

计算机网络笔记

上官凝

2020 年 11 月 7 日

目录

1 计算机网络概述	3
1.1 什么是 internet	3
1.2 网络边缘	3
1.3 网络核心	3
1.4 分组交换中的时延、丢包和吞吐量	4
1.5 协议层次和服务类型	4
1.5.1 服务和服务访问点	4
1.5.2 服务和协议	4
2 应用层	6
2.1 应用层协议原理	6
2.1.1 网络应用架构	6
2.1.2 进程通信	6
2.1.3 因特网提供的运输服务	6
2.2 Web 和 HTTP	7
2.2.1 HTTP 概况	7
2.2.2 非持续链接和持续链接	7
2.2.3 HTTP 报文格式	7
2.2.4 用户-服务器状态: cookies	7
2.3 FTP	7
2.4 Email	7
2.5 DNS	8
2.6 P2P 应用	8
2.7 CDN	8
2.8 TCP socket 编程	8
2.9 UDP socket 编程	8
3 运输层	9
3.1 概述和运输层服务	9
3.2 多路复用与解复用	9
3.3 无连接传输: UDP	11

3.4 可靠数据传输的原理	11
3.4.1 RDT 1.0: 经完全可靠信道的可靠数据传输	12
3.4.2 RDT 2.0: 经具有比特差错信道的可靠数据传输	12
3.4.3 RDT 2.1: 发送方处理出错的 ACK/NAK	14
3.4.4 RDT 2.2: 不使用 NAK 的协议	14
3.4.5 RDT 3.0: 经具有比特差错和分组丢失的信道的可靠信息传输	14
3.4.6 流水线可靠数据传输协议	14
3.4.7 回退 N 步	15
3.4.8 选择重传	15
3.5 面向连接的传输: TCP	16
3.5.1 段结构	17
3.5.2 可靠数据传输	18
3.5.3 流量控制	18
3.5.4 连接管理	18
3.6 拥塞控制原理	18
3.7 TCP 拥塞控制	18

第 1 章 计算机网络概述

第 1.1 节 什么是 internet

一个网，或者图的角度来理解，结点包括主机及其上的应用程序，和路由器、交换机等网络交换设备；边包括通信链路，分为接入网链路和主干链路。还有协议，协议定义了在两个或多个通信实体之间交换的报文格式和次序，以及在报文传输和/或接收或其他事件方面所采取的动作

第 1.2 节 网络边缘

端系统（主机）、C-S 模式、P2P 模式。网络设施的 TCP 和 UDP 服务

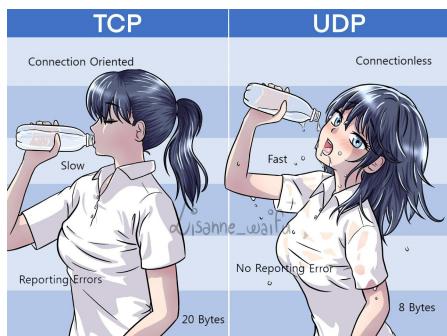


图 1.1: TCP 与 UDP

2.1.4 因特网提供的传输服务

- | | |
|---|---|
| TCP service: <ul style="list-style-type: none"><input type="checkbox"/> 面向连接: 客户进程和服务器进程需要建立连接<input type="checkbox"/> 发送进程和接收进程之间可靠传输<input type="checkbox"/> 流量控制: 发送进程不会“压垮”接收进程<input type="checkbox"/> 拥塞控制: 网络超载时抑制发送进程<input type="checkbox"/> 不提供: 及时性，最低带宽保证 | UDP service: <ul style="list-style-type: none"><input type="checkbox"/> 不可靠传输<input type="checkbox"/> 不提供: 连接建立，可靠传输，流量控制，拥塞控制，及时性，最低带宽保证 |
|---|---|

图 1.2: TCP 与 UDP

第 1.3 节 网络核心

电路交换和分组交换

电路交换：为线路预留端到端资源，链路带宽（尤其是瓶颈带宽）决定了通信能力，专用资源不共享，一旦建立起来就可以保证性能，需要建立连接。网络资源（如带宽）被划分成片，可以频分 (FDM)、时分 (TDM)、波分 (WDM)

分组交换：存储-转发（分组每次移动是一跳），在转发之前，结点必须收到整个分组

分组交换可以提升网络的容量：假设线路是 1Mbps，每个用户在活跃的时候用掉 100kbps，有 10% 的时间是活跃的。若使用分组交换，则 ≥ 10 个用户活跃的概率，由二项分布可得为

$$1 - \sum_{n=0}^9 \binom{35}{n} p^n (1-p)^{35-n}$$

第 1.4 节 分组交换中的时延、丢包和吞吐量

分组延迟的来源有四：结点处理延迟、排队延迟、传输延迟、传播延迟。传输延迟是指将分组发送到链路上的时间。

$$d_{node} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

第 1.5 节 协议层次和服务类型

1.5.1 服务和服务访问点

1. 服务 (Service): 低层实体向上层实体提供它们之间的通信的能力
 - (a) 服务用户 (service user)
 - (b) 服务提供者 (service provider)
2. 原语 (primitive): 上层使用下层服务的形式，高层使用低层提供的服务，以及低层向高层提供服务都是通过服务访问原语来进行交互的
3. 服务访问点 SAP (Services Access Point): 上层使用下层提供的服务通过层间的接口
 - (a) 例子: 邮箱
 - (b) 地址 (address): 下层的一个实体支撑着上层的多个实体，SAP 有标志不同上层实体的作用
 - (c) 可以有不同的实现，队列
 - (d) 例子: 传输层的 SAP: 端口 (port)

1.5.2 服务和协议

1. 服务与协议的区别
 - (a) 服务 (Service): 低层实体向上层实体提供它们之间的通信的能力，是通过原语 (primitive) 来操作的，垂直
 - (b) 协议 (protocol): 对等层实体 (peer entity) 之间在相互通信的过程中，需要遵循的规则的集合，水平
2. 服务与协议的联系

- (a) 本层协议的实现要靠下层提供的服务来实现
- (b) 本层实体通过协议为上层提供更高级的服务

第 2 章 应用层

第 2.1 节 应用层协议原理

2.1.1 网络应用架构

网络核心中没有应用层功能，网络应用只在端系统上存在，快速网络应用开发和部署。应用层可能的应用架构：客户-服务器模式（C/S），或者对等模式（P2P），或者混合体

客户-服务器模式 服务器：一直运行，并有固定的 IP 和熟知的端口号；客户机：与互联网有间歇性的连接，可能是动态 IP 地址，不直接与其它客户端通信

对等体体系结构 每一个节点既是客户端又是服务器；自扩展性——新 peer 节点带来新的服务能力，当然也带来新的服务请求；参与的主机间歇性连接且可以改变 IP 地址；难以管理

2.1.2 进程通信

不同主机上的进程通过交换报文进行通信。对每对通信进程，我们通常将这两个进程之一标识为客户，另一个标识为服务器。其各自的定义如下：

在一对进程之间的通信会话场景中，发起通信（即在该回话开始时发起与其他进程的联系）的进程被标识为客户，在回话开始时等待联系的是服务器

进程编址需要 IP 地址和端口号

2.1.3 因特网提供的运输服务

因特网（更一般的是 TCP/IP 网络）为应用程序提供了两个运输协议，即 UDP 和 TCP。

TCP Service TCP 服务模型包括面向连接服务和可靠数据传输服务。TCP 链接是全双工的，即连接双方的进程可以在此链接上同时进行报文收发，当应用程序结束发送时，需要拆除该链接。TCP 还有拥塞控制机制

第 2.2 节 Web 和 HTTP

2.2.1 HTTP 概况

web 页面是由对象组成的。对象可以是 HTML 文件、JPEG 图像、Java 小程序、声音剪辑文件等。Web 页含有一个基本的 HTML 文件，该基本 HTML 文件又包含若干对象的引用（链接）。通过 URL 对每个对象进行引用。访问协议，用户名，口令字，端口等。

HTTP 是无状态的，即服务器不保存有关客户请求的任何有关信息

2.2.2 非持续链接和持续链接

非持久连接在一个 TCP 连接上最多传输一个对象，持续连接可以发送多个对象。又由于持续连接可以采用流水，效率会更高

关于响应时间模型：往返时间 RTT：一个小的分组从客户端到服务器，在回到客户端的时间（传输时间忽略）。响应时间是 $2RTT + \text{传输时间}$ （握手 + 请求和响应）

2.2.3 HTTP 报文格式

2.2.4 用户-服务器状态：cookies

cookies 是在用户端系统中维护的，由用户的浏览器管理

第 2.3 节 FTP

FTP 使用了两个端口

FTP：控制连接与数据连接分开

- FTP 客户端与 FTP 服务器通过端口 21 联系，并使用 TCP 为传输协议
 - 客户端通过控制连接获得身份确认
 - 客户端通过控制连接发送命令浏览远程目录
 - 收到一个文件传输命令时，服务器打开一个到客户端的数据连接
 - 一个文件传输完成后，服务器关闭连接
-
- The diagram shows a laptop icon labeled "FTP client" connected to a server icon labeled "FTP server" by a double-headed arrow labeled "TCP control connection, server port 21". Below this, another double-headed arrow labeled "TCP data connection, server port 20" is shown between the client and server.

FTP：控制连接与数据连接分开

- FTP 客户端与 FTP 服务器通过端口 21 联系，并使用 TCP 为传输协议
 - 客户端通过控制连接获得身份确认
 - 客户端通过控制连接发送命令浏览远程目录
 - 收到一个文件传输命令时，服务器打开一个到客户端的数据连接
 - 一个文件传输完成后，服务器关闭连接
-
- The diagram shows the client and server from the previous diagram. The client has sent a command to the server, which has responded by opening a data connection back to the client on port 21. A double-headed arrow labeled "TCP data connection, client port 20" is shown between them.

第 2.4 节 Email

SMTP 和 HTTP 挺像的，总结如下：

其中拉协议指，在方便的时候，某些人在 web 服务器上装载信息，用户使用 HTTP 从该服务器拉取这些信息。特别是 TCP 连接是由想接收文件的机器发起的。而推协议，指发送邮件服务器把文件推向接收邮件服务器。特别是 TCP 连接是由要发送文件的机器发起的。

SMTP: 总结

- SMTP 使用持久连接
- SMTP 要求报文（首部和主体）为 7 位 ASCII 编码
- SMTP 服务器使用 CRLF、CRLFCRLF 决定报文的尾部

HTTP 比较：

- HTTP：拉（pull）
- SMTP：推（push）
- 二者都是 ASCII 形式的命令 / 响应交互、状态码
- HTTP：每个对象封装在各自的响应报文中
- SMTP：多个对象包含在一个报文中

值得注意的是，SMTP 要求每个报文（包括他们的体）采用 7bit ASCII 码格式。如果包含了非 7bit ASCII 字符（如具有重音的法文字符）或二进制数据（如图形文件），则必须按照 7bit ASCII 进行编码。

第 2.5 节 DNS

第 2.6 节 P2P 应用

第 2.7 节 CDN

第 2.8 节 TCP socket 编程

第 2.9 节 UDP socket 编程

第3章 运输层

第3.1节 概述和运输层服务

在应用程序看来，运输层（PPT 为传输层）为运行在不同主机上的应用进程提供了进程间的逻辑通信。从应用程序的位置来看，通过逻辑通信，运行不同进程的主机好像直接相连一样；实际上，这些主机也许位于地球的两侧通过很多路由器和多种不同类型的链路相连。同应用层一样，运输层也是只有运行在端系统上的。在发送方，将应用层的报文（拆分）并封装为报文段；在接收方做逆处理，从收到的报文段中取出载荷，重组为报文。

网络层服务是主机间的逻辑通信，而运输层是进程间的逻辑通信，它依赖于网络层的服务（继承带宽、延迟的限制）并对网络层的服务进行增强（解决数据丢失、顺序混乱，并加密）。有些服务是可以加强的：不可靠 → 可靠、安全。但有些服务是不可以被加强的：带宽，延迟

类比：两个家庭的通信（Ann 家的 12 个小孩给另 Bill 家的 12 个小孩发信）

1. 主机：家庭
2. 进程：小孩
3. 应用层报文：信封中的信件（可以类比信封为包装的报文段附加的部分）
4. 传输协议：Ann 和 Bill（为家庭小孩提供复用解复用服务）
5. 网络层协议：邮政服务（家庭·家庭的邮包传输服务）

在本书中，我们将 TCP 和 UDP 的分组统称为报文段，而将数据报名称留给网络层分组。

TCP 和 UDP 最基本的责任是，将两个端系统间 IP 的交付服务扩展为运行在端系统上的两个进程之间的交付服务。将主机间交付扩展到进程间交付被称为运输层的多路复用与多路分解（transport-layer multiplexing and demultiplexing）。TCP 力求为每一个通过一条拥塞网络链路的连接平等地共享网络链路带宽。

第3.2节 多路复用与解复用

1. 在发送方主机多路复用：从多个套接字接收来自多个进程的报文，根据套接字对应的 IP 地址和端口号等信息对报文段用头部加以封装（该头部信息用于以后的解复用）
2. 在接收方主机多路解复用：根据报文段的头部信息中的 IP 地址和端口号将接收到的报文段发给正确的套接字（和对应的应用进程）

为了将报文交给正确的套接字

1. 主机中每个套接字应分配一个唯一的标识
2. 报文段中有特殊字段指示要交付的套接字
3. 发送方传输层需在报文段中包含目的套接字标识(多路复用)
4. 接收方传输层需将报文段中的目的套接字标识与本地套接字标识进行匹配, 将报文段交付到正确的套接字(多路分解)

回忆一下 2.7 节, 一个进程(作为网络应用的一部分)有一个或多个套接字, 它们相当于在网络和进程之间传递数据的门户 端口号是 socket 标识的重要组成部分, 是一个 16 位的二进制数,

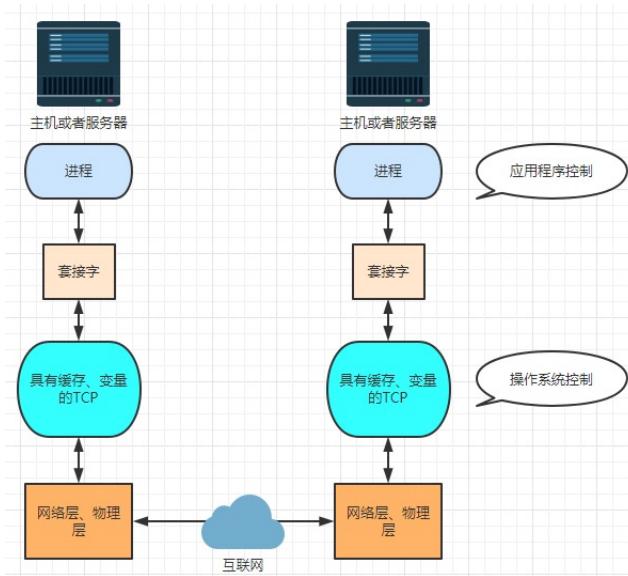
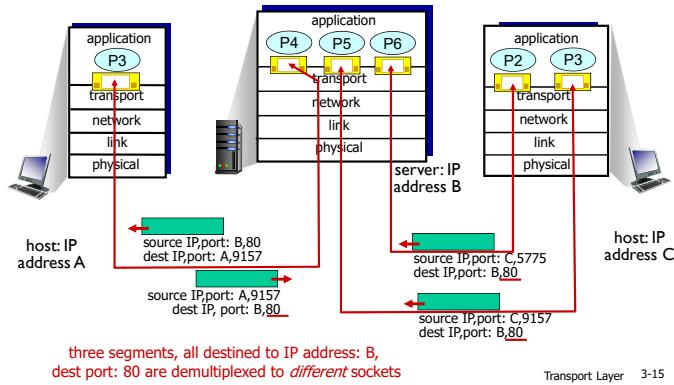


图 3.1: 应用进程、套接字、运输层

其中 0~1023 作为保留端口号给公共域协议使用, 称众所周知的端口号。一般实现公共域协议的服务器会绑定到这个区域内。在主机上的每一个套接字都能够分配到一个端口号, 当接收方传输层接收到一个 UDP 报文时, 检查其中的目标端口号, 并将这个报文交付到具有该端口号的套接字。

值得注意的是, 在多路解复用的过程中, UDP 的 socket 选择标识为报文段中的二元组(目的 IP, 目标端口号), 而 TCP 用的标识是(源 IP, 源 PORT, 目标 IP, 目标 PORT)的四元组。所以对于 UDP, 具备相同目标 IP 地址和目标端口号, 即使是源 IP 地址或/且源端口号不同的 IP 数据报, 也会被传到相同的目标 UDP 套接字上。而对于 TCP, 服务器能够在一个 TCP 端口上同时支持多个 TCP 套接字: 每个套接字由其四元组标识(有不同的源 IP 和源 PORT)。比如 Web 服务器对每个连接客户端有不同的套接字(非持久对每个请求有不同的套接字)。在上图的实际实现中, 有一个初始的 socket, 每接收到一个对应到本 PORT 的连接请求就“fork”出来一个新的 socket 来相应

面向连接的解复用: 例子



第 3.3 节 无连接传输: UDP

UDP 即用户数据报协议，其报文结构为源端口号、目的端口号、长度、检验和（奇偶校验）应用数据（报文）。对于 UDP 检验和的确定，其规则如下：UDP 校验和就是二进制反码求和（先求和然后再求反码），但在求和过程中假如首位溢出需要进位，需要回卷，即把前面多出去的 1 加到最后。比如下面这个例子：

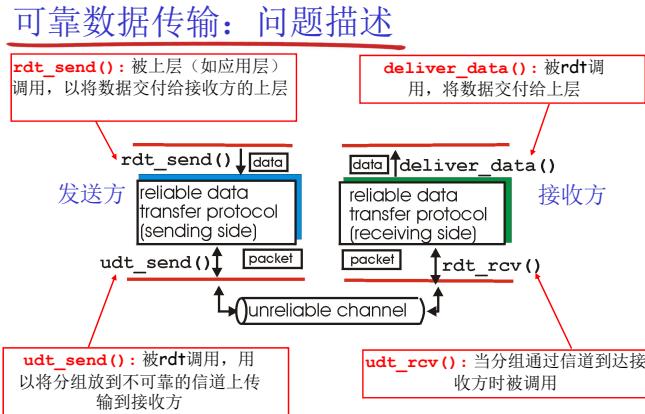
两组数据分别为 1001 和 1111，则求和时，由于首位溢出需要回卷，则为：

$$\begin{array}{r}
 1001 \\
 1111 \\
 \hline
 & 1 \\
 \hline
 1001
 \end{array}$$

取反码得到校验和为 0110。在接收端进行校验时，将校验范围与校验和相加，若为 0xFFFF 则通过校验

第 3.4 节 可靠数据传输的原理

可靠数据传输 (rdt, reliable data transfer) 在应用层、传输层、数据链路层都很重要。可靠数据传输命题大致如下图所示：



Transport Layer 3-26

表 3-1 可靠数据传输机制及其用途的总结

机制	用途和说明
检验和	用于检测在一个传输分组中的比特错误
定时器	用于超时/重传一个分组，可能因为该分组（或其 ACK）在信道中丢失了。由于当一个分组延时但未丢失（过早超时），或当一个分组已被接收方收到但从接收方到发送方的 ACK 丢失时，可能产生超时事件，所以接收方可能会收到一个分组的多个冗余副本
序号	用于从发送方流向接收方的数据分组按顺序编号。所接收分组的序号间的空隙可使接收方检测出丢失的分组。具有相同序号的分组可使接收方检测出一个分组的冗余副本
确认	接收方用于告诉发送方一个分组或一组分组已被正确地接收到。确认报文通常携带着被确认的分组或多个分组的序号。确认可以是逐个的或累积的，这取决于协议
否定确认	接收方用于告诉发送方某个分组未被正确地接收。否定确认报文通常携带着未被正确接收的分组的序号
窗口、流水线	发送方也许被限制仅发送那些序号落在一个指定范围内的分组。通过允许一次发送多个分组但未被确认，发送方的利用率可在停等操作模式的基础上得到增加。我们很快将会看到，窗口长度可根据接收方接收和缓存报文的能力、网络中的拥塞程度或两者情况来进行设置

3.4.1 RDT 1.0：经完全可靠信道的可靠数据传输

这是最简单的情形。注意到下列问题是重要的，发送方和接收方有各自的状态。¹ 在这个简单的协议中，一个单元数据和一个分组没有区别；因为信道完全可靠，接收端不需要反馈信息；由于假定了接收速率和发送速率一样，也不需要限流。

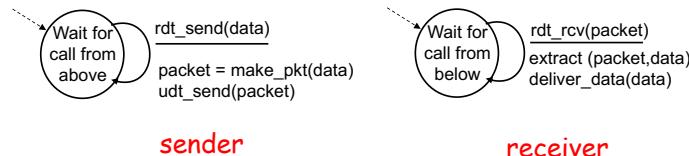
3.4.2 RDT 2.0：经具有比特差错信道的可靠数据传输

假定顺序不被打乱，但是有些比特可能受损（翻转）。处理此类模型的基本思想是“基于肯定确认和否定确认的重传机制的可靠数据传输协议”，称为自动重传请求协议 (Automatic Repeat reQuest, ARQ)。ARQ 使用了以下 4 种机制（书上没写第一个）：

¹ 本书使用的 FSM 规范：引起变迁的事件先是在表示变迁的横线上方，事件发生时所采取的动作显示在横线下方，如果事件/动作为空，则使用符号 \wedge ，以分别明确地表达缺少动作或事件。初始状态用虚线表示。

Rdt1.0: 可靠信道上的可靠传输

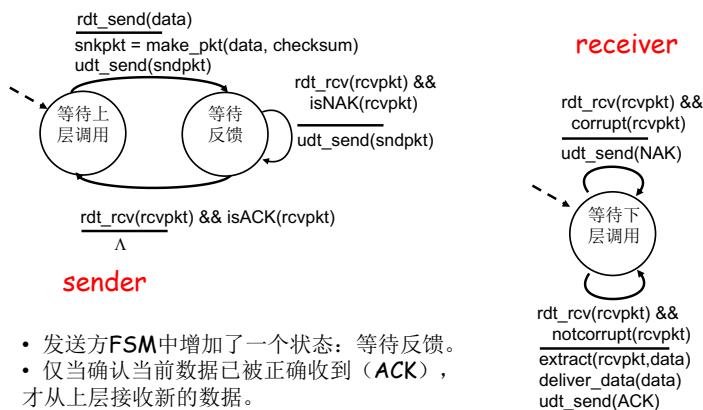
- 下层信道是完全可靠的（理想情况）
 - 没有比特错误
 - 没有分组丢失
- 发送方和接收方使用不同的FSM:
 - 发送方：从上层接收数据，封装成分组送入下层信道
 - 接收方：从下层信道接收分组，取出数据交给上层



Transport Layer 3-31

1. 发送方差错控制编码、缓存
2. 差错检测：使用校验码
3. 接收方反馈：接收方向发送方回送控制报文（“肯定确认”ACK 和“否定确认”NAK）
4. 重传：收到有差错的需要重传

rdt2.0: FSM specification



Transport Layer 3-33

注意下列事实很重要：当发送方处于等待 ACK 或 NAK 的状态时，它不能从上层获得更多的数据。因此 rdt2.0 这样的协议被称为停等 (stop-and-wait) 协议

3.4.3 RDT 2.1：发送方处理出错的 ACK/NAK

rdt2.0 有一个 fatal flaw，即由于信道的不可靠性，无法保证在反馈信号回送给发送方时，反馈信息分组可以无误到达。所以在除去增加纠错比特位使得接收方可以直接恢复原有信息这种办法以外，还可以采取发送方冗余重传的方式。收到损坏的 ACK/NAK 分组时，直接重传即可。但是这种方案可能会在接收方造成分组冗余，于是可以在发送分组上加一个序号（标志位），表示是初传还是重传

3.4.4 RDT 2.2：不使用 NAK 的协议

1. 接收方
 - (a) 对每一个正确接收的分组发送 ACK
 - (b) ACK 中显式携带所确认分组的序号
 - (c) 若收到出错的分组、或不是期待接收的分组，重发对前一个正确接收分组的 ACK
2. 发送方：若 ACK 的序号不是所期待的（表明当前分组未被确认），重发当前分组
3. 为后面的一次发送多个数据单位做一个准备
 - (a) 一次能够发送多个
 - (b) 每一个的应答都有:ACK, NACK; 麻烦
 - (c) 使用对前一个数据单位的 ACK，代替本数据单位的 nak
 - (d) 确认信息减少一半，协议处理简单

3.4.5 RDT 3.0：经具有比特差错和分组丢失的信道的可靠信息传输

使用一个倒计时装置，发送方等待 ACK 一个合理的时间（链路层的 timeout 时间是确定的，传输层 timeout 时间是适应式的），到时还没有收到 ACK 就重传。问题是如果仅仅是延迟了，可能会导致数据冗余。用序列号可以解决，但是接收方在发出 ACK 时必须指明接收的序列号。因为分组序号在 0 和 1 之间交替，因此 rdt3.0 有时被称为比特交替协议

需要注意的是，尽管 rdt3.0 是一个正确的协议，其停等协议的属性导致了它的性能不佳

3.4.6 流水线可靠数据传输协议

参考多周期 CPU 和流水线 CPU 的构造，想想为什么会有流水（并行）和如何抛出精确异常（回退 N 步和选择重传）。为了选择重传，接收方需要设置缓冲区缓存失序的包。

流水线技术对可靠数据传输可以带来如下影响：

1. 必须增加序号范围，因为每一个输送的分组（不计算重传的）必须有一个唯一的序号，而且也许由多个在输送中的未确认报文

2. 协议的发送方和接收方需缓存多个分组。发送方需那些已发送但没有确认的分组（可能重传），接收方需要已经正确接受的分组（可能乱序或者有中间的分组丢失）
3. 所需序号范围和对缓冲区的要求取决于数据传输协议如何处理丢失、损坏以及延迟过大的分组。

传输的窗口包括可以发送但还没有发出去的分组，和已经发出去但还没有 ACK 的分组，也有可能包括已经 ACK 的分组。他只是要求已发出去但没有被确认的包的数量最多为 N，最坏情况下，窗口中的包都是发出去却未收到 ACK 的。

处理异常主要有两种方法：回退 N 步（GBN）和选择重传（SR）

3.4.7 回退 N 步

允许发送方发送多个分组而不需要等待确认。GBN 的基本思想是，将分组按照一个数组进行放置，一个滑动窗口作为分组的可视范围，那些已被发送但还没有确认的，以及（由于收到 ACK 而）做好准备发送的分组的许可序号范围。

从另一个角度来看，许可序号是有限的，当一个 ACK 回到发送方时，就将这个序号传递给下一个没有被分配序号的分组，让他称为“可用，还未发送”状态。这类似于一种流水的折返跑接力赛，假设共有 N 个接力棒（窗口长度），拿到接力棒的选手进入准备状态（窗口内），这些选手每经过一定的时间就出发，到达终点就返回（ACK），返回之后将接力棒交给正好在窗口之后的分组。

分组序号承载在分组首部的一个固定长度的字段中，TCP 有一个 32bit 的序号字段，不过它是按照字节流进行计数的

GBN 的工作原理简单来说，就是

1. 哪里跌倒从哪里站起来：一旦某一个分组传输失败，那后面的都需要重新传输
2. 最高 ACK：接收方仅对正确收到的、序号连续的一系列分组中的最高序号进行确认
3. 失序复读：若收到失序的分组，丢弃（不在接收端缓存），并重发前一次（或者再之前）的 ack 分组（已正确收到、序号连续的一系列分组中的最高序号）
4. 累积确认：若 ACK 包含序号 q，表明“序号至 q 的分组均正确收到”
5. 一次性滑动：如果收到序号 q 的 ACK（即使没有收到之前的），整体滑动发送窗口，使基序号 = q+1
6. 超时重传：发送方只对基序号分组使用一个定时器，发送方重传发送窗口中从基序号开始的所有分组

计时：GBN 在应对超时采用的是维护一个计时器，记录最早的已发送的但还未确认的分组

3.4.8 选择重传

SR 协议通过让发送方仅重传那些它怀疑在接收方出错（丢失或受损）的分组而避免了不必要的重传。SR 协议与 GBN 有一些不同：因为 SR 的每一个分组都是独立的

GBN的发送方

- 收到上层的发送请求:
 - 若发送窗口满: 拒绝请求
 - 若发送窗口不满: 构造分组, 发送
 - 若原来发送窗口为空: 对基序号启动一个定时器
- 收到正确的ACK:
 - 更新基序号 (滑动窗口)
 - 若发送窗口空: 终止定时器
 - 若发送窗口不空: 对基序号启动一个定时器
- 收到出错的ACK:
 - 不做处理
- 定时器超时:
 - 启动定时器, 重发从基序号开始的所有分组

Transport Layer 3-56

1. 计时器: SR 的每一个分组都要有自己的一个计时器, 因为超时发生后只能发送一个分组
2. 窗口移动: 如果收到 ACK 是 send_base (窗口的第一个), 窗口基序号移动到具有最小序号的未确认分组处
3. 发送新的分组: (这两个都是) 发送在窗口内且还没发出去的分组
4. 收到 ACK: 标记这个分组为已接受
5. 重复 ACK: 接收方在接收到窗口头之前的分组时, 还是需要发送 ACK, 因为这个分组有可能时因为它的 ACK 没有成功到达发送方或者发送方超时, 导致发送方重传, 所以还是需要通知发送方
6. 窗口大小: 窗口长度必须小于等于序号空间的一半

第 3.5 节 面向连接的传输: TCP

TCP 即传输控制协议, 是因特网运输层的面向连接的可靠的运输协议。为了提供可靠数据传输, TCP 依赖于前面提到的许多基本原理, 包括差错检测、重传、累积确认、定时器以及用于序号和确认号的首部字段。

总的来讲, TCP 是一种较为“繁琐”的传输协议, 双方需要相互握手, 并采取很多措施, 用时间来换取传输的正确性。TCP 传输有如下特点:

1. 点到点通信: 一个发送者, 一个接收者
2. 全双工: 可以同时双向传输数据
3. 面向连接: 通信前双方先握手 (交换控制报文), 建立数据传输所需的状态 (缓存、变量等)
4. 可靠、有序的字节流: 不保留报文 (应用程序的输出) 边界

5. 流水式发送报文段: 发送窗口由拥塞控制和流量控制机制设置

6. 流量控制: 发送方不会令接收方缓存溢出

TCP 可从缓存中取出并放入报文段中的数据量受限于最大报文段长度 (MSS)，它通常根据最初确定的由本地发送主机发送的最大链路层帧长度 (即所谓的最大传输单元 (MTU)) 来设置。注意 MSS 是指在报文段里**应用层数据**的最大长度，而不是包括首部的 TCP 报文段最大长度。建立连接时，每个主机可声明自己能够接受的 MSS，缺省为 536 字节

3.5.1 段结构

TCP 报文段由首部字段和一个数据字段组成。总的来看，TCP 报文段为如下格式：

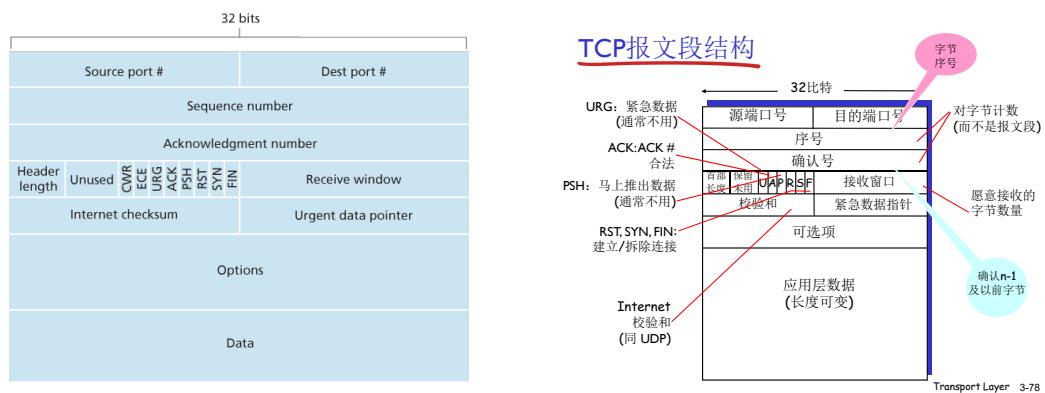


Figure 3.29 TCP segment structure

对部分内容的解释如下：

1. 4bit 的首部长度字段：由于 TCP 选项字段的原因，TCP 首部的长度是可变的。
2. 可选与变长的选项字段：用于发送方和接收方协商最大报文段长度 MSS 时，或在高速网络环境下用作窗口调节因子时使用
3. 还定义了一个时间戳选项

序号和确认号 TCP 把数据看成是一个无结构的、有序的字节流。一个报文段的序号是该报文段首字节的字节流编号。所以说相邻的报文段，其编号是不相邻的，间隔为 MSS。对于确认号，接收方发送的确认号是期望从发送方接到的下一**字节**的序号。TCP 采用的是提供累积确认

3.5.2 可靠数据传输

3.5.3 流量控制

3.5.4 连接管理

第 3.6 节 拥塞控制原理

第 3.7 节 TCP 拥塞控制