TCP传输控制协议

TCP（Transmission Control Protocol [传输控制协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)）是一种面向连接的、可靠的、基于字节流的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82)通信协议，由IETF的RFC 793定义。在简化的计算机网络OSI模型中，它完成第四层传输层所指定的功能，用户数据报协议（UDP）是同一层内[1] 另一个重要的传输协议。在因特网协议族（Internet protocol suite）中，TCP层是位于IP层之上，应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是IP层不提供这样的流机制，而是提供不可靠的包交换。[1]

应用层向TCP层发送用于网间传输的、用8位字节表示的数据流，然后TCP把数据流分区成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传输单元（[1]  MTU）的限制）。之后TCP把结果包传给IP层，由它来通过网络将包传送给接收端实体[1]  的TCP层。TCP为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（[RTT](https://baike.baidu.com/item/RTT)）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。TCP用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。[1]

网络协议

功能

当应用层向TCP层发送用于网间传输的、用8位字节表示的[数据流](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%B5%81)，TCP则把数据流分割成适当长度的报文段，最大传输段大小（[MSS](https://baike.baidu.com/item/MSS/3567770)）通常受该计算机连接的网络的数据链路层的最大传送单元（[MTU](https://baike.baidu.com/item/MTU)）限制。之后TCP把数据包传给IP层，由它来通过网络将包传送给接收端实体的TCP层。

TCP为了保证报文传输的可靠[1]  ，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的字节发回一个相应的确认(ACK)；如果发送端实体在合理的往返时延(RTT)内未收到确认，那么对应的数据（假设丢失了）将会被重传。

* 在数据正确性与合法性上，TCP用一个校验和函数来检验数据是否有错误，在发送和接收时都要计算校验和；同时可以使用md5认证对数据进行加密。
* 在保证可靠性上，采用超时重传和捎带确认机制。
* 在流量控制上，采用滑动窗口[1]  协议，协议中规定，对于窗口内未经确认的分组需要重传。

在拥塞控制上，采用广受好评的TCP拥塞控制算法（也称AIMD算法）。该算法主要包括三个主要部分：1）加性增、乘性减；2）慢启动；3）对超时事件做出反应。

实行标准

TCP/IP(Transmission Control Protocol/Internet Protocol) 即传输控制协议/[网间协议](https://baike.baidu.com/item/%E7%BD%91%E9%97%B4%E5%8D%8F%E8%AE%AE)，是一个工业标准的[协议集](https://baike.baidu.com/item/%E5%8D%8F%E8%AE%AE%E9%9B%86)，它是为[广域网](https://baike.baidu.com/item/%E5%B9%BF%E5%9F%9F%E7%BD%91)（WAN）设计的。它是由ARPANET网的研究机构发展起来的。

TCP/IP的标准在一系列称为RF[1]  C的文档中公布。文档由技术专家、特别工作组、或RFC编辑修订。公布一个文档时，该文档被赋予一个RFC编号，如RFC959（FTP的说明文档）、RFC793（TCP的说明文档）、RFC791（IP的说明文档）等。最初的RFC一直保留而从来不会被更新，[1]  如果修改了该文档，则该文档又以一个新号码公布。因此，重要的是要确认你拥有了关于某个专题的最新RFC文档。通常在RFC的开头部分，有相关RFC的更新(update)、排错（errata）、作废（obsolete）信息，提示读者信息的时效性。

首部格式

TCP的首部格式图右图所示：

---Source Port是源端口，16位。

[](https://baike.baidu.com/pic/TCP/33012/0/574e9258d109b3de49101ee8cebf6c81810a4c87?fr=lemma&ct=single)TCP首部

---Destination Port是目的端口，16位。

---Sequence Number是发送数据包中的第一个字节的序列号，32位。

---Acknowledgment Number是确认序列号，32位。

---Data Offset是数据偏移，4位，该字段的值是TCP首部（包括选项）长度除以4。[1]

---标志位： 6位，URG表示Urgent Pointer字段有意义：

ACK表示Acknowledgment Number字段有意义

PSH表示Push功能，RST表示复位TCP连接

SYN表示SYN报文（在建立TCP连接的时候使用）

FIN表示没有数据需要发送了（在关闭TCP连接的时候使用）

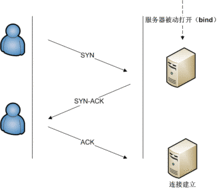
Window表示接收缓冲区的空闲空间，16位，用来告诉TCP连接对端自己能够接收的最大数据长度。

---Checksum是校验和，16位。

---Urgent Pointers是紧急指针，16位，只有URG标志位被设置时该字段才有意义，表示紧急数据相对序列号（Sequence Number字段的值）的偏移。

连接建立

TCP是因特网中的传输层协议，使用[三次握手协议](https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B%E5%8D%8F%E8%AE%AE)建立连接。当主动方发出SYN连接请求后，等待对方回答

[](https://baike.baidu.com/pic/TCP/33012/0/a1ad16fa330e9cae59ee90ca?fr=lemma&ct=single)TCP的三次握手

SYN+ACK[1]  ，并最终对对方的 SYN 执行 ACK 确认。这种建立连接的方法可以防止产生错误的连接，TCP使用的流量控制协议是可变大小的滑动窗口协议。[1]

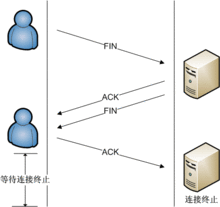
TCP三次握手的过程如下：

1. 客户端发送SYN（SEQ=x）报文给服务器端，进入SYN\_SEND状态。
2. 服务器端收到SYN报文，回应一个SYN （SEQ=y）ACK(ACK=x+1）报文，进入[SYN\_RECV](https://baike.baidu.com/item/SYN_RECV)状态。
3. 客户端收到服务器端的SYN报文，回应一个ACK(ACK=y+1）报文，进入Established状态。

三次握手完成，TCP客户端和服务器端成功地建立连接，可以开始传输数据了。

连接终止

建立一个连接需要三次握手，而终止一个连接要经过四次握手，这是由TCP的半关闭（half-close）造成的。具体过程如下图所示。[1]

[](https://baike.baidu.com/pic/TCP/33012/0/bf4875638e06d1590c33faf9?fr=lemma&ct=single)TCP连接的终止

(1) 某个应用进程首先调用close，称该端执行“主动关闭”（active close）。该端的TCP于是发送一个FIN分节，表示数据发送完毕。

(2) 接收到这个FIN的对端执行 “被动关闭”（passive close），这个FIN由TCP确认。

注意：FIN的接收也作为一个文件结束符（end-of-file）传递给接收端应用进程，放在已排队等候该应用进程接收的任何其他数据之后，因为，FIN的接收意味着接收端应用进程在相应连接上再无额外数据可接收。

(3) 一段时间后，接收到这个文件结束符的应用进程将调用close关闭它的套接字。这导致它的TCP也发送一个FIN。

(4) 接收这个最终FIN的原发送端TCP（即执行主动关闭的那一端）确认这个FIN。[1]

既然每个方向都需要一个FIN和一个ACK，因此通常需要4个分节。

注意：

(1) “通常”是指，某些情况下，步骤1的FIN随数据一起发送，另外，步骤2和步骤3发送的分节都出自执行被动关闭那一端，有可能被合并成一个分节。[2] 

(2) 在步骤2与步骤3之间，从执行被动关闭一端到执行主动关闭一端流动数据是可能的，这称为“半关闭”（half-close）。

(3) 当一个Unix进程无论自愿地（调用exit或从main函数返回）还是非自愿地（收到一个终止本进程的信号）终止时，所有打开的描述符都被关闭，这也导致仍然打开的任何TCP连接上也发出一个FIN。

无论是客户还是服务器，任何一端都可以执行主动关闭。通常情况是，客户执行主动关闭，但是某些协议，例如，HTTP/1.0却由服务器执行主动关闭。[2]

可靠性实现

可靠性

TCP提供一种面向连接的、可靠的字节流服务。面向连接意味着两个使用TCP的应用（通常是一个客户和一个服务器）在彼此交换数据包之前必须先建立一个TCP连接。这一过程与打电话很相似，先拨号振铃，等待对方摘机说“喂”，然后才说明是谁。在一个TCP连接中，仅有两方进行彼此通信。广播和[多播](https://baike.baidu.com/item/%E5%A4%9A%E6%92%AD)不能用于TCP。

TCP通过下列方式来提供可靠性：[1]

1．应用数据被分割成TCP认为最适合发送的数据块。这和UDP完全不同，应用程序产生的数据长度将保持不变。由TCP传递给IP的信息单位称为报文段或段（segment）。

2．当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。当TCP收到发自TCP连接另一端的数据，它将发送一个确认。TCP有延迟确认的功能，在此功能没有打开，则是立即确认。功能打开，则由定时器触发确认时间点。

3．TCP将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP将丢弃这个报文段和不确认收到此报文段（希望发端超时并重发）。

4．既然TCP报文段作为IP数据报来传输，而IP数据报的到达可能会失序，因此TCP报文段的到达也可能会失序。如果必要，TCP将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层。

5．既然IP数据报会发生重复，TCP的接收端必须丢弃重复的数据。[2]

6．TCP还能提供流量控制。TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。

两个应用程序通过TCP连接交换8bit字节构成的字节流。TCP不在字节流中插入记录标识符。我们将这称为字节流服务（bytestreamservice）。如果一方的应用程序先传10字节，又传20字节，再传50字节，连接的另一方将无法了解发方每次发送了多少字节。只要自己的接收缓存没有塞满，TCP 接收方将有多少就收多少。一端将字节流放到TCP连接上，同样的字节流将出现在TCP连接的另一端。

另外，TCP对字节流的内容不作任何解释。TCP不知道传输的数据字节流是二进制数据，还是ASCⅡ字符、EBCDIC字符或者其他类型数据。对字节流的解释由TCP连接双方的应用层解释。[1]  [2]

这种对字节流的处理方式与Unix操作系统对文件的处理方式很相似。Unix的内核对一个应用读或写的内容不作任何解释，而是交给应用程序处理。对Unix的内核来说，它无法区分一个二进制文件与一个文本文件。

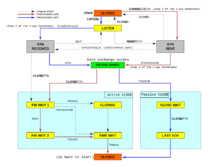
重传策略

TCP协议用于控制数据段是否需要重传的依据是设立重发定时器。在发送一个数据段的同时启动一个重传，如果在重传超时前收到确认(Acknowlegement)就关闭该重传，如[2]  果重传超时前没有收到确认，则重传该数据段。在选择重发时间的过程中，TCP必须具有自适应性。它需要根据互联网当时的通信情况，给出合适的重发时间。

这种重传策略的关键是对定时器初值的设定。采用较多的算法是Jacobson于1988年提出的一种不断调整超时时间间隔的动态算法。其工作原理是：对每条连接TCP都保持一个[2]  变量RTT（Round Trip Time），用于存放当前到目的端往返所需要时间最接近的估计值。当发送一个数据段时，同时启动连接的定时器，如果在定时器超时前确认到达，则记录所需要的时间（M），并修正[2]  RTT的值，如果定时器超时前没有收到确认，则将RTT的值增加1倍。通过测量一系列的RTT（往返时间）值，TCP协议可以估算数据包重发前需要等待的时间。在估计该连接所需的当前延迟时通常利用一些统计学的原理和算法（如Karn算法），从而得到TCP重发之前需要等待的时间值。

窗口确认

TCP的一项功能就是确保每个数据段都能到达目的地。位于目的主机的TCP服务对接受到的数据进行确认，并向源应用程序发送确认信息。[2]

[](https://baike.baidu.com/pic/TCP/33012/0/3b6833f573546965bd310948?fr=lemma&ct=single)TCP的状态机

使用数据报头序列号以及确认号来确认已收到包含在数据段的相关的数据字节。

TCP在发回源设备的数据段中使用确认号，指示接收设备期待接收的下一字节。这个过程称为**期待确认**。[2]

源主机在收到确认消息之前可以传输的数据的大小称为**窗口大小**。用于管理丢失数据和流量控制。这些变化如右图所示。[2]

配置TCP

**修改建立TCP连接的超时时间**

建立TCP连接需要经过三次握手：主动端先发送SYN报文，被动放回应SYN+ACK报文，然后主动端再回应ACK。[2]

l在主动端发送SYN后，如果被动端一直不回应SYN+ACK报文，主动端会不断的重传SYN报文直到超过一定的重传次数或超时时间。[2]

l在主动端发送SYN后，被动端回应SYN+ACK报文，但主动端不再回复ACK，被动端也会一直重传直到超过一定的重传次数或超时时间。（SYN报文攻击会出现这种情况）[2]

可以通过以下命令配置SYN报文的超时时间（发送SYN报文到三次握手成功的最大时间），也就是建立TCP连接的超时时间。

|  |  |
| --- | --- |
| 命令 | 作用 |
| R(config)#**ip tcp syntime-out***seconds* | 修改建立TCP连接的超时时间。  单位秒，取值范围5-300，缺省值20 |

使用**no ip tcp syntime-out**命令恢复参数缺省值。

**修改缓冲区大小**

TCP的接收缓冲区是用来缓存从对端接收到的数据，这些数据后续会被应用程序读取。一般情况下，TCP报文的窗口值反映接收缓冲区的空闲空间的大小。对于带宽比较大、有大批量数据的连接，增大接收缓冲区的大小可以显著提供TCP传输性能。TCP的发送缓冲区是用来缓存应[2]  用程序的数据，发送缓冲区的每个字节都有序列号，被应答确认的序列号对应的数据会从发送缓冲区删除掉。增大发送缓冲区可以提高TCP跟应用程序的交互能力，也因此会提高性能。但是增大接收和发送缓冲区会导致TCP占用比较多的内存。

|  |  |
| --- | --- |
| 命令 | 作用 |
| R (config)#**ip tcp window-size** *size* | 修改TCP连接的接收和发送缓冲区大小。  单位字节，取值范围0-65535，缺省值4096。 |

使用**no ip tcp window-size**命令恢复接收和发送缓冲区大小为缺省值。

**禁止端口不可达时的重置报文**

TCP模块在分发TCP报文时，如果找不到该报文所属的TCP连接会主动回复一个reset报文以终止对端的TCP连接。攻击者可能利用大量的端口不可达[2]  的TCP报文对设备进行攻击。

可以使用以下命令禁止/恢复在收到端口不可达的TCP报文时发送reset报文。

|  |  |
| --- | --- |
| 命令 | 作用 |
| R (config)#**ip tcp not-send-rst** | 禁止在接收到端口不可达的TCP报文时发送reset报文。 |

使用**no ip tcp not-send-rst**命令恢复发送reset报文。

**限制TCP连接的MSS的最大值**

MSS是最大传输段大小的缩写，指一个TCP报文的数据载荷的最大长度，不包括TCP选项。

在TCP建立连接的三次握手中，有一种很重要的工作那就是进行MSS协商。连接的双方都在SYN报文中增加MSS选项，其选项值表示本端[2]  最大能接收的段大小，即对端最大能发送的段大小。连接的双方取本端发送的MSS值和接收对端的MSS值的较小者作为本连接最大传输段大小。

发送SYN报文时的MSS选项值的计算方法如下。

l非直连网络中：mss = 默认值[2]  536。

l直连网络中：mss = 对端ip地址对应的出口的MTU - 20字节ip头 - 20字节tcp头。  
　　一般来说如果出口配置的某些应用影响了接口的mtu，那么该应用会相应的设置mtu，如隧道口，vpn口等。

到这里得到的rmss值就是要发送的syn报文mss选项的值[2]  。举例：一般情况下在直连网络中建立bgp邻居，那么该连接的发送的mss为1500 – 20 – 20 – 20 = 1440。

**ip tcp mss**命令的作用是限制即将建立的TCP连接的MSS的最大值。任何新建立的连接协商的MSS值不能超过配置的值。

|  |  |
| --- | --- |
| 命令 | 作用 |
| R (config)#**ip tcp mss***max-segment-size* | 限制TCP连接的MSS的最大值。  单位为字节，取值范围68-10000。 |

使用**no ip tcp mss**命令取消此限制。

**启用PMTU发现功能**

TCP的路径最大传输单元（PMTU）发现功能是按RF[2]  C1191实现的，这个功能可以提高网络带宽的利用率。当用户使用TCP来批量传输大块数据时，该功能可以使传输性能得到明显提升。

|  |  |
| --- | --- |
| 命令 | 作用 |
| R(config)#**ip tcp path-mtu-discovery** [ **age-timer** *minutes***| age-timer infinite**] | 启用PMTU发现功能。  **age-timer***minutes*：TCP在发现PMTU后，重新进行探测的时间间隔。单位分钟，取值范围10-30。缺省值10。  **age-timer infinite**：TCP在发现PMTU后，不重新探测。 |

按RFC1191的描述，TCP在发现PMTU后，隔一段时间可以使用更大的MSS来探测新的PMTU。这个时间间隔就是使用参数**age-timer**来指定。当设备发现的PMTU比TCP连接两端协商出来的MSS小时，设备就会按上述配置时间间隔，去尝试发现更大的PMTU。直到PMTU达到MS[2]  S的值，或者用户停止这个定时器，这个探测过程才会停止。停止这个定时器，使用**age-timer infinite**参数。

使用**no ip tcp path-mtu-discovery**命令关闭PMTU发现功能。

**设置接口收发SYN报文的MSS选项值**

当客户端发起一个TCP连接时，它通过TCP SYN报文中的MSS选项字段协商TCP报文数据载荷的最大值，客户端SYN报文的MSS值表示后续服务器端发送TCP报文数据载荷的最大值，反之同理。

[](https://baike.baidu.com/pic/TCP/33012/0/314e251f95cad1c8411218a77d3e6709c83d5181?fr=lemma&ct=single)

如右图的拓扑，PC用http访问服务器可能会出现无法访问的情况。因为PC与服务器端建立的连接MSS协商的都会是1460，但1460的MSS无法通过R1和R2，R1和R2用隧道相连，MTU小于1500。[2]

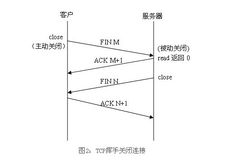
这时可以通过在R2的（1）口和（2）口上配置如下命令，修改SYN报文中的MSS选项值。从而修改经过（1）口和（2）口的TCP连接协商的MSS值。

|  |  |
| --- | --- |
| 命令 | 作用 |
| R (config-if)# **ip tcp adjust-mss***max-segment-size* | 设置接口收发SYN报文的MSS选项值。  单位为字节，取值范围500-1460。 |

使用**no ip tcp adjust-mss**命令取消此项设置，则接口收发SYN报文时，不会修改报文的MSS选项值。

在接口上配置本命令会使得该接口接收或发送SYN报文的MSS选项都被改为接口上配置的MSS值。建议出口和入口配置相同的值。如果SYN报文的入口和出口配置了不同的MSS值，经过该设备后，SYN报文的MSS选项被改为这两个口配置值的较小者。[2]

词条图册[更多图册](https://baike.baidu.com/pic/TCP/33012?fr=lemma)

[[](https://baike.baidu.com/pic/TCP/33012/0/4610b912c8fcc3cea74598b29045d688d53f20ad?fr=lemma)](https://baike.baidu.com/pic/TCP/33012/0/4610b912c8fcc3cea74598b29045d688d53f20ad?fr=lemma" \o "" \t "_blank)

怎么理解TCP的面向连接和UDP的无连接（不面向连接）？

原创|发布：2016-12-06 13:00:52    更新：2016-12-06 13:00:52

阅读 1459赞 91

*如题，在理解TCP和UDP中，一个是面向连接的传输协议，一个是不面向连接的协议。怎么能够更好的理解他们的不同。*



亚当和夏娃分别生活在两个山头，山头之间是万丈深渊，亚当采集野果需要分享给夏娃，如果他们之间有一条索道（**物理连接**），野果可以顺着索道滑到夏娃那一边，那就没有车小胖什么事了。

事实上山头之间没有索道，但是亚当何等聪明，于是他想出了一个方法，假设亚当需要给夏娃10个野果，否则她会饿死。

连接建立

亚当对着夏娃大喊：爱妃，你听得到吗？

夏娃回应：孩他爹，我听得到！

亚当接着喊：那好，我扔果子给你吃，你接到果子就喊一声，一共十个。

运送货物

于是亚当开始扔第一个，夏娃喊收到了一个。

亚当扔第二个，夏娃喊收到两个。

**超时重传 （ timeout retransmit)**

亚当扔第三个，可是夏娃迟迟没有回音，亚当意识到可能果子落到悬崖了，于是重新扔，夏娃喊收到第三个。

**Advertised window size = 0**

于是亚当连续扔了第四、五、六个，夏娃急了：孩他爹，慢点扔，臣妾忙不过来了…

**Advertised window size > 0**

于是亚当坐下休息，爱妃又开始叫了：继续扔吧。

亚当开始扔第七个，夏娃喊收到七个。

…

关闭连接

终于亚当扔完了，亚当喊：爱妃，果子扔完了，寡人去忙别的了。

夏娃回复：好的，我也休息一下，再见

亚当：再见

以上的过程类似TCP连接的过程，TCP是一个虚拟连接。

**何为虚拟连接？**

和物理连接所对应，物理连接是实实在在存在的，看得见摸得着，比如索道。而虚拟连接是不存在的，看不见摸不着，通过双向的消息、消息确认来模拟物理连接。

由于有确认机制，亚当可以确保夏娃可以收到10个果子。

**那接下来再谈谈什么是无连接的UDP？**

亚当和夏娃吵架了，任凭亚当如何大声喊，夏娃躲在树林后生闷气，一声不响，亚当害怕夏娃饿死，于是开始自说自话朝着夏娃的山头扔玉米棒子：

一个、两个、三个…

一共扔了十个，但最终扔到对方山头到底有几个，亚当没有底，也许有的玉米棒子落到悬崖了，但是这个效率高啊，可以连续扔，以前扔10个果子需要一分钟，现在只需要20秒。

亚当扔果子、扔玉米都有可能扔到悬崖下，但是扔果子为何可以确保对方收到十个？那是因为夏娃收到一个果子，然后喊收到了，如果没有收到，亚当就重新扔，直到夏娃说收到了。而扔玉米棒子对方没有确认，所以对于丢弃的情况无法知道，也无法重新扔。

通信协议

▪ 协议

▪ 协议参考模型

▪ 协议控制信息

▪ 协议转换▪ 开放系统互连

▪ 协议转换器

▪ 物理层

▪ 表示层▪ 开放系统互连参考模型

▪ 应用层

▪ 会话层

▪ 网络层▪ 传送层

▪ 数据链路层

▪ 文件传送协议

▪ 通用路由封装▪ 分组数据协议

▪ 用户数据报协议

▪ 传输控制协议

▪ IPv6协议▪ IPv4协议

▪ 因特网协议

▪ 链路协议

▪ 接入协议▪ 选路协议

▪ 应用协议

▪ 超文本传送协议

▪ 点到点协议▪ 隧道协议

▪ 点到点隧道协议

▪ 地址解析协议

▪ 网络地址转换