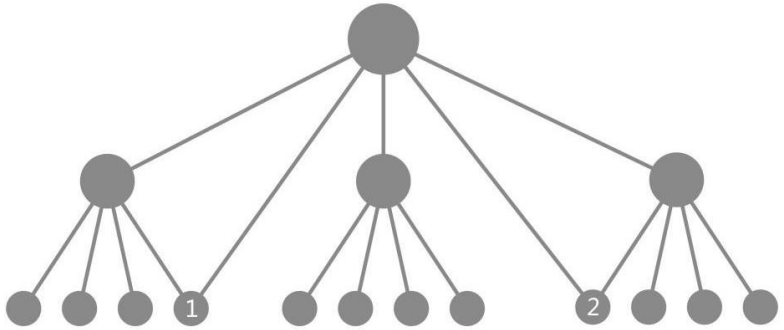


拓扑架构

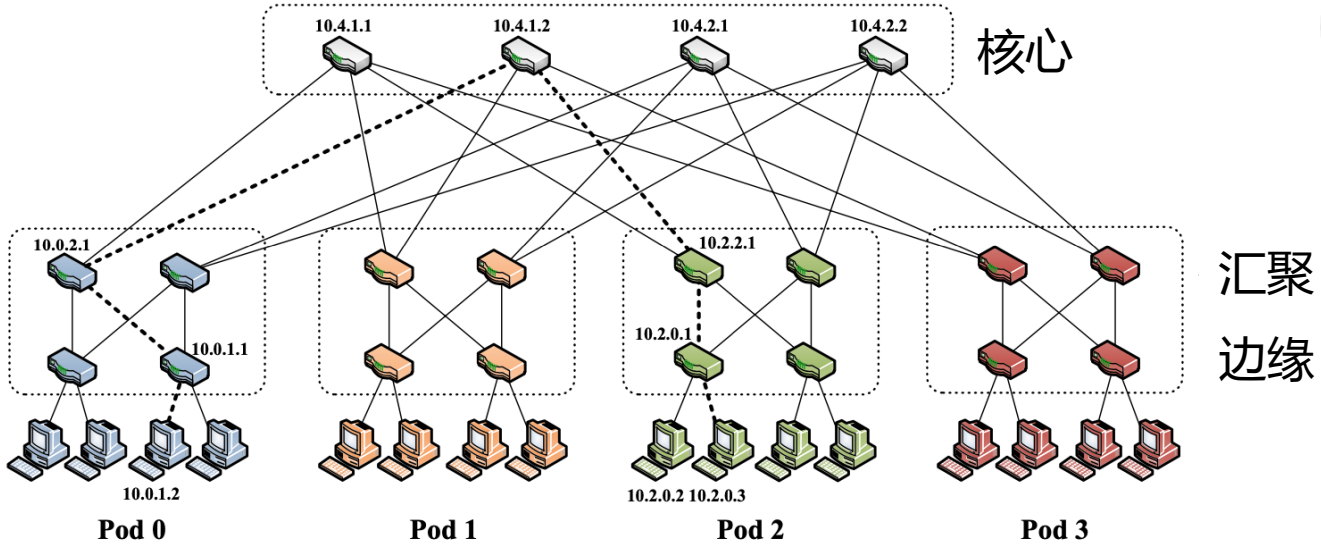


交换机参数配置

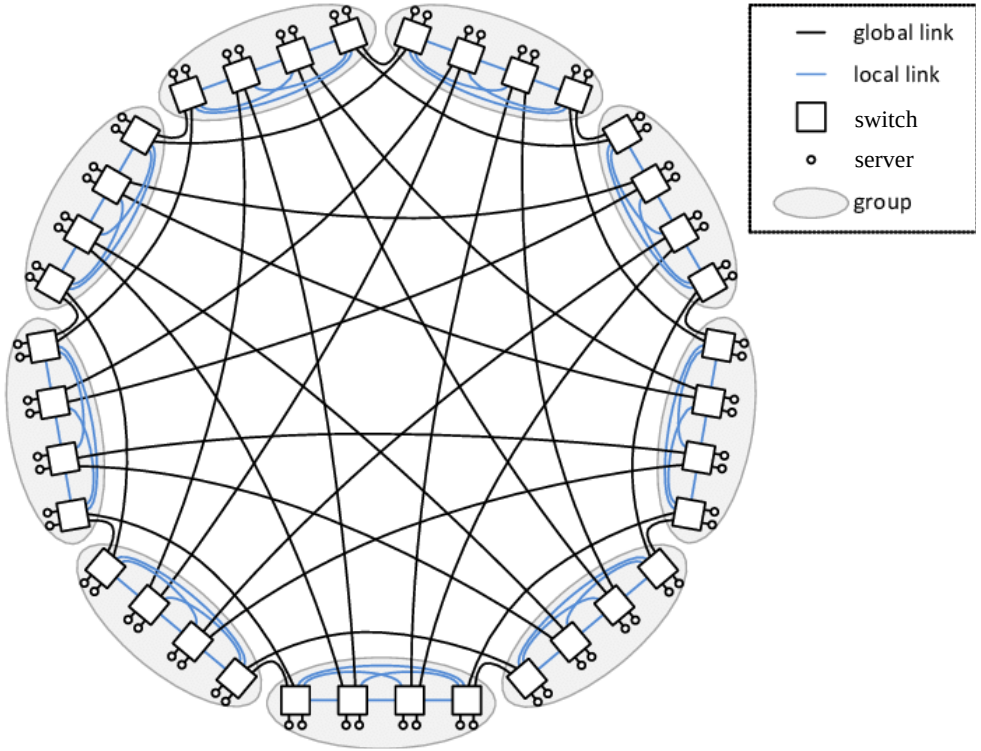
数据中心网络拓扑发展



简单树形结构



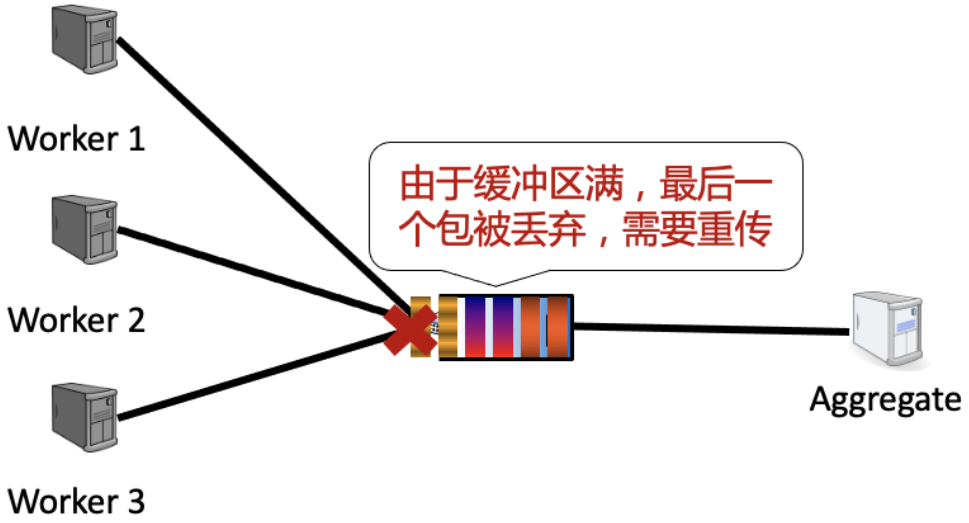
胖树(Fat Tree)结构



Dragonfly结构

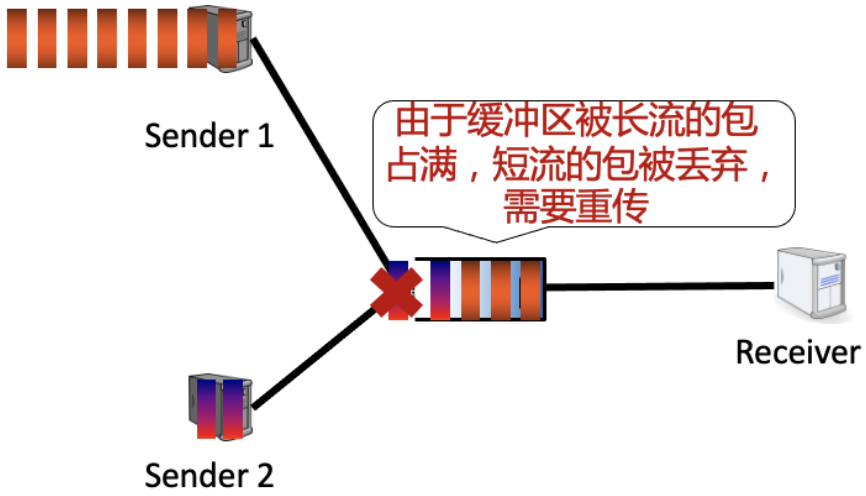
数据中心网络的性能瓶颈

1. 突发流量(Incast)



- 网页搜索等“分发/聚合”模式造成
- 多个发送端同时向同一个接收端发送数据
- 端口到达速率远大于传输速率

2. 队列累积和缓存不足



- 数据中心交换机缓存芯片成本高，可用缓存空间受限，需要端口间统计复用
- 大象流/长流占用缓冲区空间
- 老鼠流/短流排队时延增大，甚至丢包

宝贵的缓冲区留给谁好?

数据中心传输协议设计原则

- 容忍高突发流量
  - “分发/聚合”模式中，大量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
  - 需要能够避免突发流量丢包的传输协议

避免丢包
坚决避免RTO

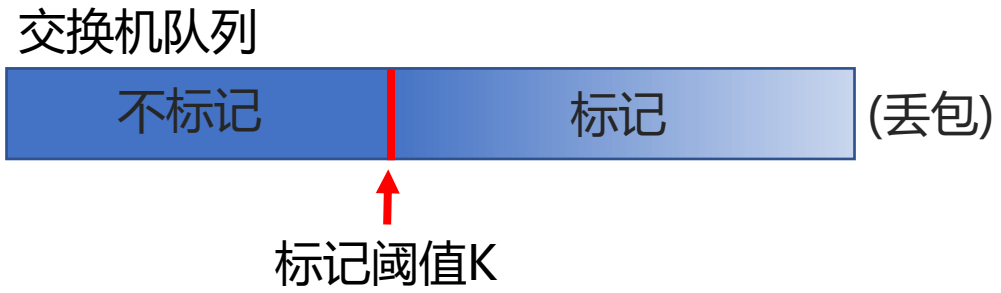
- 低时延
  - 数据中心有大量时延敏感的短流，如网页搜索等
  - 需要能够将排队时延降到最低的传输协议

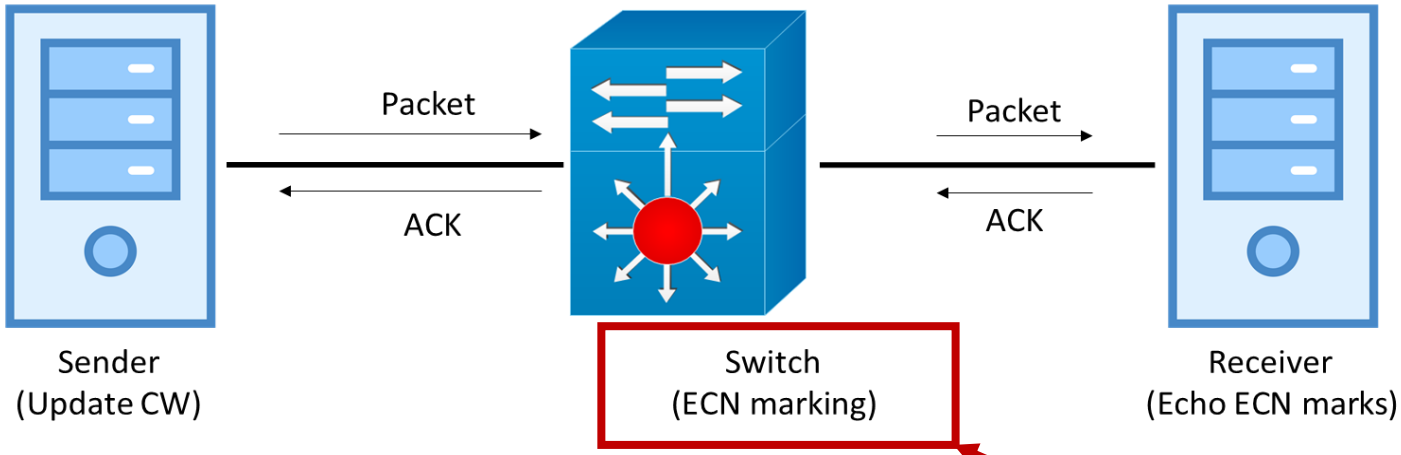
现有TCP
能够满足吗?

- 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
  - 需要能够维持高带宽利用率的传输协议

DC(Data Center)TCP核心思想

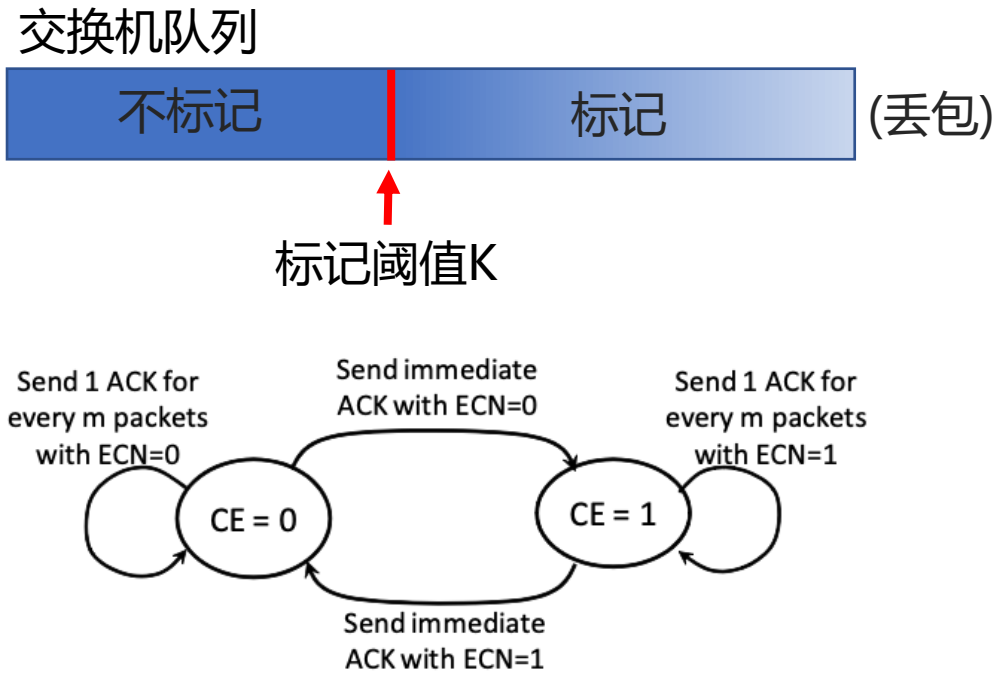
- 根据交换机队列的瞬时长度标记ECN (Explicit Congestion Notification)
  - 使用显式的拥塞反馈能够更好地控制突发流量
- 根据拥塞程度精细调整发送窗口
  - TCP: $cwnd \leftarrow cwnd/2$
  - DCTCP: $cwnd \leftarrow cwnd \times (1 - \alpha/2)$
  - α: 使用显示拥塞信号得到的“拥塞程度”





DCTCP协议实现

- 交换机标记
  - 当队列长度超过K时，给之后的包标记ECN
  - 队列长度瞬息万变，并非所有包都被标记
- 接收端ACK协同
  - 当被标记的包首次出现或不再出现时，立即发送ACK
  - 否则采取Delay ACK的策略
- 发送端控速
  - 每个RTT更新一次发送窗口
  - α为基于被标记的包比例的平滑因子
  - $cwnd \leftarrow cwnd \times (1 - \alpha/2)$

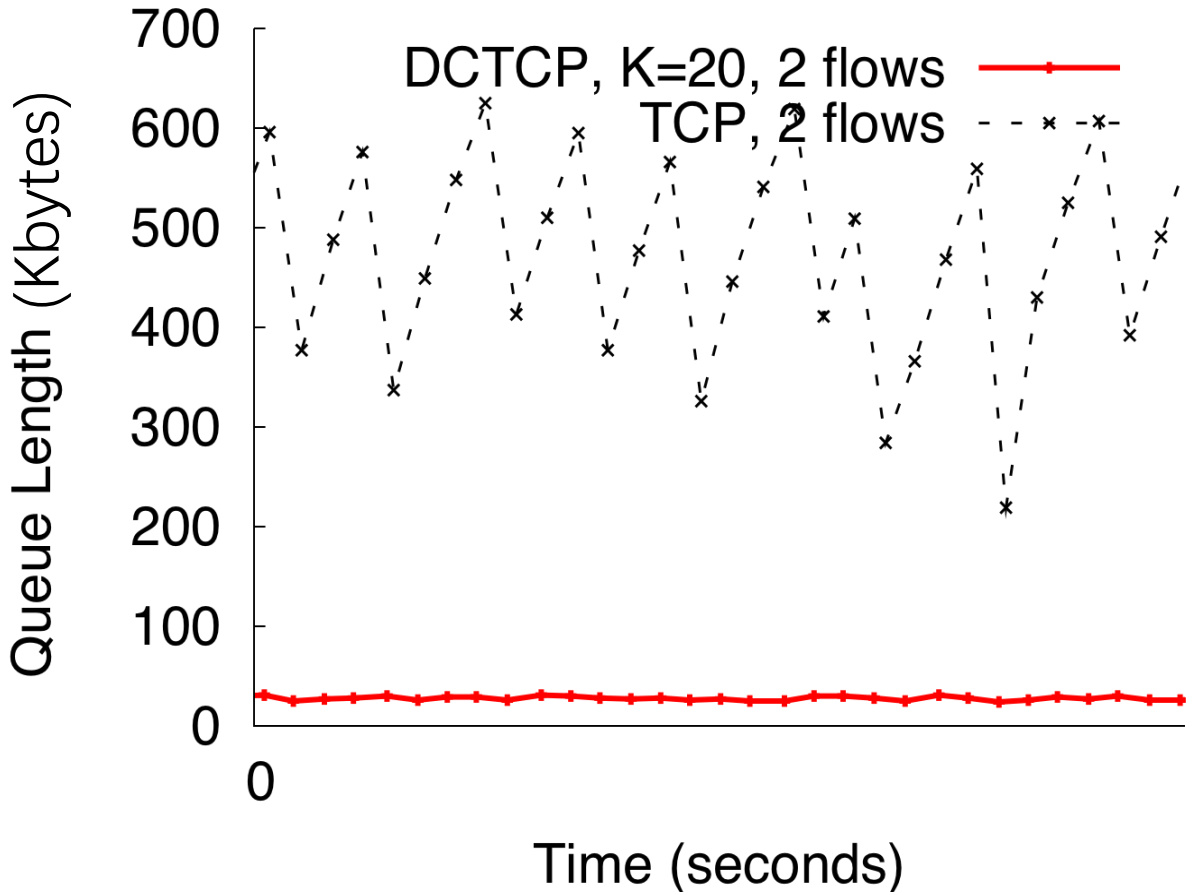


提早判断可能的拥塞
避免大流占用缓冲区

DCTCP与TCP的比较

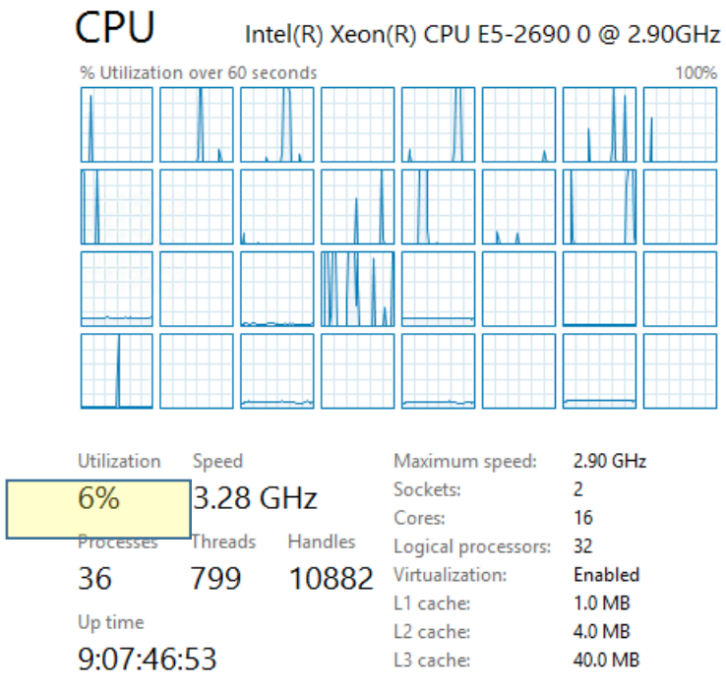
- 交换机中的队列长度分析
  - DCTCP能将队列长度稳定地维持在低水平
  - 基于ECN的帮助
  - TCP的队列长度不仅高，而且波动很大
- DCTCP适用于高带宽低时延的数据中心网络

数据中心网络
端网协同的典范



DCTCP的问题

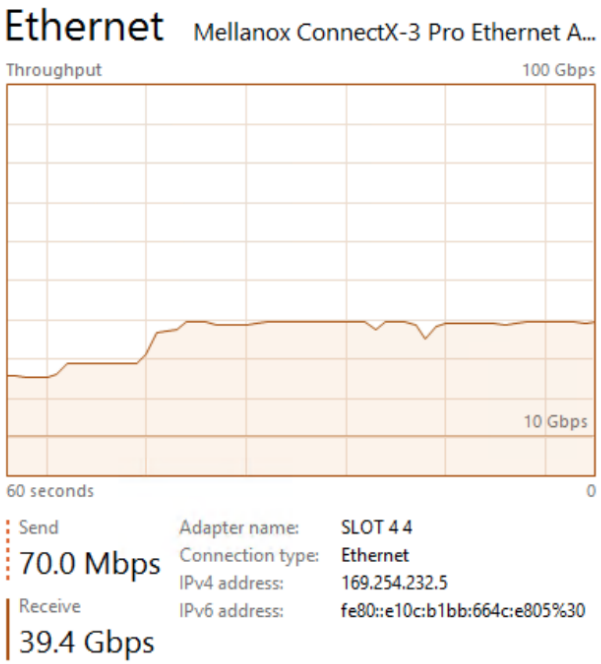
Sender



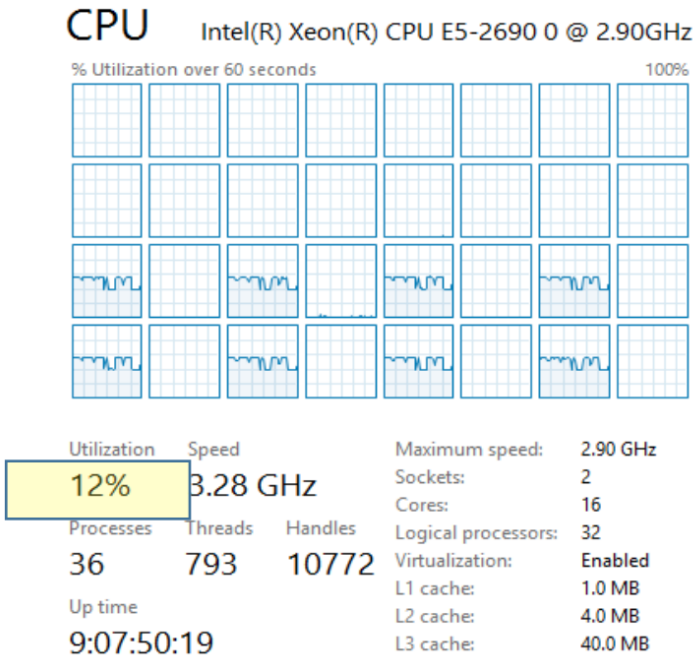
网络通信的CPU开销

8 tcp connections

40G NIC



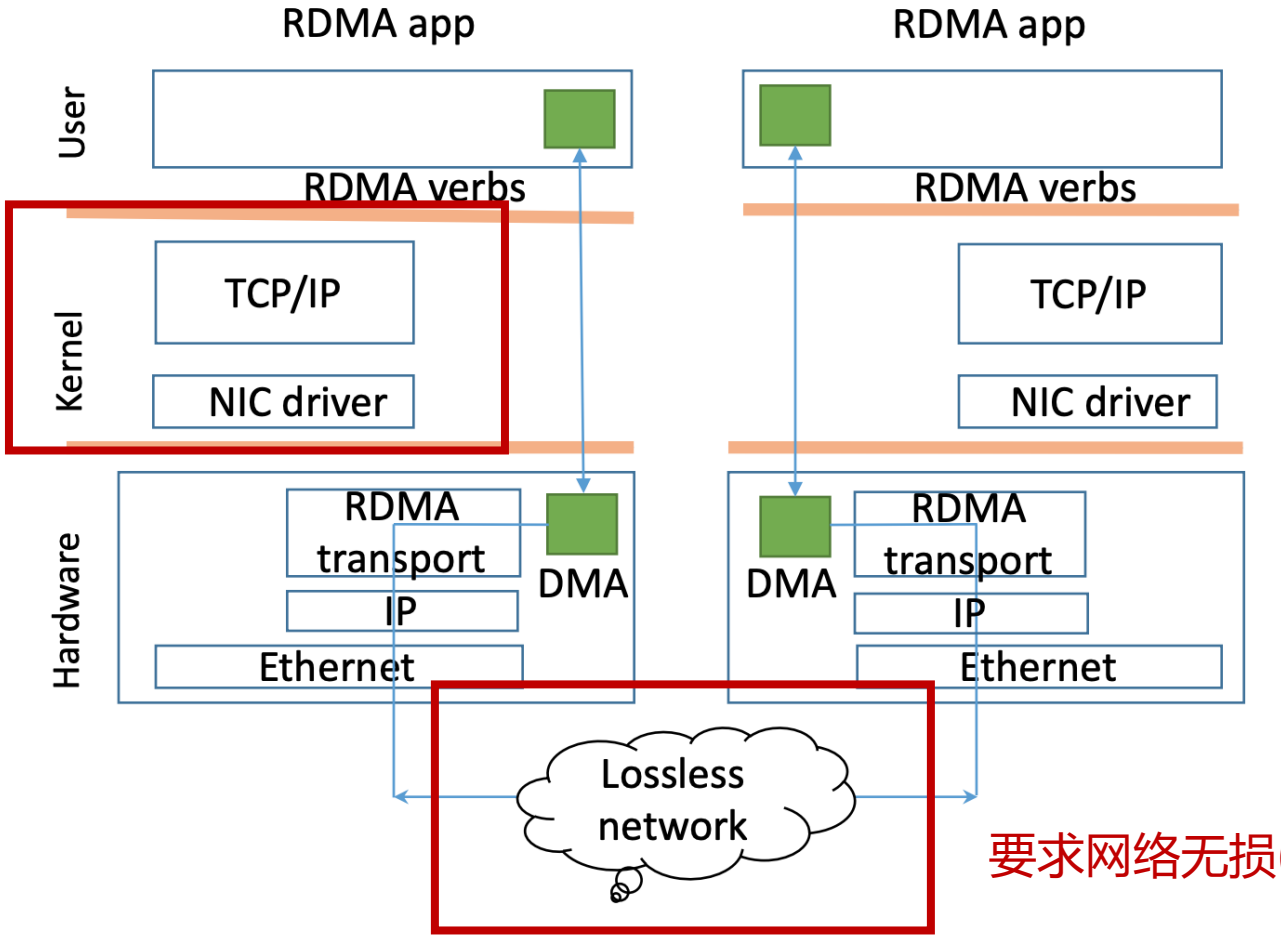
Receiver



RDMA数据中心简介

RDMA: Remote Direct Memory Access，远程直接内存访问

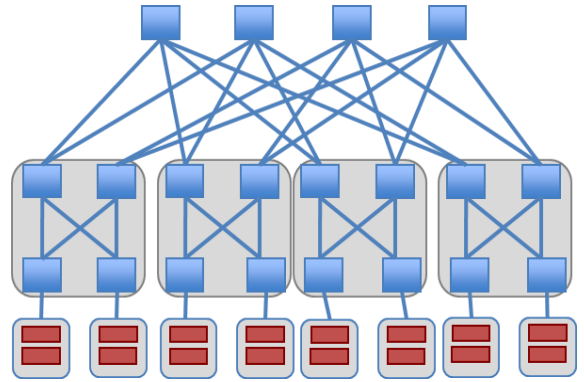
Bypass Kernel
即数据传输不经过
Linux TCP/IP协议栈



要求网络无损(不丢包)

数据中心网络小结

- 数据中心网络: 将几千台服务器相互连接的网络
- 数据中心网络的特点:
  - 特殊拓扑结构: 高对称(e.g., 胖树(FatTree)拓扑)
  - 业务需求: (极)高带宽、(极)低时延
  - 流量特点: 大小流混合、高突发
- 数据中心拥塞控制DCTCP
  - 使用ECN更精确的感知网络拥塞
  - 根据网络拥塞程度精确调整发送窗口
- 数据中心拥塞控制的开销问题
  - RDMA技术绕过Linux内核发送数据



研究方向

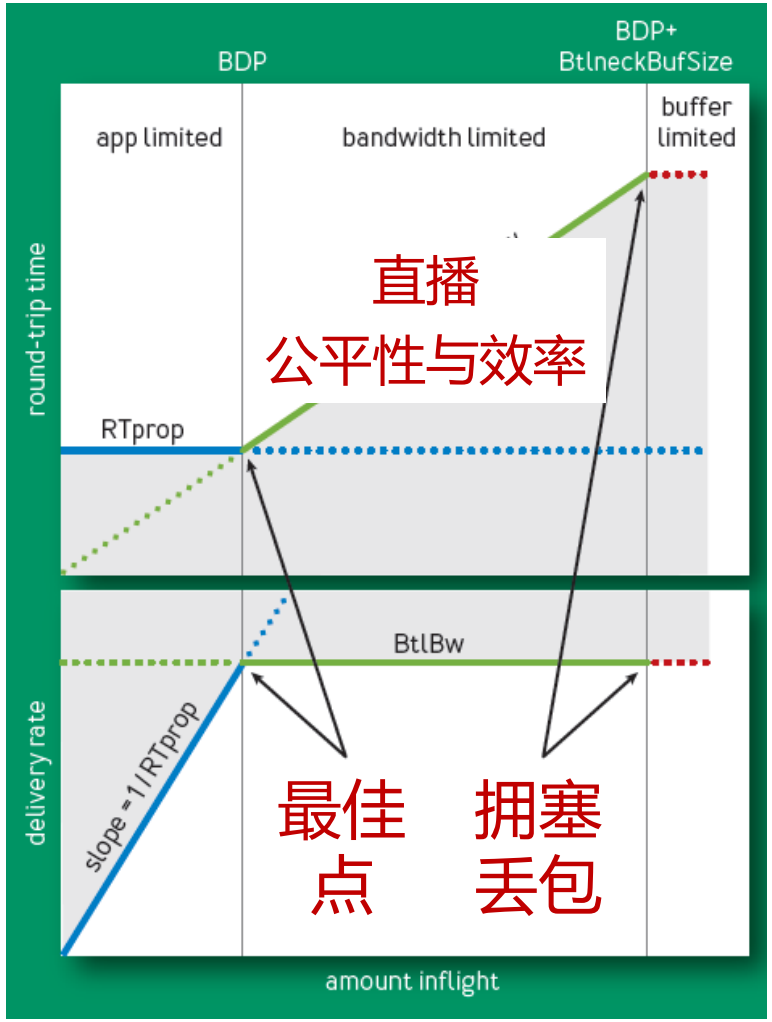
流量对称吗?

流量模型?

分布式机器学习

传输层进阶：总结

- 拥塞控制算法的发展
  - Cubic：更合理的窗口增加方式
  - BBR：测量瓶颈带宽而忽略丢包
- QUIC协议
  - TCP存在的问题：优化难度高，握手时延大，队头阻塞问题
  - QUIC协议：基于UDP实现，建连快，包序号持续递增，消除了队头阻塞问题
- MPTCP
  - 将多径上的多个TCP子流进行协同
- 端网协同的典范：数据中心网络



作业

- 《Computer Networks-5th Edition》章节末习题
  - CHAPTER 6：23（TCP），26（TCP），29（TCP）
  - 总结QUIC与TCP的技术区别（至少4点）
  - 数据中心网络中的大小流分别关心什么？如何进行协同优化?
  - 小实验5：（UDP和TCP观察，见网络学堂附件）
- 截止时间：下下周三晚11:59，提交网络学堂

致谢社区本章贡献者

华蓓

中国科学技术大学

李沁

安徽工业大学

崔勇

清华大学

参考教材：

[1] Jim Kurose, Keith Ross,《计算机网络：自顶向下方法》(第7版)，机械工业出版社，2018.6
[2] Tanenbaum, Wetherall,《计算机网络》（第5版），清华大学出版社，2012.3
[3] 谢希仁，《计算机网络》（第7版），电子工业出版社，2017.10
[4] 徐敬东，张建忠，《计算机网络》（第3版），清华大学出版社，2013.6

特别致谢：本章课件（6.1-6.7）主要改编自《计算机网络：自顶向下方法》公开的课件