


TCP/IP 之 可靠数据传输原理



六尺帐篷 (/u/f8e9b1c246f1)

2017.05.24 18:22 字数 2746 阅读 274 评论 1 喜欢 13

(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/12835308/notes/12717319)

可靠数据传输对于应用层、传输层、链路层都很重要，是网络领域的Top10问题。对于传输层来说，由于相邻的网络层是不可靠的，所以要在传输层实现可靠数据传输（rdt）就比较复杂。

那么我们来了，究竟怎样才是可靠？

我们将讨论一下几个方面的内容

□信道的(不可靠)特性

□可靠数据传输的需求

□Rdt 1.0

□Rdt 2.0, rdt 2.1, rdt 2.2

□Rdt 3.0

□流水线与滑动窗口协议

□GBN

□SR

什么是可靠？

- 不错
就是传输的数据包没有错误
- 不丢
传输的数据包不丢失
- 不乱
传输的数据包顺序要保持正确

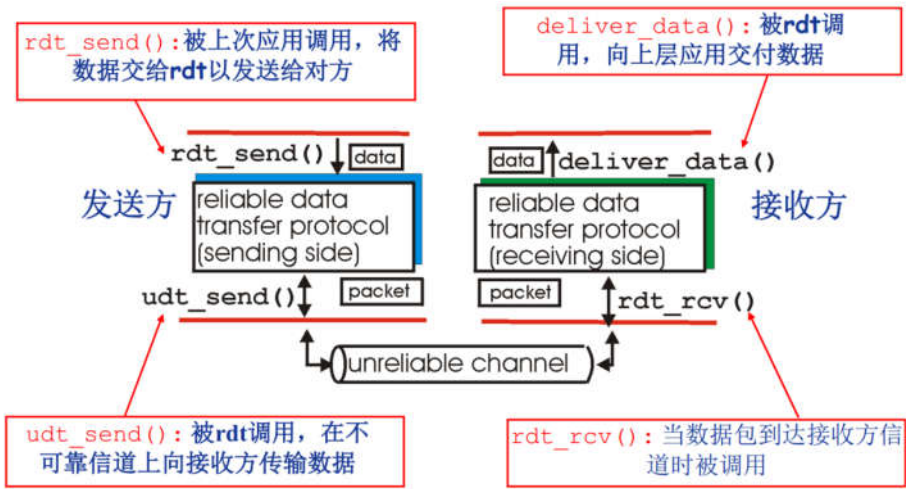


image.png

为了更好的说明，我们采取渐进式的设计可靠数据传输的发送方和接收方。

我们考虑第一个版本的可靠数据传输

Rdt 1.0: 可靠信道上的可靠数据传输

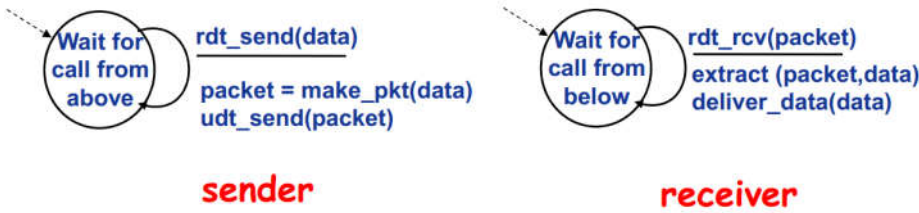
假设

底层信道完全可靠

• 不会发生错误(bit error)

• 不会丢弃分组

显然有了这个假设的话，发送方和接收方只要能正确接收数据就可以了，所以他们是相互独立的，因为发过来的数据保证一定是正确的。



Rdt 2.0: 产生位错误的信道

我们假设底层信道可能翻转分组中的位(bit)

首先如何判断错误，我们可以利用校验和来判断是否发生位错误

那么发现了错误，我们该如何处理呢？

第一种思路当然是纠正错误，但是这样实现的难度和代价都比较大，在计算机网络中，我们一般都会采取第二种思路

第二种思路就是直接重传，如果我们发现了错误，很自然，那我们就重传一次，直到接受方收到正确的分组。

还有一个问题就是假设接收方发现了错误，如果告知发送方已经发生了错误呢？

其实处理起来也很简单，就是向接收方发送一个信号，代表出现错误，如果没错误就发送一个信号，表示没错误。

如何从错误中恢复？

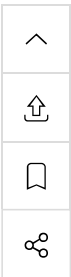
- 确认机制(Acknowledgements, ACK): 接收方显式地告知发送方分组已正确接收
- NAK:接收方显式地告知发送方分组有错误
- 发送方收到NAK后，重传分组
- 基于这种重传机制的rdt协议称为ARQ(Automatic Repeat reQuest)协议

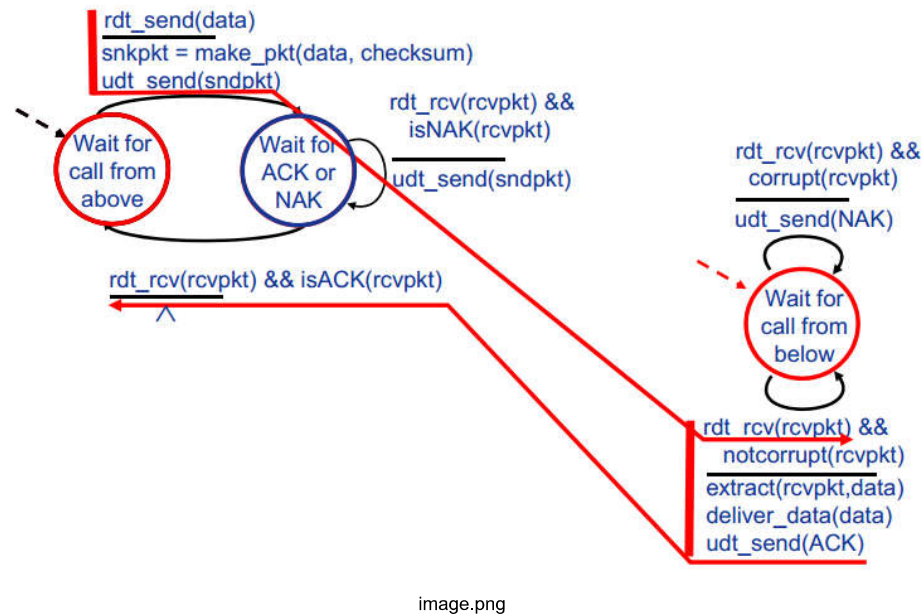
Rdt 2.0中引入的新机制

- 差错检测
- 接收方反馈控制消息: ACK/NAK
- 重传

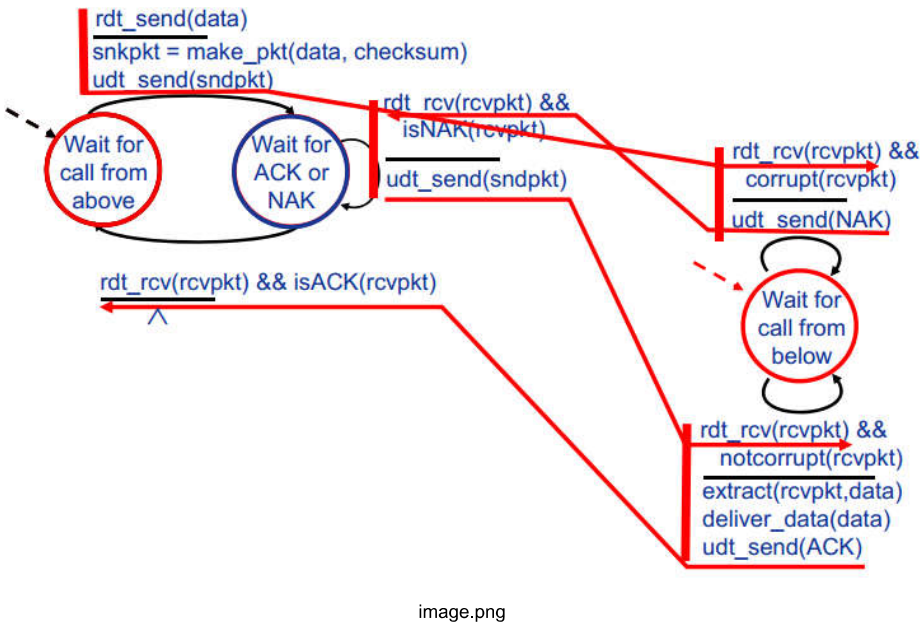
下面两个图分别模拟了有错误和无错误场景：

无错误场景





有错误场景



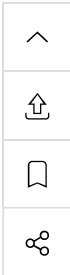
Rdt 2.1: 发送方, 应对ACK/NAK破坏

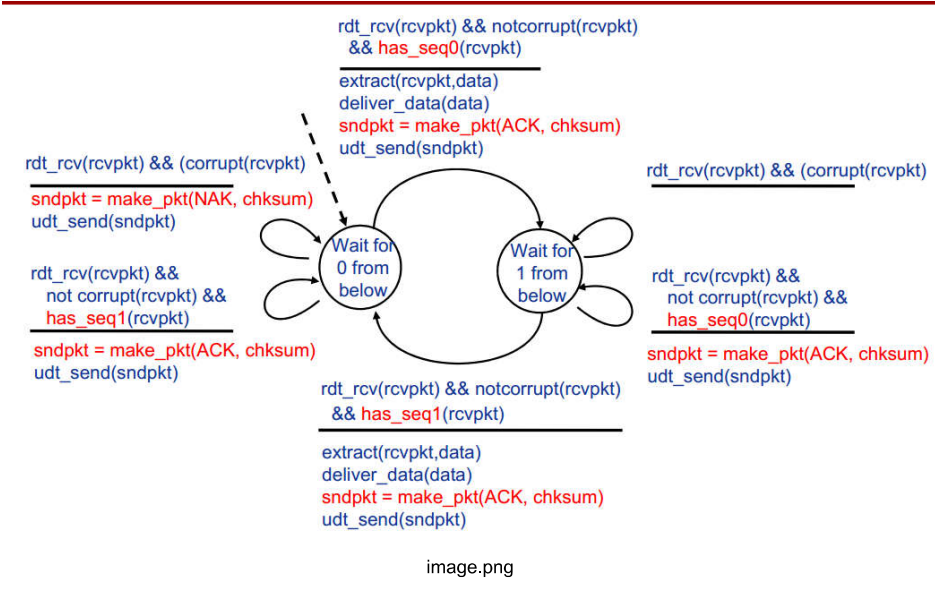
我们看rdt2.0有什么问题, 我们知道确认信号也需要通过信道传播, 那么如果ack, nck的信号发生了错误呢? 发送方应该怎么处理?

显然发生了错误, 我们就应该重传

但是这里, 又有一个问题, 接收方怎么知道发送方这次新传过来的是新的报文段还是因为ack出错而重传的报文段呢? 显然我们需要区分, 上一个报文段和当前的报文段, 我们给报文段编写好序号就可以了, 而且只需要0, 1两个序号, 一个表示上次的报文段, 一个表示新接受的。

这样接收方如果收到0, 就知道这次不是新的报文段, 可能是上次ack出错了, 发送方无法确认, 就重传了上次的报文段, 所以接收方需要丢掉这个报文段, 然后再次传一次ack确认信号, 如果收到的是序号为1的报文段, 则接收方直接接受就可以了。





Rdt 2.1 vs. Rdt 2.0

发送方：

- 为每个分组增加了序列号
- 两个序列号(0, 1)就够用，为什么？
- 需校验ACK/NAK消息是否发生错误
- 状态数量翻倍
- 状态必须“记住”“当前”的分组序列号

接收方：

- 需判断分组是否是重复
- 当前所处状态提供了期望收到分组的序列号
- 注意：接收方无法知道ACK/NAK是否被发送方正确收到

Rdt 2.2: 无NAK消息协议

我们考虑一下我们真的需要两个确认信号ack和nack么？

- ☐ 与rdt 2.1功能相同，但是只使用ACK

如何实现？

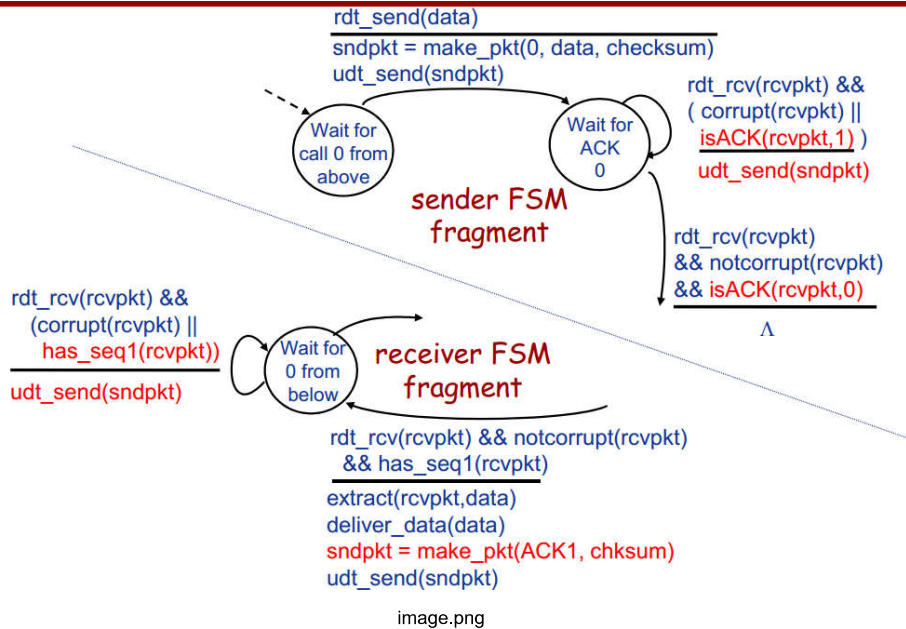
- ☐ 接收方通过ACK告知最后一个被正确接收的分组
- ☐ 在ACK消息中显式地加入被确认分组的序列号
- ☐ 发送方收到重复ACK之后，采取与收到NAK消息相同的动作
- ☐ 重传当前分组

^

📄

🔖

🔗



Rdt 3.0

到rdt2.2为止，我们基本解决了“不错”的要求，即报文和确认信息在信道上发生了错误的话，我们都可以很好的解决，解决的方法其实就是重传

那么我们接下来就该解决不丢的问题。

如果信道既可能发生错误，也可能丢失分组，怎么办？

“校验和 + 序列号 + ACK + 重传”够用吗？

显然是不够用的

我们假设这时候ack不是错误而是直接丢了，那么发送方就会无限制的等着接收方的ack，同时接收方也会无限制的等着发送方的新报文。

这样就陷入了类似死锁的机制，如果不加以处理，那么网络就卡死在这里了。

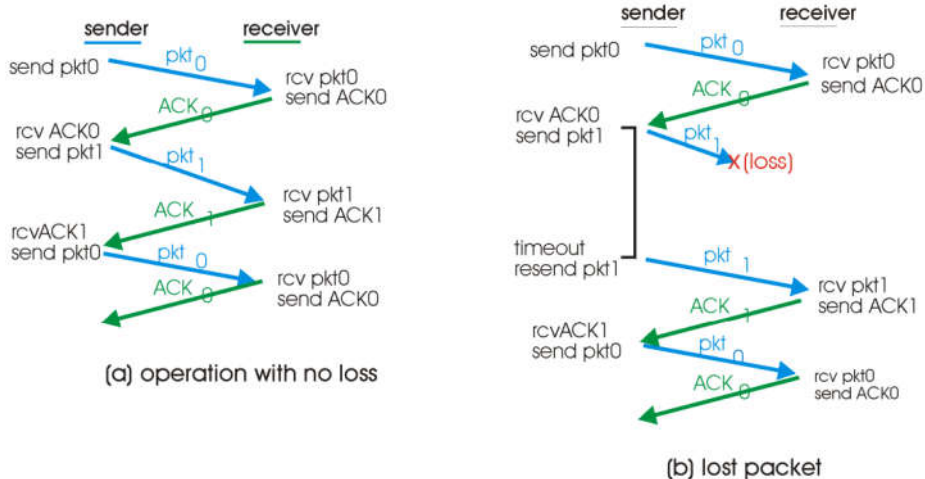
那么我们该如何处理呢？

方法：发送方等待“合理”时间

- ☐ 如果没收到ACK，重传
- ☐ 如果分组或ACK只是延迟而不是丢了

- 重传会产生重复，序列号机制能够处理
- 接收方需在ACK中显式告知所确认的分组

☐ 需要定时器



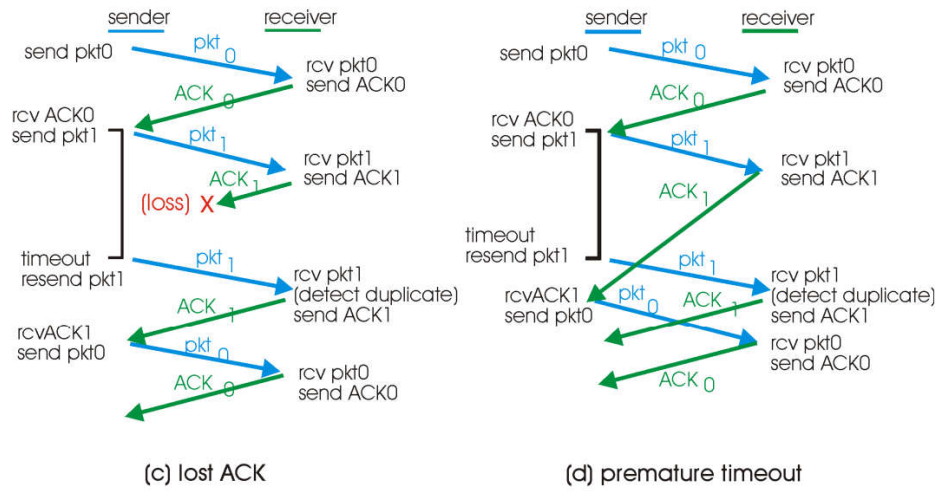


image.png

rdt3.0效率

- Rdt 3.0能够正确工作，但性能很差
- 示例：1Gbps链路，15ms端到端传播延迟，1KB分组

$$T_{\text{transmit}} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8\text{kb/pkt}}{10^9 \text{ b/sec}} = 8 \text{ microsec}$$

image.png

- 发送方利用率：发送方发送时间百分比

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

image.png

- 在1Gbps链路上每30毫秒才发送一个分组 □ 33KB/sec
- 网络协议限制了物理资源的利用

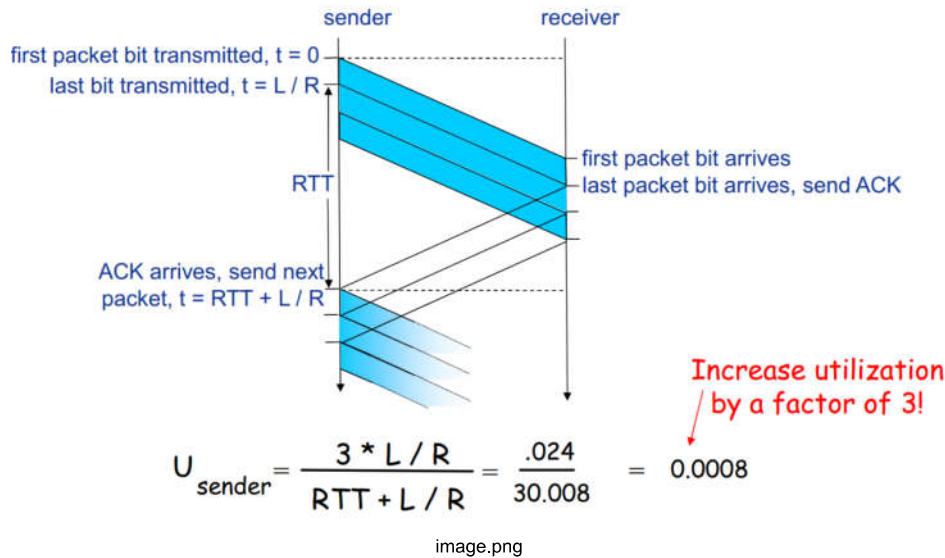
这样低效率的原因是，我们采取的是停-等操作

就是说发送方发了一个数据包，就停下来了，直到得到来自接收方的确认才发送第二个，这样就造成了很多的空余时间都在空闲等待。

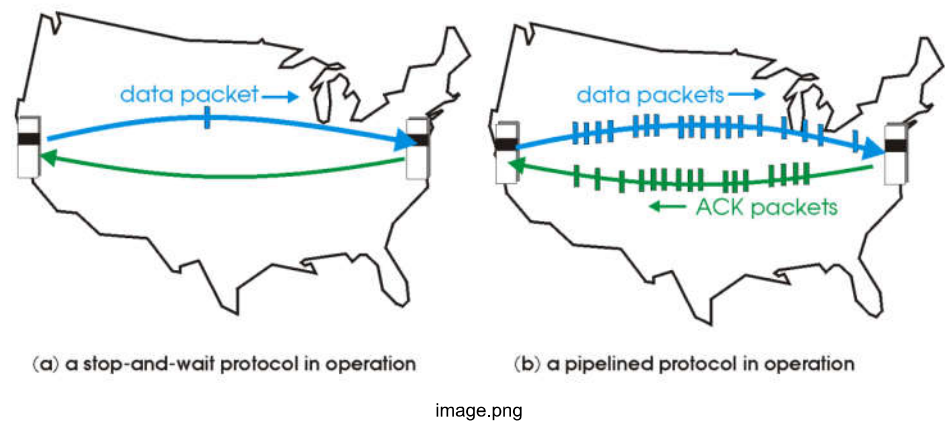
流水线机制与滑动窗口协议

为了改进停等机制所造成的效率低下，我们可以采用流水线的机制，一次发送多条报文段，充分利用空闲的时间

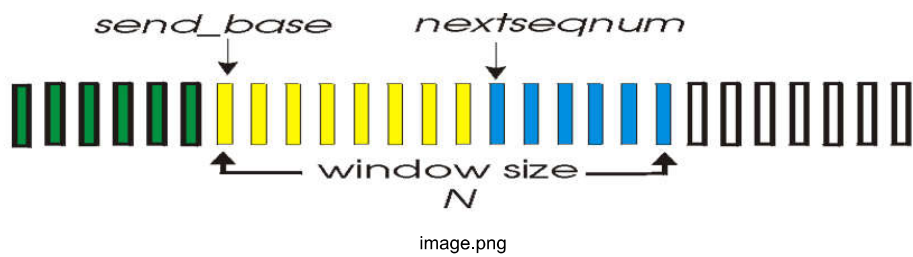




- 允许发送方在收到ACK之前连续发送多个分组
- 更大的序列号范围
 - 发送方和/或接收方需要更大的存储空间以缓存分组



进一步的，我们采用滑动窗口协议，顾名思义，就是发送给定窗口大小的报文数，随着报文被接收确认，同时窗口可以动态的向前滑动



- 滑动窗口协议: Sliding-window protocol
- 窗口
- 允许使用的序列号范围
- 窗口尺寸为N: 最多有N个等待确认的消息
- 滑动窗口
- 随着协议的运行，窗口在序列号空间内向前滑动
- 滑动窗口协议: GBN, SR

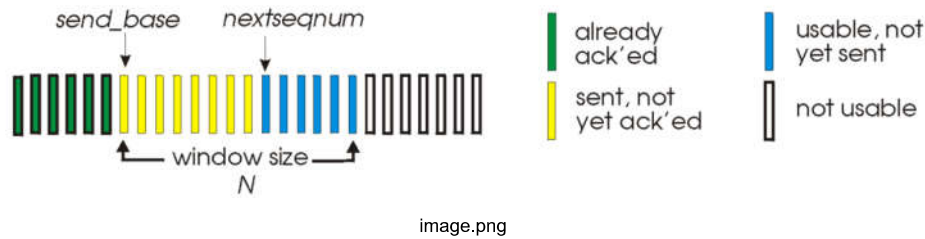
Go-Back-N(GBN)协议

^

⌂

🔖

🔗



如图所示，窗口大小N,最多允许N个分组未确认。

ACK(n): 确认到序列号n(包含n)的分组均已被正确接收

□ 可能收到重复ACK

为没收到确认的分组设置计时器(timer)

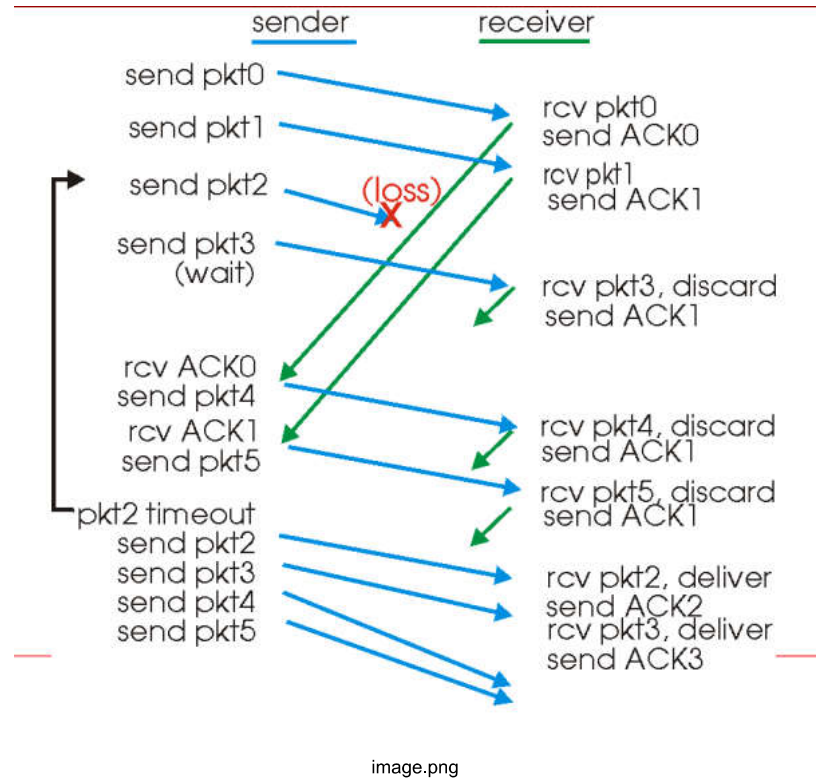
□超时Timeout(n)事件: 重传序列号大于等于n，还未收到ACK的所有分组

ACK机制: 发送拥有最高序列号的、已被正确接收的分组的ACK

□ 可能产生重复ACK

□ 只需要记住唯一的expectedseqnum

接收方是没有缓存的，所以接收方对于乱序到达的分组直接丢弃，并且重新发送目前为止接收到的分组中序列号最大的按序到达的分组



简单的习题：

□ 数据链路层采用后退N帧（GBN）协议，发送方已经发送了编号为0~7的帧。当计时器超时时，若发送方只收到0、2、3号帧的确认，则发送方需要重发的帧数是多少？分别是那几个帧？

□ 解：根据GBN协议工作原理，GBN协议的确认是累积确认，所以此时发送端需要重发的帧数是4个，依次分别是4、5、6、7号帧

Selective Repeat协议

GBN有什么缺陷？

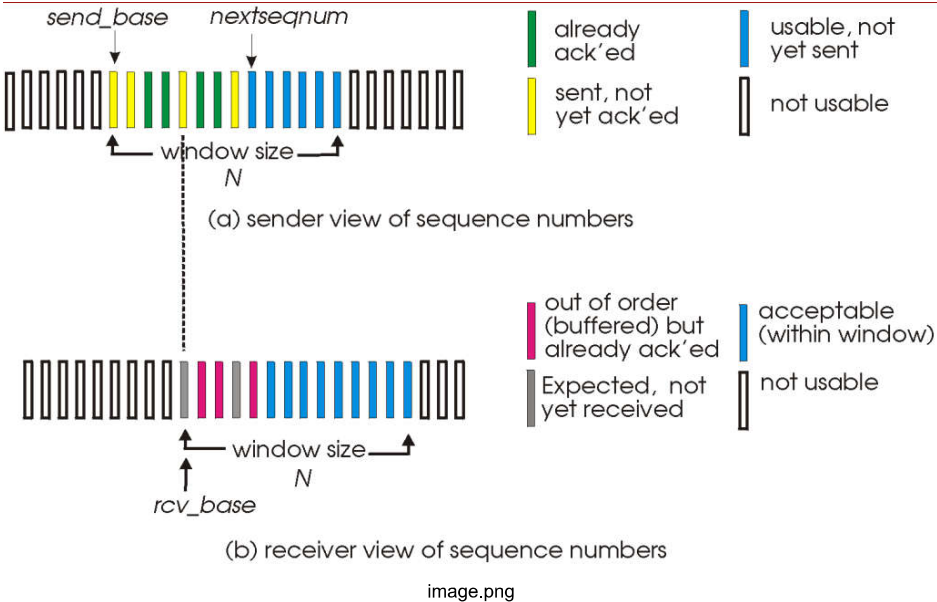
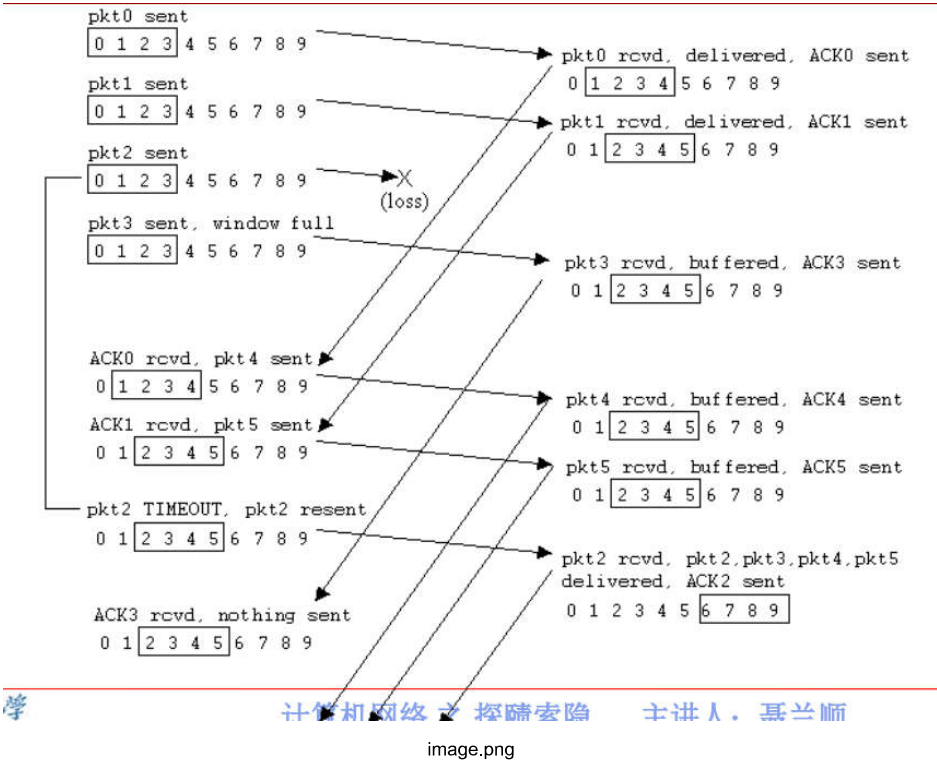
由于GBN接收方没有缓存，对于非按序的分组直接丢弃，就会造成很多到达的分组由于顺序乱了，却白发了，需要再次重新发送。

显然为了提高效率，我们可以在接收方设置缓存，对于未按序达到的分组，先存起来，而不是直接丢弃。

这就是选择重复协议的思想

□接收方对每个分组单独进行确认

- 设置缓存机制，缓存乱序到达的分组
- 发送方只重传那些没收到ACK的分组
- 为每个分组设置定时器
- 发送方窗口
- N个连续的序列号
- 限制已发送且未确认的分组



从图中我们可以看到，接收方是动态移动滑动窗口的，只有当窗口部分前面的全部正确接受并确认了，才向前移动。

可靠数据传输原理与协议回顾

- 信道的(不可靠)特性
- 可靠数据传输的需求
- Rdt 1.0
- Rdt 2.0, rdt 2.1, rdt 2.2
- Rdt 3.0

^

⌂

🔖

🔗

- 流水线与滑动窗口协议
- GBN
- SR

📖 网络 (/nb/12835308)

© 著作权归作者所有



六尺帐篷 (/u/f8e9b1c246f1)

写了 245418 字, 被 15240 人关注, 获得了 1429 个喜欢

(/u/f8e9b1c246f1)

 喜欢

13








更多分享


(<http://cwb.assets.jianshu.io/notes/images/127173>)


被以下专题收入, 发现更多相似内容


⚙ 投稿管理


+ 收入我的专题

- 

Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)
- 

Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)
- 

计算机网络 (/c/89e56c7637e1?utm_source=desktop&utm_medium=notes-included-collection)
- 

服务器学习 (/c/7a465e79bc92?utm_source=desktop&utm_medium=notes-included-collection)
- 

iOS Dev... (/c/3233d1a249ca?utm_source=desktop&utm_medium=notes-included-collection)
- 

程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)

