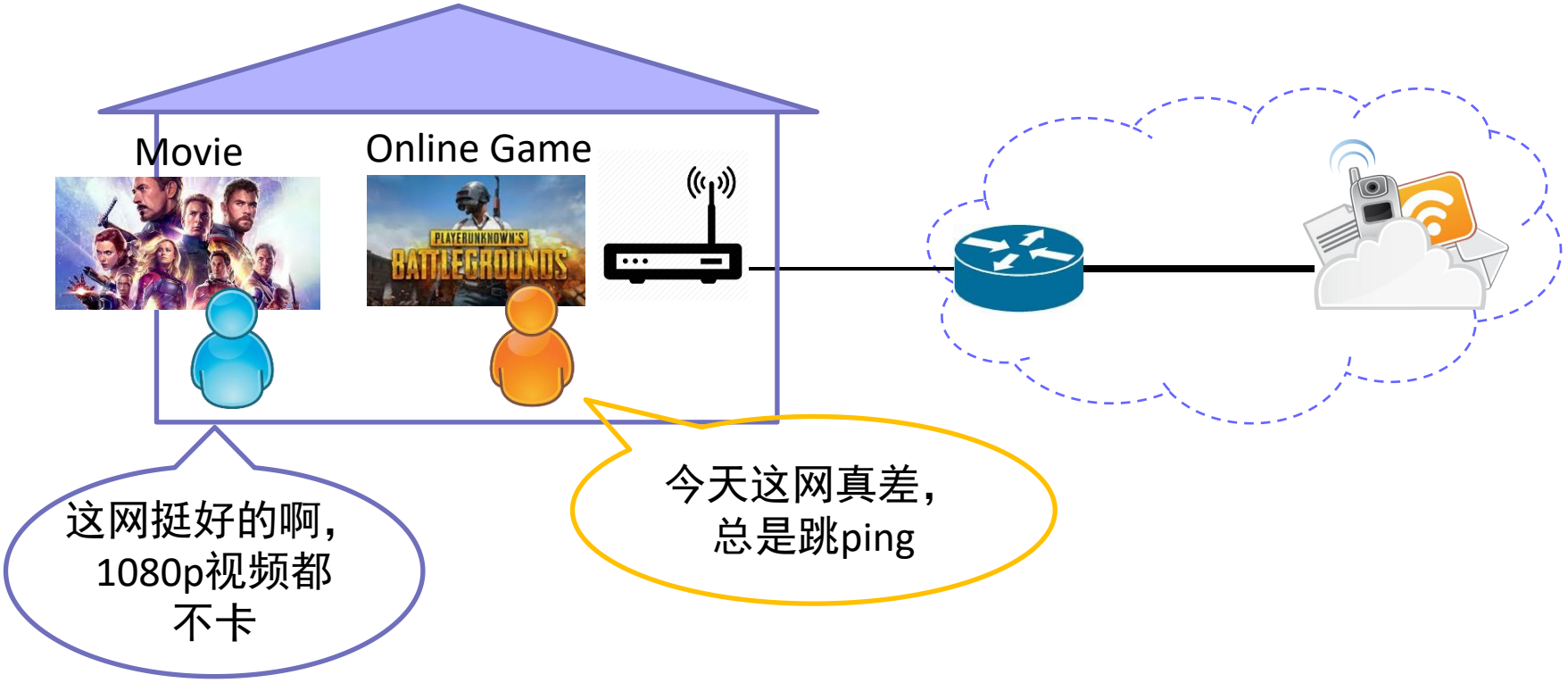
数据包队列管理实验

武庆华

wuqinghua@ict.ac.cn

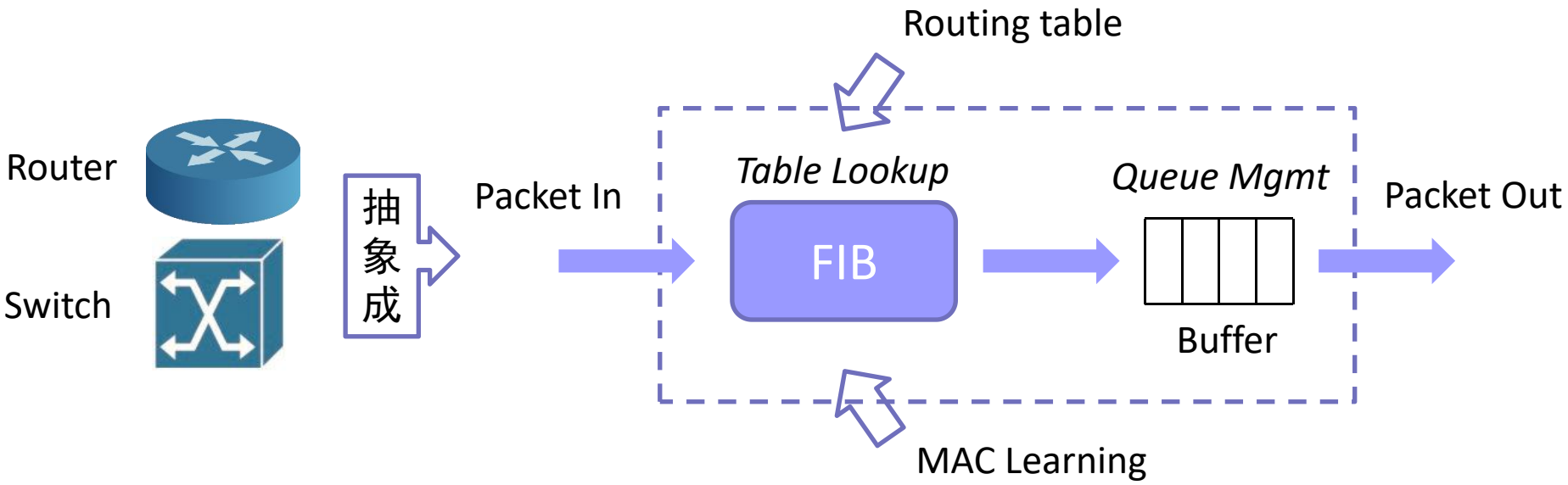
问题背景


Movie
Online Game
PLAYERUNKNOWN'S
BATTLEGROUNDS
这网挺好的啊，
1080p视频都
不卡
今天这网真差，
总是跳ping

提纲

- 数据包队列
  - 为什么要有数据包队列?
  - 数据包队列大小设置
  - 数据包队列过大或过小引起的问题
- BufferBloat问题
  - BufferBloat问题现象与解决思路
  - 数据包队列管理机制
- BufferBloat实验

数据包队列

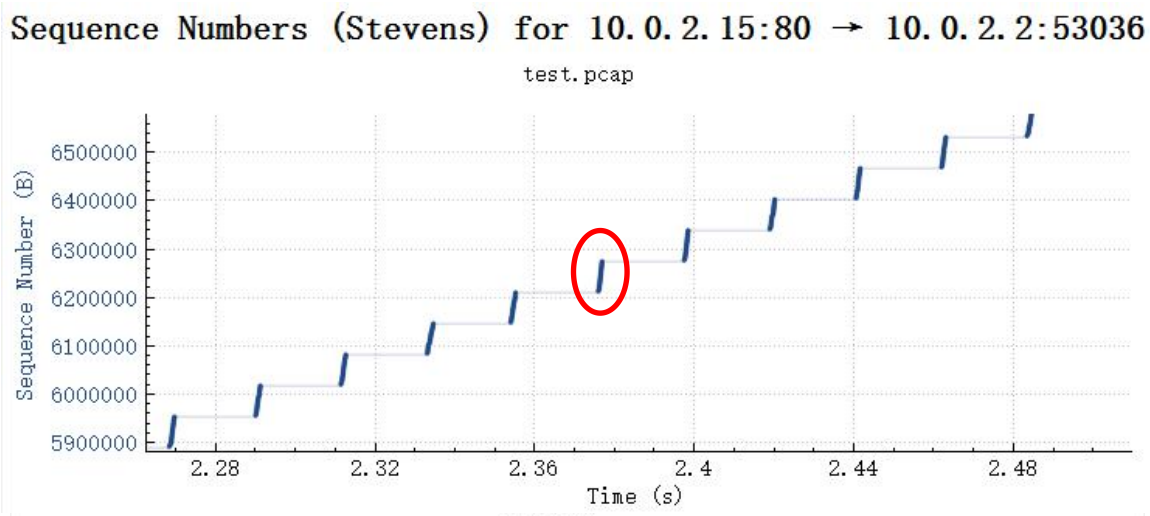


- 数据包队列是网络中间设备中最关键的部分之一
  - 其大小、管理机制等很大程度上影响了网络性能

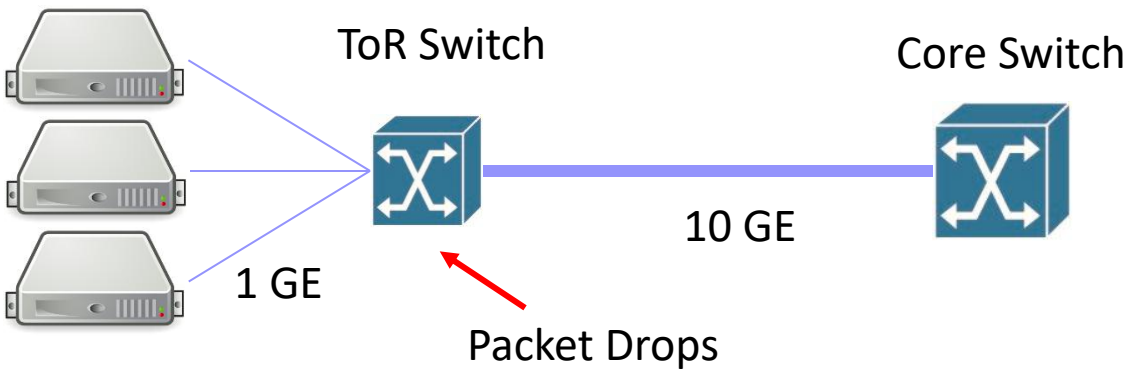
注：下文在不引起歧义的情况下，用数据包队列/队列表示Buffer

为什么需要数据包队列?

突发流量 (Burst)

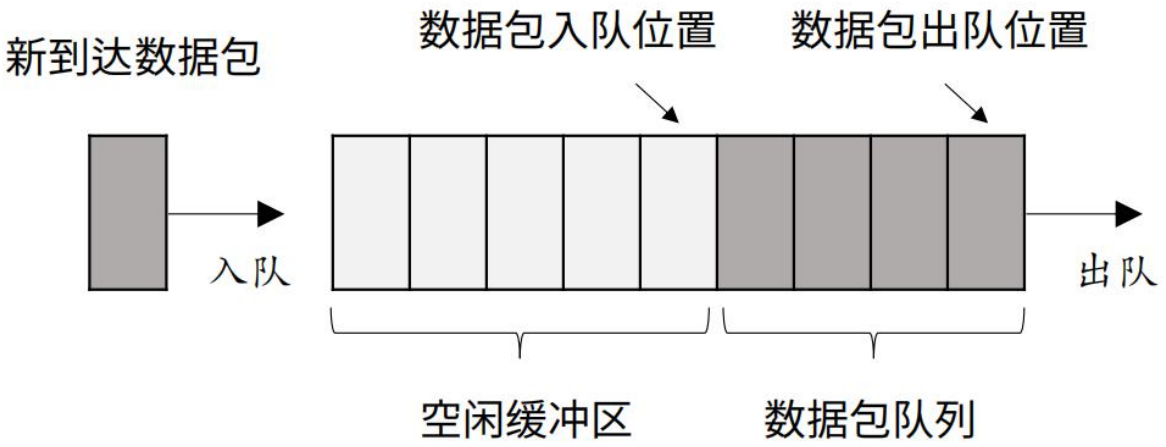


瓶颈链路

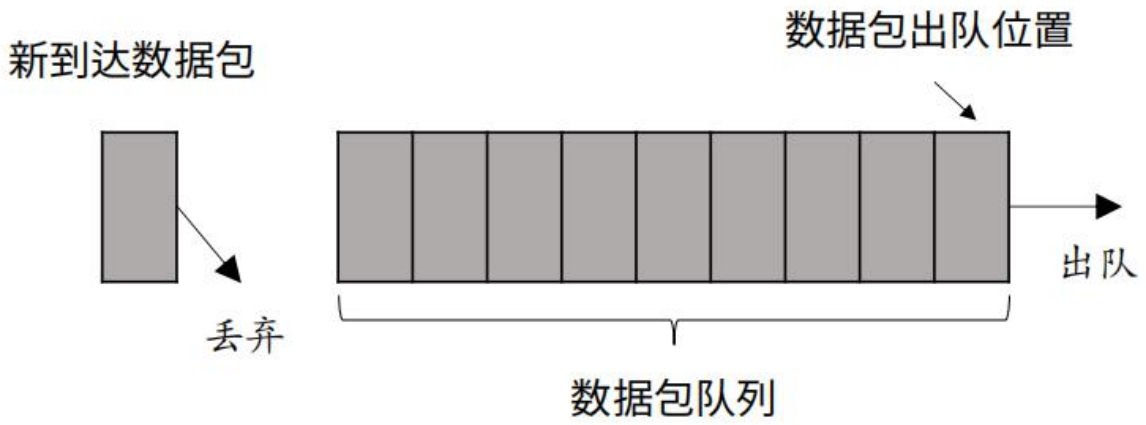


数据包队列如何工作？

调度：先进先出
(First In, First Out)



管理：尾部丢弃
(Tail Drop)



数据包队列如何工作？

- 通常Tail Drop与FIFO组合使用
  - 最简单的队列管理和调度机制：不需要设置任何参数
  - 也是目前应用最广泛的：简单意味着可靠
- 原则:
  - 中间设备的功能尽可能简单，由端设备负责拥塞控制

- 其他队列调度机制：
  - 公平队列，带权重的公平队列等
- 其他队列管理机制：
  - RED, WRED, CoDel, Choke, ...

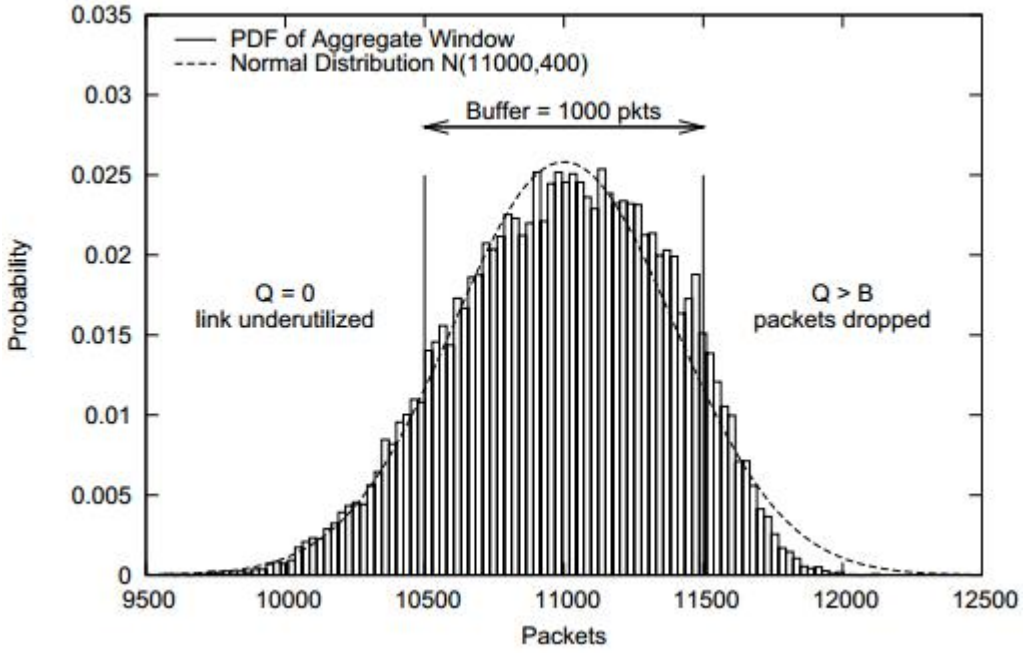
队列应该设置成多大？

- 经验法则 [Appenzeller2004]
  - $BuffSize = \overline{RTT} * C$
    - 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：200Mbps
  - 网络延迟：10 ~ 50ms (30ms)
  - 家庭网关设备的数据包队列大小：200Mbps * 30ms = 0.75MB

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

队列大小分析

- $Q(t) = \sum_i W_i(t) - BDP$, 其中$BDP$为在瓶颈带宽（路径中）传输的数据包个数（$\overline{RTT} * C$）
- 每条流的窗口大小服从均匀分布: $W_i \sim Unif\left(\frac{2}{3}\overline{w_i}, \frac{4}{3}\overline{w_i}\right)$
- 窗口大小的标准差为 $\sigma_{W_i} = \frac{1}{\sqrt{12}}\left(\frac{4}{3}\overline{w_i} - \frac{2}{3}\overline{w_i}\right) = \frac{1}{3\sqrt{3}}\overline{w_i}$
- 对于n条流，$\sigma_W \leq \sum_i \frac{\sigma_{W_i}}{\sqrt{n}}$
- 因此，队列（Queue）长度的标准差
  - $\sigma_Q = \sigma_W \leq \frac{1}{3\sqrt{3}} \frac{B+\mathrm{BDP}}{\sqrt{n}}$

给定队列大小下的链路利用率

- 链路带宽充分利用的概率 $\geq Erf\left(\frac{3\sqrt{3}}{2\sqrt{2}}\frac{B}{\frac{\mathrm{BDP+B}}{\sqrt{n}}}\right)$ [Appenzeller2004]
- N = 10000时的数值仿真实验

队列大小	链路利用率
$\mathrm{B} = 1 * \frac{\mathrm{BDP}}{\sqrt{n}}$	> 98.99 %
$\mathrm{B} = 1.5 * \frac{\mathrm{BDP}}{\sqrt{n}}$	> 99.99988 %
$\mathrm{B} = 2 * \frac{\mathrm{BDP}}{\sqrt{n}}$	> 99.99997 %

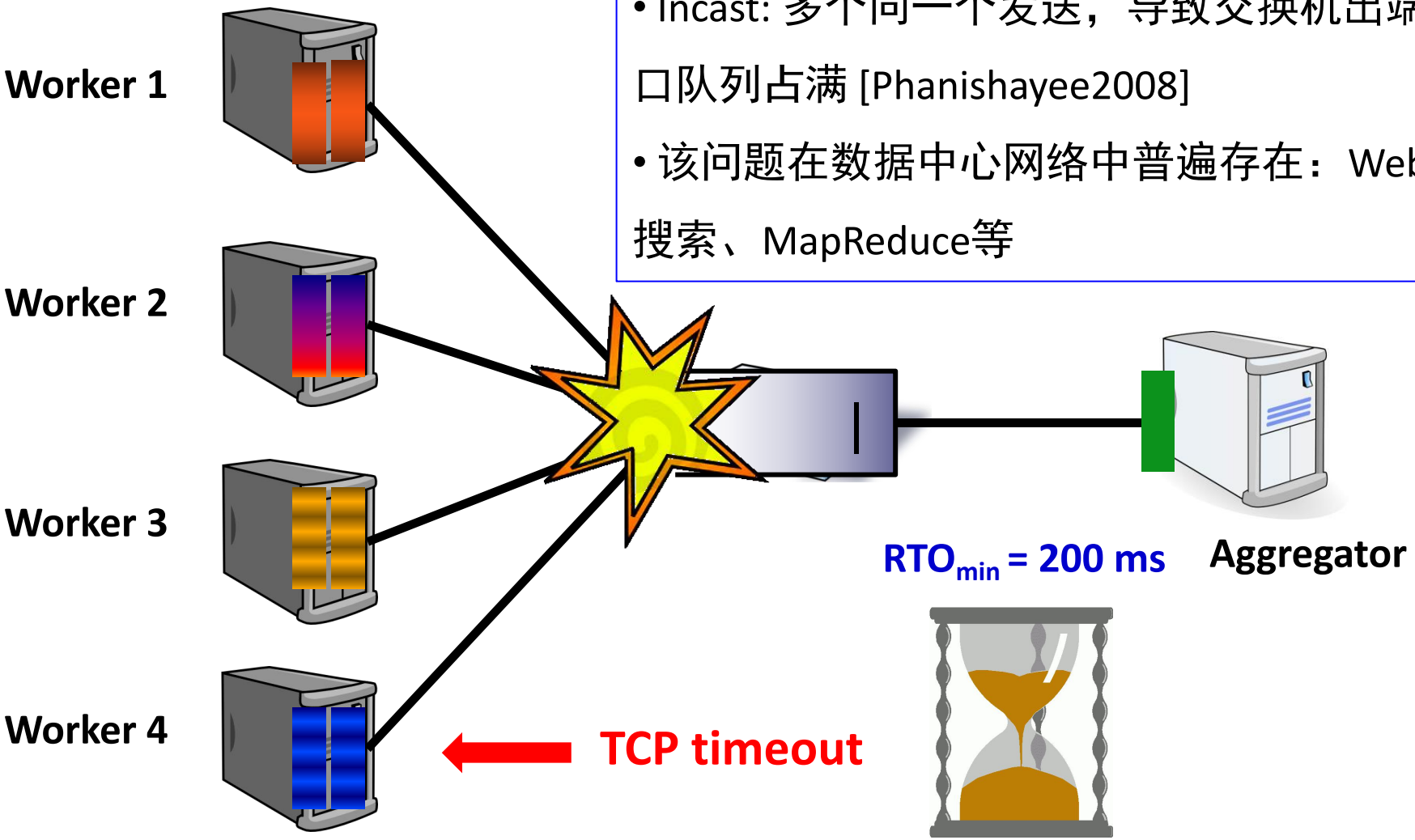
- 该结果说明，对于多流经过的路由器，其队列大小只需设置成BDP/sqrt(n)就可以充分利用链路带宽

现实中的队列设置问题

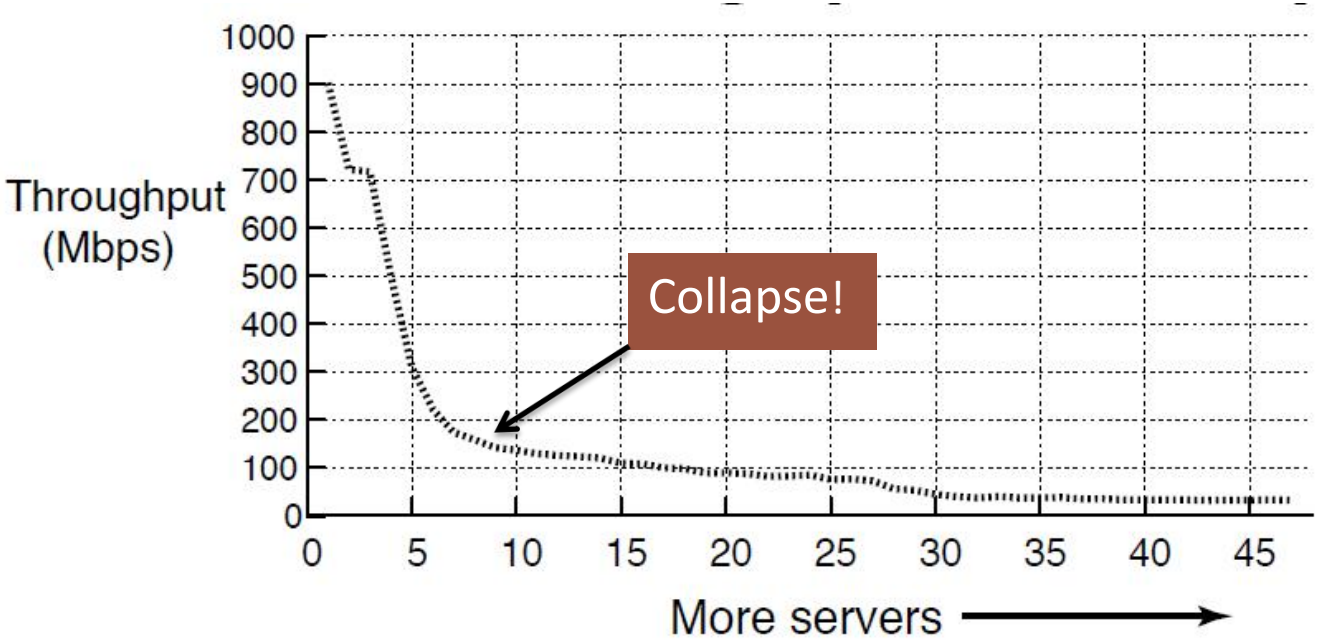
- 现实中的队列大小通常设置的比理论值大很多
  - 现实网络中的并发流不一定是异步的
  - 现实中也有很多短流
  - 设备商将队列扩容的目的
    - 网络负载较大时，降低丢包率，优化QoS
    - 通过工程手段优化TCP传输速率 T~$\frac{MSS}{\sqrt{loss}*RTT}$
- 现实中的队列大小设置会面临两个问题
  - 队列过小 (under-buffered): 数据中心网络的TCP-Incast问题
  - 队列过大 (over-buffered): 广域网边缘的BufferBloat问题

TCP Incast问题

- 交换机队列一般较小，且多个接口共享
- Incast: 多个向一个发送，导致交换机出端口队列占满 [Phanishayee2008]
- 该问题在数据中心网络中普遍存在：Web搜索、MapReduce等



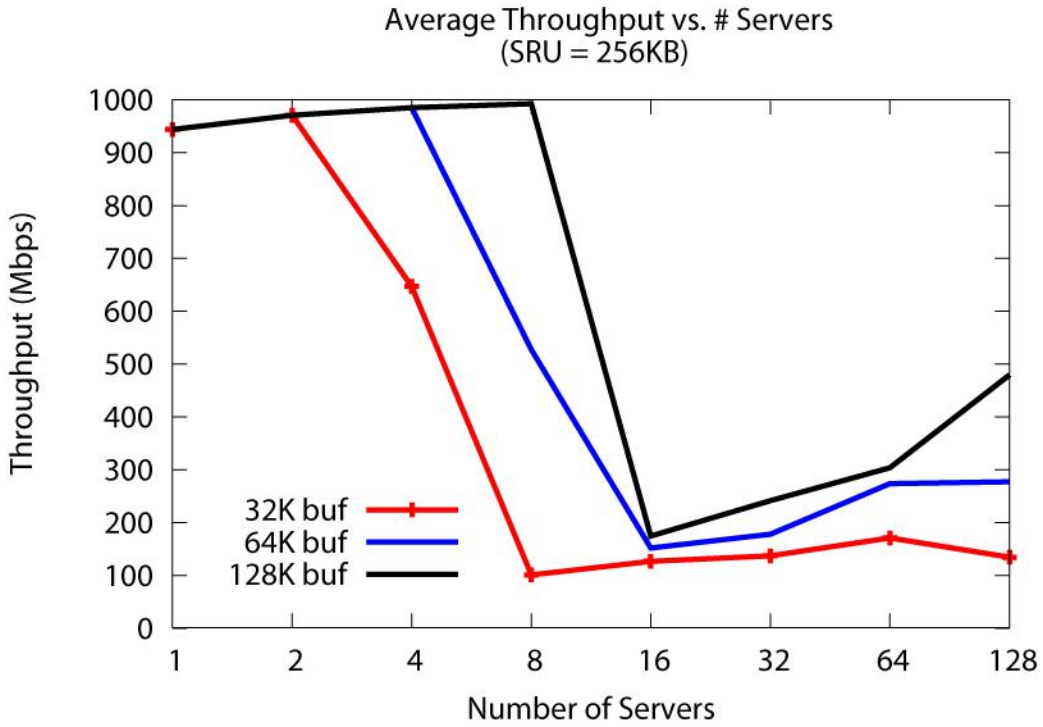
TCP Incast导致吞吐率下降



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

- TCP Incast造成传输下降的主要原因
  - 设备队列过小，难以容忍高并发数据包
  - 粗粒度、低效率的TCP丢包恢复机制

增大队列解决TCP Incast问题



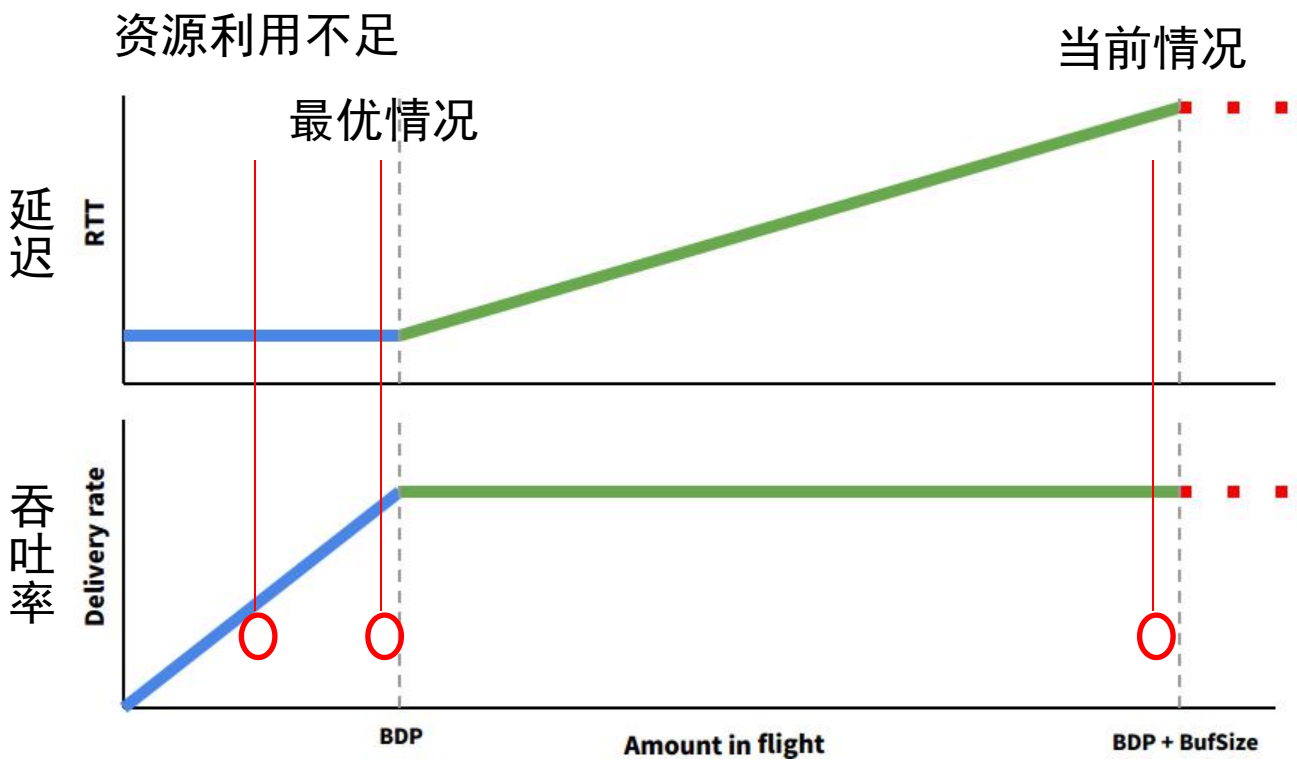
- 优势：可以支持多更Incast服务器数量
- 劣势：队列硬件（SRAM）价格较高

BufferBloat问题

- BufferBloat是指数据包在队列中存留时间过长引起的延迟过大问题[Gettys2011]
- 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）

```
icmp_seq=44 ttl=52 time=16 ms
icmp_seq=45 ttl=52 time=13 ms
icmp_seq=46 ttl=52 time=10 ms
icmp_seq=47 ttl=52 time=11 ms
icmp_seq=48 ttl=52 time=1584 ms
icmp_seq=51 ttl=52 time=1489 ms
icmp_seq=52 ttl=52 time=1464 ms
icmp_seq=55 ttl=52 time=1332 ms
icmp_seq=60 ttl=52 time=928 ms
icmp_seq=65 ttl=52 time=12 ms
icmp_seq=67 ttl=52 time=11 ms
```

BufferBloat问题本质



单位时间内的数据包发送量

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，队列设置的越大越好
    - 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：1、以丢包为拥塞控制信号；2、只要没丢包，就会试图增加窗口大小，增加吞吐率；3、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
  - 当Tail Drop开始丢包时，网络已经很拥塞了，延迟非常大

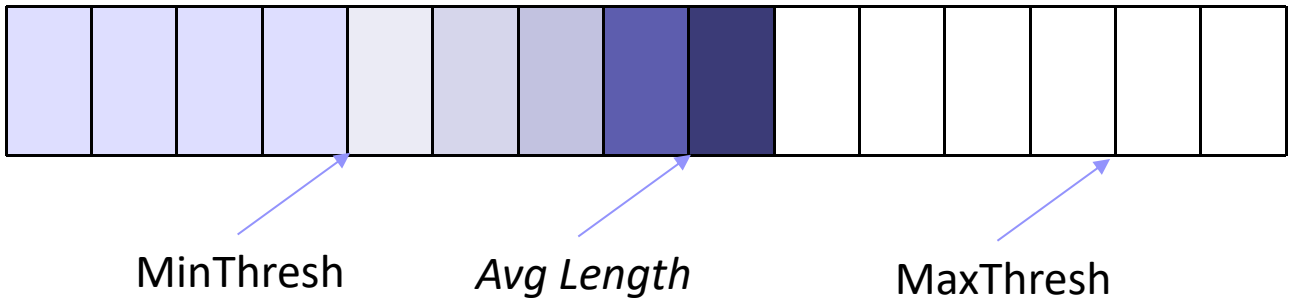
解决BufferBloat问题

- 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 [Cardwell2016]
- 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)

RED (Random Early Detection)

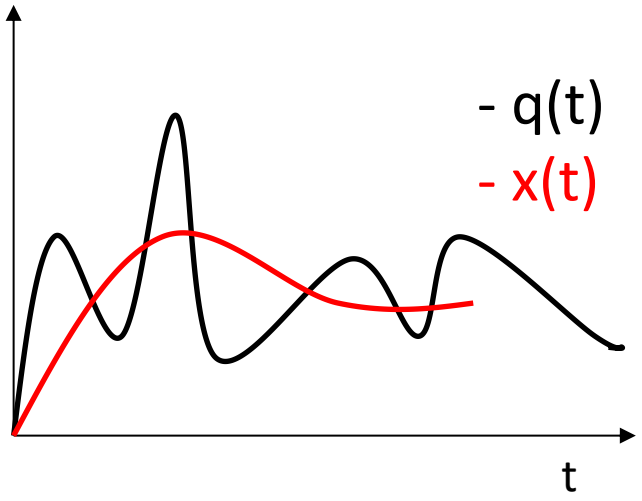
- 动机
  - 高BDP流通常需要较大的队列来适应Burst（突发流量）
  - 但是队列大小增加时，延迟也会增大
- 设计目标
  - 高吞吐率、低延迟
- 设计思路 [Floyd1993]
  - 在队列满之前，就开始主动(proactively)丢包 (Early Detection)
    - 提醒发送方即将到来的拥塞
  - 概率性的丢包，丢包概率与队列长度正相关 (Random)

RED操作

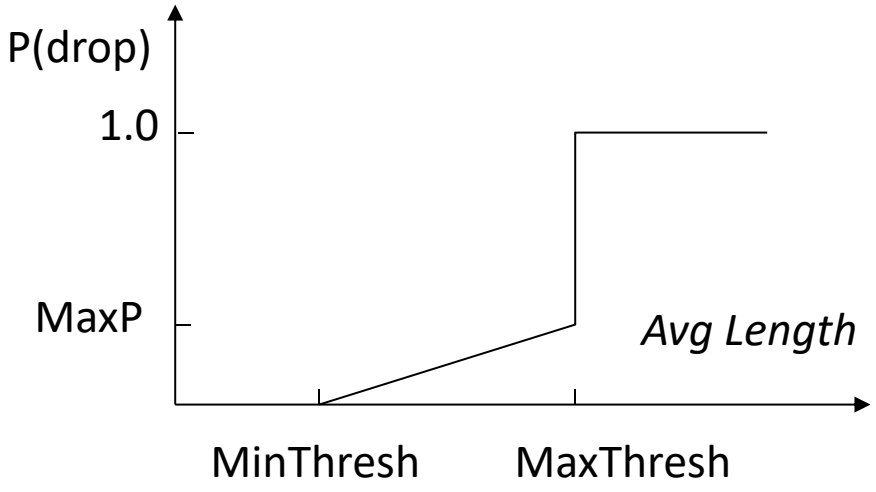


(1) 使用平滑函数计算平均队列长度x(t)

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



(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 [Jacobson1999]
  - 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

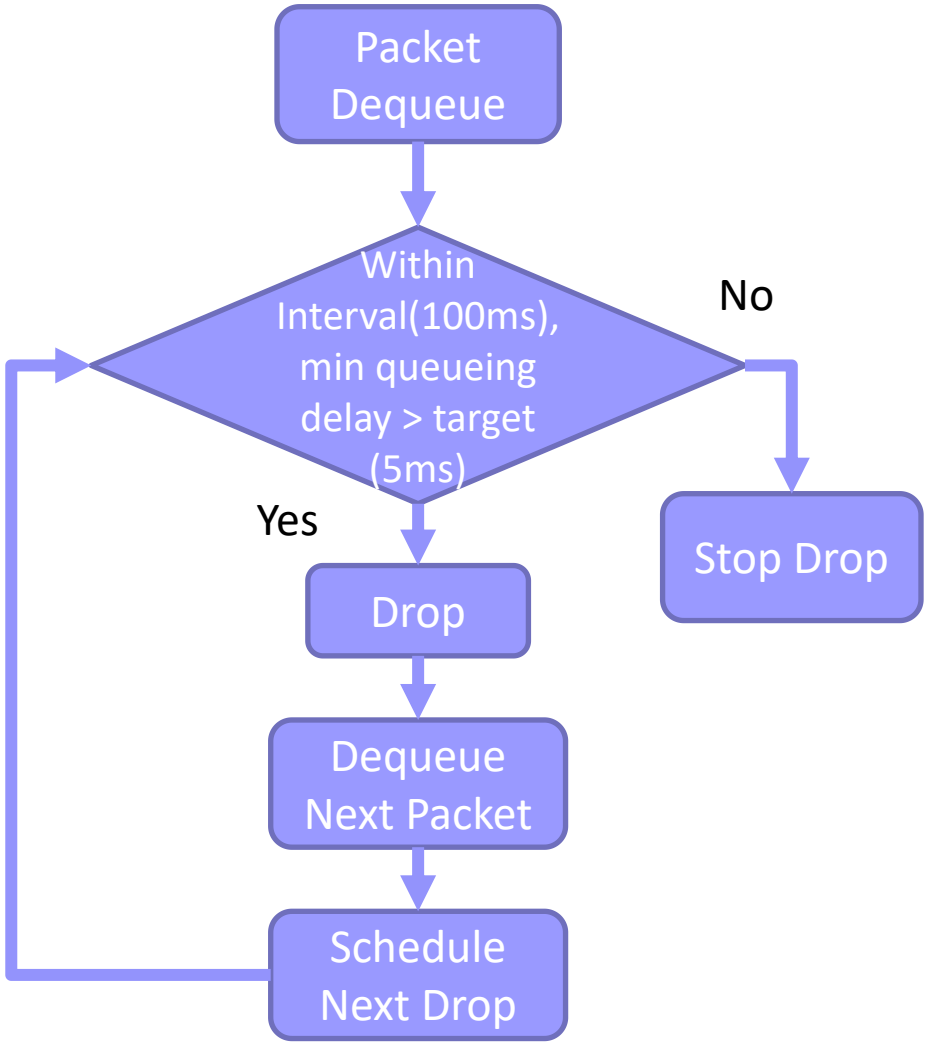
CoDel (Controlled Delay)

- 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

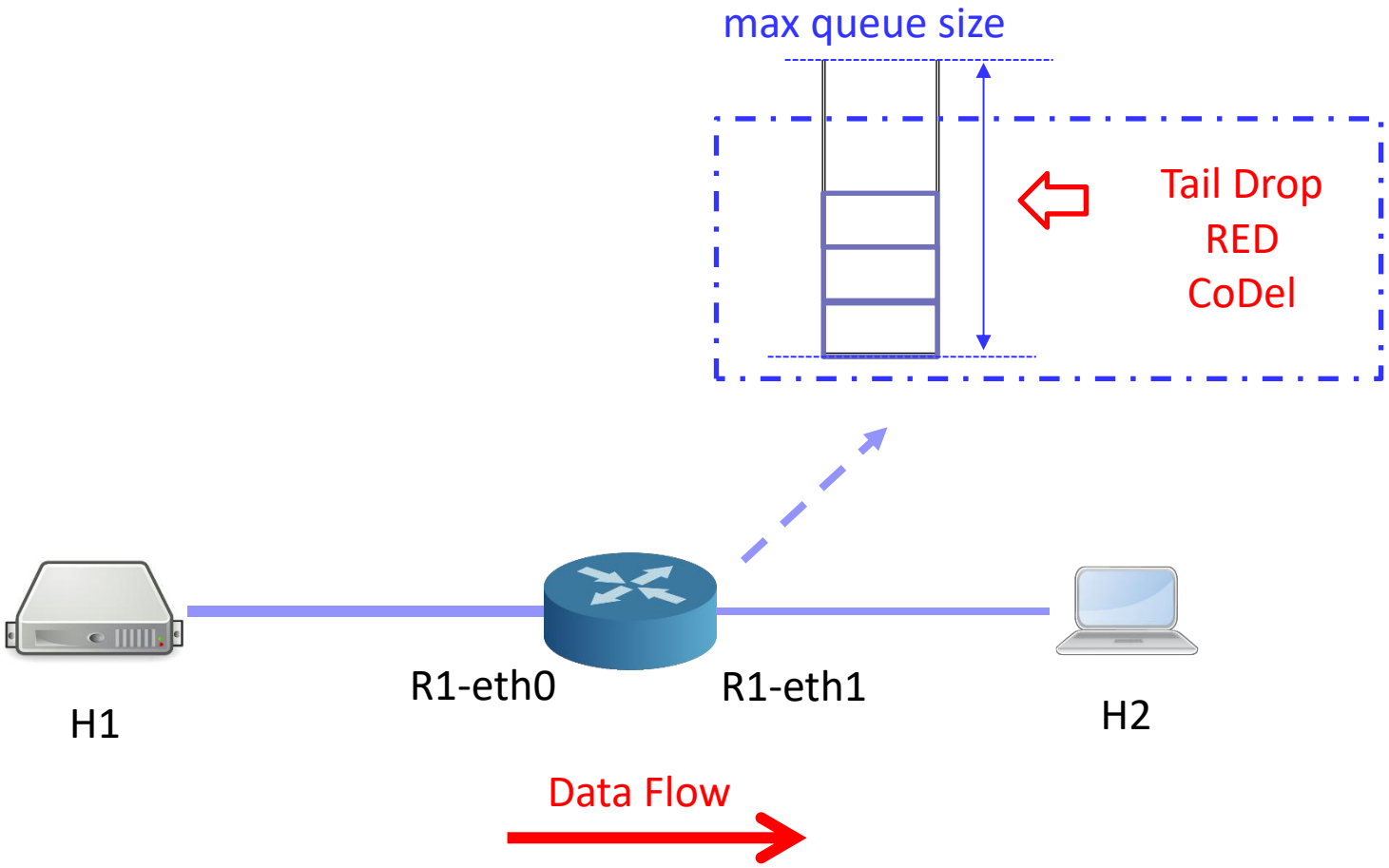
- CoDel设计思路
  - 使用包在队列中的停留时间作为度量指标
    - 而不是队列长度
- CoDel算法
  - 当包停留时间超过target值时
    - 将该数据包丢弃
    - 并根据control law设置下次丢包时间
      - interval / sqrt(count)
  - 直到所有包的停留时间都小于target值

CoDel流程示意



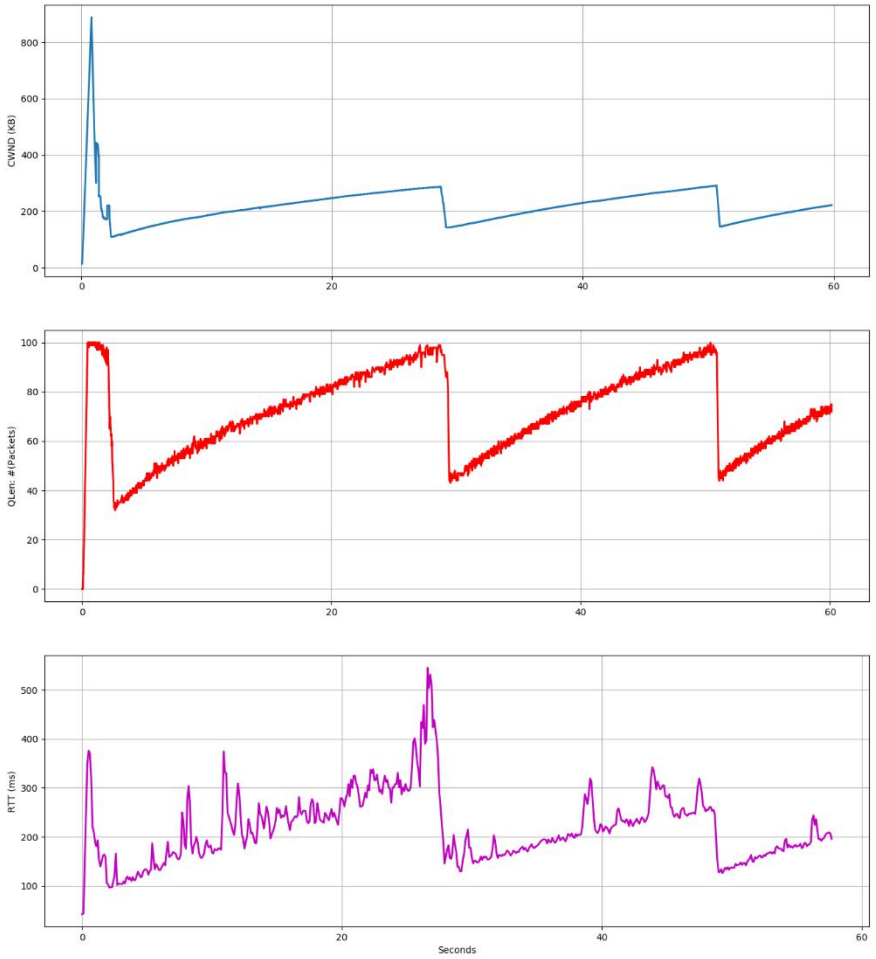
- CoDel具有Tail Drop类似的优点：不需要配置参数
  - 参数硬编码到CoDel机制中，但不一定是最优的
- CoDel可以减少延迟，但一般不会提升吞吐率

实验环境


max queue size
Tail Drop
RED
CoDel
H1
R1-eth0
R1-eth1
H2
Data Flow

重现BufferBloat问题

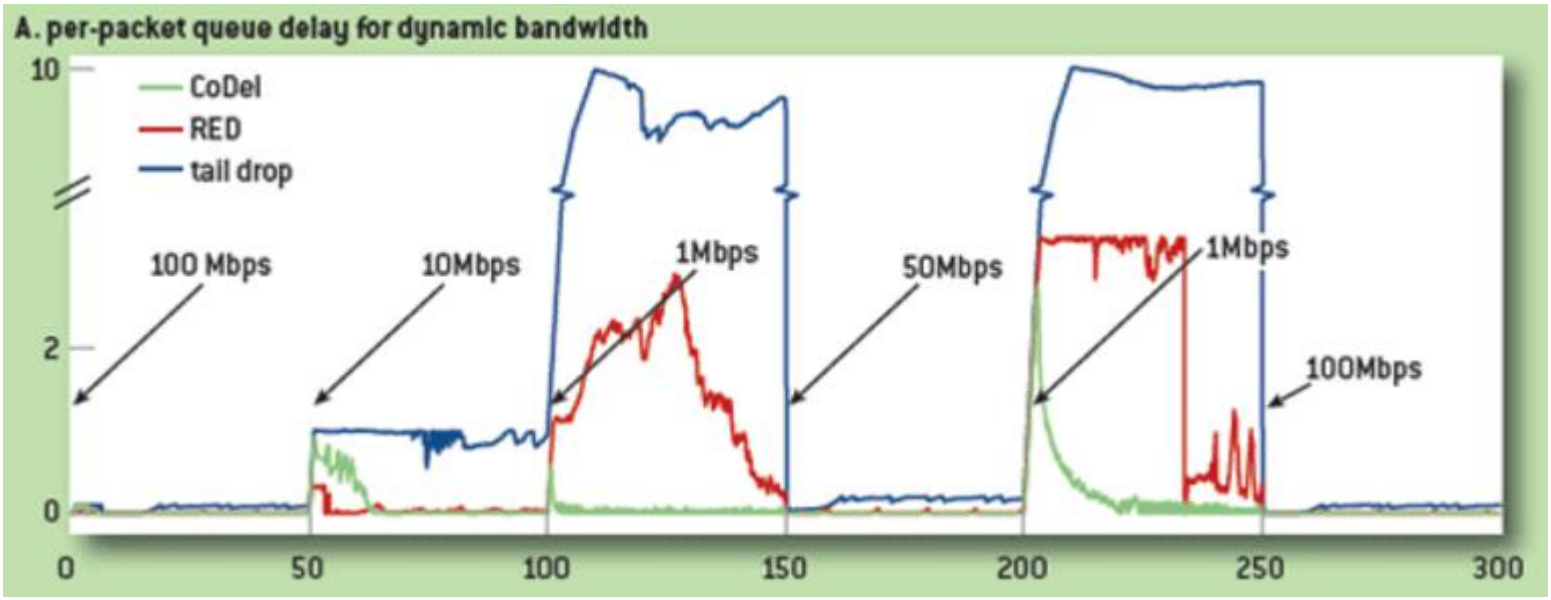
- 根据附件材料中提供的脚本，重现如下实验结果：



- 该图为h1(发送方)在对h2进行iperf的同时，测量h1的拥塞窗口值(cwnd)、r1-eth1的队列长度(qlen)、h1与h2间的往返延迟(rtt)
- 变化r1-eth1的队列大小，考察其对iperf吞吐率和上述三个指标的影响

解决BufferBloat问题

- 根据附件材料中提供的脚本，重现如下实验结果：



Controlling Queue Delay 图7.A (https://queue.acm.org/detail.cfm?id=2209336)

Linux TC (Traffic Control)

- Linux TC
  - 实验依赖于tc命令对链路进行带宽、延迟设置，对端口队列的长度和管理机制进行设置
  - Mininet中已经将tc的绝大部分功能进行封装，但没有封装red, codel管理机制，也没有封装修改带宽值的功能
- 命令格式

```
r1# tc qdisc add dev r1-eth1 parent 5:1 handle 6: codel limit 1000

r1# tc class change dev r1-eth1 parent 5:0 classid 5:1 \
htb rate 10Mbit burst 15k
```

Linux SS (Socket Statistics)

- Linux SS
  - 输出连接套接字的相应统计信息
- 数据格式
  - 时间，拥塞控制算法，指标：值，指标：值， …
  - 这里我们只使用时间和cwnd指标
    - Cwnd：单位时间（ RTT ）内允许发送的数据包数目

实验注意事项

- RTT输出结果不稳定
  - 由于ping程序实现没有很好的并行机制，相邻ping之间的间隔是变动的，低至百毫秒，高至数秒
  - 偶然情况下，Ping程序结果输出的持续时间可能比设定时间短很多，建议多尝试几次
- 解决Bufferbloat问题实验画图
  - 如果直接使用线性坐标画图，则Codel和RED的值几乎显示不出来
  - 可以用复现图中的坐标截断方法，也可以将纵坐标设置为log坐标
- 本次实验不需要在OJ平台上提交代码，只需要在SEP平台上提交报告

附件文件列表

- utils.py
- reproduce_bufferbloat.py　　# 重现Bufferbloat问题
- mitigate_bufferbloat.py　　# 解决Bufferbloat问题

调研思考题

- 拥塞控制机制对Bufferbloat的影响
  - 前文中提到，导致Bufferbloat问题的三个关键因素：队列长度，队列管理机制，和拥塞控制机制。同时，分别从上述三个角度都可以解决Bufferbloat问题。调研分析两种新型拥塞控制机制（BBR [Cardwell2016], HPCC [Li2019]），阐述其是如何解决Bufferbloat问题的。

参考文献

- [Allman2013] M. Allman, Comments on bufferbloat, ACM SIGCOMM CCR, 2013
- [Appenzeller2004] G. Appenzeller et al., Sizing Router Buffers, ACM SIGCOMM, 2004
- [Cardwell2016] N. Cardwell et al., BBR: Congestion-Based Congestion Control Measuring bottleneck bandwidth and round-trip propagation time, ACM Queue, 2016
- [Floyd1993] S. Floyd et al., Random Early Detection (RED) gateways for Congestion Avoidance, ACM ToN, 1993
- [Gettys2011] J. Gettys et al., Bufferbloat: dark buffers in the Internet. Communications of the ACM, 2011
- [Jacobson1999] v. Jacobson et al., RED in a Different Light, Tech Report, 1999
- [Jiang2012] H. Jiang et al., Tackling Bufferbloat in 3G/4G Networks, ACM IMC, 2012
- [Li2019] Y. Li et al., HPCC: High precision congestion control. ACM SIGCOMM 2019
- [Nichols2012] K. Nichols et al., Controlling Queue Delay, ACM Queue 2012
- [Phanishayee2008] Amar Phanishayee et al., Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems, Usenix FAST, 2008
- [Sivaraman2013] A Sivaraman et al., No Silver Bullet: Extending SDN to the Data Plane, ACM HotNets, 2013