第三讲 组网与网络互连

中国科学院计算技术研究所

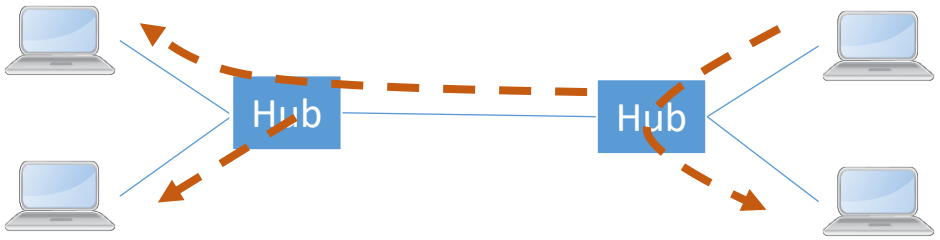
网络技术研究中心

本讲提纲

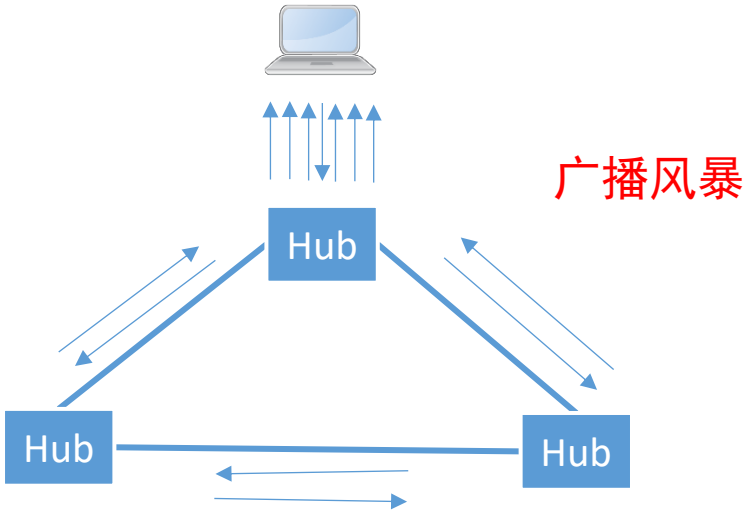
- 交换(Switching)网络
  - 交换机学习
  - 生成树协议
- 网络互连
  - IPv4协议、数据包转发
  - IPv6协议
  - IPv6过渡机制
- 数据包队列

直连网络的局限性

- 直连网络本质上是一种广播网络，带宽利用率很低

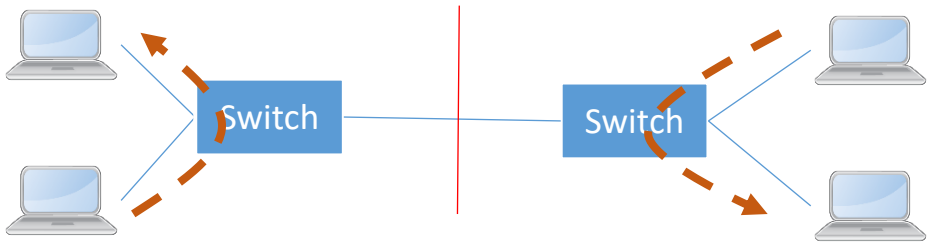


- 在有环路的拓扑中，数据包会在网络中一直被广播下去



提升网络的可扩展性

- 解决办法：
  - 网络分割：将直连网络分割成不同的段
  - 广播->单播：每个节点只将数据往目的地方向传送



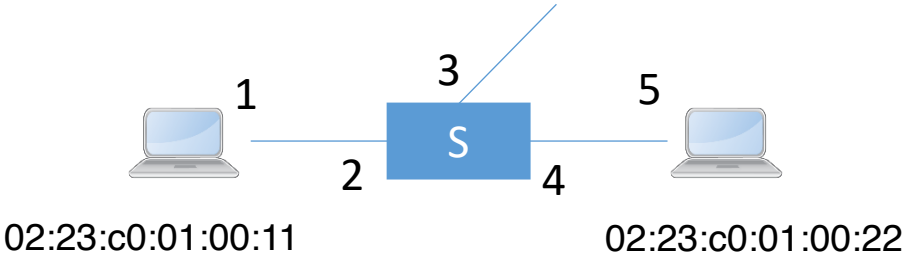
交换(Switching)网络

- 设计目标
  1. 数据只朝向目的节点方向传送（转发, Forward）
  2. 转发规则是网络自己学习生成的，不需要外界参与

- 两个主要部分
  1. 数据帧转发
  2. 学习节点位置

数据帧转发

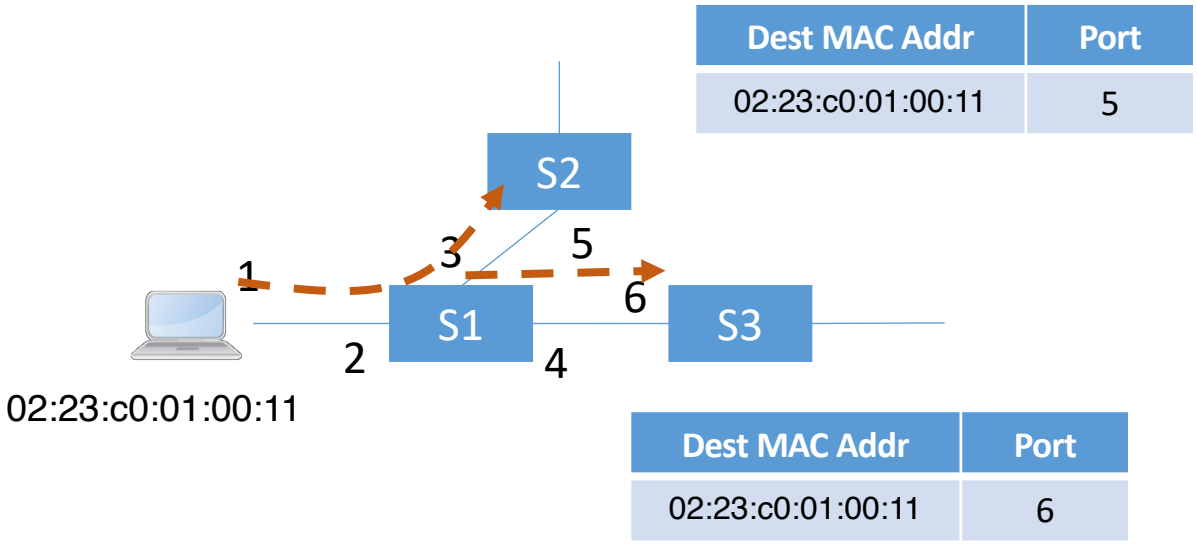
- 给定一个包含源目的MAC地址的数据帧，如何确定从哪个端口转出？
- 交换机存储目的MAC地址到（出）端口的映射关系（Forwarding Database, FDB)
- 对于每个数据帧，在FDB中查找目的MAC地址对应的端口号
  - 如果存在对应端口号，从该端口将数据转发（单播）
  - 如果FDB中不存在对应条目，将该数据包从所有端口转发（广播）
- FDB表中MAC地址通过老化机制 (Aging)来更新



Dest MAC Addr	Port	Age
02:23:c0:01:00:11	2	16
02:23:c0:01:00:22	4	20

学习节点位置

- FDB条目是如何生成的？
- 每收一个新的数据帧，记录其源MAC地址和入端口，将该映射关系写入FDB表



消除广播风暴

- 网络中存在冗余链路（提升网络健壮性等）
  - 网络拓扑由树状结构变成图状结构
  - 数据转发过程中，形成广播风暴
- 解决办法：
  - 为网络中每对源目的节点分配唯一确定的一条路径
  - 这些路径构成构成了一棵树（生成树，Spanning Tree）
- 生成树协议：
  - 选一个根节点，其它每个节点计算确定到根节点的最短路径
  - 保证是最小生成树（Minimum Spanning Tree）

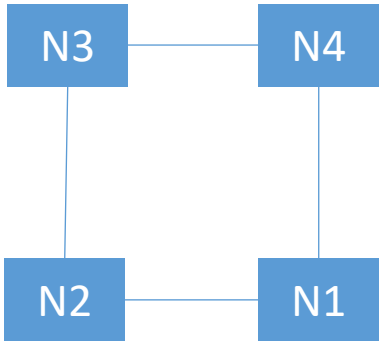
生成树算法

- 每个节点向邻居节点发送如下消息：
  - 本节点ID、其认为的根节点ID、到该根节点的距离
  - 例如节点S1发送 (S1, S1, 0)
- 当节点收到邻居的消息后，比较确定是否更新其保存的信息：
  1. 包含ID更小的根节点
  2. 同一根节点但距离更近
  3. 同一根节点、相同距离、但发送方ID更小
- 算法收敛后：
  - 网络中已经选举出根节点
  - 每个节点都可以确定到根节点的距离和出端口

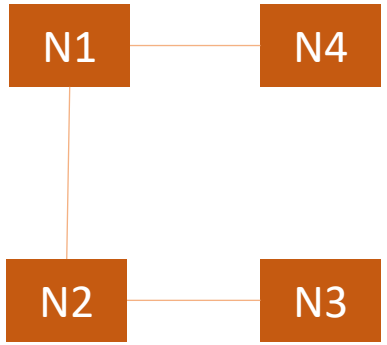
生成树算法举例

- 节点N1:
  - 一直发送(N1, N1, 0)
- 节点N2：
  - 发送(N2, N2, 0)
  - 收到N1消息后，更新并向N3发送(N2, N1, 1)
- 节点N3：
  - 发送(N3, N3, 0)
  - 收到N4的消息(N4, N1, 1)后，更新并向N2发送(N3, N1, 2)
  - 收到N2的消息(N2, N1, 1)后，更新并停止发送

网络拓扑



生成树拓扑



交换网络总结

- 连接方式
  - 集线器 -> 交换机
- 数据传输方式
  - 广播 -> 单播
- 链路共享机制
  - 每个（全双工）链路只有两个节点，不需要CSMA/CD
- 拓扑特征
  - 层次结构树
  - 冗余链路的网络：生成树拓扑

网络互连

- 是否可以将交换网络扩充到全球范围？
  - 不可以，存在可扩展性问题
    - FDB表的膨胀问题
    - STP收敛速度问题
  - 网络接入方式多样性
- 需要一种全网可达的网络协议
  - 可提供网络互连功能、端到端传输
  - 从互联网体系结构模型角度看，网络层对底层网络技术进行抽象，为上层网络应用提供统一的接口

网络层设计思路

- 向上提供最基本的端到端传输服务
  - 无连接的、尽最大努力交付（best-effort delivery）的数据报服务
- 发送数据报时不需要先建立连接
  - 数据报没有编号，每一个数据报独立发送，与其前后的分组无关
- 不提供服务质量的承诺
  - 所传送的数据报可能丢失、重复和乱序
  - 也不保证数据报传送的时限
- 优点：
  - 协议设计简单，适应性强
  - 中间转发设备功能简单，成本低

网络层设计

- IP地址与数据报文格式
  - IP地址空间划分
- 数据报文传输
  - 数据报文转发
  - IP地址向MAC地址的映射
- 连接不同网络
  - IP分片
- 网络控制与诊断
  - ICMP

IPv4地址

- IPv4地址格式：32位地址空间，理论上可提供约40亿个主机地址
  - 点分十进制表示：为便于记忆，将其划为4个字节，由小数点分开，每个字节用十进制来表示，例如：

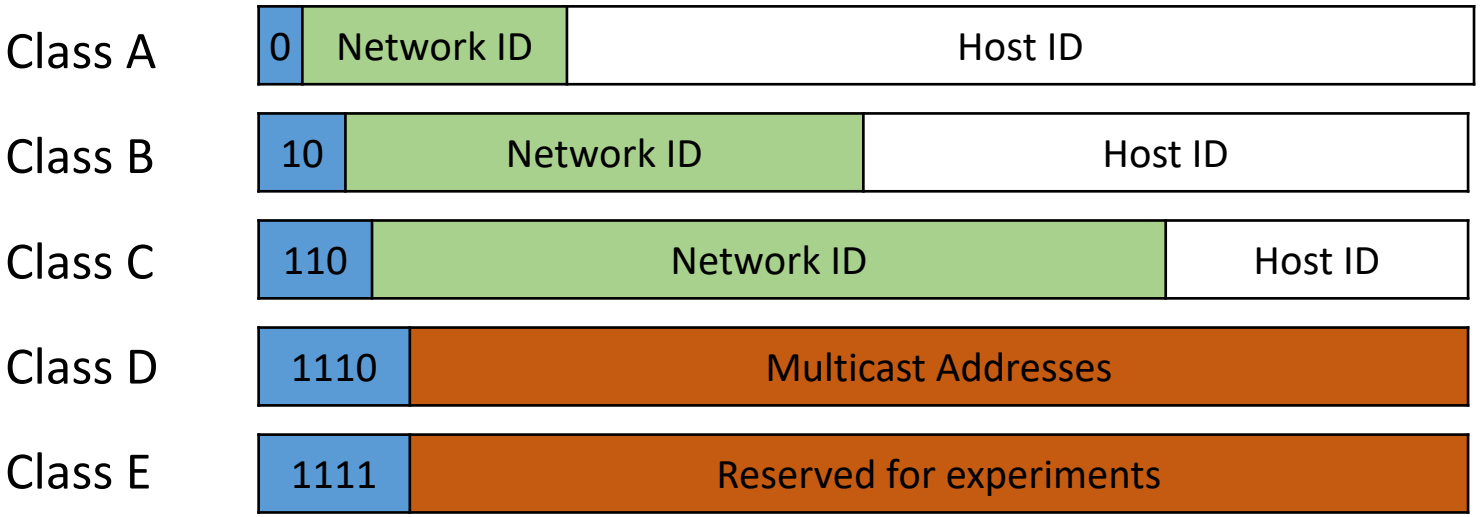
11000000 10101000 00000001 01100011

192.168.1.99

- IPv4地址的结构
  - 层次化的组织方式：IP地址 = 网络号 + 主机号
  - 网络号由 ICANN 分配，主机号由网络管理员分配
  - 前者表示主机所在网络，后者表示主机在该网络内的标识

有类别的IP地址空间划分

- 按机构规模进行分配
- 通过IP地址本身就可以确定网络号(Network ID)
- 局限性：
  - 如果一个机构需要2^16+1个IP地址，如何分配地址空间？
  - 对于很多机构，C类地址空间太小，A类、B类空间太大



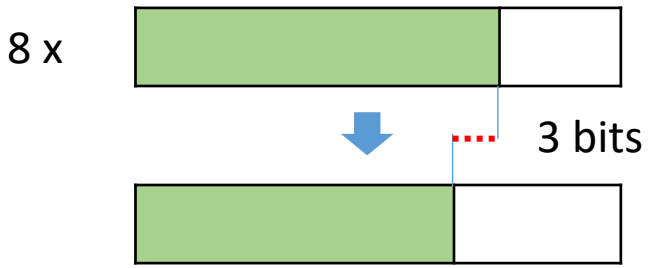
无类别域间路由（Classless Inter-Domain Routing, CIDR）

- 地址中前K位为网络号，剩余部分为主机号，K可以是任意值
  - 更高效的利用IP地址空间
  - 网络号不再由Class来确定，通过前缀（prefix）长度来确定
  - 例如：192.168.1.99/28，该地址前28位为网络号，后4位为主机号
  - 通常，前缀长度也可以用网络掩码(network mask)来表示
    - 网络掩码的长度与IP地址相同，前K位全为1，后面为0，通常也用点分十进制表示
    - 例如，28位前缀长度对应的网络掩码为255.255.255.240
  - 在一个网络内，可以使用更大的前缀长度来进一步划分子网

CIDR举例

- 如何将有类别的IP地址空间转化为CIDR地址空间？
- 假设网络分配得到192.168.0.0 – 192.168.7.255，8个连续C类地址空间
  - 将该C类地址中网络号的末3位移做主机号
  - 网络号减为21位：192.168.0.0/21
  - 将路由表中8个C类地址路由条目替换为1个长度为21位的路由条目

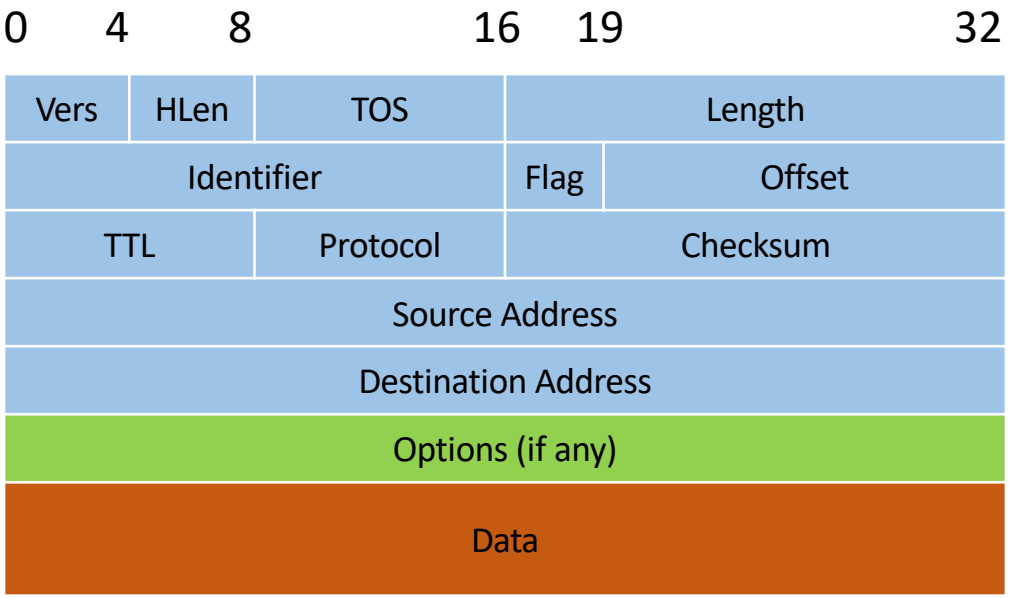
对应三位bit



IP地址的特点

- 层次化地址结构
  - 地址管理机构在分配IP地址时只分配网络号，方便地址管理
  - 提升了网络系统的可扩展性
    - 路由器仅根据目的主机的网络号来转发分组，使路由表中条目数大幅度减少
    - 当网络拓扑发生变化时，更新次数大大减少
- CIDR可更充分的使用IP地址空间
  - 会增加路由、转发的复杂度，但在现代网络中不是问题

IP数据包头部格式

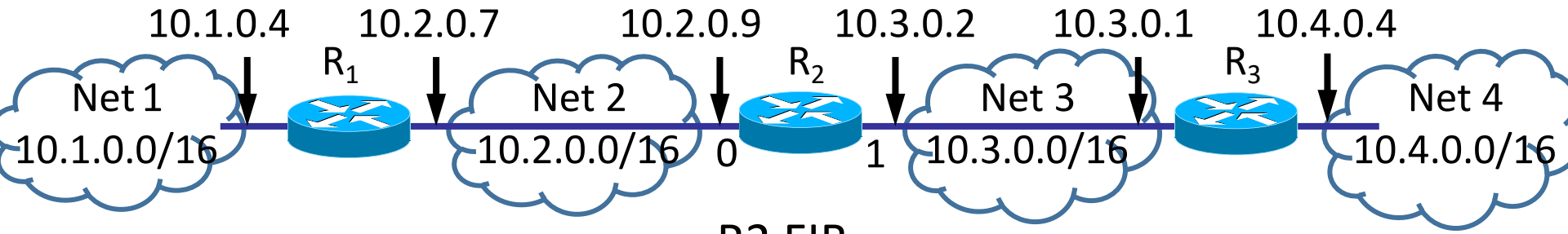


- Vers: 版本号，该处为4，对应IPv4
- HLen：IP包头部长度，通常为5，表示有5个32bit word，20个字节
- Length：IP数据包长度，最大为65535字节
- Identifier, Flag, Offset：用于IP数据包分片

- TTL：存活时间，数据包每经过一个路由器，该值减1，当为0时被丢弃
- Protocol：标识所承载协议类型，例如 TCP: 6, UDP: 17
- Checksum：报文校验值
- Source & Destination Address：源目的IP地址
- Options：包括源路由等，通常为空

IP报文转发

- 路由器将转发信息存储在转发表中（Forwarding Information Base）
  - 转发表中存储的是网络号与下一跳地址的映射关系



R2 FIB

Dest Network	Next Hop
10.2.0.0/16	Interface 0
10.3.0.0/16	Interface 1
10.1.0.0/16	10.2.0.7, Interface 0
10.4.0.0/16	10.3.0.1, Interface 1

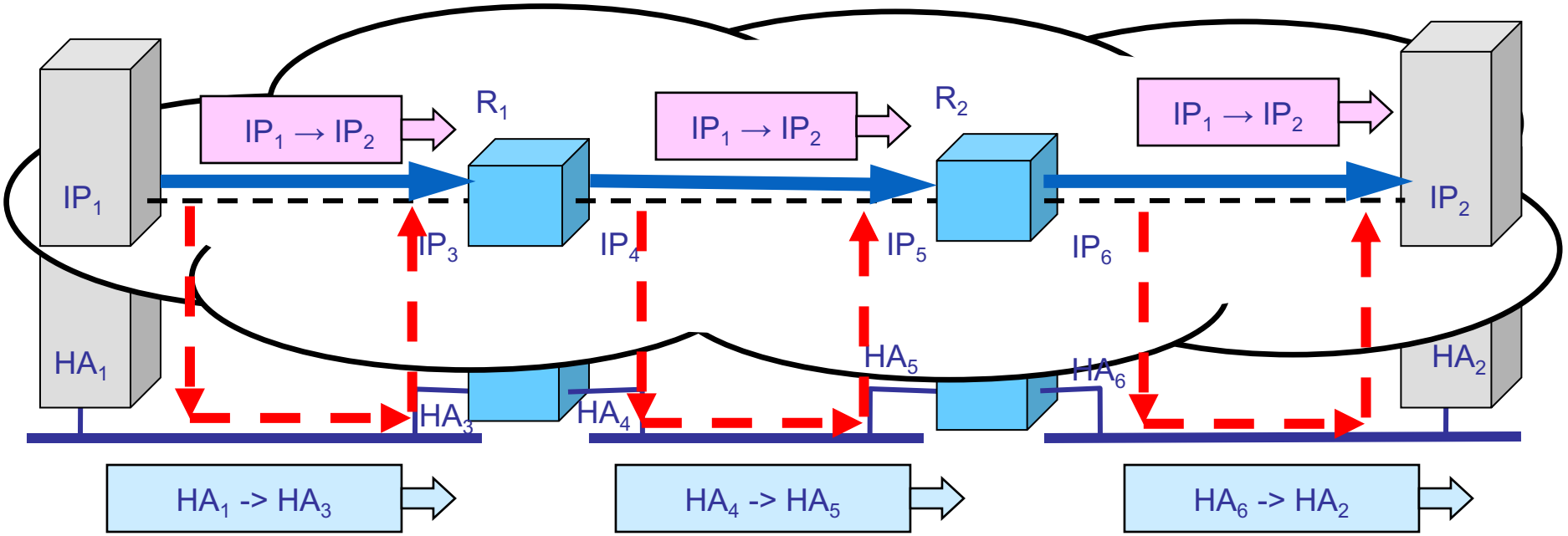
IP报文转发规则

- 路由器收到IP报文后，获取目的地址D，按照如下规则进行转发：

1. 如果D与路由器在同一网络内，直接交付给主机D，并返回；
2. 在转发表中进行最长前缀匹配，如果匹配成功，则将IP报文转发到该下一跳网关，并返回；
3. 如果转发表中有默认路由，则将IP报文转发给默认路由器，并返回；
4. 报告转发分组出错（ICMP，目的网络不可达）。

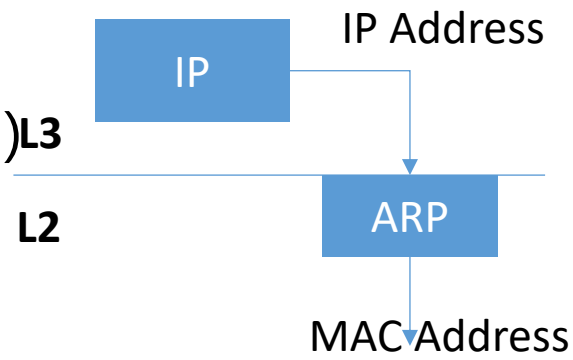
IP地址到MAC地址的映射

- IP层抽象屏蔽了下层MAC协议的细节
  - 在链路层只能看见 MAC 帧而看不见 IP 数据报
  - 数据传输时，需要IP地址到MAC地址的映射



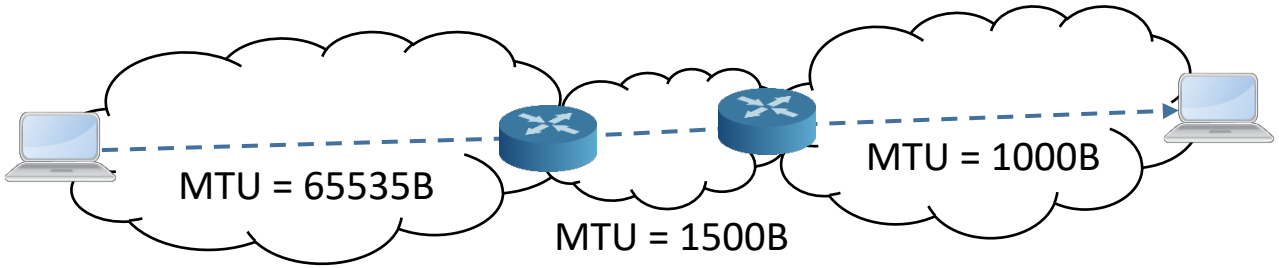
地址解析协议 (Address Resolution Protocol, ARP)

- ARP：知道下一跳IP地址，查询其MAC地址
- 每个路由器/主机中都有一个ARP缓存（Cache）
  - 存储局域网内各主机或路由器的地址映射关系
- 节点A向局域网内另一节点B发送IP报文
  - 如果ARP Cache中有对应条目，将该MAC地址作为目的地址发送数据帧
  - 否则，A向局域网内广播ARP请求，询问节点B的IP地址对应的MAC地址
    - B收到该请求后，回复自己的MAC地址
    - A和B都会将对方地址的映射关系写入ARP Cache
- 注意：ARP协议只作用于局域网，两节点IP不在同一网段时，数据报文应先转发到路由器



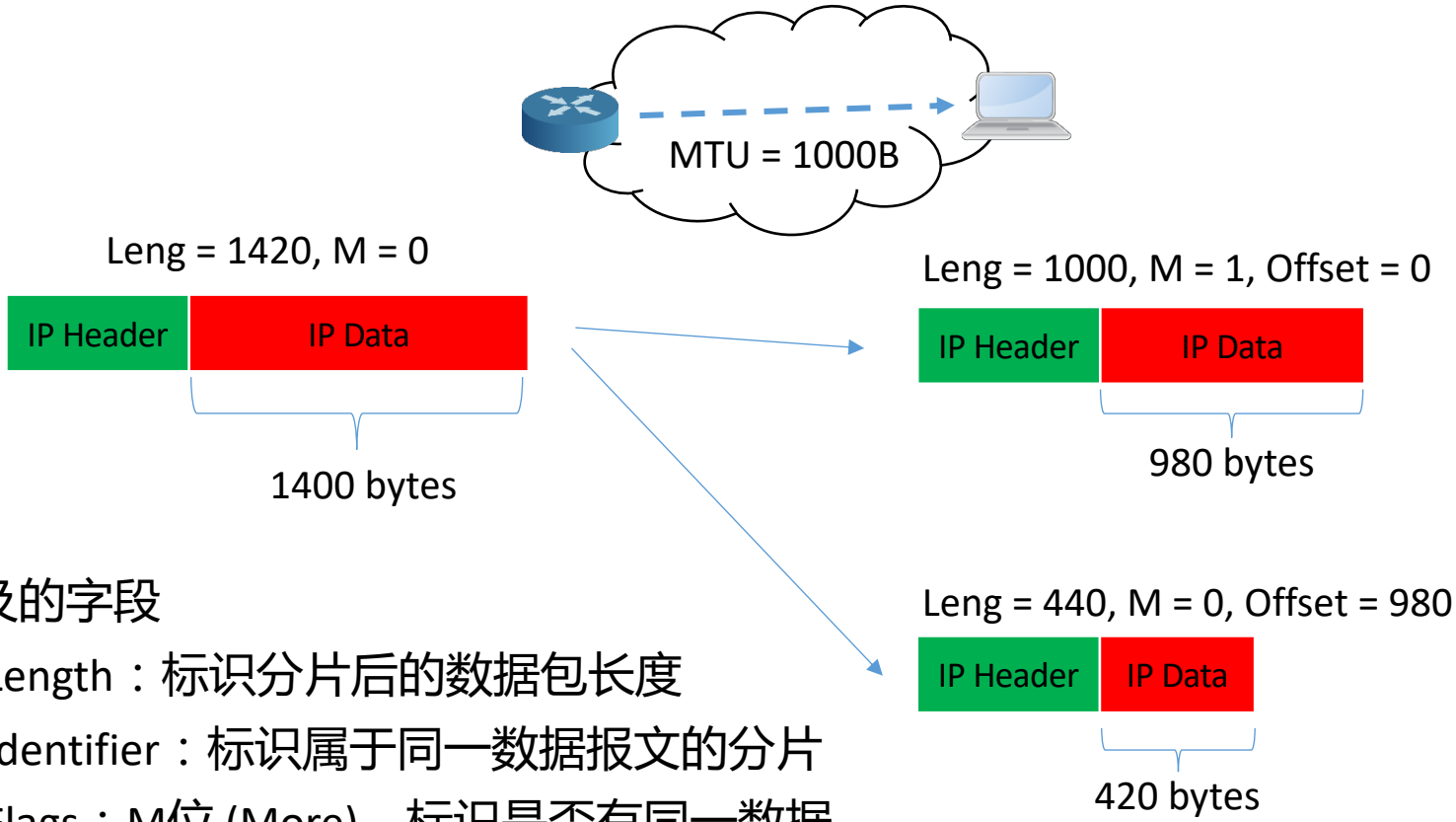
IP分片（Fragmentation）

- 每个网络拥有各自的最大传输单元长度（Maximum Transmission Unit, MTU）
  - MTU就是IP协议在一个数据包中承载的最大数据量
    - Ethernet的MTU是1500字节
  - 发送方不知道每个中间网络的MTU值



- 解决方案：
  - 当IP数据报文大于网络MTU时，路由器负责数据包分片
  - 所有分片数据包到达目的主机后，目的主机负责还原原始IP报文

IP分片举例



涉及的字段

- Length：标识分片后的数据包长度
- Identifier：标识属于同一数据报文的分片
- Flags：M位 (More)，标识是否有同一数据报文的后续分片
- Offset：标识分片在数据报文中的位置

IP分片的缺点

- 不能充分利用网络资源
  - 网络转发代价与包数目相关，与大小无关
  - IP分片不能探测传输路径MTU大小
    - 数据报文太大，导致分片
    - 数据报文太小，转发数据包增多
- 端到端性能很差
  - 当一个分片丢失时，接收端会丢弃同一报文的其他分片

- 解决方案：
  - 使用路径MTU发现机制，在数据传输过程中确定沿途网络的最小MTU

互联网控制消息协议(Internet Control Message Protocol, ICMP)

- ICMP协议通过发送错误代码、控制信息等来诊断和控制网络
  - 由IP协议封装
  - 基本不用于数据传输
- 应用举例：
  - Ping：检查对端主机是否可达
  - 终点不可达：不同的代码指示为什么数据包不可达
  - 时间超时：数据包超出最大跳数限制
  - 路由控制：重定向网络或主机
  - 源点抑制：控制节点发送速率

IP地址带来的问题

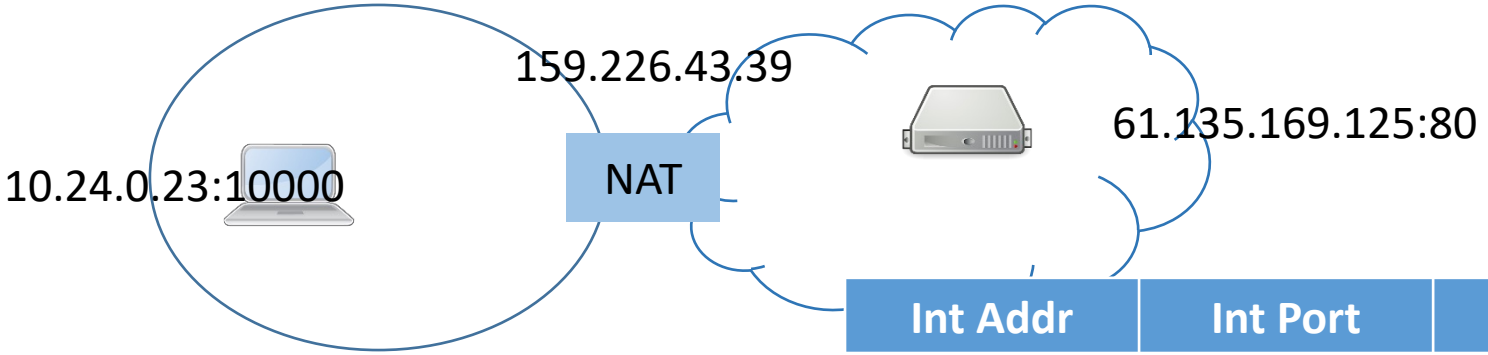
- 早期IP网络：每个主机拥有唯一的一个IP地址
  - 任何主机都可以与其他主机相连
  - 任何主机都可以作为服务器，提供服务
- 互联网系统开放性带来的安全问题
  - 任何主机都可以攻击其他主机
  - 攻击者可以伪造数据包的源地址
- 是否每个主机都需要一个”其他人可访问的”IP地址？
  - IP地址空间不足
  - 安全性问题

NAT (Network Address Translation)

- RFC1918定义全局IP地址和私有IP地址：
  - 全局IP地址：用于互联网 -- 公共主机和路由设备
  - 私有IP地址：仅用于组织的专用网内部 -- 本地主机
    - 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- 在组织内部，给每个主机分配一个私有IP地址
  - 使用标准IP路由协议，且可以进一步划分子网
- NAT负责私有地址与全局地址之间的翻译
  - 外部网络无需知道内部网络的地址情况

NAT例子1：客户端发起连接

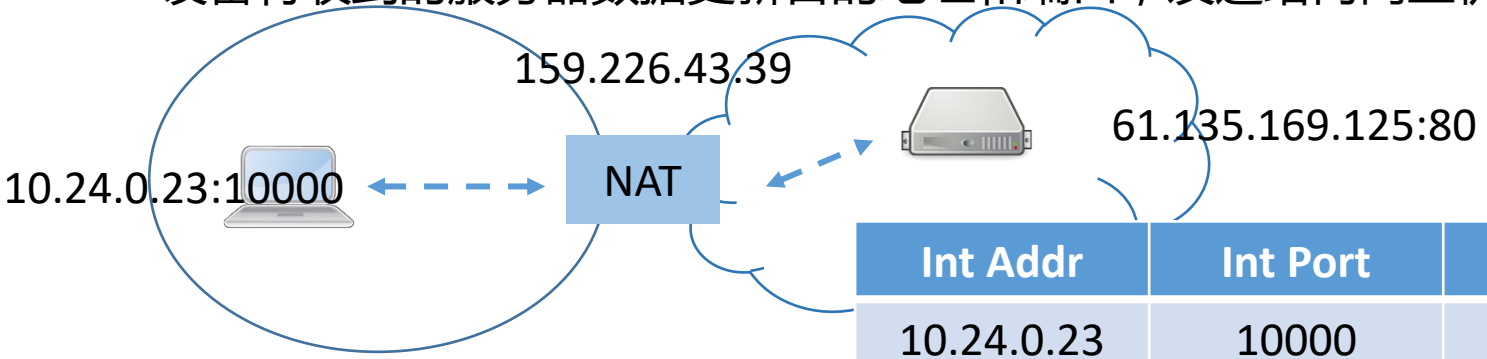
- 客户端 10.24.0.23 想连接到服务器61.135.169.125
  - 客户端协议栈分配一个临时端口号10000，并发送请求
- 连接请求经过NAT设备
  - NAT分配一个端口号50000，与该客户端请求建立映射关系
  - NAT转发请求，将客户端的源地址和源端口替换为自己的地址和端口



Int Addr	Int Port	NAT Port
10.24.0.23	10000	50000

NAT例子1：NAT设备作为两端的代理

- NAT设备作为两端的代理
  - 对于客户端，NAT拦截其消息，并标记自己为发送方
  - 对于服务器端，NAT设备作为服务器消息的接收者
  - NAT设备将收到的服务器数据更新目的地址和端口，发送给内网主机



Int Addr	Int Port	NAT Port
10.24.0.23	10000	50000

Src IP: 10.24.0.23
Src Port: 10000

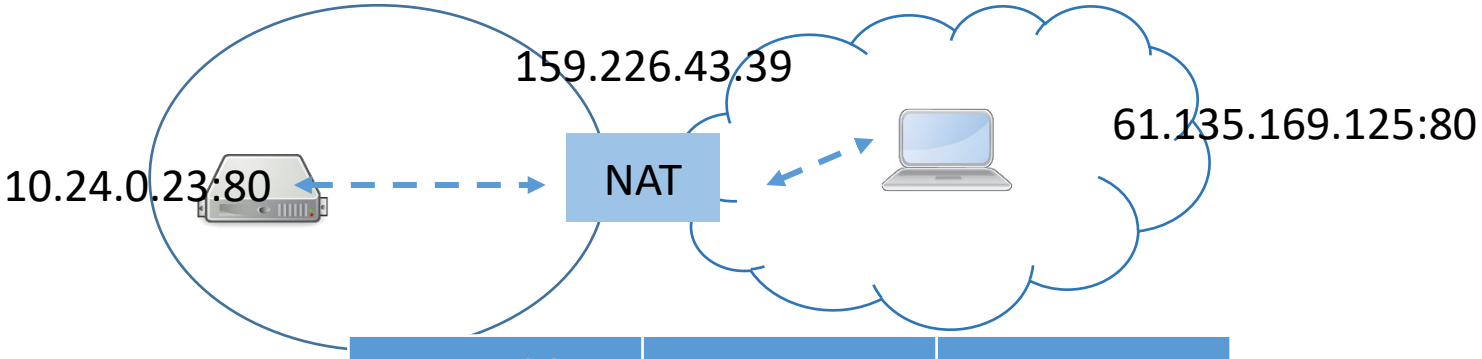
Dest IP: 61.135.169.125
Dest Port: 80

Src IP: 61.135.169.125
Src Port: 80

Dest IP: 159.226.43.39
Dest Port: 50000

NAT例子2：将内网主机作为服务器

- NAT设备将内网主机作为服务器
  - 使用端口映射技术，将内网主机的端口对外可见
    - 需要手动配置
  - 外网用户通过地址159.226.43.39:80访问该服务
  - NAT设备收到该请求后，将目的地址替换，并转发给服务主机



Int Addr	Int Port	NAT Port
10.24.0.23	80	80

为什么要设计IPv6？

- IPv4取得了极大的成功
- IPv4地址资源的紧张限制了IP技术应用的进一步发展
  - CIDR、 NAT等技术暂缓了IPv4地址紧张，无法根本解决地址问题
- 新技术的出现对IP协议提出了更多的需求
  - QoS保障、移动性、内置安全等
- IPv6的设计出发点：
  1. 近乎无限的地址空间
  2. 更简洁的报文头部
  3. 内置的安全性
  4. 更好的QoS支持
  5. 更好的移动性

IPv6地址格式

- IPv6地址长度为128位
- IPv6地址用十六进制表示，分为8段，中间用“:”隔开
  - 2001:0410:0000:0001:0000:0000:0000:45ff
- 每段的起始0可以省略，连续全零的段可用“::”表示（只能出现一次）
  - 2001:410:0:1:0:0:0:45ff，2001:410:0:1::45ff
- 不同起始码对应不同类型IPv6地址，例如::ffff/96表示与IPv4兼容的地址
- IPv6地址 = 前缀 + 接口标识
  - 前缀：相当于IPv4地址中的网络ID
  - 接口标识：相当于IPv4地址中的主机ID，固定为64位。
  - 前缀长度用“/xx”来表示
  - 2001:410:0:1::45ff /56

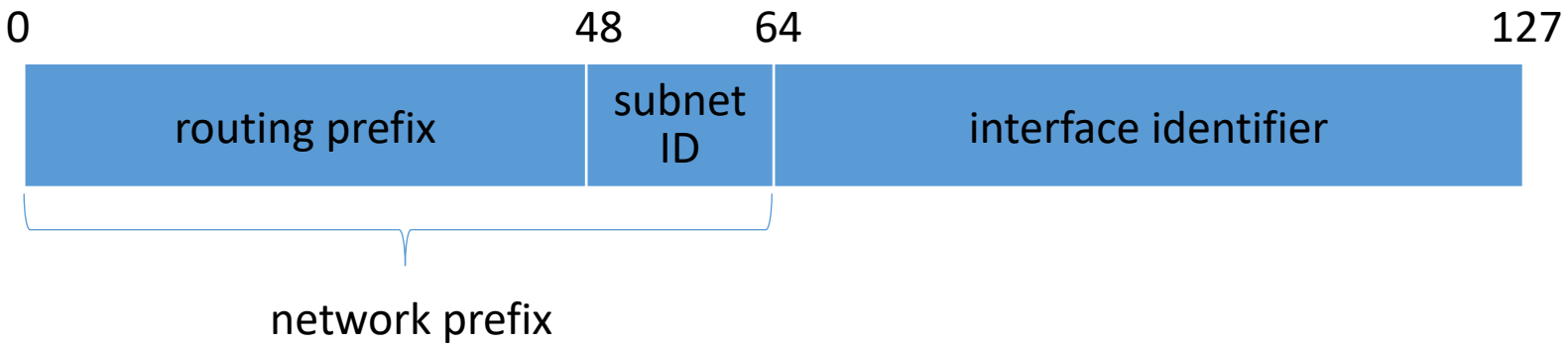
IPv6编址

- 我们确实需要这么多IPv6地址么？
  - 从长远角度看，确实有可能
  - 上世纪90年代，出现”IPv4地址耗尽”恐慌
  - 更多的（小型智能）设备将会联网，手机、传感器、电饭煲等
- 128位地址空间带来的直接好处：
  - 更容易进行层次化(Hierarchical)编址
  - 例如，为局域网分配一个48-bit的地址空间 – 可以直接使用MAC地址
  - 可以利用地理位置编址（例如，IPv4地址空间）
  - 可以减少路由表条目数（也许一个运营商就一个IPv6前缀）

IPv6单播地址的分层结构

IPv6 扩展了地址的分层概念，使用以下三个等级：

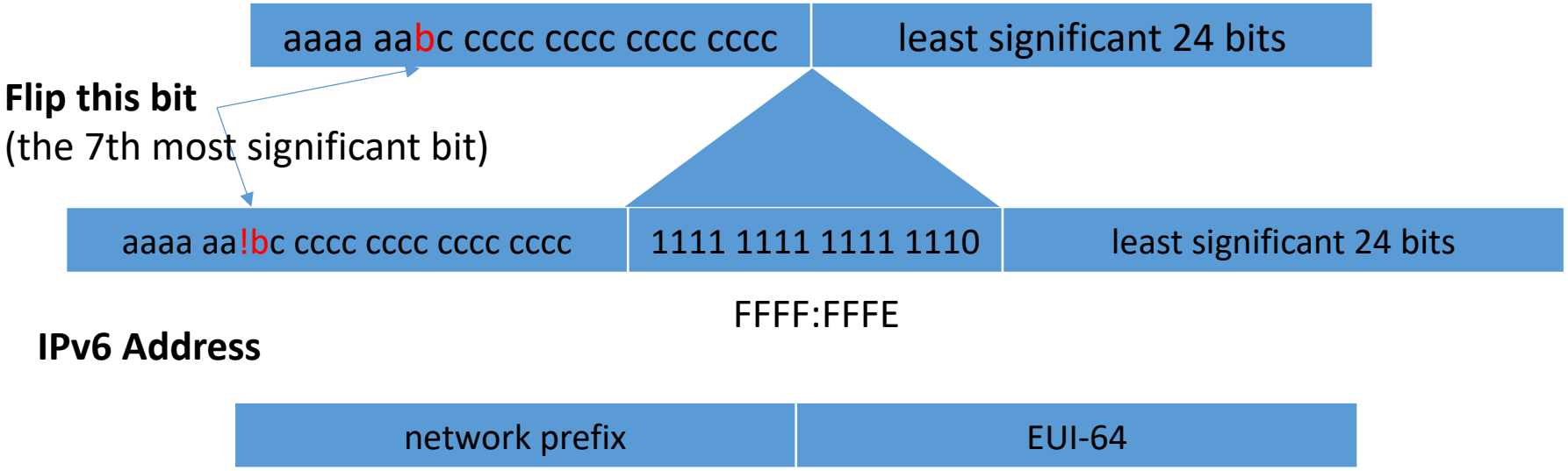
- 全球路由选择前缀，占 48 位（或更多）
- 子网标识符，占16 位（或更少）
- 接口标识符，占 64 位



以太网地址转换为IPv6地址

1. 先将MAC地址转换为EUI-64标识
   - EUI-64：64位全球唯一标识（Extended Unique Identifier）
2. IPv6网络前缀（ 64位） + EUI-64地址

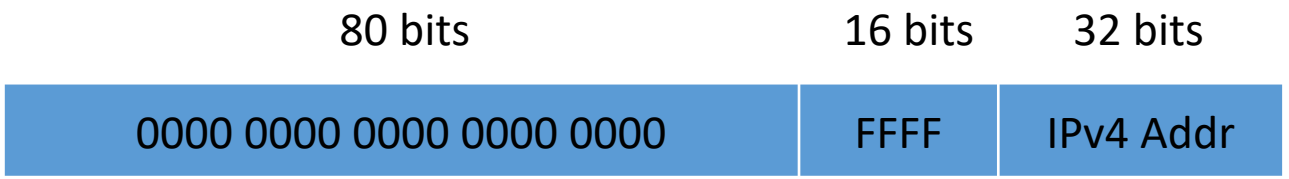
Mapping MAC Address to EUI-64



IPv4地址到IPv6地址的映射

- 前缀为 ::ffff/96是保留一小部分地址与 IPv4 兼容的
  - 考虑到在较长时期内 IPv4和 IPv6 将会同时存在，而有的节点不支持 IPv6
- 数据报在这两类节点之间转发时，需要进行地址的转换
  - NAT-PT (Protocol Translator)

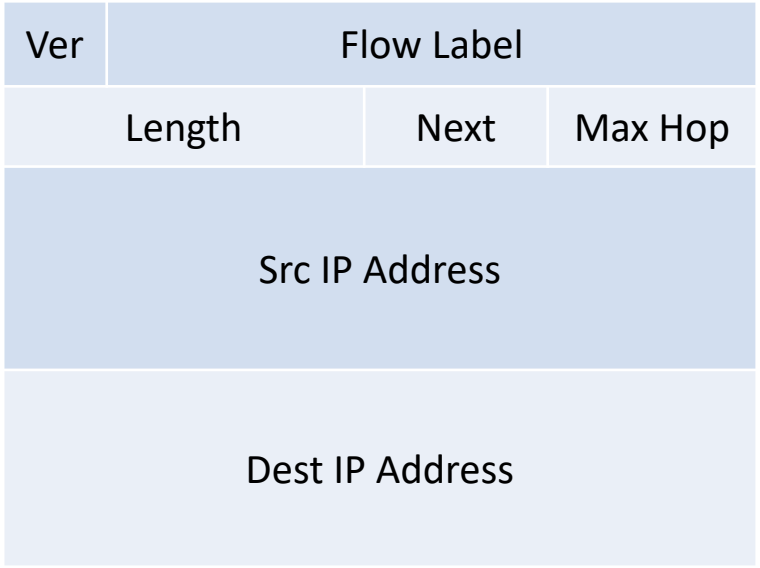
Mapping IPv4 to IPv6



IPv6数据包头部格式

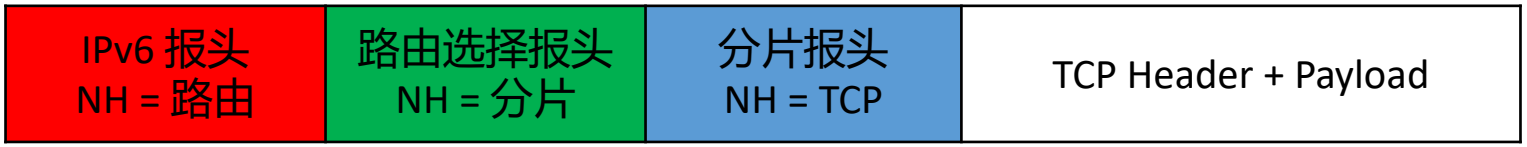
- IPv6作为下一代IP协议
  - 最根本的动机在于增大地址空间
- 简化了数据包头部，包处理速度更快
  - 没有了Checksum
  - 没有了分片
- 更好的支持不同类型的服务
  - 支持不同服务优先级、支持流标识
- 扩展选项由Next域来标记
  - 减少了处理扩展选项的开销

IPv6 Packet Header



IPv6扩展报头

- IPv6将一些网络层的可选功能放在IPv6的扩展头部中
- 主要的扩展报头：
  - Hop-by-Hop Options header
  - Destination Options header
  - Routing header
  - Fragment header
  - …
- IPv6扩展报头示意

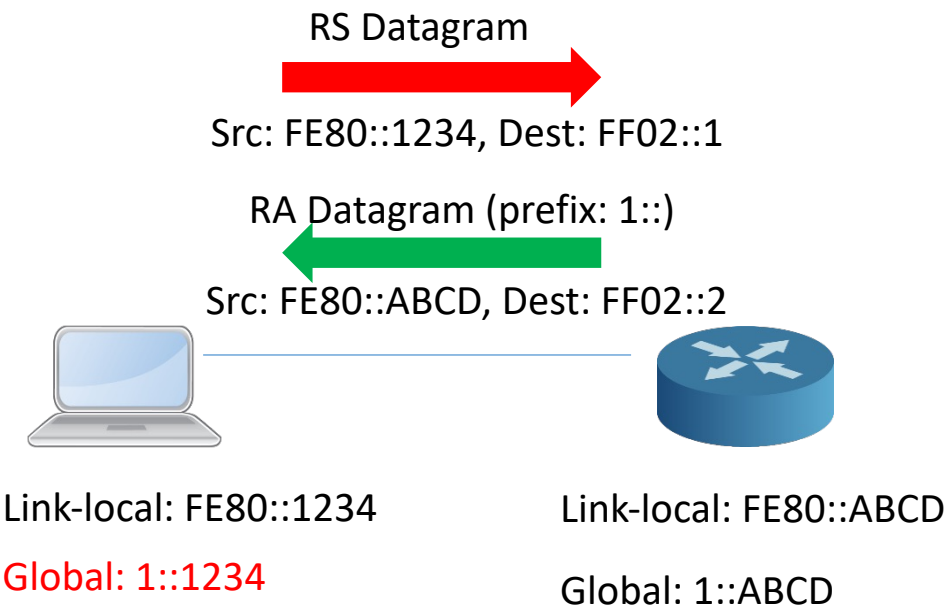


IPv6扩展报头优点

- IPv4扩展选项
  - IPv4选项对路由器转发性能产生负面影响
  - 很少使用
- 相比于IPv4扩展选项
  - 扩展报头在IPv6报头的外部
  - 路由器可以不考虑这些选项
    - 逐跳选项除外
  - 对路由器转发性能无负面影响
  - 易于通过新的扩展报头进行功能扩展

IPv6地址自动配置

- 目的是获得全局网络前缀
- 无服务器的/无状态的
  - 即不需要类似DHCP服务器
  - 只配置地址，不做任何其他事情
- Link-local地址
  - FE80 :: EUI-64
- 主机发送Router Solicitation报文
  - 使用Link-Local地址，组播发送
- 路由器回应Router Advertisement报文
  - 包含自身网络前缀



IPv6地址解析

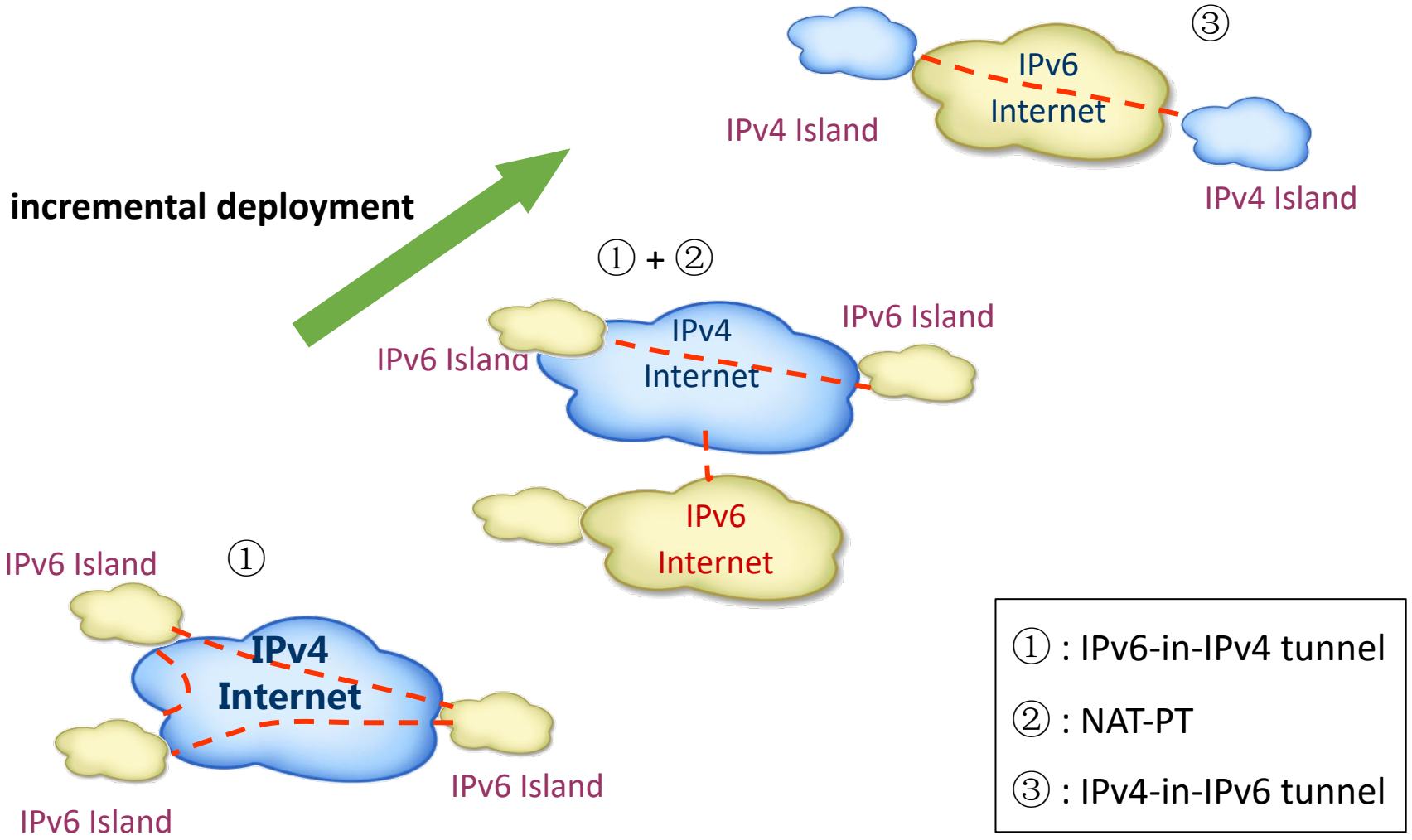
- IPv6取消了ARP协议，如何完成三层地址到二层地址的映射？
  - 通过邻居请求报文（NS）和邻居通告报文（NA）来解析三层地址对应的链路层地址

1. 发送主机在接口上发送NS报文，该报文的目的地址为目标IPv6地址所对应的请求节点组播地址，NS报文中包含了自己的链路层地址
2. 目标主机收到NS报文后，就会了解到发送主机的IP地址和相应链路层地址
3. 目标主机向源发送主机发送一个邻居通告报文（NA），该报文中包含自己的链路层地址

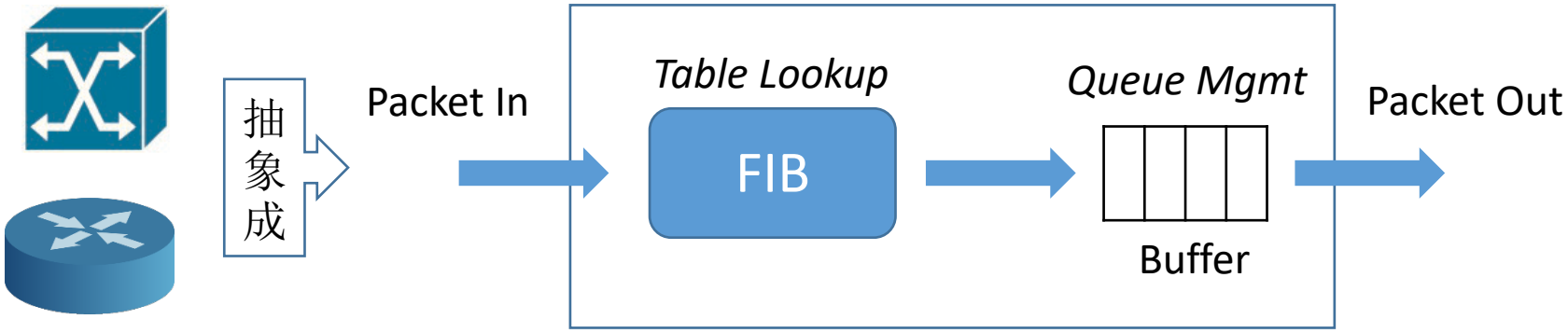
IPv4向IPv6过渡

- 为了实现增量部署(incremental deployment)，IPv6需要保证对IPv4的互操作性(Interoperability)
- 通过以下机制实现互操作性：
  - 双协议栈技术 (Dual-Stack)
    - 设备上同时支持IPv4和IPv6协议栈，是其他过渡技术的基础
  - 隧道技术(Tunnel)
    - 把IPv6报文封装在IPv4报文中，IPv6网络之间穿越IPv4网络进行通信
  - 互通技术NAT-PT
    - 在两种网络的相连处进行两种地址间的翻译，修改协议报头，使IPv4网络与IPv6网络能够互通

IPv6网络增量部署（场景）



数据包队列

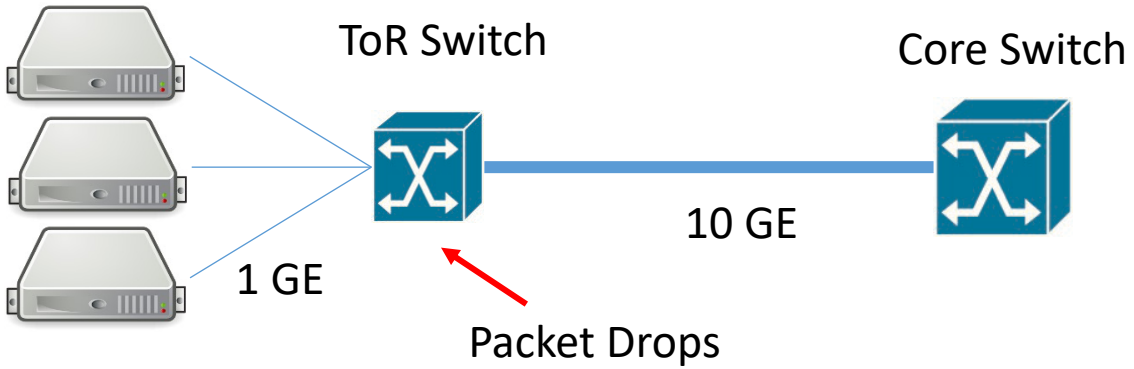


- 数据包队列是网络中间设备中最关键的部分之一
  - 其大小、管理策略等很大程度上影响了网络性能

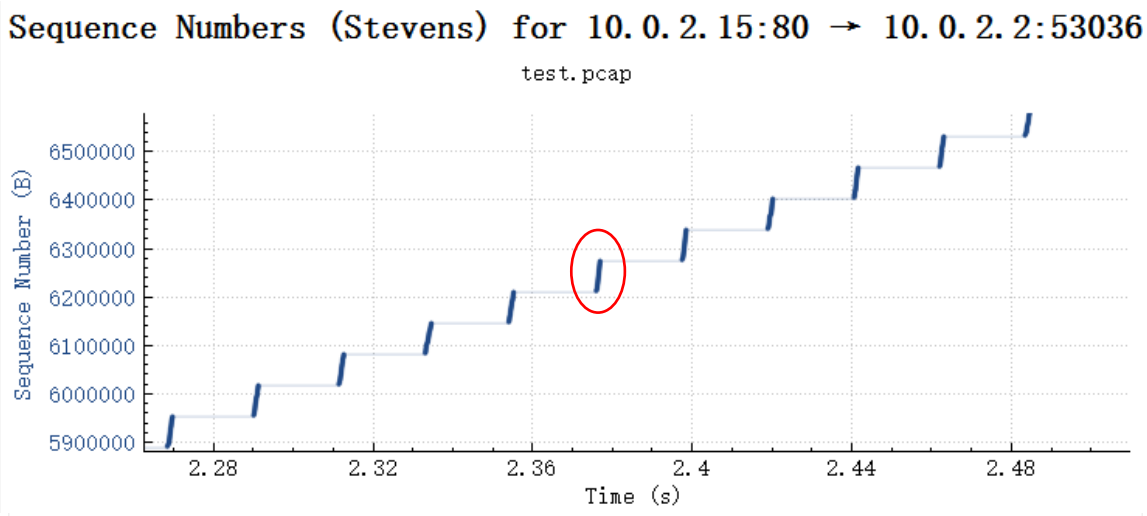
注：下文在不引起歧义的情况下，用数据包队列/队列表示Buffer

为什么需要数据包队列？

瓶颈链路



突发流量 (Burst)



队列应该设置成多大？

- 经验法则
  - $BuffSize = \overline{RTT} * C$
  - C为瓶颈链路带宽
  - $\overline{RTT}$为端到端平均链路延迟

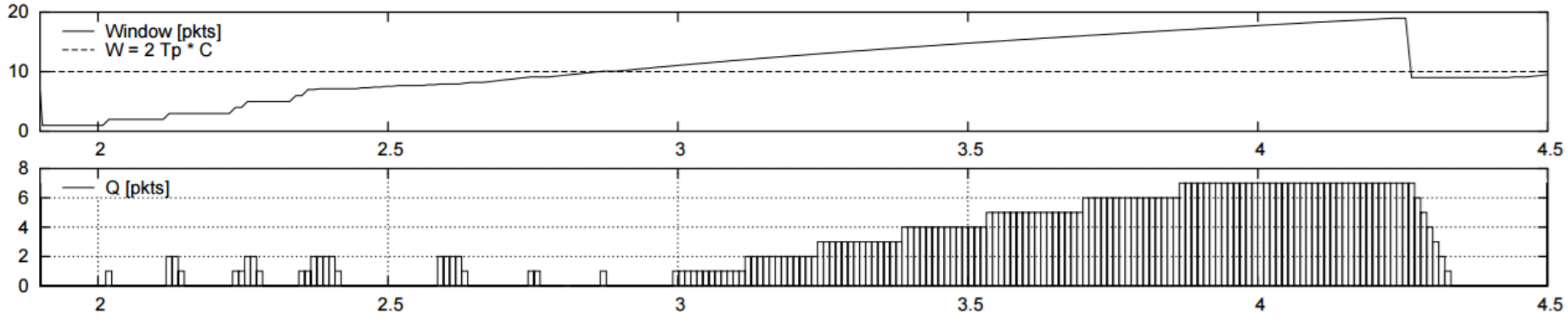
- 家庭接入网络的例子
  - 带宽：100Mbps
  - 网络延迟：20 – 100ms (60ms)
  - 家庭网关Buffer大小：100Mbps * 60ms = 6Mbps = 0.75MB

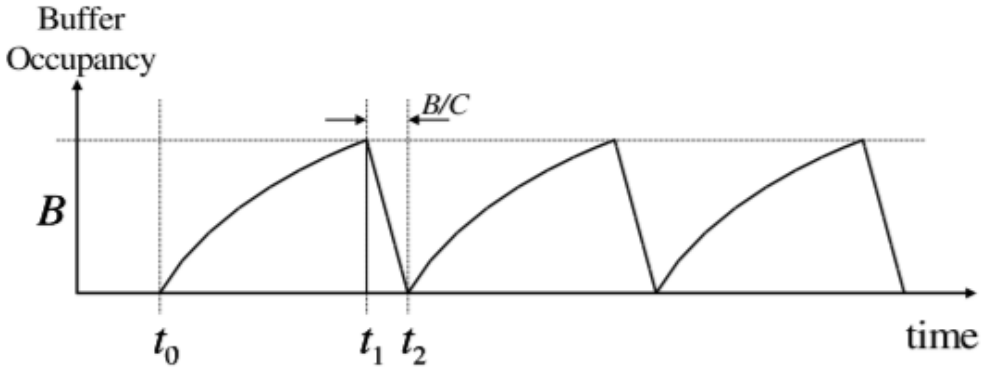
背景知识

- 队列大小与队列管理策略、传输控制策略等关系密切
- 队列管理策略
  - 先进先出队列：当可以转发数据包时，将队首的数据包转出
  - 当数据队列满时，有后续数据包到达怎么办？
    - 最简单的策略：后续的数据包直接丢弃，直到队列有空间为止
- 传输控制策略
  - 互联网中90%以上的流量都由TCP (Transport Control Protocol)承载
  - TCP通过发送窗口管理数据发送行为
    - 当在途数据量小于发送窗口时，发送数据包
    - 对方每确认一个数据包，发送窗口加一；发生数据丢包时，窗口减半

队列大小与传输控制

- 队列大小决定传输速率
  - 更大的队列允许单位时间内发送更多的数据：$T = \frac{W}{RTT}$
  - 发送窗口大小与队列中数据包个数呈正比
- 传输控制策略影响了队列行为

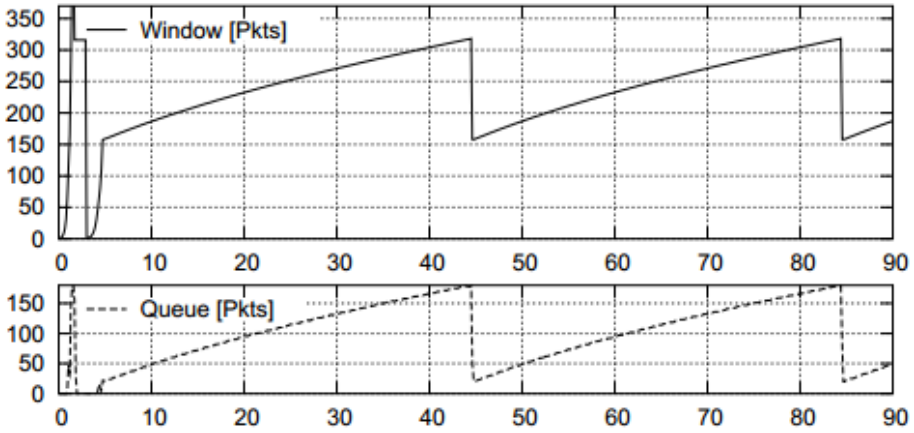




- B/C为队列从满到排空所需的时间
- 队列长度与时间的关系呈锯齿状

队列过大或过小

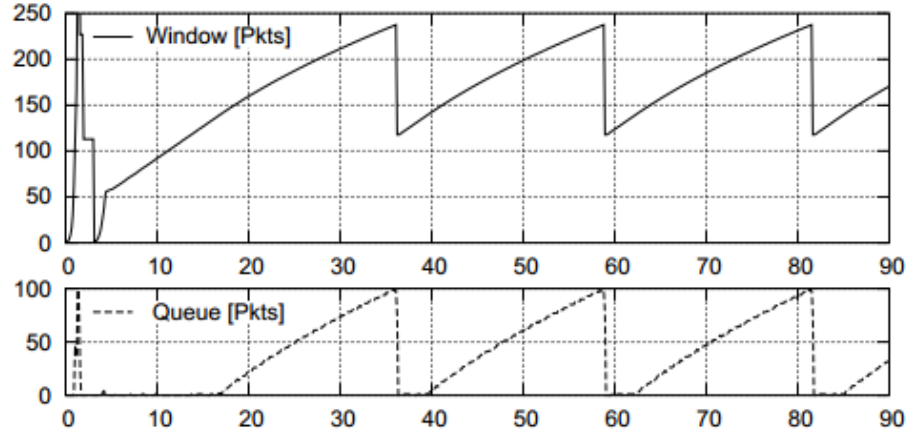
- 队列过大



队列过大时，$BuffSize > \overline{RTT} * C$，当窗口大小减半时，队列不能清空数据包，因此数据包的延迟会增加

（$\frac{BuffSize}{C} - RTT$）

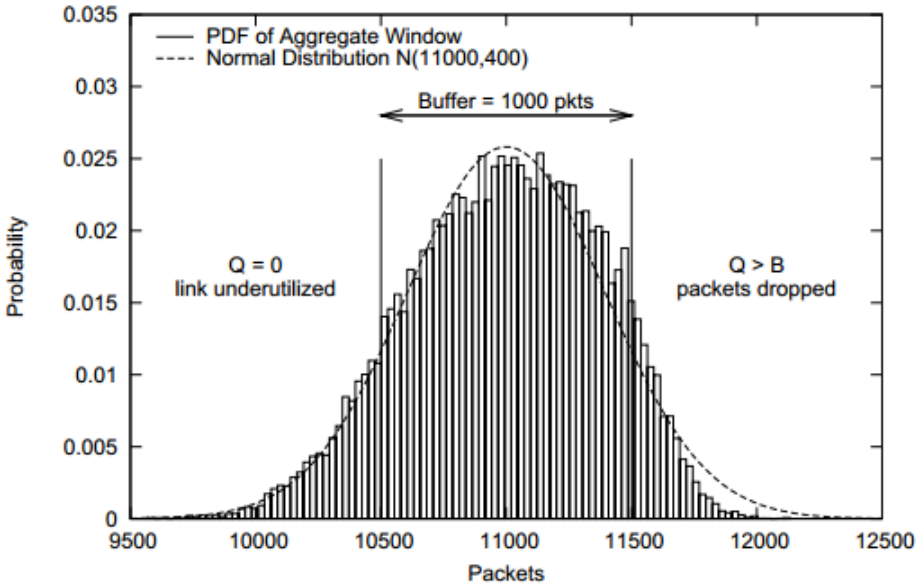
- 队列过小



队列过小时，$BuffSize < \overline{RTT} * C$，当窗口大小减半时，发送方在等待对方的数据确认，因此瓶颈链路处于空闲状态

多流环境下的队列大小

- 假设多条流的数据包到达队列的时间是随机的



所有流的窗口大小之和服从Gaussian分布

- 对于多并发流经过的队列，其大小只需设置成BDP/sqrt(n)就可以充分利用链路带宽 *

* Appenzeller G et al. Sizing router buffers. ACM SIGCOMM CCR, 2004

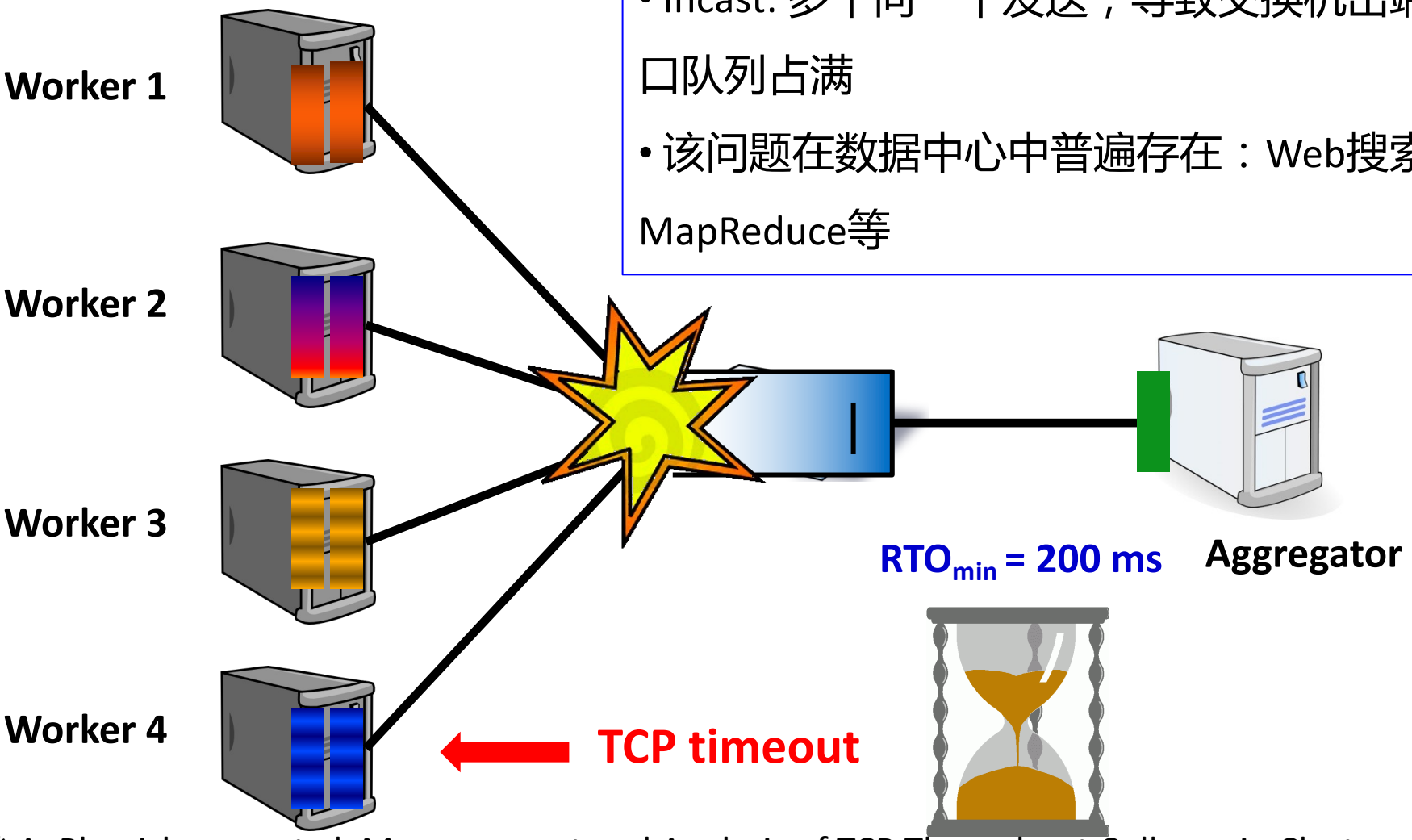
现实网络中的队列设置问题

- 数据中心网络的交换机队列
  - 交换机总的队列长度并不小（~10MB），端口数过多，业务高并发性
  - 队列相对过小 (under-buffered)造成了TCP-Incast问题

- 广域网的路由转发设备队列
  - 网络负载较大时，增大队列可降低丢包率，优化TCP传输速率 T~ $\frac{MSS}{\sqrt{loss} * RTT}$
  - 队列过大 (over-buffered)造成了BufferBloat问题

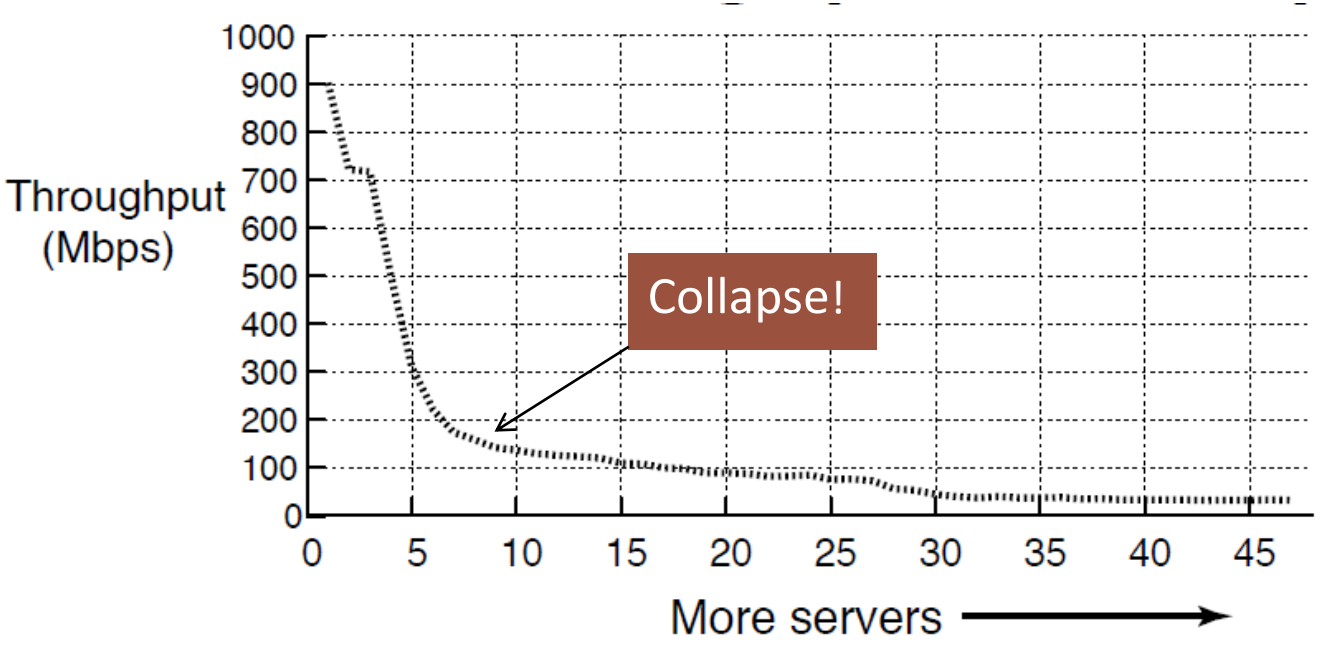
TCP Incast问题

- Incast: 多个向一个发送，导致交换机出端口队列占满
- 该问题在数据中心中普遍存在：Web搜索、MapReduce等



* A. Phanishayee, et al. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. FAST, 2008

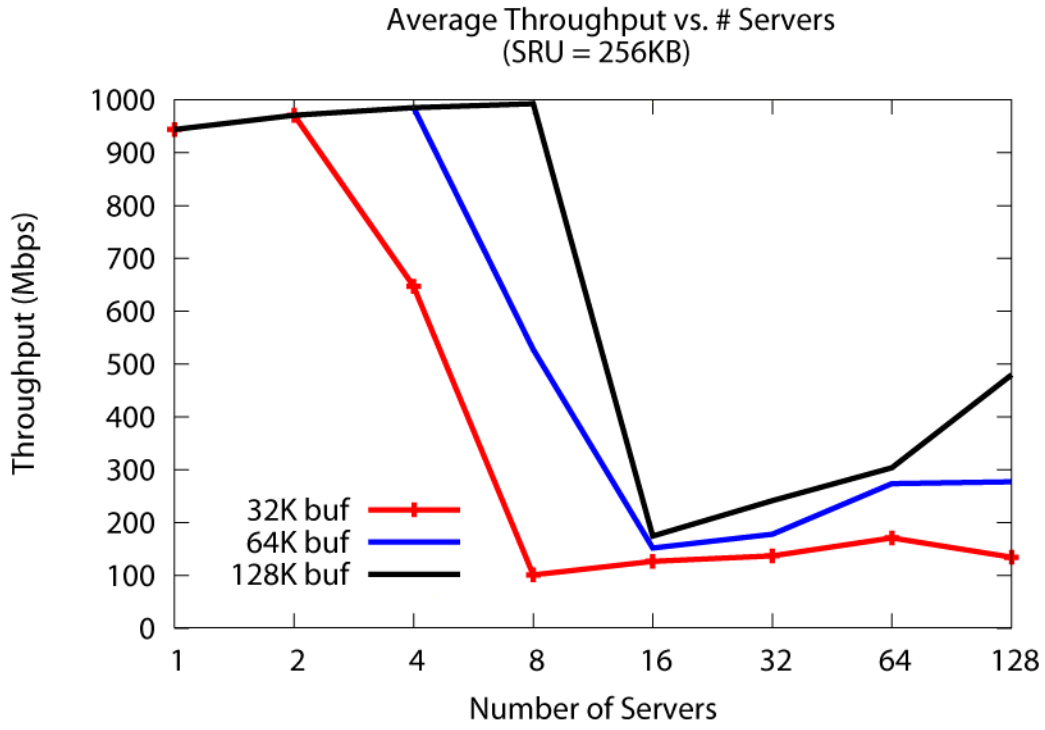
TCP Incast导致吞吐率下降



Cluster Setup
1Gbps Ethernet
Unmodified TCP
S50 Switch
1MB Block Size

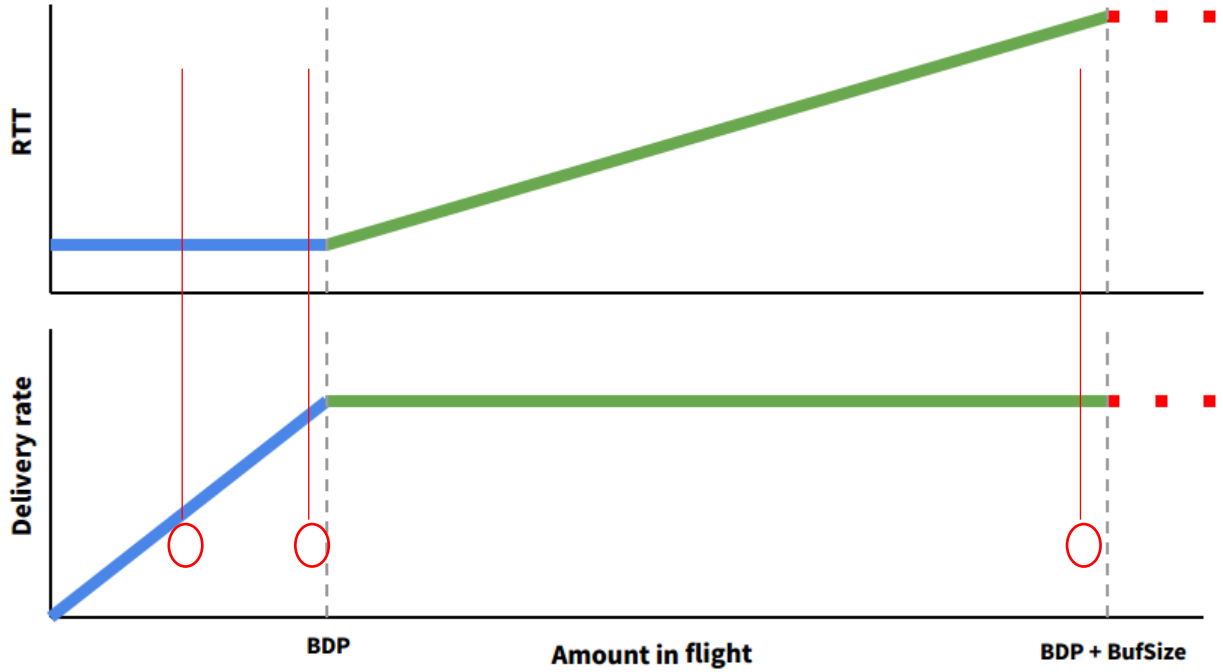
- TCP Incast造成传输下降的主要原因
  - 粗粒度、低效率的TCP丢包恢复机制

增大队列解决TCP Incast问题



- 可以支持多更Incast服务器数量，但队列硬件（SRAM）价格较高

BufferBloat问题

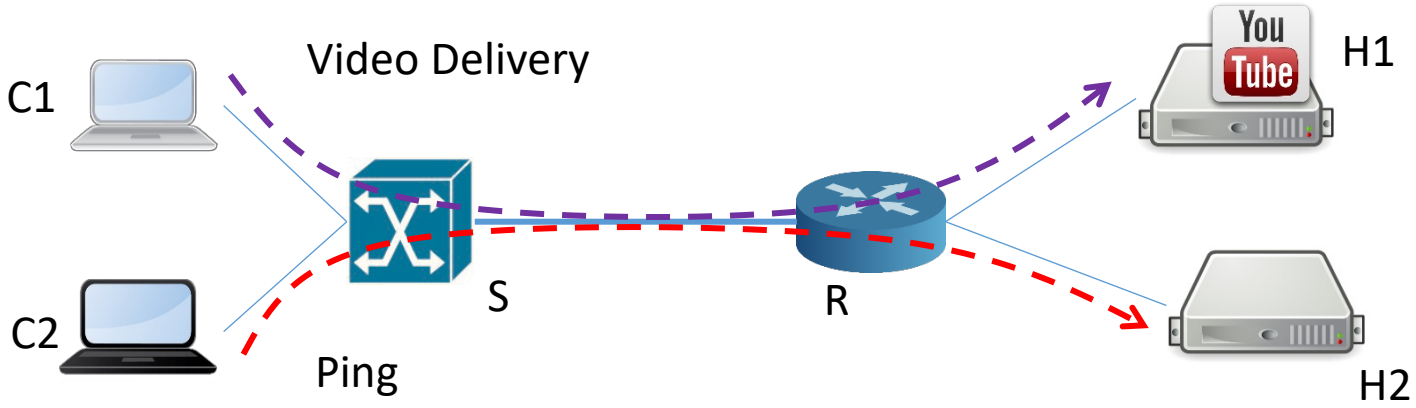


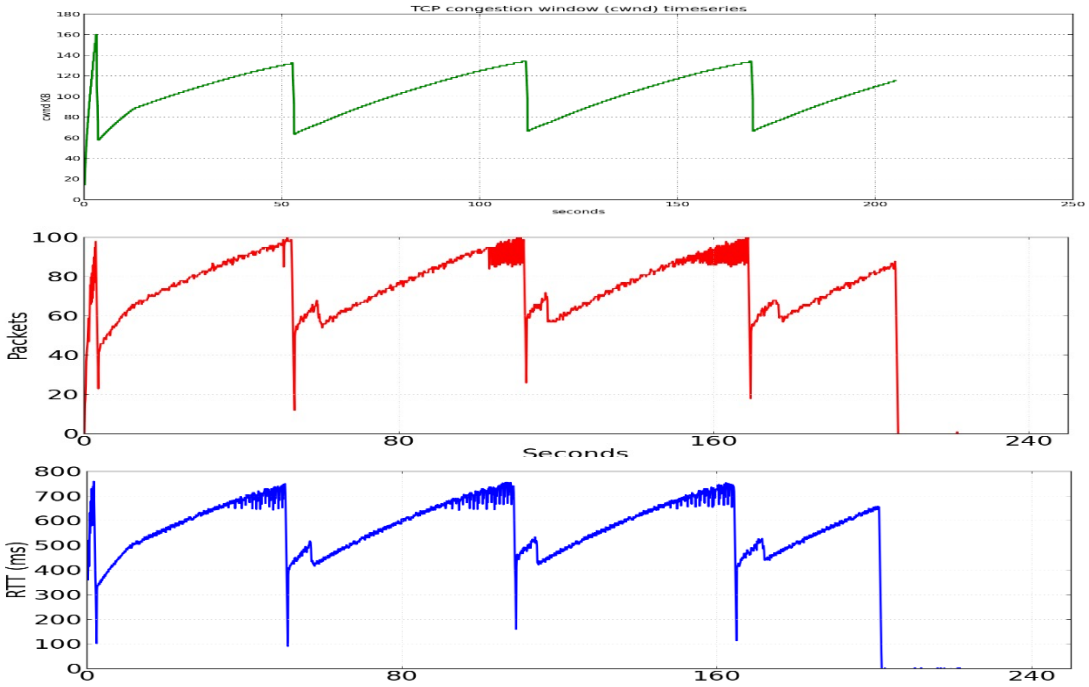
- BufferBloat是指数据包在缓冲区中存留时间过长引起的延迟过大问题

BufferBloat问题原因

- 设备的队列设置过大
  - 很难精确计算需要多大
  - 在成本允许的前提下，缓冲区越大越好
    - QoS、TCP吞吐率
- TCP传输机制
  - TCP传输协议的最初设计目标：改进吞吐率、优化带宽占用率
  - 机制：
    - 以丢包为拥塞控制信号
    - 只要没丢包，就会试图增加窗口大小：增加吞吐率
    - 当增大到BDP以后，窗口再增大，不会增加吞吐率，只会增加延迟
- 队列管理机制

BufferBloat问题重现





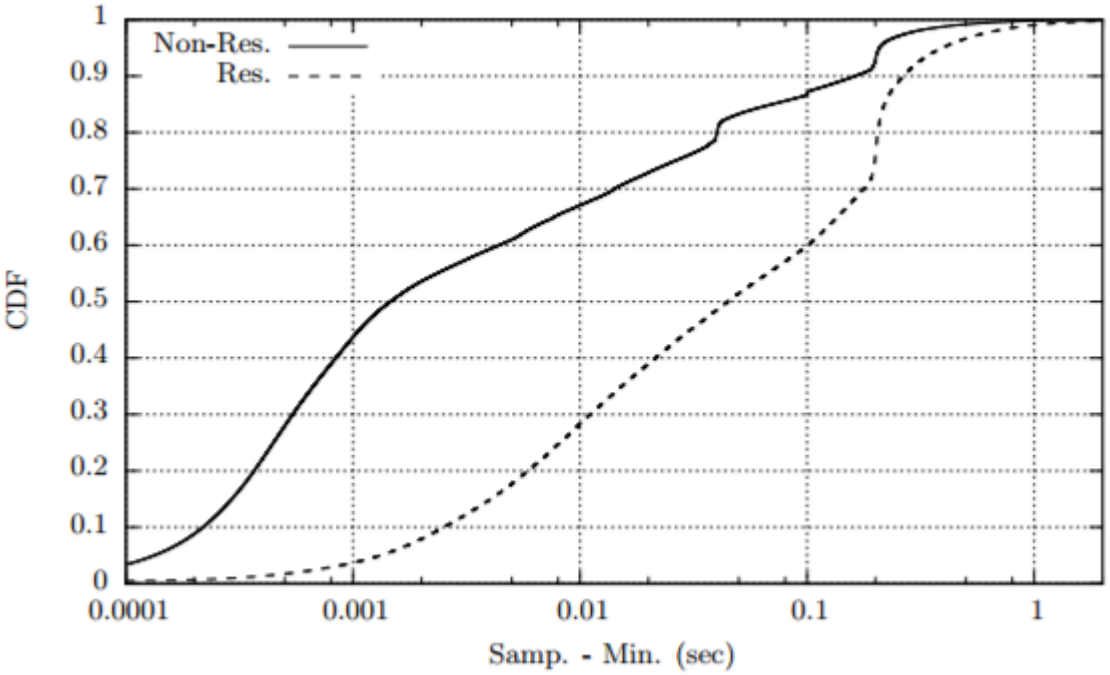
随着H1发送数据量（窗口大小）的增大，路由器队列中的数据包也随之增多，RTT相应变大

BufferBloat现象

- BufferBloat发生在
  - 网络负载较重的时间段，（不会一直持续）
  - 边缘网络，（该部分的队列大小更容易被错误配置）
  - 3G/4G网络，（运营商为了提升QoS等更容易部署大量队列）

```
64 bytes from rn-vps (168.235.98.214): icmp_seq=44 ttl=52 time=191 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=45 ttl=52 time=188 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=46 ttl=52 time=185 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=47 ttl=52 time=186 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=48 ttl=52 time=1759 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=51 ttl=52 time=1664 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=52 ttl=52 time=1639 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=55 ttl=52 time=1507 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=60 ttl=52 time=1103 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=65 ttl=52 time=187 ms
64 bytes from rn-vps (168.235.98.214): icmp_seq=67 ttl=52 time=186 ms
```

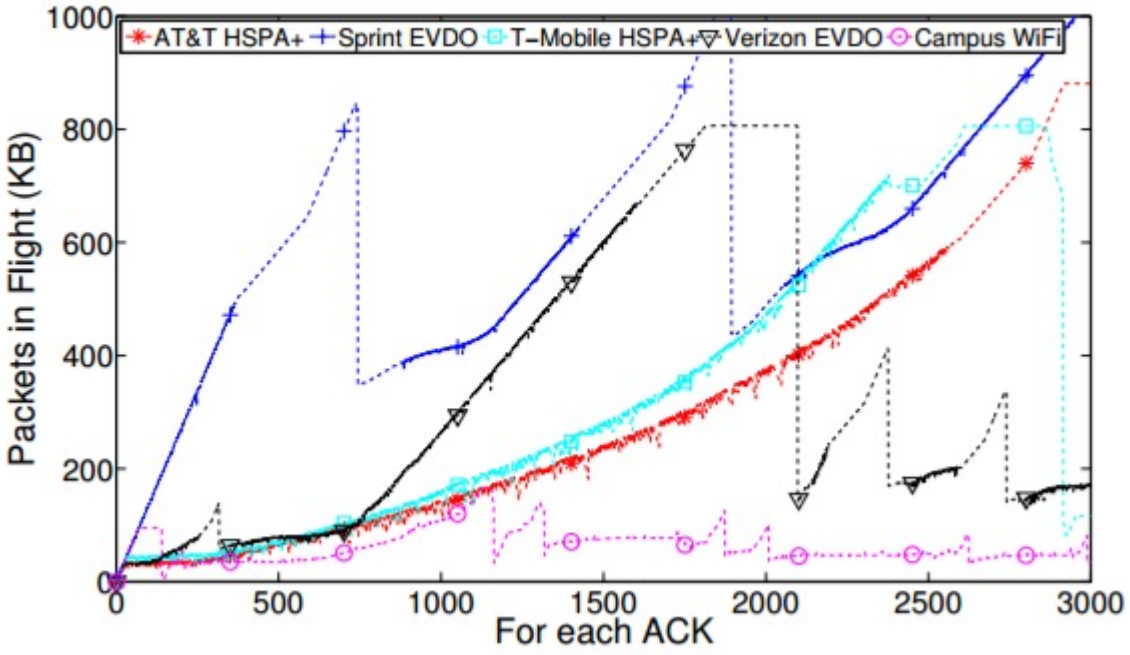
BufferBloat测量



[Allman2013]

- BufferBloat问题在边缘网络比在核心网络更明显
  - 边缘网络中，90%的BufferBloat延迟为30-40 ms
  - 秒级别的BufferBloat现象非常少见

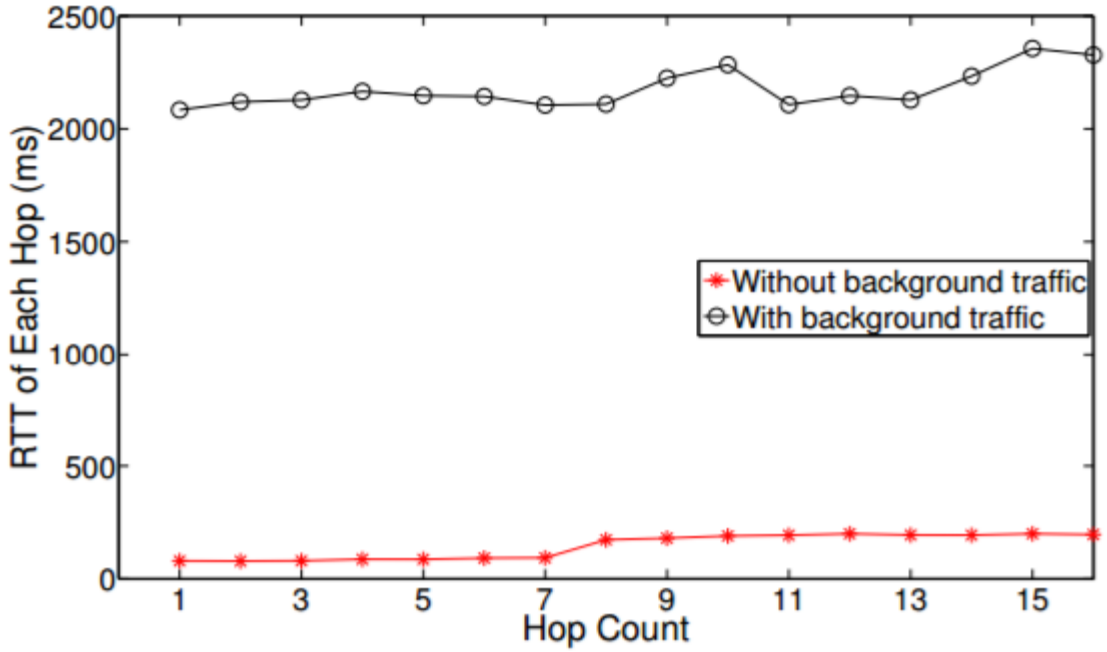
移动网络中的BufferBloat问题



[Jiang2012]

- 移动数据(3G/4G)网络中的BufferBloat问题更加严重
  - 即使下载峰值速率为3Mbps，那么最大延迟也能到达2秒

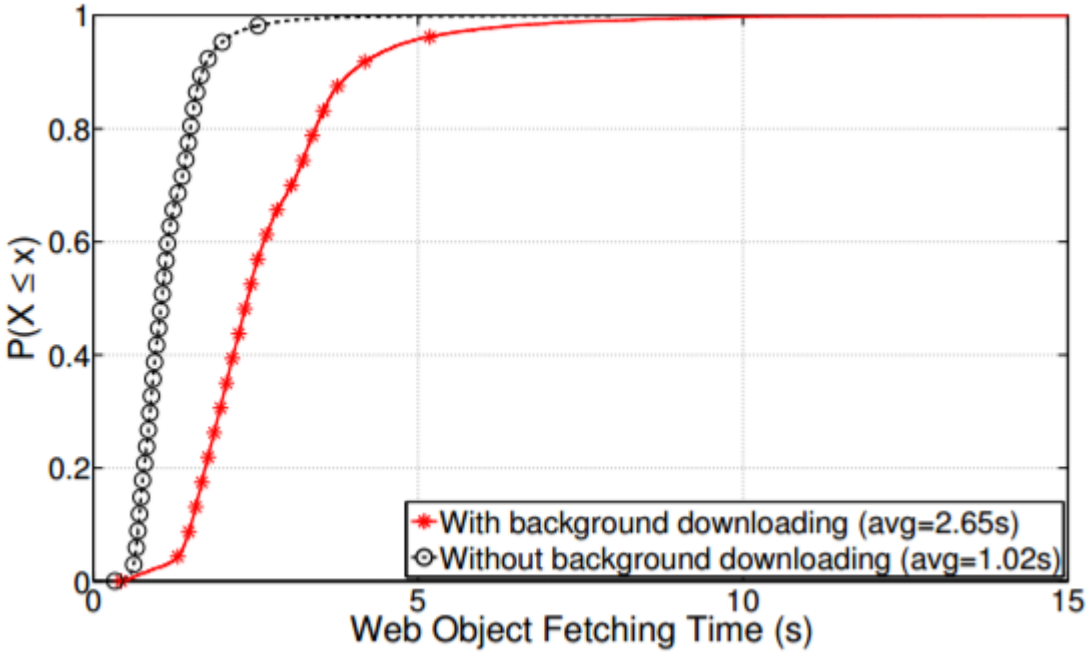
BufferBloat发生的位置



[Jiang2012]

- 通过traceroute分别测量有/无背景流量下每跳节点的RTT
  - 在移动数据网络中，BufferBloat通常发生在第一跳节点

BufferBloat问题对QoE的影响



[Jiang2012]

- 有背景流量时，BufferBloat引起的延迟增大使得Web页面下载时间增大了1.5倍
  - Web页面为小文件，下载时间与带宽关系不大，与RTT成正比

解决BufferBloat问题

- 减小队列大小
  - 减小队列大小可以降低数据包在队列中的时间
  - 但是，小队列难以容忍突发流量
- 改进传输控制策略
  - 丢包不再是TCP传输控制的唯一信号，也考虑延迟变化
- 改进队列管理策略
  - 在队列满之前就主动（概率性的）丢包
    - RED (Random Early Detection)
  - 以延迟作为队列管理的信号
    - CoDel (Controlled Delay)

默认队列管理机制：Tail Drop (尾部丢弃)

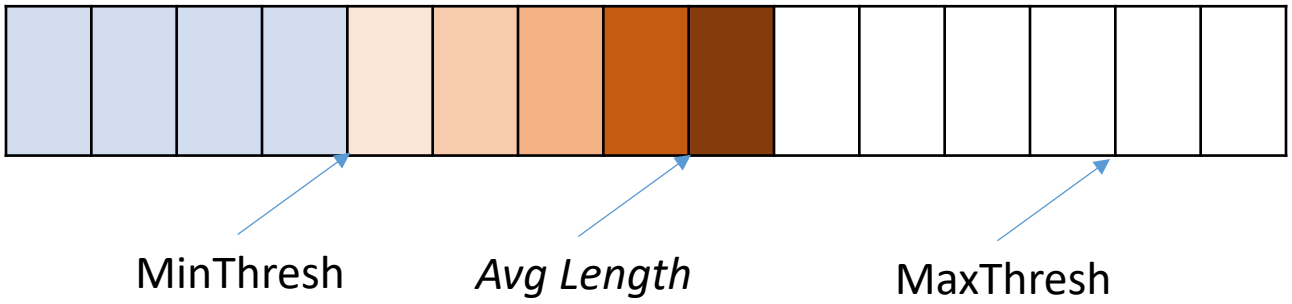
- 当队列满时，将新到达的数据包丢弃
  - 最简单的队列管理机制
- 原则:
  - 中间设备的功能尽可能简单，由端设备负责拥塞控制

- 通常和FIFO组合使用
  - 是最简单的排队算法
  - 也是目前应用最广泛的排队算法
    - 不需要设置任何参数
    - 简单意味着可靠

RED (Random Early Detection)

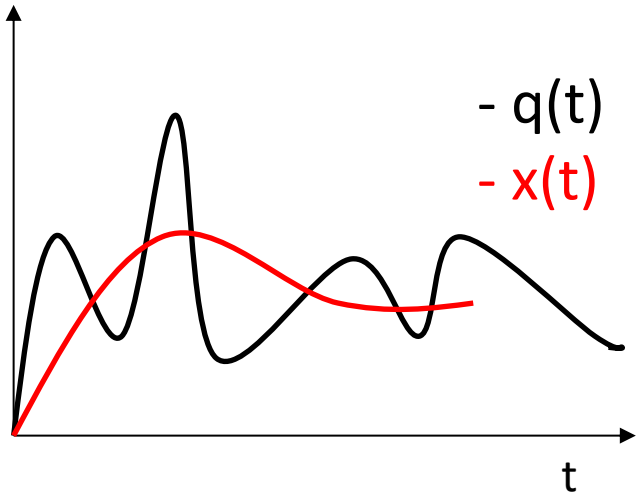
- 动机
  - 高BDP流通常需要较大的队列来适应Burst（突发流量）
  - 但是队列大小增加时，延迟也会增大
- 设计目标
  - 高吞吐率、低延迟
- 设计思路
  - 在队列满之前，就开始主动(proactively)丢包 (Early Detection)
    - 提醒发送方即将到来的拥塞
  - 概率性的丢包，丢包概率与队列长度正相关 (Random)

RED操作

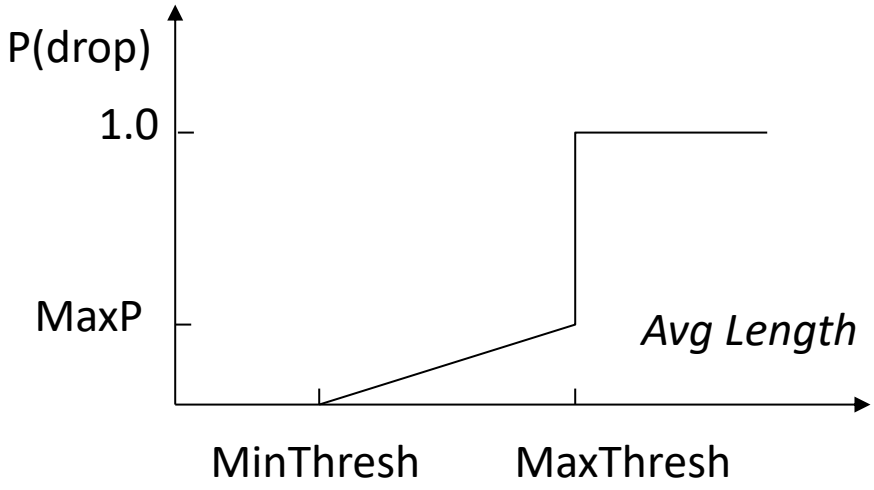


(1) 使用平滑函数计算平均队列长度x(t)

- $x(t) <- (1 - W_q) * x(t - T) + W_q * q(t)$



(2) 根据平均队列长度x(t)进行概率丢包



RED主要问题

- 需要设置很多参数
  - MinThresh、 MaxThresh、W_q、 MaxP、采样周期、...
- 性能对参数设置很敏感
  - 调优非常困难
  - 不恰当的设置可能比Tail Drop性能更差
- 自1993年提出以来，路由器都支持，但基本上都被关掉了

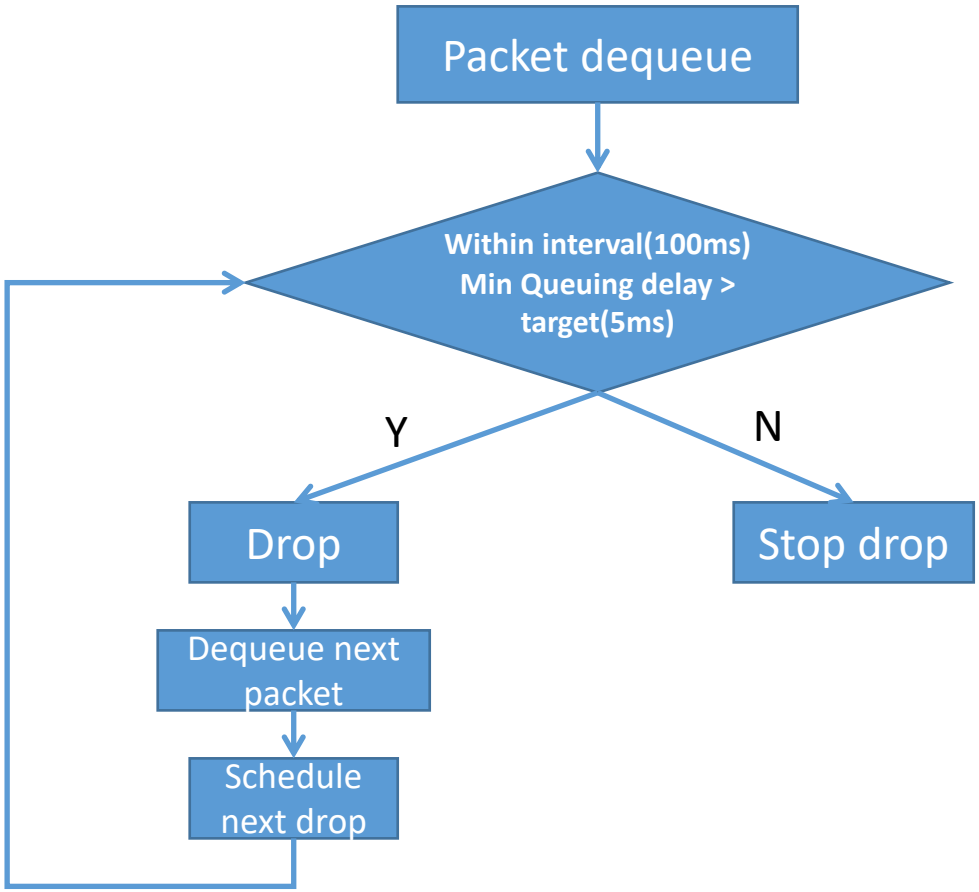
CoDel (Controlled Delay)

- BufferBloat问题
  - 设备制造商为了减少网络丢包，通常会配置很大的队列
  - 当网络负载很大时，网络延迟会变得很大（秒级别）
- CoDel设计目标
  - 减少大队列下的延迟
  - 对RTT、速率、负载不敏感
- CoDel核心思想
  - 控制数据包在队列中的时间（延迟），而不是队列长度

CoDel设计

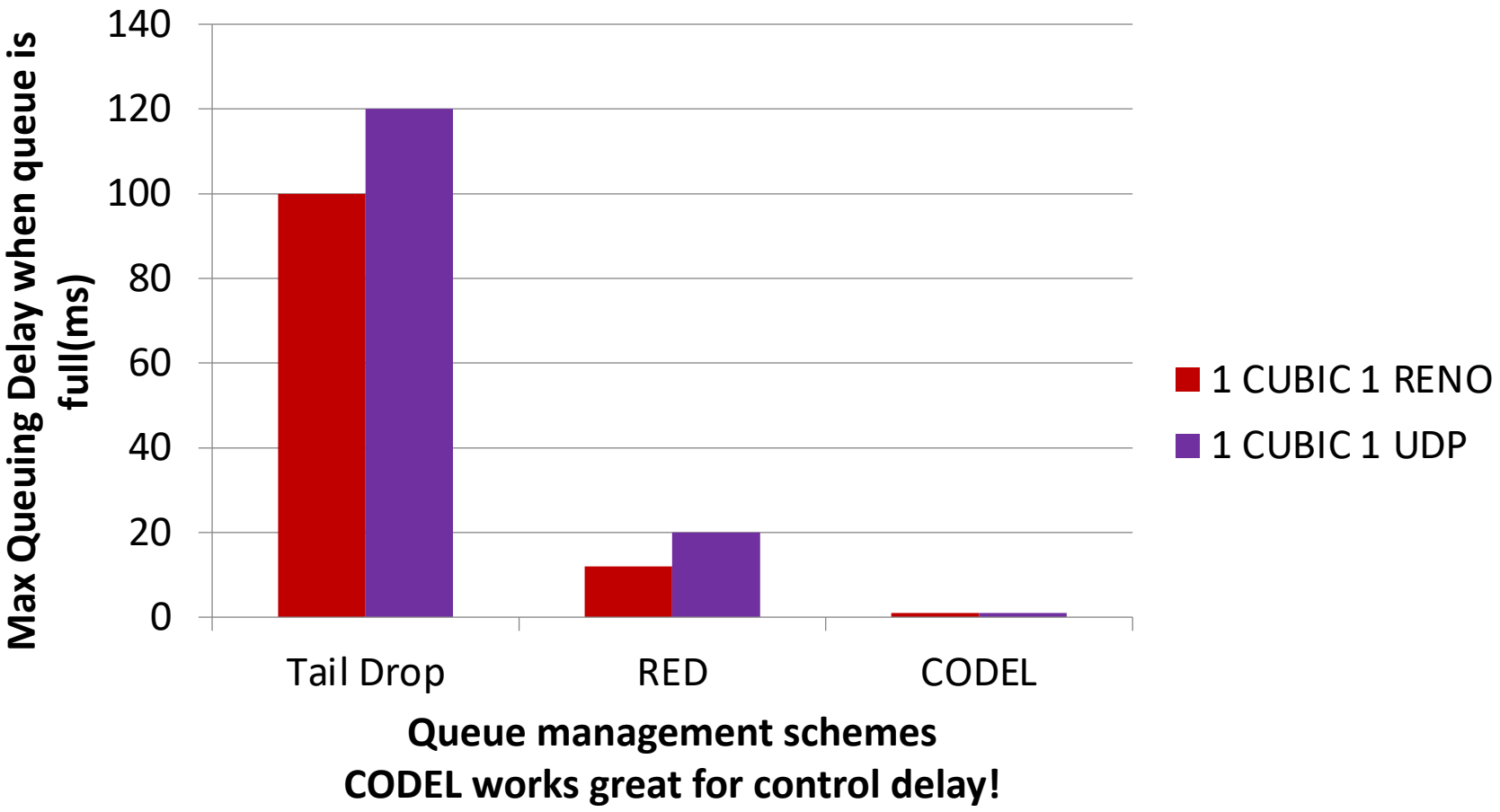
- CoDel设计思路
  - 使用包在队列中的停留时间作为度量指标
    - 而不是队列长度
- CoDel算法
  - 当包停留时间超过target值时
    - 将该数据包丢弃
    - 并根据control law设置下次丢包时间
      - interval / sqrt(count)
  - 直到所有包的停留时间都小于target值

CoDel流程示意

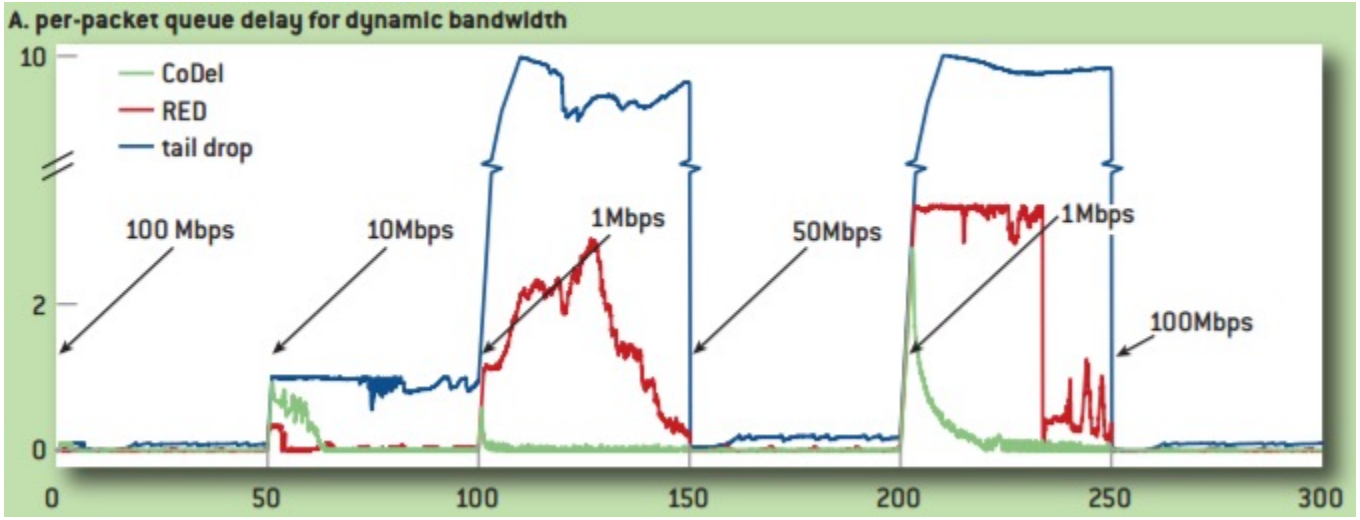


- CoDel具有Tail Drop类似的优点：不需要配置参数
  - 参数硬编码到CoDel机制中，但不一定是最优的
- CoDel可以减少延迟，但一般不会提升吞吐率

CoDel延迟性能分析



CoDel在移动网络环境下的性能



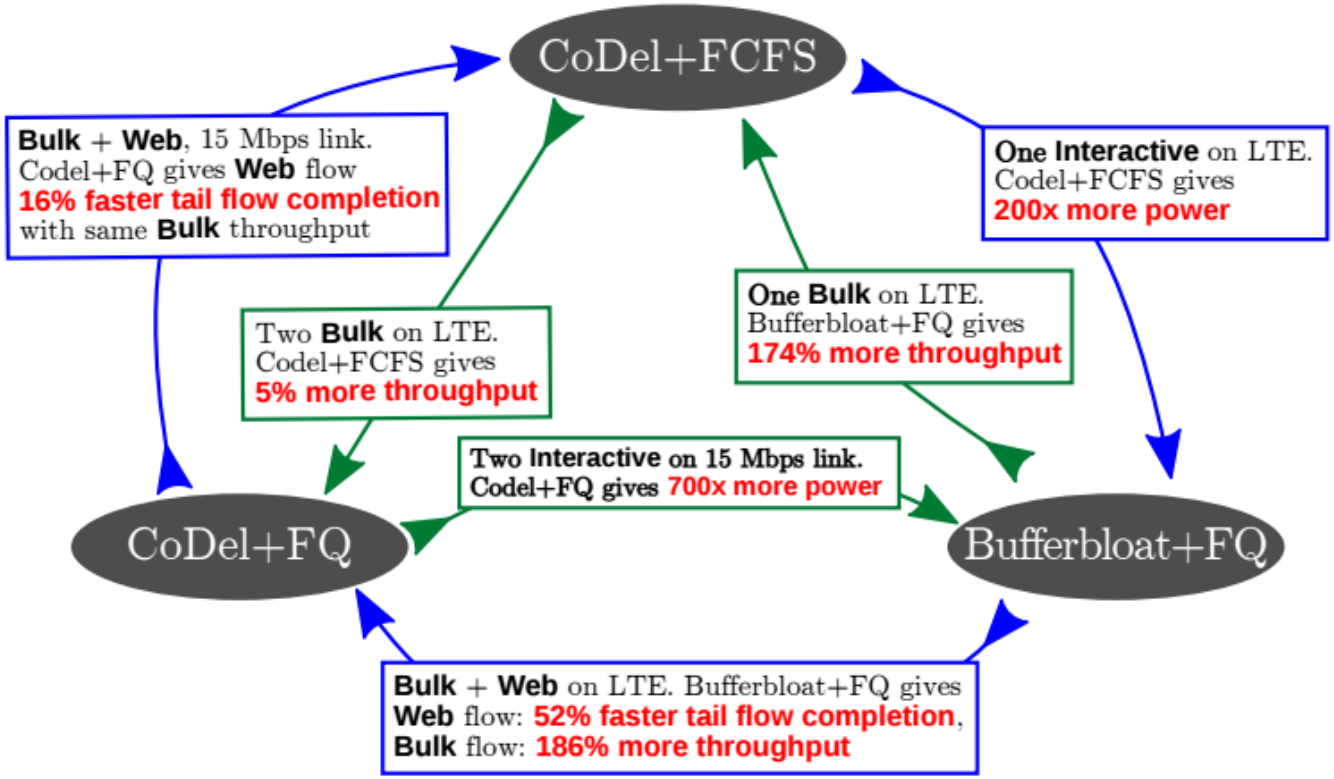
- 移动网络中的带宽容量会一直变化
  - Tail Drop在队列满之前不会丢包，当带宽变小时，需要很长时间才能将数据包发送出去，造成了很大的延迟
  - RED虽然可以保证队列中存在较少的数据包，但对带宽变化（延迟变化）反应不够快

数据包队列对传输性能的影响

- 网络中间设备的排队算法
  - 队列大小
    - 队列较小时，如果存在多对一传输，会引起TCP Incast问题
    - 队列很大时，可以减少丢包，但会引起BufferBloat问题
  - 队列管理
    - Tail Drop、 RED、 CoDel有不同的丢包行为
    - 丢包次数: Tail Drop < RED < CoDel
    - 延迟：Tail Drop > RED > CoDel
  - 队列调度
    - FIFO: 实现简单，通常会有更高吞吐率
    - FairQueue: 提升不同流之间的公平性

队列管理与传输性能

- 没有一种策略在所有网络环境下是最优的



* A. Sivaraman, et al. No silver bullet: extending SDN to the data plane. ACM HotNets 2013.

数据包队列总结

- 队列是网络设备中的关键部分
  - 其大小、管理策略影响了网络性能
- 队列设置过大或过小都容易产生问题
  - 过小 -> TCP Incast问题
  - 过大 -> BufferBloat问题
- 队列大小、队列管理策略、传输控制策略之间的关系非常紧密
  - 解决TCP Incast和BufferBloat问题都应该从上述三个维度展开
  - 由于应用QoE的复杂性，没有一种组合能够在所有场景下均达到最优性能

课后阅读

- 《计算机网络－系统方法》
  - 第3.1、3.2、4.1节
- 数据中心网络种的交换网络拓扑
  - Mohammad Al-Fares et al. A scalable, commodity data center network architecture. ACM SIGCOMM 2008, pp 63 -- 74
- 网络增量部署和过渡
  - Matthew Mukerjee et al. Understanding tradeoffs in incremental deployment of new network architectures. ACM CoNEXT 2013, 271 -- 282

谢谢！