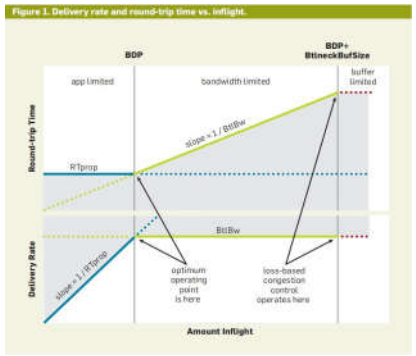
拥塞控制（下）

中国科学技术大学
自动化系　郑烃



BBR:BtlBW and RTprop based Congestion Control

提纲

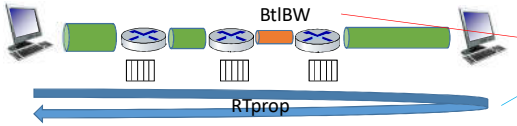
1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

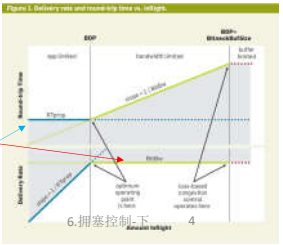
提纲

1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

1.BBR概述

- Reno、Cubic等基于丢失的拥塞控制：吞吐振荡，延迟大，侵略性强
- BBR：基于模型的拥塞控制，不是基于丢失，也不是基于延迟
- 主要思想：
  - 模型：将通信分成应用受限、带宽受限等阶段
  - 经常测量BtlBW和RTprop，计算BDP，反映网络通信量和路由的变化
  - 按照BtlBW控制主机注入速率，按照BDP控制inflight的数量
- 目标：吞吐量大、延迟低、公平性好
- 在谷歌B4等网络中使用，实际效果好





提纲

1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

2.基于丢失的拥塞控制算法的问题

- 原理：丢失=拥塞，拥塞之后（慢启动之后cwnd减半，或减半）
  - Reno，New Reno，SACK，Cubic等
- 效果还不错，支撑了互联网40多年的发展（1980年以后）
  - 前提：有线链路多，带宽不大，网络交换节点buffer不大情况下
  - 丢失=拥塞：99%以上
- 情况发生了变化
- 链路带宽达数十Gbps，无线链路的大量采用：丢失≠拥塞，结果：带宽振荡
  - 高带宽链路：出错造成丢失当成拥塞，在拥塞控制方法下的高吞吐所要求的丢失率与链路天然出错率数量级相当
  - 无线链路特点：出错率高，丢包率高
- 高速buffer便宜，大容量buffer在网络交换节点采用
  - 基于丢失的拥塞控制倾向于将通路上(先是瓶颈链路交换节点)的buffer充满
  - 先拥塞，之后很久才会丢失，时机迟，延迟大

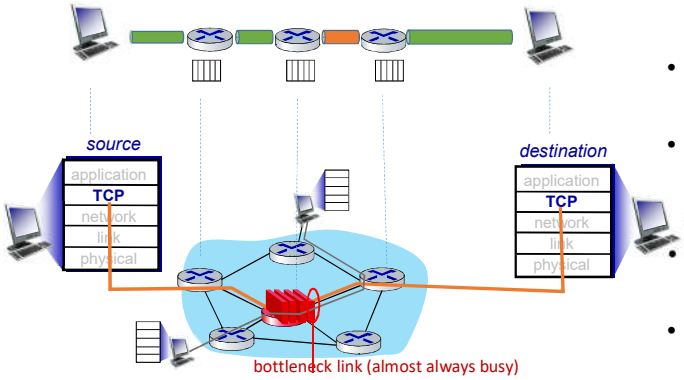
2.基于丢失的拥塞控制算法的问题

- 基于丢失的拥塞控制算法的问题：
  - 反复丢失（就是靠着丢失尝试通路的带宽上限）带来吞吐振荡（链路利用率不高）
  - 端到端延迟大（buffer溢出，排队延迟大）
  - 算法侵略性强，整网效果不好（对于其他的拥塞控制算法不友好，带宽分配不公平）

提纲

1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

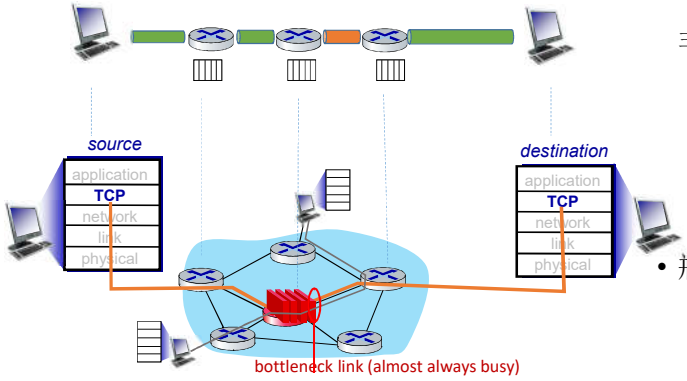
3.主机之间的通信模型



- 源端进程通过若干段链路形成的Path将分组发给目标端
- 每一条路由器按照分组转发策略如：FCFS来转发分组
- 链路的带宽划分实际上相当于各主机对在这条链路上注入的速率
- 这条链路带宽被主机对通信所划分
- 从源到目标的每段链路都是如此划分带宽，与其他主机对通信共享

下图来自Kurose教授的第8版计算机网络-自顶向下方法对应教学资源

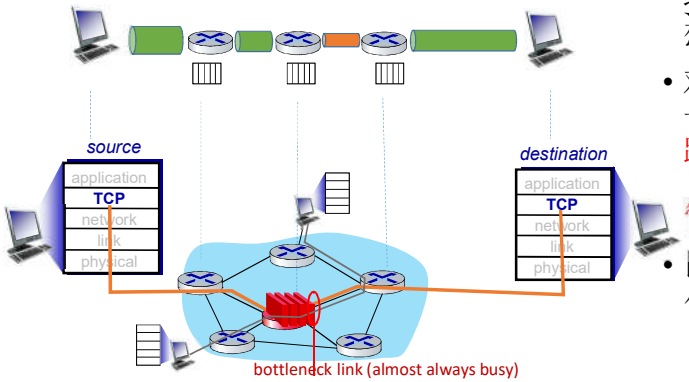
3.主机之间的通信模型



下图来自Kurose教授的第8版计算机网络-自顶向下方法对应教学资源

- 对于一个特定主机对之间的通信而言，瓶颈链路是Path上“分得”带宽最小的那段链路，非常关键
  - 限制了源-目标吞吐量
  - 注入比瓶颈链路分得的带宽快，在相应路由器排队，
    - 慢则队列减少
  - RTprop延迟最有可能变大地方
    - 受限再次拥堵
- 瓶颈链路的变化
  - 路由变化
  - 路由不变，但瓶颈变成其他段
  - 其他主机对加入或者退出竞争，瓶颈链路的带宽本身变化
- 特定时刻，总有瓶颈链路

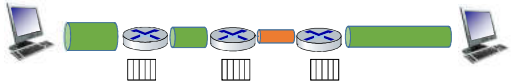
3.主机之间的通信模型



来自Kurose教授的第8版计算机网络-自顶向下方法对应教学资源

- 抽象：其他通信对的存在，抽象成路由和瓶颈的变化，想象成不存在
- 对于拥塞控制而言，就我一个通信对，我看到的是路由形成的**通路**及之上“分得”带宽的每段小水管构成
- 比方：像若干段之前有个小池子的水管
  - 水管刚性
  - 池子柔性，但是有极限
- 瓶颈就是那个最细的管道

3.主机之间的通信模型-控制思路

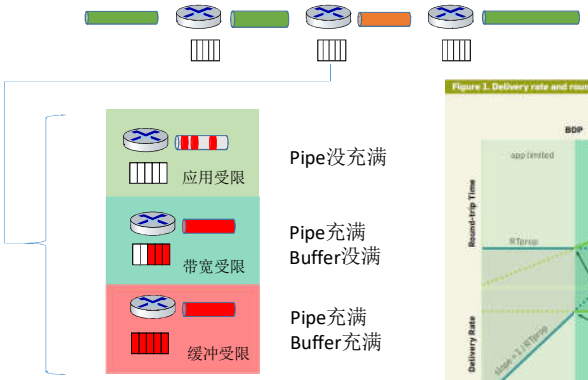


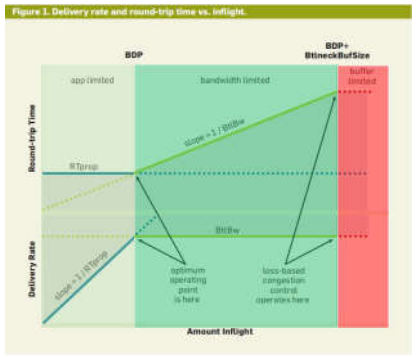
1. 源端注入的速率不超过瓶颈链路带宽BtlBWBW
   - 超过则在瓶颈链路拥堵，瓶颈水管就那么粗
   - 肯定会在对应的队列排队
   - 从而延迟增加，而有效吞吐不可能增加

3.与水管不同，通信是经常变化的，需经常测量2个量，按照以上思路进行控制

2. BDP=RTprop* BtlBW，从源端注入的等待被确认的inflight数据不超过BDP
   - RTprop是指轻载时，各队列都没有排队（瓶颈链路队列没排队，其他肯定不排队）情况下的往返传播传输延迟之和
   - BDP：通路的容量就那么大，水管容量（双向）就那么大，一旦超过会积在瓶颈水管前池子
   - 一旦超过，就首先会在拥塞路由器的队列中排队，增加延迟而不增加

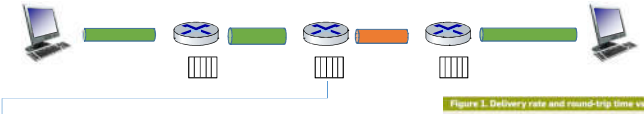
3.主机之间的通信模型

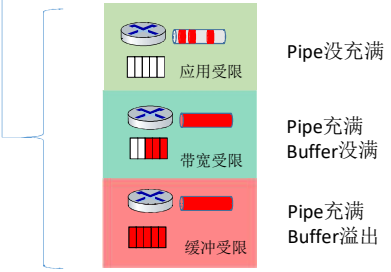


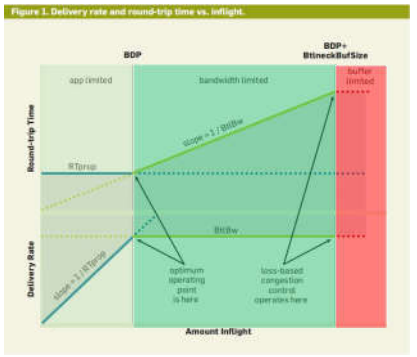


右图来自 NEAL CARDWELL等BBR: Congestion-Based Congestion Control，余下未标记的图都来自该文章

3.主机之间的通信模型

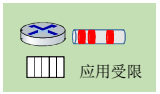


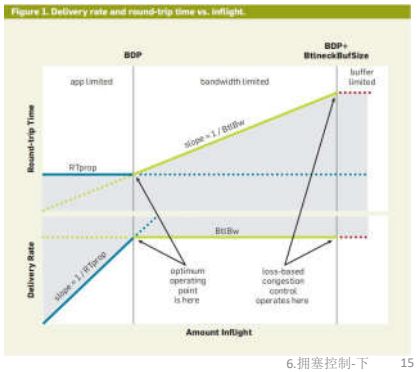




3.主机之间的通信模型

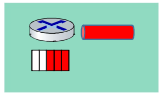
a. Inflight<BDP：应用受限阶段
- 比方：注入水速<瓶颈水管，注入水量小于管道容量（也不可能超，管道排的快）：管道和池子都没满
- 网络：主机注入速率< BtlBW，且 inflight<BDP
- 通路能力没有得到充分发挥，吞吐量受限于应用数据速率比较低
- RTprop不变，吞吐量随着应用速率增加而增加（应用数据受限）

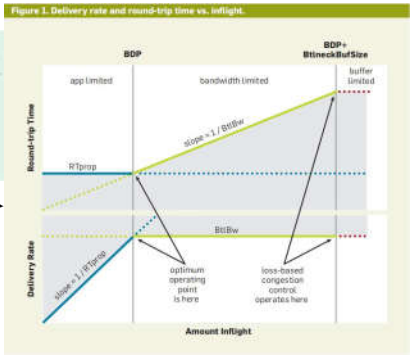




3.主机之间的通信模型

b. (BDP+buffer)>Inflight>BDP：带宽受限阶段
- 1）注入水速=瓶颈水管（才有可能inflight超过BDP，进来得快，出得慢）
- 2）**注入水量** 大于（管道容量） 且 *小于 管道容量+瓶颈池子*）
- 管道满了，但池子没满（池子伸缩作用，注入水速超过瓶颈速度，迟早要满）
- 通信：主机注入速率=BtlBW，且 (BDP+buffer)>Inflight>BDP
- 表现：吞吐量不变，而RTprop随着队列中的分组数量增加而增加

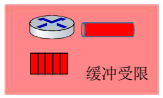


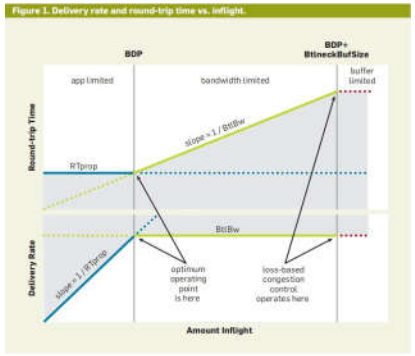


3.主机之间的通信模型

c. Inflight>(BDP+瓶颈链路队列)：buffer受限阶段（注入水速=瓶颈水管，且注入水量大于容量+池子）

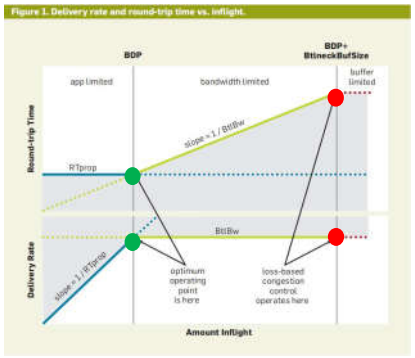
- 1）注入水速=瓶颈水管（或者过去=，现在<）2）**注入水量 大于 *BDP+瓶颈池子***）
- 管道满了，池子满
- 主机注入速率=BtlBW（或者过去大于，持续一段时间），且Inflight>(BDP+buffer)，Buffer溢出
- 结果：吞吐量不可知（拥塞控制要动作，降速），RTprop极大（排队长）





3.主机之间的通信模型

- 结合流量模型看基于丢失的拥塞控制算法的主要问题：
  - 拥塞在**优化运行点绿点**之后已经发生
  - 丢失发生在红点
  - 拥塞比丢失要来的早，早期buffer小，绿点红点间距小，问题不大
  - 基于丢失的拥塞控制的主要问题
    - 时机较迟（特别是现在buffer很大，绿点红点距离大），需雷霆手段矫枉过正，cwnd减半吞吐振荡
    - 基于丢失的拥塞控制倾向于让（瓶颈）buffer溢出，造成丢失（促成丢失，从而探测可用带宽，但丢失本身代价大），延迟大
    - 超过优化运行点，吞吐没增加而延迟增加

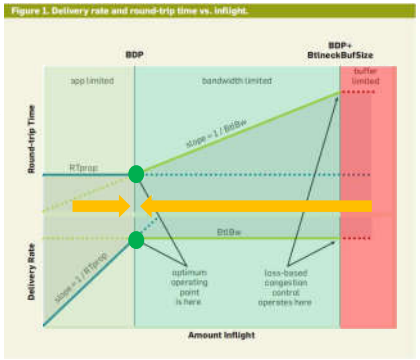


提纲

1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

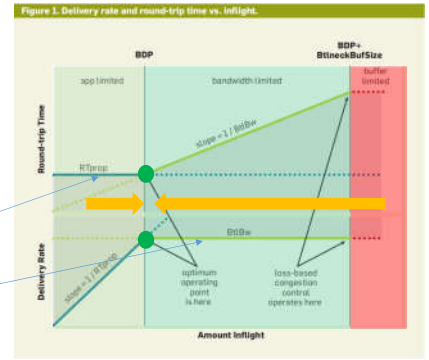
4.BBR拥塞控制思路-历史

- 1979 Lenard Kleinrock建立通信模型：洞察运行规律和揭示优化运行点
- 感知BtlBW和RTprop，将系统运行在优化运行点
  - 水管注入速率和瓶颈水管相当
  - 注入水量=管道的容量（双向，不包括池子）
- Jeffrey M.Jaffe证明分布式算法无法100%收敛到优化运行点
- 拥塞控制算法方向改变，基于丢失进行拥塞控制
- Buffer小，优化点和丢失点差距不大，能够使用



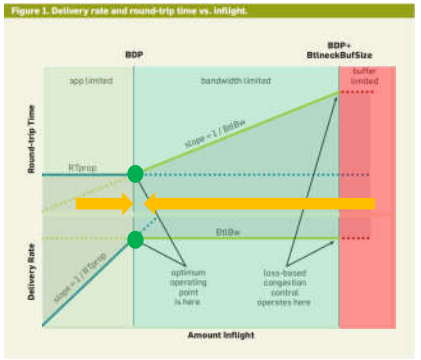
4.BBR拥塞控制思路-历史

- 随着链路带宽增大、无线链路的大量使用，以及高速Buffer容量增大，基于丢失的拥塞控制问题越来越大
- 最核心问题：BtlBW和RTprop不可同时测量
  - 必须使得inflight小于BDP才能够测量RTprop，轻载时的往返延迟
  - 使得inflight大于BDP，才能够测量BtlBW
  - 不能够同时测量两个量



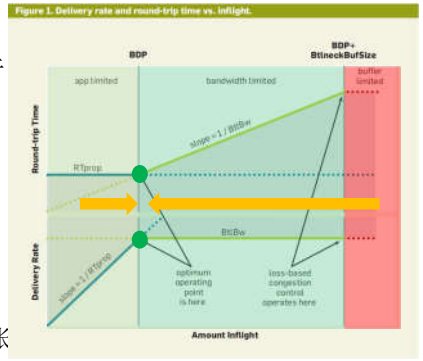
4.BBR拥塞控制思路

- 谷歌团队
  - B4网络Reno类和CUBIC算法效果不好
  - 分析问题，获取大量运行数据
  - 采用现代鲁棒伺服控制系统最新成果
  - 思路：可以创造机会激励系统（控制速度和注入量），让它运行在不同状态，分别测量BtlBW和RTprop，大概率测准
  - 从一小段时间来看，系统参数基本稳定，大概率能测准和控好
  - 2015年应用于B4网络，并且发布



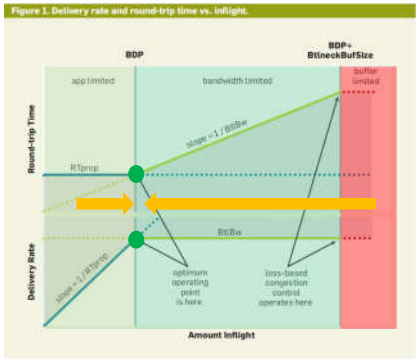
4.BBR拥塞控制思路-总体

- 问题：模型已知，参数(BtlBW和RTprop)未得
- 手段：测量两个值，基于BtlBW和RTprop进行拥塞控制，BBR
- 目标：吞吐大，延迟少，较公平
- 源端注入的速率等于瓶颈管道，注入的量不超过若干管道形成的容量
  - 注入的速度快，瓶颈管道前面的池子要膨胀（延迟大，吞吐却不大）
  - 一个RTT时间内注入多，管道容纳不了，池子膨胀
- 关键：如何测量这两个值
  - BtlBW
  - Rtprop
  - 适应它们的变化



4.BBR拥塞控制思路-a.测量RTprop

- 应用受限阶段，测量RTprop，
  - 交互式应用应用数据不多，本身就在该阶段
    - RTprop WR＝10s有更新，不用单独测量RTprop
  - 高突发情况：
    - RTprop近10s没有更新
    - 2%间或降低速率，形成条件测量RTprop，适应路由变化



4.BBR拥塞控制思路-a.测量RTprop

- 确保在应用受限阶段，测量RTprop
  - TCP连接建立时或者应用握手时，交互式应用数据不多时
  - 控制条件：高带宽通信时10s抽出200ms，条件是一个RTprop发送4MSS情况下

$$\widehat{RTprop} = RTprop + \min(\eta_t) = \min(RTT_t) \quad \forall t \in [T - W_R, T]$$

  - 只有路由变化了，RTprop才变化
  - W_R时间窗口：10s中或者若干分钟（不会经常变化）
    - 时间过滤器，防止老的测量值对现在测量的影响，适应路由的变化
  - 是真实RTprop的无偏估计子
  - 实现：TCP根据放出去的时间和ack的时间差计算（TCP选项）

4.BBR拥塞控制思路-b.测量BtlBW

- 在带宽受限阶段，测量交付速率，将近期最大的交付速率当做BtlBW
  - 连接建立后不断增加inflight量，连续三个RTprop交付速率不增加25%进入BL状态
  - 在高带宽通信时，适应瓶颈链路带宽的变化（续）

$$deliveryRate = \Delta delivered / \Delta t$$

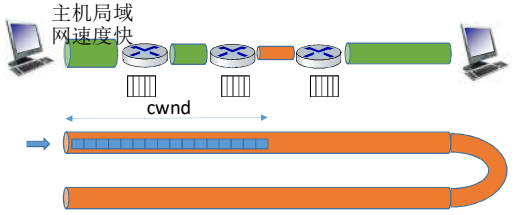
$$\widehat{BtlBw} = \max(deliveryRate_t) \quad \forall t \in [T - W_B, T]$$

  - W_B是过滤器窗口 6~10RTprop，防止过老的测量对新的测量的影响

4.BBR拥塞控制思路-c.inflight<BDP

- BDP=BtlBW*RTprop
- cwnd=cwnd_gain*BDP（在一个RTT内，发送出去待确认的上限）
- inflight<=cwnd （inflight实际待确认的数据）
  - 在一个RTT内，待确认的数据inflight 不超过 计算出来的BDP
    - 确认必须要一个RTT内才能够物理返回，必须要花的时间
    - 控制inflight<=cwnd，实际上就是控制粗颗粒度的速率
  - inflight<=cwnd情况下，发新数据，待确认的数据inflight增加
  - 来的有效确认，inflight减少
- cwnd_gain是个重要参数，1以上适应网络的变化，动作不要太唐突

4.BBR拥塞控制思路-d.按照BtlBW控速



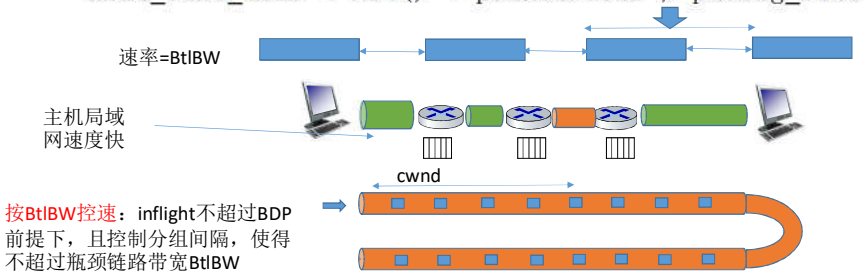
突发方式不控速：
- inflight不超过BDP，一个RTT发BDP
- 但会超过瓶颈链路带宽
- 队列堆积延迟大，丢包

- cwnd是在对方未确认情况下，在一个RTprop内发送出去的量
  - Reno和Cubic的是突发的，分组连续发送，对网络造成的冲击较大
  - 形成的burst会造成部分路由器队列溢出
- 控速方法
  - cwnd对应的n个分组，需要控制它们之间的间隔使得速率不超过BtlBW

4.BBR拥塞控制思路-d.按照BtlBW控速

- cwnd对应的n个分组，控制它们之间的间隔使得速率不超过BtlBW
- 输入：BtlBW和分组大小size
- 计算分组之间的发送间隔:pacing_rate=pacing_gain*BtlBW

```
next_send_time = Now() + packet.size / pacing_rate
```



按BtlBW控速：inflight不超过BDP前提下，且控制分组间隔，使得不超过瓶颈链路带宽BtlBW

4.BBR拥塞控制思路-e.适应BtlBW变化

- 在带宽受限阶段

周期性地增速适应BtlBW的增加

- 一个Cycle 8个节拍（RTprop），一个节拍pacing_gain=1.25，增速（1RTprop发的多了，1.25*BDP）

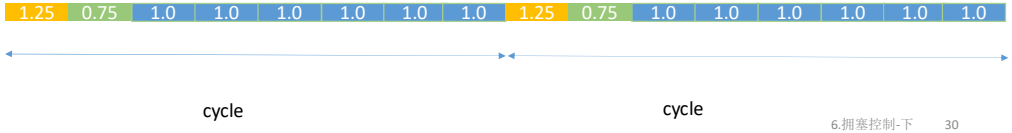
1：如RTprop没变大=>交付速率增加， BtlBW按公式更换成更大的交付速率

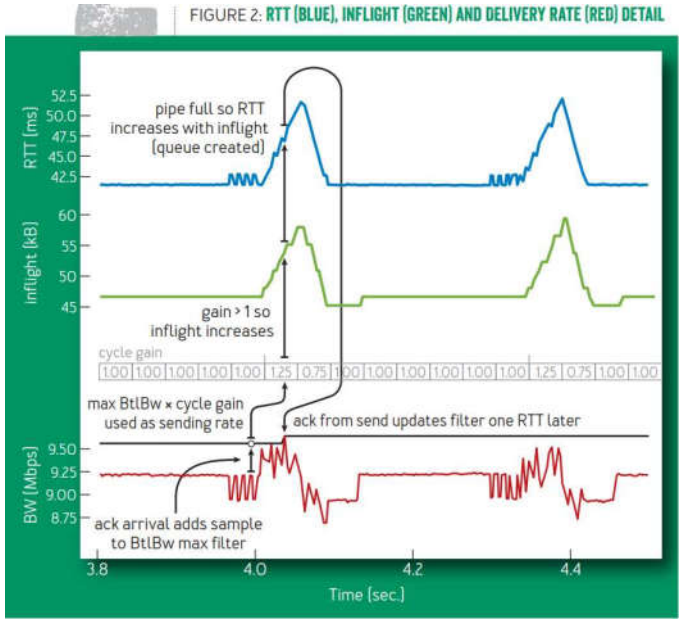
2：如RTprop变大=>交付速率没变大，BtlBW不更新

- 一个节拍RTprop，0.75倍BDP，放空瓶颈buffer
- 余下6个节拍原有速度，稳定

$$deliveryRate = \Delta delivered / \Delta t$$

$$\widehat{BtlBw} = \max(deliveryRate_t) \quad \forall t \in [T - W_B, T]$$



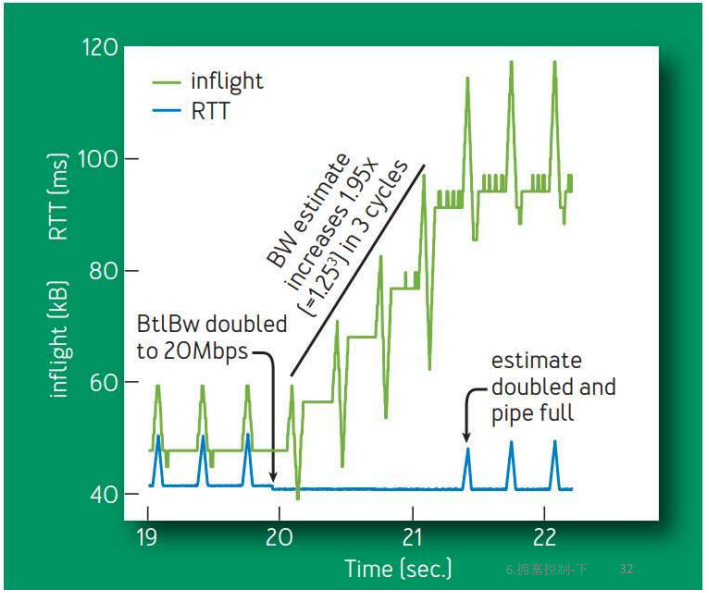


1. 延迟（蓝）增加
2. 速率（绿）增加，inflight数量
3. BtlBW按照公式更新（黑）
4. 交付速率（红）增一点

右在增速，延迟增加，BtlBw没增加

适应BtlBW增速

- 1.25倍增加速率
- 如延迟不增
- 交付速率增加，更新BtlBW
- 下个节拍按照增加后的BtlBW工作
  - 速率变成：1.25^n
  - 1.25^3=1.95
  - =10Mbps变成了20Mbps
- 右边延迟增加了，BtlBW不增



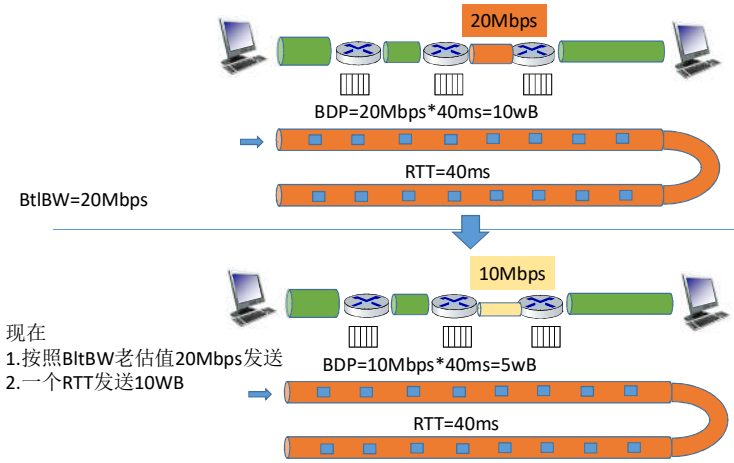
4.BBR拥塞控制思路-e.适应BtlBW的变化

- 过滤器窗口让老的过高BtlBw过期适应BtlBW降低
  - 老的BtlBW估值较高20Mbps，BDP估值较大10WB
    - 发送过快20Mbps->瓶颈队列瞬间堆积
    - 一个RTT 40ms，发出 较多10WB->通路总体堆积 5WB
  - 而交付速率就是10Mbps，有排队延迟大，计算出交付速率低，10RTT内不更新
  - 10RTT之后，老的高的估值BtlBW过期，新的估值低于10Mbps
  - 按照新BtlBW估值<10Mbps速度，一个RTT 注入<5WB，排的快
  - 堆积队列分组排空，之后估值速度上升，达到10Mbps

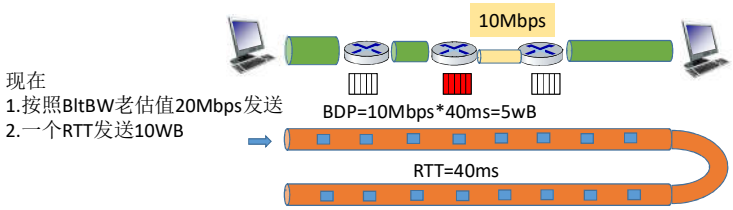
$$deliveryRate = \Delta delivered / \Delta t$$

$$\widehat{BtlBw} = \max(deliveryRate_t) \quad \forall t \in [T - W_B, T]$$

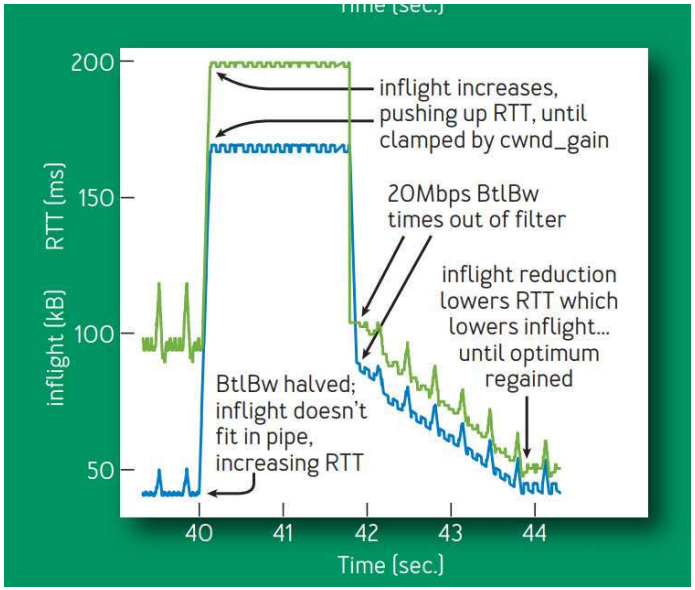
4.BBR拥塞控制思路-e.适应BtlBW的变化



4.BBR拥塞控制思路-e.适应BtlBW的变化



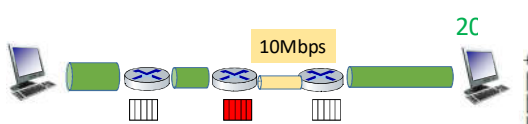
1. 瓶颈链路前面buffer，其次其他路由器Buffer，缓存10WB-5WB数据
2. 延迟增大，交付速率 降低成10Mbps
3. 老的BtlBW估值20Mbps会失效，新的估值10Mbps替代

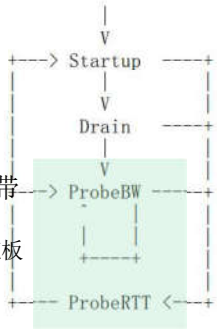


提纲

1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

5.具体算法

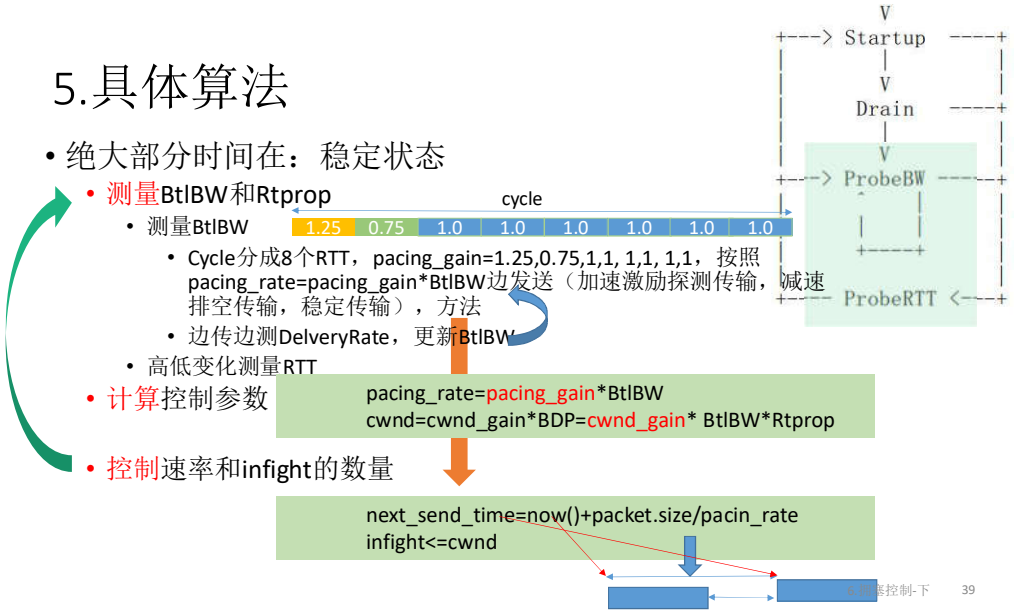




- BBR状态机
  - Startup：连接建立时，指数型增加cwnd，从应用受限进入到带宽受限阶段
    - 把水管注满，让瓶颈水管前面的水池有水，达到瓶颈水管水速的天花板
    - 测量RTprop，同时从上升期进入到平台期高速传输，记下BtlBW
  - Drain：
    - 将Startup阶段注入多的分组排空，使得inflight=BDP
    - 队列没有分组，延迟才会小；水管满了利用率才能够大，吞吐量大
  - ProbeBW：高速发送，周期性探测新的天花板或者退缩到新的低速平衡，适应瓶颈带宽的变化
  - ProbeRTprop：在高速发送期间，2%时间创造条件测量RTprop
    - 如应用本身受限，发送的量比较小，近M_R有RTT更新，无需专门测量
    - 如应用一直有数据发送，高带宽运行，创造条件测量RTprop
    - 适应路由的变化

5.具体算法

- 绝大部分时间在：稳定状态
  - 测量BtlBW和Rtprop
    - 测量BtlBW
      - Cycle分成8个RTT，pacing_gain=1.25,0.75,1,1, 1,1, 1,1，按照pacing_rate=pacing_gain*BtlBW边发送（加速激励探测传输，减速排空传输，稳定传输），方法
      - 边传边测DelveryRate，更新BtlBW
    - 高低变化测量RTT
  - 计算控制参数
  - 控制速率和infight的数量

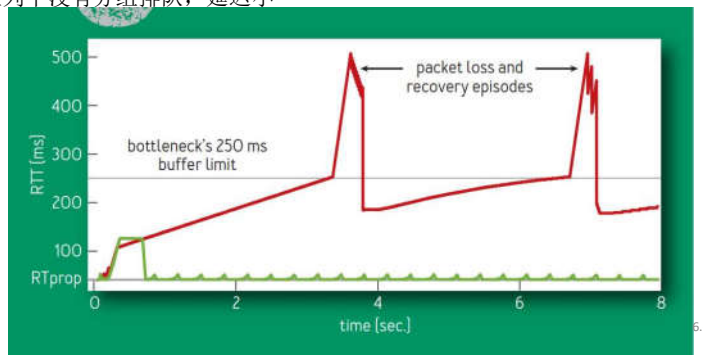


5.具体算法-a.Startup

- 链路带宽范围非常大(10^{12}bps)，需快速达到链路的能力
  - 二分搜索，指数增加
- cwnd=cwnd_gain*1MSS=2/ln2=2.885，起点2.885
- 一个RTprop倍增cwnd=(2/ln2)*2^n，直到达到infight>BDP
  - 如：连续三个RTprop，交付速率增加不超过25%，达到平台期
    - 三个RTprop，防止接收方窗口受限，给一个接收方增大接收窗口的机会
  - 倍增cwnd，而且要超过BDP，一定是2*BDP及以上
  - 结果：
    - 到达了带宽受限阶段
    - 在通路上的分组滞留了BDP，需要排空瓶颈队列，增大RTprop

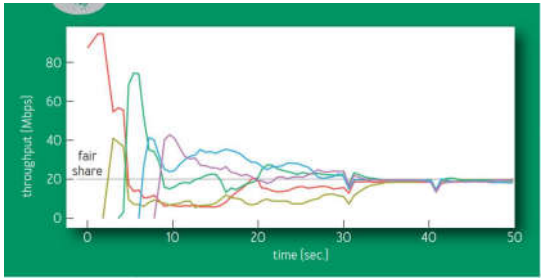
5.具体算法-b.Drain

- 目的：排空Startup后通路上的多余1个BDP分组，减少延迟
- 手段：ln2/2为系数，cwnd=cwnd_gain*BDP，减少拥塞窗口直到<=BDP
  - 一个RTT注入 0.347 BDP，很快排空
- 对比：cubic没有排空，60-100%时间处在队列满状态，延迟大
  - BBR队列中没有分组排队，延迟小



5.具体算法-c.ProbeRTprop

- 目标：减少注入的量，测量RTprop；
  - 同时大流减少发送的量，让其他流看到小的RTprop，减少BDP，各流同步
  - 后进来的流有机会抢更大的带宽，各流通过这个事件同步，公平划分流
- 测量方法
  - 如应用数据受限，近w_R时间内有效更新，无需单独测测量
  - 如应用数据不受限，有很多数据要发送，近期WR没有更新Rtprop了
    - 10s中200ms
    - 一个RTprop 发送4MSS，RTprop更新为最小的RTT




```
function onAck(packet)
  rtt = now - packet.sendtime
  update_min_filter(RTpropFilter, rtt)
  delivered += packet.size
  delivered_time = now
  deliveryRate = (delivered - packet.delivered)
                 /(now - packet.delivered_time)
  if (deliveryRate > BtlBwFilter.currentMax
      || ! packet.app_limited)
      update_max_filter(BtlBwFilter,
                        deliveryRate)
  if (app_limited_until > 0)
      app_limited_until - = packet.size
```

过滤器试图更新RTprop
在应用受限阶段+RTT比
RTprop小

该分组发送和接收时间差

已经确认的字节数+本分组的size

分组发送时间=当前时间

总的确认字节-该分组发送之前已经交付过的

交付速率计算=Delta Data/DletaT

当计算得到的传输速率大于以前最大速率时且 不在最左边的app受限阶段，过滤器更新当前交付速率为最大带宽BltBW

app..应用层尚未发送出去的数据
如果app..>0则不属于应用受限
app=0且 cwnd还可以发，应用受限
减去当前数据报字节


```
function send(packet)
  bdp = BtlBwFilter.currentMax
        * RTpropFilter.currentMin
  if (inflight >= cwnd_gain * bdp)
      // wait for ack or timeout
      return
  if (now >= nextSendTime)
      packet = nextPacketToSend()
  if (! packet)
      app_limited_until = inflight
      return
  packet.app_limited =
          (app_limited_until > 0)
  packet.sendtime = now
  packet.delivered = delivered
```

参数是要发送的数据报 对象
计算BDP =RTT*BTLBW
近期最小延迟和最大带宽的乘积

如果inflight字节大于 cwndgain*bpd不能发，发的已超BDP了>需要等待确认来了或者超时；ack会将inflight的分组数减少，使得具备发送的条件

当前时间大于等于（以前计算的）下一个要发送的时间（发送时间是计算出来的pacing算出来的时间）具备pacing节拍条件
不是可以发就直接发，而是等到下个节拍发

packet是下一个要发送的分组，取下一个分组

没有下一个要发送的分组
还未确认的字节=inflight，返回
可以发送，但当前没有要发送的

应用受限状态赋值
app..=0
且inflight<cwnd
则：应用受限

该分组被放出去的时间是当前
分组被放出去了状态

4.BBR具体算法

确认的时间

```
packet.delivered_time = delivered_time
ship(packet) 真正发走 送上船
nextSendTime = now + packet.size /
       (pacing_gain *
        BtlBwFilter.currentMax)
timerCallbackAt(send, nextSendTime)
```

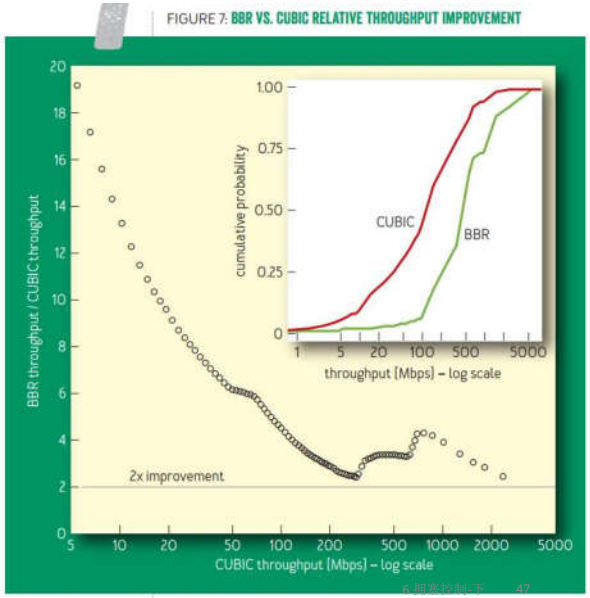
计算分组之间的发送间隔
下一个分组发送的时间
在此之前不允许发送
回调函数设置

提纲

1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

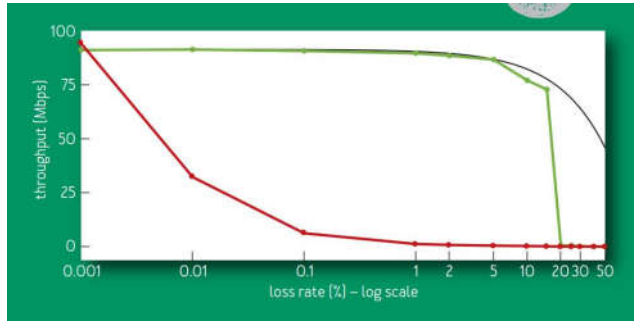
6.BBR应用及效果

- 2015年谷歌B4网络，从CUBIC迁移到BBR
- BBR吞吐量是CUBIC的2-25倍
- 如果将接收端的缓冲区大小加大，BBR是CUBIC吞吐的133倍



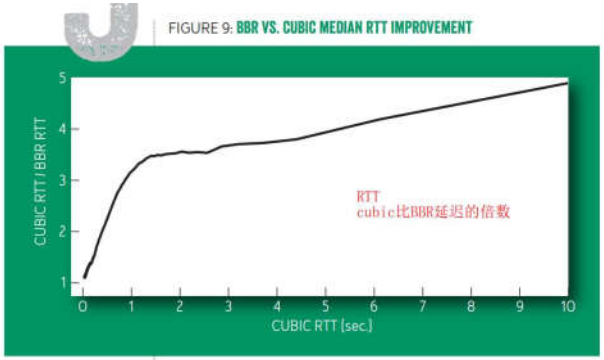
6.BBR应用及效果

- 丢包率在0.001%到50%情况下
- CUBIC吞吐量BBR的10倍以上
  - 随机丢包率越高，BBR吞吐优势越大
  - 0.1%丢包率，BBR吞吐量是CUBIC的100倍



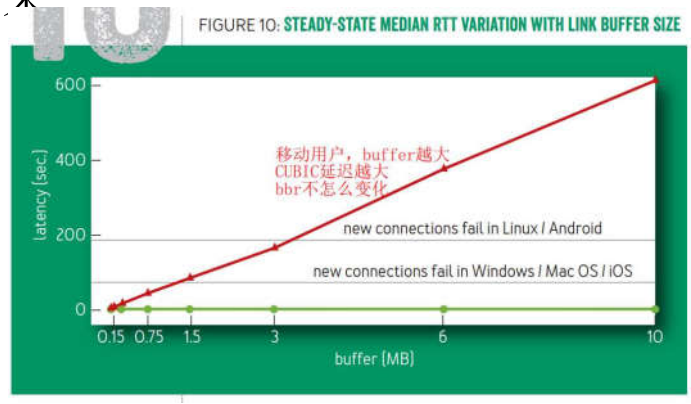
6.BBR应用及效果

- 延迟
  - CUBIC的延迟是BBR的延迟的数倍



6.BBR应用及效果

- 移动通信场景
  - 在缓存加大情况下
  - CUBIC随着Buffer增大，延迟增大
  - 而BBR随着buffer增大几乎不增加
  - 200ms，70ms是不同OS连接超时的时间
  - 采用CUBIC连接超时概率大



提纲

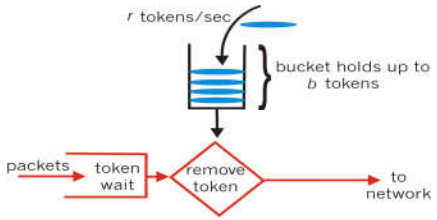
1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

7.一些实际问题

- Ack延迟或者聚集
  - 接收方并不是收到一个包，就给一个确认
  - 改进：延迟发确认或者几个该发的确认形成一个累计确认发给发送方
    - 提升效率
  - 带来的问题：交付速率计算偏小，按此计算BtlBW小，吞吐不高
  - 解决办法：采用cwnd_gain 一个小的倍数*BDP(开始2.885，2)，使传送速率更加平滑
    - 即使ack延迟1个RTT，也不会使得网路熄火
    - cwnd_gain也可以对抗其他的网络变化，cwnd_gain从小到大变化，小心动作不要剧烈

7.一些实际问题

- 网络监管policer
  - 令牌桶监管发送的速率，超过就抛弃
  - 带来问题：BBR学到的BtlBW比真正的瓶颈链路带宽来的小，效率低
  - 解决方案：监测有无令牌桶监管，建立有令牌桶监管的通信模型



来自Kurose教授的第8版计算机网络-自顶向下方法对应教学资源

7.一些实际问题

- 公平性：与基于丢失的拥塞控制算法的竞争
  - 基于丢失的拥塞控制倾向于占满队列，让分组丢失
  - 影响BBR的运行，尽管BBR与CUBIC共同运行不吃亏
    - 本质上BtlBW降低，本质上靠着10RTT超时之前的持续注入，让队列丢失
    - 让CUBIC超时，退缩，但整网效果受限
  - 协作运行是另外一个问题，需要进一步研究

提纲

1. BBR概述
2. 基于丢失的拥塞控制算法的问题
3. 主机之间的通信模型
4. BBR拥塞控制思路
5. 具体算法
6. 应用及效果
7. 一些实际问题
8. 总结

8.BBR总结

- BBR：基于模型的拥塞控制
- 控制思路：
  - 模型：将通信分成应用受限、带宽受限等阶段
  - 经常测量BtlBW和RTprop，计算BDP，反映网络通信量和路由的变化
  - 按照BtlBW控制主机注入速率，按照BDP控制inflight的数量
- 效果
  - 吞吐量大、延迟低、公平性好
- 在谷歌B4等网络中实际使用，和QUIC协作
- 应用前景好，还有很对具体问题需要进一步研究

部分示意图来自：

1）Jim kurose, Keith Ross教材和ppt

2）NEAL CARDWELL等BBR: Congestion-Based Congestion Control