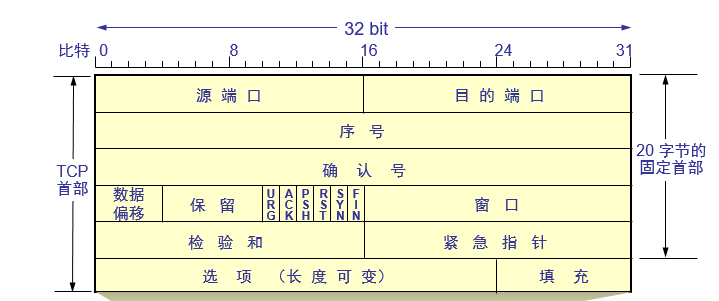
TCP首部



**SourcePort(源端口)和DestinationPort（目的端口）**：各占2个字节，端口是运输层与应用层的服务接口,运输层的复用和分用功能都要通过端口才能实现。

   注：TCP的包是没有IP地址的，那是IP层上的事。但是有源端口和目标端口。  
**SequenceNumber（序号）**：占4个字节，TCP连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。  
**AcknowledgmentNumber（确认号）**：占4字节，是期望收到对方的下一个报文段的数据的第一个字节的序号。  
**DataOffset（数据偏移（即首部长度））**：占4位，它指出TCP报文段的数据起始处距离TCP报文段的起始处有多远。“数据偏移”的单位是32位字，即以4字节为计算单位。   
**Reserved(保留字段)**:占6位，保留为今后使用，但目前应置为0。  
**TCPFlag(标志位)**:占6位，分别表示6个标志：**紧急URG**，**确认ACK**，**推送PSH(PuSH)**，**复位RST(ReSeT)**，**同步SYN**，**终止FIN(FINis)**：  
URG—— 当URG= 1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。  
**ACK**—— 当ACK= 1 时确认号字段有效,当ACK= 0 时，确认号无效。  
**PSH(PuSH)** ——接收TCP收到PSH= 1 的报文段，就尽快地交付接收应用进程，而不再等到整个缓存都填满了后再向上交付。  
**RST(ReSeT)**—— 当RST= 1 时，表明TCP连接需要释放连接，然后再重新建立运输连接。  
**SYN**—— 同步SYN= 1 表示这是一个连接请求或连接接受报文。  
**FIN(FINis)**——用来释放一个连接。FIN= 1 表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。

**Window（窗口字段）**：占2字节，用来让对方设置发送窗口的依据，单位为字节。表示接收缓冲区的空闲空间，16位，用来告诉TCP连接对端自己能够接收的最大数据长度。  
**Checksum（检验和）**：占2字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在TCP报文段的前面加上12字节的伪首部。  
**UrgentPointers（紧急指针字段）**：占16位，指出在本报文段中紧急数据共有多少个字节（紧急数据放在本报文段数据的最前面）。  
**options（选项字段）**：长度可变。TCP最初只规定了一种选项，即最大报文段长度MSS。MSS告诉对方TCP：“我的缓存所能接收的报文段的数据字段的最大长度是MSS个字节。”在TCP的MSS选项中，MSS值是一个16位的字段，最大值为65535.

MSS是TCP报文段数据部分的最长长度,不是总长度.MSS太小,首部20字节,有效数据1个字节,网络效率太低.MSS太大,IP要分层,浪费时间.

TCP三次握手

TCP三次握手过程

1. 服务器处于LISTEN状态,等待客户端连接.
2. 客户端发送SYN请求连接包(SYN = 1,seq = m,进入SYN-SENT状态)
3. 服务器收到SYN包,同意连接.回复ACK-SYN包(SYN = 1,ACK = 1,ack = m + 1,seq = n,进入SYN-RCVD状态,第一次握手完成).
4. 客户端收到ACK包,检查其ack,若ack = m + 1,表示服务器同意连接,回复ACK包(ACK = 1,seq = m + 1,ack = n + 1,进入ESTABLISHED状态,第二次握手完成).
5. 服务器收到ACK包,检查其ack = n + 1, 进入ESTABLISHED状态,第三次握手完成.

TCP为什么握手三次,而不是两次或四次

假设一种情况,客户端发送了第一个SYN包,但是由于第一个SYN包网络延时,没按时到达服务器,客户端没及时收到确认,会再发出第二个SYN包,如果采用了两次握手机制,服务器一旦收到SYN包便进入ESTABLISHED状态,服务器收到第二个SYN包,成功建立了一个TCP连接,此时,延迟的SYN包到达了服务器,服务器又建立了一个连接并回复了一个无效的确认包,客户端收到无效的确认包直接丢弃.那么,延迟的SYN包使得仅有服务器进入ESTABLISHED状态,但永远不会有数据传输,造成了资源浪费.

TCP的四次分手过程

1. A发送FIN包(FIN = 1,seq = u = 上次传输的最大的字节序号+1,进入FIN-WAIT-1状态)
2. B收到FIN包,回复ACK包(ACK = 1,ack = u + 1,进入CLOSE-WAIT状态)
3. A收到ACK包,进入FIN-WAIT-2状态

此时TCP连接处于半关闭状态,B若还有数据发给A,A仍可接收.

1. 若B没有数据要发给A,发送FIN包(FIN = 1,seq = v = 上次传输的最大的字节序号+1,进入LAST-ACK状态)
2. A发送ACK包(ACK = 1,ack = v + 1,进入TIME-WAIT状态)
3. B收到ACK包后关闭连接
4. A等待两个最长报文段寿命时间后,关闭

发现:A先提出的关闭,但是最终B先关闭的.第三次握手后,A没直接关闭,而是等待两个最长报文段寿命才关闭,原因如下:(1)如果第三次握手发送的ACK包失效,B无法关闭.(2)两个最长报文段寿命内,能使当前即将关闭的连接运输的报文段全部从网络中消失,不会造成本次的报文段出现在A的下一次连接中.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Client |  |  |  |  | Sever |  |
|  |  |  |  |  |  |  |  |
| 序列号 | | seq = m |  |  |  |  |  |
| 确认号 | |  |  |  |  |  |  |
| SNY = 1 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | | seq = n |
|  |  |  |  |  | 确认号 | | ack = m+1 |
|  |  |  |  |  | SNY = 1 | ACK = 1 |  |
|  |  |  |  |  |  |  |  |
| 序列号 | | seq = m+1 |  |  |  |  |  |
| 确认号 | | ack = n +1 |  |  |  |  |  |
|  | ACK = 1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | |  |
|  |  |  |  |  | 确认号 | |  |
|  |  |  |  |  |  |  | 建立连接 |
|  |  |  |  |  |  |  |  |
| ………传输数据……… | | | | | | | |
|  |  |  |  |  |  |  |  |
| 序列号 | | seq = m+1 |  |  |  |  |  |
| 确认号 | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | |  |
|  |  |  |  |  | 确认号 | | ack = m+201 |
|  |  |  |  |  |  | ACK = 1 |  |
|  |  |  |  |  |  |  |  |
| 序列号 | | seq = m+201 |  |  |  |  |  |
| 确认号 | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | |  |
|  |  |  |  |  | 确认号 | | ack = m+301 |
|  |  |  |  |  |  | ACK = 1 |  |
|  |  |  |  |  |  |  |  |
| 序列号 | | seq = m+301 |  |  |  |  |  |
| 确认号 | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | |  |
|  |  |  |  |  | 确认号 | | ack = m+401 |
|  |  |  |  |  |  | ACK = 1 |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| ………关闭连接……… | | | | | | | |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 序列号 | | seq = u |  |  |  |  |  |
| 确认号 | |  |  |  |  |  |  |
|  | FIN = 1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | |  |
|  |  |  |  |  | 确认号 | | ack = u+1 |
|  |  |  |  |  |  | ACK = 1 |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | | seq = v |
|  |  |  |  |  | 确认号 | |  |
|  |  |  |  |  |  | FIN = 1 |  |
|  |  |  |  |  |  |  |  |
| 序列号 | |  |  |  |  |  |  |
| 确认号 | | ack = v+1 |  |  |  |  |  |
|  | ACK = 1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 序列号 | |  |
|  |  |  |  |  | 确认号 | |  |
|  |  |  |  |  |  |  | 关闭 |
|  |  |  |  |  |  |  |  |
| 序列号 | |  |  |  |  |  |  |
| 确认号 | |  |  |  |  |  |  |
|  |  | 关闭 |  |  |  |  |  |

TCP为什么能实现可靠传输

实现可靠传输需要保证三个条件:  
(1)无比特差错传输

(2)字节流不丢不重不乱序

(3)接收方的处理能力大于发送速率

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TCP的首部检验和会检查首部和数据段,保证报文段无比特差错,满足条件(1).

应用程序把数据缓存到发送缓存,接收方TCP把数据放到接收缓存.发送方根据接收方反馈回来的窗口设置发送窗口,窗口是缓存的前片段,缓存内窗口外的字节不允许发送,实现了流量控制,满足条件(3).

窗口内的字节分成两部分:已发送还未确认 & 可发送还未发送,接收缓存的存储的是:按序到达还未提取,未按序到达的(4没到,5到了),接收方按序发送确认包,比如4的确认包没发送,就不会发送5的确认包.接收方收到确认包,就丢弃窗口中对应的字节,窗口往前滑动,新增字节.确认机制满足条件(3).

所以TCP能实现无差错传输.

|  |  |  |  |
| --- | --- | --- | --- |
| 发送缓存 | | 接收缓存 | |
| 已发送未确认 | 可发送未发送 | 未提取的正确字节 | 未按序到达的字节 |

TCP中的计时器

TCP中的计时器?

1. 重传计时器

TCP发送完一个报文段,就设置一个专属于此报文段的计时器,规定时间内收到此报文段的确认,撤销计时器,时间走完还没收到确认包,重传此报文段并重置计时器.

1. 持续计时器

客户端收到的确认包窗口是0,便停止发送数据了.过了一会,接收端缓过来劲了,继续发送一个更高序号的字节的确认包,它的窗口大于0,客户端如果收到此确认包,检测到窗口大于0,就会重新发送数据.但是如果此”激活”确认包万一丢失,双方都会永久静默下去(TCP不会重传ACK确认包).所以为每个TCP连接设置一个计时器,每隔一段时间,探测一下是不是丢包了.

1. 保活计时器

数据传输中途,客户端崩溃,防止服务器傻傻等待而浪费资源,会每隔一段时间,检查一下客户端是不是崩溃了.

1. 时间等待计时器,TCP四次分手的时候使用的.

Socket编程

TCP编程服务器步骤:

1. 创建一个socket,函数socket()
2. 设置socket属性,函数setsockeopt()
3. 绑定IP地址,端口到socket,函数bind()
4. 开启监听,函数listen()
5. 接收客户端连接accept()
6. 收发数据,函数read(),write(),send(),rev()
7. 关闭网络连接
8. 关闭监听

TCP编程客户端步骤:

1. 创建一个socket
2. 设置socket属性
3. 绑定IP地址和端口
4. 设置要连接的对方的IP地址和端口属性
5. 连接服务器,函数connect()
6. 收发数据
7. 关闭网络连接

UDP编程服务器:

1. socket()
2. setsockopt()
3. bind()
4. 循环接收数据,recvfrom()
5. 关闭网络连接

UDP编程客户端:

1. socket()
2. setsockopt()
3. 绑定IP地址和端口
4. 绑定对方的IP地址和端口
5. 发送数据,sendto()
6. 关闭网络连接

TCP首部的确认号是期望收到对方下一次报文段的第一个字节的序号,确认号是N,意味着N-1序号前的字节全部被正确收到.

每一条TCP连接唯一的被通信两端的两个端点(两个套接字)所确定.

TCP并不关心应用进程一次把多长的报文放到TCP的缓存,而是根据对方给的窗口值和当前网络拥塞的程度来决定一个报文段应包含多少字节.

如果应用进程传送到TCP缓存的数据太长,TCP就可以把它划分的更短一些,如果应用进程只发来一个字节,TCP就可以积累有足够的字节后再构成报文段发送出去.

UDP首部

UDP首部固定8个字节 🡪 源端口,目的端口,数据报总长度,检验和,各占2个字节.

TCP和UDP区别

|  |  |
| --- | --- |
| TCP | UDP |
| 面向连接 | 无连接 |
| 可靠的全双工通信 | 尽力而为 |
| 拥塞控制,流量控制,确认机制 | 有就发,发出去死活不管 |
| 首部最少20字节 | 首部固定8字节 |
| 面向字节流 | 面向报文 |
| 一对一 | 任意量对任意量 |

运输层习题

运输层为应用程序提供端到端的逻辑通信,网络层负责主机寻址和分组交换.

应用进程之间的通信是可靠还是尽力而为,由运输层决定并加载到网络层.

什么是复用和分用?

应用层的数据通过不同端口加载到同一个TCP/IP连接传输,叫复用.

运输层把从IP层拿到的数据,通过正确的交给不同的端口,叫分用.

UDP/TCP和实时音频

实时场景,宜采用UDP,UDP产生数据就直接发送,不考虑网络拥塞,接收端流量,错误丢弃不重传,所以数据传输速度快,实时性好.虽然有可能数据报错误,但是人眼耳有一定的承受能力,如果采用TCP可靠传输,虽然没有数据报错误,但是重传,流量控制拥塞控制,会造成时延,综合考虑,实时性重于可靠,故选用UDP.

如果采用TCP观看收听音频,可以回看.

为什么说UDP是面向数据报的,TCP是面向字节流的?

UDP对应用层数据不拆不并,只保留边界交个IP层.TCP视应用层数据是无结构的字节流,根据需要拆分合并,但是不打乱字节流的顺序.

什么是端口?

端口16bit,来标记应用进程.各种服务器使用的端口号都是保留端口号,以便客户端能够找到服务器.

什么是套接字?

套接字 = {IP地址 : 端口号},套接字由应用程序程序产生,并指明它将由客户端还是服务器使用.

是否TCP和UDP都需要计算往返时间RTT?

TCP发送方的报文需要确认,所以要计算RTT,来设置合适的重传时间.UDP发出去是死是活就不管了,不需要计算RTT.

TCP在建立新的连接时所选择的初始序号,一定要和前面的一些连接所使用的序号不同,防止后效作用.

为什么ACK包不消耗序号,SYN包消耗序号?

TCP给传输的每个字节编号,目的是为了保证每一个字节能正确到达对方.接收方按字节的编号对收到的数据进行核对,用发送确认报文段的方法保证能够收到发送方的每一个字节.

确认报丢失不会重传,因为没必要重传,假设对1000字节的确认报丢失,可能会造成发送方多发送一个900-1000的字节段,但是对于多收的字节,接收方会丢弃.等对1100字节的确认包到达发送方,那有没有对1000字节的确认包无所谓了,因为收到对1100字节的确认包,已经意味着1100字节前的所有字节都已正确收到.

所以对ACK包没必要编号,ACK包不消耗序号,但是ACK包如果携带数据,会消耗序号.

SYN包只有首部,数据部分是空的,但是SYN包非常重要,没有它无法建立连接,且SYN包的序列号用来协商初始序列号,所以有必要也必须对SYN包编号.

为什么传输大文件用TCP而不是TCP?

因为一个大文件稍有错误,就无法打开,那不是白搭功夫了.

拥塞控制:对资源的需求>可用资源,硬件设施不行.

解决方法是控制端口注入网络的数据或升级网络硬件设施.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5-13 | 5-14 | 5-22 | 5-23 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |