计算机网络

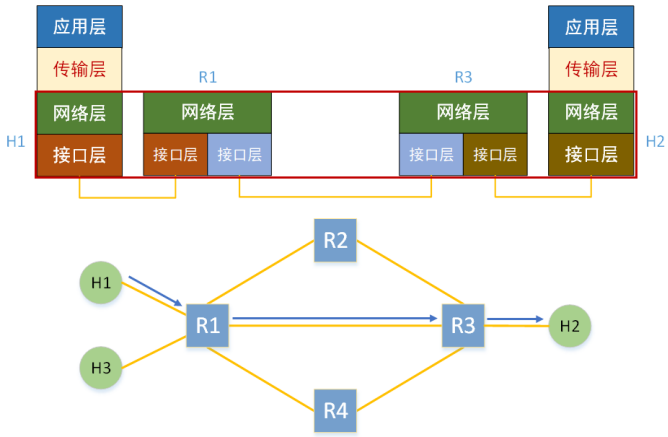
第三章 传输层协议

徐敬东 张建忠
xujd@nankai.edu.cn
zhangjz@nankai.edu.cn

计算机网络与信息安全研究室

内容提纲

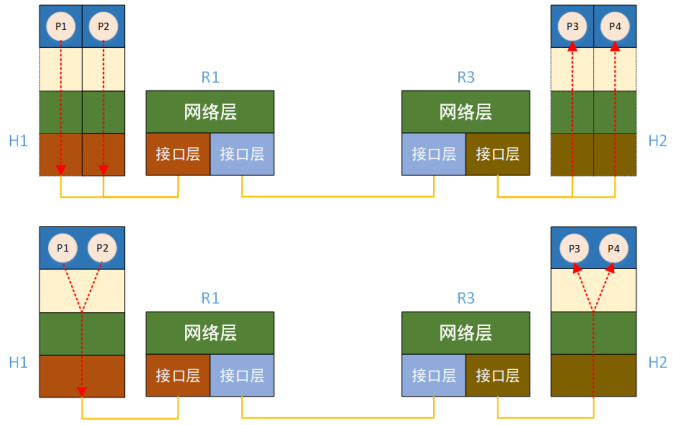
3.1 传输层需要解决的基本问题



- 网络层：将IP数据包从源主机传送到目的主机，提供无连接不可靠服务
- 数据包传输存在的问题：延迟、乱序、出错、丢失等
- 传输层解决的问题之一：可靠性，向应用层提供可靠服务

3.1 传输层需要解决的基本问题

- 应用层运行多个应用进程



- 共享单一的网络层协议（IP）和网络接口
- 传输层解决的问题之二：复用（Multiplexing）和分用（Demultiplexing）

3.2 TCP/IP体系结构中传输层协议与服务

- 传输层协议的基本功能
  - 复用和分用
  - 可靠性保证
- 传输层实体执行的动作
  - 发送端：将应用层的消息封装成传输层的数据单元，传递到网络层
  - 接收端：将从网络层接收的传输层数据单元，处理后交给应用层
- 传输控制协议TCP（Transport Control Protocol）
  - 为进程间通信提供面向连接的、可靠的传输服务
  - 实现复用分用、差错检测、确认重传、流量控制等传输层功能
- 用户数据报协议UDP（User Datagram Protocol）
  - 为进程间通信提供非连接的、不可靠的传输服务
  - 实现复用分用、差错检测等传输层功能

3.3 用户数据报协议UDP

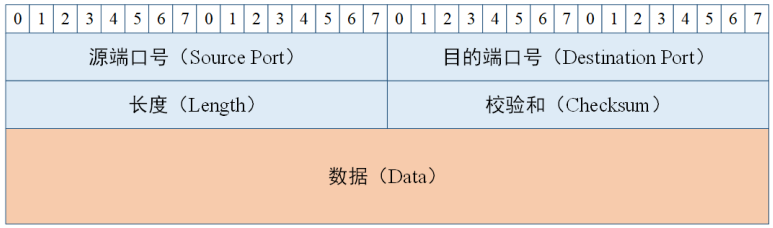
RFC 768

- UDP协议特点
  - 发送方和接收方不需要握手过程
  - 每个UDP数据单元（数据报）独立传输
  - 提供复用分用功能和可选的差错检测功能
  - 支持组播通信（点到多点通信）
  - 不提供可靠性保证：无确认重传、可能有出错、丢失、乱序等现象

3.3 用户数据报协议UDP

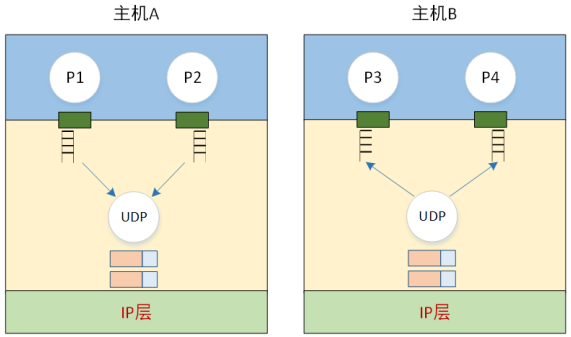
RFC 768

- UDP数据报格式
  - 长度：包含头部、以字节计数
  - 校验和：为可选项，用于差错检测



3.3 用户数据报协议UDP

- UDP的复用和分用

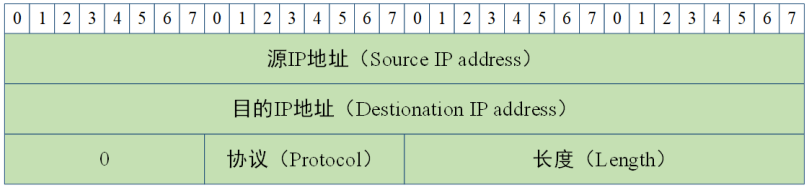


- 进程标识：目的IP地址+目的端口号
- 例如：P1与P3通信，P2与P4通信，P1使用端口6000， P2使用端口7000，P3使用端口6000， P4使用端口8000

3.3 用户数据报协议UDP

UDP数据报的差错检测

- 可选项，利用数据报中携带冗余位（校验和域段）来检测数据报传输过程中出现的差错
- 发送端：利用自己产生的伪首部和发送的UDP数据报计算校验和
- 接收端：利用自己产生的伪首部和接收的UDP数据报计算校验和
- 伪首部：包含源IP地址、目的IP地址、协议类型等域段



3.3 用户数据报协议UDP

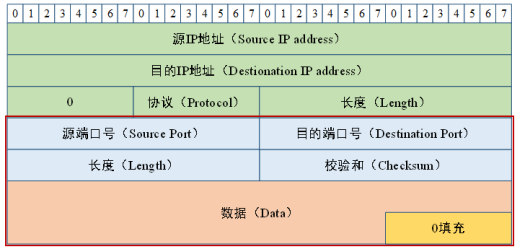
UDP校验和的计算方法

发送端：

- 产生伪首部，校验和域段清0，将数据报用0补齐为16位整数倍
- 将伪首部和数据报一起看成16位整数序列
- 进行 16 位二进制反码求和运算，计算结果取反写入校验和域段

接收端：

- 产生伪首部，将数据报用0补齐为16为整数倍
- 按16位整数序列，采用 16 位二进制反码求和运算
- 如果计算结果位全1，没有检测到错误；否则，说明数据报存在差错

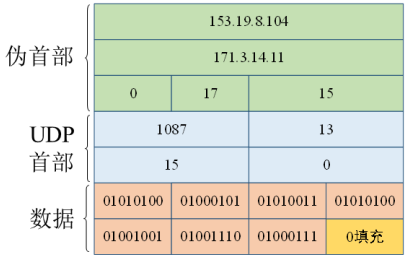


3.3 用户数据报协议UDP

TCP/IP校验和计算方法示例

3.3 用户数据报协议UDP

计算 UDP校验和示例



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（校验和）
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，将得出的结果求反码

10010110 11101101 → 求和得出的结果
01101001 00010010 → 校验和

3.3 用户数据报协议UDP

■ UDP校验和计算伪码

```
u_short cksum(u_short *buf, int count)
  {
    register u_long sum = 0;
     while (count--)
     {
         sum += *buf++;
         if (sum & 0XFFFF0000)
         {
             sum &= 0XFFFF;
             sum++;
         }
     }
      Return ~(sum & 0XFFFF);
  }
```


单选题 5分

假设要发送的数据D为11001010，采用校验和的计算方法，将D划分成两个4位二进制（1100和1010）进行运算，产生4位校验和，放在数据D后面一起传输，则传输的数据为：

A 110010100111

B 110010101000

C 110010100110

D 110010101001

3.3 用户数据报协议UDP

■ UDP校验和计算几点说明

- IPv4中UDP校验和是可选项，IPv6中变成强制项
  - 0无，非0有（如果计算结果为0，则以全1代替）
- UDP校验和覆盖的范围超出了UDP数据报本身，使用伪首部的目的是检验UDP数据报是否到达真正的目的地
  - 正确的目的地包括了特定的主机和该主机上特定的端口
- 伪首部不随用户数据报一起传输，接收方需自己形成伪首部进行校验
- 伪首部的使用破坏了层次划分的基本前提，即每一层的功能独立
  - 目的主机的IP地址UDP通常知道，源IP的使用需要通过路由选择决定

IP首部、ICMP、UDP、TCP都需要计算校验和，方法类似

3.3 用户数据报协议UDP

■ 使用UDP服务的应用

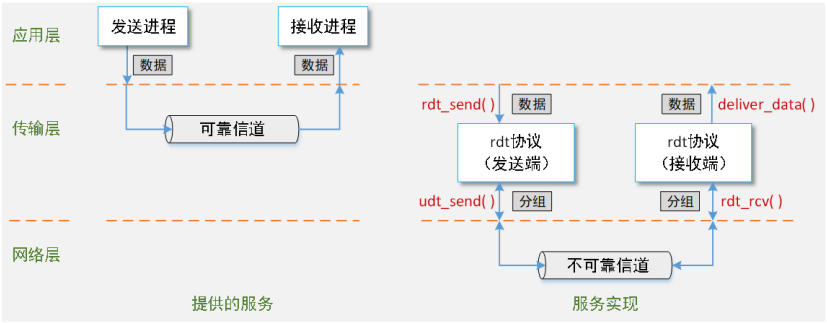
- 流媒体应用（实时音频和视频）通常使用UDP服务
  - 能够容忍一定的丢失
  - 对时延敏感
- 其他使用UDP服务的应用，如：
  - DNS
  - SNMP
- 需要在UDP之上实现可靠传输，即在应用层增加可靠机制

■ 为什么提供UDP服务？

- 不需要建立连接，建立连接需要增加延时，特别对于简单的交互应用
- 协议简单：在发送端和接收端不需要维护连接状态
- 数据报头部短，额外开销小
- 无拥塞控制

3.4 可靠数据传输

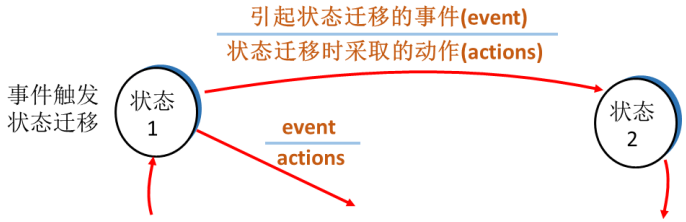
- 可靠传输基本原理
  - 可靠传输在多个层次有所涉及，如应用层、传输层、链路层等
  - 计算机网络中的Top 10问题



rdt：reliable data transfer protocol（可靠数据传输协议）

3.4 可靠数据传输

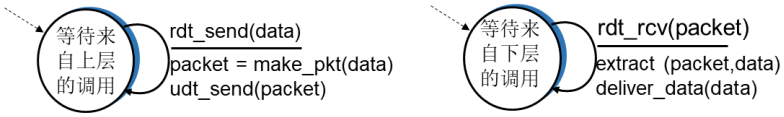
- 可靠传输基本原理
  - 考虑可靠协议的复杂性，采用渐进的方法
  - 考虑单向数据传输，控制信息可以双向传输
  - 使用有限状态机（FSM）描述发送端和接收端的状态和状态迁移



FSM：Finite State Machines

3.4 可靠数据传输

- 完全可靠通道上的可靠数据传输：rdt1.0
  - 下层通道是完全可靠的
    - 无位错误
    - 无分组丢失



发送端　　接收端

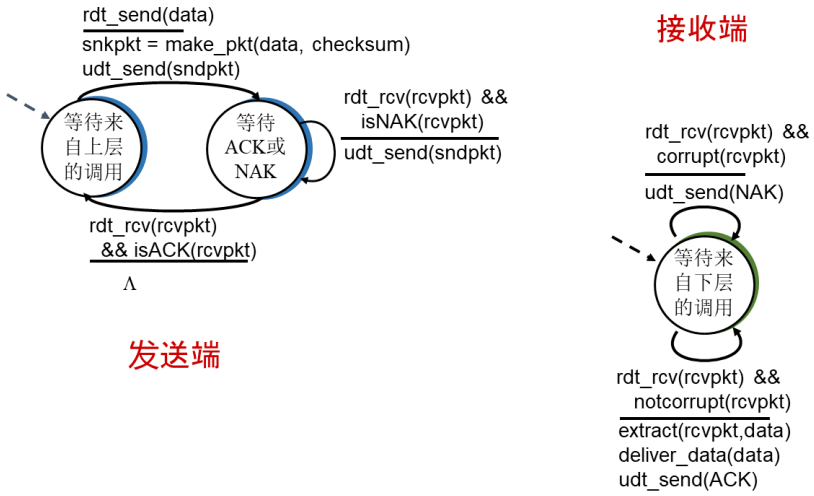
3.4 可靠数据传输

- 具有位错误通道上的可靠数据传输：rdt2.0
  - 下层通道可能造成某些位出现错误（如：1变0，0变1）
  - 需要解决的问题：如何恢复差错（自动重传请求ARQ）
    - ACK：接收端通知发送端分组正确接收
    - NAK：接收端通知发送端接收的分组存在错误
    - 发送端收到NAK，则重传分组
  - rdt2.0需要增加的功能
    - 差错检测
    - 接收端反馈：ACK或NAK（控制分组）
    - 发送端重传分组

ARQ：Automatic Repeat ReQuest

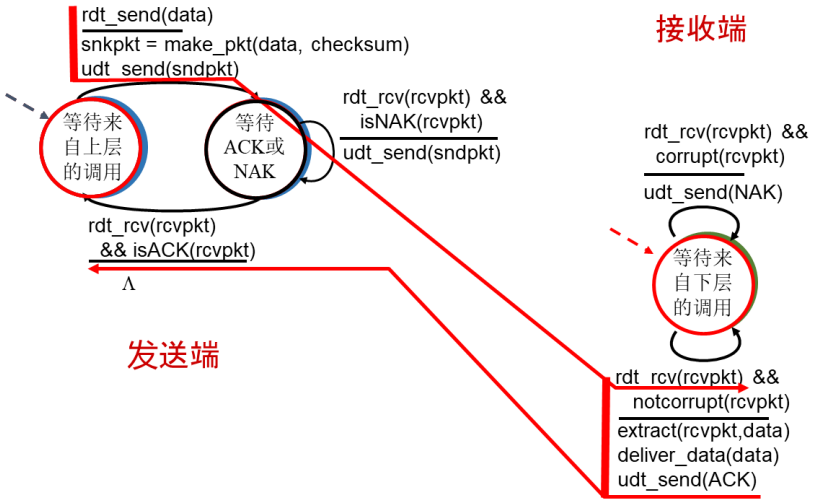
3.4 可靠数据传输

■ rdt2.0：有限状态机



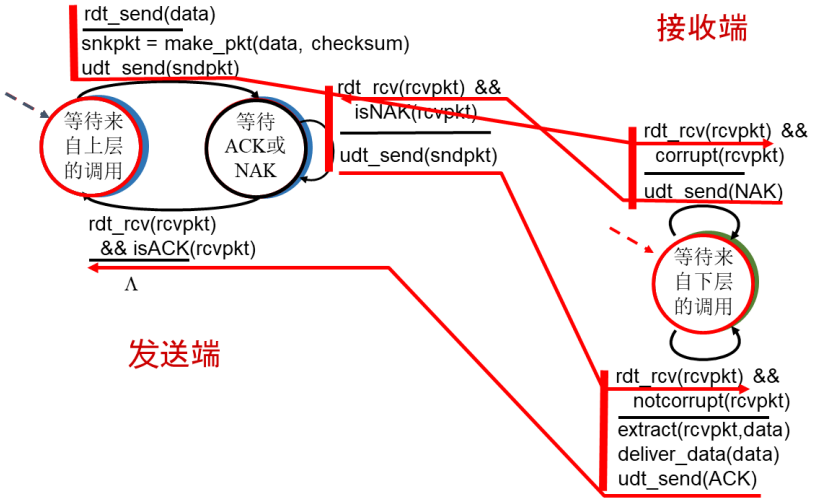
3.4 可靠数据传输

■ rdt2.0：无差错情况



3.4 可靠数据传输

■ rdt2.0：有差错情况



3.4 可靠数据传输

■ rdt2.0：存在的问题

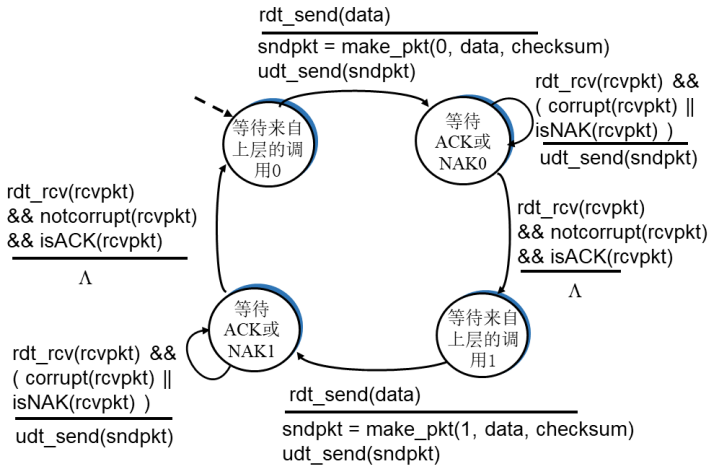
- 如果ACK/NAK受损会产生什么状况？
  - 发送端无法确认接收端的状况（ACK或NAK）
  - 不能简单进行重传：可能会造成重复接收
- 处理重复接收问题
  - 发送端在每个分组中增加序列号
  - 如果无法判断是ACK或NAK，则重传当前的分组
  - 接收端丢弃重复的分组

rdt2.1：解决rdt2.0的问题

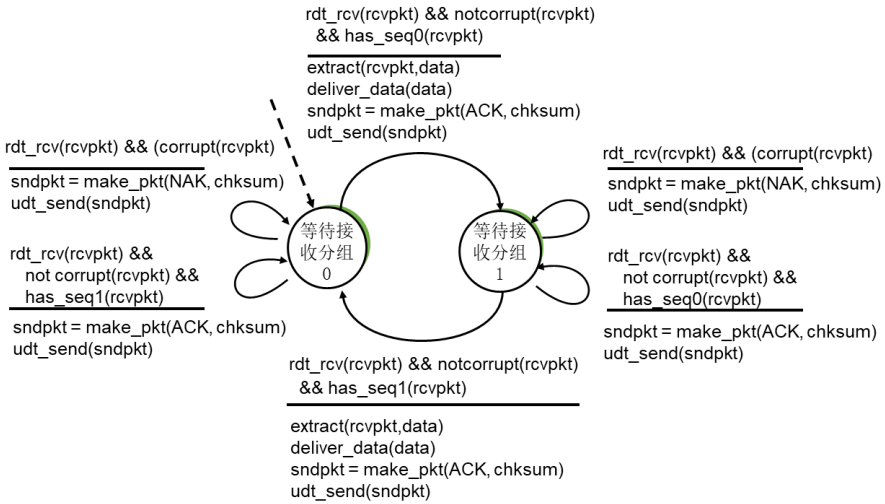
- 在rdt2.0基础上需要增加哪些功能？
  - 发送端在每个分组中增加序列号 使用几个序列号够用？
  - 发送端通过校验字段验证ACK/NAK分组是否损坏
  - 如果ACK/NAK分组损坏，发送端重传当前的分组
  - 接收端根据序列号判断是否是重复的分组
  - 接收端在ACK/NAK分组中增加校验字段

停等机制：发送端发送一个分组，然后等待接收端响应

rdt2.1：发送端状态机



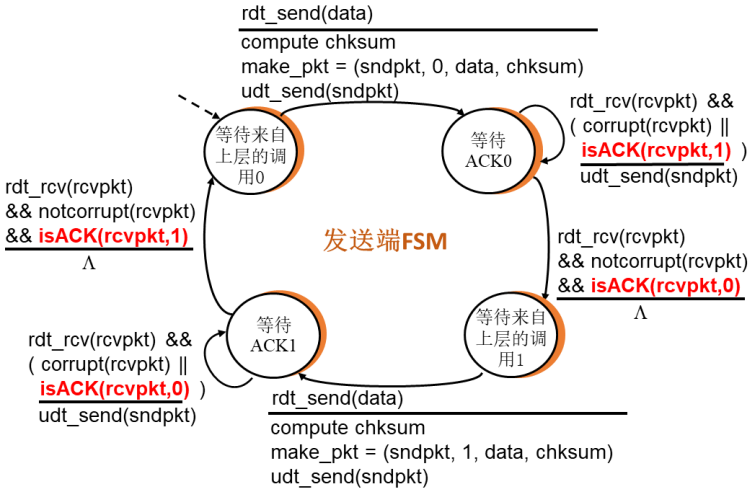
rdt2.1：接收端状态机



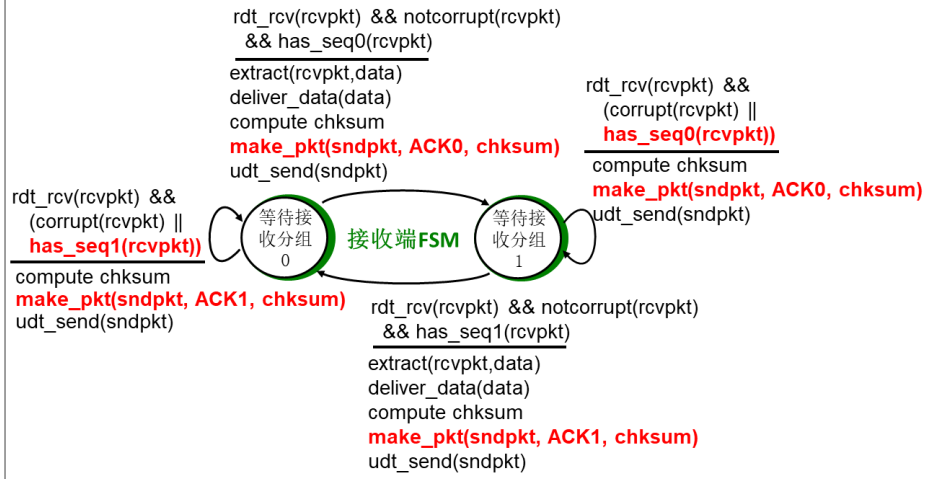
rdt2.2：对rdt2.1的改进

- 与rdt2.1功能相同，只使用ACK，不再使用NAK (NAK-free)
- 接收端通过发送对最后正确收到的分组的ACK代替NAK
  - ACK中必须携带所确认分组的序列号
- 发送端接收到重复的ACK，代表对当前分组的NAK，则
  - 重传当前的分组

■ rdt2.2：发送端状态机



■ rdt2.2：接收端状态机

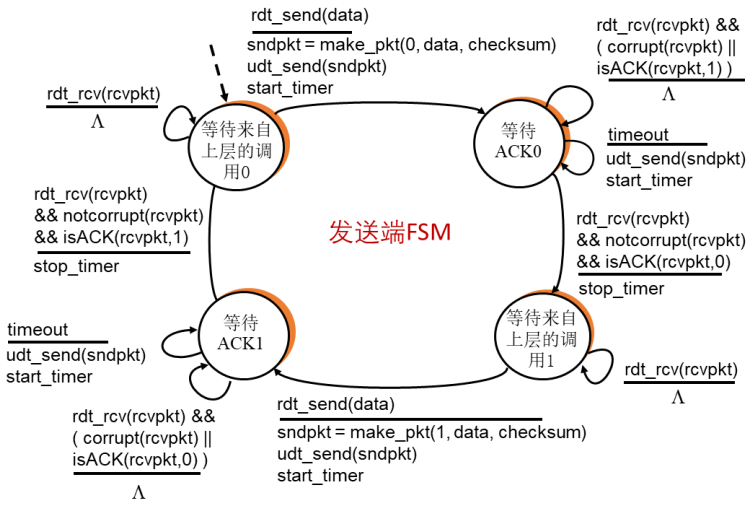


■ rdt3.0：通道既有差错又有丢失

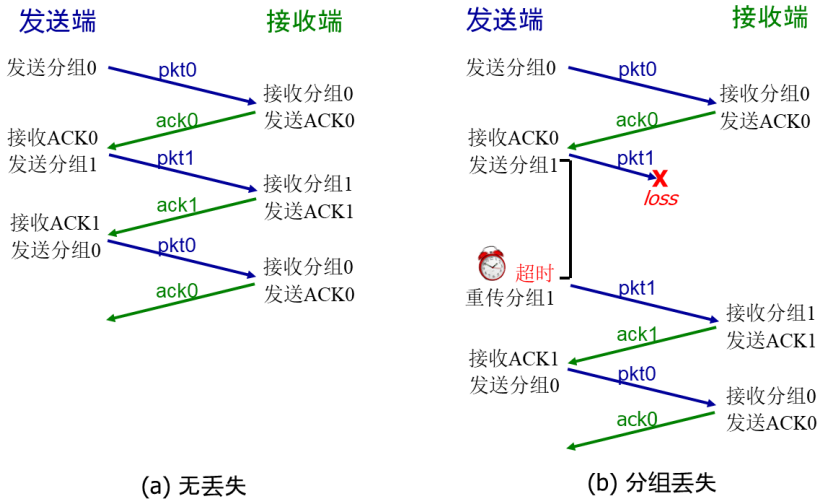
- **新的假设**：下层通道可能会有分组丢失（数据分组或ACK分组）
  - 如何检测丢失，当丢失发生时如何解决？
  - 前面的校验和、序列号、ACK、重传机制等不足以解决丢失检测问题
- **解决方法**：发送端等待一个合理的时间（需要一个定时器）
  - 如果未收到ACK，则重传当前的分组
  - 如果分组仅仅是被延迟，或是ACK丢失，会造成接收端重复接收
  - 接收端需要根据序列号判断重复的分组，并丢弃

合理时间：多长时间合理？过长和过短有什么问题？

■ rdt3.0：发送端状态机



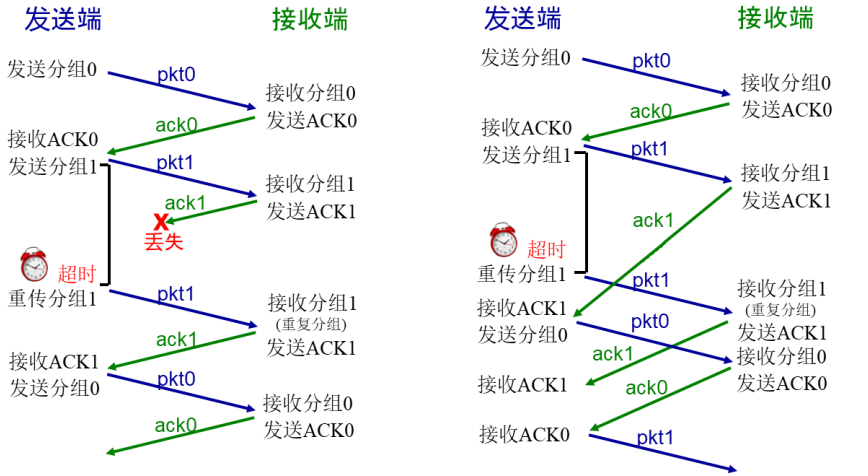
■ rdt3.0：场景示例



(a) 无丢失　　(b) 分组丢失

■ rdt3.0：场景示例

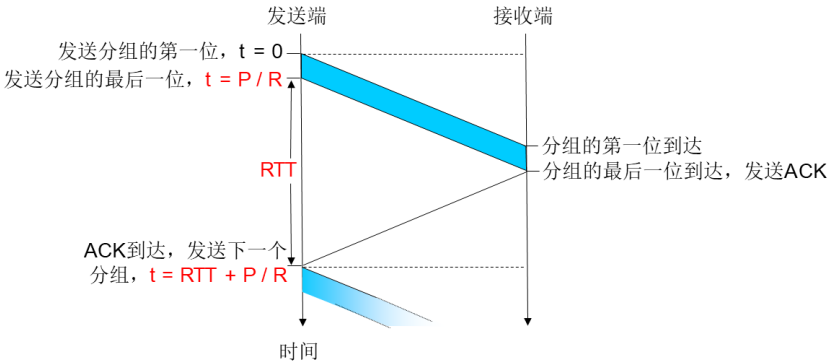
rdt 3.0可以解决失序问题吗？



(c) ACK丢失　　(d) 提前超时

■ rdt3.0：性能问题（停等协议）

- rdt3.0可以实现可靠机制，但存在性能问题



■ rdt3.0：性能问题（停等协议）

- 例如：R=1Gbps (10^9 bps) 的链路，往返延时RTT=30毫秒，分组长P=1000字节 (8000位)，发送一个分组所用的时间$TRANSP$为：

$$TRANSP = \frac{P}{R} = \frac{8000\ bit}{10^9\ bit/s} = 8\mu s$$

- 发送端的链路利用率（忽略发送ACK的时间）

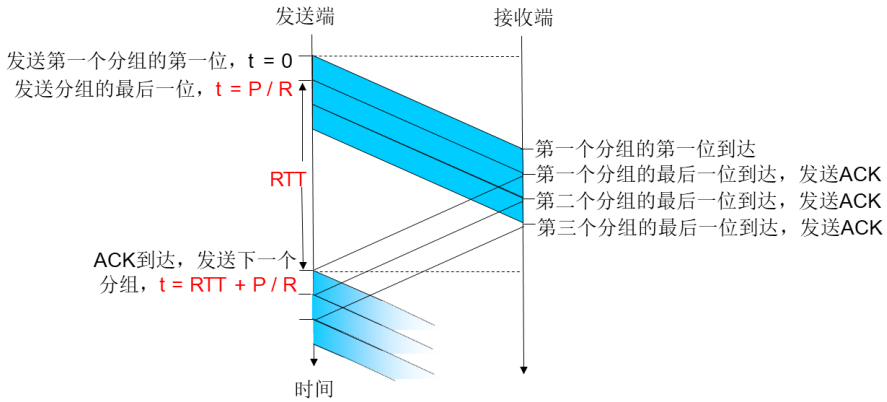
$$U_{sender} = \frac{P/R}{RTT + P/R} = \frac{0.008}{30 + 0.008} \approx 0.00027$$

- 大约每30毫秒发送一个分组，1Gbps链路上的吞吐率约为264Kbps

如何提高链路利用率？Rdt 3.0如何改进？

3.4 可靠数据传输

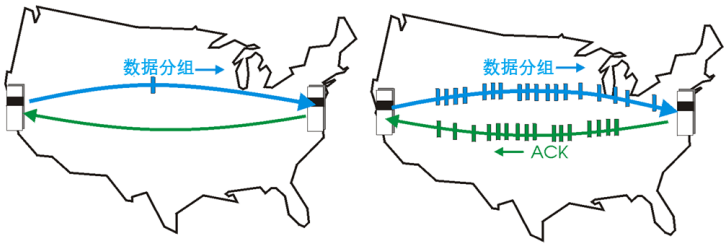
- rdt3.0：性能优化



$$U_{sender} = \frac{3P/R}{RTT + P/R} = \frac{0.008 \times 3}{30 + 0.008} \approx 0.00081$$

3.4 可靠数据传输

- 流水线协议
  - 流水线机制：在确认未返回之前允许发送多个分组
    - 0和1两个序列号还够用吗？
    - 发送端和接收端缓冲区需要增加吗？



  - 两种典型的流水线协议：
    - 回退N：*Go-Back-N (GBN)*
    - 选择重传：*Selective Repeat (SR)*