

共享需要协调！——困难性对比

内核怎么协调共享（存储与处理器）？

- 存储：用（**页**）表进行简单的映射，通过虚拟化，进行共享
- 处理器：用（**线程**）进行换入/换出，通过虚拟化，进行共享
- 二者都在**同一物理环境**下，协调是集中式的，处理是相对容易的

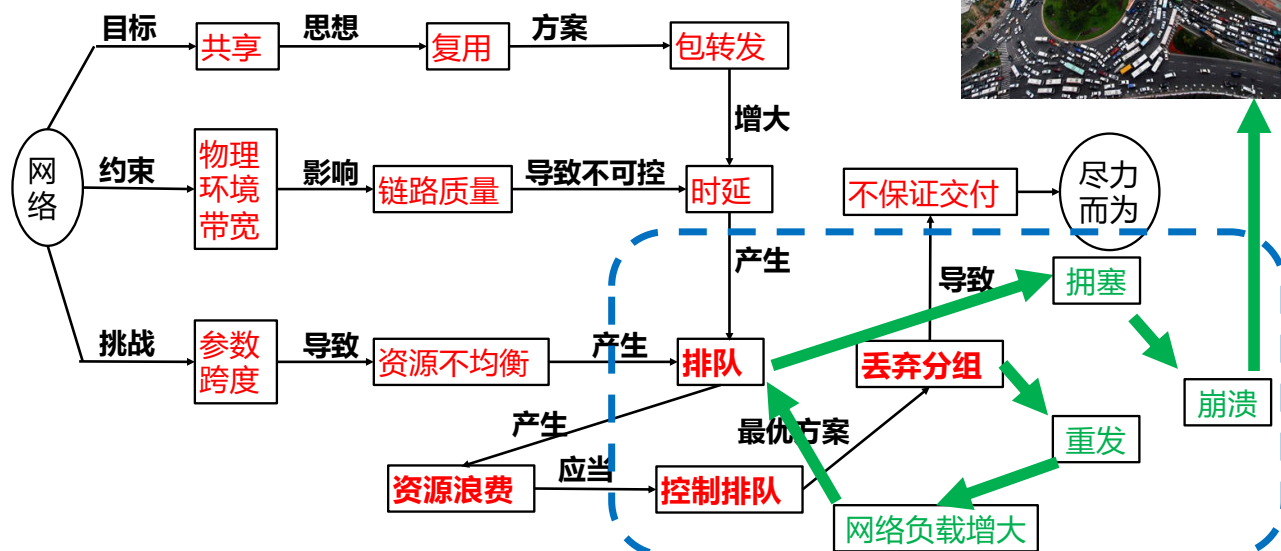
网络怎么协调共享（通信链路与转发器）？

- 链路、转发器的**分布式的共享**：
 - 地理和管理的分布式
 - 动态变化的性质
- 充满**未知**，更具**挑战**，所以我们要作为专题来进行研究

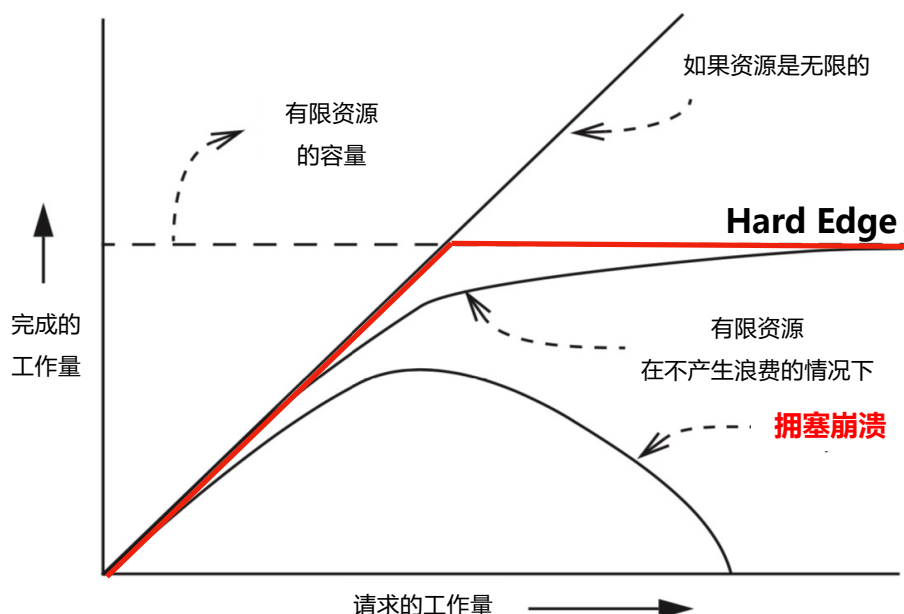
网络拥塞
是网络共享导致的问题
是分布式和动态变化的
需要从全局进行解决

尽力而为的互联网为何导致拥塞？

权衡取舍 (Trade-off) 的结果



现实：从拥塞到崩溃



第11章 作为总体的网络系统

11.1 网络拥塞与控制

11.1.1 拥塞的产生

11.1.2 拥塞控制的思路

11.2 跨层合作进行拥塞控制

11.2.1 跨层合作：反馈与控制

11.2.2 TCP协议的技术演进

11.3 更多讨论

11.3.1 超额配备与看不见的手

11.3.2 回顾时延

11.3.3 案例分析

解决拥塞问题的重要性与时机

拥塞是互联网丢包的主要原因

拥塞很容易转入崩溃，所以**避免拥塞**是首要目标！——拥塞控制

- 解决问题的思路

▸ 从吞吐量下降时开始防范

- Tahoe、Reno等

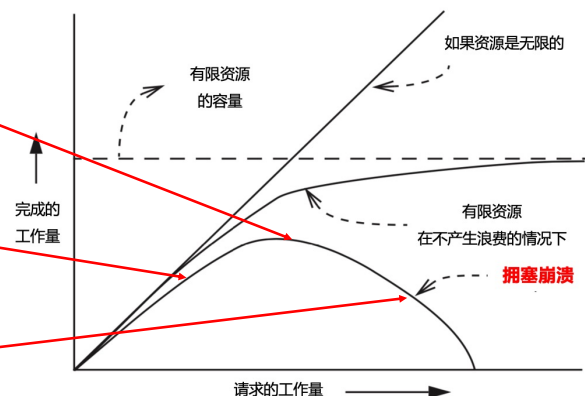
▸ 从产生变化时开始防范

- Vegas等

▸ 崩溃发生时难以防范

- 只能考虑如何分配延迟

- 逐步恢复到拥塞前的状态


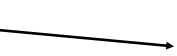


You can't squeeze blood out of turnip!

拥塞控制的思路

目标：防止长时间排队

思路：尽快恢复到容量大于负载的状态

- 增加容量  增加容量不能在短时间内进行,
- 减少负载  唯一可行办法是减少负载

- **回顾第9章：平均队列长度 = $\rho/(1-\rho)$

减少负载的思路

通过控制，使实际提交的网络负载低于节点期望提交的负载

- offered load < intended load
- 削减的负载怎么办？
 - 借助非拥塞时段来摊平

与排队对比

- 相同：都是为了解决服务能力不足，都是利用摊平来确保服务
- 不同：排队利用缓冲区，减少负载需要负载生产者的参与

如何通知负载生产者？

与现实类比：超市柜台如何解决拥塞？

- ~~提高服务容量——网络不可行~~ ×
- 减少负载，让队伍变短 ✓
 - 思路1：与队伍源头联系
 - 通知端系统，在网络上可行吗？

通知端系统的困难性

1. 反馈到控制点需要较长的时间，面临较多的变数，**怎么反馈？**
(如何进行快速有效的反馈？)
 2. **反馈给谁？** 给端系统的网络层模块，还是上层模块？端系统模块有能力配合吗？(如何找到正确的模块？)
 3. 端系统模块都**愿意配合吗？** (哈定悲剧的教训.....)
- 综上：通知端系统，需要做**跨层协议合作**
 - 面临的限制条件：须与原有协议兼容

因特网的规模增长巨大，但到现在只有微小调整👍

从这个案例，瞧瞧他们是怎么做到的？

第11章 作为总体的网络系统

11.1 网络拥塞与控制

11.1.1 拥塞的产生

11.1.2 拥塞控制的思路

11.2 跨层合作进行拥塞控制

11.2.1 跨层合作：反馈与控制

11.2.2 TCP协议的技术演进

11.3 更多讨论

11.3.1 超额配备与看不见的手

11.3.2 回顾时延

11.3.3 案例分析

怎么反馈？

与现实类比：超市柜台如何解决拥塞？

- 有没有更巧妙的办法来通知负载生产者？
 - 用排队时间来控制



网络：

1. 如何让负载生产者**知晓**排队时间？
2. **谁**是能控制发送速度的负载生产者？
3. 如何让负载生产者愿意**配合**？

公共地悲剧（哈定悲剧）

“设想有一片对所有人开发的牧场……作为理性人，每个牧羊人都会寻求最大化收益……他会问：“如果我的羊群增加一只羊，对我有何效用”？效用包括正和负两个部分……由于牧羊人从增加的羊获得销售收益，正效用接近于1。然而，由于过度放牧的后果由所有牧羊人分担，每个人分得的负效用只是-1的 $1/n$ 。”

“理性的牧羊人认为，当把各部分效用相加时，唯一明智的做法是增加羊的数量，再加一只……但是这个结论是所有理性的牧羊人共同得出来的，这就是悲剧所在。每个人都被禁锢在无限增加羊的系统中——而在资源有限的世界中……公共地的自由给所有人带来毁灭。”

— Garrett Hardin, *Science* 162, 3859

跨层合作的反馈方式

~~source quench → 不可行~~

~~flag “slow down” → 路径太长、风险大~~

丢弃一个包 → 虽然不保证行，但是简单可靠

- 丢哪个？这是个策略问题（问问deepseek？）
 - 尾丢弃（tail drop）：简单
 - 随机丢弃（random drop）：更好（why？）
 - 丢弃第一个：最容易在发送方超时的
 - 改进：early drop：快满的时候开始 - 预防
 - 合并：RED（random early detection）

跨层合作的控制方式

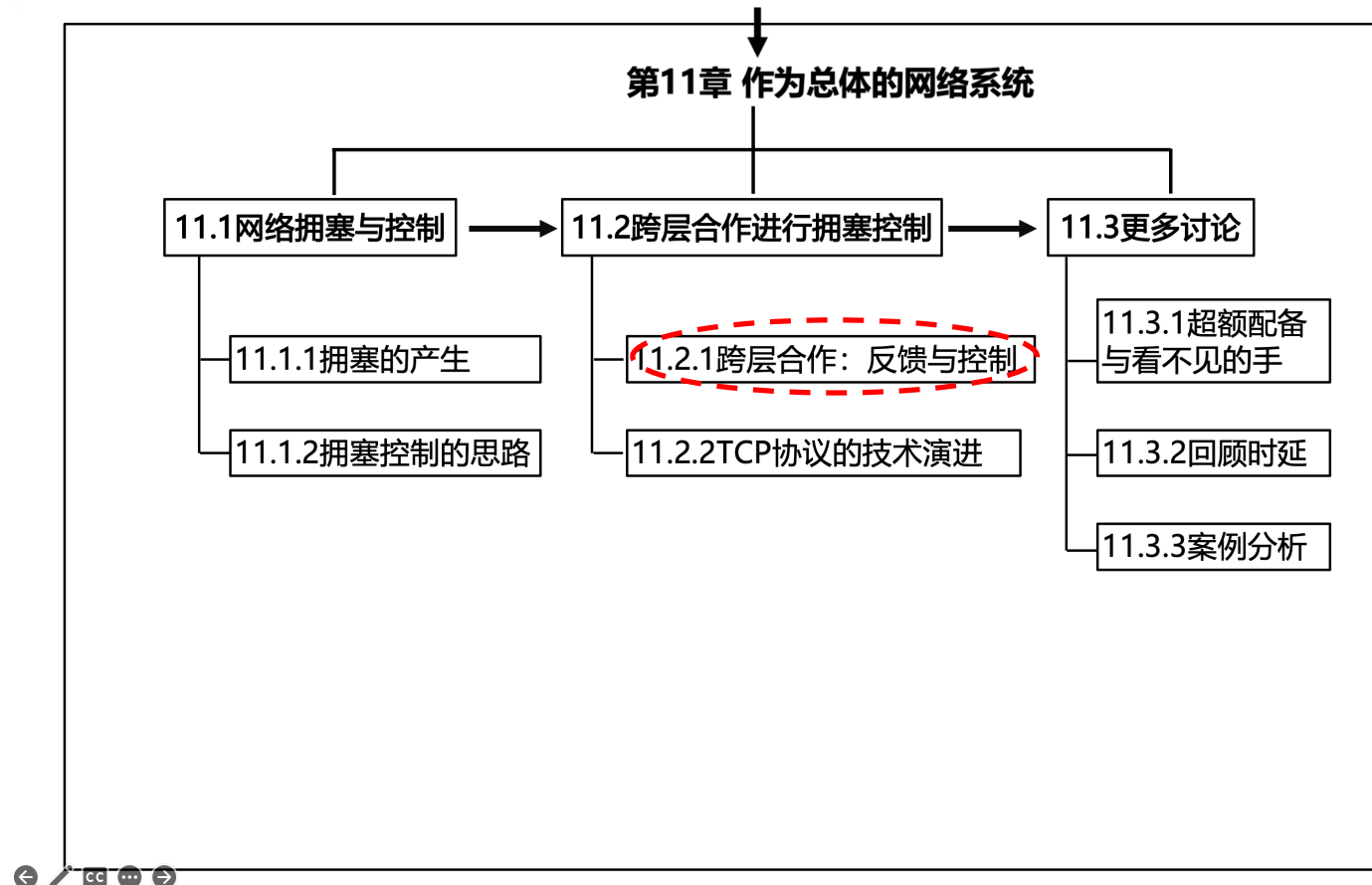
端到端协议的进程发现包丢弃，怎么办？

- 全力以赴发包，更容易成功 → 哈定悲剧
- 配合
 - 用**定时器**检测延迟与丢包
 - 延迟 → rtt增加，丢包 → 指数退避
 - Automatic Rate Adaptation
 - 丢包能否提示减小流量控制滑动窗口？窗口不变，rtt增加，意味着数据率降低？
 - 如果接收窗口过大，会导致链路上过长的排队。所以，发送方需要控制窗口不大于必要值

跨层合作的控制方式

TCP是如何防止拥塞的？

- （拥塞控制的副作用：从此难以估计网络的期望流量）



TCP协议

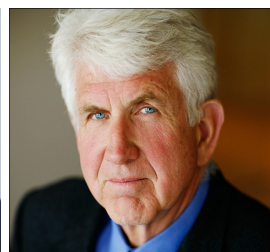
为在不可靠的包交换网络上可靠传输数据，Cerf和Kahn在1974年提出TCP/IP协议，同年TCP协议发布为RFC675

2004 ACM A.M. Turing Award
VINTON GRAY CERF, US ROBERT ELLIOT KAHN, US



For pioneering work on internetworking, including the design and implementation of the Internet's basic communications protocols, **TCP/IP**, and for inspired leadership in networking.

2016 ACM A.M. Turing Award 2022 ACM A.M. Turing Award
SIR TIM BERNERS-LEE, UK Bob Metcalfe, US



For inventing the **World Wide Web**, the first web browser, and the fundamental protocols and algorithms allowing the Web to scale. For Invention, Standardization, and Commercialization of **Ethernet**.

早期TCP协议

为在不可靠的包交换网络上可靠传输数据，Cerf和Kahn在1974年提出TCP/IP协议，同年TCP协议发布为RFC675

- 思路：端系统使用传输控制程序（Transmission Control Program）
- 特点：面向连接，可靠，使用字节流
- 工作方式：
 - 唯一应答方式：**收到X之前的包**，不通知未收到包的序号
 - 重发：计时器包括rtt和完全的流量控制窗口，超时立即重发，直到回应

TCP协议面临的问题

为在不可靠的包交换网络上可靠传输数据，Cerf和Kahn在1974年提出TCP/IP协议，同年TCP协议发布为RFC675

- 问题：
 - 长距离的路由往往伴随大的接收窗口，当过载导致丢包时，发送者会因超时而重发一个完整窗口的包！
 - **这导致了拥塞极易发生！但开始没有考虑拥塞控制**

10多年后，在TCP已经广泛使用的情况下发现了这个问题，怎么办？

限制条件与可行路线

限制条件

- 不能修改包格式

可行路线：

- 新实现能与旧实现互操作，并逐渐取代它

研究者的努力

先后提出几十种方案

Variant	Feedback	Required changes	Benefits	Fairness
(New) Reno	Loss	—	—	Delay
Vegas	Delay	Sender	Less loss	Proportional
High Speed	Loss	Sender	High bandwidth	
BIC	Loss	Sender	High bandwidth	
CUBIC	Loss	Sender	High bandwidth	
C2TCP ^{[11][12]}	Loss/Delay	Sender	Ultra-low latency and high bandwidth	
NATCP ^[13]	Multi-bit signal	Sender	Near Optimal Performance	
Elastic-TCP	Loss/Delay	Sender	High bandwidth/short & long-distance	
Agile-TCP	Loss	Sender	High bandwidth/short-distance	
H-TCP	Loss	Sender	High bandwidth	
FAST	Delay	Sender	High bandwidth	Proportional
Compound TCP	Loss/Delay	Sender	High bandwidth	Proportional
Westwood	Loss/Delay	Sender	Lossy links	
Jersey	Loss/Delay	Sender	Lossy links	
BBR ^[14]	Delay	Sender	BLVC, Bufferbloat	
CLAMP	Multi-bit signal	Receiver, Router	Variable-rate links	Max-min
TFRC	Loss	Sender, Receiver	No Retransmission	Minimum delay
XCP	Multi-bit signal	Sender, Receiver, Router	BLFC	Max-min
VCP	2-bit signal	Sender, Receiver, Router	BLF	Proportional
MaxNet	Multi-bit signal	Sender, Receiver, Router	BLFSC	Max-min
JetMax	Multi-bit signal	Sender, Receiver, Router	High bandwidth	Max-min
RED	Loss	Router	Reduced delay	
ECN	Single-bit signal	Sender, Receiver, Router	Reduced loss	

Jacobson提出的协议

实现思路：

- 发送者操纵流量控制窗口

实现要点：

1. 拥塞窗口 (CWND)
2. 慢开始+拥塞避免
3. 快重传 (收到重复确认)
4. 相平衡 (AIMD)
 - 加法增加
 - 乘法减少 (快恢复)
5. 重启

TCP窗口定义及规则

1. 窗口：

- 标记TCP可发送数据包数量的上限的TCP状态变量

2. 拥塞窗口cwnd：

- 用于拥塞控制目的的窗口，由发送者确定

3. 接收窗口rwnd：

- 用于流量控制目的的窗口，由接收者确定并发给发送者

4. 规则

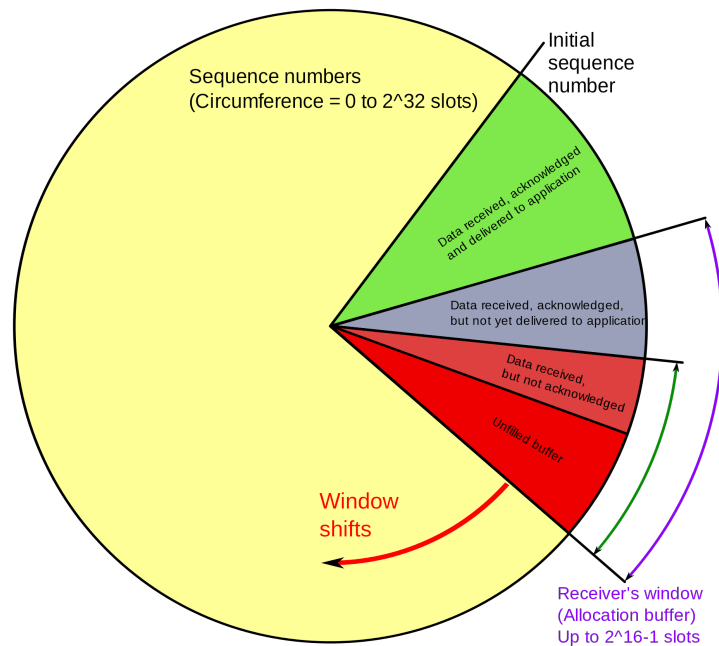
- 不能发送序列号大于 (最大回复序列号+Min{cwnd, rwnd}) 的包

5. 其他窗口

- 初始化窗口 (IW)：TCP发送者建立连接后的cwnd
- 丢失窗口 (LW)：TCP发送者通过超时检测到丢包后，设定的cwnd
- 重启窗口 (RW)：TCP经历空闲期之后设定的cwnd

6. SMSS：发送端最大报文段大小，取决于RMSS、MTU

窗口的滑动



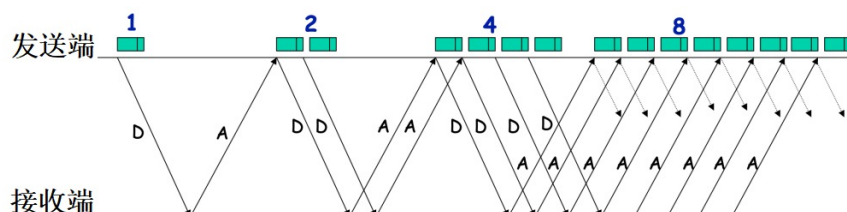
慢开始

发1W个包（通常为1-4），等待回应。每收到1个，窗口+1

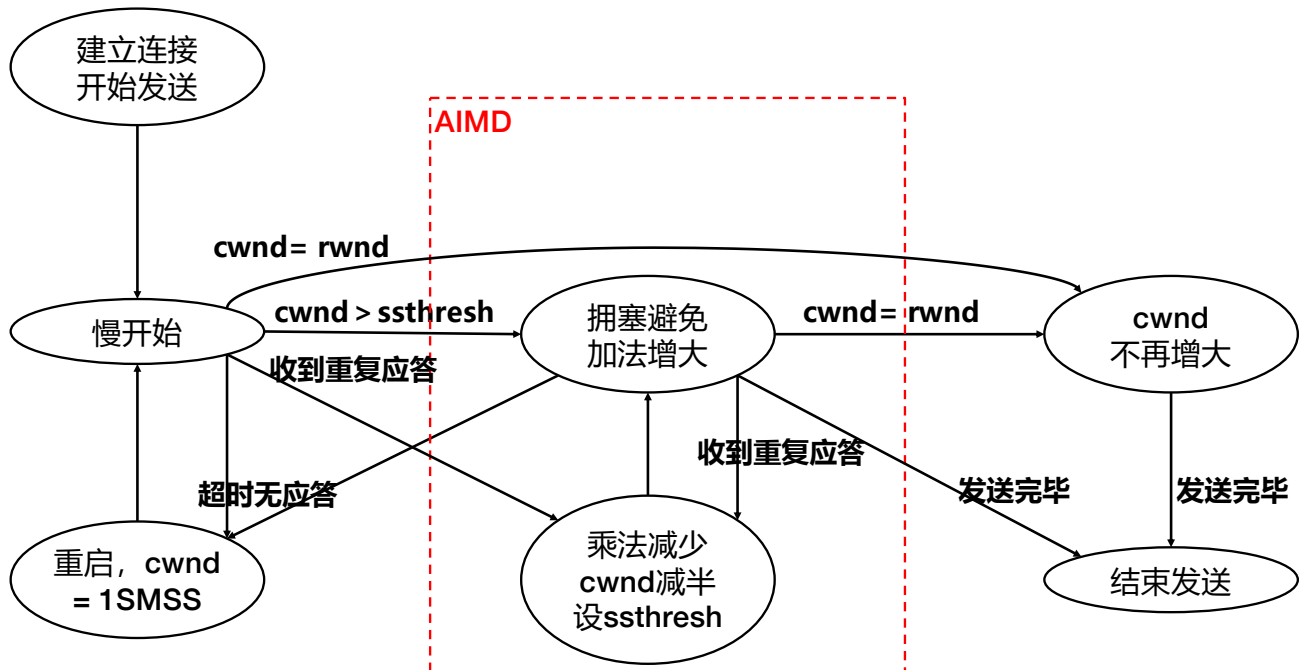
- 现象：每个rtt，发送方的发包翻倍

直至

- $cwnd = ssthresh$ （进入拥塞避免算法）
- $cwnd = rwnd$ 大小（此时网络不是瓶颈），持续发送完成，或
- 发送者检测到丢包（重启），或收到重复确认包（AIMD）



流程图 (Tahoo)



拥塞避免

慢开始阈值ssthresh：

- 用于确定使用慢开始还是拥塞避免算法的门限值
- 慢开始阶段可以设置为任意值，出现拥塞后进行调整
 - 丢包后： $ssthresh = \max(\text{FlightSize} / 2, 2 * SMSS)$
 - FlightSize：在途字节数
- 当 $cwnd < ssthresh$ 时，使用慢开始算法
 - 每个rtt, cwen指数增长
- 当 $cwnd > ssthresh$ 时，使用拥塞避免算法
 - 每个rtt大约cwnd增加1

当 $cwnd = ssthresh$ 时，应该使用什么算法？

重复应答

接收方的TCP实现做了轻微修改：

- 收到一个乱序包，则重复发送最后的应答包
- 思路：重复应答可被发送者解释为对未应答包的负应答

收到重复应答包，发送者

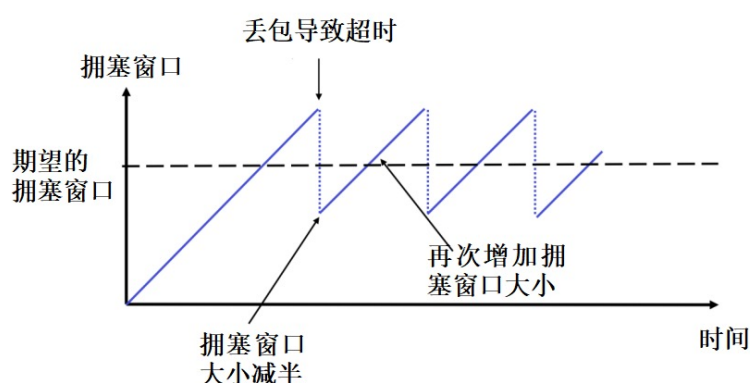
- 重新发送第一个未应答包
- 同时将窗口大小乘固定分数（如1/2）
- 进入相平衡模式

对TCP的修改：兼容旧版本，实施新算法，可过渡，满足限制条件和可行路线

AIMD：相平衡

持续观察重复应答，并尝试更多的容量（AIMD）：

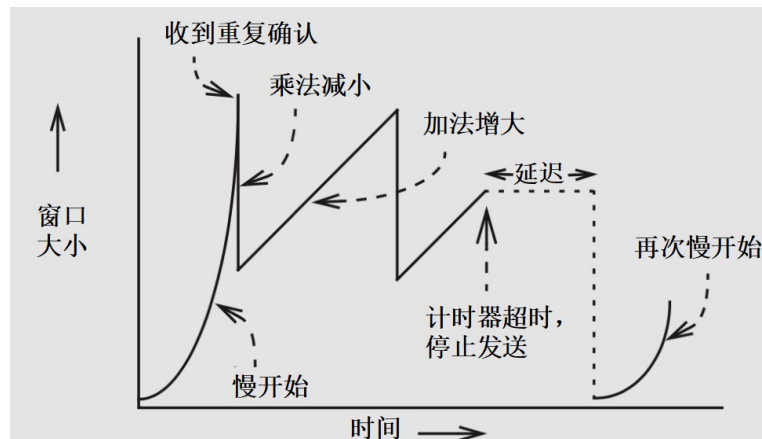
- 加法增加
 - 如发送成功，cwnd随rtt线性增加
 - 一般做法， $cwnd += \min\{n, SMSS\}$
- 乘法减少
 - 如丢包，按比例减少



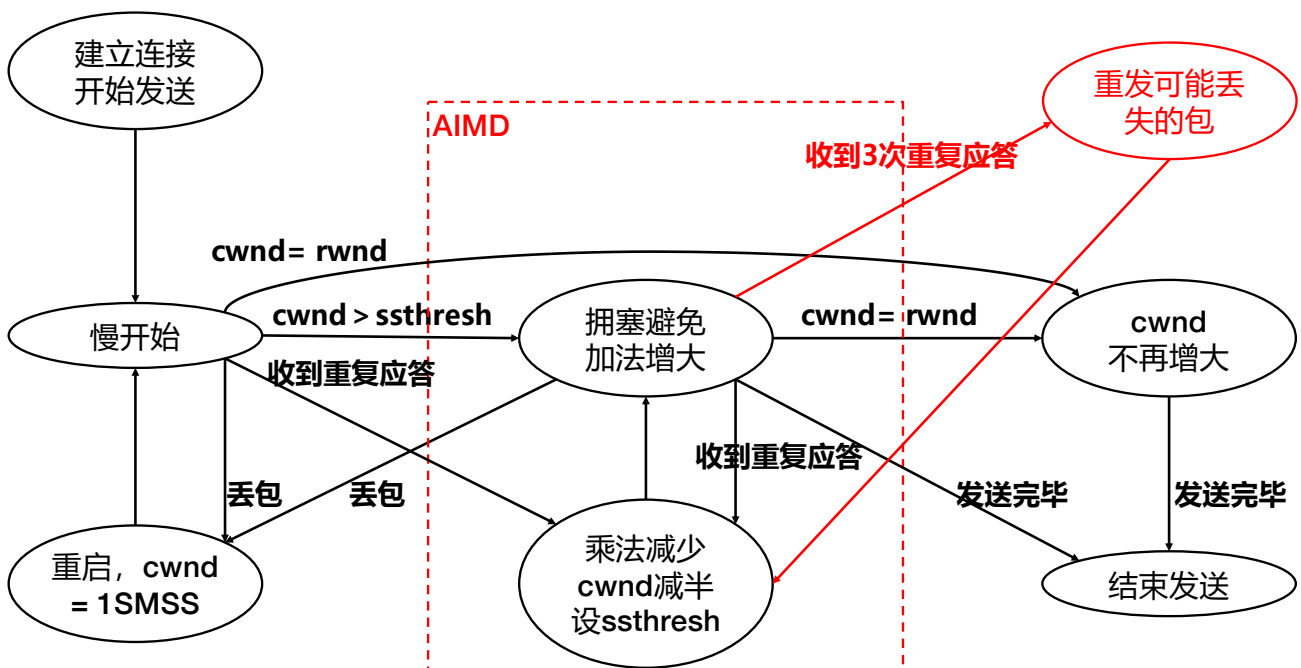
重启

如果计时器超时，网络可能发生了彻底改变

- 发送者等待一段时间
- 之后重新进入慢开始

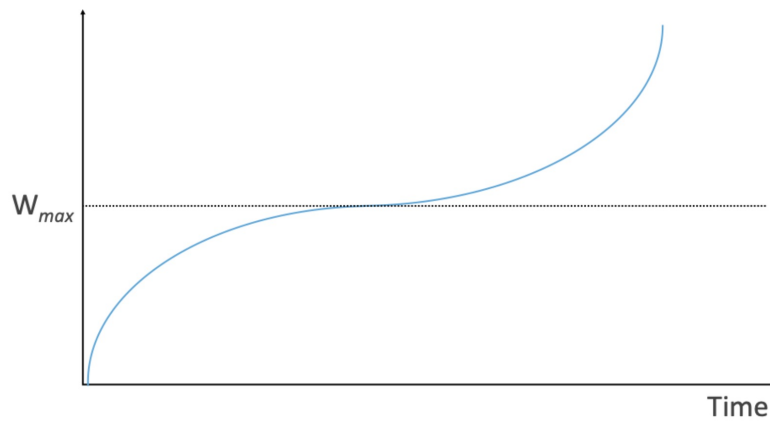


快重传 (Reno)



慢开始可以改进吗？

Binary Increase Congestion Control (BIC) 方法



Ha S, Rhee I, Xu L. CUBIC: a new TCP-friendly high-speed TCP variant[J]. ACM SIGOPS operating systems review, 2008, 42(5): 64-74.

讨论

公平性

- 应用只能使用TCP吗？
 - 使用其他协议的应用只获利不付出
- 应用只能使用1个TCP连接吗？
 - 如果同时使用n个连接，可能获得的通信资源是其他应用的n倍

该怎么去分析并改进？

讨论

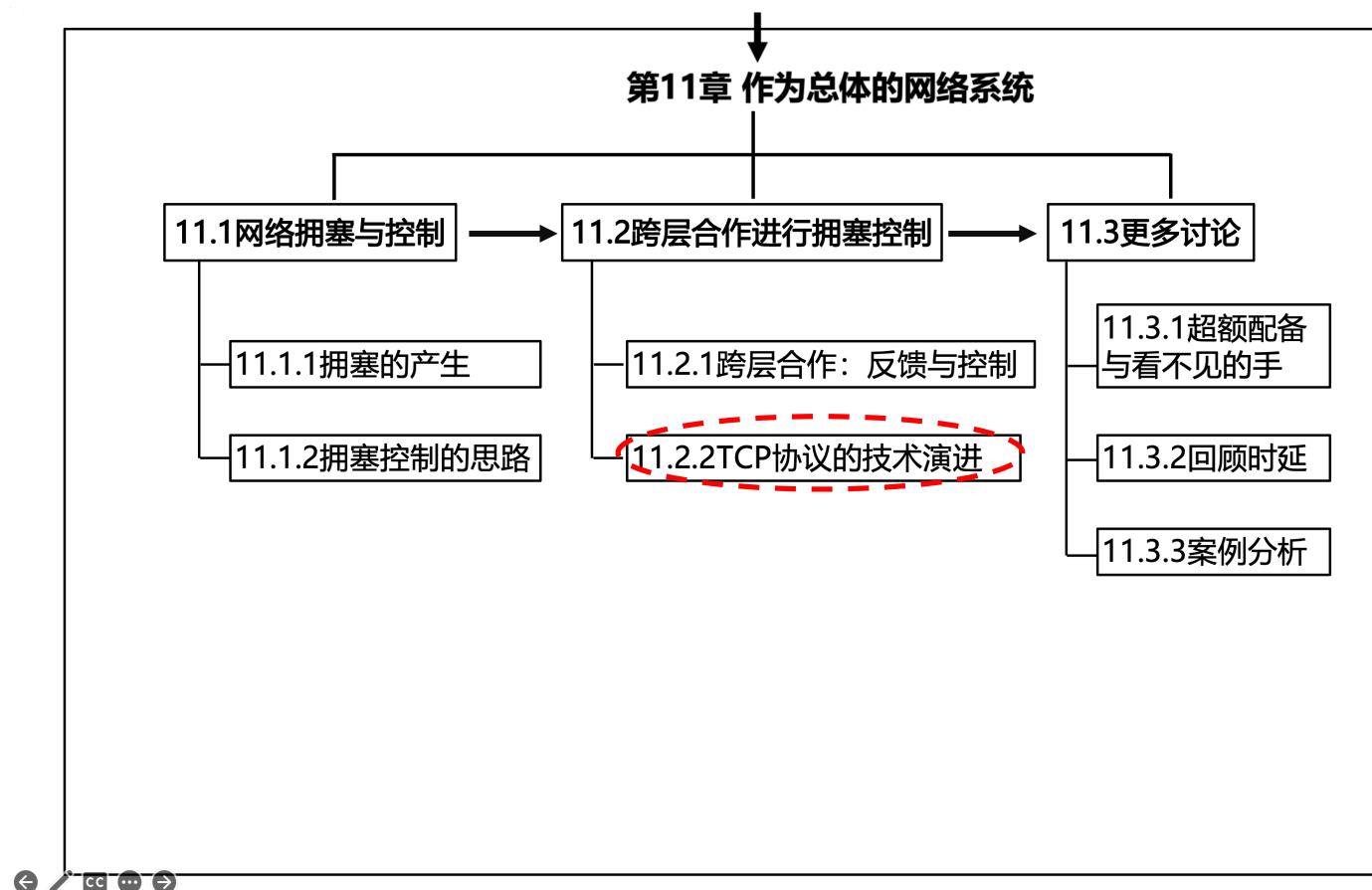
随着技术条件的演变，这种拥塞控制方法是否还合适？

分析

- 它的思想是基于丢包的
- 如果内存成本降低，配置增高，丢包有所改善的话，会怎样呢？
- 失效：“在泥泞中缓慢前进”
- J. Gettys, "Bufferbloat: Dark Buffers in the Internet," in *IEEE Internet Computing*, vol. 15, no. 3, pp. 96-96, May-June 2011

总结：网络资源管理的困难性

1. 行为不同的分布式资源，导致死锁/活锁，导致拥塞
2. 传输层独立于转发网络的特性，容易引发拥塞崩溃
3. 扩展容量的手段有限
 - 受限于物理条件，容量提升空间有限。
 - 更换路径，则需要知道整个网络的情况。
 - 文献中有若干尝试（[拓展阅读](#)），但是现实策略还是减少需求
4. 减少负载的手段难以实施



超额配备

方案：为所有链路配备最大流量的1.25-2倍的容量

- 可以应对的问题：（ ）
- 不能应对的问题：
 1. 偶发的小概率大流量事件，原因可能是（ ）
 2. 各链路实施的不同步性
 3. 边界节点的突发流量
 4. 用户自适应行为对效果的影响
 - 为什么免费高速公路的扩建对交通的缓解往往只有几年的效果？

超额配备

方案：为所有链路配备最大流量的1.25-2倍的容量

- 适用：大型网络的内部链路
- 实践：Internet骨干网普遍采用

因特网的现状

- 容量稀缺（capacity-scarce）转变为容量过剩（capacity-rich）
- 技术的变化，会如何改变我们的设计？

看不见的手

如果涉及到定价，就可以通过“看不见的手”进行调节

- 适用于某些收费的网络接入服务，可以在不同时间段制定不同的价格，从而借助经济杠杆来减轻拥塞

看不见的手

- 在自由市场中，买者有购买商品或离开的选择，卖者同样有提供商品或离开市场的选择。价格越高，就会有越多的卖家被这个获利机会所吸引，他们就会集体增加这种商品的供应量。与此同时，价格越高，越多的买家会犹豫不决，他们会共同减少对这种商品的需求。这两种效应创造了一种均衡，在这种均衡中，商品的供给与需求完全匹配。每个买者都对支付的价格满意，每个卖者都对收到的价格满意。当允许市场定价时，过剩和短缺就会被这种寻求均衡的机制系统地驱逐出去。
- 每个人都在努力增加社会的总收入。通常情况下，他们并不是出于促进公共利益的目的，也不清楚自己对公共利益的贡献。他们只追求个人利益，然而在这个过程中，就像在许多其他情况下一样，**他们被看不见的手引导着朝着一个非本意的目标前进。在追求个人利益的过程中，他们通常比真正想着促进社会利益时更有效地促进了社会利益。**
 - 亚当·斯密 (1723–1790). 《国富论》(1776) 卷4，第2篇

第11章 作为总体的网络系统

11.1 网络拥塞与控制

11.1.1 拥塞的产生

11.1.2 拥塞控制的思路

11.2 跨层合作进行拥塞控制

11.2.1 跨层合作：反馈与控制

11.2.2 TCP协议的技术演进

11.3 更多讨论

11.3.1 超额配备与看不见的手

11.3.2 回顾时延

11.3.3 案例分析

②回顾时延

从网络总体来回顾和分析时延的产生

- 物理性质所引起的时延：传播时延

- 青岛-乌鲁木齐：3000 km 10 毫秒
- 青岛-同步卫星-乌鲁木齐： $36000 \times 2 = 72000 \text{ km}$ 240 毫秒
- 临近计算机：3米 10 纳秒
- 不能消除，但是可以隐藏

- 服务容量不够引起的时延：排队时延

- 排队不仅会引发时延，还会引发错误
- 可以消除，通过增大容量或减少负载

②回顾时延

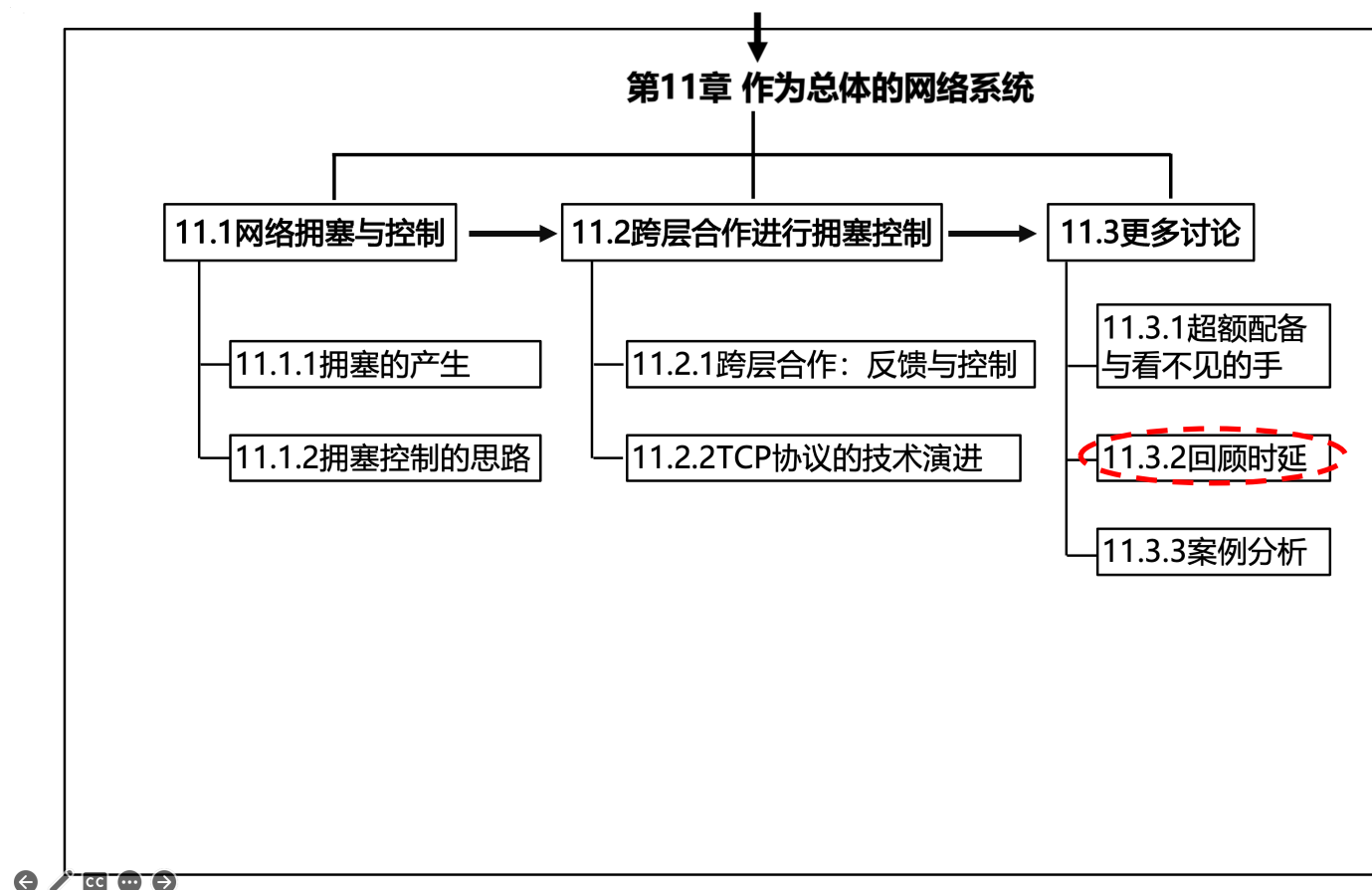
从网络总体来回顾和分析时延的产生

- 发送与转发数据包所需时间：处理时延

- 链路层：校验、位填充
- 网络层：校验、查转发表
- 端到端层：校验、压缩、消息重组、加密解密、鉴别.....
- 可以优化，但受限于算法，也可以隐藏

- 串行传输导致的时延：传输时延

- 链路层：帧的长度/信道带宽
- 网络层：选择的路由的带宽决定了一组传输时延
- 端到端层：在流量和拥塞控制的窗口机制下，瓶颈数据率、rtt、窗口大小决定了传输时延，控制乱序、抖动影响传输时延
- 可以优化，但不同优化机制适用不同的场景



总结与讨论

讲授的概念和技术是万能药吗？

- 取决于
 - 应用需求
 - 底层设置
- 当链路很少，路由转发变得简单，复用的需要更为强烈。
 - Eehernet?
- 广播链路的网络层协议举例：backplane bus
- 会话和表示什么时候需要？

全面考虑目标与因素、深入分析、深思熟虑的权衡是最重要的。

取舍和抉择是必须的。

案例1：NFS导致的拥塞崩溃

NFS文件系统（回顾“第4章 模块化”中的案例）

- Sun公司开发的无连接、无状态的网络文件系统
- 可以使用mount进行远程挂载
- 服务器端
 - 实现了幂等、无状态的at-least-once语义
 - 服务策略是FIFO（先来先服务）
- 客户端
 - 如果没有收到回应，会持续定时尝试

案例1：NFS导致的拥塞崩溃

出现的问题

1. 假如客户端请求超过服务能力，会发生排队
2. 排队持续增长，客户端请求会超时
3. 超时发生时，客户端会重发请求
4. 服务器不能区分重复的请求，会应答每个请求，导致服务资源浪费
5. 服务资源浪费导致更长的排队
6. 导致更多超时，最终发生拥塞崩溃

解决方案

- 超时发生后，客户端指数时间退避，并将超时定时器加倍

教训：

- 固定时间的计时器是问题的主要根源

案例2: Autonet广播风暴

DEC公司设计的实验局域网

- 拓扑: 树状网络
- 广播包的处理: 首先路由到root节点, 再由root发给各节点, 确保每个节点只收到1次广播包

问题现象

- 每次广播都会出现大量重复的广播包, 导致网络崩溃

案例2: Autonet广播风暴

问题分析

- 对软件进行分析, 没有发现问题
- 对硬件进行分析, 发现问题:
 - 链路使用了双绞线
 - 双绞线的特点: 无终端接头的双绞线, 会产生回声
 - 有人将网线从网卡拔下, 产生回声, 导致风暴

教训:

- 涌现效应往往是由表面不相关的、在系统不同层面的系统特性相互作用产生的。在这个案例中, 链路层产生了回声, 而网络层产生了广播风暴。

理解网络系统的视角：从不同的视角抽象互联网

链路层一般硬件实施(有自己的cpu和程序)，网络层大多软件实施，从这条分隔线来看。网络可以看成网卡组成的链路单元，一个单元是一个server，主机调用这些server，在此上再抽象其他层面

网络层一般操作系统实施，所有的操作系统组成一个巨大的分布式服务系统，应用程序作为client、申请服务。（raw socket）

端到端层是一条虚拟链路。应用程序作为client申请服务。（socket）

洋葱网在端到端层上再建立一层端到端虚拟链路。

如果我们用洋葱网搭建vpn，那么洋葱网的链路也从端到端变成点到点。

学习目标：抽象计算机系统，具象计算机抽象

理解网络系统的路线：自顶向下还是自底向上？

自顶向下：需求驱动

- 我要什么
- 解构
- 模块化、层次结构

自底向上：技术驱动

- 我有什么
- 架构
- 模块化、层次结构

底层的设计者要始终了解应用需求，才知道往何处研究

高层的设计者要始终了解技术发展，才知道找何种方案



重要术语中英文对照



拥塞控制: congestion control
提交负载 (控制后提交的负载): offered load
期望负载 (无控制时的提交负载): intended load
反直觉现象: counter-intuitive
控制点: control point
尾丢弃 (排队满了, 最后来的被丢弃): tail drop
自适应数据率: automatic rate adaptation
拥塞窗口: Congestion window
慢开始: Slow start
重复确认: duplicate acknowledgement
加法增加: Additive increase
乘法减少: Multiplicative decrease
加法增加、乘法减少: AIMD
Equilibrium: 平衡
超额配备: overprovisioning



本章相关的参考文献



- Vinton G. Cerf, Robert E. Kahn. "A Protocol for Packet Network Intercommunication". IEEE Transactions on Communications. 1974, 22 (5): 637–648.
- Cohen D. Flow control for real-time communication[J]. ACM SIGCOMM Computer Communication Review, 1980, 10(1-2): 41-47.
- Jacobson V. Congestion avoidance and control[J]. ACM SIGCOMM computer communication review, 1988, 18(4): 314-329.
- Kleinrock L. The latency/bandwidth tradeoff in gigabit networks[J]. IEEE communications magazine, 1992, 30(4): 36-40.
- Chiu, Dah-Ming, Raj Jain (1989). "Analysis of increase and decrease algorithms for congestion avoidance in computer networks". Computer Networks and ISDN Systems. 17: 1–14.
- Hardin G. Extensions of "the tragedy of the commons"[J]. Science, 1998, 280(5364): 682-683.

第11章 结束

