

传输层

可靠传输协议 RDT(reliable data transfer)

版本

rdt1.0

rdt1.0是基于理想情况下的协议，假设所有信道都是可靠的，没有比特位的翻转，没有数据包的丢失与超时，所以rdt1.0的传输功能就是发送方发送数据，接收方接受数据。

rdt2.0

rdt2.0在rdt1.0的基础上解决了比特位翻转的问题，这里的比特位翻转发生在运输层下面的不可信信道包中的1可能会变0，0可能会变成1。rdt2.0增加了3种新机制：

- 错误检验

在运输层对应用层的数据进行打包处理时，新增checksum（校验和）。

- 接收者反馈接受信息（ACK,NAK）

接收端可以对其数据包进行检验，如果正确，返回ACK，发送者继续发送下一个数据包；如果不正确，返回NAK，发送者重传数据。

- 重传机制

发送者重传数据。

rdt2.1

但是rdt2.0有着一个致命的缺点，只考虑了发送方到接收方的数据传输，如果反馈信息ACK，NAK传输时发生比特位翻转会出现什么情况？如果ACK发生翻转，那么发送方会再次重复的发送相同的数据包；如果NAK发生翻转，那么发送方会认为数据传输情况很好，但是接收方却已经收到了一个错误的数据包。

由此rdt2.1，在rdt2.0的基础之上，发送方在打包数据包时添加了0或者1编号，同样ACK,NAK字段上也添加了0，1字段，表示0,1号字段的确认或者否定。发送方就有了2种状态发送0号数据包，1号数据包，接收方也有了2种状态等待0号数据包和等待1号数据包。

现在假设情景发送方向接收方发送0号数据包，如果接收方接收到0号数据包，返回ACK，但是ACK出现翻转，接收方处于等待1号数据状态，发送方重复发送0号数据，接收方会拒绝0号数据，避免重复。如果接收方接收到0号数据包出现错误，返回NAK，但是NAK出现翻转，接收方处于等待0号数据状态，发送方继续发送1号数据，接收方会拒绝1号数据，避免错序。

rdt2.2

rdt2.2是在rdt2.1上的基础之上做了小小的改善，摒弃了NAK，只需采用ACK。我们在ACK的信息上加上了期望的序号号。

现在假设情景发送方向接收方发送0号数据包，如果接收方接收到0号数据包，返回（ACK，1），发送方接着发送1号数据包。如果接收方接收到0号数据包出现错误，返回（ACK，0），发送方重传0号数据包。

rdt3.0

rdt2.2之前的版本都重在处理数据包的比特位翻转情况，却没有考虑到数据包在传输过程中出现的数据包丢失问题，**这样数据包丢失会使得网络处于拥塞状态。**

rdt3.0在rdt2.2的基础之上处理了数据包丢失的情况，增加了的机制，如果在RTT时间段内，发送方没有接收到反馈信息，那么发送方默认数据包已经丢失了，会**自动重传**。

在rdt3.0的阶段，这时的主要问题是性能问题，rdt3.0的性能很差。主要原因在RTT的时间段内，网络处于空闲状态，而RTT时间段比较长，使用率变得十分的低。

因此在此基础上增加了 **流水线协议** 改进 rdt3.0。

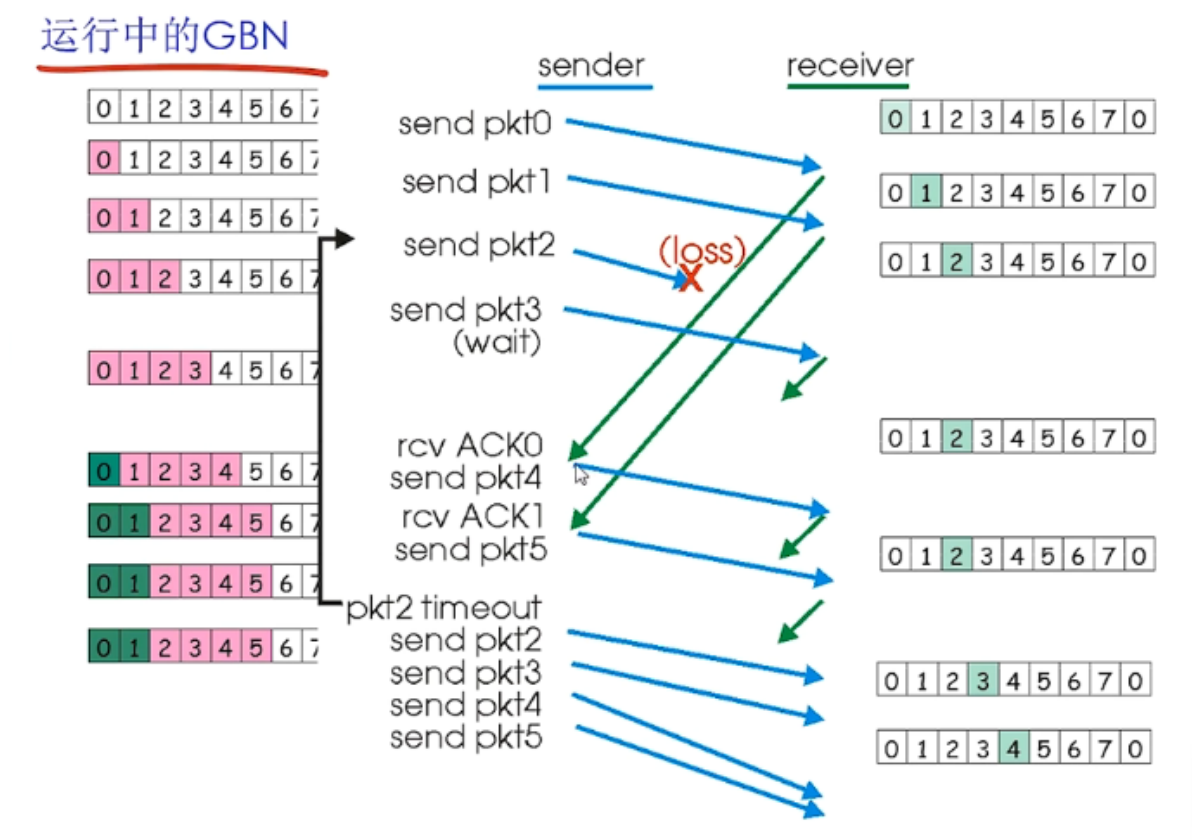
流水线协议

GO-BACK-N(回退N步协议)

发送窗口 >1 ，接收窗口 $=1$ ；并且采取累计确认的方式返回ACK；

回退N协议的名字由来就是超时重传来的，如果发生超时重传事件，发送方重传所有的已发送但未收到确认的帧。

对于接收方，如果中间n号帧丢失了，接收方就会一直等待，这时再次收到后边的帧会直接丢弃，只想等着丢失的帧到来，接收方会返回一个n的确认帧，代表n以及n之前的都已经收到了，这时发送方会从n+1号帧开始在发送一遍后边所有的帧（很专一）。



selective repeat(选择重传协议)

- GBN的弊端

累积确认导致批量重传，所以想要有一种协议可以只重传出错的帧；解决方法是：设置单个确认，同时加大接收窗口，设计接收缓存，缓存乱序到达的帧。这种机制就是选择重传协议(SR协议)。

- 重点

- 每个帧都有自己的定时器，一个超时事件发生后只重传一个帧
- 对数据帧逐一确认，收一个确认一个
- 只重传出错帧
- 接收方有缓存

- SR发送方必须相应的三件事

- 上层的调用

从上层网络层收到数据之后，SR发送方检查下一个可用于该帧的序号，如果序号位于发送窗口内，则发送数据帧；否则就像GBN一样，要么将数据缓存，要么返回给上层之后再传输。

- 收到了一个ACK

如果收到ACK，假如该帧序号在窗口内，则SR发送方将那个被确认的帧标记为已接收。如果该帧序号是窗口的下界(最左边第一个窗口对应的序号)，则窗口向前移动到具有最小序号的未确认帧处。如果窗口移动了并且有序号在窗口内的未发送帧，则发送这些帧。

- 超时事件

每个帧都有自己的定时器，一个超时事件发生后只重传一个帧。

- SR接收方必须相应的三件事

- SR接收方将确认一个正确接收的帧而不管其是否按序。失序的帧将被缓存，并返回给发送方一个该帧的确认帧【收到谁确认谁】，直到所有帧（即序号更小的帧）皆被收到为止，这时才可以将一批帧按序交付给上层网络层，然后向前移动滑动窗口。

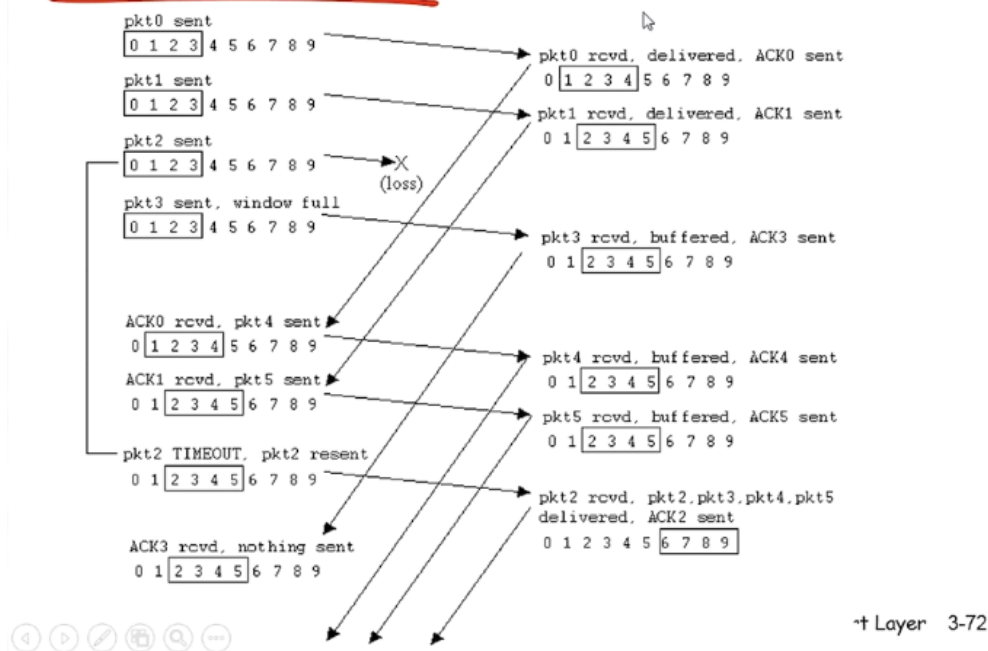
- SR接收方如果收到了窗口序号之外并且小于窗口下界的帧，则直接返回一个ACK。其他情况忽略该帧。

- 滑动窗口的长度

- 发送窗口最好等于接收窗口，发送窗口大了会发生溢出，发送窗口小了没有意义。
 - 窗口过大会产生二异性的问题，即接收方无法判断当前发送方发送的帧是某个序号下新的的帧还是旧的帧。

- 计算公式如下： $WT_{max} = WR_{max} = 2n - 1$

选择重传SR的运行



对比

对比GBN和SR

	GBN	SR
优点	简单，所需资源少（接收方一个缓存单元）	出错时，重传一个代价小
缺点	一旦出错，回退N步代价大	复杂，所需要资源多（接收方多个缓存单元）

□ 适用范围

- 出错率低：比较适合GBN，出错非常罕见，没有必要用复杂的SR，为罕见的事件做日常的准备和复杂处理
- 链路容量大（延迟大、带宽大）：比较适合SR而不是GBN，一点出错代价太大

可靠传输的保证详解

校验和

校验和是UDP与TCP都具备的；

发送端将package分成多个16位的二进制数组，最低位+1，然后回卷进位，将最终所有的和加起来，作为检验和的值；发送到接收端后接收端重复同样的步骤后，对比两个校验和的值是否一致，不一致则丢弃该包并返回NAK；

停等协议

滑动窗口 (slide window) 协议

- 定义

传输的每个部分被分配唯一的连续序列号，接收方使用数字并以正确的顺序放置接收到的数据包，丢弃重复的数据包并识别丢失的数据。

协议中规定，对于窗口内未经确认的分组需要重传。这种分组数量最多可以等于发送窗口的大小，即滑动窗口的大小 n 减去 1（因为发送窗口不可能大于 $(n-1)$ ，起码接收窗口要大于等于 1）。

- 应用实例

1. 停止等待协议 (stop-and-wait)

发送方和接收方的窗口都等于 1。

发送方这时自然发送每次只能发送一个，并且必须等待这个数据包的 ACK，才能发送下一个。

2. 回退 n 步协议 (GO-BACK-N)

发送方 > 1 接收方 = 1

3. 选择重传协议 (selective repeat)

发送方 > 1 接收方 > 1

- 发送缓冲区

- 形式：内存中的一个区域，落入缓冲区的分组可以发送
- 功能：用于存放已发送，但是没有得到确认的分组
- 必要性：需要时可以重传

- 接收窗口 (receiving window) = 接受缓冲区

- 接收窗口用于控制哪些分组可以接受：
 - 只有收到的分组序号落入接收窗口才可以接受
 - 若序号在接收窗口之外，则丢弃
- 接收窗口 $W_r = 1$ ，则只能顺序接受
- 接收窗口 $W_r > 1$ ，则可以乱序接受
 - 提交给上层的分组，要按序

- 接收窗口的滑动和发送确认

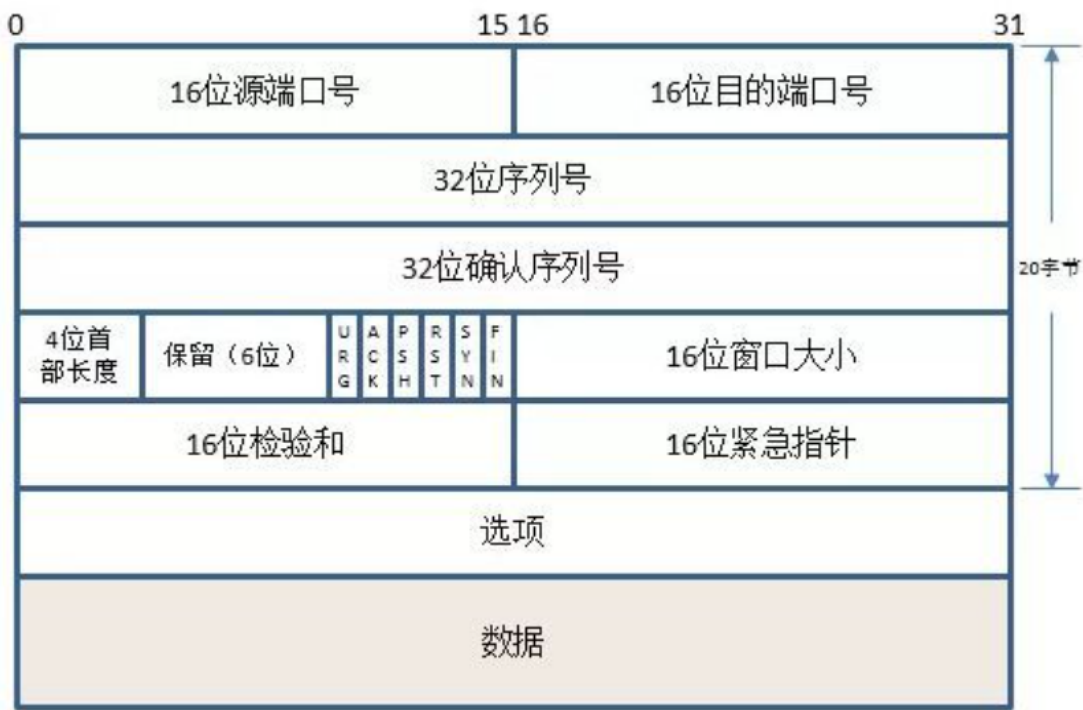
- 滑动
 - 低序号的分组到来，接受窗口滑动
 - 高序号的分组到来，缓存但不交付（因为要实现 rdt，不允许失序），不滑动
- 发送确认
 - 接收窗口尺寸 = 1：发送方连续收到的最大的分组确认（累计确认）
 - 接收窗口尺寸 > 1 ：收到分组，发送那个分组的确认（非累计确认）

TCP详解

特性

- 点对点
 - 一个发送方，一个接收方
- 可靠的、按顺序的字节流
 - 没有报文边界
- 管道化（流水线）
 - TCP拥塞控制和流量控制设置窗口大小
- 发送和接收缓存
- 全双工数据
 - 在同一连接中数据流双向流动
 - [MSS: 最大报文段大小](#)
- 面向连接
 - 在数据交换之前，通过握手（交换控制报文）初始化发送方、接收方的状态变量
- 有流量控制
 - 发送方不会淹没接受方

TCP头部结构



1. **16位端口号:** 告知主机该报文段来自哪里（源端口）以及传给哪个上层协议或应用程序（目的端口）的。进行tcp通信时，客户端通常使用系统自动选择的临时端口号，而服务器则使用知名服务端端口号。

2. **32位序号**：一次tcp通信过程中某一个传输方向上的字节流的每个字节的编号。假设主机A和主机B进行tcp通信，A发送给B的第一个tcp报文段中，序号值被系统初始化为某个随机值ISN。那么在该传输方向上（从A到B），后续的tcp报文段中序号值将被系统设置成ISN加上该报文段所携带数据的第一个在整个字节流中的偏移。例如，某个tcp报文段传送的数据时字节流中的第1025~2048字节，那么该报文段的序号值就是ISN+1025。另一个传输方向（从B到A）的tcp报文段的序号值也具有相同的含义。
3. **32位确认号**：用作对另一方发送来的tcp报文段的相应。其值是收到的tcp报文段的序号值加1。假设主机A和主机B进行tcp通信，那么A发送出的tcp报文段不仅携带自己的序号，而且包含对B发送来的tcp报文段的确认号。反之，B发送出的tcp报文段也同时携带自己的序号和对A发送来的报文的确认号。
4. **4位头部长度的**：标识该tcp头部有多少个32bit字（4字节）因为4位最大能表示15，**所以tcp头部最长是60字节。**
5. **6位标志位**（即图中的保留6位）：标志位有如下几项
 - URG标志，表示紧急指针是否有效
 - ACK标志，表示确认号是否有效。称携带ACK标志的tcp报文段位确认报文段
 - PSH标志，提示接收端应用程序应该立即从tcp接受缓冲区中读走数据，为接受后续数据腾出空间（如果应用程序不将接收的数据读走，它们就会一直停留在tcp缓冲区中）
 - RST标志，表示要求对方重新建立连接。携带RST标志的tcp报文段为复位报文段。
 - SYN标志，表示请求建立一个连接。携带SYN标志的tcp报文段为同步报文段。
 - FIN标志，表示通知对方本端要关闭连接了。携带FIN标志的tcp报文段为结束报文段。
6. **16位窗口大小**：是tcp流量控制的一个手段。这里说的窗口，指的是接收通告窗口。它告诉对方本端的tcp接收缓冲区还能容纳多少字节的数据，这样对方就可以控制发送数据的速度。
7. **16位校验和**：由发送端填充，接收端对tcp报文段执行CRC算法以校验tcp报文段在传输过程中是否损坏。注意，这个校验不仅包括tcp头部，也包括数据部分。这也是tcp可靠传输的一个重要保障。
8. **16位紧急指针**：是一个正的偏移量。它和序号字段的值相加表示最后一个紧急数据的下一个字节的序号。因此，确切的说，这个字段是紧急指针相对当前序列号的偏移，称为紧急偏移。tcp的紧急指针是发送端向接收端发送紧急数据的方法。
9. **选项**：最多为40个字节

TCP的ACK

产生TCP ACK的建议

产生TCP ACK的建议 [RFC 1122, RFC 2581]

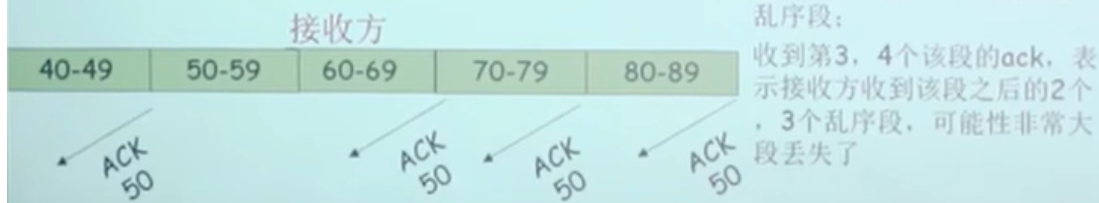
接收方的事件	TCP接收方动作
所期望序号的报文段按序到达。 所有在期望序号之前的数据都已经被确认	延迟的ACK。对另一个按序报文段的到达最多等待500ms。如果下一个报文段在这个时间间隔内没有到达，则发送一个ACK。
有期望序号的报文段到达。 另一个按序报文段等待发送ACK	立即发送单个累积ACK，以确认两个按序报文段。
比期望序号大的报文段乱序到达。 检测出数据流中的间隔	立即发送重复的ACK，指明下一个期待字节的序号
能部分或完全填充接收数据间隔的报文段到达。	若该报文段起始于间隔（gap）的低端，则立即发送ACK。

Transport Layer 3-90

快速重传机制

快速重传

- ❑ 超时周期往往太长：
 - 在重传丢失报文段之前的延时太长
- ❑ 通过重复的ACK来检测报文段丢失
 - 发送方通常连续发送大量报文段
 - 如果报文段丢失，通常会引起多个重复的ACK
- ❑ 如果发送方收到同一数据的3个冗余ACK，重传最小序号的段：
 - 快速重传：在定时器过时之前重发报文段
 - 它假设跟在被确认的数据后面的数据丢失了
 - 第一个ACK是正常的；
 - 收到第二个该段的ACK，表示接收方收到一个该段后的乱序段；



TCP连接管理

TCP的三次握手

TCP的四次挥手

为什么需要三次握手和四次挥手？

三次握手和四次挥手会存在什么问题？

TCP的拥塞控制

端到端的拥塞控制，路由器是不会向端系统提供辅助的信息的（ATM中核心网络会提供辅助信息）。端系统根据自身得到的信息判断是否发生拥塞，从而采取动作。

拥塞感知

TCP 拥塞控制：拥塞感知

Ack=3460

发送端如何探测到拥塞？

1460B 1460B 1460B 1460B

- ❑ 某个段超时了（丢失事件）：**拥塞**
 - 超时时间到，某个段的确认没有来
 - 原因1：网络拥塞（某个路由器缓冲区没空间了，被丢弃）概率大
 - 原因2：出错被丢弃了（各级错误，没有通过校验，被丢弃）概率小
 - 一旦超时，就认为拥塞了，有一定误判，但是总体控制方向是对的
- ❑ 有关某个段的3次重复ACK：**轻微拥塞**
 - 段的第1个ack，正常，确认绿段，期待红段
 - 段的第2个重复ack，意味着红段的后一段收到了，蓝段乱序到达
 - 段的第2、3、4个ack重复，意味着红段的后第2、3、4个段收到了，橙段乱序到达，同时红段丢失的可能性很大（后面3个段都到了，红段都没到）
 - 网络这时还能够进行一定程度的传输，拥塞但情况要比第一种好

Transport Layer 3-123

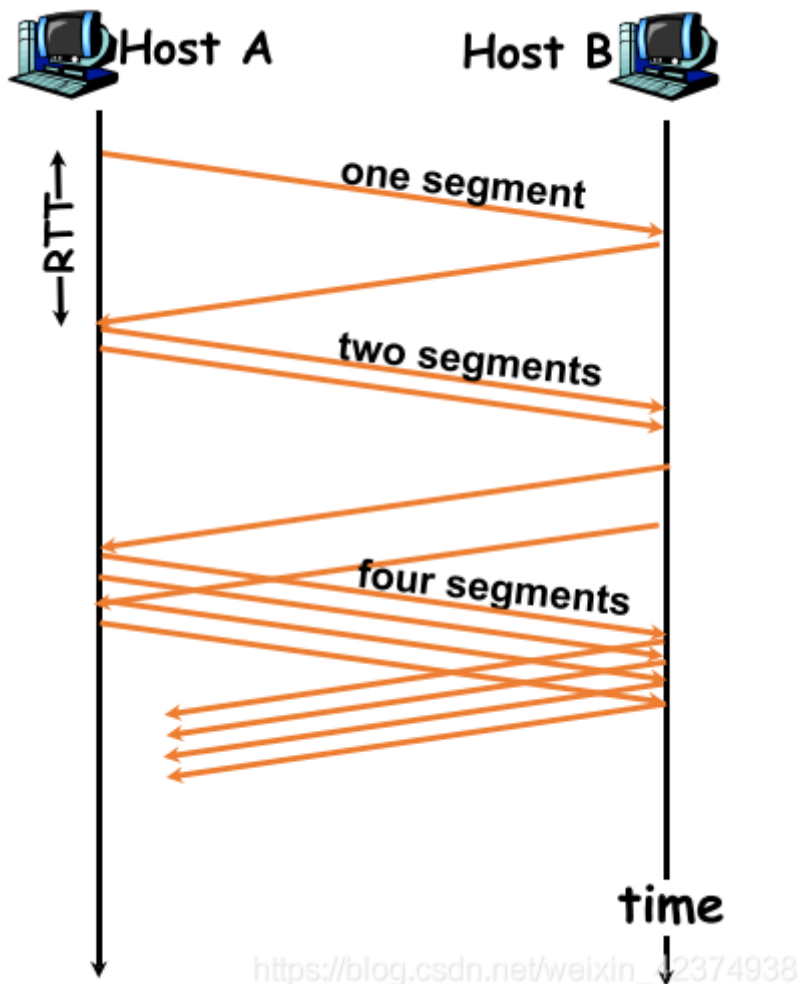
TCP拥塞策略（TCP的流量控制于拥塞控制一起实现）

慢启动

当一条TCP连接开始时，cwnd（拥塞窗口）的值通常初始置为一个最大报文长度（MSS）的较小值。因此，在慢启动状态， $cwnd=1MSS$ 。

例如 $MSS=500$ 字节且 $RTT=200ms$ ，那么初始速率为20kbps。

由于对TCP发送方而言，有效带宽将远大于初始发送速率（ MSS/RTT ），因此TCP发送方希望尽快达到期待的速率然后将以2的指数方式增加速率，直到产生丢包事件，或者达到某个阈值。



如上图所示，TCP向网络发送第一个报文段并等待一个确认。当该确认到达时，TCP发送方将拥塞控制窗口增加一个MSS，并发送出两个最大长度的报文段。这两个报文段被确认，则发送方对每个确认报文段将拥塞窗口增加一个MSS，使得拥塞窗口变为4个MSS，并这样下去。这一个过程每过一个RTT，发送速率就翻番。因此，TCP发送速率起始慢，但在慢启动阶段以指数增长。

慢启动结束方式有如下几种：

- (1) 如果存在一个由超时指示的丢包事件即出现拥塞，TCP发送方将cwnd设置为1并重新开始慢启动过程，并且将慢启动阈值置为cwnd值的一半，窗口开始指数增长。
- (2) 因为当检测到拥塞时阈值设为拥塞窗口的一半，重新开始慢启动，这时当cwnd的值等于阈值时结束慢启动并且TCP转移到拥塞避免模式，这时窗口开始线性增长。
- (3) 如果检测到3个冗余的ACK，这时TCP执行一种快速重传并进入快速恢复状态，窗口线性增长。

拥塞避免

一旦进入拥塞避免状态，cwnd的值大约是上次遇到拥塞时的值的一半，即距离拥塞并不遥远。因此，TCP无法每过一个RTT再将拥塞窗口的值翻番。而是采取了一种较为保守的方法，每个RTT只将拥塞窗口的值增加一个MSS。通用增加方法如下：

对于TCP发送方无论何时到达一个新的确认，就将cwnd的值增加一个MSS ($MSS/cwnd$) 字节。

例如MSS是1460字节并且cwnd的值为14600字节，则在一个RTT内发送10个报文段，每个到达ACK（每个报文段一个ACK）增加1/10MSS的拥塞窗口的长度，因此在收到对所有10个报文段的确认后，cwnd的值将增加了一个MSS。

拥塞避免的结束方式有如下几种：

(1) 当出现超时导致的丢包事件时，TCP的拥塞避免算法行为相同。与慢启动的情况一样，cwnd的值被设置为1个MSS，阈值被更新为cwnd的一半，窗口指数增长。

(2) 当收到3个冗余的ACK导致丢包时：TCP将cwnd值减半，并且将阈值记录为cwnd值的一半加3MSS或直接为1MSS（由不同版本决定），窗口线性增长。接下来进入快速恢复状态。

快速恢复

1. 当出现超时导致的丢包事件时，cwnd的值被设置为1个MSS，阈值被更新为cwnd的一半，迁移到慢启动状态，窗口指数增长。
2. 当收到3个冗余的ACK导致丢包时，TCP在降低cwnd后进入拥塞避免状态。

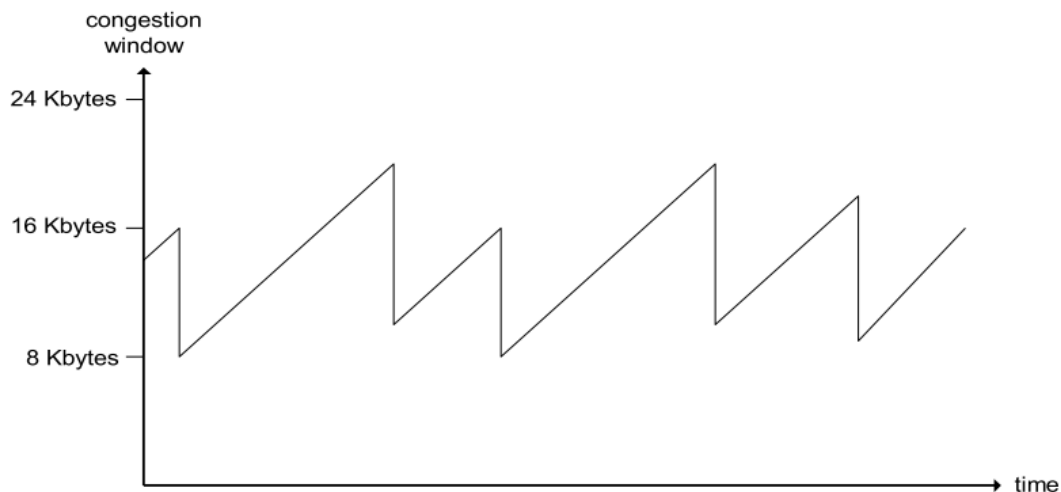
对于TCP拥塞控制算法的三个过程来说，我们可以简单的概括为：

- (1) 当cwnd低于阈值，发送方处于慢启动阶段，窗口指数增长。
- (2) 当cwnd高于阈值，发送方处于拥塞避免阶段，窗口线性增长。
- (3) 当三个重复的ACK出现时，阈值置为cwnd值的一半，并且cwnd为阈值+3或直接为1（根据不同版本决定）。
- (4) 当超时发生时，阈值为cwnd值的一半并且cwnd置为1MSS。

AIMD(Additive Increase Multiplicative Decrease 加性增，乘性减)

我们现在来回顾以下全局，忽略一天连接开始时初始的慢启动阶段，假定丢包由3个冗余的ACK而不是超时所导致，TCP的拥塞控制是：每个RTT内cwnd线性增加1MSS（加性增），然后出现3个冗余ACK事件时cwnd减半（乘性减）。因此，TCP拥塞控制常常被称为加性增、乘性减（AIMD）拥塞控制方式。

AIMD拥塞控制方式引发了下图的“锯齿”行为。



Long-lived TCP connection https://blog.csdn.net/weixin_42374938

TCP的公平性

大致公平。

多个TCP连接，经过线性增，乘性减，最终结果就是他们获得的链路带宽一致。

网络层

导论

数据平面、控制平面

网络层：数据平面、控制平面


数据平面

- 本地，每个路由器功能
- 决定从路由器输入端口到达的分组如何转发到输出端口
- 转发功能：
 - 传统方式：基于目标地址+转发表
 - SDN方式：基于多个字段+流表

控制平面

- 网络范围内的逻辑
- 决定数据报如何在路由器之间路由，决定数据报从源到目标主机之间的端到端路径
- 2个控制平面方法：
 - 传统的路由算法：在路由器中被实现
 - *software-defined networking (SDN)*: 在远程的服务器中实现

values in arriving packet header



网络层：数据平面 4-7

子网

SDN

软件定义网络(SDN) 是一种软件集中控制、网络开放的三层体系架构,如图 (1) 所示.应用层实现对网络业务的呈现和网络模型的抽象;控制层实现网络操作系统功能, 集中管理网络资源;转发层实现分组交换功能.应用层与控制层之间的北向接口是网络开放的核心,控制层的产生实现了控制面与转发面的分离,是集中控制的基础。

SDN最主要的特征就是数据转发和控制分离, 同时还具有网络虚拟化和开放接口等特征。

路由选择算法

主机通常直接与一台**路由器**相连接, 该路由器即为该主机的默认路由器(default router), 又称该主机的**第一跳路由器**(first-hop router)每当主机发送一个分组时, 该分组被传送给它的默认路由器。

源主机的**默认路由**器称作源路由器(sourcerouter), 目的主机的默认路由器称作目的路由器(destination router)。

一个分组从源主机到目的主机的路由选择问题显然可归结为**从源路由器到目的路由器的路由选择问题**。

分类

- 路由算法的分类一
 - **全局式路由选择算法 (global routing algorithm)**
 - 所有的路由器掌握完整的网络拓扑和链路费用信息
 - 实践中, 具有全局状态信息的算法常被称作**链路状态 (Link State, LS) 算法**, 因为该算法必须知道网络中每条链路的费用
 - **分散式路由选择算法 (decentralized routing algorithm)**
 - 路由器只掌握物理相连的邻居及链路费用
 - 以迭代、分布式的方式计算出最低费用路径
 - 例如, **距离向量 (Distance- Vector, DV) 算法**
- 路由算法的分类二
 - **静态路由选择算法 (Static routing algorithm)**
 - 人工干预进行调整(如人为手工编辑一台路由器的转发表)
 - **动态路由选择算法 (Dynamic routing algorithm)**
 - 能够当网络流量负载或拓扑发生变化时动态改变路由选择路径
- 路由算法的分类三
 - **负载敏感算法 (load-sensitive algorithm)**
 - 链路费用会动态地变化以反映出底层链路的当前拥塞水平
 - 早期的 ARPAnet 路由选择算法就是负载敏感的
 - **当今的因特网路由选择算法都是负载迟钝的 (load-insensitive)**, 因为某条链路的费用不明思地反映其当前(或最近)的拥塞水平。

算法

- 下面的路由选择协议都是在一个自治区域之内的路由协议，所有的路由器都是在一个平面之中
- 针对不同自治区域之间的路由，可以使用 **BGP**

链路状态路由选择(link state routing)

LS路由的基本工作过程：

1. 发现相邻节点，获知对方网络地址
2. 测量到相邻节点的代价（延迟、开销）
3. 组装一个 **LS分组**，描述它到相邻节点的代价情况
4. 将分组通过扩散(泛洪)的方法发到其他所有路由器

以上4步让每个路由器获得拓扑和边代价

5. 通过 **Dijkstra算法** 找出最短路径（这才是路由算法）
 1. 每个节点独立算出来到其他节点（路由器=网络）的最短路径
 2. 迭代算法：第K步能够知道本节点到k个其他节点的最短路径

距离矢量路由选择（distance-vector routing）

距离矢量路由协议是基于贝尔曼-福特算法（工程实践中称D-V算法）的动态路由协议，使用距离矢量的路由器，最关心的是到目的网段的距离（Metric）和矢量（方向，从哪个接口转发数据）

距离（Metric）是指数据报在传递过程中所经过的路由器的“跳数”（hop count）

距离矢量协议和链路状态协议的区别

一．什么是距离向量路由协议以及什么是链接状态路由协议？

1. 这类协议使用[贝尔曼-福特算法](#)（Bellman-Ford）计算路径。在距离-矢量路由协议中，每个路由器并不了解整个网络的拓扑信息。它们只是向其它路由器通告自己的距离、也从其它路由器那里收到类似的通告。（如果在90秒内没有收到相邻站点发送的路由选择表更新，它才认为相邻站点不可达。每隔30秒，距离向量路由协议就要向相邻站点发送整个路由选择表，使相邻站点的路由选择表得到更新。这样，它就能从别的站点（直接相连的或其他方式连接的）收集一个网络的列表，以便进行路由选择。距离向量路由协议使用跳数作为度量值，来计算到达目的地要经过的路由器数。）

每个路由器都通过这种路由通告来传播它的路由表。在之后的通告周期中，各路由器仅通告其路由表的变更。该过程持续至所有路由器的路由表都收敛至一稳定状态为止。

这类协议具有收敛缓慢的缺点，然而，它们通常容易处理且非常适合小型网络。距离-矢量路由协议的一些例子包括：[路由信息协议](#)（RIP）[内部网关路由协议](#)（IGRP）

2. 链接状态路由协议更适合大型网络，但由于它的复杂性，使得路由器需要更多的C P U资源。在链路状态路由协议中，每个节点都知晓整个网络的拓扑信息。各节点使用自己了解的网络拓扑情况来各自独立地对网络中每个可能的目的地址计算出其最佳的转发地址（下一跳）。所有最佳转发地址汇集到一起构成该节点的完整路由表。

与距离-矢量路由协议使用的那种每个节点与其相邻节点分享自己的路由表的工作方式不同，链路状态路由协议的工作方式是节点间仅传播用于构造网络连通图所需的信息。

最初创建这类协议就是为了解决距离-矢量路由协议收敛缓慢的缺点，然而，为此链路状态路由协议会消耗大量的内存与处理器能力。（它能够在更短的时间内发现已经断了的链路或新连接的路由器，使得协议的会聚时间比距离向量路由协议更短。通常，在10秒钟之内没有收到邻站的HELLO报文，它就认为邻站已不可达。一个链接状态路由器向它的邻站发送更新报文，通知它所知道的所有链路。它确定最优路径的度量值是一个数值代价，这个代价的值一般由链路的带宽决定。具有最小代价的链路被认为是最优的。在最短路径优先算法中，最大可能代价的值几乎可以是无限的。）

如果网络没有发生任何变化，路由器只要周期性地将没有更新的路由选择表进行刷新就可以了（周期的长短可以从30分钟到2个小时）。

链路状态路由协议的例子有：[开放式最短路径优先协议（OSPF）](#)，[中间系统到中间系统路由交换协议（IS-IS）](#)

二. 具体理解链路状态和距离矢量路由协议

距离矢量（DV）是“传说的路由”，A发路由信息给B，B加上自己的度量值又发给C，路由表里的条目是听来的，虽说“兼听则明，偏信则暗”，但是选出最优路径的同时会引发环路问题，当然，DV协议也使用水平分割，毒性逆转，触发更新等特性来避免，无奈的是，这种问题对于竞争对手LS而言是天生免疫的。

链路状态（LS）是“传信的路由”，A将信息放在一封信里发给B，B对其不做任何改变，拷贝下来，并将自己的信息放在另一封信里，两封信一起给C，这样，信息没有任何改变和丢失，最后所有路由器都收到相同的一堆信，这一堆信就是LSDB。然后，每个路由器运用相同的SPF算法，以自己为根，计算出SPF Tree（即到达目的地的各个方案），选出最佳路径，放入转发数据库中（即路由表）。

链路状态协议有三样看家本领：LSDB，SPF算法，SPF Tree。还有三张表：邻居表，拓扑表，路由表，但这三张表并不是DV和LS的根本区别，EIGRP作为高级的距离矢量路由协议同样有这三张表，关键点在于表的内容和传递信息的过程。

DV的拓扑表事实上是邻居通告的路由条目的集合，依据算法从中选出最佳的放进路由表，它并不完全了解网络拓扑；而LS的拓扑表是真正意义上的网络拓扑，路由器对网络信息完全了解，所以可以独立的做出决策，确定最佳路由。举例来说，如果我是DV的思维，我从华师去火车站，通过询问知道，我可以在走到师大暨大车站坐515路车，也可以走到坐177路车，这样问下来有几种方案，我再选一个最优的，以这样的方式我就知道广州市内的一些地方该怎么去；而如果我是LS的思维，我会先去四下打听，搜集信息然后汇总成一张广州市区的地图，然后依据这张地图自己决定如何去火车站以及其它地方。

路由层次

一个平面的路由

- 一个网络中的所有路由器的地位一样
- 通过LS、DV，或者其他路由算法，所有路由器都要知道其他所有路由器（子网）如何走
- 所有路由器在一个平面

平面路由的问题

- 规模巨大的网络，路由信息的存储、传输和计算代价巨大
 - DV：距离矢量很大，且不能收敛
 - LS：几百万个节点的LS分组的泛洪传输，存储以及最短路径算法的计算
- 管理问题
 - 不同的网络所有者希望按照自己的方式管理网络

- 希望对外隐藏自己网络的细节
- 还要和其他网络互联

层次路由：将互联网分成一个个AS（路由器区域）

- 某个区域内的路由器集合，自治系统 "autonomus systems"(AS)
- 一个AS用AS Number (ASN) 唯一标识
- 一个ISP可能包括1个或者多个AS

路由变成了：2个层次路由

- AS内部路由：在同一个AS内部路由器运行相同的路由协议
 - “intra-AS” routing protocol: 内部网关协议
 - 不同的AS可能运行着不同的内部网关协议
 - 如：RIP、OSPF、IGRP
 - 路由信息协议RIP (Routing Information Protocol) 是基于[距离矢量](#)算法的[路由协议](#)，利用跳数来作为计量标准。
 - OSPF(Open Shortest Path First[开放式最短路径优先](#)) 是一个[内部网关协议](#)(Interior Gateway Protocol, 简称IGP)，用于在单一[自治系统](#) (autonomous system,AS) 内决策[路由](#)。是对[链路状态路由协议](#)的一种实现，隶属内部网关协议 (IGP)，故运作于自治系统内部。著名的迪克斯彻 (Dijkstra) 算法被用来计算[最短路径树](#)。OSPF支持负载均衡和基于服务类型的选路，也支持多种路由形式，如特定主机路由和子网路由等。
 - 能够解决规模和管理问题
 - 网关路由器：AS边缘路由器，可以连接到其他AS
- AS间运行AS间的路由协议
 - “inter-AS” routing protocol: 外部网关协议
 - 解决AS之间的路由问题，完成AS之间的互联互通

层次路由的优点

- 解决了规模问题
 - 内部网关协议解决: AS内部数量有限的路由器相互到达的问题，AS内部规模可控
 - 如果AS节点过多，可以分割AS，使得AS内部的节点数量有限
 - AS路由之间的规模问题
 - 增加一个AS，对于AS之间的路由从总体上来说，只是增加了一个节点=子网（每个AS可以用一个点来表示）
 - 对于其他AS来说只是增加了一个表项，就是这个新增的AS如何走的问题
 - 扩展性强，规模增大，性能不会减的太多
- 解决了管理问题
 - 各个AS之间可以运行不同的内部网关协议
 - 可以使自己网络的细节不向外透露

边际网关协议（Border Gateway Protocol） BGP

- 自治区域间路由协议“事实上的标准”
 - 将各个AS连接在一起
- BGP提供给每个AS以下方法：
 - eBGP：从相邻的ASes那里获得子网可达信息
 - iBGP：将获得的子网可达信息传输到AS内部的所有路由器
 - 根据子网可达信息和策略来决定到达子网的“好”路径
- 允许子网向互联网其他网络通告“我在这里”
- 基于距离矢量算法（路径矢量）
 - 不仅仅是距离矢量，还包括到达各个目标网络的详细路径（AS序号的列表）能够避免简单DV算法的路由环路问题

BGP的路径选择

- 路由器可能获得一个网络前缀的多个路径，路由器必须进行路径的选择，路由选择可以基于：
 - 本地偏好属性：偏好策略决定
 - 最短AS-PATH：AS的跳数
 - 最近的NEXT-HOP路由器：热土豆路由
 - 热土豆策略：选择具备最小内部区域代价的网关作为出口（即使该出口可能有更多的AS跳数，不用操心域间的代价）
 - 附加的判据：使用BGP标示
- 一个前缀对应着多种路径，采用消除规则直到留下一条路径

数据链路层

链路层提供的服务

- 成帧

多点访问协议

网络安全

什么是网络安全？

- 机密性：只有发送方和接收方能理解传输得报文内容
 - 发送方加密保密
 - 接收方解密报文
- 认证：发送方和接收方需要确认对方得身份
- 报文完整性：发送方、接收方需要确认报文在传输过程中或时候没有被改变
- 访问控制和服务的可用性：服务可以接入以及对用户而言是可用的

附录

传输层

MTU（Maximum Transmission Unit）最大传输单元

最大传输单元MTU（Maximum Transmission Unit，MTU），是指网络能够传输的最大数据包大小，以字节为单位。MTU的大小决定了发送端一次能够发送报文的最大字节数。如果MTU超过了接收端所能够承受的最大值，或者是超过了发送路径上途经的某台设备所能够承受的最大值，就会造成报文分片甚至丢弃，加重网络传输的负担。如果太小，那实际传送的数据量就会过小，影响传输效率。

为什么需要MTU

网络中通常以数据包为单位进行信息传递，那么，一次传送多大的包合适、多大的包最高效就成为一个核心问题之一。如果包大小设置的很大，意味着报文中的有效数据也更多，通信效率更高，但传送一个数据包的延迟也越大，数据包中bit位发生错误的概率也越大。并且如果这个报文丢掉了，重传的代价也很大。如果包大小设置的过小，则意味传输相同的数据量，设备需要处理更多的报文，这样会极大的考验设备的线速转发能力。通过设置MTU来调节网络上数据包的大小，让不同的网络找到最适宜的MTU从而提高转发效率，这就是MTU的作用。

MTU是数据链路层的概念，指数据链路层对数据帧长度的限制。不同链路介质类型的网络有不同的默认MTU值，以下是一些常见网络的默认值：

网络类型	MTU (单位: 字节)	RFC
Token Ring(16Mbps)	17914	-
IEEE 802.4	8166	RFC 1042
Token Ring(4Mbps)	4464	RFC 1042
FDDI	4352	RFC 1390
Ethernet	1500	RFC 894
IEEE 802.3	1492	RFC 1042
X.25	576	RFC 879

为什么以太网MTU通常被设置为1500?

RFC标准定义以太网的默认MTU值为1500。那么这1500的取值是怎么来的呢?

早期的以太网使用共享链路的工作方式, 为了保证CSMA/CD (载波多路复用/冲突检测) 机制, 所以规定了以太帧长度最小为64字节, 最大为1518字节。最小64字节是为了保证最极端的冲突能被检测到, 64字节是能被检测到的最小值; 最大不超过1518字节是为了防止过长的帧传输时间过长而占用共享链路太长时间导致其他业务阻塞。所以规定以太网帧大小为64~1518字节, 虽然技术不断发展, 但协议一直没有更改。

以太网最大的数据帧是1518字节, 这样刨去帧头14字节和帧尾CRC校验部分4字节, 那么剩下承载上层IP报文的地方最大就只有1500字节, 这个值就是以太网的默认MTU值。这个MTU就是网络层协议非常关心的地方, 因为网络层协议比如IP协议会根据这个值来决定是否把上层传下来的数据进行分片, 如果单个IP报文长度大于MTU, 则会在发送出接口前被分片, 被切割为小于或等于MTU长度的IP包。

超过MTU的报文如何进行分片?

以太网缺省MTU=1500字节, 这是以太网接口对IP层的约束, 如果IP层有<=1500字节需要发送, 只需要一个IP包就可以完成发送任务; 如果IP层有>1500字节数据需要发送, 需要分片才能完成发送。

以主机发送一个数据载荷长度为2000字节的报文为例说明其分片的过程 (假设出接口的MTU值为1500)。在网络层会对报文进行封装, 其结构组成: IP头部20字节+数据载荷长度2000字节, 报文封装后, 整个报文长度为2020字节。在出接口进行转发的时候, 发现IP报文的长度超过了MTU的值1500, 因此要进行分片处理。

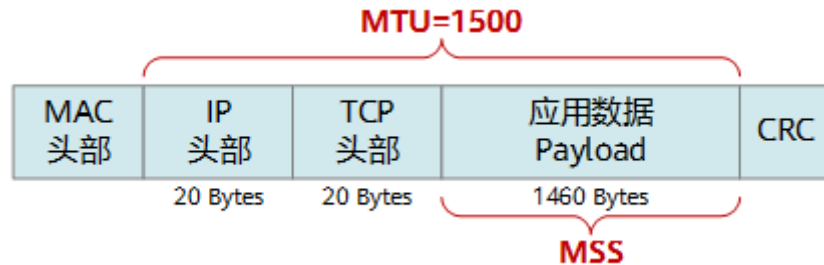
第一片报文, IP报文头固定20字节, 数据载荷可以封装1480字节 (MTU值1500字节-IP报文头20字节, 数据载荷长度须是8的倍数);

第二片报文, 复制第一片的IP头, IP报文头固定20字节, 数据载荷为剩余的520字节 (总数据载荷长度2000字节减去第一片中已封装的1480字节)。如果最后一片报文的长度不足46字节, 会自动填充至46字节。

所有分片报文在发送至目的主机后, 在目的主机进行分片重组, 恢复为原报文。在进行重组时, 通过IP标志位中的MF用来分辨这是不是最后一个分片, 片偏移用来分辨这个分片相对原数据报的位置。通过这几个字段, 可以准确的完成数据报的重组操作。

TCP MSS与MTU

TCP MSS (Maximum Segment Size) 是指TCP协议所允许的从对方收到的最大报文长度，即TCP数据包每次能够传输的最大数据分段，只包含TCP Payload，不包含TCP Header和TCP Option。MSS是TCP用来限制application层最大的发送字节数。为了达到最佳的传输效能，TCP协议在建立连接的时候通常要协商双方的MSS值，这个值TCP协议在实现的时候往往根据MTU值来计算（需要减去IP包头20字节和TCP包头20字节），所以通常MSS为 $1460=1500(\text{MTU})-20(\text{IP Header})-20(\text{TCP Header})$ 。



网络层

DHCP（动态主机配置协议）

动态主机设置协议（Dynamic Host Configuration Protocol, DHCP）是一个[局域网](#)的[网络协议](#)，使用[UDP](#)协议工作，主要有两个用途：

- 给内部网络或网络服务提供商自动分配[IP](#)地址给用户
- 给内部网络管理员作为对所有电脑作中央管理的手段

DHCP的工作原理：

1. 客户端将发送DHCP发现报文。这是广播报文，因为它没有DHCP服务器的IP地址，也不知道网络上是否有DHCP服务器。当然，在我们的网络中，我们确实有一个DHCP服务器，因此它将响应该广播报文。
2. DHCP服务器将以一条包含计算机IP地址的DHCP提供消息进行响应（我们必须配置DHCP服务器来定义我们要提供的IP地址——地址池）。如果需要，我们还可以为计算机分配默认网关和DNS服务器。
3. 当客户端接收到服务器的响应报文后，客户端将根据响应报文提供的信息而发送DHCP请求，询问服务器是否可以使用它收到的信息。
4. DHCP服务器接收到后将以DHCP ACK消息响应，告诉客户端可以使用此信息。

DHCP与NAT的区别：

DHCP是用来[分配IP地址](#)的。

NAT是用来[转换IP地址](#)的。