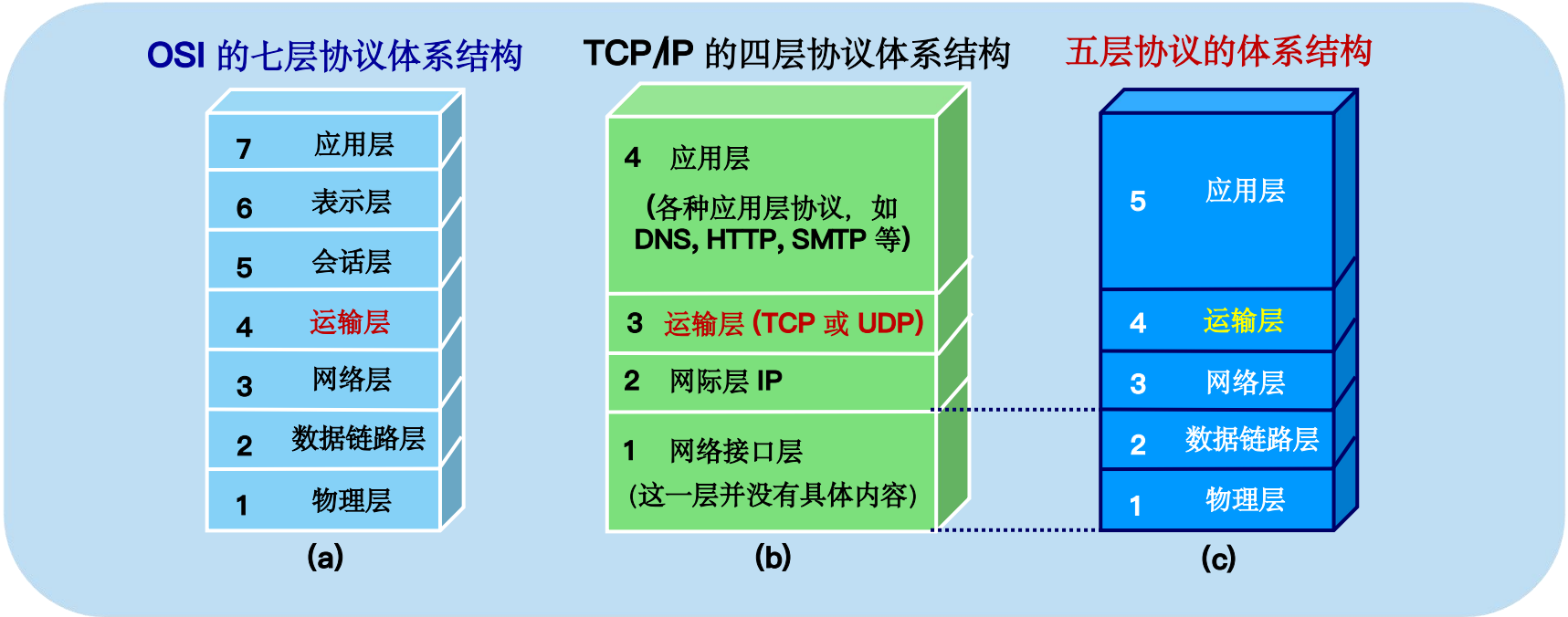
第5章
运输层

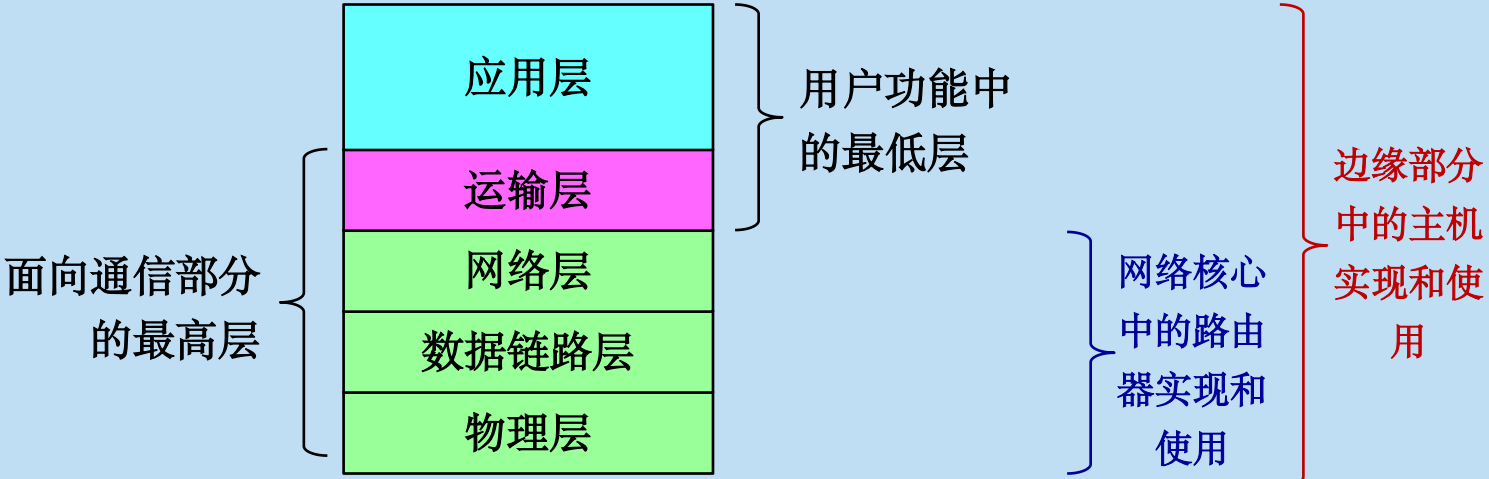
计算机网络体系结构


OSI 的七层协议体系结构
7 应用层
6 表示层
5 会话层
4 运输层
3 网络层
2 数据链路层
1 物理层
(a)
TCP/IP 的四层协议体系结构
4 应用层
(各种应用层协议，如 DNS, HTTP, SMTP 等)
3 运输层 (TCP 或 UDP)
2 网际层 IP
1 网络接口层
(这一层并没有具体内容)
(b)
五层协议的体系结构
5 应用层
4 运输层
3 网络层
2 数据链路层
1 物理层
(c)

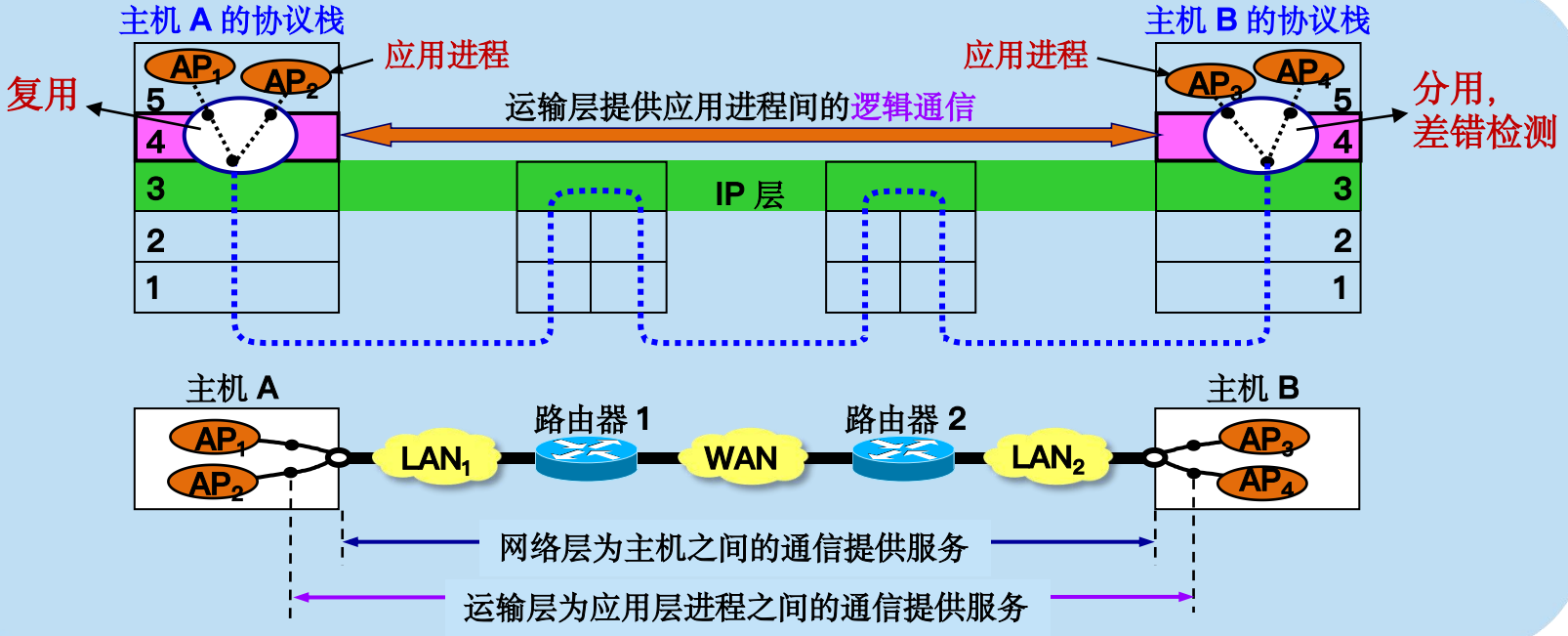
5.1 运输层协议概述

- 5.1.1 进程之间的通信
- 5.1.2 运输层的两个主要协议
- 5.1.3 运输层的端口

5.1.1 进程之间的通信


应用层
运输层
网络层
数据链路层
物理层
用户功能中的最低层
面向通信部分的最高层
网络核心中的路由器实现和使用
边缘部分中的主机实现和使用

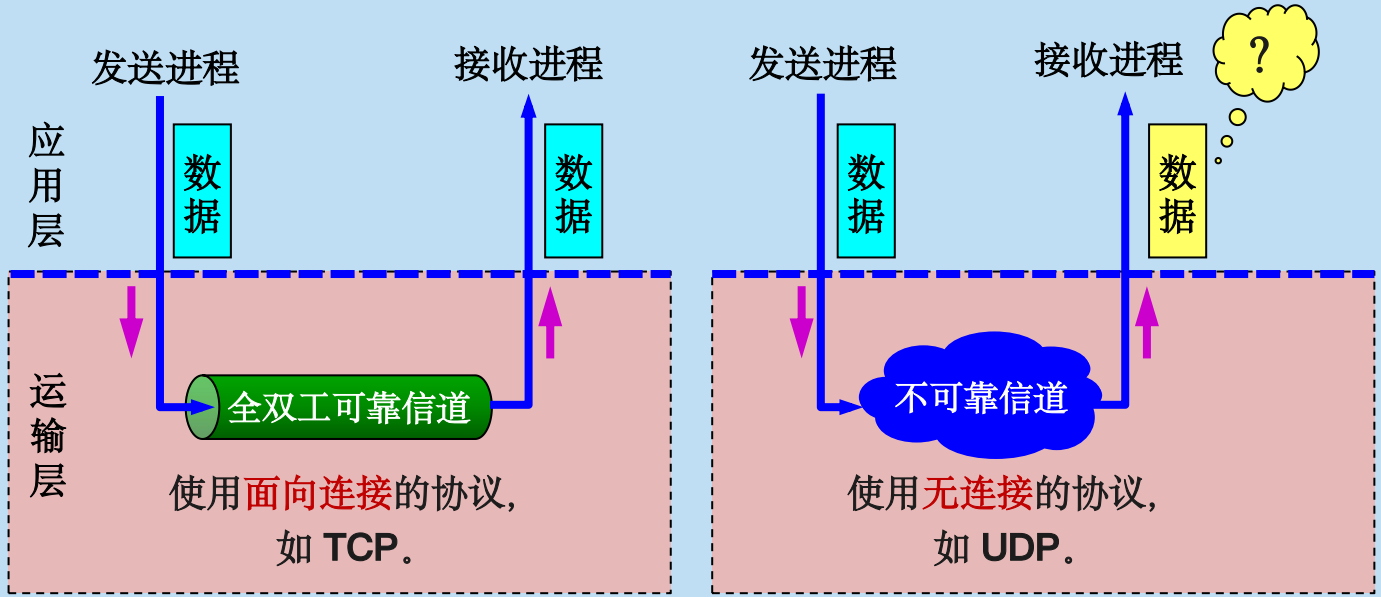
运输层的作用


主机 A 的协议栈
应用进程
复用
AP1
AP2
5
4
3
2
1
运输层提供应用进程间的逻辑通信
IP 层
主机 B 的协议栈
应用进程
AP3
AP4
分用，
差错检测
主机 A
路由器 1
路由器 2
主机 B
LAN1
WAN
LAN2
网络层为主机之间的通信提供服务
运输层为应用层进程之间的通信提供服务

屏蔽作用

- 运输层向高层用户屏蔽了下面网络核心的细节（如网络拓扑、所采用的路由选择协议等），使应用进程看见的就是好像在两个运输层实体之间有一条端到端的逻辑通信信道。

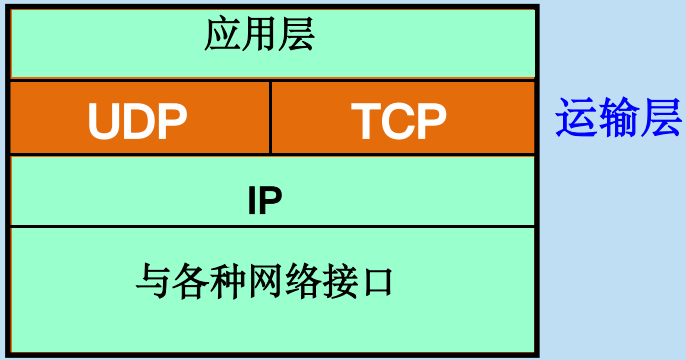
可靠信道与不可靠信道


发送进程
接收进程
应用层
数据
数据
运输层
全双工可靠信道
使用面向连接的协议，
如 TCP。
发送进程
接收进程
?
数据
数据
不可靠信道
使用无连接的协议，
如 UDP。

5.1.2 运输层的两个主要协议

互联网的正式标准：

1. 用户数据报协议 UDP (User Datagram Protocol)
2. 传输控制协议 TCP (Transmission Control Protocol)



- TCP 传送的数据单位协议是 TCP 报文段 (segment)。
- UDP 传送的数据单位协议是 UDP 报文或用户数据报。

UDP 与 TCP 的区别

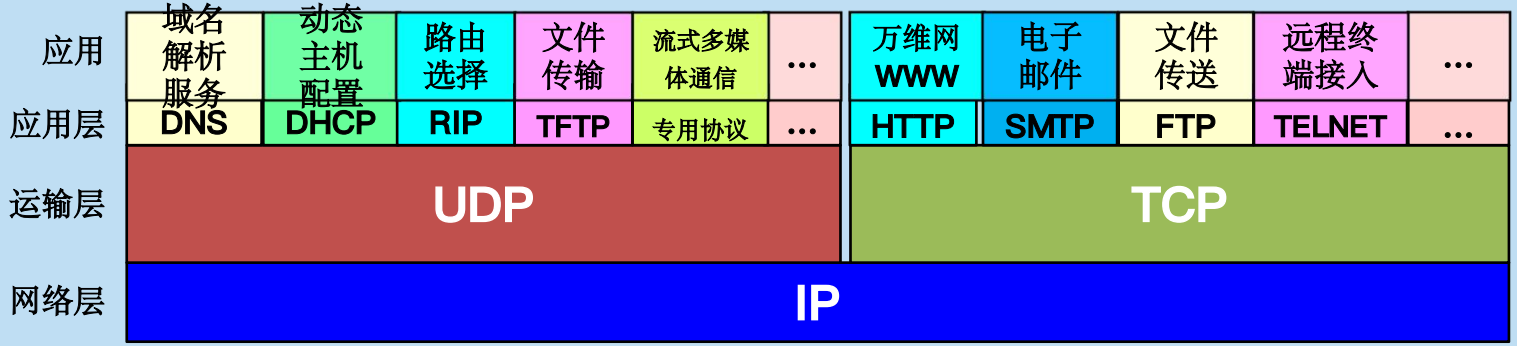
UDP

- 传送数据之前不需要先建立连接。
- 收到 UDP 报后，不需要给出任何确认。
- 不提供可靠交付，但是一种最有效的工作方式。

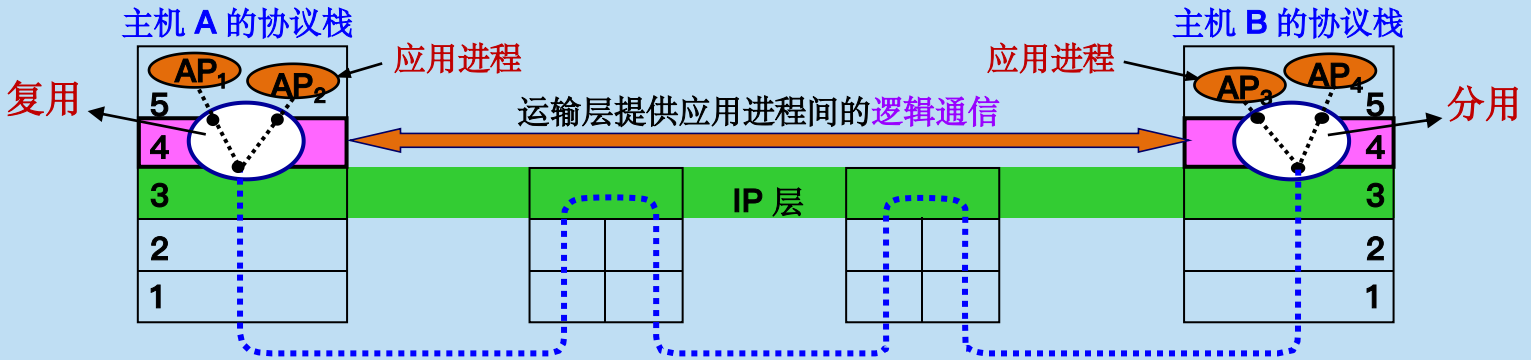
TCP

- 提供可靠的、面向连接的运输服务。
- 不提供广播或多播服务。
- 开销较多。

使用 UDP 和 TCP 的典型应用和应用层协议



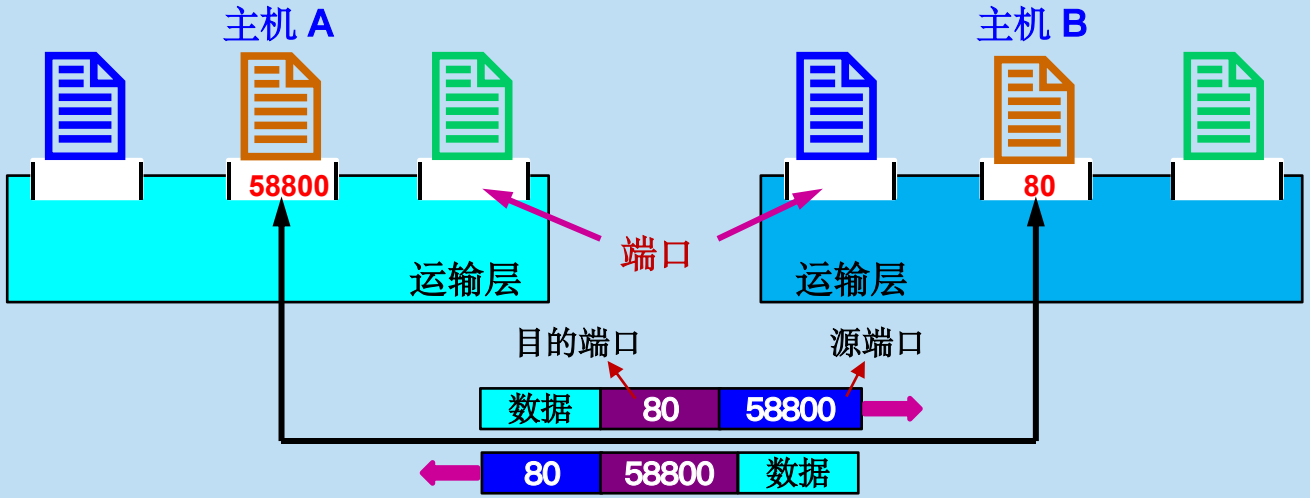
5.1.3 运输层的端口



- 复用：应用进程都可以通过运输层再传送到 IP 层（网络层）。
- 分用：运输层从 IP 层收到发送给应用进程的数据后，必须分别交付给指明的各应用进程。 如何指明各应用进程?

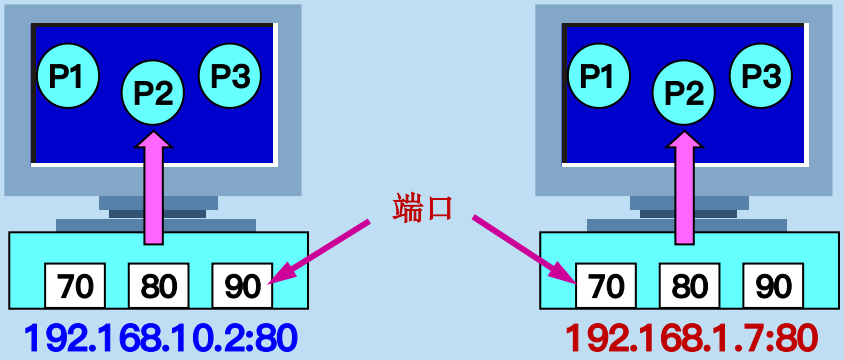
端口号 (protocol port number)

- 解决方法：在运输层使用协议端口号 (protocol port number)，或通常简称为端口 (port)。把端口设为通信的抽象终点。



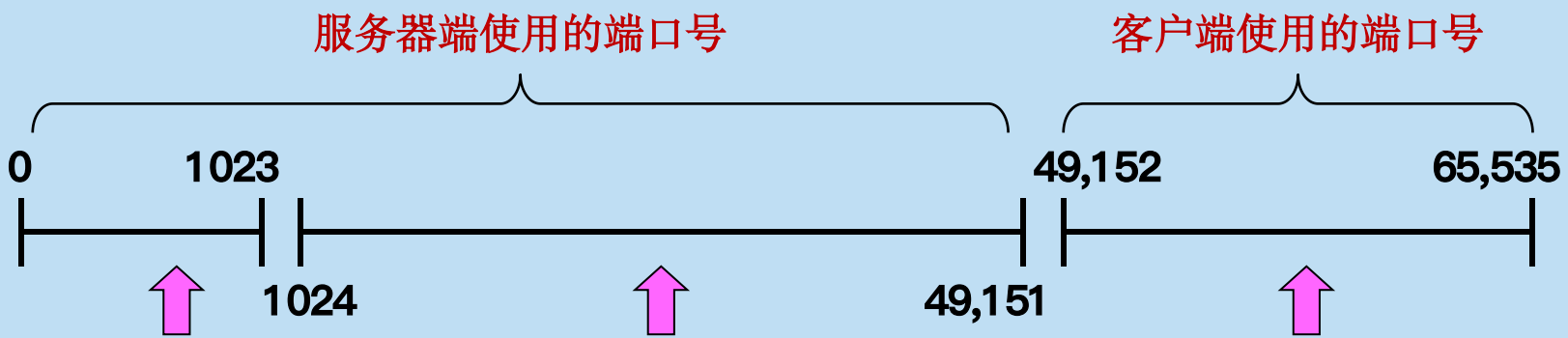
TCP/IP 运输层端口的标志

- 端口用一个 16 位端口号进行标志，允许有 65,535 个不同的端口号。
- 端口号只具有本地意义，只是为了标志本计算机应用层中的各进程。
- 在互联网中，不同计算机的相同端口号没有联系。

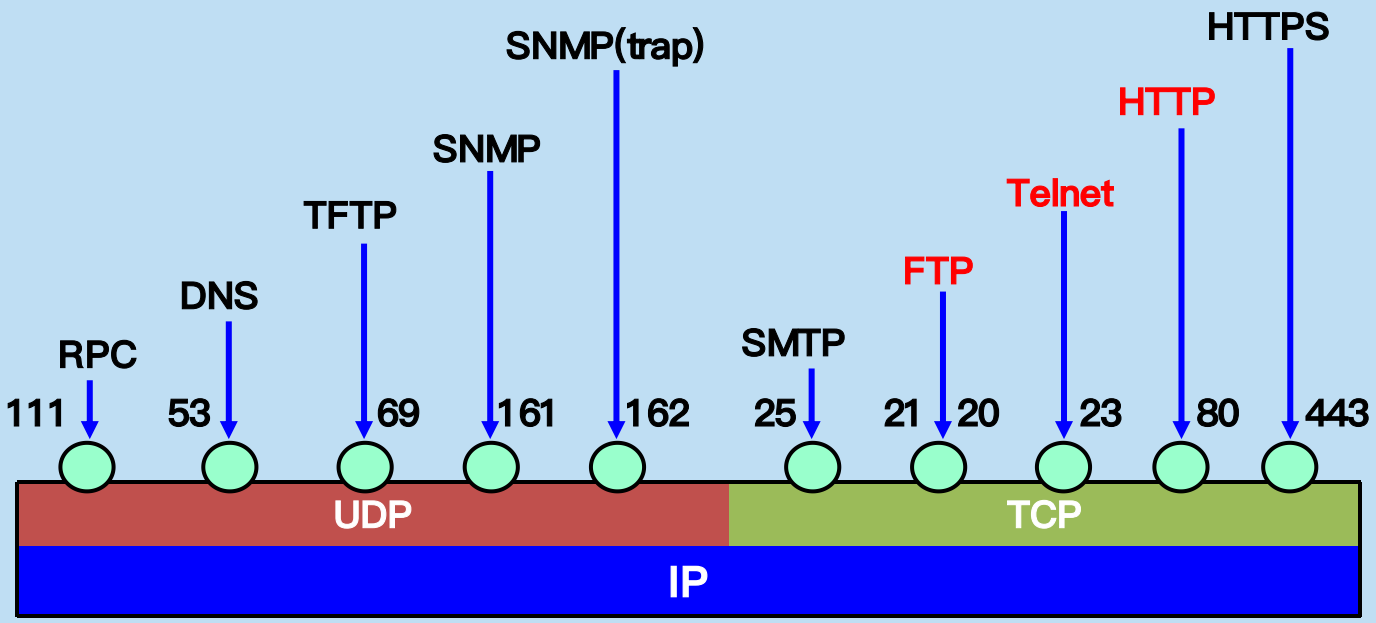


由此可见，两个计算机中的进程要互相通信，不仅必须知道对方的端口号，而且还要知道对方的 IP 地址。

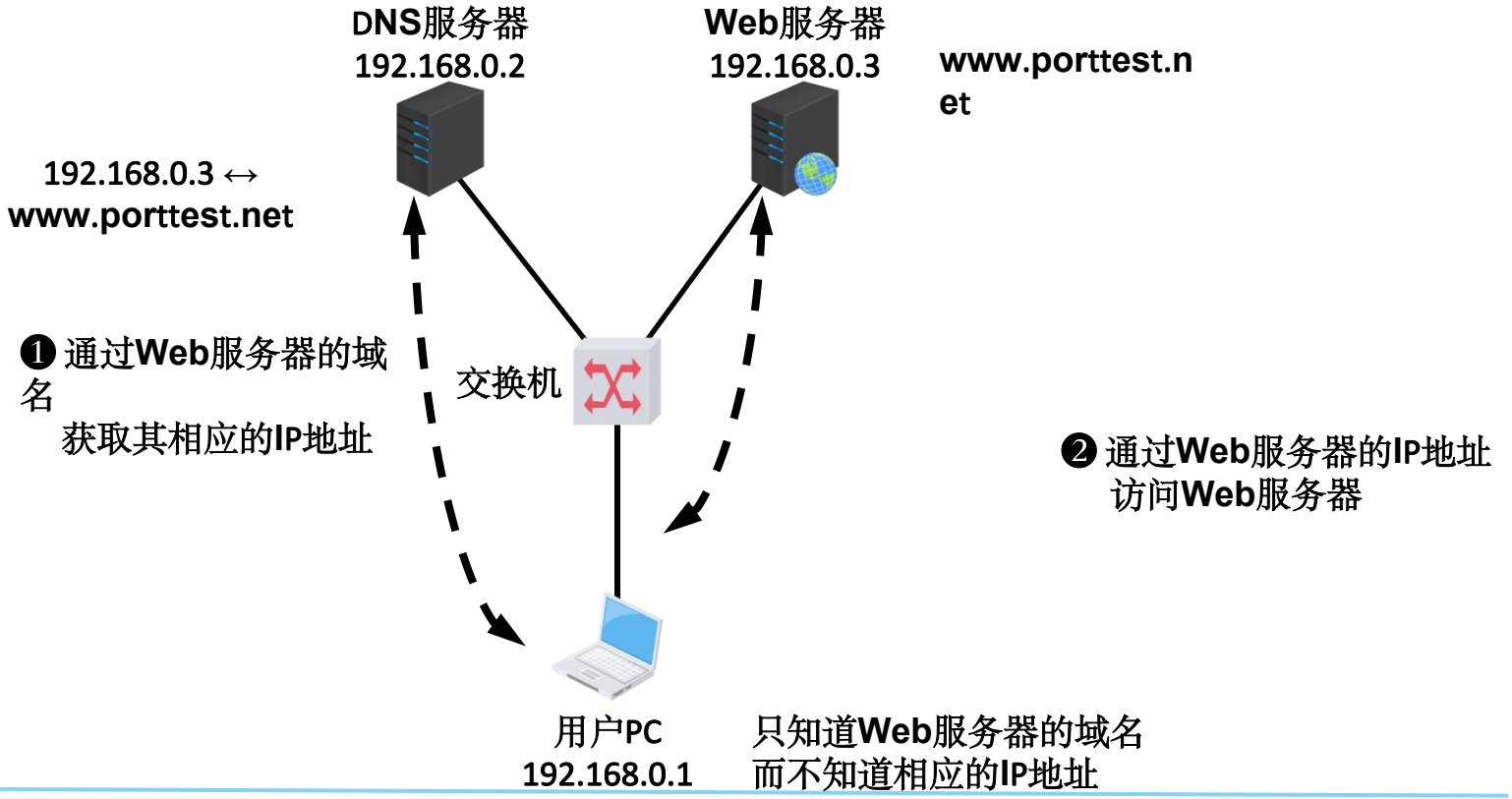
两大类、三种类型的端口


服务器端使用的端口号
客户端使用的端口号
0
1023
1024
49,151
49,152
65,535
熟知端口
(全球通用端口号)
(IANA 负责分配)
登记端口
(在 IANA 登记)
短暂端口
(通信结束后，被系统收回)

常用的熟知端口


SNMP(trap)
HTTPS
HTTP
SNMP
Telnet
TFTP
FTP
DNS
SMTP
RPC
111
53
69
161
162
25
21
20
23
80
443
UDP
TCP
IP

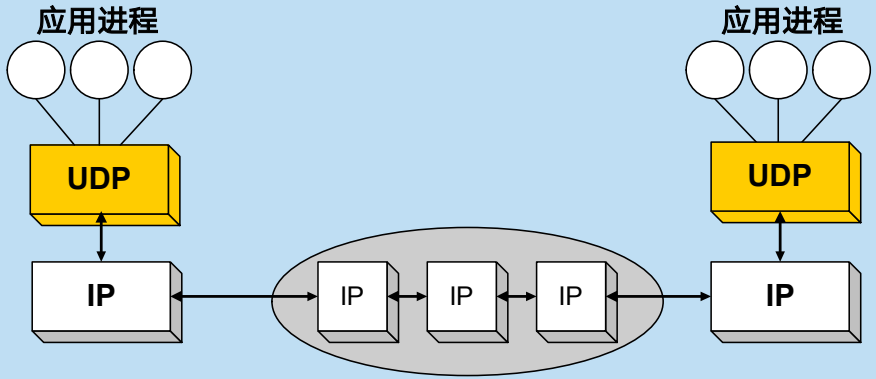
05 运输层端口号应用举例



5.2 用户数据报协议 UDP

5.2.1 UDP 概述

- UDP 只在 IP 的数据报服务之上增加了一些功能：
  1. 复用和分用
  2. 差错检测



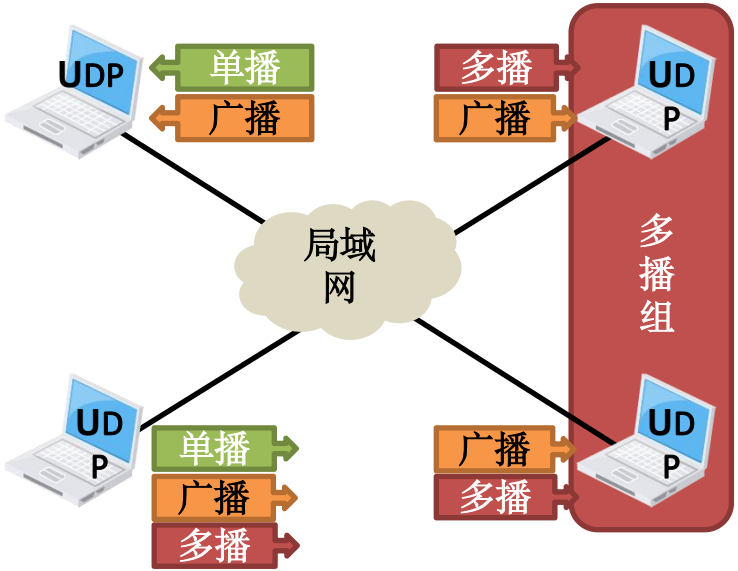
UDP 的主要特点

1. 无连接。发送数据之前不需要建立连接。
2. 使用尽最大努力交付。即不保证可靠交付。
3. 面向报文。UDP 一次传送和交付一个完整的报文。
4. 没有拥塞控制。网络出现的拥塞不会使源主机的发送速率降低。很适合多媒体通信的要求。
5. 支持一对一、一对多、多对一、多对多等交互通信。
6. 首部开销小，只有 8 个字节。

UDP 通信的特点：简单方便，但不可靠。

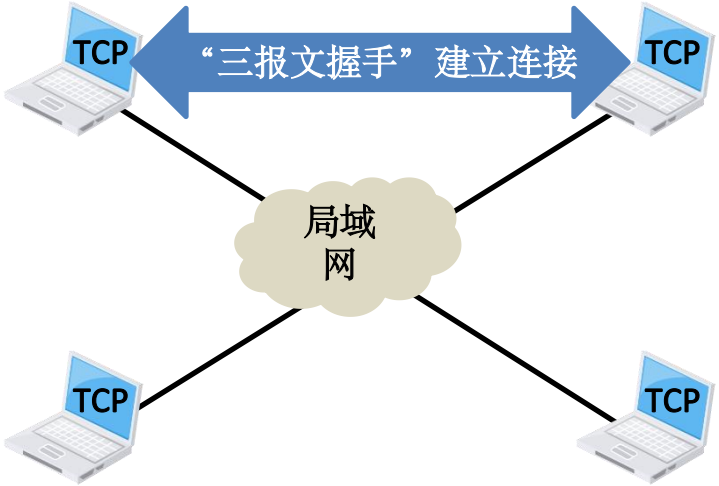
02 UDP和TCP对单播、多播和广播的支持情况

用户数据报协议UDP
(User Datagram Protocol)

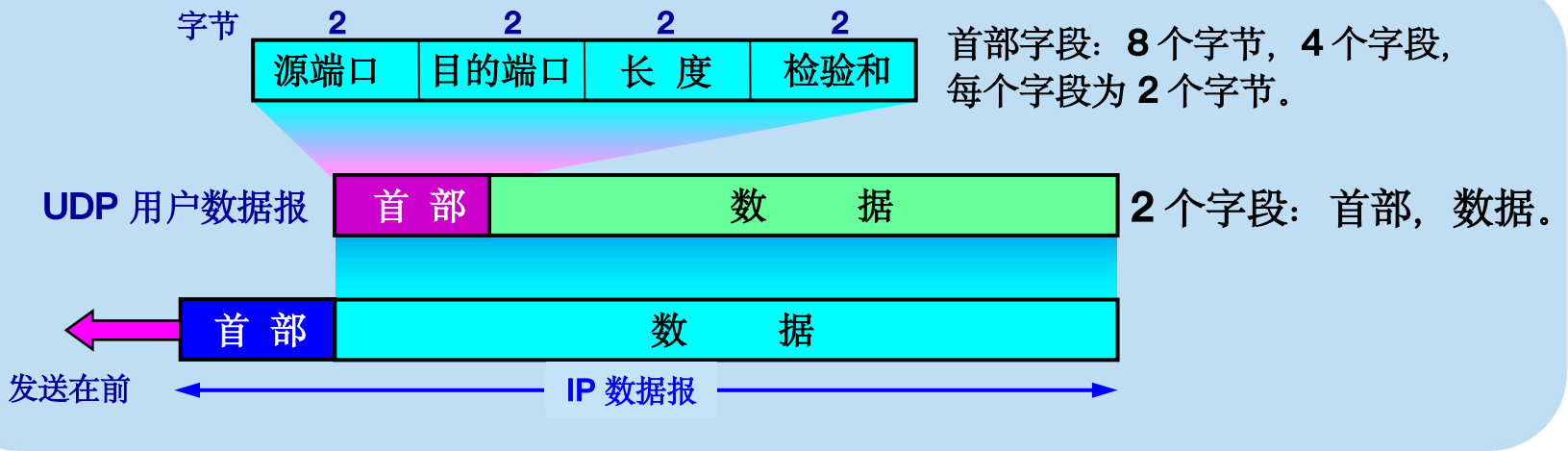


UDP支持单播、多播和广播

传输控制协议TCP
(Transmission Control Protocol)



5.2.2 UDP 的首部格式



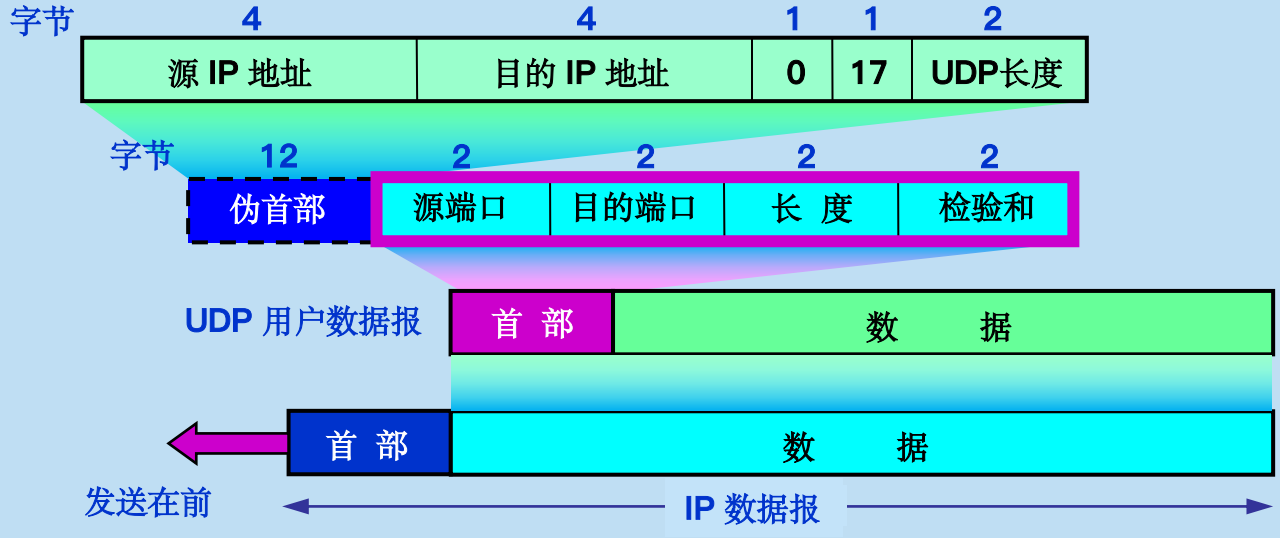
(1) 源端口：源端口号。在需要对方回信时选用。不需要时可用全 0。

(2) 目的端口：目的端口号。终点交付报文时必须使用。

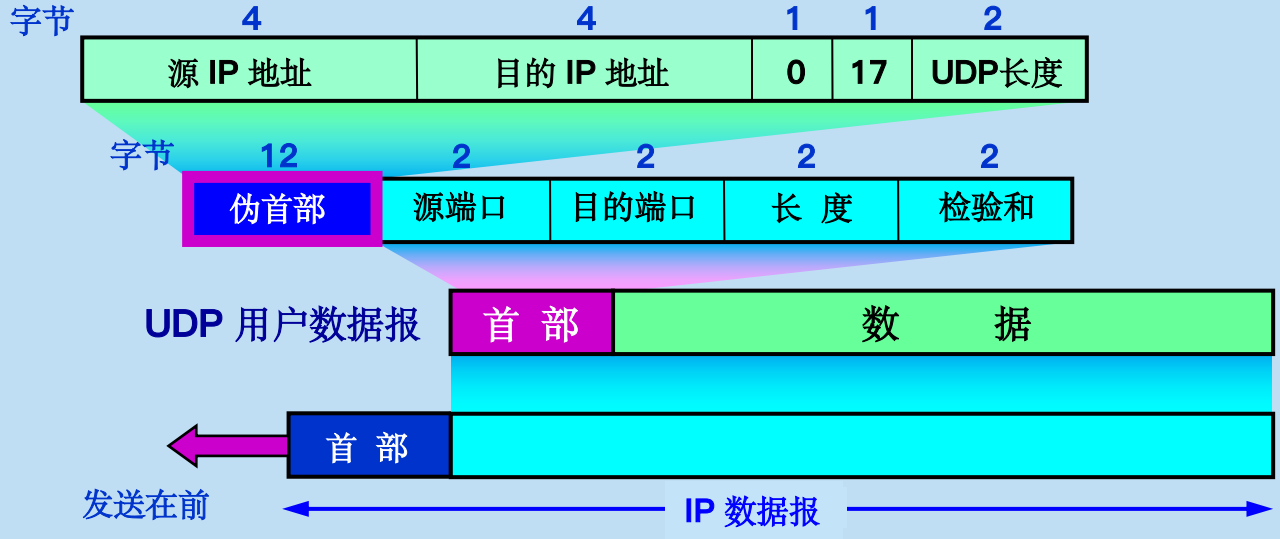
(3) 长度：UDP 用户数据报的长度，其最小值是 8（仅有首部）。

(4) 检验和：检测 UDP 用户数据报在传输中是否有错。有错就丢弃。

用户数据报 UDP 有两个字段：数据字段和首部字段。首部字段有 8 个字节，由 4 个字段组成，每个字段都是 2 个字节。



在计算检验和时，临时把 12 字节的“伪首部”和 UDP 用户数据报连接在一起。伪首部仅仅是为了计算检验和。

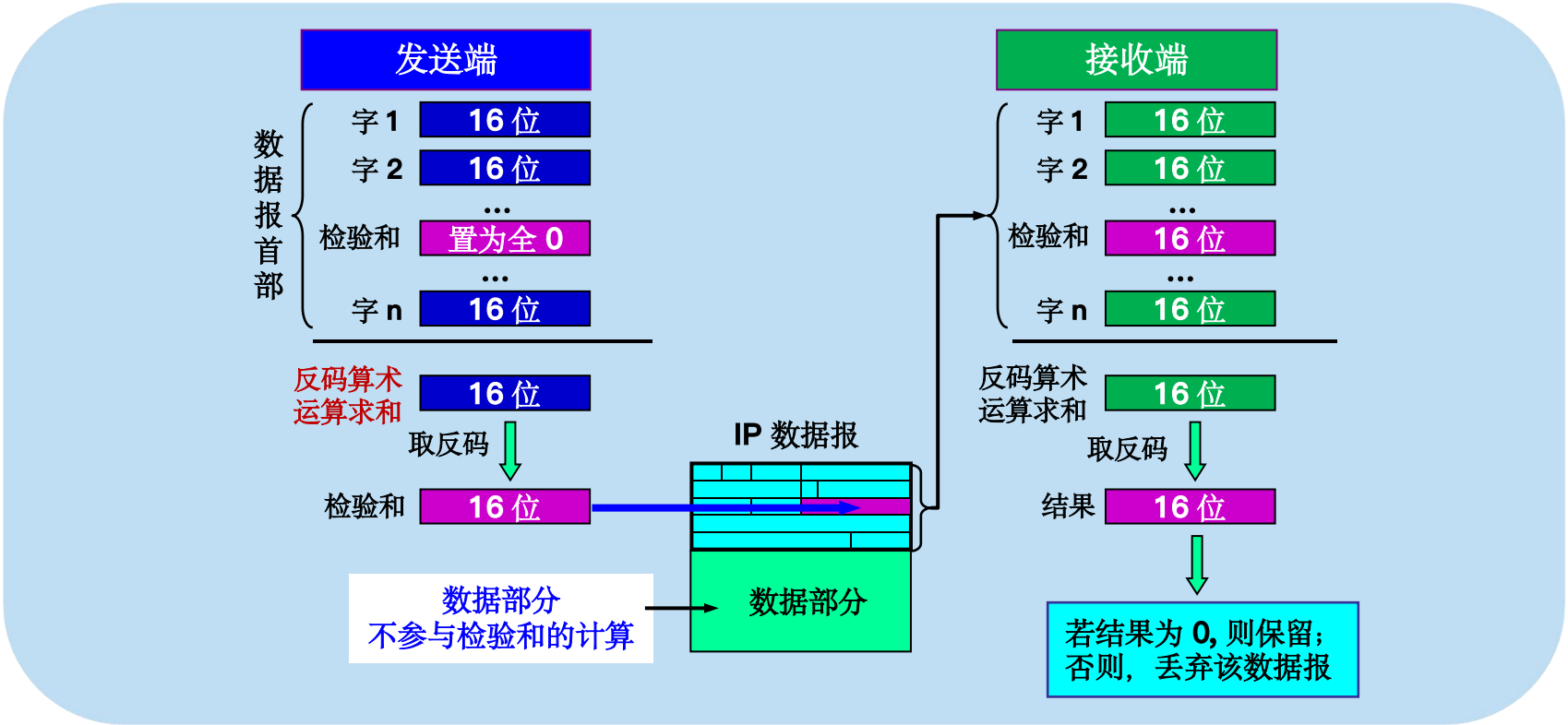


计算 UDP 检验和的例子

UDP 的检验和是把首部和数据部分一起都检验。

10011001 00010011 → 153.19
00001000 01101000 → 8.104
10101011 00000011 → 171.3
00001110 00001011 → 14.11
00000000 00010001 → 0 和 17
00000000 00001111 → 15
00000100 00111111 → 1087
00000000 00001101 → 13
00000000 00001111 → 15
00000000 00000000 → 0（检验和）
01010100 01000101 → 数据
01010011 01010100 → 数据
01001001 01001110 → 数据
01000111 00000000 → 数据和 0（填充）

按二进制反码运算求和 10010110 11101101 → 求和得出的结果
将得出的结果求反码 01101001 00010010 → 检验和

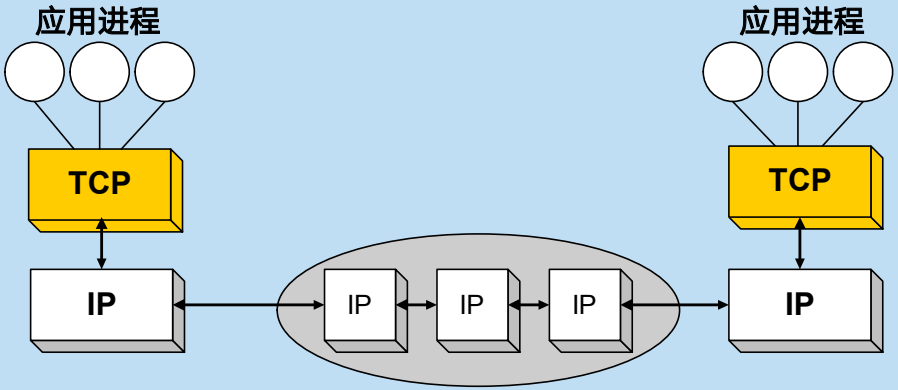


数据报每经过一个路由器，路由器都要重新计算一下首部检验和

5.3 传输控制协议 TCP 概述

5.3.1 TCP 最主要的特点

- TCP 是面向连接的运输层协议，在无连接的、不可靠的 IP 网络服务基础之上提供可靠交付的服务。为此，在 IP 的数据报服务基础之上，增加了保证可靠性的一系列措施。



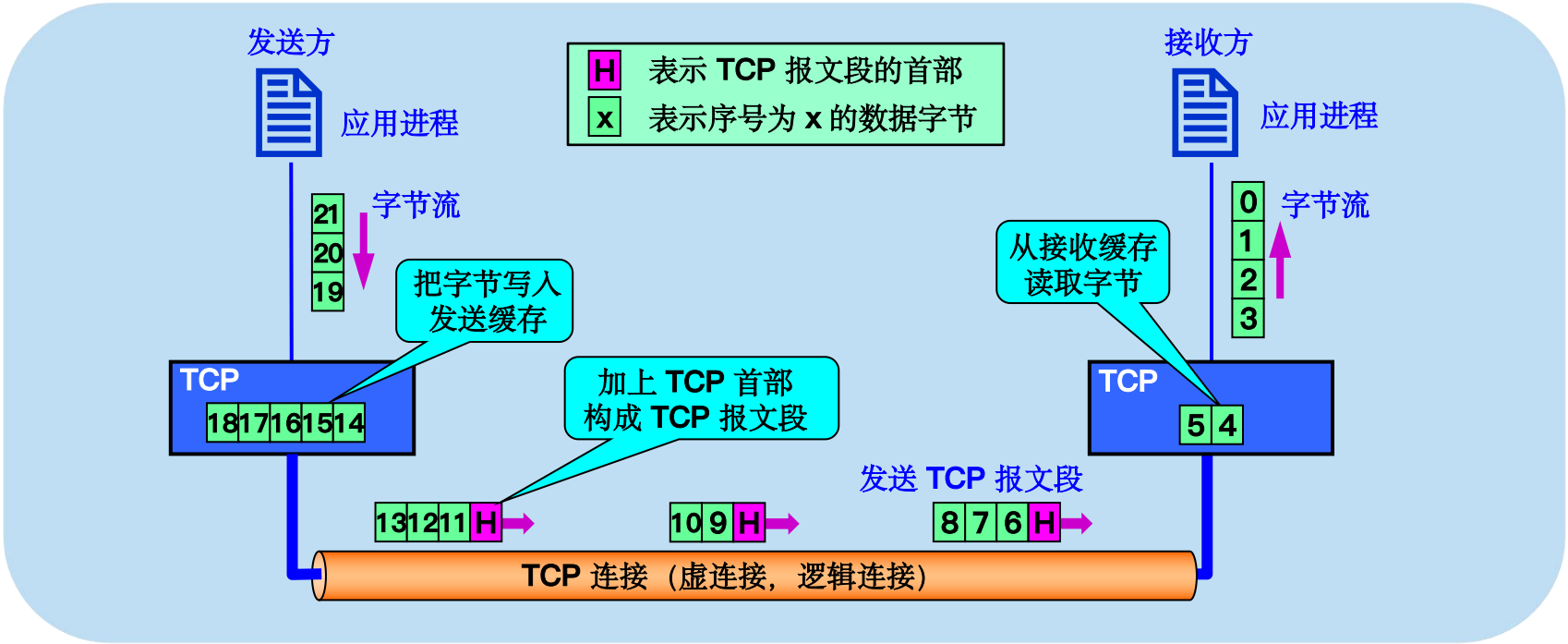
5.3.1 TCP 最主要的特点

- TCP 是面向连接的运输层协议。
- 每一条 TCP 连接只能有两个端点 (endpoint)，每一条 TCP 连接只能是点对点的（一对一）。
- TCP 提供可靠交付的服务。
- TCP 提供全双工通信。
- 面向字节流
  1. TCP 中的“流”(stream) 指的是流入或流出进程的字节序列。
  2. 面向字节流：虽然应用程序和 TCP 的交互是一次一个数据块，但 TCP 把应用程序交下来的数据看成仅仅是一连串无结构的字节流。

TCP 面向流的概念

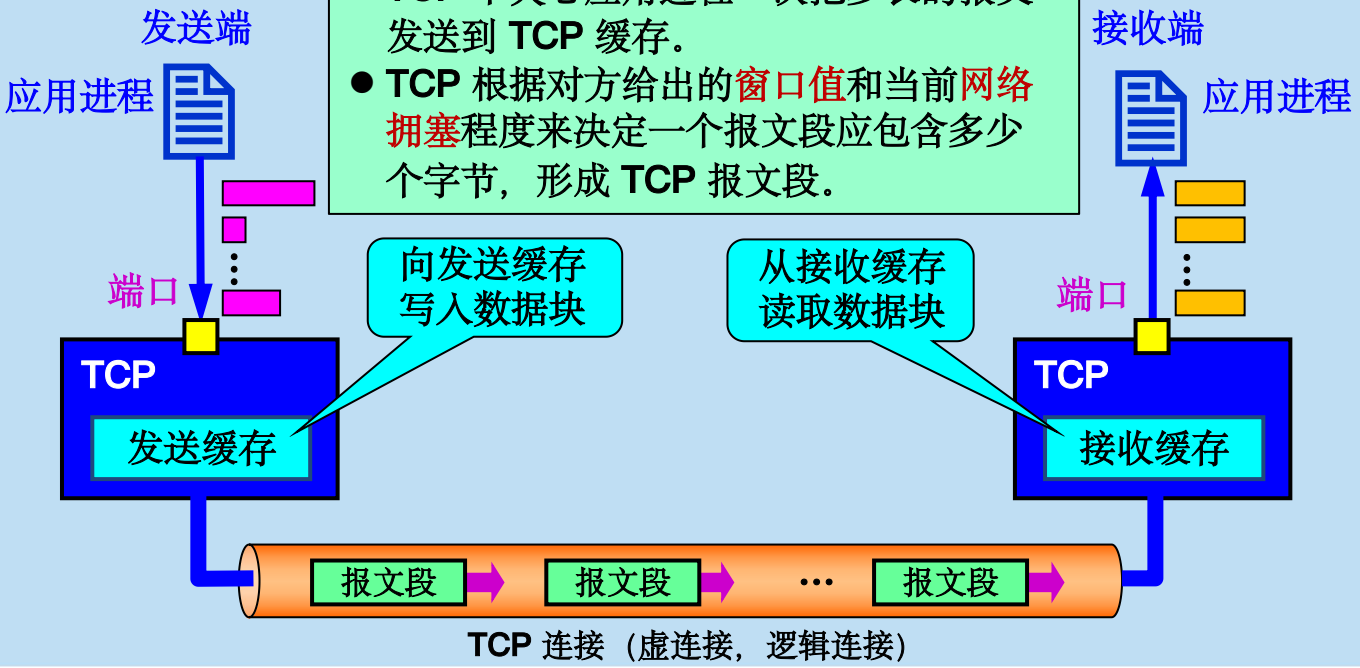
- TCP 不保证接收方应用程序所收到的数据块和发送方应用程序所发出的数据块具有对应大小的关系。
- 但接收方应用程序收到的字节流必须和发送方应用程序发出的字节流完全一样。

TCP 面向流的概念


发送方
应用进程
H 表示 TCP 报文段的首部
x 表示序号为 x 的数据字节
接收方
应用进程
字节流
21
20
19
把字节写入
发送缓存
TCP
18 17 16 15 14
加上 TCP 首部
构成 TCP 报文段
13 12 11 H
10 9 H
发送 TCP 报文段
8 7 6 H
TCP 连接（虚连接，逻辑连接）
从接收缓存
读取字节
0
1
2
3
字节流
TCP
5 4

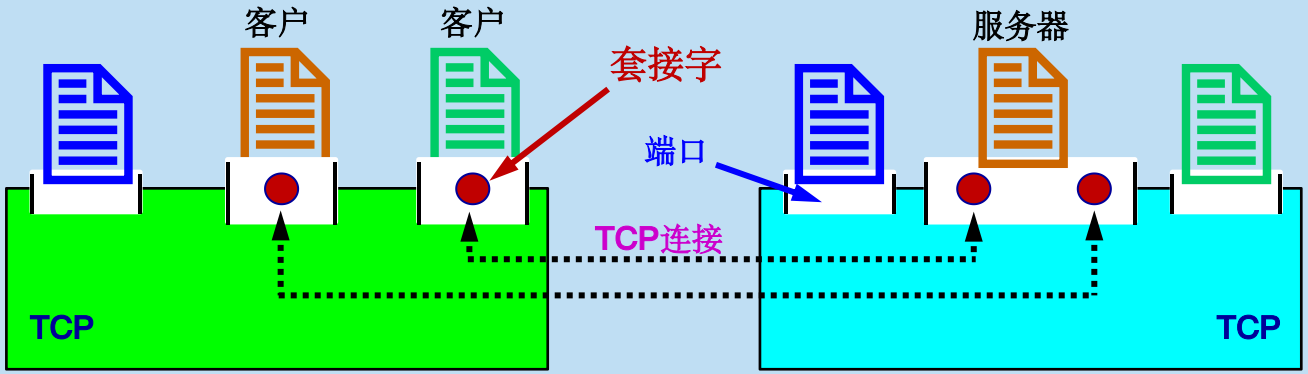
TCP 面向流的概念

- TCP 不关心应用进程一次把多长的报文发送到 TCP 缓存。
- TCP 根据对方给出的窗口值和当前网络拥塞程度来决定一个报文段应包含多少个字节，形成 TCP 报文段。



5.3.2 TCP 的连接

- TCP 把连接作为最基本的抽象。



每一条 TCP 连接有两个端点。

TCP 连接的端点：套接字 (socket) 或插口。

套接字 (socket)

套接字 socket = (IP地址 : 端口号)　　(5-1)

例如:

套接字 socket = (192.169.1.20 : 2028)

每一条 TCP 连接唯一地被通信两端的两个端点（即两个套接字）所确定:

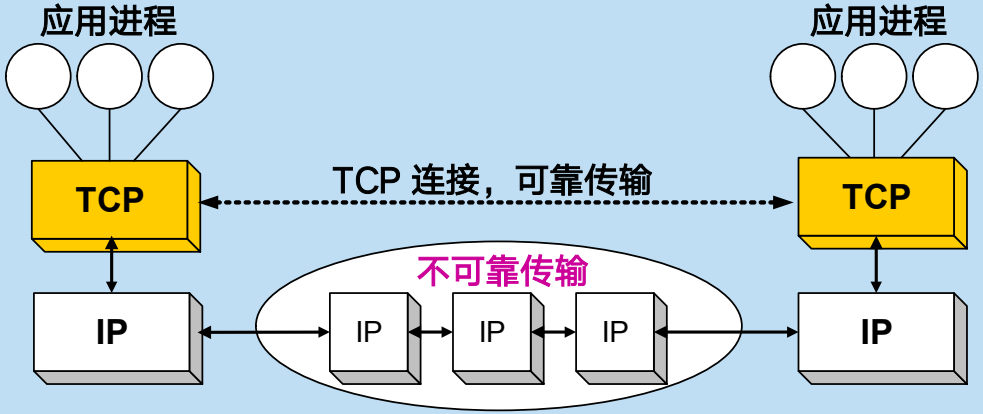
TCP 连接 ::= {$socket_1$, $socket_2$} = {(IP_1: $port_1$), (IP_2: $port_2$)}　　(5-2)

TCP 连接，IP 地址，套接字

- TCP 连接就是由协议软件所提供的一种抽象。
- TCP 连接的端点是抽象的套接字，即（IP 地址：端口号）。
- 同一个 IP 地址可以有多个不同的 TCP 连接。
- 同一个端口号也可以出现在多个不同的 TCP 连接中。

5.4 可靠传输的工作原理

IP 网络提供的是不可靠的传输



理想传输条件的特点

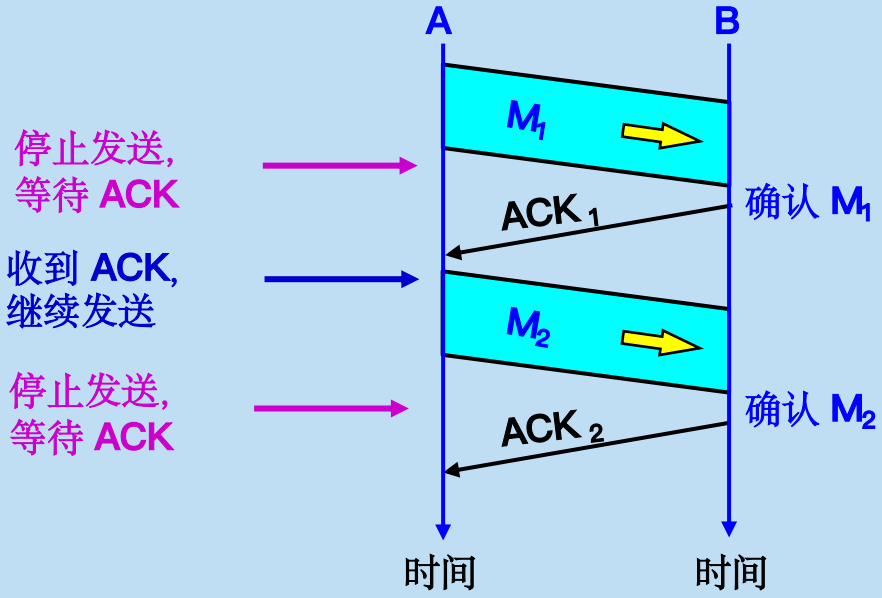
1. 传输信道不产生差错。
2. 不管发送方以多快的速度发送数据，接收方总是来得及处理收到的数据。

- 在理想传输条件下，不需要采取任何措施就能够实现可靠传输。
- 但实际网络都不具备理想传输条件。必须使用一些可靠传输协议，在不可靠的传输信道实现可靠传输。

5.4.1 停止等待协议

- 每发送完一个分组就停止发送，等待对方的确认。在收到确认后再发送下一个分组。
- 全双工通信的双方既是发送方也是接收方。
- 假设仅考虑 A 发送数据，而 B 接收数据并发送确认。因此 A 叫做发送方，而 B 叫做接收方。

1. 无差错情况



- A 发送完分组 M_1 后就暂停发送，等待 B 的确认 (ACK)。
- B 收到 M_1 向 A 发送 ACK。
- A 在收到了对 M_1 的确认后，就再发送下一个分组 M_2。

2. 出现差错

- 两种情况：
  1. B 接收 M_1 时检测出了差错，就丢弃 M_1，其他什么也不做（不通知 A 收到有差错的分组）。
  2. M_1 在传输过程中丢失了，这时 B 当然什么都不知道，也什么都不做。

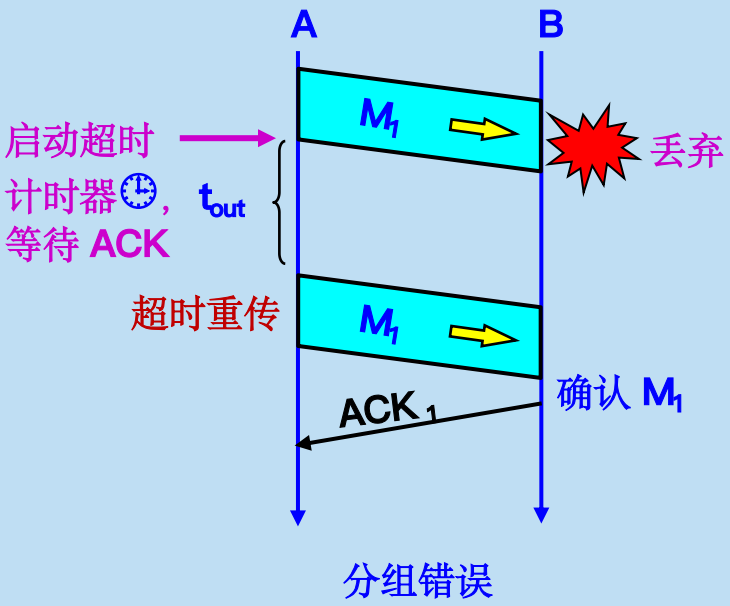
- 在这两种情况下，B 都不会发送任何信息。

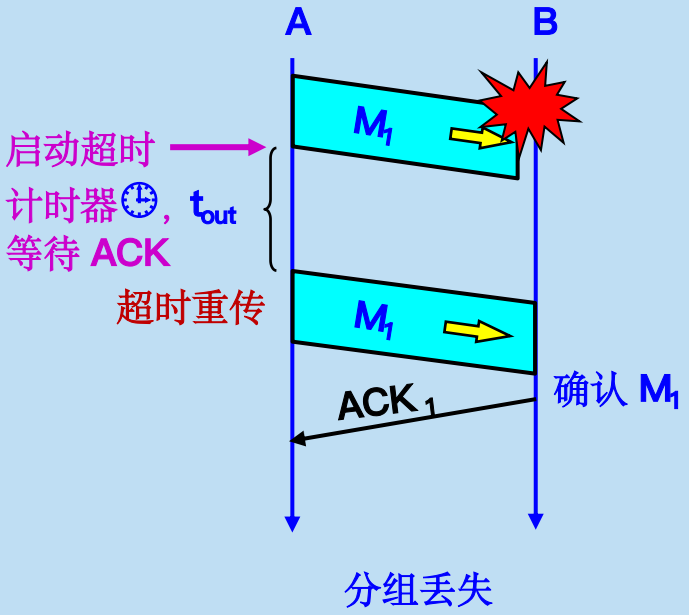
2. 出现差错

- 问题：A 如何知道 B 是否正确收到了 M_1 呢?
- 解决方法：超时重传

1. A 为每一个已发送的分组设置一个超时计时器。
2. A 只要在超时计时器到期之前收到了相应的确认，就撤销该超时计时器，继续发送下一个分组 M_2 。
3. 若 A 在超时计时器规定时间内没有收到 B 的确认，就认为分组错误或丢失，就重发该分组。

2. 出现差错


A
B
启动超时
计时器，
等待 ACK
t_{out}
M_1
丢弃
超时重传
M_1
ACK_1
确认 M_1
分组错误


A
B
启动超时
计时器，
等待 ACK
t_{out}
M_1
超时重传
M_1
ACK_1
确认 M_1
分组丢失

3. 确认丢失和确认迟到

- 确认丢失

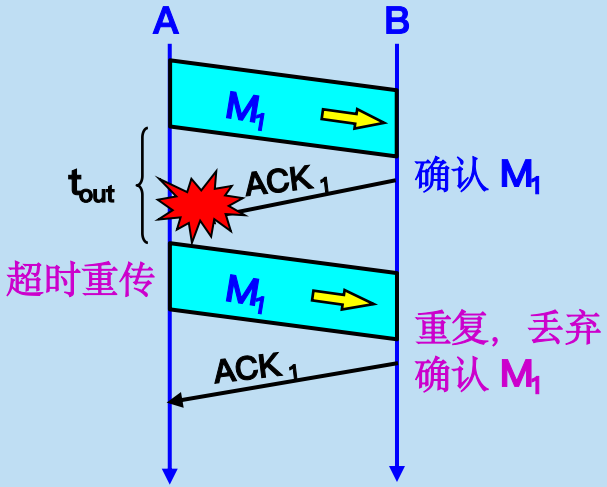
1. 若 B 所发送的对 M_1 的确认丢失了，那么 A 在设定的超时重传时间内将不会收到确认，因此 A 在超时计时器到期后重传 M_1。
2. 假定 B 正确收到了 A 重传的分组 M_1。这时 B 应采取两个行动：
   - (1) 丢弃这个重复的分组 M_1，不向上层交付。
   - (2) 向 A 发送确认。

3. 确认丢失和确认迟到

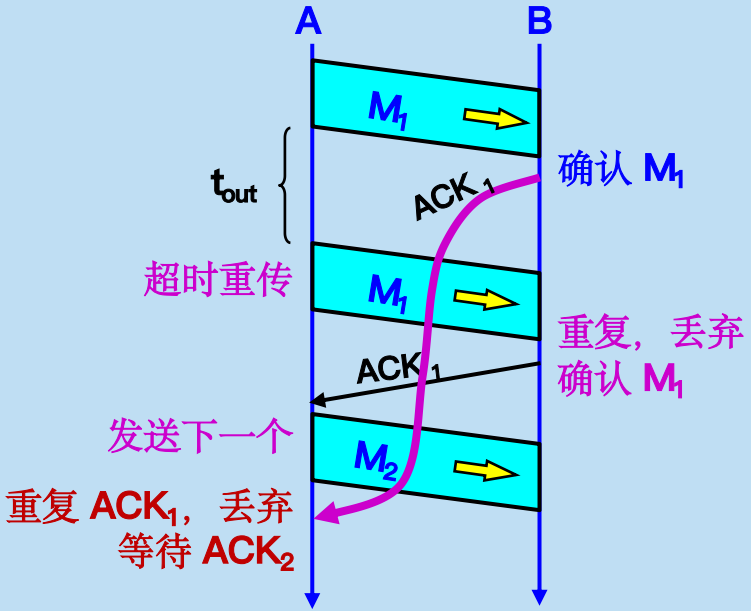
- 确认迟到

1. B 对分组 M_1 的确认迟到了，因此 A 在超时计时器到期后重传 M_1。
2. B 会收到重复的 M_1，丢弃重复的 M_1，并重传确认分组。
3. A 会收到重复的确认。对重复的确认的处理：丢弃。

3. 确认丢失和确认迟到

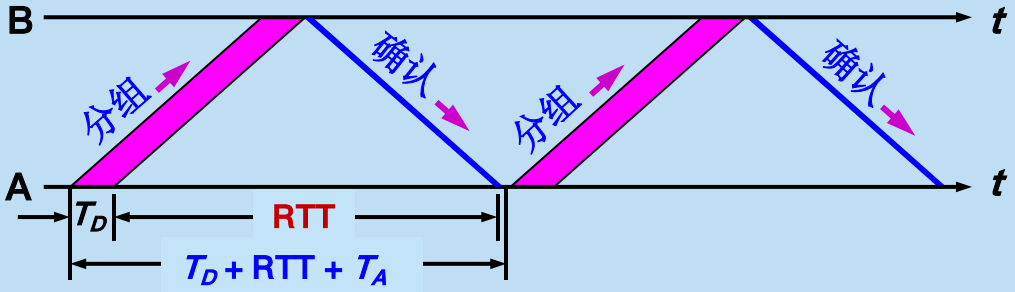


确认丢失



确认迟到

4. 信道利用率



信道利用率 $$U = \frac{T_D}{T_D + \mathrm{RTT} + T_A}$$ (5-3)

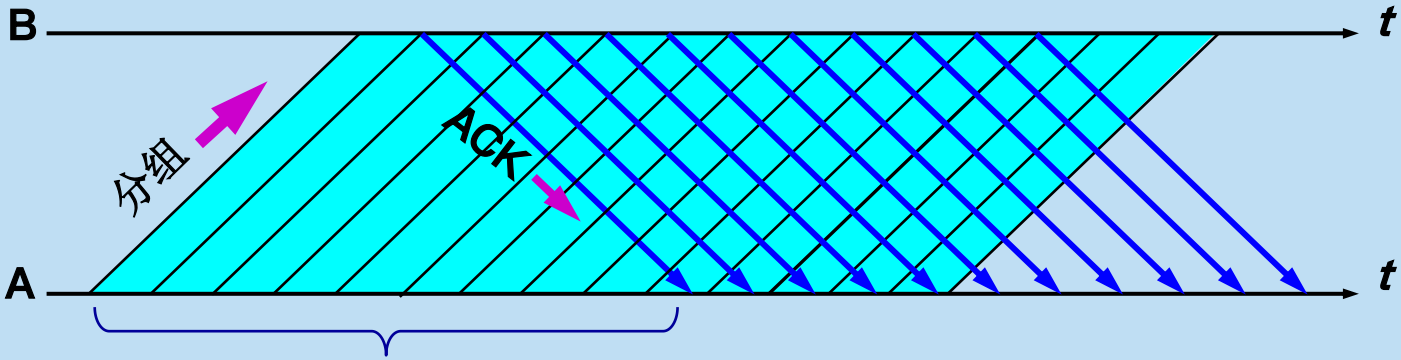
当往返时间 RTT 远大于分组发送时间 T_D 时，信道的利用率会非常低。

优点：简单。缺点：信道利用率太低。

停止等待协议要点

- 停止等待。发送方每次只发送一个分组。在收到确认后再发送下一个分组。
- 暂存：在发送完一个分组后，发送方必须暂存已发送的分组的副本，以备重发。
- 编号。对发送的每个分组和确认都进行编号。
- 超时重传。发送方为发送的每个分组设置一个超时计时器。若超时计时器超时位收到确认，发送方会自动超时重传分组。
- 超时计时器的重传时间应当比数据在分组传输的平均往返时间更长一些，防止不必要的重传。
- 简单，但信道利用率太低。

提高传输效率：流水线传输



流水线传输：在收到确认之前，发送方连续发出多个分组。

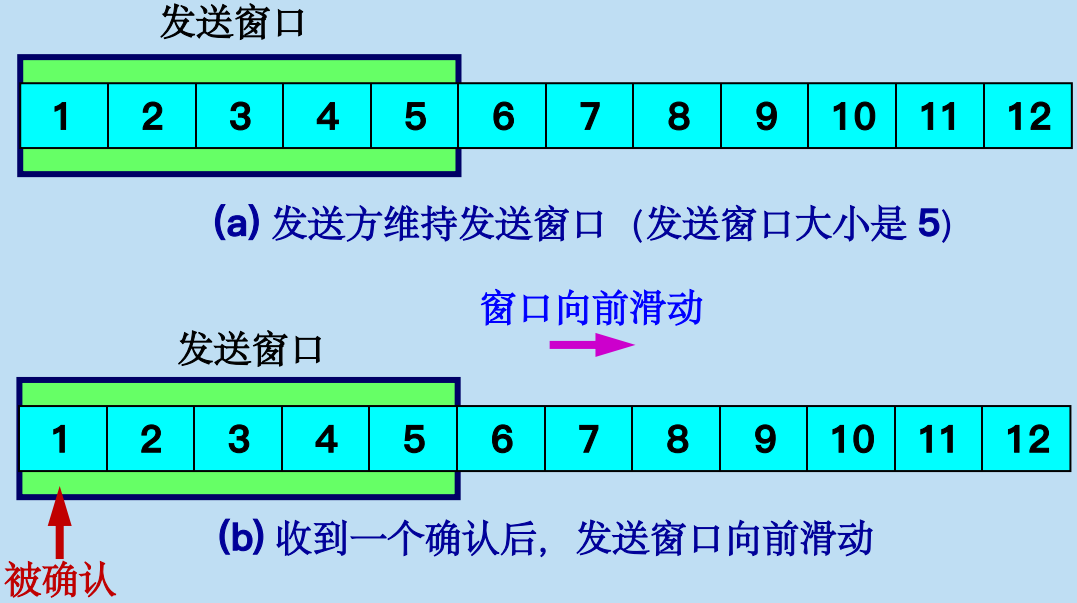
由于信道上一直有数据不间断地传送，
流水线传输可获得很高的信道利用率。

连续 ARQ 协议和滑动窗口协议采用流水线传输方式。

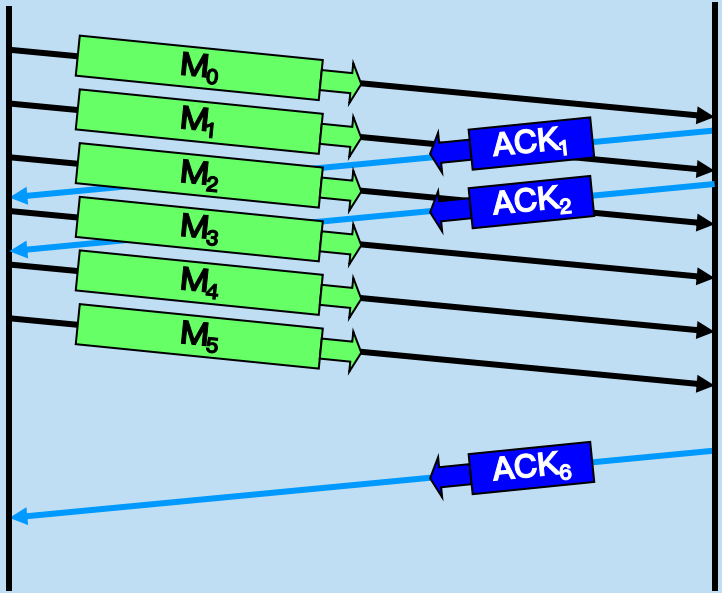
5.4.2 连续 ARQ 协议

- 发送窗口：发送方维持一个发送窗口，位于发送窗口内的分组都可被连续发送出去，而不需要等待对方的确认。
- 发送窗口滑动：发送方每收到一个确认，就把发送窗口向前滑动一个分组的位置。
- 累积确认：接收方对按序到达的最后一个分组发送确认，表示：到这个分组为止的所有分组都已正确收到了。

发送窗口


发送窗口
1 2 3 4 5 6 7 8 9 10 11 12
(a) 发送方维持发送窗口（发送窗口大小是 5)
窗口向前滑动
发送窗口
1 2 3 4 5 6 7 8 9 10 11 12
(b) 收到一个确认后，发送窗口向前滑动
被确认

累积确认



ACK_1 确认 M_0，将 M_0 提交给上层协议或用户

ACK_2 确认 M_1，将 M_1 提交给上层协议或用户

M_2 正确

M_3 正确

M_4 正确

M_5 正确

ACK_6 为累积确认，表示 M_5 及之前的 M_2、M_3、M_4 都正确。

将M_2、M_3、M_4、M_5提交给上层协议或用户