Linux网络协议和管理

一.网络设备基本知识

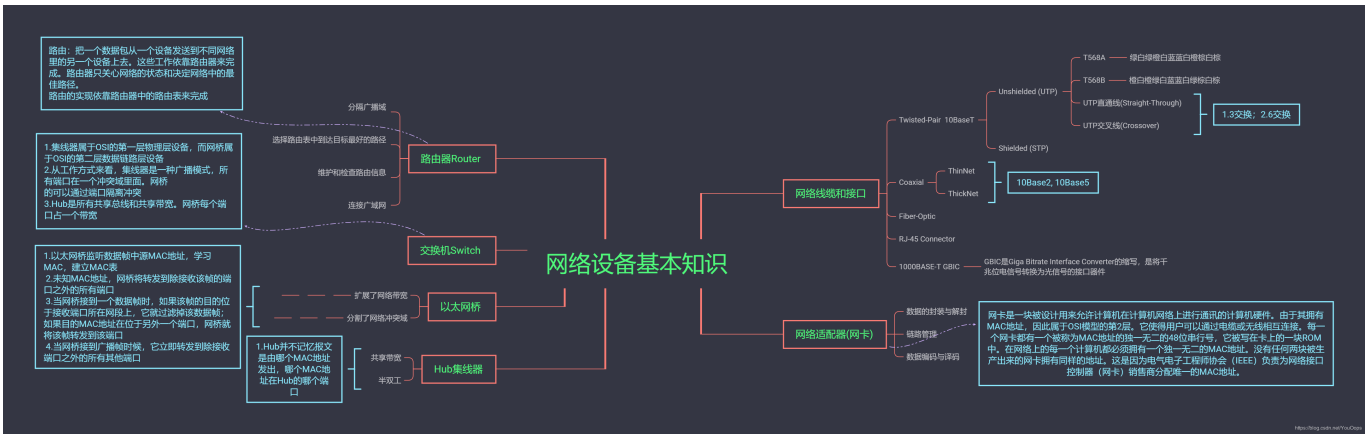


图1-网络设备基本知识

二.TCP/IP协议栈简介

1.概述

- 网络协议通常工作在不同的层中，每一层分别负责不同的通信功能。一个协议族， 比如TCP/IP，是一组不同层次上的多个协议的组合。TCP/IP通常被认为 是一个四层协议系统。如图2.1所示。
- 其每一层负责不同的传输任务：
  - 1)数据链路层，有时也称网络接口层，通常包括操作系统中的设备驱动程序和计算机 中对应的网络接口卡。它们一起处理与电缆（或其他任何传输媒介）的物理接口细节。 (ISO网络7层模型中最底层为物理层，其为启动、维护和关闭物理链路规定了： 电器特性、机械特性、过程特性和功能特性)
  - 2)网络层，有时也称作互联网层，其主要负责处理分组在网络中的活动，例如分组的选路。在TCP/IP协议族中，网络层协议包括IP协议（网际协议），ICMP协议（Internet互联网控制报文协议），以及IGMP协议（Internet组管理协议）。
  - 3)运输层主要为两台主机上的应用程序提供端到端的通信。在TCP/IP协议族中，有两个 互不相同的传输协议：TCP（传输控制协议）和UDP（用户数据报协议）。 TCP为两台主机提供高可靠性的数据通信。它所做的工作包括把应用程序交给它的数据分 成合适的小块交给下面的网络层，确认接收到的分组，设置发送最后确认分组的超时时钟 等。由于运输层提供了高可靠性的端到端的通信，因此应用层可以忽略所有这些细节。 而另一方面，UDP则为应用层提供一种非常简单的服务。它只是把称作数据报的分组 从一台主机发送到另一台主机，但并不保证该数据报能到达另一端。任何必需的可靠 性必须由应用层来提供。这两种运输层协议分别在不同的应用程序中有不同的用途。
  - 4)应用层负责处理特定的应用程序细节。几乎各种不同的TCP/IP实现都会提供下面这些 通用的应用程序： •Telnet远程登录。 •FTP文件传输协议。 •SMTP简单邮件传送协议。 •SNMP简单网络管理协议。

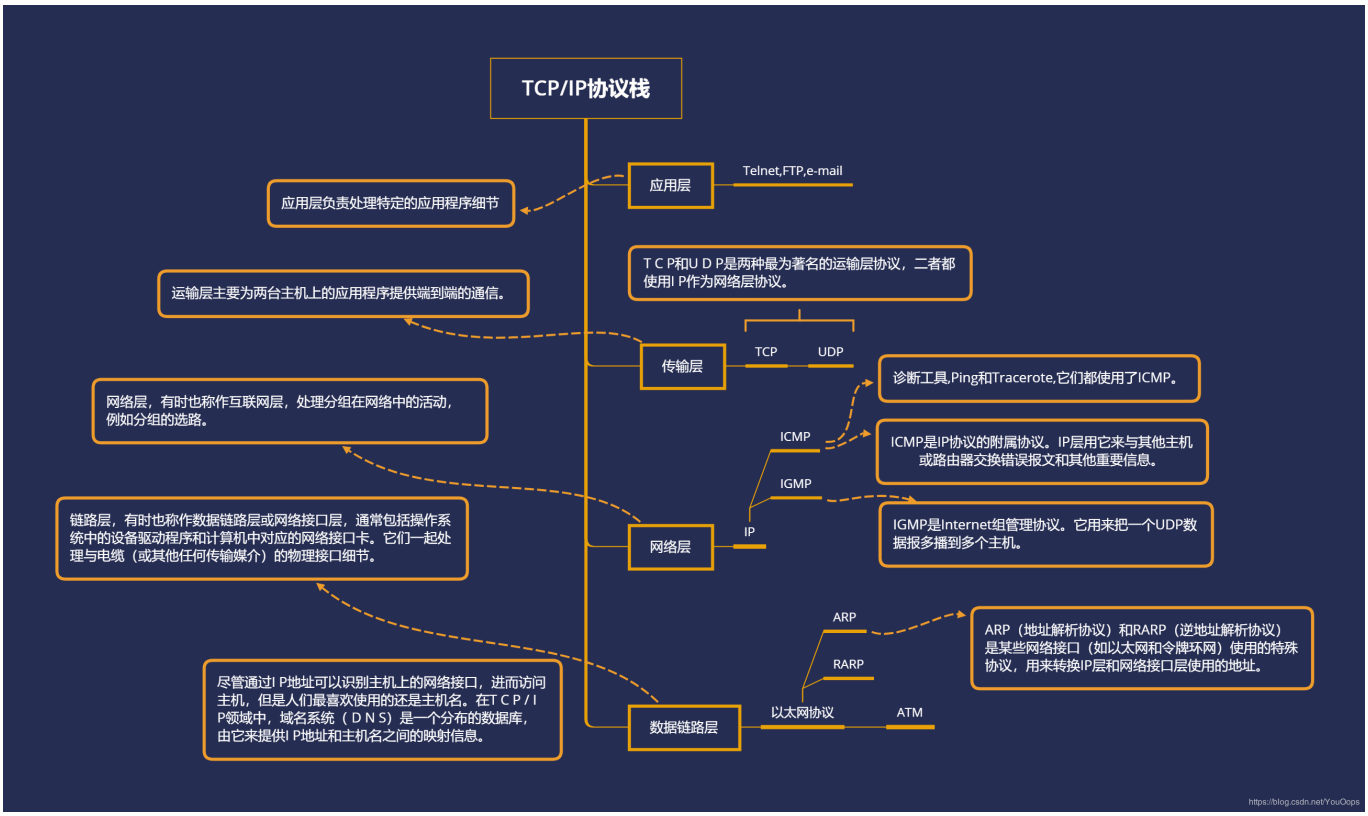


图2. 1-TCP/IP协议栈所在的4层

2.数据链路层

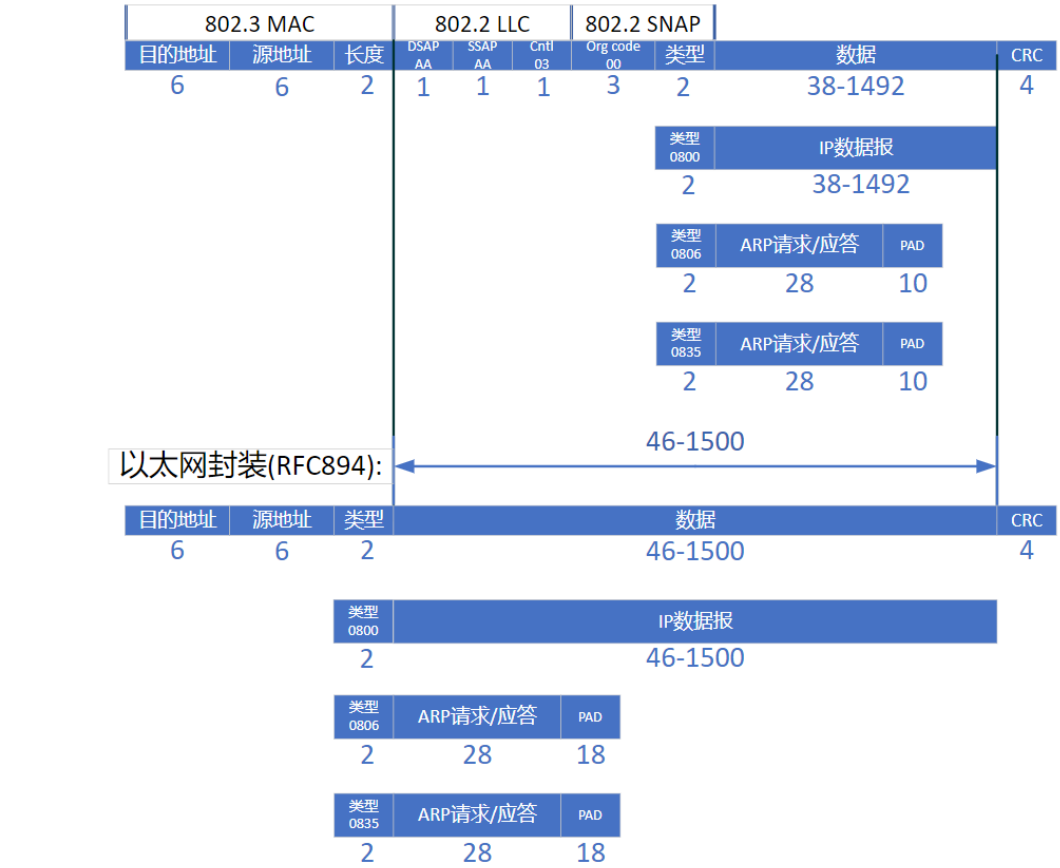
- 在TCP/IP协议族中，数据链路层主要有三个目的
    - （1）为IP模块发送和接收IP数据报；
    - （2）为ARP模块发送ARP请求和接收ARP应答；
    - （3）为RARP发送RARP请求和接收RARP应答。
- TCP/IP支持多种不同的链路层协议，这取决于网络所使用的硬件，如以太网、 令牌环网、FDDI（光纤分布式数据接口）及RS-232串行线路等。

以太网

- 以太网这个术语一般是指数字设备公司（DigitalEquipmentCorp.）、英特尔公司（IntelCorp.） 和Xerox公司在1982年联合公布的一个标准。它是当今TCP/IP采用的主要的局域网技术。它采用一种 称作CSMA/CD的媒体接入方法，其意思是带冲突检测的载波侦听多路接入（CarrierSense, MultipleAccesswithCollisionDetection）。它的速率为10Mb/s，地址为48bit。
- 后来，IEEE（电子电气工程师协会）802委员会公布了一个稍有不同的标准 集，其中802.3针对整个CSMA/CD网络，802.4针对令牌总线网络， 802.5针对令牌环网络。这三者的共同特性由802.2标准来定义，那就是8 02网络共有的逻辑链路控制（LLC）。
- IEEE802.2-802.3和以太网的数据帧封装格式如图2.2.1。
    - IEEE802.2-802.3和以太网的数据帧封装,两种帧格式都采用48bit（6字节）的目 的地址和源地址（802.3允许使用16 bit的地址，但一般是48bit地址）。这就是硬件 地址,也就是电脑网卡的MAC地址。ARP和RARP协议对32bit的IP地址和48bit的硬件地 址进行映射。

  - 接下来的2个字节在两种帧格式中互不相同。在802标准定义的帧格式中，长度字段 是指它后续数据的字节长度，但不包括CRC检验码。以太网的类型字段定义了后续数据 的类型。在802标准定义的帧格式中，类型字段则由后续的子网接入协议（Sub-network AccessProtocol，SNAP）的首部给出。
  - 在以太网帧格式中，类型字段之后就是数据；而在802帧格式中，跟随在后面的是3字节 的802.2LLC和5字节的802.2SNAP。目的服务访问点（DestinationServiceAccessPoint, DSAP）和源服务访问点（SourceServiceAccessPoint,SSAP）的值都设为0xaa。 Ctrl字段的值设为3。随后的3个字节orgcode都置为0。再接下来的2个字节类型字段和 以太网帧格式一样。
  - CRC字段用于帧内后续字节差错的循环冗余码检验（检验和）（它也被称为FCS或帧检验 序列）。802.3标准定义的帧和以太网的帧都有最小长度要求。802.3规定数据部分必须 至少为38字节，而对于以太网，则要求最少要有46字节。

- IEEE802.2-802.3和以太网的数据帧封装格式



IEEE 802.2/802.3（RFC 1042）和以太网的封装格式（RFC 894）

图2. 2. 1

串行接口链路层协议SLIP

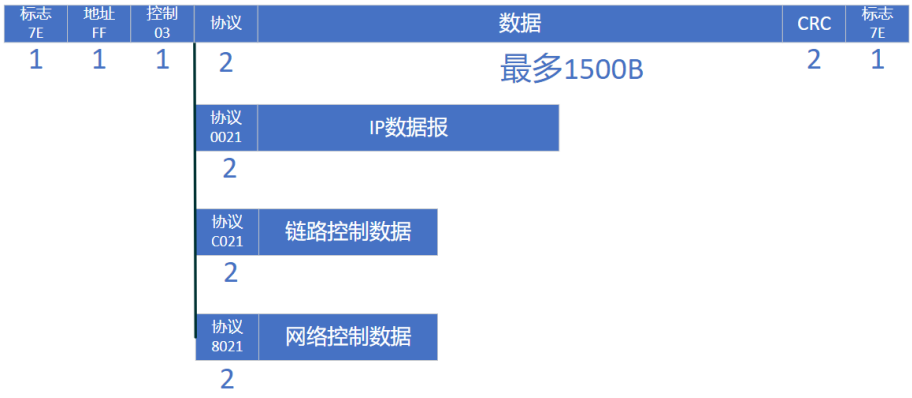
- SLIP的全称是Serial Line IP。它是一种在串行线路上对IP数据报进行封装的 简单形式，在RFC1055[Romkey1988]中有详细描述。SLIP适用于家庭中每台计算 机几乎都有的RS-232串行端口和高速调制解调器接入Internet。

串行接口链路层协议PPP(点对点协议)

- PPP，点对点协议修改了SLIP协议中的所有缺陷。PPP包括以下三个部分:

  - 1)在串行链路上封装IP数据报的方法。PPP既支持数据为8位和无奇偶检验的 异步模式（如大多数计算机上都普遍存在的串行接口），还支持面向比特的同步链接。
  - 2)建立、配置及测试数据链路的链路控制协议（LCP：LinkControlProtocol）。 它允许通信双方进行协商，以确定不同的选项。
  - 3)针对不同网络层协议的网络控制协议（NCP：NetworkControlProtocol）体系。 当前RFC定义的网络层有IP、OSI网络层、DECnet以及AppleTalk。例如， IPNCP允许双方商定是否对报文首部进行压缩，类似于CSLIP（缩写词NCP也可用在TCP的前面）。

- PPP数据帧的格式



PPP数据帧的格式

图2. 2. 2

- PPP数据帧字段

  - 每一帧都以标志字符0x7e开始和结束。紧接着是一个地址字节，值始终是0xff， 然后是一个值为0x03的控制字节。
  - 接下来是协议字段，类似于以太网中类型字段的功能。当它的值为0x0021时， 表示信息字段是一个IP数据报；值为0xc021时，表示信息字段是链路控制数据； 值为0x8021时，表示信息字段是网络控制数据。
  - CRC字段（或FCS，帧检验序列）是一个循环冗余检验码，以检测数据帧中的错误。

- PPP&SLIP相同点

  - 与SLIP类似，由于PPP经常用于低速的串行链路，因此减少每一帧的字节数可以 降低应用程序的交互时延。利用链路控制协议，大多数的产品通过协商可以省略标志符和 地址字段，并且把协议字段由2个字节减少到1个字节。

- 总的来说，PPP比SLIP具有下面这些优点：

  - (1)PPP支持在单根串行线路上运行多种协议，不只是IP协议；
  - (2)每一帧都有循环冗余检验；
  - (3)通信双方可以进行IP地址的动态协商(使用IP网络控制协议)；
  - (4)与CSLIP类似，对TCP和IP报文首部进行压缩；
  - (5)链路控制协议可以对多个数据链路选项进行设置。为这些优点付出的代价是在 每一帧的首部增加3个字节，当建立链路时要发送几帧协商数据，以及更为复杂的实现。

环回接口

- 大多数的产品都支持环回接口（Loopback Interface），以允许运行在同一台主机上 的客户程序和服务器程序通过TCP/IP进行通信。A类网络号127就是为环回接口预留的。根 据惯例，大多数系统把IP地址127.0.0.1分配给这个接口，并命名为localhost。一个传 给环回接口的IP数据报不能在任何网络上出现。

- 我们所想象的是，一旦传输层检测到目的端地址是环回地址时，应该可以省略部分传输 层和所有网络层的逻辑操作。但是大多数的产品还是照样完成传输层和网络层的所有过 程，只是当IP数据报离开网络层时把它返回给自己。

- 图2.2.3是环回接口处理IP数据报的简单过程

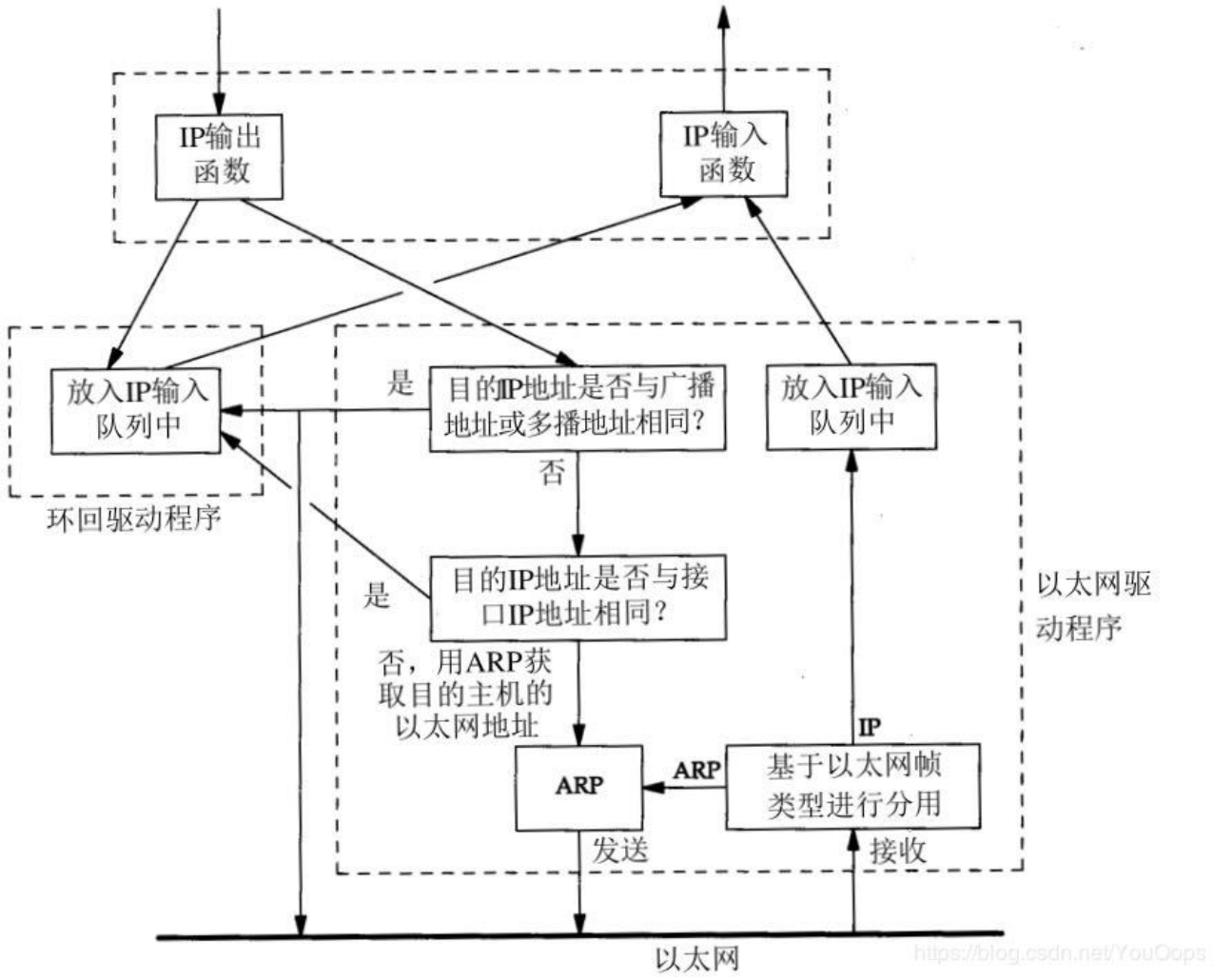


图2. 2. 3环回接口处理IP数据报的过程

- 图中需要指出的关键点有：

    - 1. 传给环回地址（一般是127.0.0.1）的任何数据均作为IP输入。
    - 2. 传给广播地址或多播地址的数据报复制一份传给环回接口，然后送到以太网上。 这是 因为广播传送和多播传送的定义（第12章）包含主机本身。
    - 3)任何传给该主机IP地址的数据均送到环回接口。

- 看上去用传输层和IP层的方法来处理环回数据似乎效率不高，但它简化了设计，因为 环回接口可以被看作是网络层下面的另一个链路层。网络层把一份数据报传送给环回接 口，就像传给其他链路层一样，只不过环回接口把它返回到IP的输入队列中。

- 图2-4中，另一个隐含的意思是送给主机本身IP地址的IP数据报一般不出现在相应的 网络上。例如，在一个以太网上，分组一般不被传出去然后读回来。某些BSD以太网的 设备驱动程序的注释说明，许多以太网接口卡不能读回它们自己发送出去的数据。

> 4.4BSD系统定义了变量useloopback，并初始化为1。但是，如果这个变量置为0，
> 以太网驱动程序就会把本地分组送到网络，而不是送到环回接口上。它也许不能工作，
> 这取决于所使用的以太网接口卡和设备驱动程序。

3.网络层

IP协议

- IP是TCP/IP协议族中最为核心的协议。所有的TCP、UDP、ICMP及IGMP数据都以IP数据 报格式传输。许多刚开始接触TCP/IP的人对IP提供不可靠、无连接的数据报传送服务感到 很奇怪。IP提供的是一种不可靠的服务。也就是说，它只是尽可能快地把分组从源结点送 到目的结点，但是并不提供任何可靠性保证。而另一方面，TCP在不可靠的IP层上提供了 一个可靠的运输层。为了提供这种可靠的服务，TCP采用了超时重传、发送和接收端到端的 确认分组等机制。
    - 不可靠（unreliable）的意思是它不能保证IP数据报能成功地到达目的地。IP仅提供最好 的传输服务。如果发生某种错误时，如某个路由器暂时用完了缓冲区，IP有一个简单的错误 处理算法：丢弃该数据报，然后发送ICMP消息报给信源端。任何要求的可靠性必须由上层来 提供（如TCP）。
    - 无连接（connectionless）这个术语的意思是IP并不维护任何关于后续数据报 的状态信息。每个数据报的处理是相互独立的。这也说明，IP数据报可以不按发送顺序接收。 如果一信源向相同的信宿发送两个连续的数据报（先是A，然后是B），每个数据报都是独立地 进行路由选择，可能选择不同的路线，因此B可能在A到达之前先到达。

图2. 3. 1-IP报文头部格式

- 头部各字节的具体含义如下:

字段	含义
4位版本号	指的是IP协议的版本信息目前的协议版本号是4，因此IP有时也称作IPv4。将来IPv6将会普及。
4位首部长度	指的是头部占32bit字的数目(头部的"选项"字段也计算在内)，最多标识15个32bit，也就是首部最长位60字节。不带选项的IP报文该字段值为5，也就是首部固定大小20字节。
8位服务类型	服务类型(TOS)字段包括一个3bit的优先权子字段(现在已被忽略)，4bit的TOS子字段和1bit未用位但必须置0。4bit的TOS分别代表：最小时延、最大吞吐量、最高可靠性和最小费用。4bit中只能置其中1bit。如果所有4bit均为0，那么就意味着是一般服务。新的路由协议如OSPF和IS-IS都能根据这些字段的值进行路由决策。不同应用建议的TOS值见图2.3.2。

字段	含义
16位总长度(字节数)	总长度字段是指整个IP数据报的长度，以字节为单位。利用首部长度字段和总长度字段，就可以知道IP数据报中数据内容的起始位置和长度。由于该字段长16比特，所以IP数据报最长可达65535字节,超级通道的MTU为65535。尽管可以传送一个长达65535字节的IP数据报，但是大多数的链路层都会对它进行分片。
16位标识	标识字段唯一地标识主机发送的每一份数据报。通常每发送一份报文它的值就会加1。
3位标志	标志字段和片偏移字段在分片时使用
13位片偏移	标志字段和片偏移字段在分片时使用
8位生存时间(TTL)	TTL(time-to-live)生存时间字段设置了数据报可以经过的最多路由器数。它指定了数报的生存时间。TTL的初始值由源主机设置(通常为32或64)，一旦经过一个处理它的路由器，它的值就减去1。当该字段的值为0时，数据报就被丢弃，并发送ICMP报文通知源主机。可以使用traceroute命令来分析。
8位协议	指明其上层协议类型(TCP/UDP/ICMP/IGMP,etc.)，在分用^1的时候各报文的首部协议字段会被程序检查，以确定其上层协议类型。
16位首部检验和	首部检验和字段是根据IP首部计算的检验和码。它不对首部后面的数据进行计算。ICMP、IGMP、UDP和TCP在它们各自的首部中均含有同时覆盖首部和数据检验和码。[^2]
32位源IP地址	数据的源主机IP地址
32位目的地址	数据的目的主机IP地址
选项	最后一个字段是任选项，是数据报中的一个可变长的可选信息。目前，这些任选项定义这些选项很少被使用，并非所有的主机和路由器都支持这些选项。

应用程序		最小时延	最大吞吐量	最高可靠性	最小费用	(1bit未用位)	16进制值(tcpdump输出)
Telnet/Rlogin		1	0	0	0	0	0x10
FTP	控制	1	0	0	0	0	0x10
	数据	0	1	0	0	0	0x08
	任意块数据	0	1	0	0	0	0x08
TFTP		1	0	0	0	0	0x10
SMTP	命令阶段	1	0	0	0	0	0x10
	数据阶段	0	1	0	0	0	0x08
DNS	UDP查询	1	0	0	0	0	0x10
	TCP查询	0	0	0	0	0	0x00
	区域传输	0	1	0	0	0	0x08
ICMP	差错	0	0	0	0	0	0x00
	查询	0	0	0	0	0	0x00
	任IGP	1	0	1	0	0	0x04
SNMP		1	0	1	0	0	0x04
BOOTP		0	0	0	0	0	0x00
NNTP		0	0	0	1	0	0x02

图2. 3. 2–不同应用建议的TOS值

- Telnet和Rlogin这两个交互应用要求最小的传输时延，因为人们主要用它们来传输少量的 交互数据。另一方面，FTP文件传输则要求有最大的吞吐量。最高可靠性被指明给网络管理 （SNMP）和路由选择协议。用户网络新闻（Usenet news, NNTP）是唯一要求最小费用的应用。

ICMP协议

- ICMP协议全称为：Internet控制报文协议，主要用于传递差错报文和其他信息。ICMP报文通常被 IP层或者更高层协议(TCP/UDP)使用。某些ICMP报文用来将差错报文传给用户进程。图2.3.3位ICMP报文格式。

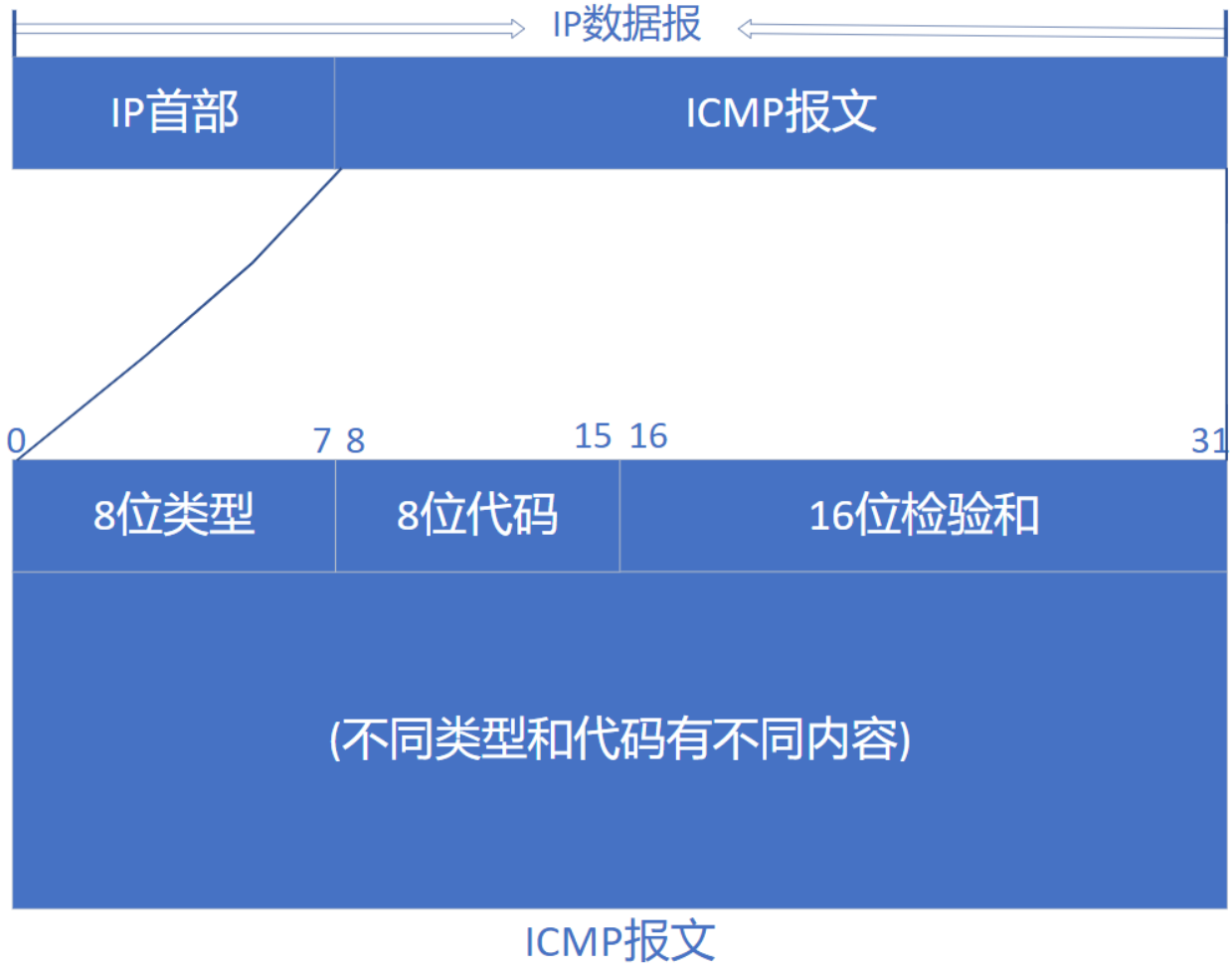


图2.3.3-ICMP报文格式

IGMP协议

- IGMP协议全称为：Internet组管理协议，用于支持主机和路由器进行多播。它让一个物理网络 上的所有系统知道主机当前所在的多播组。多播路由器需要这些信息以便知道多播数据报应该向哪些接口转发。
- 和ICMP一样，IGMP也被当作IP层的一部分。IGMP报文通过IP数据报进行传输。不像我们已经见 到的其他协议，IGMP有固定的报文长度，没有可选数据。IGMP报文通过IP首部中协议字段值为2来指明。见图2.3.4。
- 图3.4是版本为1的IGMP。IGMP类型为1说明是由多播路由器发出的查询报文，为2说明是主机发出的报告报文。 检验和的计算和ICMP协议相同。组地址为D类IP地址。在查询报文中组地址设置为0，在报告报文中组地址 为要参加的组地址。

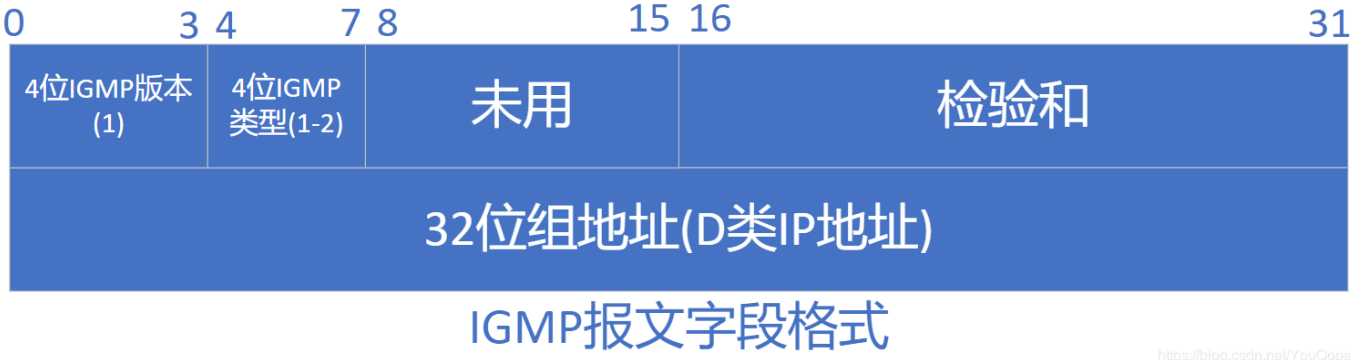


图2.3.4-IGMP报文格式

最大传输单元MTU

- 以太网和802.3对数据帧的长度都有一个限制，其最大值分别是1500和1492字节。 链路层的这个特性称作MTU，最大传输单元。不同类型的网络大多数都有一个上限。
- 如果I P层有一个数据报要传，而且数据的长度比链路层的MTU还大，那么IP层 就需要进行分片（fragmentation），把数据报分成若干片，这样每一片都小于MTU。

网　　络	MTU字节
超通道	65535
16 Mb/ s令牌环(IBM)	17914
4 Mb/ s令牌环(IEEE 802.5)	4464
FDDI	4352
以太网	1500
IEEE 802.3/802.2	1492
X.25	576
点对点(低时延)	296

图2.3.5几种常见的MTU

路径MTU

- 当在同一个网络上的两台主机互相进行通信时，该网络的MTU是非常重要的。但是如果 两台主机之间的通信要通过多个网络，那么每个网络的链路层就可能有不同的MTU。重要的 不是两台主机所在网络的MTU的值，重要的是两台通信主机路径中的最小MTU。它被称作 路径MTU。两台主机之间的路径MTU不一定是个常数。它取决于当时所选择的路由。而选 路不一定是对称的（从A到B的路由可能与从B到A的路由不同），因此路径MTU在两个方向上 不一定是一致的。

4.传输层

TCP协议

- 尽管TCP和UDP都使用相同的网络层（IP），TCP却向应用层提供与UDP完全 不同的服务。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。在第12章介绍的广播和多播不能用于TCP。 TCP通过下列方式来提供可靠性： •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 数据报长度将保持不变。由TCP传递给IP的信息单位称为报文段或段（segment）（参见 图1-7）。在18.4节我们将看到TCP如何确定报文段的长度。 •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。在第21章我们将了解TCP协议中自适应的超时 及重传策略。 •当TCP收到发自TCP连接另一端的数据，它将发送一个确认。这个确认不是立即发送， 通常将推迟几分之一秒，这将在19.3节讨论。 •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）。 •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。 •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。 •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。收方可以分4次接 收这80个字节，每次接收20字节。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。

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。

- TCP数据被封装在一个IP数据报中,TCP头部字段格式见图2.4.1

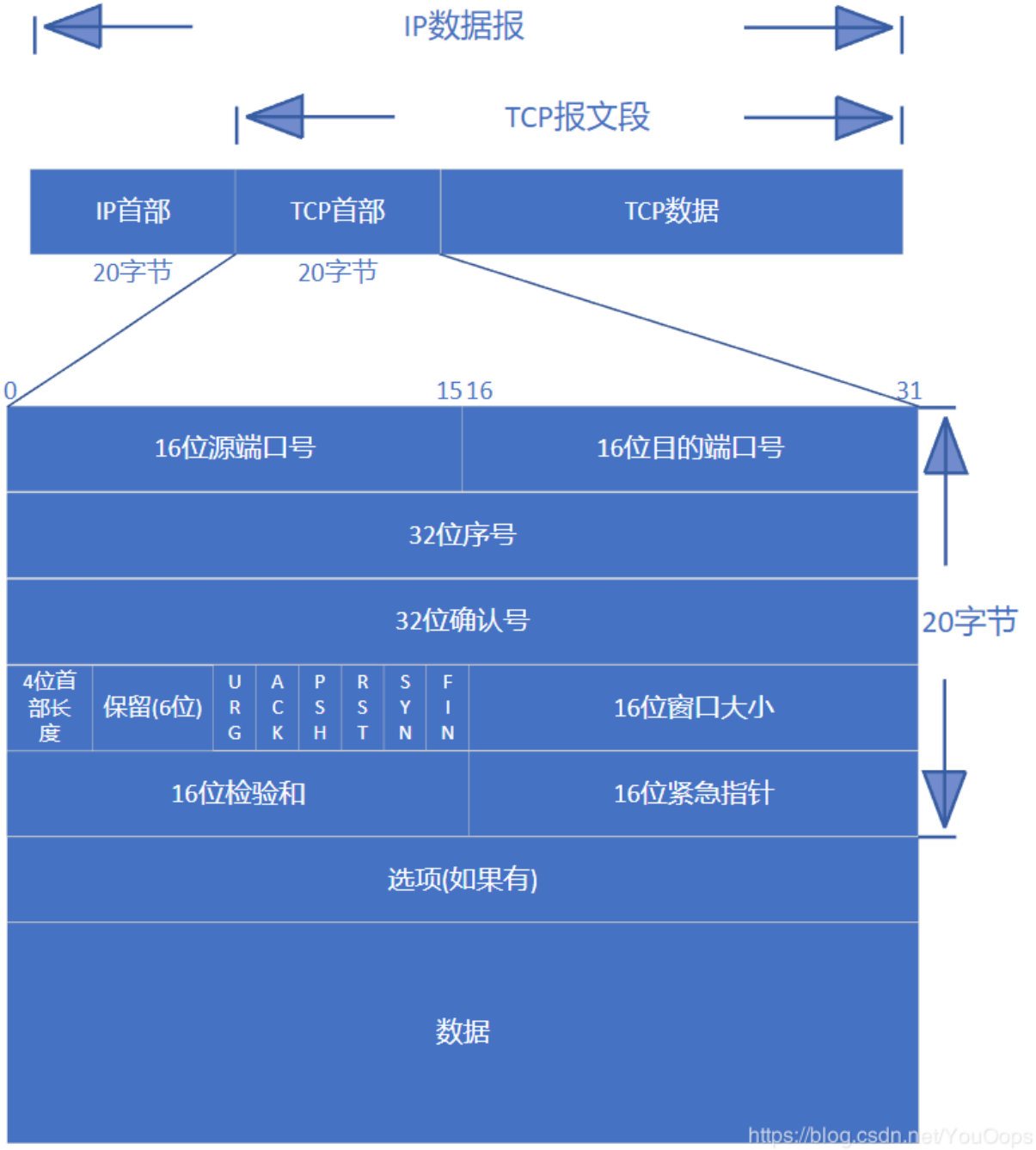


图2.4.1-TCP头部字段格式

- 每个TCP段都包含源端和目的端的端口号，用于寻找发端和收端应用进程。这两个值加 上IP首部中的源端IP地址和目的端IP地址唯一确定一个TCP连接。
- 有时，一个IP地址和一个端口号也称为一个插口（socket）。这个术语出现在最早的TCP 规范（RFC793）中，后来它也作为表示伯克利版的编程接口（参见1.15节）。插口对 （socket pair）(包含客户IP地址、客户端口号、服务器IP地址和服务器端口号的四元组)可 唯一确定互联网络中每个TCP连接的双方。
- 序号用来标识从TCP发端向TCP收端发送的数据字节流，它表示在这个报文段中的的第一 个数据字节。如果将字节流看作在两个应用程序间的单向流动，则TCP用序号对每个字节进 行计数。序号是32bit的无符号数，序号到达232－1后又从0开始。
- 当建立一个新的连接时，SYN标志变1。序号字段包含由这个主机选择的该连接的初始序 号ISN（InitialSequenceNumber）。该主机要发送数据的第一个字节序号为这个ISN加1，因 为SYN标志消

耗了一个序号。

- 既然每个传输的字节都被计数，确认序号包含发送确认的一端所期望收到的下一个序号。 因此，确认序号应当是上次已成功收到数据字节序号加1。只有ACK标志（下面介绍）为1时 确认序号字段才有效。
- 发送ACK无需任何代价，因为32bit的确认序号字段和ACK标志一样，总是TCP首部的一 部分。因此，我们看到一旦一个连接建立起来，这个字段总是被设置，ACK标志也总是被设 置为1。
- TCP为应用层提供全双工服务。这意味数据能在两个方向上独立地进行传输。因此，连 接的每一端必须保持每个方向上的传输数据序号。
- TCP可以表述为一个没有选择确认或否认的滑动窗口协议（滑动窗口协议用于数据传输 ）。我们说TCP缺少选择确认是因为TCP首部中的确认序号表示发方已成功收到字节，但还不包 含确认序号所指的字节。当前还无法对数据流中选定的部分进行确认。例如，如果1～1024 字节已经成功收到，下一报文段中包含序号从2049～3072的字节，收端并不能确认这个新的 报文段。它所能做的就是发回一个确认序号为1025的ACK。它也无法对一个报文段进行否认。 例如，如果收到包含1025～2048字节的报文段，但它的检验和错，TCP接收端所能做的就是发 回一个确认序号为1025的ACK。在21.7节我们将看到重复的确认如何帮助确定分组已经丢失。
- 首部长度给出首部中32bit字的数目。需要这个值是因为任选字段的长度是可变的。这个 字段占4bit，因此TCP最多有60字节的首部。然而，没有任选字段，正常的长度是20字节。
- 在TCP首部中有6个标志比特。它们中的多个可同时被设置为1。
    - URG 紧急指针（urgentpointer）有效。
    - ACK 确认序号有效。
    - PSH 接收方应该尽快将这个报文段交给应用层。
    - RST 重建连接。
    - SYN 同步序号用来发起一个连接。
    - FIN 发端完成发送任务。
- TCP的流量控制由连接的每一端通过声明的窗口大小来提供。窗口大小为字节数，起始 于确认序号字段指明的值，这个值是接收端正期望接收的字节。窗口大小是一个16bit字段， 因而窗口大小最大为65535字节。新的窗口刻度选项允许这个值按比例变化以提供更大的窗口。
- 检验和覆盖了整个的TCP报文段：TCP首部和TCP数据。这是一个强制性的字段，一定是 由发端计算和存储，并由收端进行验证。TCP检验和的计算和UDP检验和的计算相似。
- 只有当URG标志置1时紧急指针才有效。紧急指针是一个正的偏移量，和序号字段中的值 相加表示紧急数据最后一个字节的序号。TCP的紧急方式是发送端向另一端发送紧急数据的 一种方式。
- 最常见的可选字段是最长报文大小，又称为MSS(MaximumSegmentSize)。每个连接方 通常都在通信的第一个报文段（为建立连接而设置SYN标志的那个段）中指明这个选项。它 指明本端所能接收的最大长度的报文段。
- 从图中可以看到到TCP报文段中的数据部分是可选的。接建立和一个连接终止时，双方交换 的报文段仅有TCP首部。如果一方没有数据要发送，也使用没有任何数据的首部来确认收到的 数据。在处理超时的许多情况中，也会发送不带任何数据的报文段。

小结

TCP提供了一种可靠的面向连接的字节流运输层服务。
TCP将用户数据打包构成报文段；它发送数据后启动一个定时器；另一端对收到的数据进行确认，对失序的数据重新排序，丢弃重复数据；TCP提供端到端的流量控制，并计算和验证一个强制性的端到端检验和。
许多流行的应用程序如Telnet、Rlogin、FTP和SMTP都使用TCP。

TCP连接的建立与终止

- TCP是一个面向连接的协议。无论哪一方向另一方发送数据之前，都必须先在双方之间 建立一条连接。

三次握手建立连接

在使用TCP建立端到端(客户端到服务器)的通讯前必须进行以下三步,如图2.4.2红线以上部分所示。：

- 1)请求端（通常称为客户）发送一个SYN段指明客户打算连接的服务器的端口，以及初 始序号（ISN，在这个例子中为1415531521）。这个SYN段为报文段1。
- 2)服务器发回包含服务器的初始序号的SYN报文段（报文段2）作为应答。同时，将确认序号设置为客户的ISN加1以对客户的SYN报文段进行确认。一个SYN将占用一个序号。
- 3)客户必须将确认序号设置为服务器的ISN加1以对服务器的SYN报文段进行确认（报文段3）。
- 通过这三个报文后就建立连接了。这个过程也称为三次握手(three-way handshake)。
- 图2.4.4记录了客户端使用telnet连接服务器后再使用quit命令关闭连接的过程。

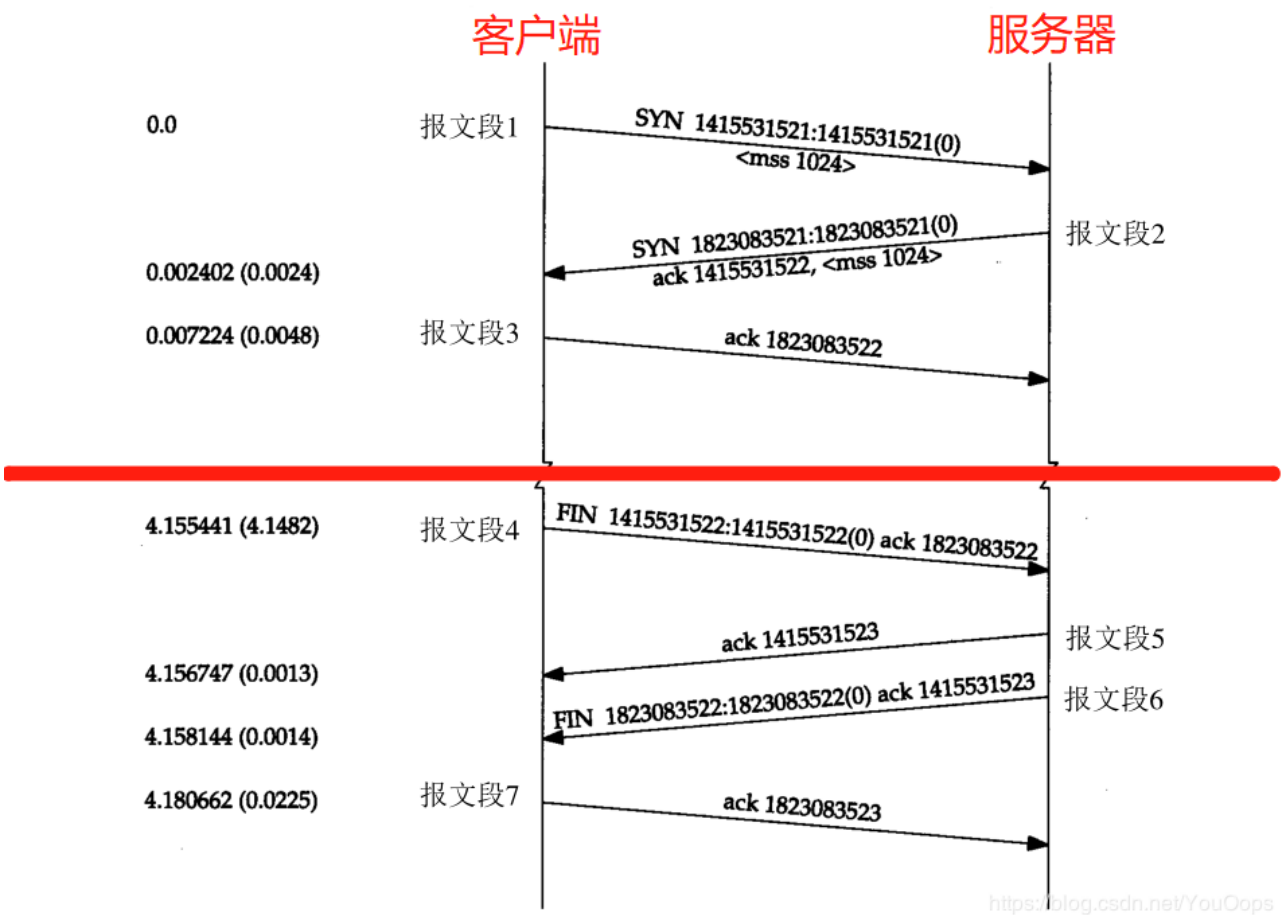


图2.4.2-TCP建立连接的三步和断开连接的四步

四次握手断开连接

- 建立一个连接需要三次握手，而终止一个连接要经过4次握手。这由TCP的半关闭（half-close） 造成的。既然一个TCP连接是全双工（即数据在两个方向上能同时传递），因此每个方向必须 单独地进行关闭。这原则就是当一方完成它的数据发送任务后就能发送一个FIN来终止 这个方向连接。当一端收到一个FIN，它必须通知应用层另一端几经终止了那个方向的数据传 送。发送FIN通常是应用层进行关闭的结果。
- 收到一个FIN只意味着在这一方向上没有数据流动。**一个TCP连接在收到一个FIN后仍能 发送数据**。而这对利用半关闭的应用来说是可能的，尽管在实际应用中只有很少的TCP应用 程序这样做。正常关闭过

程如图2.4.2红线以下所示。

- 首先进行关闭的一方（即发送第一个FIN）将执行主动关闭，而另一方（收到这个FIN） 执行被动关闭。通常一方完成主动关闭而另一方完成被动关闭，但也可能双方都执行主动关闭。
- 图2.4.2中的报文段4发起终止连接，它由Telnet客户端关闭连接时发出。这在键入quit 命令后发生。它将导致TCP客户端发送一个FIN，用来关闭从客户到服务器的数据传送。

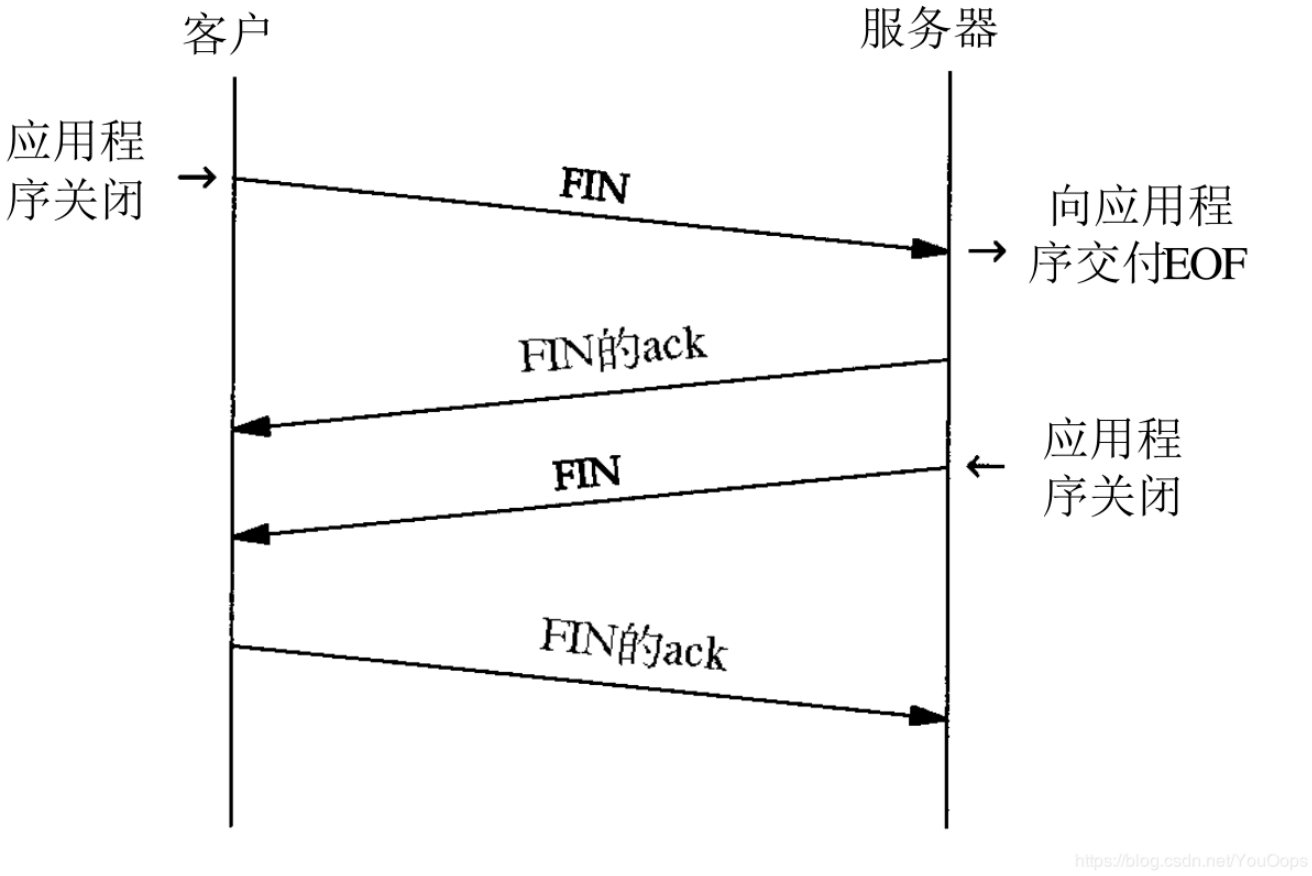


图2.4.3-连接终止期间报文段的交换过程

- 当服务器收到这个FIN，它发回一个ACK，确认序号为收到的序号加1（报文段5）。和SYN一样， 一个FIN将占用一个序号。同时TCP服务器还向应用程序（即丢弃服务器）传送一个文件结束符。 接着这个服务器程序就关闭它的连接，导致它的TCP端发送一个FIN（报文段6），客户必须发回 一个确认，并将确认序号设置为收到序号加1（报文段7）。
- 图2.4.3显示了终止一个连接的典型握手顺序。我们省略了序号。在这个图中，发送FIN将导致 应用程序关闭它们的连接，这些FIN的ACK是由TCP软件自动产生的。连接通常是由客户端发起的， 这样第一个SYN从客户传到服务器。每一端都能主动关闭这个连接（即首先发送FIN）。然而，一般 由客户端决定何时终止连接，因为客户进程通常由用户交互控制，用户会键入诸如“quit”一样的 命令来终止进程。在图2.4.3中，可以改变上边的标识，将左方定为服务器，右方定为客户，一切仍将像显示的一样工作。

TCP的状态变迁图(有限状态机)

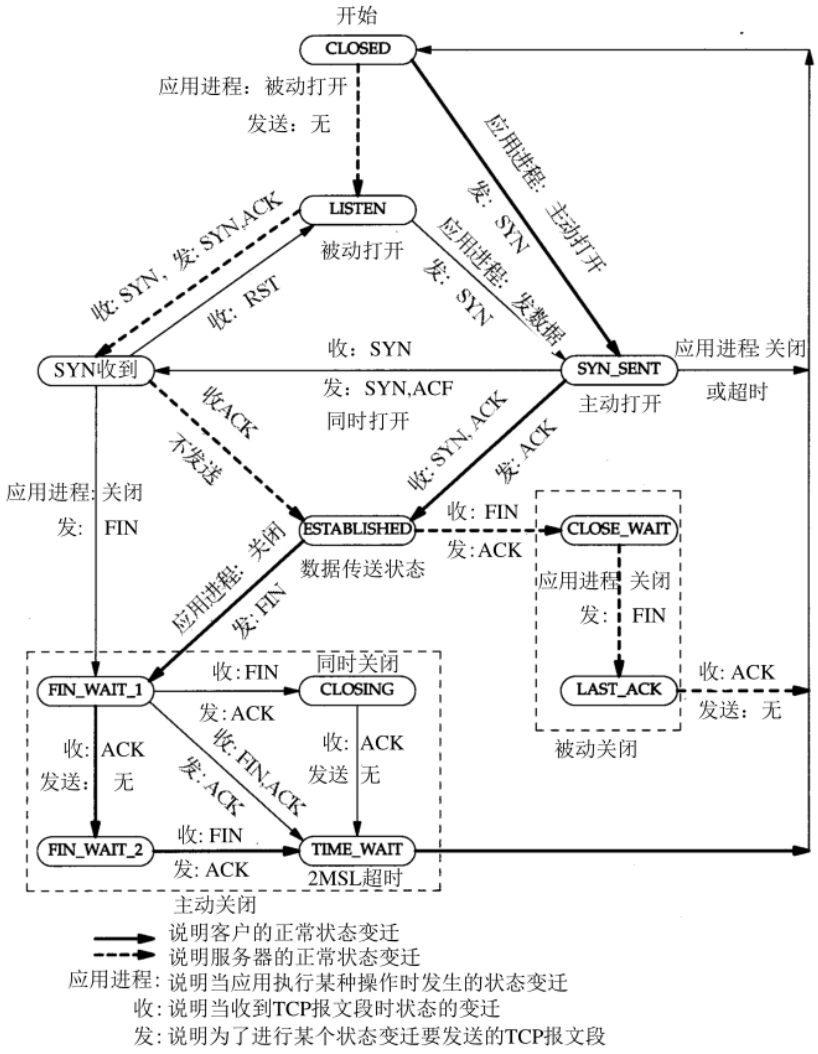


图2. 4. 4-TCP的状态变迁图

- 如上图所示，所有有关发起和终止TCP连接的规则都可以在图中得到。
- 图2.4.4中使用粗的实线箭头表示正常的客户端状态变迁，用粗的虚线箭头表示正常的 服务器状态变迁。
- 两个导致进入ESTABLISHED状态的变迁对应打开一个连接，而两个导致从ESTABLISHED 状态离开的变迁对应关闭一个连接。ESTABLISHED状态是连接双方能够进行双向数据传递的状 态。

2MSL等待状态

- TIME_WAIT状态也称为2MSL等待状态。每个具体TCP实现必须选择一个报文段最大生 存时间MSL（Maximum Segment Lifetime）。它是任何报文段被丢弃前在网络内的最长时 间。我们知道这个时间是有限的，因为TCP报文段以IP数据报在网络内传输，而IP数据报则有 限制其生存时间的TTL字段。

RFC 793 [Postel 1981c] 指出MSL为2分钟。然而，实现中的常用值是30秒，1分钟，或2分钟。

- 在实际应用中，对IP数据报TTL的限制是基于跳数，而不是定时器。对一个具体实现所给定 的MSL值，处理的原则是：当TCP执行一个主动关闭，并发回最后一个ACK，该连接必须在TIME_WAIT 状态停留的时间为2倍的MSL。这样可让TCP再次发送最后的ACK以防这个ACK丢失（另一端超时并 重发最后的FIN）
- 这种2MSL等待的另一个结果是这个TCP连接在2MSL等待期间，定义这个连接的插口（客户的IP地址 和端口号，服务器的IP地址和端口号）不能再被使用。这个连接只能在2MSL结束后才能再被使用。

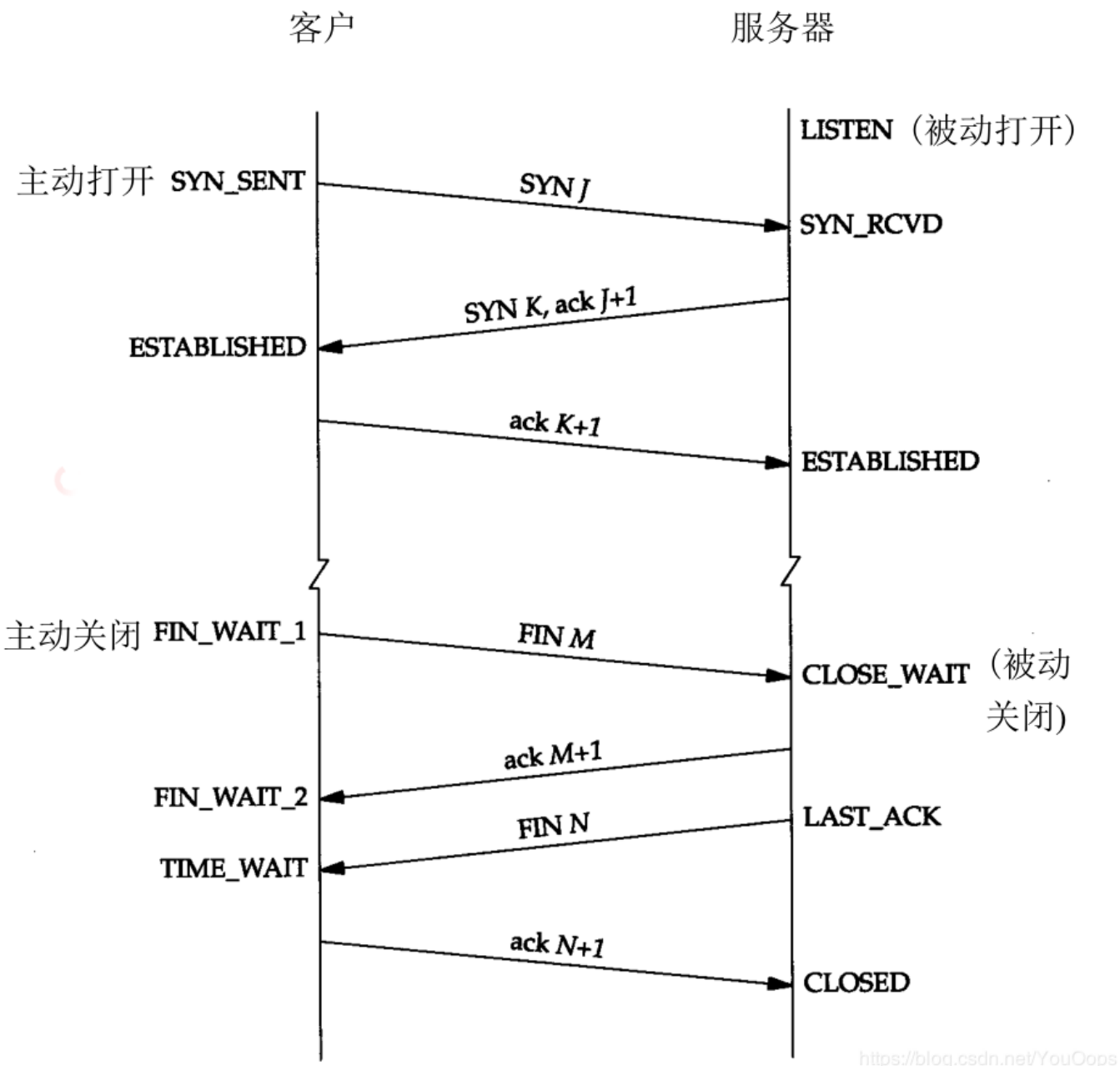


图2.4.5-TCP正常连接建立和终止所对应的状态

- 在连接处于2MSL等待时，任何迟到的报文段将被丢弃。因为处于2MSL等待的、由该插 口对(socketpair)定义的连接在这段时间内不能被再用，因此当要建立一个有效的连接时，来 自该连接的一个较早替身（incarnation）的迟到报文段作为新连接的一部分不可能不被曲解 （一个连接由一个插口对来定义。一个连接的新的实例（instance）称为该连接的替身）
- 我们说图2.4.5中客户执行主动关闭并进入TIME_WAIT是正常的。服务器通常执行被动 关闭，不会进入TIME_WAIT状态。这暗示如果我们终止一个客户程序，并立即重新启动这个 客户程序，则这个新客户程序将不能重用相同的本地端口。这不会带来什么问题，因为客户 使用本地端口，而并不关心这个端口号是什么。
- 然而，对于服务器，情况就有所不同，因为服务器使用熟知端口。如果我们终止一个已 经建立连接的服务器程序，并试图立即重新启动这个服务器程序，服务器程序将不能把它的 这个熟知端口赋值给它的端点，因为那个端口是处于2MSL连接的一部分。在重新启动服务器 程序前，它需要在1~4分钟。

UDP协议

- UDP是一个简单的面向数据报的运输层协议：进程的每个输出操作都正好产生一个UDP 数据报，并组装成一份待发送的IP数据报。这与面向流字符的协议不同，如TCP，应用 程序产生的全体数据与真正发送的单个IP数据报可能没有什么联系。UDP数据报被封装成一份 IP数据报的格式,如图2.4.4所示。

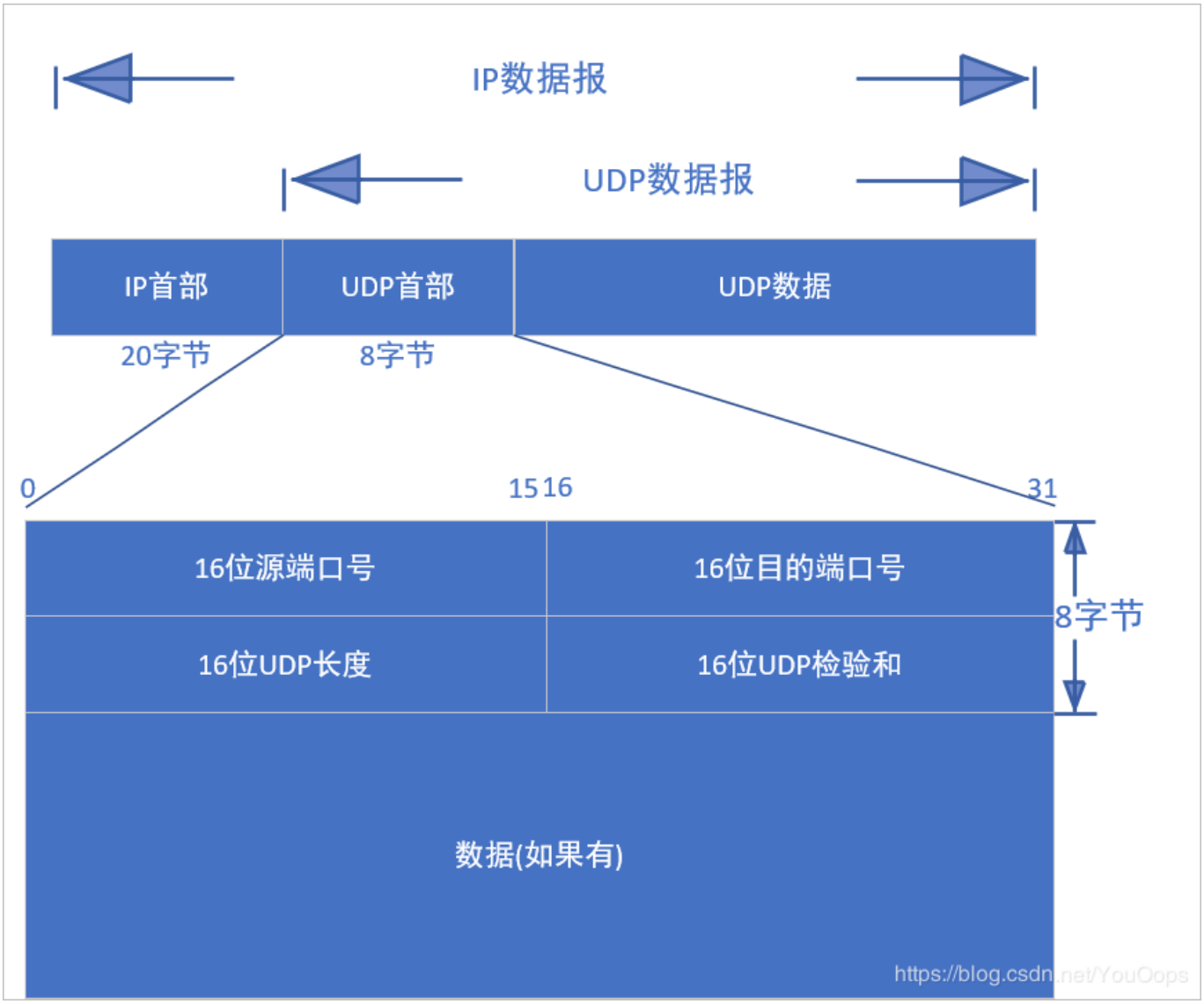


图2. 4. 6–UDP头部字段格式

三.IP地址规划

1.传统的IP地址分类

- 传统的地址分类方法分为A、B、C、D和E类地址，E类保留未使用。如图3.1

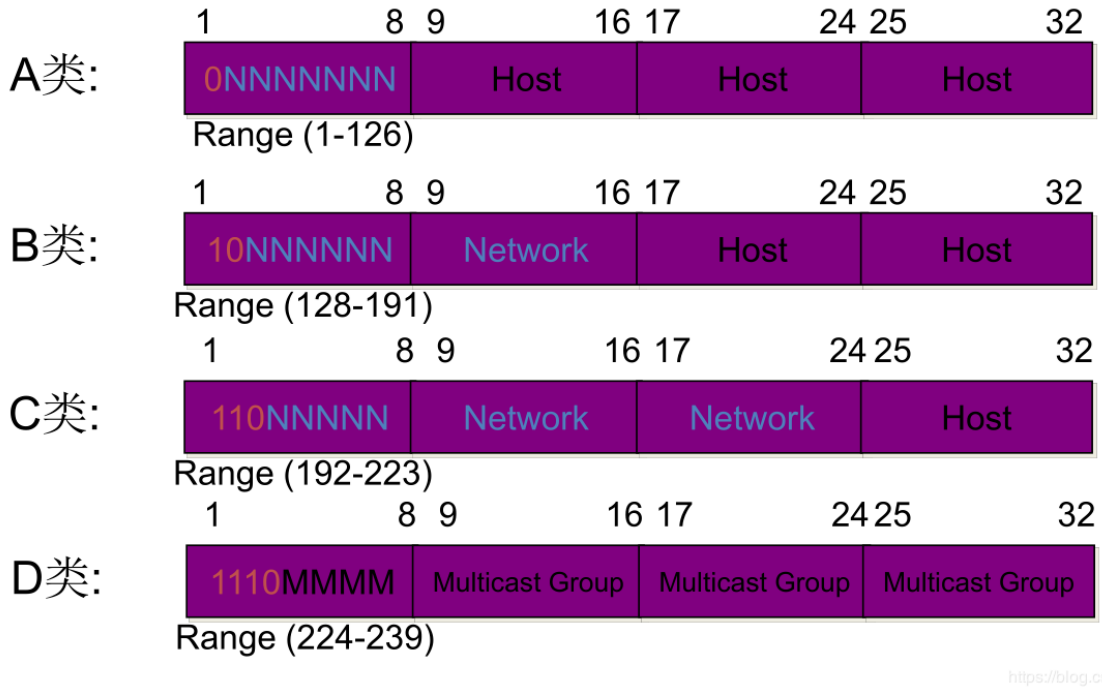


图3.1.1-IPV4地址划分

- 具体的网络数、所容纳的主机数及私网地址等如下：

```
1.A类地址：
  0000 0000 - 0111 1111: 1-127
  网络数：126, 127
  每个网络中的主机数：2^24-2
  默认子网掩码：255.0.0.0
  私网地址：10.0.0.0
2.B类地址：
  1000 0000 - 1011 1111：128-191
  网络数：2^14
  每个网络中的主机数：2^16-2
  默认子网掩码：255.255.0.0
  私网地址：172.16.0.0-172.31.0.0
3.C类地址：
  1100 0000 - 1101 1111: 192-223
  网络数：2^21
  每个网络中的主机数：2^8-2
  默认子网掩码：255.255.255.0
  私网地址：192.168.0.0-192.168.255.0
4.D类地址：
  D类地址为组播地址
  1110 0000 - 1110 1111: 224-239
```

- 公有地址
- 每个公网地址都是独立拥有的，例如49.235.246.92这个地址目前就只有我能用:)，所以公网地址数量地有限的。

类	公共 IP 地址范围
A	1.0.0.0到 9.255.255.255 11.0.0.0 到 126.255.255.255
B	128.0.0.0 到 172.15.255.255 172.32.0.0 到 191.255.255.255
C	192.0.0.0 到 192.167.255.255 192.169.0.0 到 223.255.255.255

图3.1.2-IPV4公有地址

- 私有地址
- 为了节省IP，避免每台电脑都分配一个公网地址，还提供了私有地址。任何人都可以拥有私有地址，只能用于局域网而不能用于公网。
- A、B和C类地址中都保留了一些范围的地址作为私有地址。

类	私有地址范围
A	10.0.0.0 到 10.255.255.255
B	172.16.0.0 到 172.31.255.255
C	192.168.0.0 到 192.168.255.255

图3.1.3-IPV4私有地址

- 特殊地址

```
0.0.0.0
        0.0.0.0不是一个真正意义上的IP地址。它表示所有不清楚的主机和目的网络
255.255.255.255
```

```
        限制广播地址。对本机来说，这个地址指本网段内(同一广播域)的所有主机
127.0.0.1～127.255.255.254
        本机回环地址，主要用于测试。在传输介质上永远不应该出现目的地址为“127.0.0.1”的数
据包
224.0.0.0到239.255.255.255
        组播地址，224.0.0.1特指所有主机，224.0.0.2特指所有路由器。224.0.0.5指OSPF路由
器，地址多用于一些特定的程序以及多媒体程序
169.254.x.x
        如果Windows主机使用了DHCP自动分配IP地址，而又无法从DHCP服务器获取地 址，系统会
为主机分配这样地址
```

2.子网掩码

- 子网掩码用来表示IP地址中的那些位是网络位，那些位是主机位。所以，子网掩码是决定IP地址 属于哪个网段的。子网掩码必须结合IP地址使用，否则是没有意义的。
- 在子网掩码中连续为1的高位表示网络位，连续全为0的低位表示主机位。
- 子网掩码非常重要，配置错误的主机一定会出现通讯问题：每次主机之间通讯时先根据自己的IP 地址和子网掩码来判断出自己的网段，再用自己的子网掩码和对方的IP地址计算目标所在网段，如 果在同一网段，那么在链路层以太网数据封装成帧时封装的是目标主机的MAC地址，如果不在同一 个网段，则封装的是本地路由器的MAC地址作为目标MAC地址。

3.CIDR表示的网络地址

- CIDR全称为：无类域内路由选择（Classless Inter-Domain Routing）
- 描述IP和网段时，子网掩码一般和IP成对出现，例如：192.168.123.234 255.255.255.0 但是这种表示方法比较长，另一种比较方便的表示方法就叫CIDR表示法。其直接将子网掩码为1的位数写在IP地址的后面。例如：192.168.123.234 255.255.255.0 可以改写为：192.168.123.234/24(24就是子网掩码全为1的位数)。

4.路由的概念

- 每个数据包要在不同网络中传输，都需要进行路由，每次路由时都根据路由表中的记录决定 下一跳该给谁。
- 路由分为：
    - 主机路由
    - 网段路由
    - 默认路由

四.Linux网络配置详解

1.Linux网络管理命令及配置文件概览

命令概览

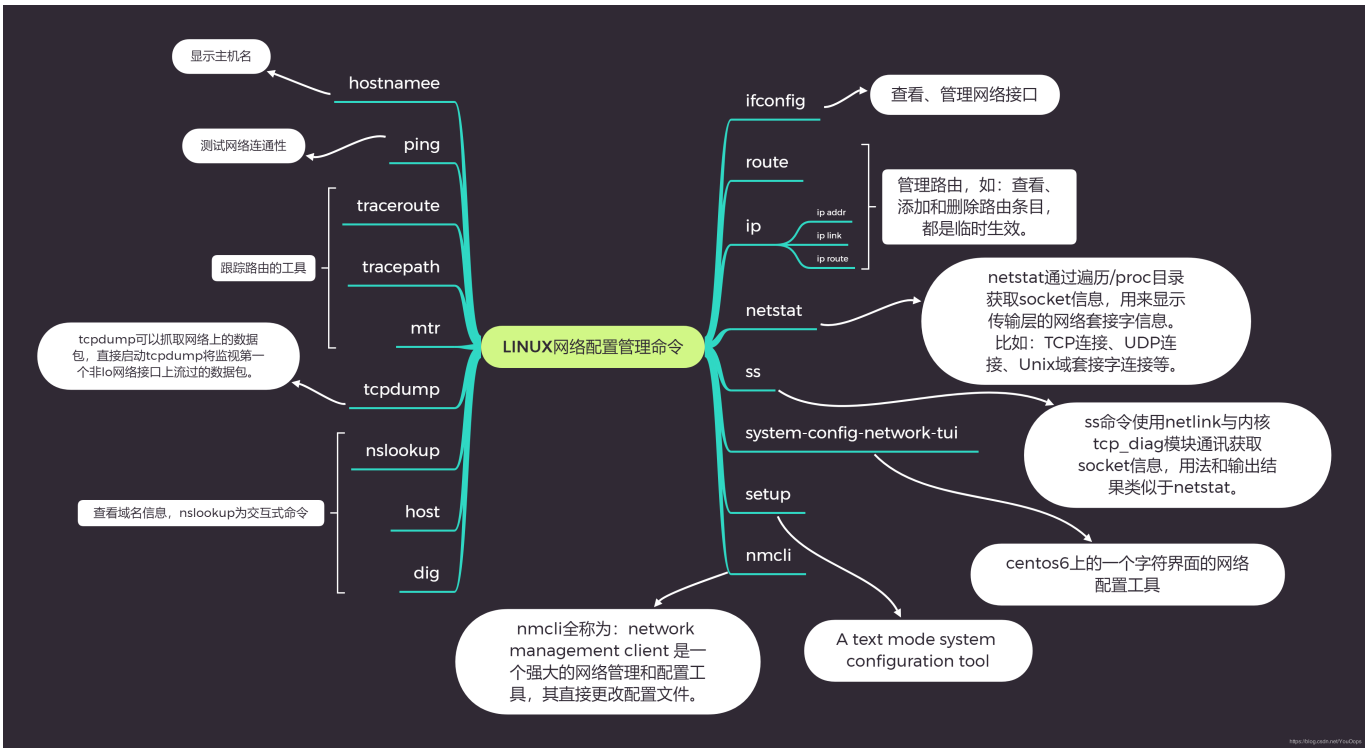


图4. 1. 1–Linux网络相关配置命令

配置文件概览

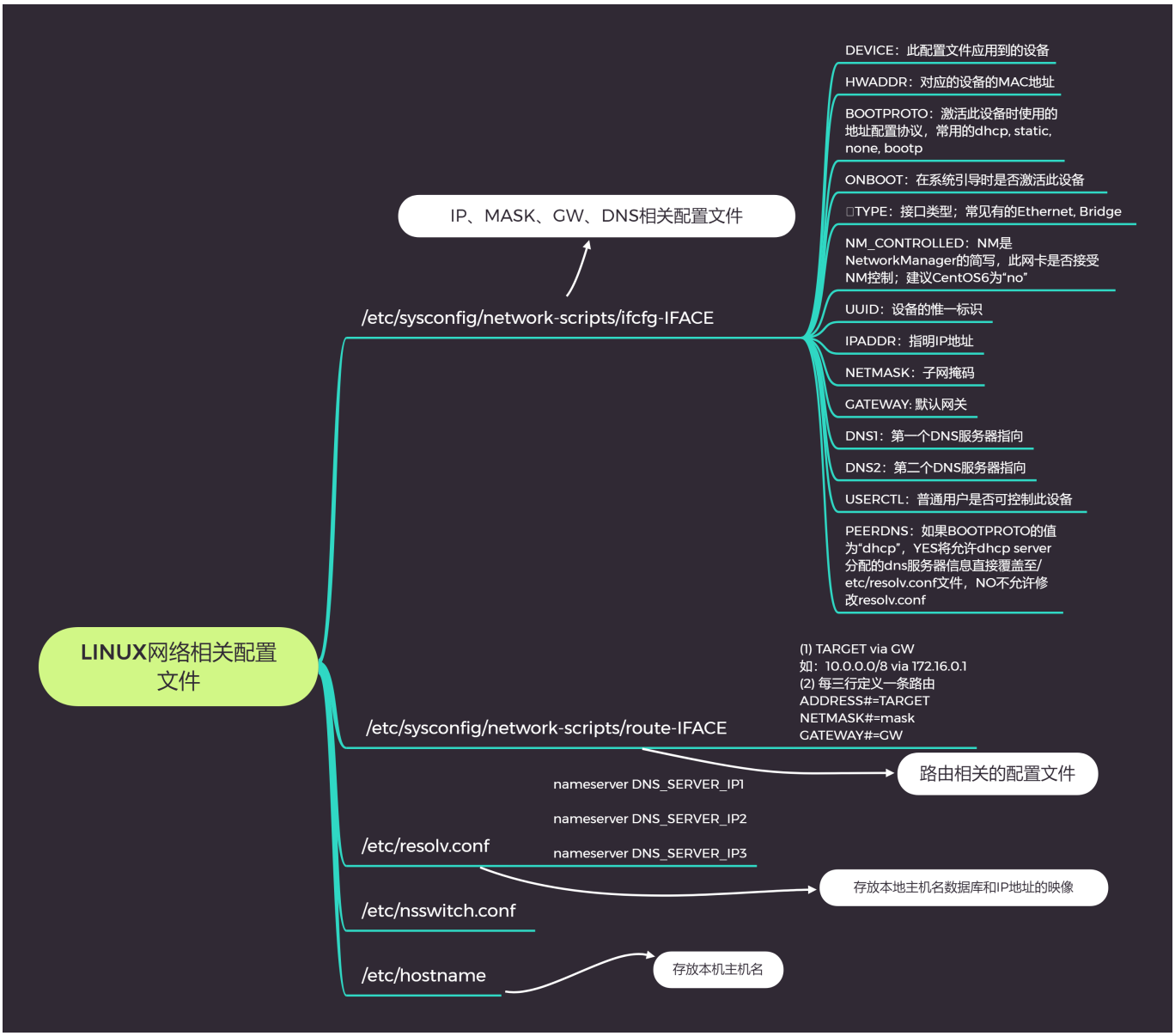


图4. 1. 2–Linux网络相关配置文件

- /etc/sysconfig/network-scripts/ifcfg-ethX

动态配置	静态配置
DEVICE=eth*X* HWADDR=0:02:8A:A6:30:45 BOOTPROTO=dhcp ONBOOT=yes Type=Ethernet	DEVICE=eth*X* HWADDR=0:02:8A:A6:30:45 IPADDR=192.168.0.123 NETMASK=255.255.255.0 GATEWAY=192.168.0.254 ONBOOT=yes Type=Ethernet

2.命令详细介绍

ifconfig

- ifconfig命令

```
ifconfig [interface]
ifconfig -a
ifconfig IFACE [up|down]
                ifconfig interface [aftype] options | address ...
                ifconfig IFACE IP/netmask [up]
                ifconfig IFACE IP netmask NETMASK
注意：该命令更改后立即生效
启用混杂模式：[-]promisc
```

route

- 路由管理命令

```
查看：route -n
添加：route add
        route add  [-net|-host]  target [netmask Nm] [gw Gw] [[dev] If]
目标：192.168.1.3  网关：172.16.0.1
        route add -host 192.168.1.3 gw 172.16.0.1 dev eth0
目标：192.168.0.0 网关：172.16.0.1
        route add -net 192.168.0.0 netmask 255.255.255.0 gw 172.16.0.1 dev eth0
        route add -net 192.168.0.0/24 gw 172.16.0.1 dev eth0
默认路由，网关：172.16.0.1
        route add -net 0.0.0.0 netmask 0.0.0.0 gw 172.16.0.1
        route add default gw 172.16.0.1
删除：route del
        route del [-net|-host] target [gw Gw] [netmask Nm] [[dev] If]
目标：192.168.1.3  网关：172.16.0.1
        route del -host 192.168.1.3
目标：192.168.0.0 网关：172.16.0.1
        route del -net 192.168.0.0 netmask 255.255.255.0
```

ip

- 配置Linux网络属性：ip 命令

```
ip - show / manipulate routing, devices, policy routing and tunnels
ip [ OPTIONS ] OBJECT { COMMAND | help }
OBJECT := { link | addr | route }
ip link - network device configuration
        set dev IFACE
                可设置属性：
                up and down：激活或禁用指定接口
                        ifup/ifdown
        show [dev IFACE]：指定接口
                [up]：仅显示处于激活状态的接口

ip addr { add | del } IFADDR dev STRING
        [label LABEL]：添加地址时指明网卡别名
```

```
        [scope {global|link|host}]：指明作用域
                global: 全局可用
                link: 仅链接可用
                host: 本机可用
        [broadcast ADDRESS]：指明广播地址
        ip addr add 172.16.100.100/16 dev eth0 label eth0:0
        ip addr del 172.16.100.100/16 dev eth0 label eth0:0

ip address show - look at protocol addresses
        [dev DEVICE]
        [label PATTERN]
        [primary and secondary]

ip addr flush   使用格式同show
        ip addr flush dev eth0

ip route - routing table management
        添加路由：ip route add
        ip route add TARGET via GW dev IFACE src SOURCE_IP
                TARGET:
                主机路由：IP
                网络路由：NETWORK/MASK
                ip route add 192.168.0.0/24 via 172.16.0.1
                ip route add 192.168.1.100 via 172.16.0.1
                添加网关：ip route add default via GW dev IFACE
                ip route add  default via 172.16.0.1
        删除路由：ip route del TARGET
        显示路由：ip route show|list
        清空路由表：ip route flush [dev IFACE]  [via PREFIX]
                  ip route flush dev eth0
```

netstat

- 显示网络连接

```
netstat [--tcp|-t] [--udp|-u] [--raw|-w] [--listening|-l] [--all|-a] [--
numeric|-n] [--extend|-e[--extend|-e]]  [--program|-p]
        -t: tcp协议相关
        -u: udp协议相关
        -w: raw socket相关
        -l: 处于监听状态
        -a: 所有状态
        -n: 以数字显示IP和端口
        -e：扩展格式
        -p: 显示相关进程及PID
常用组合：
        -tan, -uan, -tnl, -unl
显示路由表：
        netstat  {--route|-r} [--numeric|-n]
                -r: 显示内核路由表
                -n: 数字格式
```

```
显示接口统计数据：
netstat  {--interfaces|-I|-i} [iface] [--all|-a] [--extend|-e] [--program|-p]
[--numeric|-n]
        netstat -i
        netstat –I=IFACE
        ifconfig -s eth0
```

ss

- 格式：ss [OPTION]... [FILTER]
- netstat通过遍历proc来获取socket信息，ss使用netlink与内核tcp_diag模块 通信获取socket信息

```
        -t: tcp协议相关
        -u: udp协议相关
        -w: 裸套接字相关
        -x: unix sock相关
        -l: listen状态的连接
        -a: 所有
        -n: 数字格式
        -p: 相关的程序及PID
        -e: 扩展的信息
        -m: 内存用量
        -o: 计时器信息
FILTER :  [ state TCP-STATE ] [ EXPRESSION ]
TCP的常见状态:(tcp finite state machine)
        LISTEN: 监听
        ESTABLISHED: 已建立的连接
        FIN_WAIT_2
        SYN_SENT
        SYN_RECV
        CLOSED
EXPRESSION:
        dport =
        sport =
        示例: '( dport = :ssh or sport = :ssh )'
常用组合:
-tan, -tanl, -tanlp, -uan
常见用法
        ss -l 显示本地打开的所有端口
        ss -pl 显示每个进程具体打开的socket
        ss -t -a 显示所有tcp socket
        ss -u -a 显示所有的UDP Socekt
        ss -o state established '( dport = :ssh or sport = :ssh )' 显示所有已建立的
ssh连接
        ss -o state established '( dport = :http or sport = :http )' 显示所有已建立
的HTTP连接
        ss -s 列出当前socket详细信息
```

nmcli

- 功能强大的地址配置工具：nmcli

```
nmcli  [ OPTIONS ] OBJECT { COMMAND | help }
        device - show and manage network interfaces
        nmcli device help
        connection - start, stop, and manage network connections
        nmcli connection help
修改IP地址等属性：
        nmcli connection modify IFACE [+|-]setting.property value
                setting.property:
                ipv4.addresses
                ipv4.gateway
                ipv4.dns1   ipv4.method manual | auto
修改配置文件执行生效：systemctl restart network
        nmcli con reload
nmcli命令生效：  nmcli con down eth0 ;nmcli con up eth0
显示所有包括不活动连接
        nmcli con show
显示所有活动连接
        nmcli con show --active
显示网络连接配置
        nmcli con show  "System eth0"
显示设备状态
        nmcli  dev  status
显示网络接口属性
        nmcli  dev show eth0
创建新连接default，IP自动通过dhcp获取
        nmcli  con add con-name default  type Ethernet ifname eth0
删除连接
        nmcli con del default
创建新连接static ，指定静态IP，不自动连接
        nmcti con add con-name static   ifname eth0 autoconnect no type Ethernet
ipv4.addresses 172.25.X.10/24 ipv4.gateway   172.25.X.254
启用static连接配置
        nmcli con up static
启用default连接配置
        nmcli con up default
查看帮助
        nmcli con add help
修改连接设置
    nmcli con mod “static” connection.autoconnect no
    nmcli con mod “static”  ipv4.dns 172.25.X.254
    nmcli con mod “static”  +ipv4.dns  8.8.8.8
    nmcli con mod “static”  -ipv4.dns  8.8.8.8
    nmcli con mod “static” ipv4.addresses “172.16.X.10/24  172.16.X.254”
    nmcli con mod “static”  +ipv4.addresses 10.10.10.10/16
DNS设置，存放在/etc/resolv.conf文件中
        PEERDNS=no 表示当IP通过dhcp自动获取时，dns仍是手动设置，不自动获取
等价于下面命令：
        nmcli con mod “system eth0” ipv4.ignore-auto-dns yes
修改连接配置后，需要重新加载配置
        nmcli  con reload
        nmcli  con down “system eth0” 可被自动激活
```

```
        nmcli  con up  “system eth0”
        nmcli dev dis eth0 禁用网卡，访止被自动激活
图形工具
        nm-connection-editor
字符工具
        nmtui
        nmtui-connect
        nmtui-edit
        nmtui-hostname
```

五.多网卡绑定技术

1.概念

- 将多块网卡绑定同一IP地址对外提供服务，可以实现高可用或者负载均衡。直接给两块网卡设置同 一IP地址是不可以的。通过bonding，虚拟一块网卡对外提供连接，物理网卡的被修改为相同的MAC地址。
- Bonding工作模式
    - 共7种模式：0-6 Mode

Mode 0 (balance-rr)	轮询（Round-robin）策略，从头到尾顺序的在每 一个slave接口上面发送数据包。本模式提供负载均衡和容错的能力
Mode 1 (active-backup)	活动-备份（主备）策略，只有一个slave被激活，当且仅当活动的slave接口失败时才会激活其他slave.为了避免交换机发生混乱此时绑定的MAC地址只有一个外部端口上可见
Mode 3 (broadcast)	广播策略，在所有的slave接口上传送所有的报文,提供容错能力

active-backup、balance-tlb 和 balance-alb 模式不需要交换机的任何特殊配置。其他绑定模式需要配置交换机以便整合链接。如：Cisco 交换机需要在模式 0、2 和 3 中使用 EtherChannel，但在模式4中需要LACP和EtherChannel

2.修改配置文件实现bond0

- 创建bonding设备的配置文件结合网卡硬件就可以实现bond0

```
/etc/sysconfig/network-scripts/ifcfg-bond0
        DEVICE=bond0
        BOOTPROTO=none
        BONDING_OPTS="miimon=100  mode=0"
/etc/sysconfig/network-scripts/ifcfg-eth0
        DEVICE=eth0
        BOOTPROTO=none
        MASTER=bond0
```

```
        SLAVE=yes
        USERCTL=no
/etc/sysconfig/network-scripts/ifcfg-eth1
        DEVICE=eth1
        BOOTPROTO=none
        MASTER=bond0
        SLAVE=yes
        USERCTL=no
查看bond0状态：/proc/net/bonding/bond0
miimon 是用来进行链路监测的。如果miimon=100，那么系统每100ms 监测
一次链路连接状态，如果有一条线路不通就转入另一条线路
删除bond0
        ifconfig bond0 down
        rmmod bonding
详细帮助：
        /usr/share/doc/kernel-doc- version/Documentation/networking/bonding.txt
        https://www.kernel.org/doc/Documentation/networking/bonding.txt
```

3.使用nmcli实现bonding

- 添加bonding接口 `nmcli con add type bond con-name mybond0 ifname bond0 mode active-backup`
- 添加从属接口 `nmcli con add type bond-slave ifname ens7 master bond0` `nmcli con add type bond-slave ifname ens3 master bond0` 注：如无为从属接口提供连接名，则该名称是接口名称加类型构成
- 要启动绑定，则必须首先启动从属接口 `nmcli con up bond-slave-eth0` `nmcli con up bond-slave-eth1`
- 启动绑定 `nmcli con up mybond0`

六.Linux软件网桥

1.概念

- 桥接：把一台机器上的若干个网络接口“连接”起来。其结果是，其中一个网 收到的报文会被复制给其他网口并发送出去。以使得网口之间的报文能够互 相转发。网桥就是这样一个设备，它有若干个网口，并且这些网口是桥接起来 的。与网桥相连的主机就能通过交换机的报文转发而互相通信。
- 如图6.1.1所示，主机A发送的报文被送到交换机S1的eth0口，由于eth0与eth1、 eth2桥接在一起，故而报文被复制到eth1和eth2，并且发送出去，然后被主机B和交换机S2 接收到。而S2又会将报文转发给主机C、D。

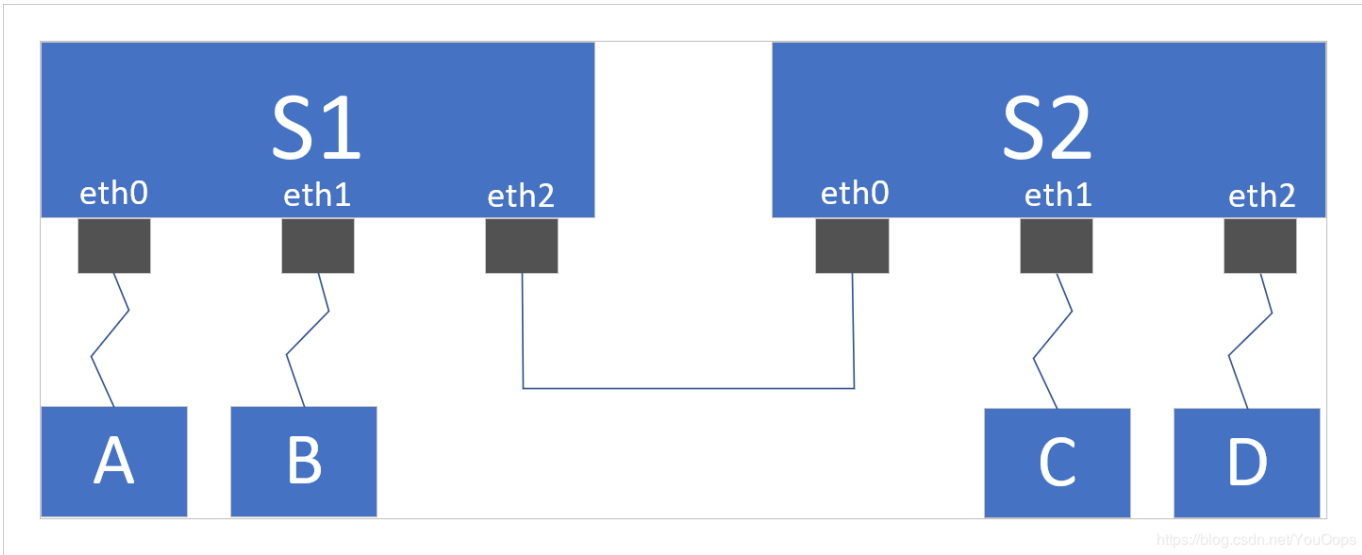


图6. 1. 1–交换机示意

2.实现

- 创建软件网桥 nmcli con add con-name mybr0 type bridge ifname br0 nmcli con modify mybr0 ipv4.addresses 192.168.0.100/24 ipv4.method manual nmcli con add con-name br0-port0 type bridge-slave ifname eth0 master br0
- 查看配置文件 cat /etc/sysconfig/network-scripts/ifcfg-br0 cat /etc/sysconfig/network-scripts/ifcfg-br0-port0
- 工具包 yum install bridge-utils
- 查看网桥 brctl show
- 查看CAM表 brctl showmacs br0
- 添加和删除网桥
  brctl addbr | delbr br0
- 添加和删除网桥中网卡
  brctl addif | delif br0 eth0 注意：NetworkManager只支持以太网接口接口连接到网桥，不支持聚合接口

七.Ubuntu的网络设置

网卡名称

- 默认ubuntu的网卡名称和CentOS 7类似，如：ens33，ens38等 修改网卡名称为传统命名方式：

```
        vim /etc/default/grub
        GRUB_CMDLINE_LINUX="net.ifnames=0"
生效新的grub.cfg文件
        grub-mkconfig -o /boot/grub/grub.cfg
reboot
```

- Ubuntu的.yaml网络配置文件需要严格控制缩进

```
官网文档：
https://help.ubuntu.com/lts/serverguide/network-configuration.html.zh-CN

配置自动获取IP
cat /etc/netplan/01-netcfg.yaml
network:
    version: 2
    renderer: networkd
    ethernets:
       ens33:
           dhcp4: yes
修改网卡配置文件后需执行命令生效：netplan apply

配置静态IP：
cat /etc/netplan/01-netcfg.yaml
network:
    version: 2
    renderer: networkd
    ethernets:
        eth0:
            addresses:
                              - 192.168.6.10/24
                              - 10.10.10.10/24
            gateway4: 192.168.6.1
            nameservers:
                search: [mydomain, otherdomain]
                addresses: [223.5.5.5, 8.8.8.8, 1.1.1.1]
```

- 查看ip和gateway

```
        ip addr
        route -n
查看DNS
        ls -l /etc/resolv.conf
        lrwxrwxrwx 1 root root 39 Dec 12 11:36 /etc/resolv.conf ->
../run/systemd/resolve/stub-resolv.conf

        systemd-resolve --status
修改主机名
        hostnamectl set-hostname  ubuntu1904
```

八.Linux下的网络测试工具

hostname显示主机名

man hostname

ping测试网络连通性

man ping

ip/route显示正确的路由表

跟踪路由

man traceroute man tracepath man mtr

确定名称服务器

man nslookup man host man dig

注脚

各层协议加上的报文首部。每层协议盒都要去检查报文首部中的协议标识，以确定接收数据 的上层协议。这个过程称作分用(Demultiplexing) [^2]:为了计算一份数据报的IP检验和，首先把检验和字段置为0。然后，对首部中每个16bit 进行二进制反码求和(整个首部看成是由一串16bit的字组成)，结果存在检验和字段中。当 收到一份IP数据报后，同样对首部中每个16bit进行二进制反码的求和。由于接收方在计算过 程中包含了发送方存在首部中的检验和，因此，如果首部在传输过程中没有发生任何差错， 那么接收方计算的结果应该为全1。如果结果不是全1（即检验和错误），那么IP就丢弃收到的 数据报。但是不生成差错报文，由上层去发现丢失的数据报并进行重传。ICMP、IGMP、UDP和TCP 都采用相同的检验和算法，尽管TCP和UDP除了本身的首部和数据外，在IP首部中还包含不同的字段。 在RFC1071[Braden, Borman and Patridge 1988]中有 关于如何计算Internet检验和的实现技术。 由于路由器经常只修改TTL字段（减1），因此当路由器转发一份报文时可以增加它的检验和，而不需要对 IP整个首部进行重新计算。RFC1141[Mallory and Kullberg 1990]为此给出了一个很有效的方法。