传输层

网络层只把分组发送到了目的主机，但是真正通信的并不是主机而是主机中的进程。传输层提供了进程间的逻辑通信，传输层向高层用户屏蔽了下面网络层的核心细节，是应用程序看起来像是两个传输层实体之间有一条端到端的路基通信信道；

UDP和TCP的特点

用户数据报协议UDP是无连接的，尽最大可能交付的，没有拥塞控制，面向报文(对上层协议的报文不作任何处理直接加上UDP首部)，支持一对一，一对多，多对一和多对多的交互通信；

传输控制协议TCP，是面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流(把应用层的报文看成是字节流，把字节流组织成大小不等的数据块)，每一条TCP连接只能是点对点的；

UDP首部格式

首部只有8个字节，包括源端口，目的端口，长度，检验和。12字节的伪首部是为了计算检验和临时添加的；

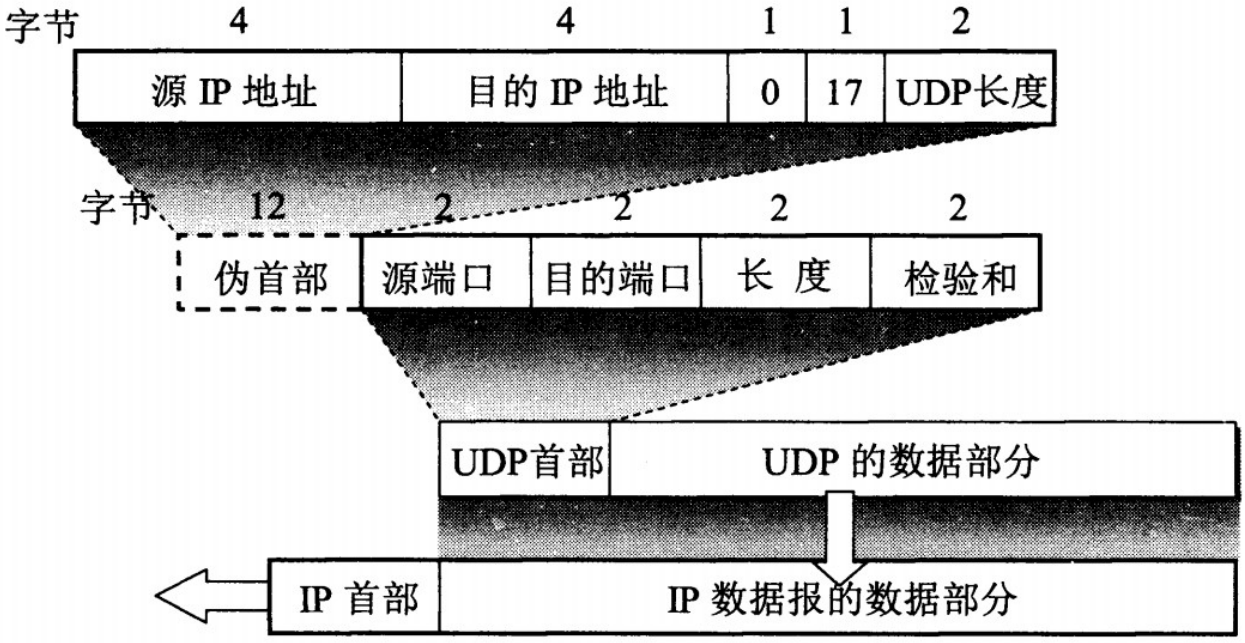

图 5-5　UDP 用户数据报的首部和伪首部

TCP首部格式

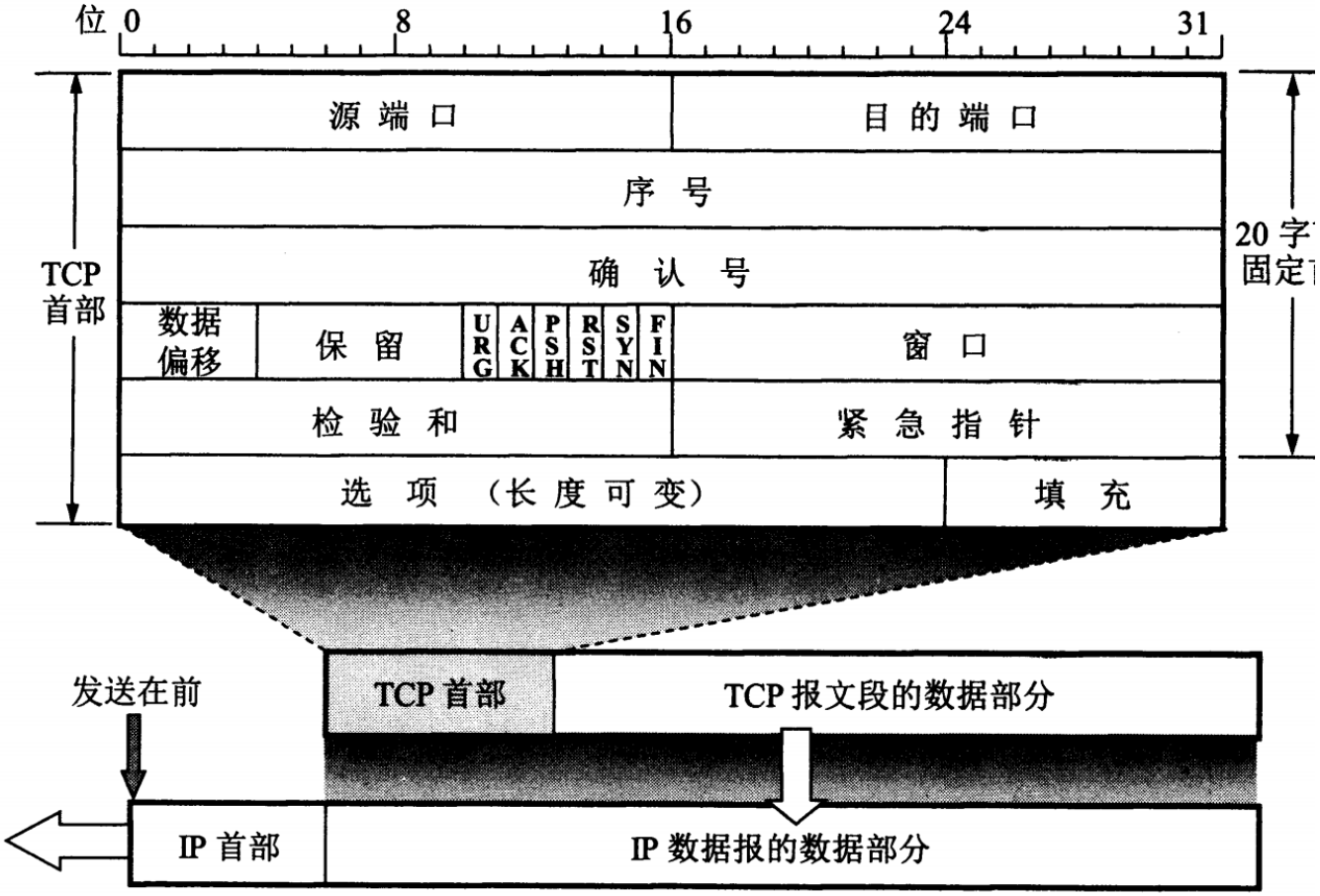

图 5-14　TCP 报文段的首部格式

1. **序号**：用于对字节流编号，例如序号为 301，表示第一个字节的编号为 301，如果携带的数据长度为 100 字节，那么下一个报文段的序号应为 401；
2. **确认号**：期望收到的下一个报文段的序号。例如A已经收到了序号是501的报文，长度200，那么期望下一个就是701，所以B给A的确认报文确认号就是701；
3. **数据偏移**：指的是数据部分距离报文段起始位置的偏移量，实际指的就是首部的长度；
4. **确认ACK**：当ACK=1时，确认号字段有效，否则无效。TCP规定，建立连接之后的报文段必须把ACK置一；
5. **同步SYN**：连接建立时用来同步序号。当SYN=1，ACK=0，这个时候表示这个是连接请求报文段，若对方同意建立连接，则响应报文中SYN=1，ACK=1；
6. **终止FIN**：用来释放一个连接，当FIN=1，表示此报文段的发送方数据已发送完毕，并且要求释放连接；
7. **窗口**：窗口值作为接收方让发送发设置其发送窗口的依据，之所以要有这个限制，是因为接收方的数据缓存空间是有限的。

TCP的三次握手

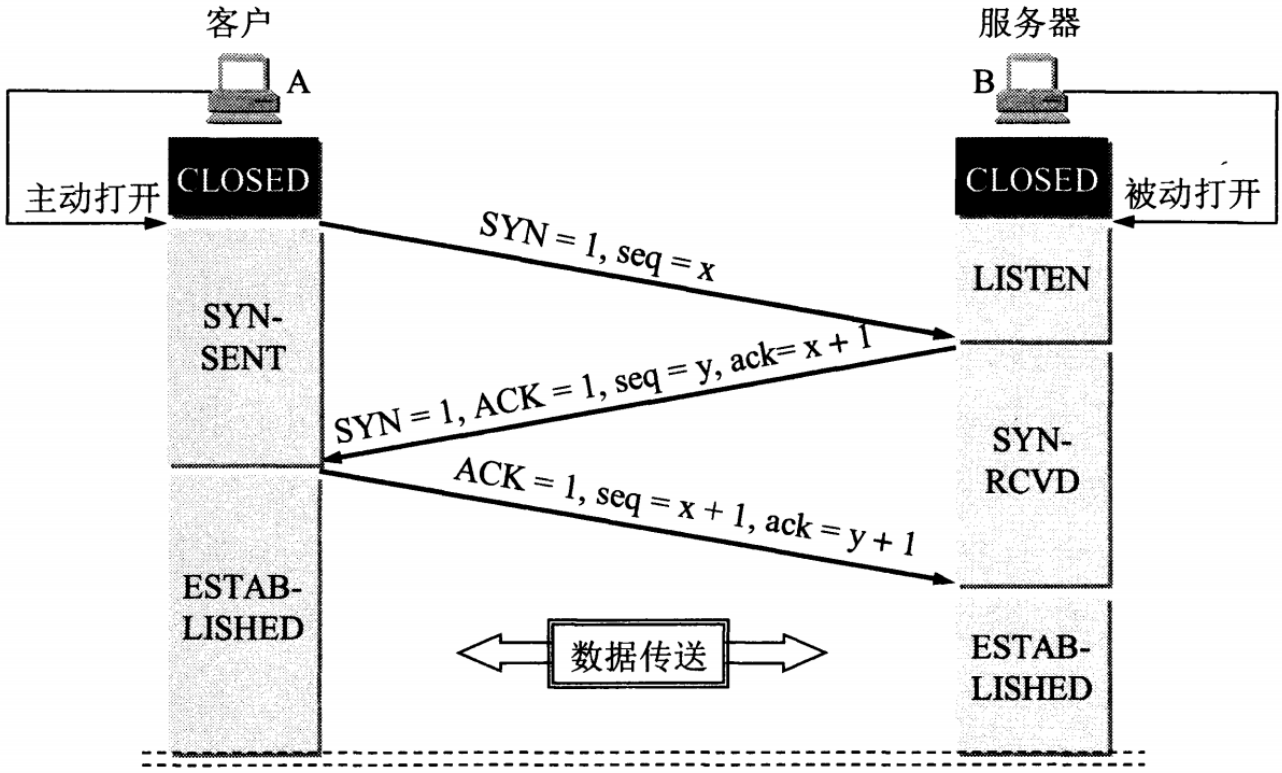

图 5-28 用三报文握手建立 TCP 连接

上图A是客户端，B是服务器端：

1. 首先B要处于LISTEN状态，等待客户的连接请求；
2. A想B发送连接请求报文，SYN=1，ACK=0，seq=x；
3. B收到连接请求报文，如果同意建立连接，则向A发送连接确认报文，SYN=1，ACK=1，确认号ack=x+1，同时选择一个自己的初始序号seq=y；
4. A收到B的连接确认报文后，还要向B发出确认，确认号为y+1，序号为x+1；
5. B收到A的确认连接后，连接建立；

三次握手的原因

第三次握手是为了防止失效的连接请求到达服务器，让服务器错误的打开连接；

客户端发送的请求如果在网络中滞留，那么就会隔很长时间才能收到服务器端发回的连接确认。客户端等待一个超时重传时间之后，就会重新请求连接，但是这个滞留的连接请求最后还是会到达服务器，如果不进行第三次握手，那么服务器就会打开两个连接，如果有第三次握手，客户端就会忽略服务器之后发送的对滞留连接请求的连接确认，不进行第三次握手，因此就不会再次打开连接；

TCP的四次挥手

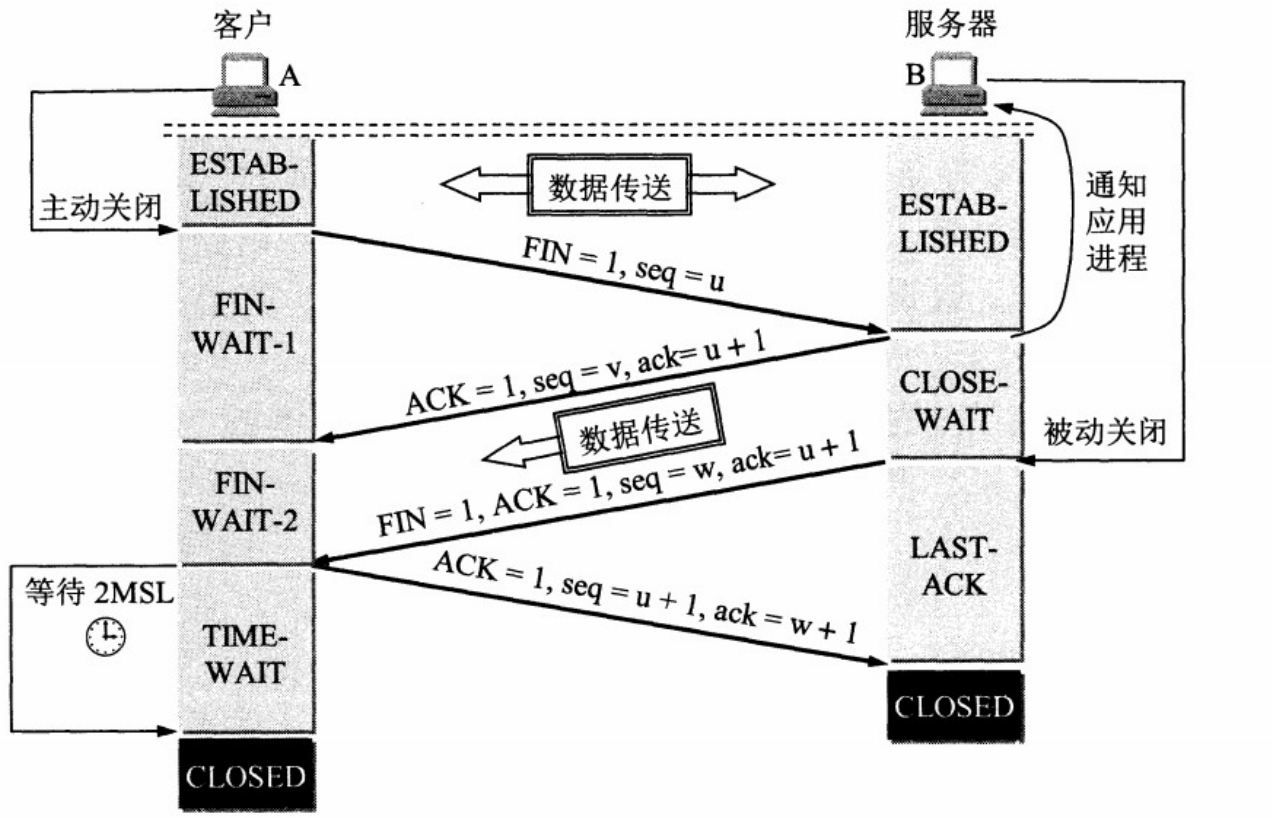

图 5-29　TCP 连接释放的过程

1. A发送连接释放报文，FIN=1，且将此时自己的报文序号seq=u发送过去；
2. B收到之后发送确认。确认号为ack=u+1此时TCP属于半关闭状态，B能向A发送数据但是A不能向B发送数据；
3. 当B不在需要连接时，发送连接释放报文，FIN=1，且将此时自己的序号seq = w发送给A；
4. A收到后发出确认，进入TIME-WAIT状态，等待2MSL后释放连接；
5. B收到A的确认后释放连接；

四次挥手的原因

客户端发送了FIN连接释放报文之后，服务器收到了这个报文，进入了CLOSE-WAIT状态。这个状态是为了让服务器端发送未发送完的数据，传送完毕之后，服务器会发送FIN连接释放报文；

TIME-WAIT 客户端接收到服务器端的 FIN 报文后进入此状态，此时并不是直接进入 CLOSED 状态，还需要等待一个时间计时器设置的时间 2MSL。这么做有两个理由：

确保最后一个确认报文能够到达。如果 B 没收到 A 发送来的确认报文，那么就会重新发送连接释放请求报文，A 等待一段时间就是为了处理这种情况的发生。

等待一段时间是为了让本连接持续时间内所产生的所有报文都从网络中消失，使得下一个新的连接不会出现旧的连接请求报文。

TCP可靠传输

TCP使用超时重传来实现可靠传输：如果一个已经发送的报文段在超时时间内没有收到确认，那么就重传这个报文段。

一个报文段从发送再到接收到确认所经过的时间称为往返时间 RTT，加权平均往返时间 RTTs 计算如下：

```
RTTs = (1 - a) * RTTs + a * RTT;
```

其中0<= a < 1,RTTs随着a的增加更容易收到RTT的影响； 超时时间RTO应该略大于RTTs，TCP使用的超时时间计算如下：

```
RTO = RTTs + 4 * RTTd;
```

其中RTTd为偏差的加权平均值；

TCP滑动窗口

发送方和接收方各有一个窗口，接收方通过 TCP 报文段中的窗口字段告诉发送方自己的窗口大小，发送方根据这个值和其它信息设置自己的窗口大小。

发送窗口内的字节都允许被发送，接收窗口内的字节都允许被接收。如果发送窗口左部的字节已经发送并且收到了确认，那么就将发送窗口向右滑动一定距离，直到左部第一个字节不是已发送并且已确认的状态；接收窗口的滑动类似，接收窗口左部字节已经发送确认并交付主机，就向右滑动接收窗口。

接收窗口只会对窗口内最后一个按序到达的字节进行确认，例如接收窗口已经收到的字节为 {31, 34, 35}，其中 {31} 按序到达，而 {34, 35} 就不是，因此只对字节 31 进行确认。发送方得到一个字节的确认之后，就知道这个字节之前的所有字节都已经被接收。

TCP流量控制

流量控制是为了控制发送方发送速率，保证接收方来得及接收。

接收方根据情况，发送给发送方的确认报文中的窗口字段用来控制发送方窗口大小，从而影响发送方的发送速率；将窗口字段设置为0，则发送方不能发送数据；

TCP拥塞控制

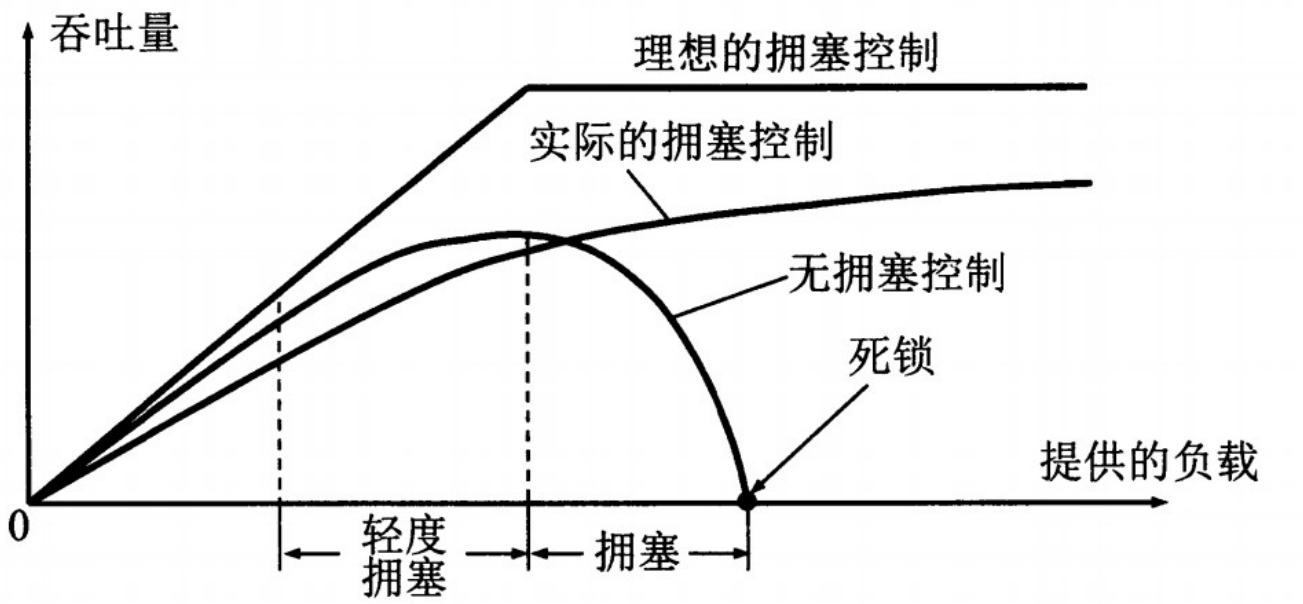

图 5-23 拥塞控制所起的作用

TCP通过四个算法来进行拥塞控制：慢开始，拥塞避免，快重传，快恢复。

发送方需要维持一个叫做拥塞窗口（cwnd）的状态变量，注意拥塞窗口和发送窗口的区别，拥塞窗口只是一个状态变量，实际决定发送方能发送多少数据的是发送方窗口；

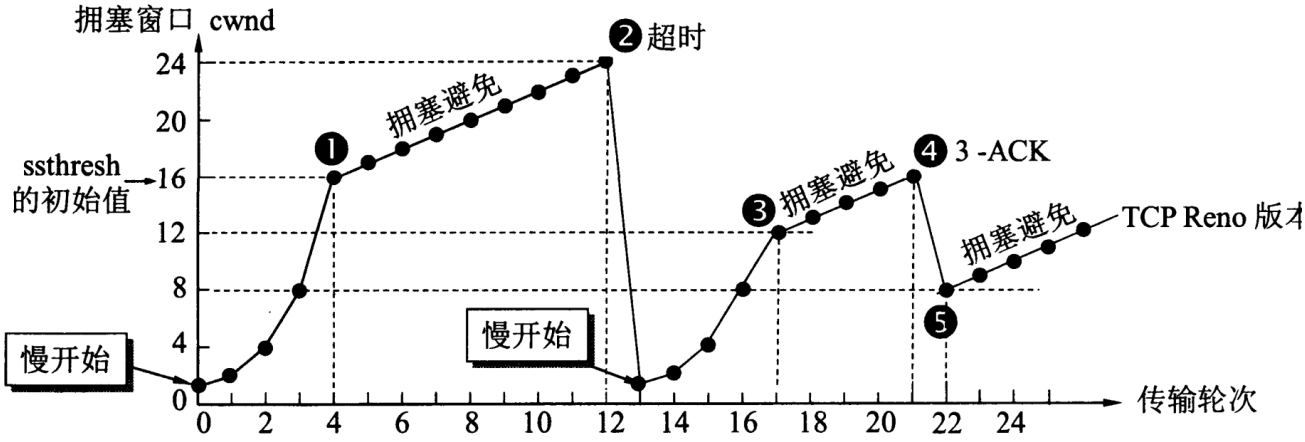

图 5-25 TCP 拥塞窗口 cwnd 在拥塞控制时的变化情况

1. **慢开始与拥塞避免** 发送的最初阶段执行慢开始策略，令cwnd = 1，发送方只能发送一个报文段；当收到确认后，将cwnd加倍，因此之后发送方能够发送的报文数量为，2，4，8……； 如果持续这样增长，拥塞可能性很高；这时候设置一个慢开始门限ssthresh，当cwnd >= ssthresh,进入拥塞避免阶段，每个轮次cwnd加1； 如果出现超时，则令ssthresh = cwnd / 2;然后再慢开始；

2. **快重传与快恢复** 在接收方，要求每次接受到的报文段都应该对最后一个已收到的有序报文段进行确认。例如已经接受到M1，M2，此时收到M4，应当对M2进行确认。

   在发送方，如果收到三个重复确认，那么可以知道下一个报文段丢失，此时执行快重传，立即重传下一个报文段，例如收到三个M2，则M3丢失，立即重传M3；

这个时候只是丢失个别报文段，不是网络拥塞，因此执行快恢复，令ssthresh = cwnd / 2, cwnd = ssthresh,注意此时直接进入拥塞避免；

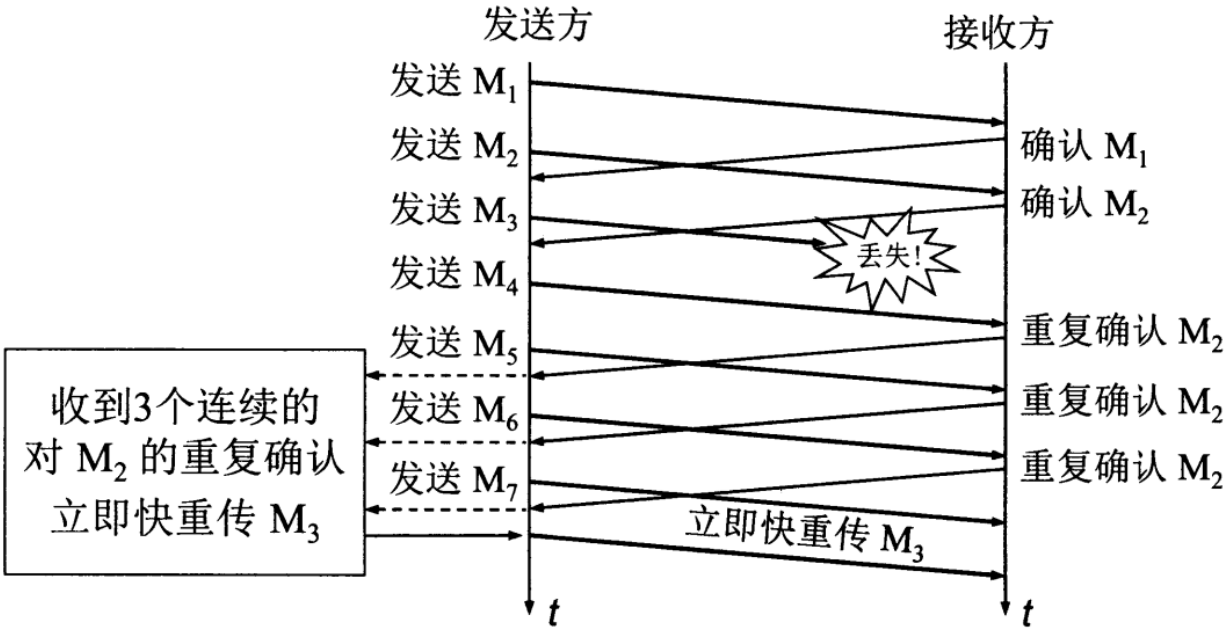

图 5-26　快重传的示意图