# 自顶向下的计算机网络 传输层

## 运输层概述

运输层在不同主机上的进程提供了**逻辑通信（logic communication）**，从应用程序的角度看，运行不同进程的主机似乎在直接相连。

运输层协议是在端系统而不是在路由器实现。**运输层的分组单位为报文段**（segment），实现的方法是将应用层的报文划分为小块，并且为**每一个小块加上运输层首部**，生成运输层报文段。

### 1.1 运输层和网络层的关系：

在协议栈protocol stack中，运输层在网络层之上。**网络层提供主机之间的逻辑通信，而运输层提供不同主机上的进程之间的通信。**

在分组传输时，运输层把报文段通过API下放到网络层，网络层对报文段进行封装成为数据报（datagram），通过路由器在链路上进行转发。**中间路由器不处理或识别运输层加在应用层中的信息，运输层的首部也无法规定这些报文段如何在网络中移动。**

**运输层协议受底层网络层协议的约束**，如果网络层协议无法为端系统之间报文段传输提供时延和带宽的保证，运输层协议就会无法对进程之间发送的报文进行保证。

但是，即使网络层协议不能提供相应的服务，运输层协议也能够提供某些服务，如果网络层的协议会使得某些分组丢失、篡改或冗余，运输层协议也能提供可靠数据服务。底层网络层如果不能保证报文段的机密性，运输层协议也可以保证应用程序的报文不被入侵者读取。

### 1.2因特网运输层概述

根据RFC文档，运输层分组叫做报文段，TCP分组也叫做报文段segment；UDP分组叫做数据报，网络层分组也叫数据报datagram。所以这里统一把TCP和UDP分组都叫做报文段。

在讨论TCP和UDP协议前，需要**简单介绍一下网络层**。网络层使用的协议主要是IP协议（网际协议），为主机提供了逻辑通信。I**P服务的模型为尽努力交付服务(best-effort delivery service)**。IP服务无法保证报文段的交付，无法保证报文段到达顺序和数据完整性，所以也被称为**不可靠服务(unreliable service).** （“也”是因为UDP也是这样）顺带一提, 每台主机至少有一个网络层地址（IP地址。）

回到运输层。**UDP和TCP的基本责任是把两个端系统之间的IP交互服务拓展为运行在端系统的两个进程之间的交付服务。**将主机间交付拓展为进程间的交付叫做运输层的**多路复用**和**多路分解**。

TCP和UDP可以通过报文段首部中差错检查字段提供完整性检查，**数据交付和差错检测是运输层最低限度的两种服务，也是UDP仅能提供的两种服务**。

而TCP除了可靠数据传输，差错检测，还提供了拥塞控制。为了更好理解TCP协议这些特性，需要先介绍可靠数据传输和拥塞控制。

### 1.3多路复用和多路分解

一台主机一般来说会同时进行多个应用层通信。比如在访问一个Web服务器的同时，可能还在使用FTP和telnet，多个应用层进程都需要通过套接字向运输层，并且一个进程都有一个或者多个套接字。

当讨论接受主机怎样将一个到达的运输层报文定向到适当的套接字，在接收端，运输层检查这些字段，标识出接收套接字，进而将报文段定位到该套接字。将运输层报文中的数据交付到正确的套接字的工作叫做**多路分解(demultiplexing)**。在源主机不同套接字中收集数据块，为每个数据块添加首部信息从而生成报文段，然后将报文传输到网络层的工作过程叫做**多路复用(multiplexing)。**

通过上述讨论，得知**多路复用multiplexing有要求：**

1. 套接字socket有唯一标志符
2. 每个报文段有特殊字段来指示需要交付的套接字，这些特殊字段为**源端口号字段（source port number filed）**和**目的端端口号（destination port number filed）**.端口号是16位比特的数，0-1023是周知端口号(well-know port)，这些端口留给了周知的应用层程序使用。

#### 无连接的多路复用

假设主机A的某进程需要通过UDP端口19157往主机B进程的UDP端口46428发送报文。

主机A的运输层根据上层报文创建报文段（包含应用层数据、源端口和目的端口），然后运输层把报文段传输给网络层。网络层将报文段封装到IP数据报中，尽力交付给目的主机。

如果报文段到达主机B，接受主机的运输层检查该报文段的端口号，并且交付给UDP端口46428。

**一个UDP套接字是一个由二元组全面标识的，该二元组包含了一个目的IP地址和一个目的端口号。**因此，如果两个UDP报文段有不同的源IP地址和源端口号，但是**有相同的目的IP地址和目的端口，这俩报文段会通过同一个套接字进入相同的目的进程。**

那么源IP和端号口有什么用呢？

用作“返回地址”的一部分。当B需要回发报文段给A时，B到A的报文段需要通过的端口号取决于之前A发送到B时用的IP地址和端口号。

#### 面向连接的多路复用

与UDP的二元组不同，**TCP是由四元组标识（源IP地址，源端口号，目的IP地址，目的端口号）**。所以两个具有不同的源IP地址和源端口的报文段会通过不同的套接字，除非TCP报文段携带了创建初始链接的请求。

**服务器主机支持多个并行的TCP套接字，每个套接字和一个进程联系，并且由四元组来标识每个套接字**。

#### Web服务器和TCP

关于Web和如何使用它们的端口：

如今高性能服务器通常只有一个进程，为每个客户链接创建一个具有新连接套接字的新线程，对这样的服务器，任意给定的时间都可能有具有不同标识的套接字连接到相同的进程。

如果客户与服务器使用持续连接的HTTP，则整个期间由同一个服务器套接字交换HTTP报文。

### 1.4 UDP

**UDP只做最少的工作：多路复用/分用（分组交付），和差错检测。**

正如之前提及，UDP从应用层得到报文时进行拆分，加头部数据封装称为报文段时，只附加上源端端口号、目的端口号和其他两个字段。然后通过API把数据传给网络层，通过IP协议尽努力交付数据报给对方主机。

#### UDP的特性：

##### 应用层对于数据的内容和何时发送控制更加精细

采用UDP时，应用层只需要将数据下方，UDP会封装成报文段并且立刻发送给网络层传输。

而TCP不一样，它有拥塞控制的特性，链路拥塞会遏制TCP发送方，并且TCP发送方需要重复确认：多次发送数据报文段直到收到对方确认报文。所以在需要报文段不延迟的传输，并且能够容忍一些内容丢失时，UDP显然是更好的选择。

##### 无连接性

UDP不会引入建立链接的时延，**于TCP协议不同，UDP并没有三次握手阶段，所以UDP是无连接协议。**这也是DNS使用UDP的主要原因（否则使用TCP会很慢）

##### 无连接状态

TCP需要在端系统中维护链接状态，需要跟踪收发缓存，拥塞控制，序号和确认号的参数，这些信息是必要的。而UDP不需要维护链接状态，也不跟踪这些参数。

##### 分组首部开销小

UDP仅有8个字节，而TCP需要20个。

UDP的使用是可能可以实现可靠数据传输的，这可通过应用程序自身的可靠性来实现。比如增加确认与重传机制。

目前常见使用TCP和UDP的应用层协议如下图所示：

表格

描述已自动生成

#### 表格 描述已自动生成UDP报文段结构：

UDP报文段结构如图所示。

除去应用层报文，UDP在首部在插入源端端口号和目的端

口号时，还有两个字段：长度和检验和。

首部长32比特，四个字段，首部每个字段由两个字节

构成（8\*2+8\*2）。

通过目的端口号可以，

使得目的主机把报文传输给对应套接字（多路分解）

长度指示UDP报文段总长度（首部+应用层报文）

接收方使用检验和检查报文段是否有差错（差错检测）

源端端口号作为返回地址的一部分（需要回信时使用）

#### UDP检验和

检验和字段占2个字节，即16比特。发送方的UDP对报文段中的16比特字的和进行反码运算，求和时任何溢出都算作反卷。以下列16比特字为例子：

表格

低可信度描述已自动生成

最后一次的出来的16比特字就是检验和。而接收方，全部的4个比特字相加，如果分组没有差错，则接收方的和应该为1111111111111111。

**为什么UDP需要做差错检测？**因为在链路层运输时，链路层无法保证所有链路都提供差错检测功能；并且即使在链路上正确传输，途中经过路由器内存时，也可能引入比特差错。而传输层端到端的通信必须要保证差错检测，UDP就必须在端到端的基础上在运输层提供差错检测功能。

不过即使UDP能够检测出差错，**它也没有差错修复的功能**，它可能会丢弃受损的报文段，或者对应用程序发送警告。

### 可靠数据传输原理

图示

描述已自动生成

图3.8表示了可靠数据传输的框架，上层数据借助一条可靠信道进行传输，所有的数据都会按序到达，不会受到损坏，实现这种传输的就是**可靠数据传输协议(reliable data transfer protocol)**, 但是它的下层协议(网络层的IP)是不可靠的，所以实现这种协议其实相对困难。

所以我们需要考虑在底层信道能够损坏或者丢失分组时，需要什么协议来解决。本节只考虑**单向数据传输**。

#### 构造可靠数据传输协议

##### 情况1. 经底层完全可靠信道传输：

图示

描述已自动生成

上图显示了发送端和接受端的**有限状态机（finite-state machine, FSM）**

**发送端**接收高层数据，并且产生一个包含该数据的分组 packet = make\_pkt(data)，然后将分组发送到信道中udt\_send(packert)

**接受端**接收底层数据，rdt\_rcv（data），并且将数据进行解压提取extract(packert,data)，并且将数据传输给上层。

在完全可靠的信道下，接收方不需要反馈任何消息给发送方，因为数据就不会被丢弃或者更改。并且假设接收方的接收速率和发送方的发送速率一致，接收方也不用告诉发送方发送慢一些。

##### 情况2. 经过有比特差错信道的可靠数据传输

在有比特差错信道传输时，为了保证报文的可靠传输，此时需要用到**肯定确认(positive acknowledgement)**和**否定确认(negative acknowledgement)**, 这些控制报文可以是的接收方让发送方知道，那些内容正确接受到了，那些内容需要重新发送。所以这里就是用了**自动充传请求(automatic repeat request, ARQ)协议**

ARQ协议需要另外三种协议支撑：