二——拥塞控制与资源分配

资源分配

TCP/IP协议栈是多路复用的，当用户竞争资源时，问题在于如何有效和公平地分配资源。

> 资源分配（resource allocation）是指一个过程，网络设备通过它来尽量满足应用对网络资源的竞争需求。
> 拥塞控制（congestion control）这个术语来描述网络节点为防止和相应过载状态所做的工作。

评价标准：

- 有效的资源分配
  - 主要度量因素：吞吐量、延迟
- 公平的资源分配

排队规则决定如何缓存等待的分组，确定了分组的等待传输时间，进而影响了资源的分配。FIFO（先进先出）也成为先来先服务排队，更确切地说它叫带队尾丢弃的FIFO（FIFO with tail drop）：它的调度规则（scheduling discipline）是FIFO，丢弃策略（drop policy）为队尾丢弃。FIFO无法按照分组所属的流分离他们，因此一些恶意的流就可能占用网络任意多的容量。公平排队（FQ）就是解决这一问题：它的核心思想是为路由器当前处理的每一个流维护一个独立的队列，路由器以轮转方式为这些队列服务。

> 所谓流（flow）是指在源和目的主机对之间发送的一系列分组，他们沿相同的路由经过网络。[2]

Linux下可用`tc qdisc`设置排队规则。

拥塞控制

拥塞避免（congestion avoidance）与拥塞控制（congestion control）是两种对待拥塞的策略。前者要求预测拥塞将在何时发生，在分组刚被丢弃前降低主机发送速率；后者需要制造丢失分组来发现可用带宽。但丢包不等于拥塞，这在无线网络中体现明显，他可能因为位错误产生高频率的丢包。

相关算法可分为链路算法（Linker Algorithm）与源算法（Source Algorithm）。前者在网络设备中执行，后者在主机与网络边缘设备中执行。

造成拥塞的的主要原因是网络流量通常是突发性的，那么可以强迫数据包以一定速度发送，将不平滑的数据包流转换成网络上平滑的数据包流。漏桶算法（The Leaky Bucket Algorithm）

便是这样的一个流量整形算法，令牌桶算法（The Token Bucket Algorithm）改进了漏桶算法，允许突发流量。

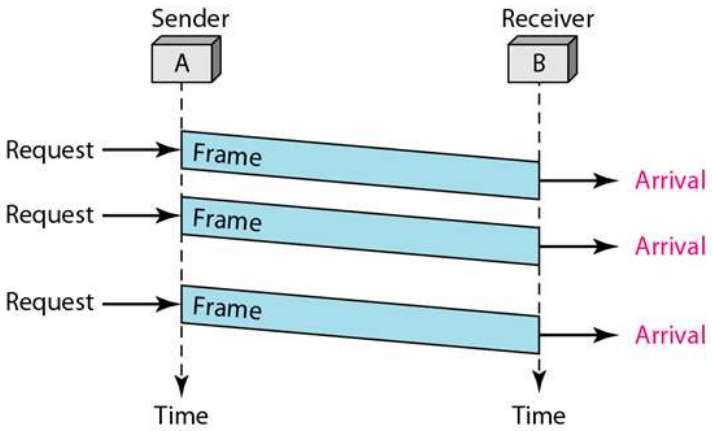
TCP拥塞控制

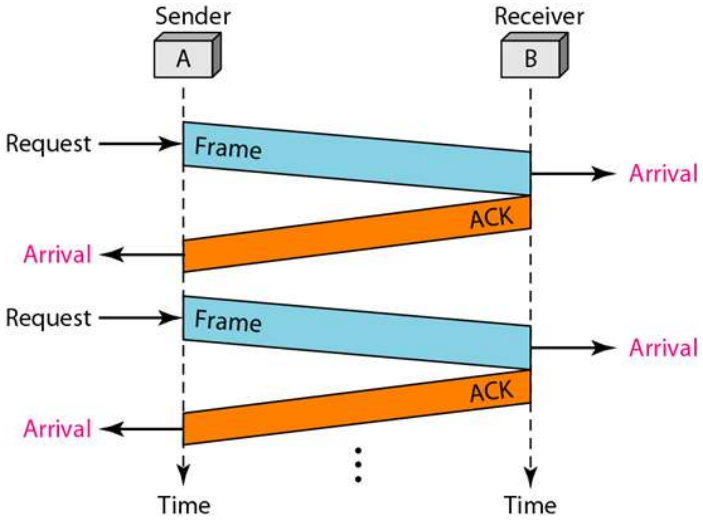
TCP中的拥塞控制是用于**端到端拥塞控制**的主要实例。实现TCP的拥塞控制需要两个基本的元素：

- Acknowledgments（ACK）
- Congestion Window

拥塞窗口（Congestion Window）并非滑动窗口（Sliding Window），即体现在TCP首部中的Window（又叫Advertised Window）。

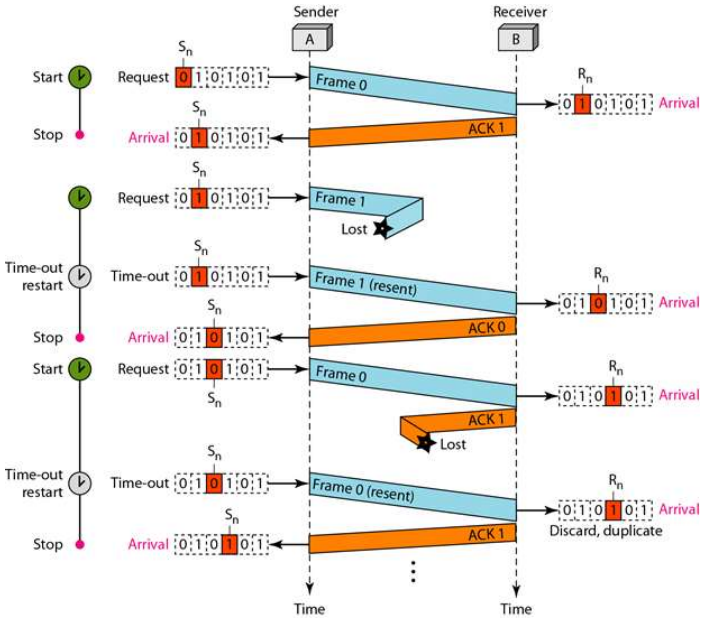
滑动窗口协议保证了数据包可靠且有序的到达，用于流量控制（flow control）和差错控制（error control），它不仅应用在TCP中，而且在数据链路层中。[1]流量控制是一系列用来限制发送方在等到确认之前发送数据流量的过程。差错控制基于自动重复请求，即重传数据。

> 在数据链路层中，滑动窗口有着以下的演进：
>
> 1. 在无噪声信道下，即一种不会丢失帧、复制帧或损坏帧的理想信道。最简单的协议（Simplest Protocol）不使用流量控制，也不使用差错控制，仅仅是发送一个帧序列而不用考虑接收方。
>
> 
>
>
> 【图：没有流量控制和差错控制的最简单协议】
>
> 2. 考虑流量控制，需要增加ACK，因此有了停止等待协议（Stop-and-Wait Protocol）：发送方发送一帧后要等待接收方的ACK帧的反馈确认，才能发送下一帧。



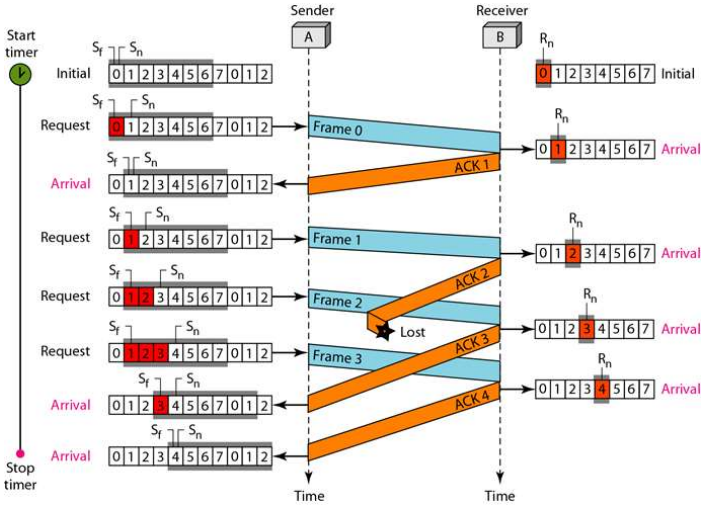
【图：停止等待协议】

3. 但在有噪声的信道下，需要考虑差错控制，设计了停止等待ARQ协议（Stop-and-Wait Automatic Repeat Request Protocol）：为每一帧编号，帧编号保证了有序性，确认帧是期望收到的下一帧编号，差错检测由重传定时器超时确定，通过重传上次保留下来的已发送帧副本实现。

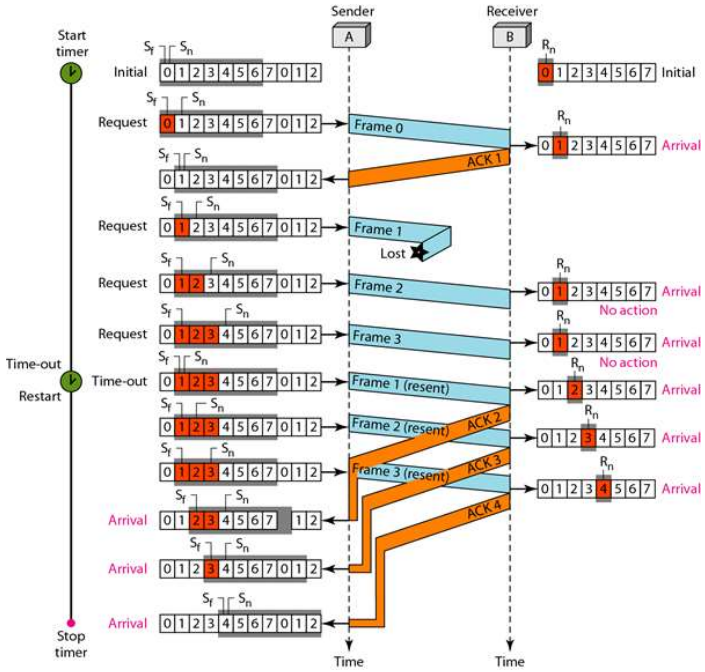


【图：停止等待ARQ协议】

4. 为了提高传输效率，在等待确认时传输多个帧，出现了滑动窗口，分为发送滑动窗口与接收滑动窗口，发展出后退N帧ARQ协议（Go-BACK-N Automatic Repeat Request Protocol）。停止等待ARQ是窗口大小为1的后退N帧ARQ的特殊情况。

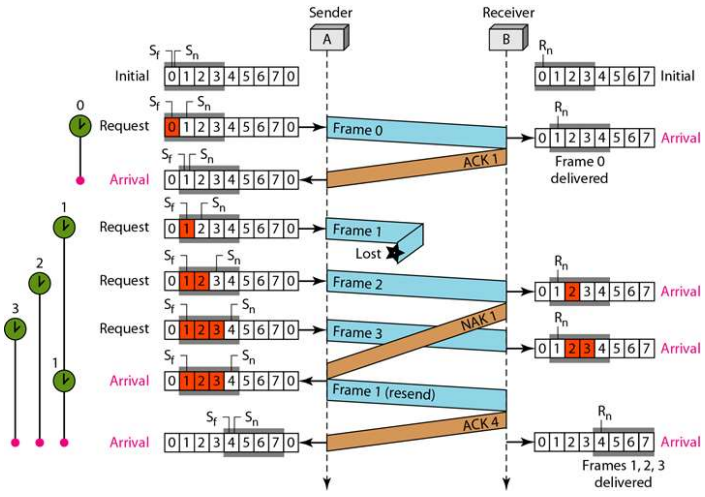


【图：后退N帧ARQ协议中ACK丢失时累计确认的情况】



【图：后退N帧ARQ协议中发送帧丢失情况】

5. 为了尽可能地减少重传的数据帧，有了选择重传ARQ协议（Selective Automatic Repeat Request Protocol），它告知了发送方丢失的帧。



【图：选择重传ARQ协议中帧丢失情况和NAK帧的情况】

TCP中的滑动窗口，记得大学老师把他称为信贷滑动窗口，与后退N帧协议中滑动窗口的区别在于窗口大小可变。TCP首部中的滑动窗口字段向发送方通告了自己接收窗口的大小，防止发送方发送的数据过多导致的缓冲区溢出。它的拥塞窗口，或者理解成发送窗口，限制给定时间内允许传送的数据量。

标准的TCP拥塞控制机制分为三个部分：

1. 加性增/乘性减（additive-increase/multiplicative-decrease，AIMD）
2. 慢启动（slow start）
3. 快速重传和快速恢复（fast retransmit，fast recovery）

TCP源根据它所获得的网络中存在的拥塞级别来**寻找拥塞窗口大小**：拥塞级别上升时减少拥塞窗口，拥塞级别下降时加大拥塞窗口，这两种机制通常称为AIMD。在bbr算法中，并非AIMD，而是采用带宽延迟积评估拥塞窗口的大小[6]。

在TCP**确定网络拥塞**上，基于以下观察：分组不能被传送和导致超时的主要原因在于拥塞造成分组被丢弃，因此大多数拥塞控制算法认为超时是拥塞发生的标志，并据此降低正在传输的速率；也可观察到：若源正在发送的额外数据量太多，则会引起长时间的延迟，并可能导致拥塞，因此一些算法，比如Vegas，采用拥塞避免的策略，努力将正在发送的额外数据量维持在正常的标准上，根据诸如RTT的变化来确定网络拥塞。

当源的操作从开始启动时，使用加性增的机制**增长太慢**，对此TCP提供了慢启动机制：初次增加，拥塞窗口呈指数增长，乘性减后利用拥塞阈值（Congestion Threshold）来记录此时的拥塞窗口大小，之后的慢启动在拥塞阈值之前指数增长，之后每次增加一个分组。一些拥塞算法，比如Cubic，采用二分搜索的方式来决定拥塞窗口的增长尺度。

为了**改进超时机制与拥塞窗口回退策略**，提出快速重传与快速恢复，它是一种细粒度的方法。从概率上讲，接收两次重复确认（duplicate ACK）意味着40%的丢包可能性，因此引入了快速重传机制。丢包并不一定发生拥塞，因此当快速重传机制发出拥塞信号时，可能利用还在管道中的ACK去同步分组的发送，这种机制称为快速恢复。

同选择确认ARQ协议一样，TCP也有选择确认（Selective Acknowledgments，SACK）机制，在三次握手的时候在TCP选项中协商使用。选择确认可以令发送方选择性地重传丢失的数据，但可能会导致拥塞的加剧，因此引入FACK（Forward Acknowledgment）算法解决这个问题。

历史：

Tahoe： 慢启动、拥塞避免、快速重传三算法。paper： congest avoid.pdf。

Reno： RFC 5681 加上快速恢复。

NewReno：引入了部分确认和全部确认的概念。

SACK：规范了TCP中带选择的确认消息.

Vegas：采用带宽估计,缩短了慢启动阶段的时间，源拥塞避免。

Reference

[1]: https://en.wikipedia.org/wiki/Sliding_window_protocol

[2]: 数据通信与网络
[3]: https://allen-kevin.github.io/2017/12/21/TCP%E6%8B%A5%E5%A1%9E%E6%8E%A7%E5%88%B6%E4%B9%8BCUBIC/
[4]: https://www.zhihu.com/question/21789252
[5]: https://wiki.aalto.fi/download/attachments/69901948/TCP-CongestionControlFinal.pdf
[6]: https://www.zhihu.com/question/53559433
[7]: https://netbeez.net/blog/how-to-use-the-linux-traffic-control/

---

by river[river@vvl.me]
2019.0215: initialization