六 传输层

传输层是TCP/IP协议栈中的一层，它对上层提供了可靠的端到端数据传输服务。传输层的主要功能如下：

1. 提供端到端的可靠数据传输服务。传输层使用TCP协议对数据进行分段、传输、重组和确认，以确保在网络上数据的可靠传输。
2. 提供端到端的无连接数据传输服务。传输层使用UDP协议提供一种简单的无连接数据传输服务，适用于不需要确认和重传的应用。
3. 提供多路复用和分用服务。传输层可以将多个应用程序的数据流通过一个端口进行复用和分用，实现多个应用程序共享一个网络连接。
4. 提供流量控制和拥塞控制。传输层使用TCP协议对传输的数据进行流量控制，使得发送方不会因为发送速度过快而过载接收方；同时，TCP协议还具备拥塞控制机制，能够在网络拥塞时自适应调整发送速度，防止网络出现拥堵。

总之，传输层提供了一系列服务来确保网络上的数据可以有效地传输，使应用程序能够实现端到端的可靠和高效的数据交换。

流量控制

TCP流量控制是指通过动态调整TCP发送方发送数据包的速率，来避免网络拥塞和数据丢失的现象。TCP流量控制的主要思想是，发送方在发送数据之前，与接收方协商一个窗口大小（即接收缓冲区大小），发送方在发送数据时，要根据接收方当前的状态和能够接收的数据量来动态调整发送速度

TCP拥塞控制是指通过监测网络传输时的拥塞状态，动态地调整TCP发送方的发送速率，以避免网络拥塞和数据丢失。TCP的拥塞控制算法主要有三种，分别是慢启动、拥塞避免和拥塞控制。

- 慢启动：当TCP连接建立时，发送方将初始拥塞窗口设置为一个很小的数值，然后每次接收到一个确认应答后，拥塞窗口大小就会加倍，直到拥塞窗口大小达到一个阈值。这个阈值被称为拥塞避免阈值，在这个值之前，TCP是以指数级别增长的方式增加发送窗口。
- 拥塞避免：当拥塞窗口大小超过了拥塞避免阈值时，TCP的发送方就会变成线性增长，每次成功的确认应答只能增加一个MSS（最大报文长度）。这样可以尽可能地利

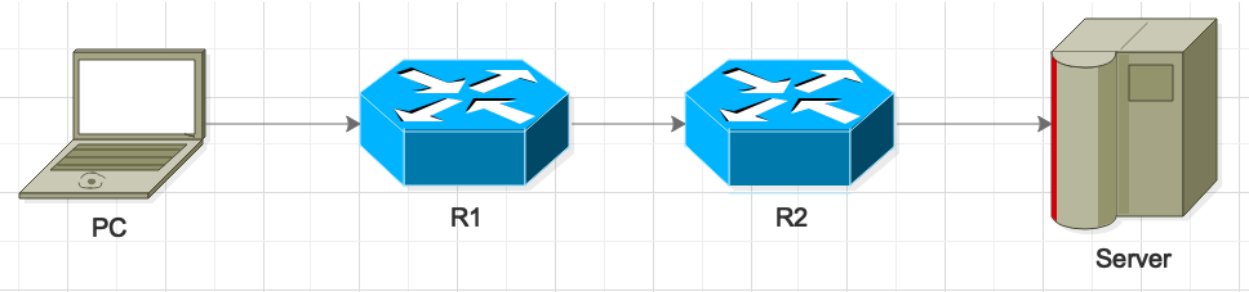
用网络的带宽，同时避免网络由于过度拥塞而崩溃。

- 拥塞控制：当发生网络拥塞时，TCP需要将发送方的发送速率降低，以避免继续加重网络的拥塞。在网络拥塞的情况下，TCP的发送方会减小拥塞窗口的值，并且把拥塞避免阈值设为当前拥塞窗口的一半。当拥塞已经缓解时，TCP发送方会逐渐增加拥塞窗口大小，以恢复到正常的发送速率。

拥赛控制

TCP拥塞控制是指通过监测网络传输时的拥塞状态，动态地调整TCP发送方的发送速率，以避免网络拥塞和数据丢失。

上面提到的端到端(End to End)通信和点到点(Point to Point)通信，下图为常见的网络通信场景，



客户端 PC 发送请求到服务器端 Server，而PC 和 Server 位于不同的网络，PC 请求到达 Server 需要经过路由器 R1 和 路由器 R2，那么 在这个场景中发生的点对点通信包括：

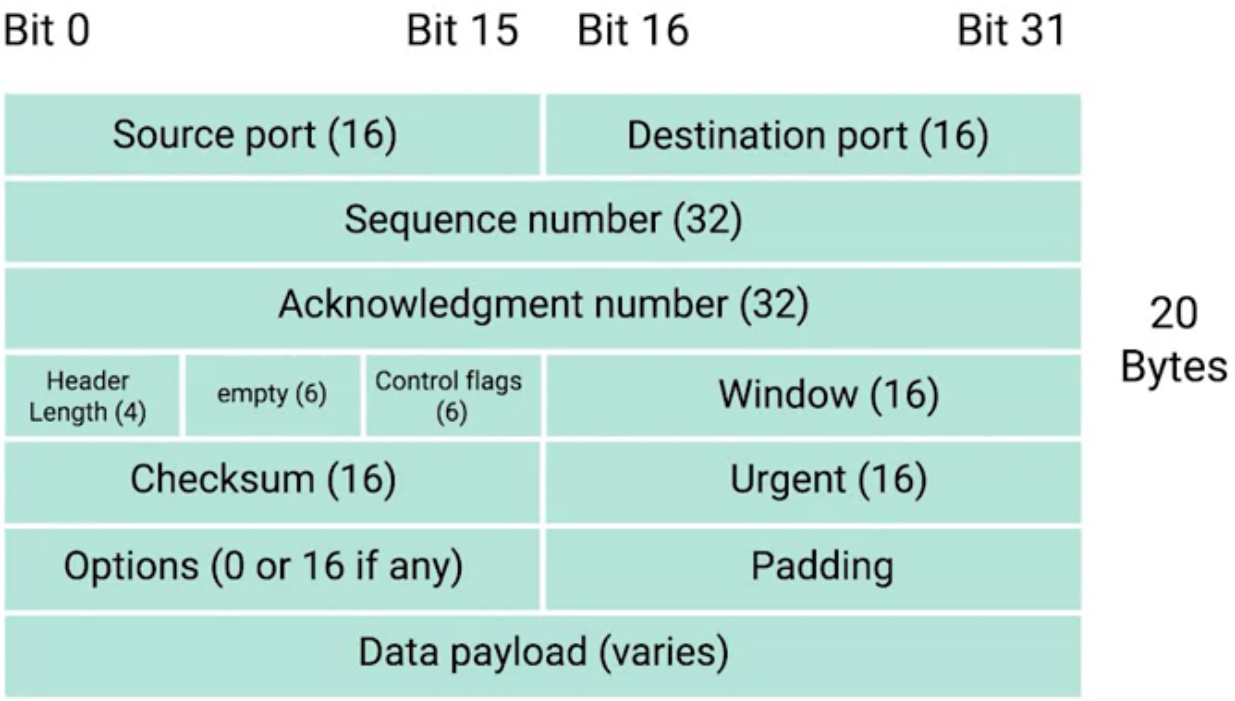
- PC → R1
- R1 → R2
- R2 → Server

端到端的通信只有一个，即 PC → Server。

如前面的内容描述，数据链路层可确保点对点的网络传输可靠，网络层可让数据包在不同的网络之间转发，而网络模型中的传输层负责的是**端到端的可靠网络通信**。

TCP Segment 结构

一个 TCP Segment 是有 TCP 头和数据部分构成。



- **Destination port** - 目的地端口，是目的服务所监听的端口，是最终接收TCP Segment 的服务的端口。
- **Source port** - 源端口，是一个较大的数字，发送 TCP Segment 的客户端从随机端口中随机选择
- **Sequence number** - 32 bit长度，用来跟踪 TCP Segment 在传输序列中预期的位置。
- **Acknowledgment number** - 32 bit长度，用来确定下一个期望的 TCP Segment。
- **Header Length** - 长度为 4 bit，也叫数据偏移字段，它定义了在一个 TCP Segment 中 TCP 头的长度，这也使接收端的网络设备知道真正数据 负载开始的位置。
- **Control flags** - TCP Segment 控制标签。
- **TCP window** - 16 bit 长度的数字，指定在需要确认前可能发送的序列号范围。
- **Checksum** - 长度为 16 位字节，和 IP、Ethernet 中的 Checksum 字段类似，当接收者接收到这个 TCP Segment 后，Checksum 会进行一次计算，计算整个 TCP Segment 的长度，并和改字段定义的长度进行比较，以确保传输的过程中没有数据的丢失或损坏。

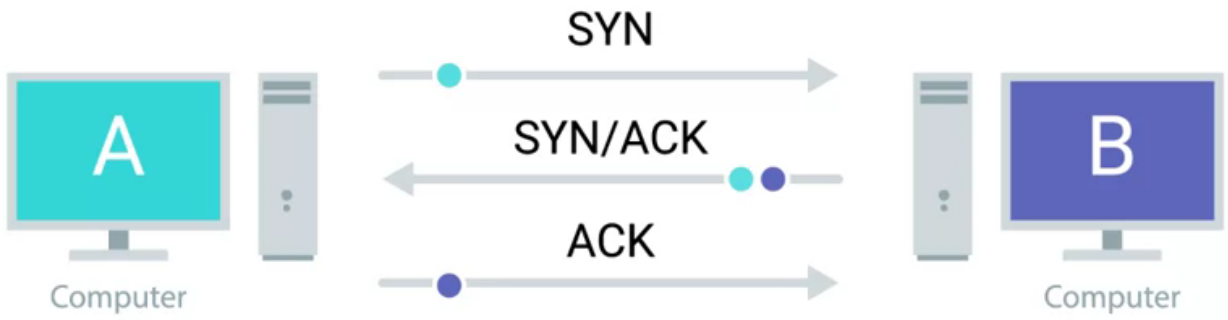
- **Urgent** - 该字段通常与 TCP 控制标签中的某个标签联合使用，来说明某个 Segment 比其他 Segment 重要，或有特定含义。
- **Options** - 该字段通常比较少用，但有时会用于更复杂的流控制协议。
- **Padding** - 零序列，以确保数据有效负载部分从预期位置开始。

TCP 控制标签

名称	描述
URG(urgent)	如果此标签值为 1，则表示当前 TCP Segment 特别重要，该标签通常 与 TCP 头中的 **Urgent** 字段一起使用， **Urgent** 字段有更多信息。
ACK(acknowledge)	如果此标签值为 1，则表示 **Acknowledgment number** 字段应该被检查。
PSH(push)	传输设备想 让接收端设备尽快将缓冲中的数据推送到应用。
RST(reset)	TCP 连接中的一方无法从一系列丢失或格式错误的段中正确恢复。
SYN(synchronize)	初次建立一个 TCP 连接时使用，让接收端知道需要检查 **Sequence number** 字段。
FIN(finish)	提示传输计算机端没有更多数据，连接可以关闭。

三次握手

如下图所示，TCP 连接的建立至少需要交换三个 TCP Segment，三次握手是对 TCP 连接建立的一个抽象。




```
// Step One
A 发送一个 TCP Segment 到 B，主要包括一个 SYN 标签，告诉 B 客户端 A 的初始序列号为 J。（让我们开始建立连接吧，我的序列号为 J，这样我们会知道我们交流开始的位置）
A sends a TCP segment to B with SYN flag set (`Let's establish a connection and look at my sequence number field, so we know where this conversation starts.`)

// Step Two
B 回复一个 TCP Segment 到 A，包括两个标签 SYN 和 ACK，告诉 A 服务器端（B）的初始序列号为 K，同时确认 A 服
```

```
务器（B）确认客户端 A 的序列号（ACK 的值为 J + 1）
B then responds with a TCP segment, where both the SYN and ACK flags are set(`Sure, let's establish a connection and I acknowledge your sequence number.`)

// Step Three
A 回复一个 TCP Segment 到 B，主要包括一个 ACK 标签，告诉服务器端 B 客户端 A 确认服务端的序列号（ACK 的值为 K + 1）。
A responds again with just the ACK flag set* (`I acknowledge your acknowledgement. Let's start sending data.`)
```

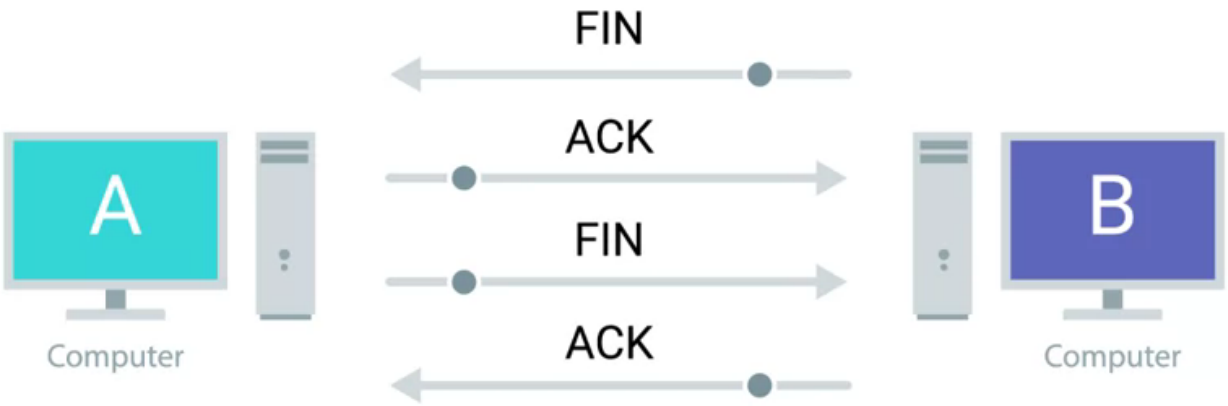
一次握手是两个设备确保他们所使用同一个协议，并且能够彼此相互理解。

TCP 连接是一个多路复用的模式，每一个 TCP Segment 的发送都会有一个 TCP Segment 的回复（ACK 标签），这样发送端就知道接收端接收到相应的片段。

四次握手

TCP 建立一个连接需要三次 TCP Segment 交换，而终止一个连接需要四次 TCP Segment 交换，四次握手是对 TCP 连接终止的一个抽象。

TCP 连接可以从客户端和服务器端的任何一方发起，发起关闭连接的一次通常通过调运 close() 方法，我们将这一动作称为**主动关闭（Active Close）**，相应的另一端则称为**被动关闭（Passive Close）**，下图演示的是从服务器端 B 发起的主要关闭过程：



1. B 发起fin位为1的FIN报文，此时客户端进入FIN_WAIT_1状态
2. A 接受到FIN 报文后，发送ack应答报文，此时 A 进入close_wait状态
3. B 接受到ack应答报文后，进入FIN_WAIT_2状态
4. A 处理完数据后，向 B 发送FIN报文，此时 A 进入LAST_ACK状态
5. B 接受到FIN报文后，B 发送应答ack报文，进入TIME_WAIT阶段
6. A 接受到ack报文后，断开连接，处于close状态

7. B 过一段时间后，也就是2MSL后，进入close状态

TCP 套接字状态

一个 TCP 套接字是一个潜在 TCP 连接一端的实例化，实例化。TCP 套接字有多个状态。

名称	描述
LISTEN	一个 TCP 套接字准备就绪，可以接收进入的连接，这个状态只会在服务器端。
SYN_SENT	客户端发送了一个 SYN 标签的请求到服务器端，且连接建立还没有完成，这个状态只会在客户端。
SYN_RECEIVED	前序处于 LISTEN 状态，接收到 SYN 标签的请求，并且给客户端回复了 SYN 和 ACK，但是连接还没有建立，等待客户端的 ACK 请求。这个状态只会在服务器端。
ESTABLISHED	TCP 连接建立后的状态，客户端和服务器可以自由相互发送数据，这个状态即可以是客户端，也可以是在服务器端。
FIN_WAIT	一个 FIN 标签的请求发送，同时没有接收到另一侧回复的 ACK。
CLOSE_WAIT	传输层 TCP 连接已经关闭, 但应用层还没有释放相应的套接字。
CLOSED	TCP 连接完全关闭，没有任何进一步通信的可能性。

使用 Netstat 命令查看套接字状态

Linux 中使用 netstat命令查看 State 列

```
$ netstat -nplat
  Active Internet connections (servers and established)
  Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Prog
ram name
  tcp        0      0 127.0.0.1:43375         0.0.0.0:*               LISTEN      765/cont
ainerd
  tcp        0      0 0.0.0.0:9999            0.0.0.0:*               LISTEN      787/ngin
x: master p
  tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      657/syst
emd-resolve
  tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      855/ssh
d: /usr/sbin
  tcp        0    144 10.0.2.15:22            10.0.2.2:58513          ESTABLISHED 2295/ssh
d: vagrant
  tcp6       0      0 :::22                   :::*                    LISTEN      855/ssh
d: /usr/sbin
  tcp6       0      0 :::8888                 :::*                    LISTEN      978/(squ
```

```
id-1)
  tcp6       0      0 :::9090                 :::*                    LISTEN      1/init
```

UDP(用户数据报协议)

不像 TCP，UDP 不依赖一个连接，没有类似 TCP 中 ACK 的理念，UDP 中只需要设定一个目的地端口。

系统端口和临时端口

传输层是根据端口号来确保端到端的通信，传输层的协议不管是 TCP 还是 UDP，都与端口号关联，端口号是一个 16 个字节长度的数字（范围为 0 - 65535）。端口号又分为系统端口和临时端口。

范围	描述
0	端口 0 不会使用在网络连接中，但有时候如果同一个主机上又多个程序，那么使用 0 可以随机选择一个端口。
1 - 1023	系统端口范围，或被称为众所周知端口号，这些端口通常被一些大家熟知的服务所有，例如 80 为 HTTP 端口，21 是 FTP 等。
1024 - 49151	已注册端口，它们松散地绑定于一些服务。也就是说有许多服务绑定于这些端口，例如，3306 是 Mysql 的端口，8080 为 Tomcat/JBoss 端口。
49152 - 65535	这些端口被称为私有或临时端口，这些端口用在 TCP 连接的客户端随机选用（Source Port），一个客户端/服务器端通信的程序，服务器端通常监听与一个已注册的端口，客户端建立一个连接时会分配一个临时端口。