网络拥塞控制机制

武庆华

wuqinghua@ict.ac.cn

什么是网络拥塞

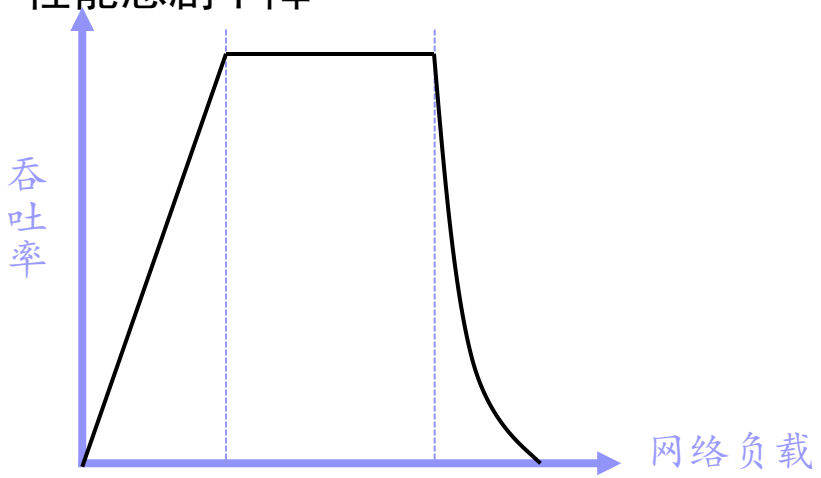
- 拥塞：发送方向网络中发送了过多的数据，超过了网络的处理能力
- 现象：
  - 延迟变高
  - 丢包
  - 传输速率下降

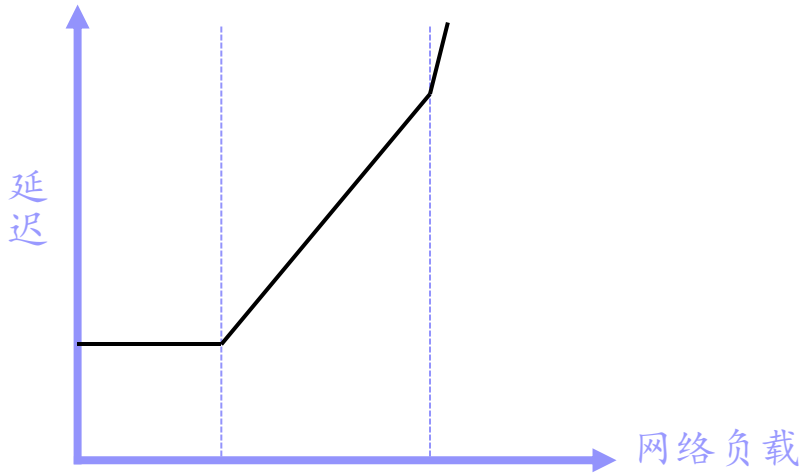
congestion: too many senders, sending too fast

- 不同于流量控制：发送方的发送速度不能超过接收方的处理速度

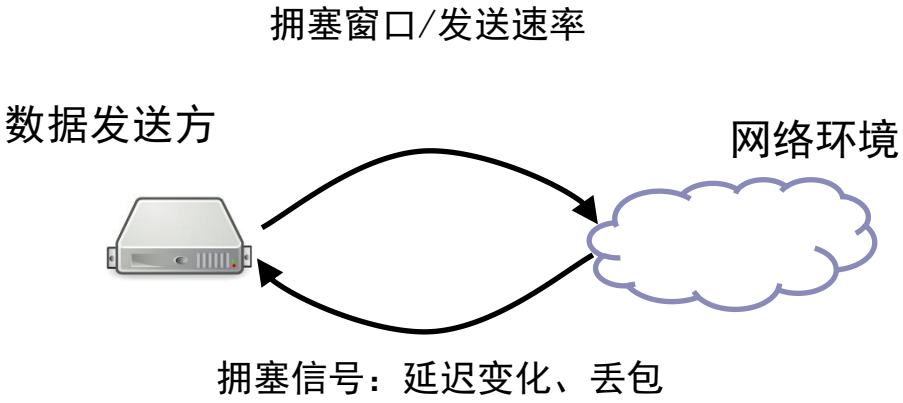
网络拥塞的起源

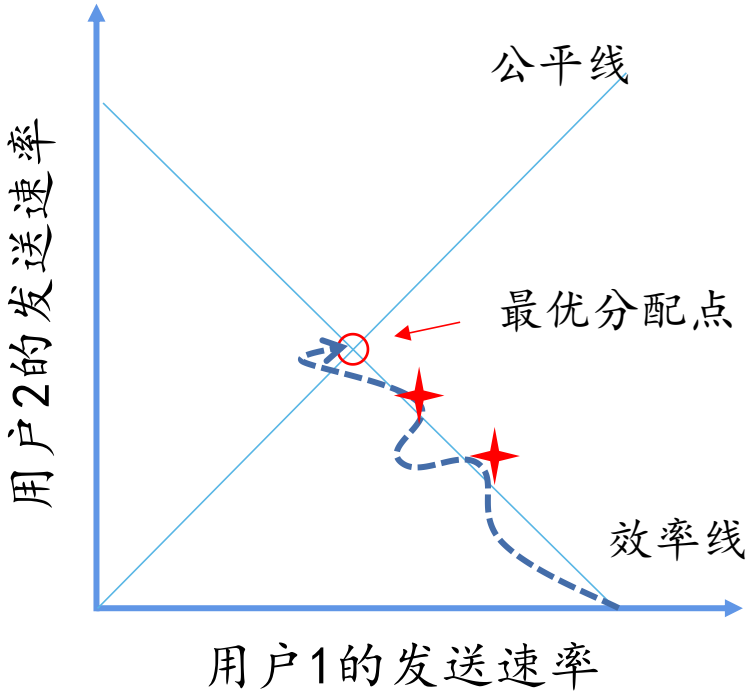
- 拥塞崩溃（Congestion Collapse）：1986年10月，从LBL到UC Berkeley之间的数据吞吐率从32kbps降到了40bps.
  - 成因：对大量的重传没有进行控制，持续拥塞导致更严重的丢包
- 网络负载与性能
  - 当网络中存在过多的数据包时，网络的性能会下降；当网络负载超过某阈值，性能急剧下降





网络拥塞控制机制



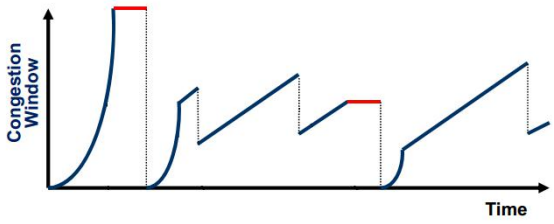


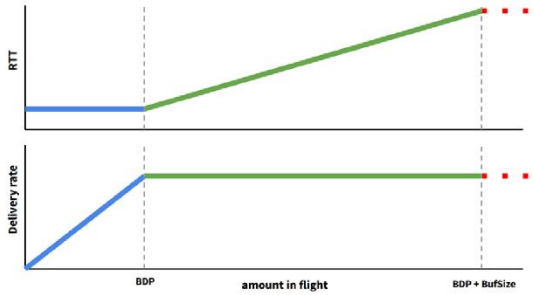
设计目标：高效、公平的利用网络可用带宽

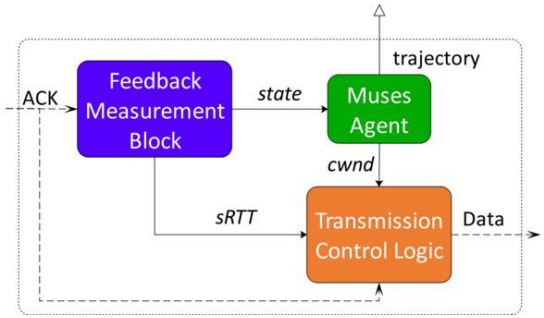
拥塞控制机制设计难点

- 高适应性
  - 网络可用带宽从Kbps到Gbps，如何设计传输控制机制，使得数据发送方可以在高差异化网络中高效传输？
- 高探测效率
  - 移动无线网络的带宽高动态特性，如何设计带宽探测机制，充分利用动态的可用带宽？
- 高收敛速率
  - 现有Web应用大都是短流（<1MB)，如何设计窗口管理机制，保证短流传输充分利用可用带宽？

三种拥塞控制机制设计思路







AIMD机制

- 无拥塞：窗口线性增加(AI)
- 遇到拥塞：窗口减半(MD)

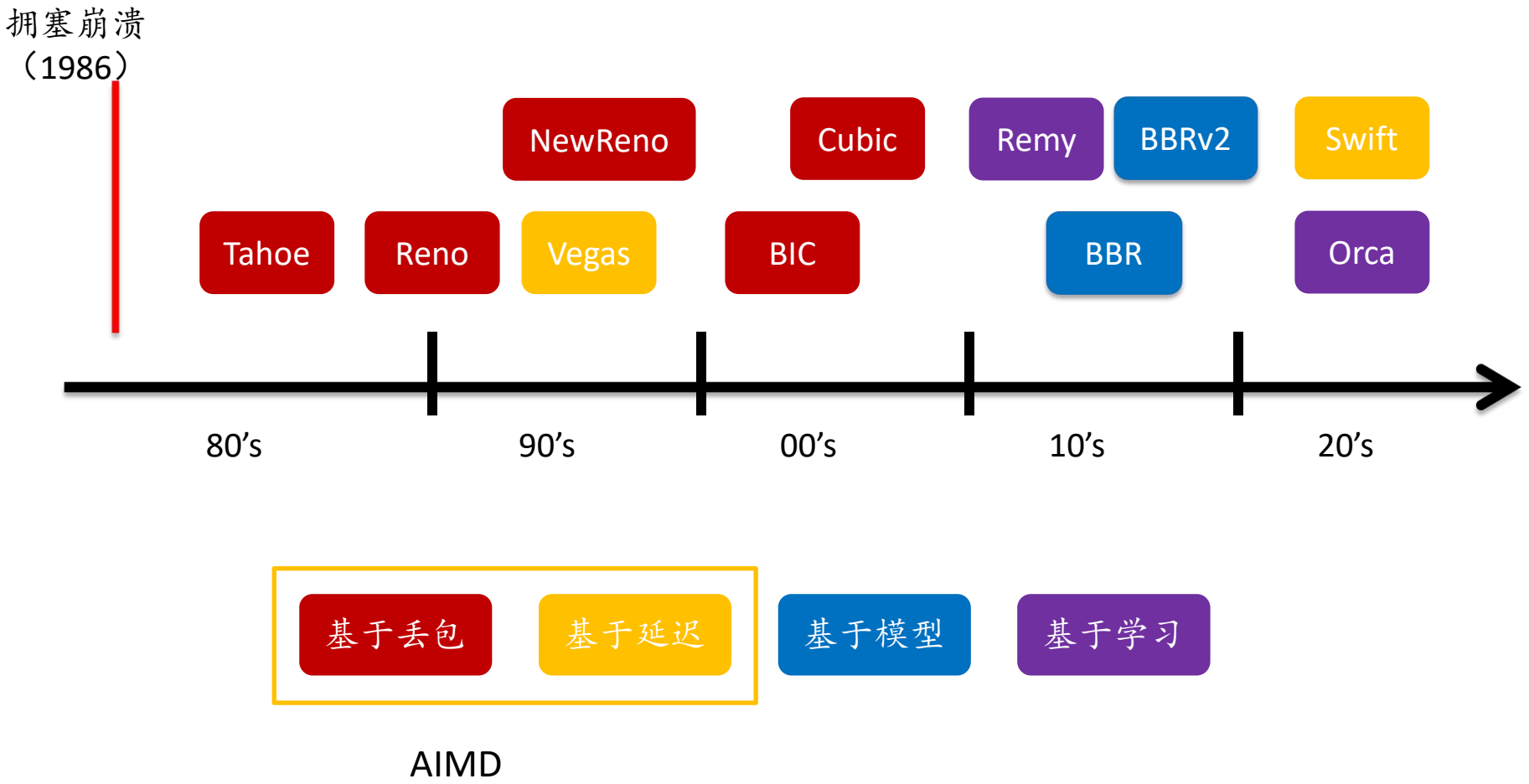
瓶颈带宽-RTT模型

- 先增加窗口探测可用带宽
- 再减小窗口探测最小RTT

学习驱动

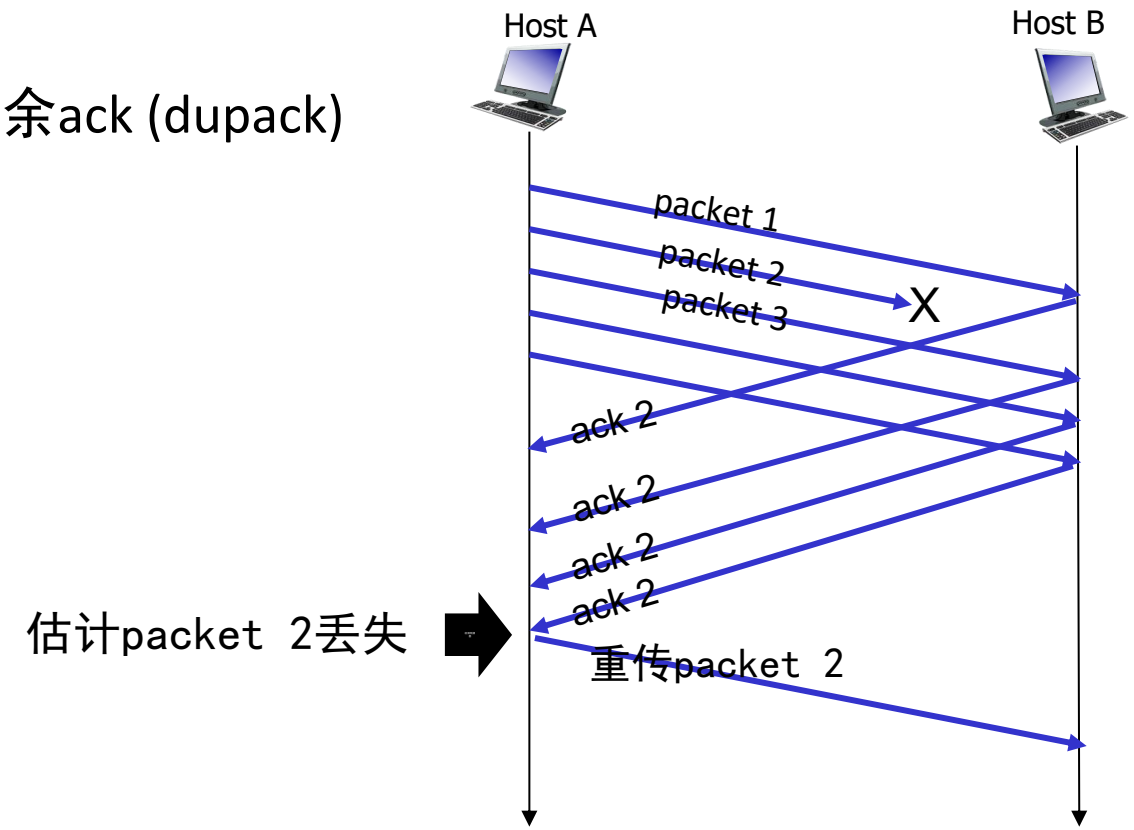
- 学习模型计算发送窗口大小
- 网络环境反馈丢包和RTT
- 使用强化学习方法在线学习

拥塞控制的发展历史


拥塞崩溃
（1986）
NewReno
Cubic
Remy
BBRv2
Swift
Tahoe
Reno
Vegas
BIC
BBR
Orca
80’s
90’s
00’s
10’s
20’s
基于丢包
基于延迟
基于模型
基于学习
AIMD

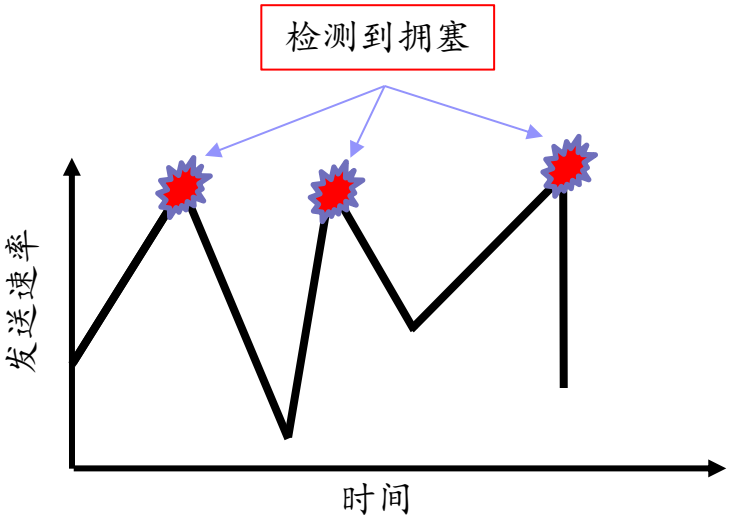
基于丢包的拥塞控制：丢包作为拥塞信号

- 冗余ack (dupack)



拥塞控制的基本方法——速率控制

- 速率控制：
  - 没有发生拥塞，增大发送速率
  - 收到拥塞信号，减小发送速率



关键问题：速率的增减方式

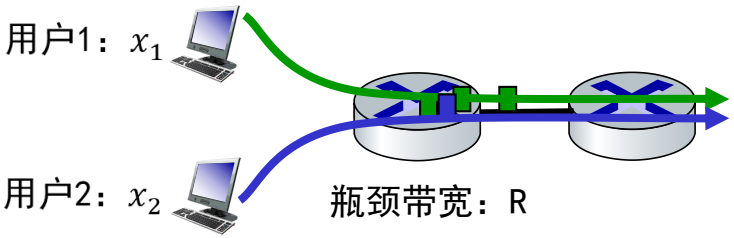
速率的增减方式

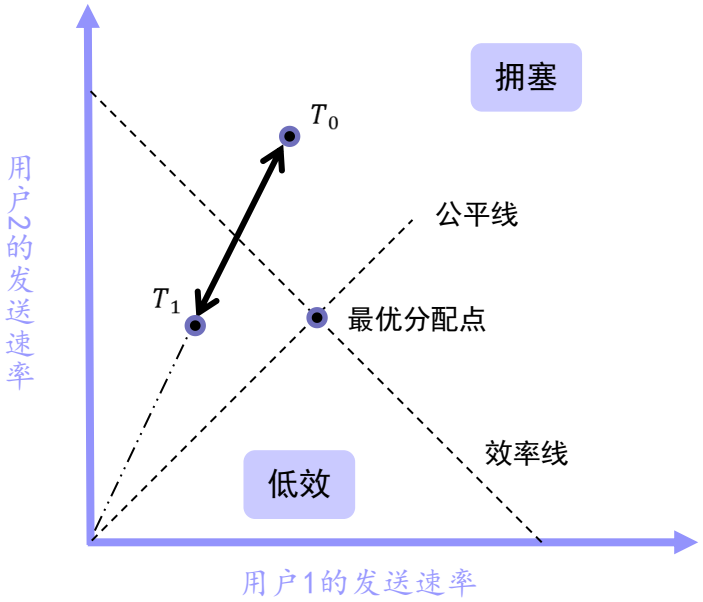
- 线性控制： $x(t+1)=\begin{cases} a_I + b_I x(t), & \text{无拥塞} \\ a_D + b_D x(t), & \text{拥塞} \end{cases}$
- 四种不同的控制组合：
  - 增策略：加/乘（AI/MI）
  - 减策略：加/乘（AD/MD）
- 需要根据拥塞控制目标来选择
  - 充分利用网络资源
  - 公平性

乘性增+乘性减（MI+MD）

- x_1、x_2同时乘上相同的增减系数
  - 不改进公平性

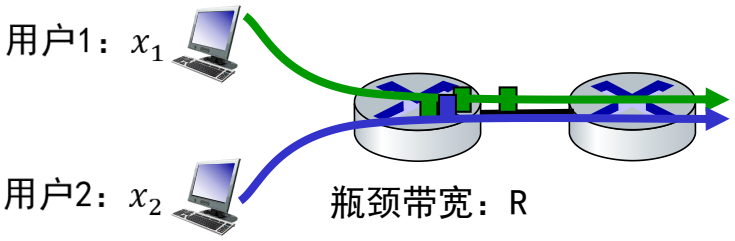
$$x(t+1)=\begin{cases} b_I x(t), & \text{无拥塞} \\ b_D x(t), & \text{拥塞} \end{cases}$$

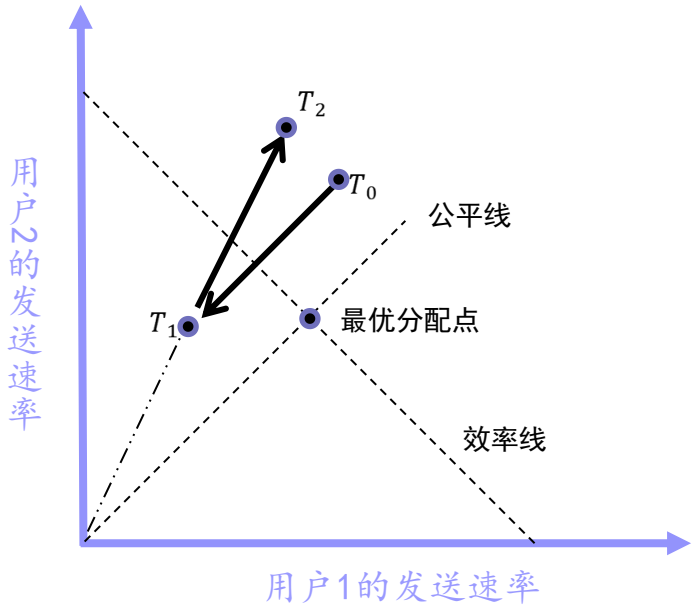




乘性增+加性减（MI+AD）

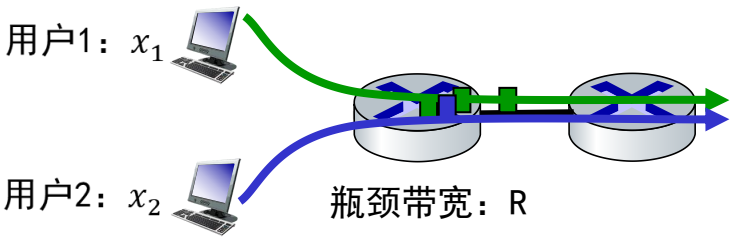
- 无法收敛到最优点

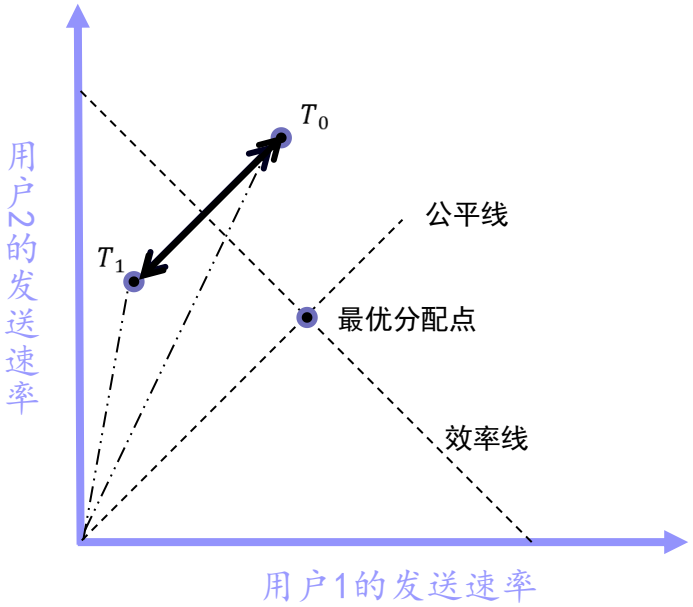




加性增+加性减（AI+AD）

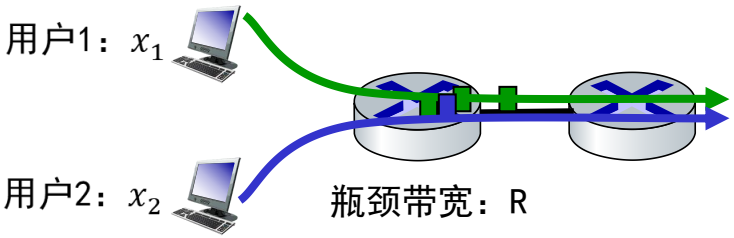
- x_1、x_2同时增减相同的量
  - 可以改进公平性
  - 但不会收敛到最优点

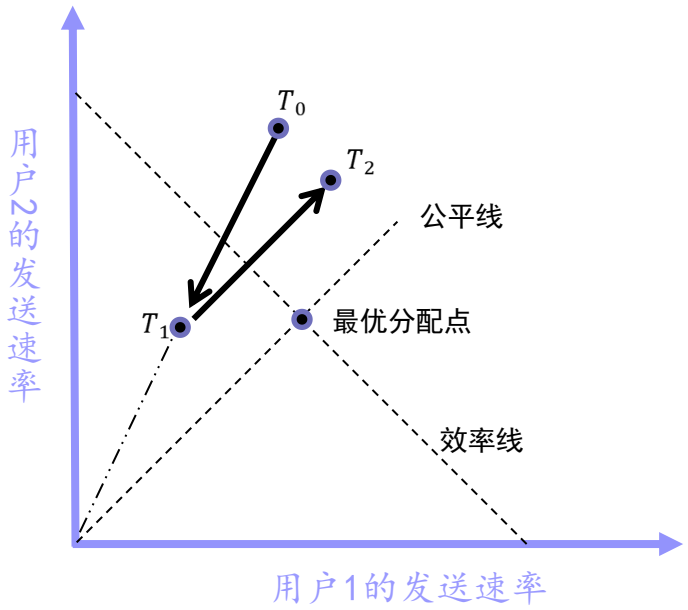




加性增+乘性减（AI+MD）

- 可以收敛到最优点
- 可以改进公平性





AIMD拥塞控制逻辑

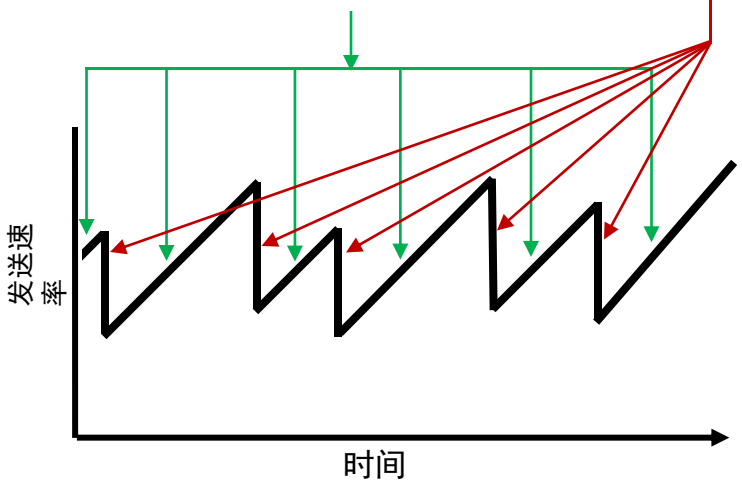
- 没有拥塞：加性增；发生拥塞：乘性减

Additive Increase

单位时间增加的发送速率为常数

Multiplicative Decrease

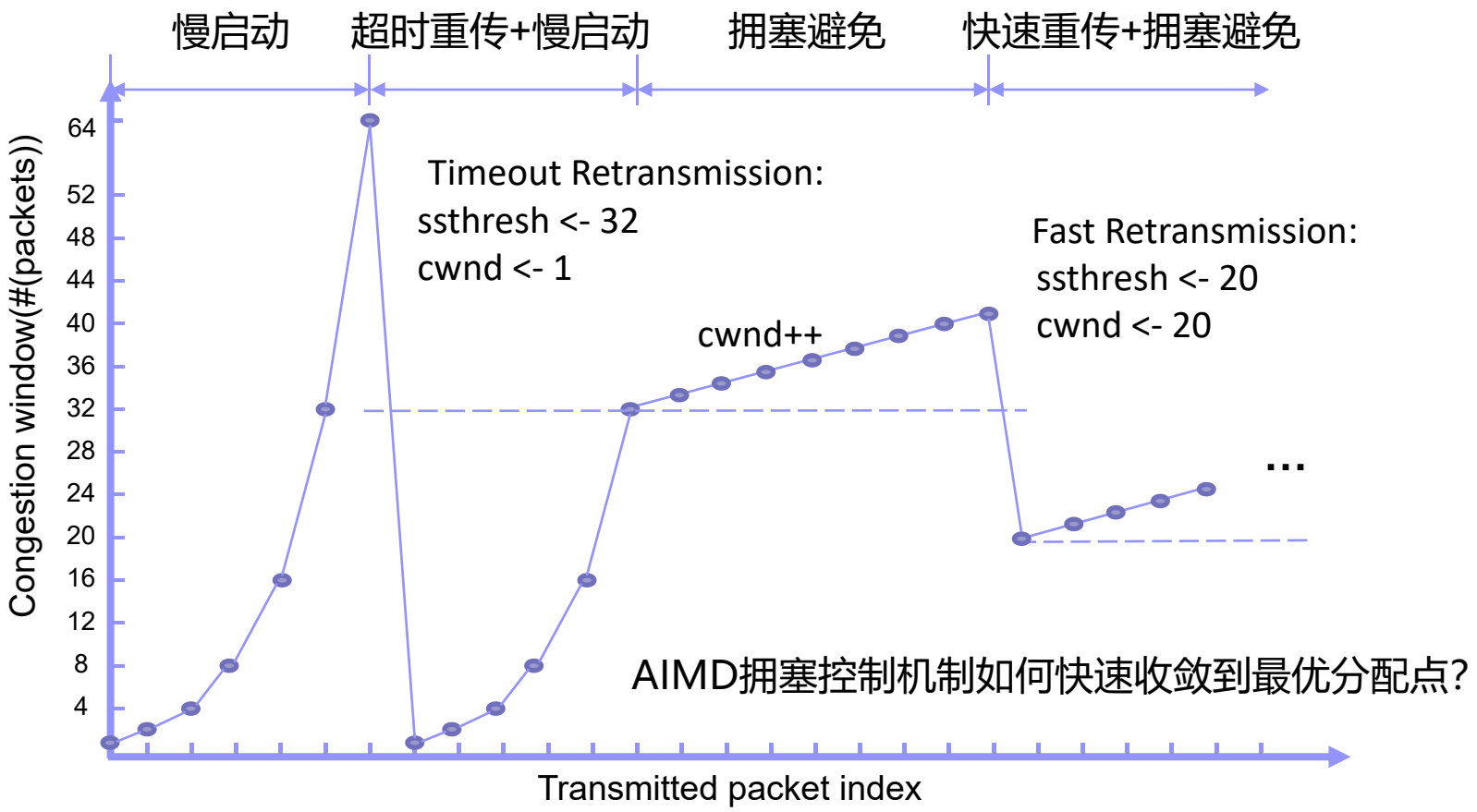
检测到拥塞，发送速率减半



基于AIMD的拥塞控制算法：NewReno

- TCP中通过控制窗口（$cwnd$）来控制发送速率，表示允许发送方发到网络中数据量
- Reno算法：慢启动+拥塞避免+快速恢复
  - 慢启动：单位时间$cwnd$翻倍，发送速率快速增长，达到稳定运行状态
  - 拥塞避免+快速恢复：稳定运行状态，窗口$cwnd$锯齿状变化

NewReno流拥塞窗口调节过程

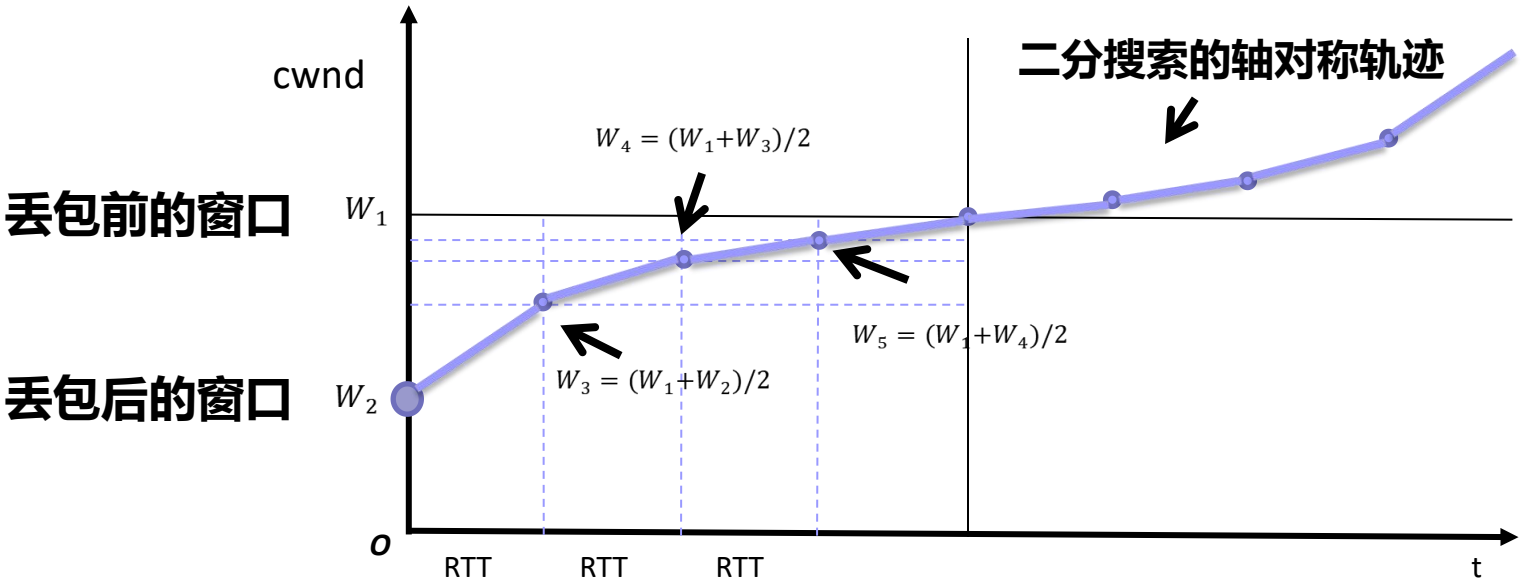


NewReno的两个主要问题

- 问题一：RTT不公平，RTT小的流更容易获得更高吞吐率
  - 拥塞避免阶段，窗口增长的速度与ack的到达速度呈线性关系
  - ack的到达速度与链路RTT的大小呈负相关
  - RTT较小的流更容易竞争到网络资源
- 问题二：在高RTT场景下带宽拥塞避免（AI）效率低
  - 例如，BW=10Gbps，RTT=100ms，MSS=1500B
  - 拥塞避免阶段窗口从1增长到充分利用带宽，需要1.2小时

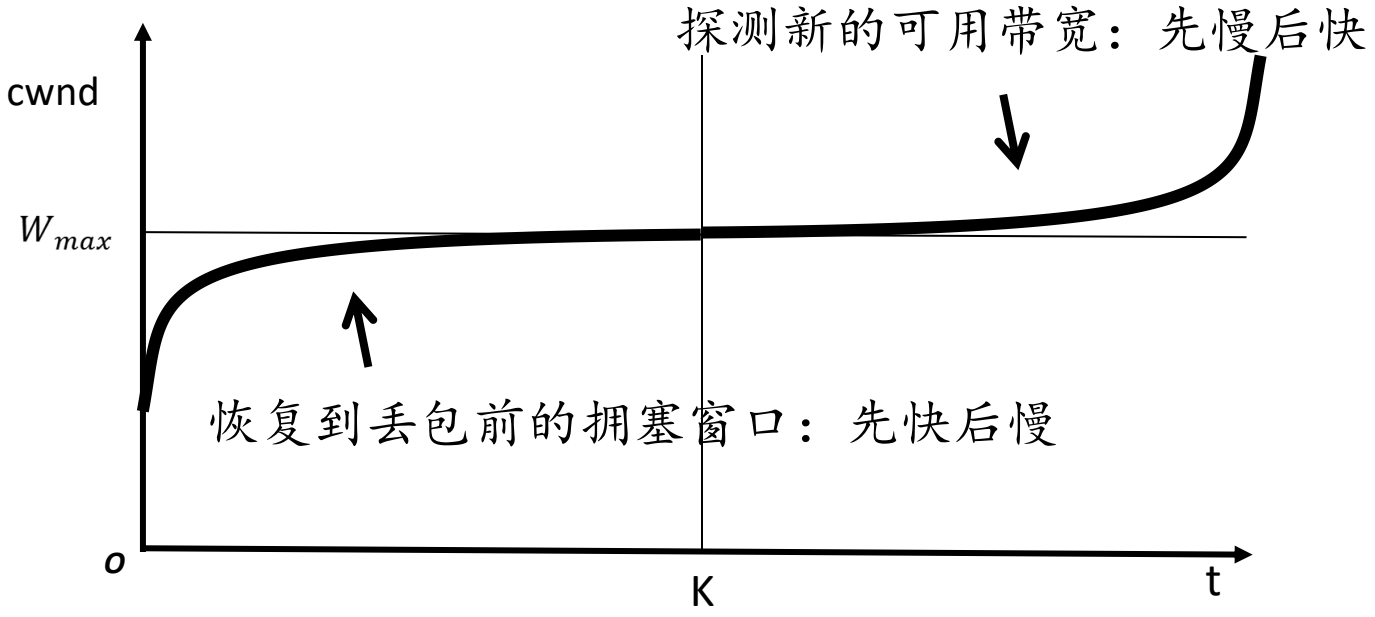
BIC：二分搜索拥塞窗口

- NewReno：线性增加拥塞窗口，速度太慢
- BIC：使用二分搜索找到合适的拥塞窗口



Cubic：基于三次函数的窗口增长策略

- RTT公平性：将拥塞窗口变为绝对时间t的函数：$cwnd = W(t)$，与RTT无关
- 探测效率低：将BIC窗口增长拟合为三次函数：$W(t) = C(t - K)^3 + W_{max}$



TCP Cubic

- **RTT公平性**
  - Reno：收到一个ack，窗口增量固定
  - BIC：每个RTT进行一次搜索，小RTT的会更快搜索到窗口
  - **Cubic：拥塞窗口变为绝对时间t的函数：$cwnd = f(t)$，RTT无关**

- **线性探测带宽太慢**
  - BIC：二分搜索的方法够快，但是实现复杂
  - **Cubic：窗口线性增长 → 三次函数，$f(t)$是一个关于t的三次函数**

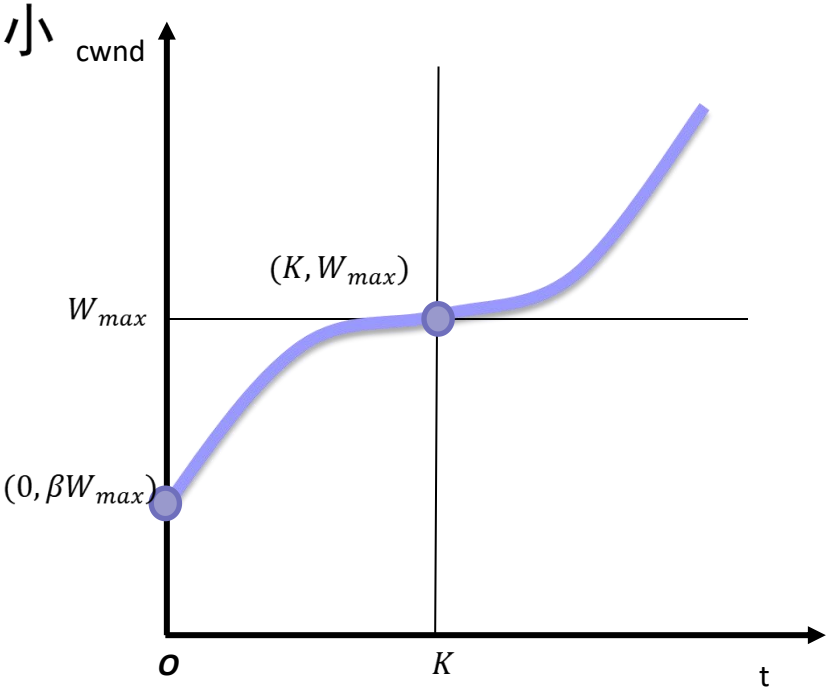
确定TCP Cubic的三次函数参数

- W_{max}：检测到丢包时候的拥塞窗口大小
- $cwnd$关于t的函数：

$$\mathrm{W}(t) = C(t-K)^3 + W_{max}$$

其中C决定了三次函数的形状，定值

- 关键参数
  - K：拥塞发生之后多久会重新达到W_{max}
  - 根据$W(0) = \beta W_{max}$算出：

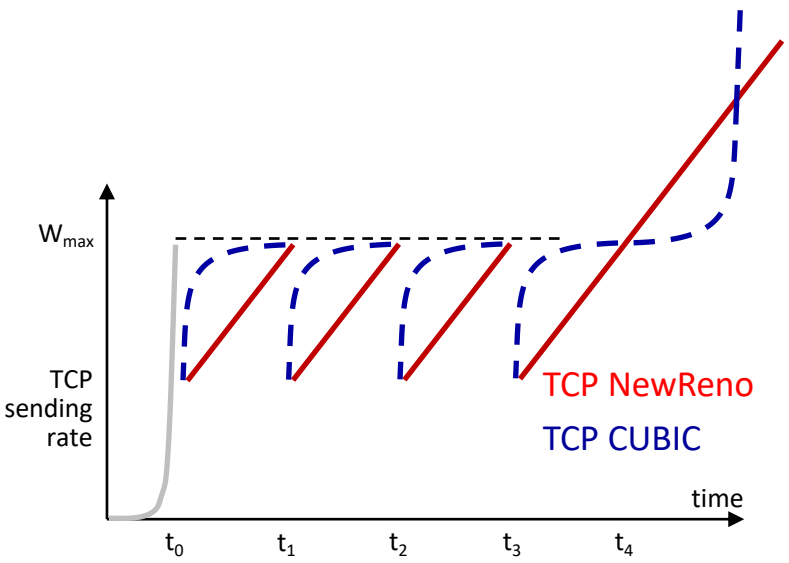


$$K = \sqrt[3]{\frac{(1-\beta)W_{max}}{C}}$$

TCP Cubic定性分析

- 当把窗口乘性减之后，根据提前计算好的三次函数曲线来增长拥塞窗口，开始很快，接近W_{max}的时候变慢

- 如果网络变得更好，在探测可用带宽时，拥塞窗口超过W_{max}之后，利用三次函数的另一侧，先慢后快增加拥塞窗口



小结：基于丢包的拥塞控制

- NewReno、Cubic均以丢包作为拥塞信号，Cubic改进了增窗方法和RTT公平性
- 发送方无法区分错误丢包和拥塞丢包。当网络中发生错误丢包，发送方放弃速率增长，不利于充分利用网络带宽
- Bufferbloat：倾向于填满网络中的buffer，增加排队时延，减小buffer的缓冲突发流量的作用

TCP Vegas：基于延迟的拥塞控制

- 延迟不仅反映拥塞是否出现，还可以体现拥塞程度
- TCP Vegas：基于RTT的测量，调整拥塞窗口
  - 拥塞信号反映了目标发送速率与实际发送速率之间的差值
- TCP Vegas的运行机制：
  - $expected\_Rate = \frac{cwnd}{base\_rtt}$
  - $actual\_Rate = \frac{cwnd}{rtt}$
  - $diff = expected\_Rate - actual\_Rate$
  - 调整拥塞窗口：$cwnd = \begin{cases} cwnd + 1, & diff < \alpha \\ cwnd - 1, & diff > \beta \\ cwnd, & otherwise \end{cases}$

TCP Vegas的问题

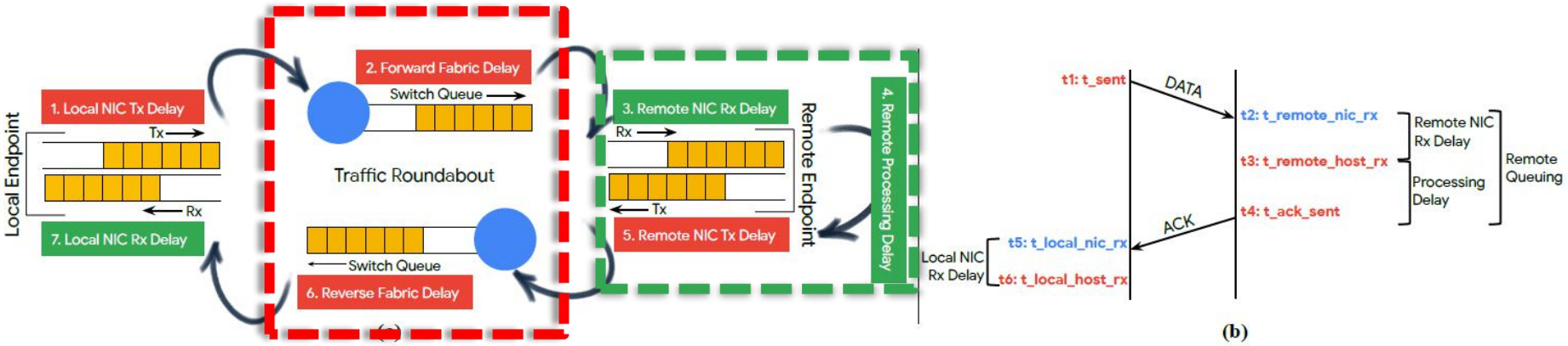
- 协议间的友好性
  - 与基于丢包的方法相比过于保守，竞争力不足
- 链路发生改变
  - 当链路路由发生改变，测量到的$base_rtt$失效。当$base_rtt$变大，Vegas发送方却不知道，认为网络发生了拥塞，发送减速
- 移动无线网络
  - 移动无线网络中的信号质量也会影响RTT变化，这并不代表网络拥塞程度

受控环境中基于延迟的拥塞控制：Swift

- 数据中心
  - 网络完全受控，拓扑信息已知，延迟可以准确测量
  - 无需与采用其他类型的拥塞控制算法的流竞争
- 发挥延迟作为拥塞信号的优势

Swift中的延迟测量

- 终端延迟：3. Remote NIC Rx Delay
  - $remote\ queuing\ delay = t_4 - t_2$
- 网络延迟：2. Forward Fabric Delay
  - $RTT - remote\ queuing\ delay$



Swift使用AIMD调整拥塞窗口

- 如果 测量延迟 < 目标延迟: AI
  - $cwnd \leftarrow cwnd + \frac{ai}{cwnd} \times num\_acked$
- 如果 测量延迟 > 目标延迟: MD
  - $cwnd \leftarrow cwnd \times max\left(1 - \beta\left(\frac{delay - target\_delay}{delay}\right), 1 - max\_mdf\right)$
- 怎样同时考虑终端拥塞和网络拥塞？维护两个窗口
  - $fcwnd$：根据网络延迟计算出来的窗口
  - $ecwnd$：根据终端延迟计算出来的窗口
  - 最终窗口大小 $cwnd = \min(fcwnd, ecwnd)$

Swift：目标延迟的调整

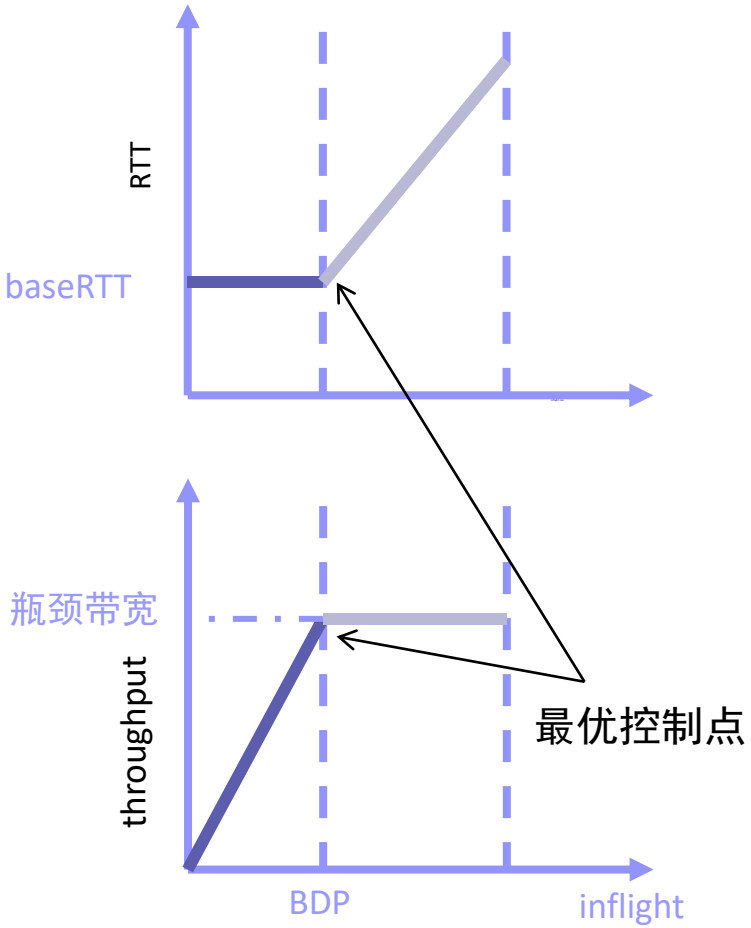
- 目标延迟不应该是固定的，需要根据网络拓扑和瓶颈中流的数量进行调整
- 目的：RTT公平性和公平分享带宽
- Topo-based（考虑距离远近，RTT公平性）
  - $topo\_scaling\_target = \#hops \times \hbar$
- Flow-based（考虑链路瓶颈中流的数量）
  - 观察：队列长度与$\sqrt{N}$成正比，N为流的数量
  - 瓶颈中流的数量无法知道，但是流的数量N与窗口大小$cwnd$成反比
  - 因此调整目标延迟与$\frac{1}{\sqrt{cwnd}}$成正比
- $Target\ delay = base + Topo_{based} + Flow_{based}$

小结：基于延迟的拥塞控制

- 延迟作为细粒度拥塞信号，不仅可以反映是否拥塞，还能表示拥塞程度
- 如果只考虑延迟作为拥塞控制信号，其带宽竞争性小于基于丢包的拥塞控制机制
- Swift等基于延迟的拥塞控制机制一般应用于受控环境中，例如数据中心网络

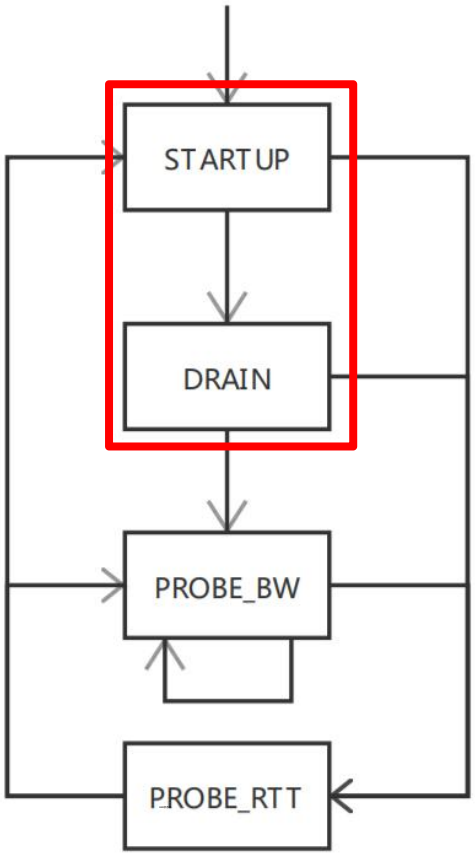
TCP BBR

- 主动感知网络，网络模型通过两个参数刻画：
  - 瓶颈带宽：$BtlBW$
  - 链路传播时延：$RTprop$
- $BtlBW$和$RTprop$不能同时测得
  - 在BDP左侧测得Rtprop；在BDP右侧测得BtlBW
- 网络状态在不断变化
  - 探测+回退机制，获得准确、即时的网络模型



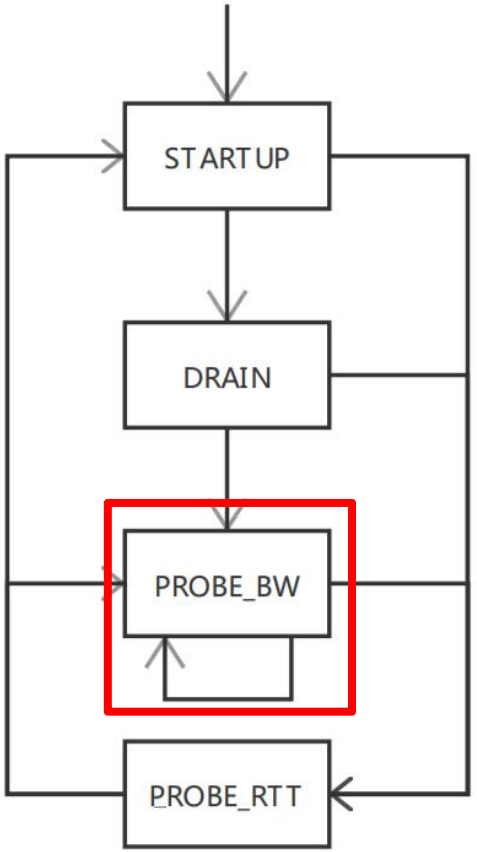
TCP BBR启动阶段：探测+回退

- BBR流开始阶段：
  - 探测：STARTUP状态；指数增长发送速率，快速探测可用带宽，直到观测到的带宽不再增长
  - 回退：DRAIN状态；经过了STARTUP阶段，此时网络中的inflight数量已经大于BDP，并非运行在最佳控制点；降低发送速率到估测带宽以下，把超过BDP的数据包排空
- 之后进入PROBE_BW稳定状态



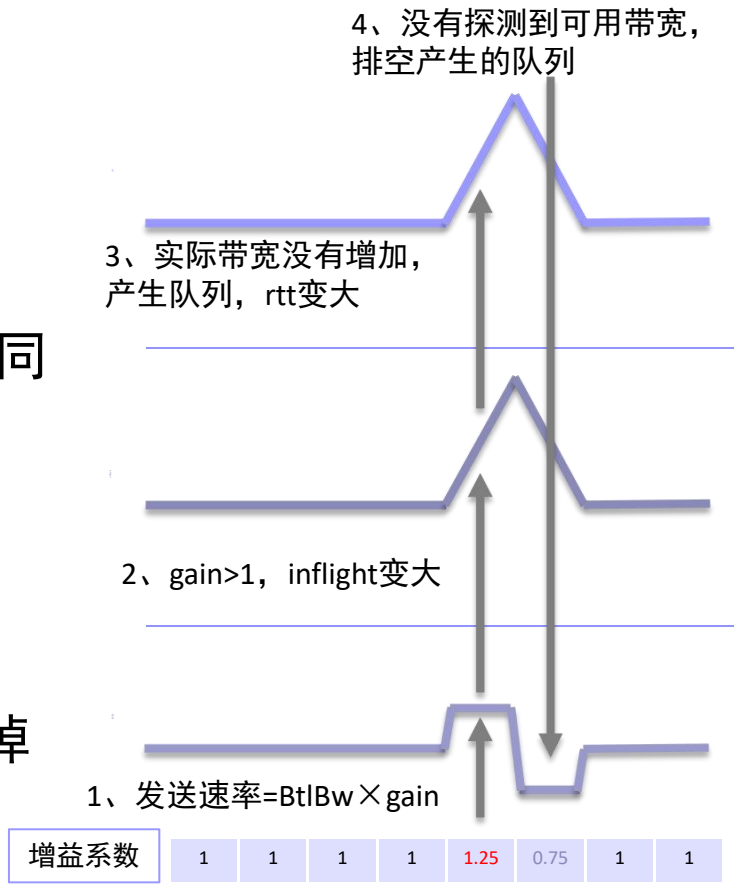
TCP BBR稳定状态：探测+回退

- PROBE_BW状态：
  - 实际发送速率=估计带宽×增益系数
  - 分成8个阶段不断循环，每个阶段有不同的增益系数：[1.25，0.75，1,1,1,1,1,1]
  - 1.25：探测是否有可用带宽
  - 0.75：回退，把探测带宽多发的包排掉
  - 1：以估测到的带宽进行发送



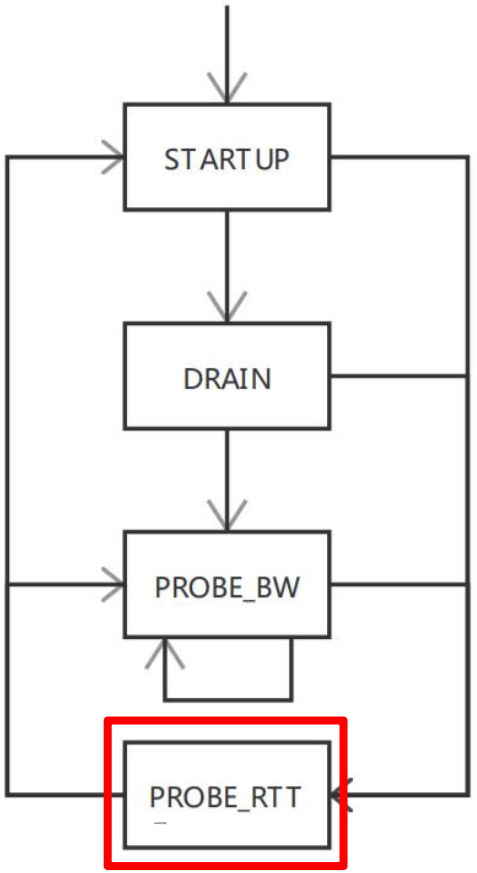
TCP BBR稳定状态：探测+回退

- PROBE_BW状态：
  - 实际发送速率=估计带宽×增益系数
  - 分成8个阶段不断循环，每个阶段有不同的增益系数：[1.25，0.75，1,1,1,1,1,1]
  - 1.25：探测是否有可用带宽
  - 0.75：回退，把探测带宽多发的包排掉
  - 1：以估测到的带宽进行发送

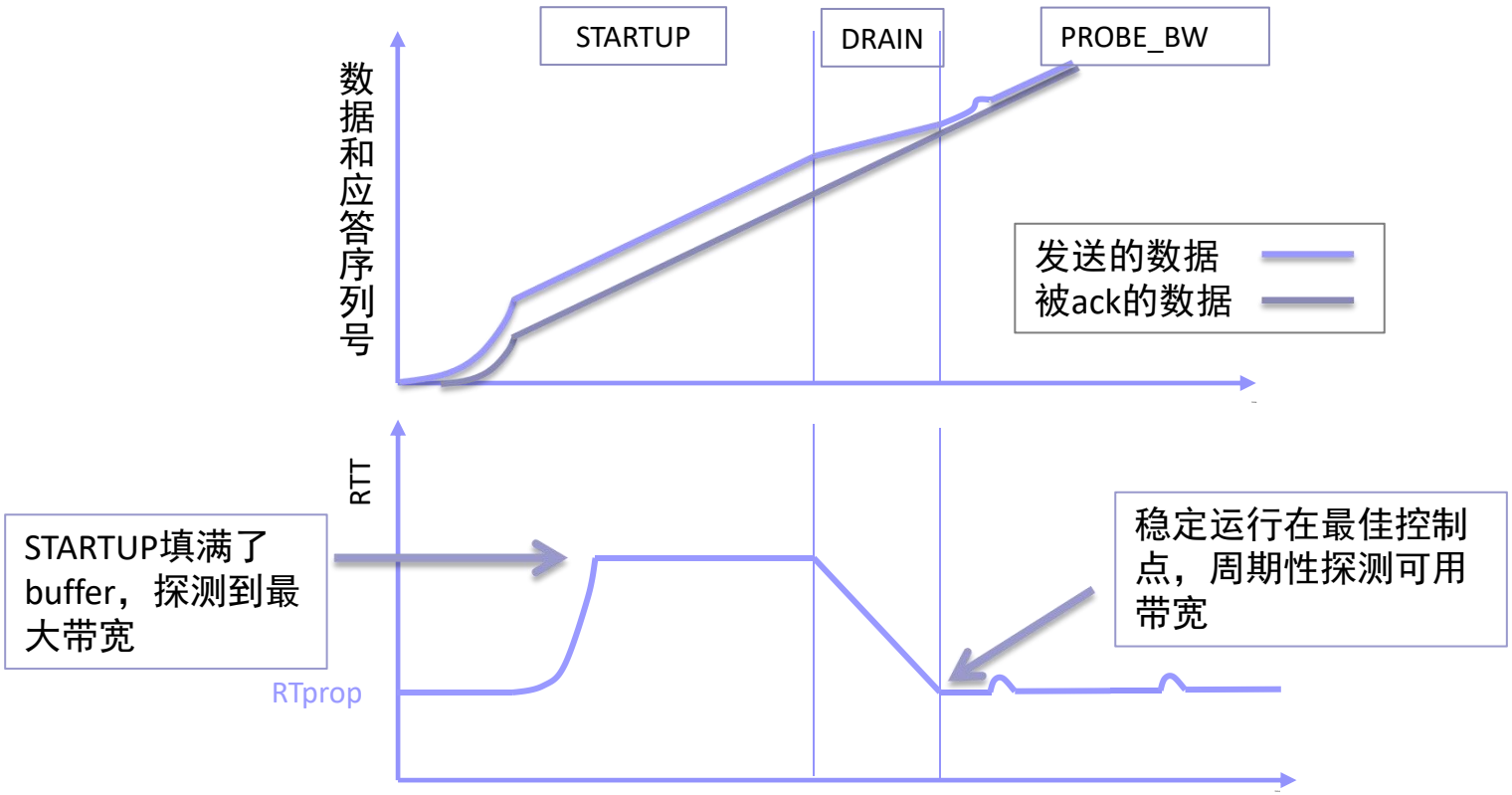


TCP BBR：探测延迟

- PROBE_RTT
  - 长时间（10s）没有探测到更小的RTprop，主动进行延迟探测：把发送窗口设为4个包，持续至少一个RTT，把这段时间观察到的最小RTT作为新的RTprop



TCP BBR：一条流的典型行为



TCP BBR的问题

- 完全忽略丢包，为了提高带宽有可能造成高丢包率
- PROBE_RTT状态造成的吞吐率波动较大
- 收敛需要时间较长

TCP BBRv2

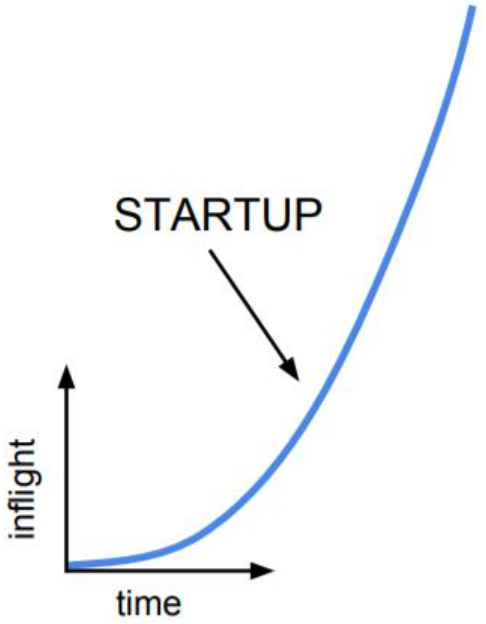
- BBR的网络模型：BtlBW、Rtprop
- BBRv2考虑更多的方面：丢包、ECN、inflight data
  - 基于丢包、ECN信息，为发送方估计一个inflight数据量的安全上界（即发生丢包前的inflight）
  - 调整PROBE_RTT，“少量多餐”
    - 减小延迟探测带来的吞吐率波动
    - 尽快更新路径时延的估计值，避免过低估计路径时延，导致拥塞窗口过小从而影响带宽利用率

TCP BBRv2的改进：感知丢包

- 不再忽略丢包，设定明确的丢包率上限：

  loss_ceiling(1%)

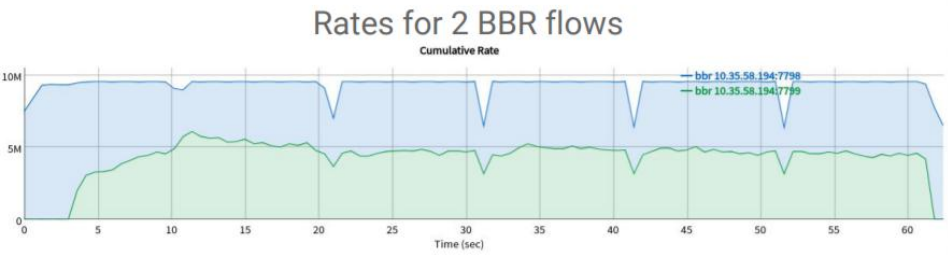
- 更早退出STARTUP阶段：
  - 三个RTT带宽没有增长25%，退出STARTUP，进入DRAIN
  - 如果丢包率超过loss_ceiling且丢包个数超过上限，退出STARTUP，进入DRAIN



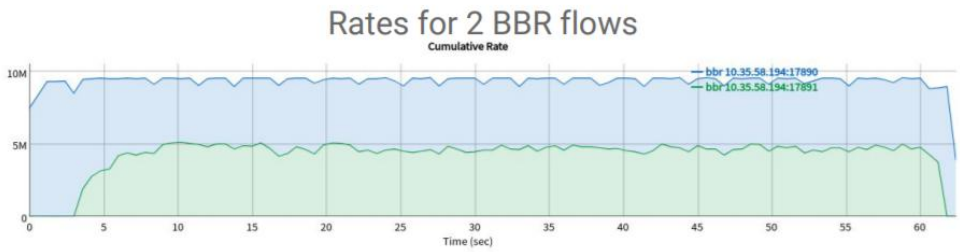
TCP BBRv2的改进：延迟探测带来的带宽波动

- PROBE_RTT状态更频繁、更不剧烈——“少量多餐”
- 多餐: 10s → 2.5s
- 少量: inflight: 4 packets → 0.75x estimated_bdp

Before (BBR v1):



After (BBR v2):

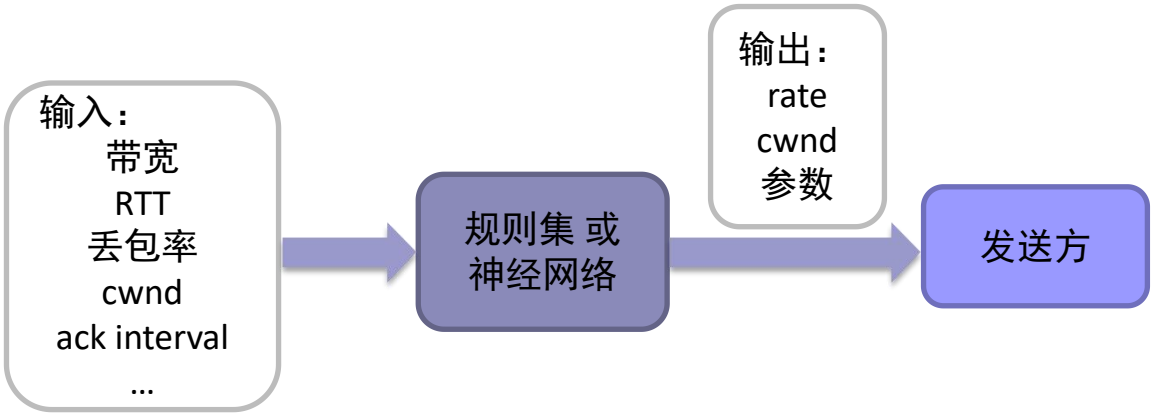


小结：基于瓶颈带宽-RTT模型的拥塞控制

- 优点：
  - 运行在最优控制点，不损失吞吐的情况下降低延迟
- 不足：
  - BBR过于激进：与其他流竞争时存在公平性问题
  - BBRv2：BBRv2的实际性能仍有待于广泛部署和验证

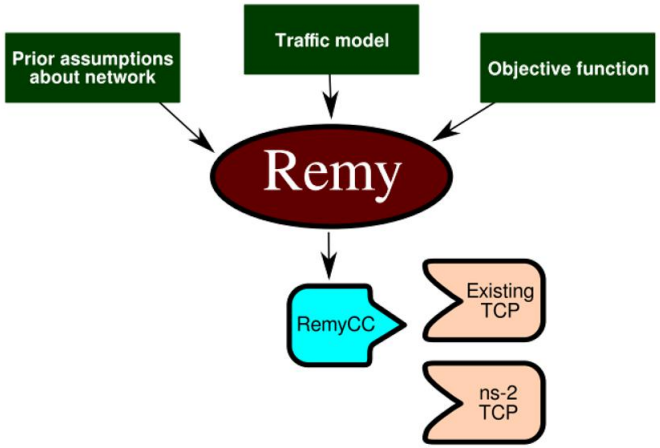
基于学习的拥塞控制

- 经典拥塞控制算法： 基于对网络的假设，人工制定规则
- 基于学习的方法：希望把人工制定规则的过程自动化，搜索到适用于不同网络环境的方法，得到一系列规则集，或用神经网络来表示，得到一个黑盒



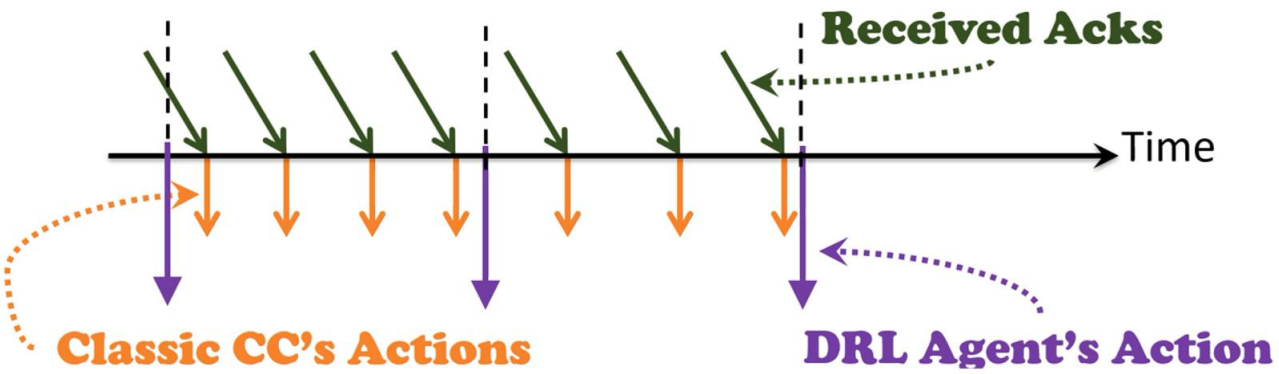
Remy：离线生成规则

- 根据输入的网络模型、流量模型和目标，生成拥塞控制算法
- RemyCC：一组规则
  - [网络状态] → [拥塞窗口的计算参数、是否发包]
  - ……



Orca：传统方法与学习方法结合

- 两层的控制逻辑
  - 粗粒度控制：深度强化学习的输出
  - 细粒度的控制：传统方法的输出



Orcas are believed to be the most intelligent mammals(excluding humans).

两层控制的好处

- 增强收敛性
  - 使用传统方法对拥塞窗口进行调节，相当于对强化学习做出的决策加入扰动，降低了收敛到错误位置/不收敛的可能
- 决策更容易理解
  - 传统CC：容易理解；完全基于学习的CC：决策难以理解
  - Orca：两者的结合

小结：基于学习的拥塞控制

- 基于学习的拥塞控制机制可以自适应于不同的网络环境，无需手动调整参数
- 不足：
  - 面对没有见过的网络环境时可能存在收敛性问题
  - 生成的算法是一个黑盒，缺乏可理解性
  - 复杂学习模型的时间、空间复杂度相对较高