计算机网络基础

红岩网校工作站运维安全部 黄凯升

（暑假通识课专版）

2022/7/26

Table Of Contents

Part I. Network Architecture and Protocols

概论 —— 网络体系结构与协议

网络协议

- 什么是网络协议?
- 协议是一组控制数据交互过程的通信规则，它明确地规定了数据交换的格式和顺序。
- 组成要素:
  - 语义：解释控制信息的每个部分的意义。 → 做什么
  - 语法：语法是用户数据与控制信息的结构、格式，数据出现的顺序。 → 怎么做
  - 时序：对于事件发生顺序的详细说明。 → 谁先谁后

协议、层次与接口

- 协议：事先约定的通信规则
  - 协议栈：为一个特定的系统制定的一组协议
- 层次：处理计算机网络问题的基本方法
  - 层次结构将复杂的问题抽象为较为简单的数个层，从而简化问题
  - 层次结构的优点：
    - 各层之间相互独立，上层只需要关心下层提供的接口，不需要关心下层实现
    - 灵活性好，下层的功能和接口保持一致的情况下，技术变更不会影响上层
    - 易于实现和标准化，各个层次规范定义的情况下工程实现更加容易，
- 接口：相邻层之间信息交换的连接点，接口需要被明确定义

网络体系结构

- 网络体系结构的组织方式 —— 层次结构
- 定义：网络层次结构模型和各层协议的集合
- 网络体系结构模型：OSI 参考模型和 TCP/IP 参考模型
  - OSI 模型：1974 年国际标准化组织 (ISO) 发布的“开放系统互连”参考模型
  - TCP/IP 参考模型：1974 年以来被广泛采用并最终被 IETF 标准化的模型
- 网络体系结构的共同特点：
  - 发送数据时，数据自顶向下依次被封装。
  - 接收数据时，数据自底向上依次被展开。
  - 在这个过程中，对上层而言底层的处理是透明的。

OSI 参考模型

- OSI 参考模型如右侧表格所示。
- 物理层：实现比特流的透明传输
- 数据链路层：建立数据连接，并通过差错控制和流量控制等方式将有差错的物理线路转变为无差错的数据链路
- 网络层：根据路由选择算法为主机实现网络互联
- 传输层：为应用程序提供可靠的端到端连接和数据传输服务
- 会话层：负责维护连接的建立、管理、终止和数据交换
- 表示层：负责进行数据格式的变换
- 应用层：最高层，应用程序的通信控制

应用层
表示层
会话层
传输层
网络层
数据链路层
物理层

TCP/IP 参考模型

- TCP/IP 模型相较 OSI 模型显得更加简洁。
- OSI 模型和 TCP/IP 模型各个层级的对应关系如右图。

Part II. Data Link Layer

数据链路层

数据链路层简介

- 基本的传输单元：帧
- 帧是一组数据组成的数据块。
- 数据链路层的主要功能：数据帧传输和差错控制。
  - 数据帧传输的功能需要能够管理数据链路、进行流控和寻址等。
- 数据链路层内部也有分层，这里不过多赘述。
- 数据链路层主要的协议有点对点协议 (PPP) 和介质访问控制协议 (MAC) 等。

局域网

- 以太网 (Ethernet) 是现在最广泛使用的局域网标准。
- 以太网使用 MAC 协议，主机互联采用的是 MAC 地址。
  - MAC 地址是一个 48 位的地址。每块网卡出厂时都会带有一个唯一的 MAC 地址。
  - MAC 地址书写时通常用 6 组冒号隔开的 2 位十六进制数，例如 00:e2:c3:11:45:14。
  - MAC 地址并不是一个安全的验证设备的方式，它可以轻易被伪造。
- 目前，局域网中的数据链路层（二层）设备主要是交换机和无线 AP。

以太帧和广播域

- 以太帧的组成部分主要有前导码、帧开始符、以太报文头和校验码组成。
- 其中，以太报文头包含目的 MAC 地址、源 MAC 地址。
- 以太网是一种广播网络，任意一个节点发出的信息都能够被其它所有节点收到。
- 带有目标 MAC 地址为 FF:FF:FF:FF:FF:FF 的以太帧为广播帧，它能够传递的范围称为广播域。
- 位于同一广播域内的主机可以直接通过 MAC 地址通信。

冲突域

- 在共享的物理介质上，同一时间只能有一个主机收发数据。否则将会出现冲突导致通信失败。
- 冲突域即为同一时间有多台主机传输数据时触发冲突的范围。
- 集线器本身没有数据识别能力，收发数据是广播的方式，因此集线器下的主机处于同一个冲突域。
- 交换机能够识别 MAC 地址并建立端口转发表，因此它可以将冲突域隔离到端口。

桥接和 VLAN

- 实际应用中，我们经常需要将两个不同的以太网连接起来。
- 桥接是将两个不同的网络的广播域连接起来的操作，桥接的两个网卡处于同一个数据链路层。
- VLAN 的作用则是分割，它能够在一个数据链路层上构建出逻辑上隔离的广播域。不同 VLAN 中的主机无法直接互访。

MTU

- 最大传输单元 (Maximum Transmission Unit, MTU) 表示上层协议一次能发送的最大数据量。
- 在接下来的网络层中我们会讲到 IP 分片。如果用户传输的数据大于 MTU，则数据会被切分成若干分片进行发送，由接收方进行重组。
- 通常以太网 MTU 为 1500，但如果使用了 PPPoE 拨号上网或 VPN 等方式，这些虚拟网卡的 MTU 会偏小。

PPP

- 点对点协议 (Point-to-Point Protocol, PPP) 是另一种数据链路层协议，它能够在两台主机之间建立起点对点连接，并提供验证和数据加密等功能。
- PPP 连接下上层协议发送的数据会被封装成 PPP 帧进行发送。
- PPP 也被扩展和其它协议配合工作，例如：
  - 以太网上的 PPP (PPPoE) 是现在宽带运营商为用户提供接入的最广泛的方式。
  - 许多常见的 VPN 协议也使用了 PPP 协议。

Part III. Network Layer

网络层

网络层简介

- 基本的传输单元：分组
- 因为网络总是不稳定的，数据需要经过分片后形成分组数据包进行发送。
- 现行的网络层协议：IP 协议
  - IP 协议分为 IPv4 和 IPv6 两个版本。我们讲解时示例以 IPv4。
  - IP 协议是无连接和不可靠的，它提供的是一种“尽力而为”的服务。
    - 无连接：即 IP 协议本身是无状态的，每个分组的传输相互独立。
    - 不可靠：意味着 IP 协议不能保证每个分组都正确地、不丢失和顺序抵达目的主机。IP 协议提供的是“尽力而为”的传输服务。
  - IP 是点对点的网络层通信协议。

IP 地址和子网掩码

- IP 地址长度为 32 位，以点分十进制的形式书写，每 8 位为一组。
- IP 地址的基本构成为网络号和主机号。
- 这里我们要引入子网的概念：
  - 如果两台计算机能够直接不经过路由器直接通过主机号通讯，那么我们说这两台计算机处于同一个子网下。
- 子网掩码的作用则是标示出 IP 地址中的网络号和主机号，其也为 32 位。
- 例如：

IP：　　192.168.77.234　　(11000000 10101000 01001101 11101010)

子网掩码：255.255.255.0　　(11111111 11111111 11111111 00000000)

IP 地址和子网掩码

```
 (11000000 10101000 01001101 11101010)
                                      &
 (11111111 11111111 11111111 00000000)
=(11000000 10101000 01001101 00000000) (192.168.77.0) 网络号


 (11000000 10101000 01001101 11101010)
                                      &
~(11111111 11111111 11111111 00000000)
=(00000000 00000000 00000000 11101010) (0.0.0.234)    主机号
```

IP 分类与 CIDR

- 传统的 IP 分类分为 A、B、C、D、E 五类地址：
  - A 类地址：0.0.0.0 - 127.255.255.255 ，其子网掩码为 255.0.0.0
  - B 类地址：128.0.0.0 - 191.255.255.255，其子网掩码为 255.255.0.0
  - C 类地址：192.0.0.0 - 223.255.255.255，其子网掩码为 255.255.255.0
  - D 类地址：224.0.0.0 - 239.255.255.255，这是组播 IP 段
  - E 类地址：240.0.0.0 - 255.255.255.255，这些地址被保留
- 可以看出，传统的 IP 分类方式固定了不同类型 IP 的子网掩码，这就使得子网划分较为不灵活。人们为了解决这个问题，提出了 CIDR。

IP 分类与 CIDR

- 无类别域间路由 (Classless Inter-Domain Routing, CIDR) 是一个用于给用户分配 IP 地址和在互联网上有效地路由 IP 数据包的 IP 地址归类的方法。
  - 在 CIDR 下人们提出了可变长子网掩码来进行任意长度的前缀划分，从而可以对地址块进行精细的划分与聚合。
  - CIDR 描述的地址块使用 IP/PREFIXLEN 的格式描述。
  - 例如，一个传统的 C 类地址 192.168.1.123，子网掩码为 255.255.255.0，使用 CIDR 可以写成 192.168.1.123/24。
  - 其中，24 指前缀长度，即网络号在地址中的长度。

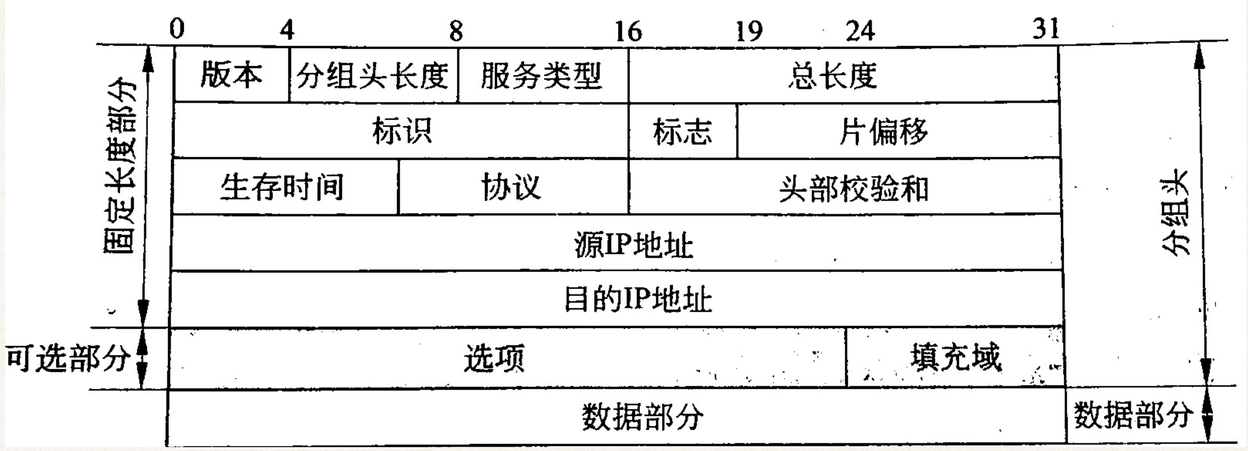
IP 分类与 CIDR

- 通过前缀长度或子网掩码，我们就可以从一个任意给定的 CIDR 块中匹配出网络号和主机号，接下来无论是进行路由还是其它操作都非常简单。
- 在 IPv4 中前缀长度的取值范围是 [0, 32]，这是因为 IPv4 只有 32 位。
- 例：重庆邮电大学所拥有的公网 IP 地址范围为 202.202.32.0 – 202.202.47.255 和 222.177.140.0 - 222.177.140.127，
- 可以用两个 CIDR 块描述：
- 202.202.32.0/20　　202.202.32.0 - 202.202.47.255
- 222.177.140.0/25　　222.177.140.0 - 222.177.140.127

特殊的 IP 地址

- 广播地址：
  - 直接广播地址：主机号全 1 的地址。路由器将其以广播形式发送到特定网络的所有主机。
  - 受限广播地址：255.255.255.255。路由器将其以广播形式发送到本地网络的所有主机，不对外转发。
- 特定主机地址：网络号全 0 的地址。路由器不向外转发而是将其转发到本地网络拥有对应主机号的主机上。
- 回环地址：127.0.0.0/8 下的所有地址。操作系统将发往这些地址的数据包作为连接本机处理。TCP/IP 规定 127.0.0.0/8 的分组不能出现在任何网络中。

IP 分组的结构



- 这里以 IPv4 为例子。分组由分组头和数据部分构成。
- 特别值得注意的是这里的生存时间，它和路由有着很大的关系。
- IPv4 分组头长度可变，但最少为 20 字节，可选部分最大可达 40 字节，即总共可达 60 字节长。

IPv6 —— 下一代 IP 地址

- IPv6 是下一代的 IP 地址，长度为 128 位。
- 比起 32 位的 IPv4，IPv6 可以表示的地址数量达到了 $2^{128} \approx 3.40 \times 10^{34}$，而 IPv4 仅能表示约 42 亿个地址。
- IPv4 使用点分十进制表示，IPv6 使用冒号分割十六进制的形式，分为八组，每组 16 位长度。连续的 0 可以使用 :: 进行压缩（一个地址中只能压缩一次）。
- 例：重庆邮电大学官网的 IPv6 地址 `2001:da8:c807:20:202:202:32:60`
- IPv4 现在已经枯竭，最后一块 IPv4 地址于 2019 年 11 月 26 日被分配。

网关和路由

- 网关是网络层上实现网络互连的设备，可将数据包跨子网传输。
- 狭义上的路由器就是有路由功能的设备。
- 相同子网下的主机可以通过主机号直接互连，不需要经过网关。
- 跨子网的主机原则上需要网关进行路由才能进行互连。
- 路由是对相同子网或不同子网间传输的数据包进行路径选择的过程。
- 路由决定了数据包接下来将被发送到哪里。
- 计算机中将数据包进行路由所依据的是路由表。

路由的过程

- 对于一个特定的数据包，系统会首先提取出它的目的地址(destination)
- 对于这个 destination，系统查询路由表，以一定的规则命中最合适的路由规则，然后按照这个路由规则转发数据包，转发时生存时间 (TTL) -1。
- 生存时间 (Time-To-Live, TTL) 在 IP 分组头部，它决定了一个包最多被转发多少次，这是为了防止错误配置的路由导致数据包在网络中循环从而严重消耗资源。
- 如果无法命中任何可用的路由，则抛出错误信息 (Linux 中为网络不可达)

路由的过程

- 路由表是一系列路由组成的集合。
- 一条路由的基本要素有：目的地址、下一跳地址、源地址、接口、Scope (作用域)、类型、Metric (代价)。
- 这些要素未必需要全部满足，例如在链路上的路由可能就没有下一跳地址，通常的路由也不指定源地址，但目的地址不可或缺。
- 路由选择的基本原则：
  - 精确匹配原则：优先选择和数据包头部信息更匹配的路由。
  - 最长前缀原则：多条可选路由时优先选择前缀最长、子网最小的子网路由。
  - 最小代价原则：多条可选路由时优先选择代价最小的路由。

路由的过程

- 下面是一个简化后的路由表的例子：
  - `default            via 172.23.0.254`
  - `172.23.0.0/16     via 172.23.0.254`
  - `10.16.0.0/13      via 100.69.19.50`
  - `172.16.9.0/24     via 7.0.0.5`
  - `172.22.161.0/24   via 100.69.19.50`
  - `172.23.26.0/23    via 100.69.19.50`
  - `202.202.0.0/19    via 100.69.19.50`
  - `202.202.32.0/20   via 100.69.19.50`

ICMP

- 互联网控制消息协议 (Internet Control Message Protocol, ICMP) 在 IP 协议中负责发送控制消息，以及提供关于网络的差错报告与询问、控制机制。
- Ping 命令使用的是 ICMP Echo 消息，对方主机收到 ICMP Echo 后会回复一个 ICMP Reply 消息。
- 当数据包 TTL 减至 0 时，收到该数据包的路由器向源主机返回一个 ICMP Time Exceeded 报文并丢弃数据包。基于这个特性能够实现路由追踪。
- ICMP Redirect 报文由路由器产生，它能够通知主机更新路由表信息从而实现链路的优化。

ARP

- 地址解析协议 (Address Resolution Protocol, ARP) 是用于实现网络层地址和数据链路层地址映射的协议。
- ARP 的工作方式是广播，主机通过向本地网络发送目的 MAC 地址为 FF:FF:FF:FF:FF:FF 的广播包，询问所有主机是否持有目的 IP。当一台主机拥有所请求的 IP 时，它以单播的形式告知询问的主机自己的 MAC 地址。
- 当主机请求的 IP 不在同一子网时，它会转而去询问路由器的 MAC 地址，并将以太帧的目的 MAC 地址设置为路由器的 MAC 地址发出。
- 在 IPv6 网络中，ARP 被邻居发现协议 (Neighbor Discovery Protocol) 取代，后者相对于 ARP 有更多的功能。

组播

- IP 网络传输有三种方式：单播、组播和广播
  - 单播就是点对点的数据传输，也是目前最常见的数据传输方式。如果一个主机希望向多个主机发送数据，它需要将数据逐份发送到所有目标主机。
  - 广播是简单的一对多数据传输，所有本地网络的主机都会收到广播数据，无论它们是否乐意，并且广播仅限于本地网络使用。
- 组播则综合了单播和广播的优点，避免了源主机大量的数据拷贝负担，同时规避了广播的接收负担和本地网络限制，实现了高效的一对多传输。
- 人们通过组播组的方式定义一组接受组播的主机。在 IPv4 规范中，组播 IP 范围为 D 类 IP 地址 (224.0.0.0/4)。
- 我们通过 IGMP 协议进行组播的控制，这里受限于篇幅不过多讲解。

NAT

- 网络地址转换 (Network Address Translation, NAT) 是一种部署在网关上的伪装技术。
- 我们可以从一个方面把它分为以下类型：
  - 基本的网络地址转换 (NAT)：
    - 仅支持地址转换，端口不变。
    - 一对一映射，因此需要维护一个外部地址池。
  - 网络地址端口转换 (NAPT)：
    - 目前应用最广泛的 NAT 类型，同时转换地址和端口号。
    - 由于依赖于传输层端口号的变化，支持的传输层协议有限 (通常为 TCP 和 UDP)。
    - 可以多对一映射，外部需要的地址可以较少。
    - 可分为锥形 NAT 和对称 NAT。

Part IV. Transport Layer

传输层

传输层简介

- 基本的传输单元：数据包
- 现行的传输层协议：TCP 和 UDP
- 传输层的基本功能：负责两台主机之间的进程进行网络通信提供服务。
- 端口号：传输层协议引入了端口号的概念，用于区分不同进程提供的服务。TCP 和 UDP 的端口号的范围都为 1 - 65535。

UDP

- 用户数据报协议 (User Datagram Protocol, UDP) 是一种面向报文的无连接的传输层协议。
  - 面向报文：UDP 对发送数据不进行过多处理，在发送端仅增加一个 UDP 头部就可以发送报文，接收端也仅需要去除头部就可以得到原始数据。
  - 无连接：UDP 协议不需要进行连接的建立，双方可以通过端口号直接进行数据收发。
- UDP 是不可靠的，表现在：
  - UDP 不保证数据都能够有序抵达接收端。接收端收到报文无需应答，发送端也不会确认数据是否抵达，双方都不清楚发送过程中是否出现了丢包。
  - UDP 没有拥塞控制，在网络拥堵时也不会调整数据的收发速率。

UDP

- UDP 的数据包结构：

- UDP 校验和校验数据的范围包括 UDP 伪头部 (主要是 IP 头部的信息)，UDP 头部信息和数据，如果接收端计算出的校验和和提供的校验和不匹配，则认为数据包损坏。
- UDP 校验和是可选的，应用程序可以选择不使用校验和 (置 0)。

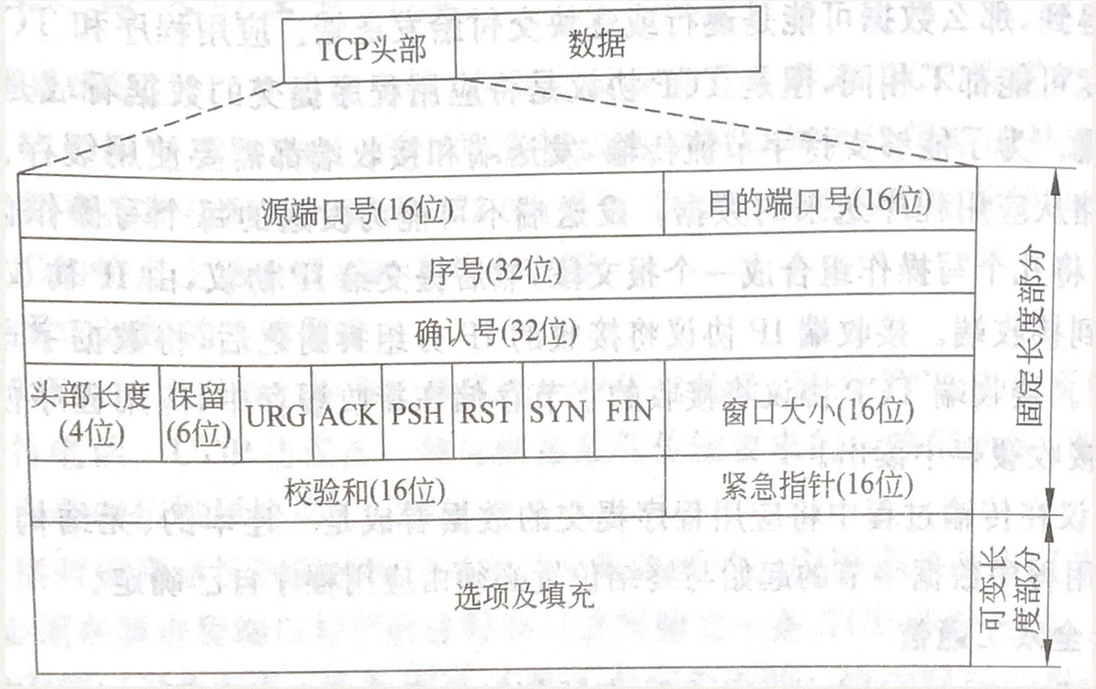
UDP

- 由于 UDP 是无连接的，UDP 可以实现单播、组播和广播三种传输方式。
- UDP 的应用场景主要是对差错不敏感，但对于实时性有较高要求的场景，另外还有一些特殊用途。
- 常见的 UDP 应用：
  - P2P 下载和 NAT 穿透，利用了 UDP 无连接的特性来穿越 NAT 暴露端口
  - 绝大多数 MOBA 游戏
  - IP 电话、视频聊天和直播串流

TCP

- 传输控制协议 (Transmission Control Protocol, TCP) 是一种面向连接的字节流式传输层协议。
  - 面向连接：TCP 在进行数据收发之前，需要通信双方建立连接。
  - 字节流：应用无需考虑报文大小，TCP 对于上层而言是可以进行连续传输的管道。
- 不同于 UDP，TCP 是可靠的，这体现在：
  - TCP 的数据收发是存在跟踪、确认和重传机制的，通信双方能够获知数据是否正确地传输。且遇到丢包时，TCP 能够进行重传，确保数据能够正确有序抵达。

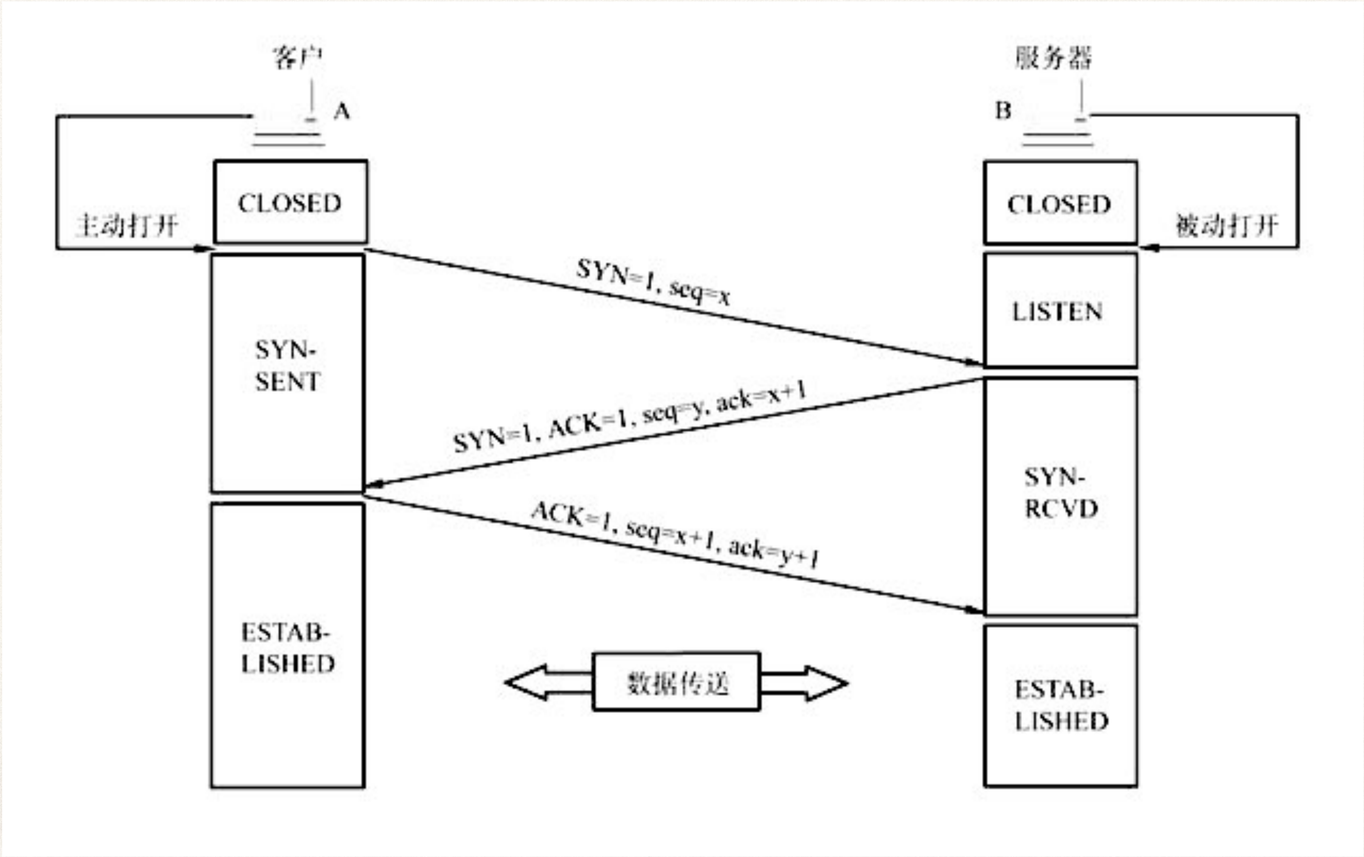
TCP 报文格式



- TCP 报头的长度为 20 ~ 60 字节，其中前 20 字节是必须的，后 40 字节为可变长度的选项部分。

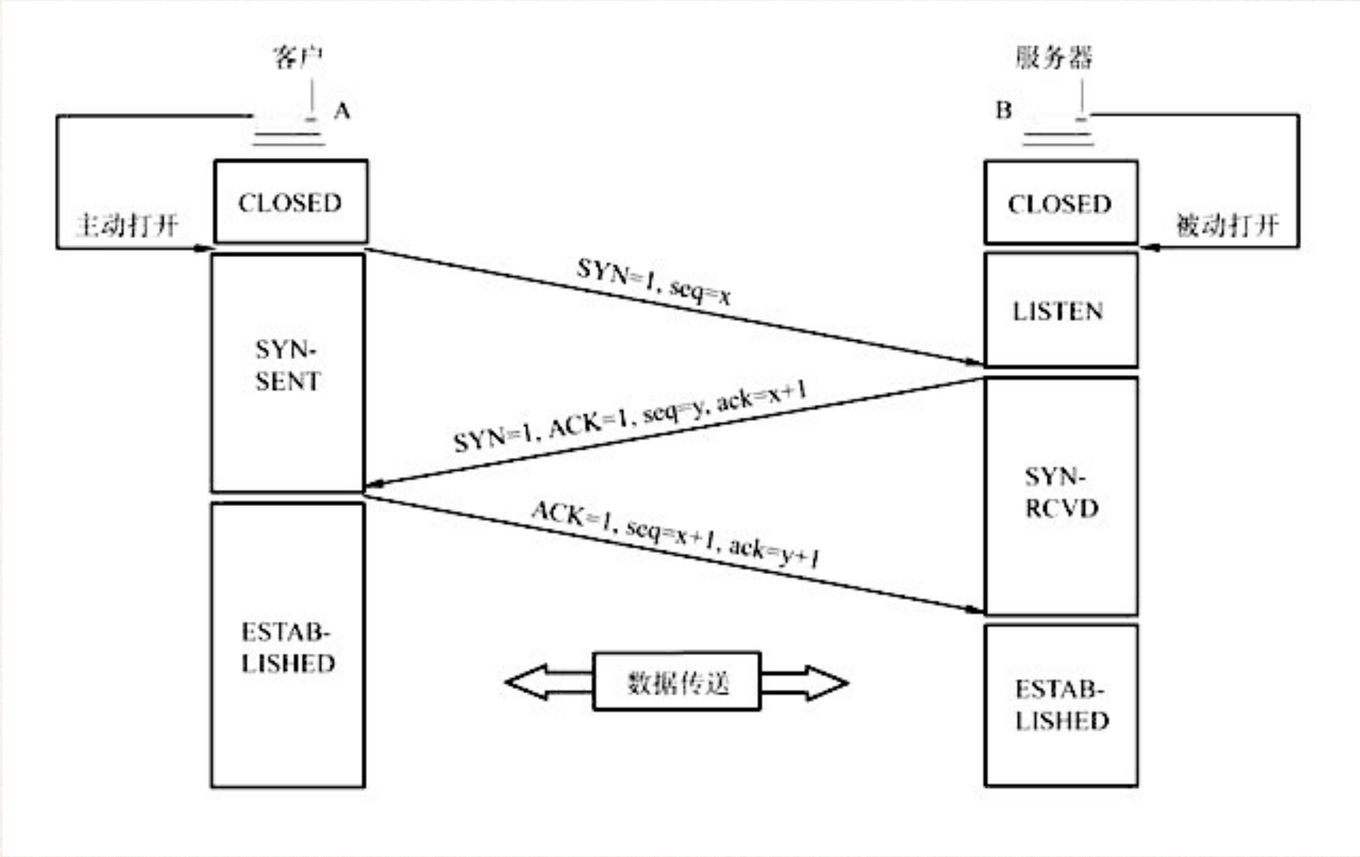
TCP 连接

- TCP 是有连接的，因此通信时需要进行连接的建立和释放。
- TCP 的连接建立和释放可以简单地概括为“三次握手，四次挥手”
- TCP 连接的建立过程：



TCP 连接的建立

- 连接建立时，首先客户端向服务端发送一个 SYN (synchronize) 包，并随机设置一个包序号 seq=x，连接进入 SYN-SENT 状态。
- 服务端收到 SYN 后，向客户端返回一个 SYN-ACK 包。其中 ACK 字段为 x+1，并且服务端也会随机生成一个包序号 seq=y。
- 客户端收到 SYN-ACK 后，向服务端发送一个 ACK (acknowledge) 包，其中 seq 字段为 x+1，ack 字段为 y+1。连接状态变为 ESTABLISHED，并立即开始数据传输。

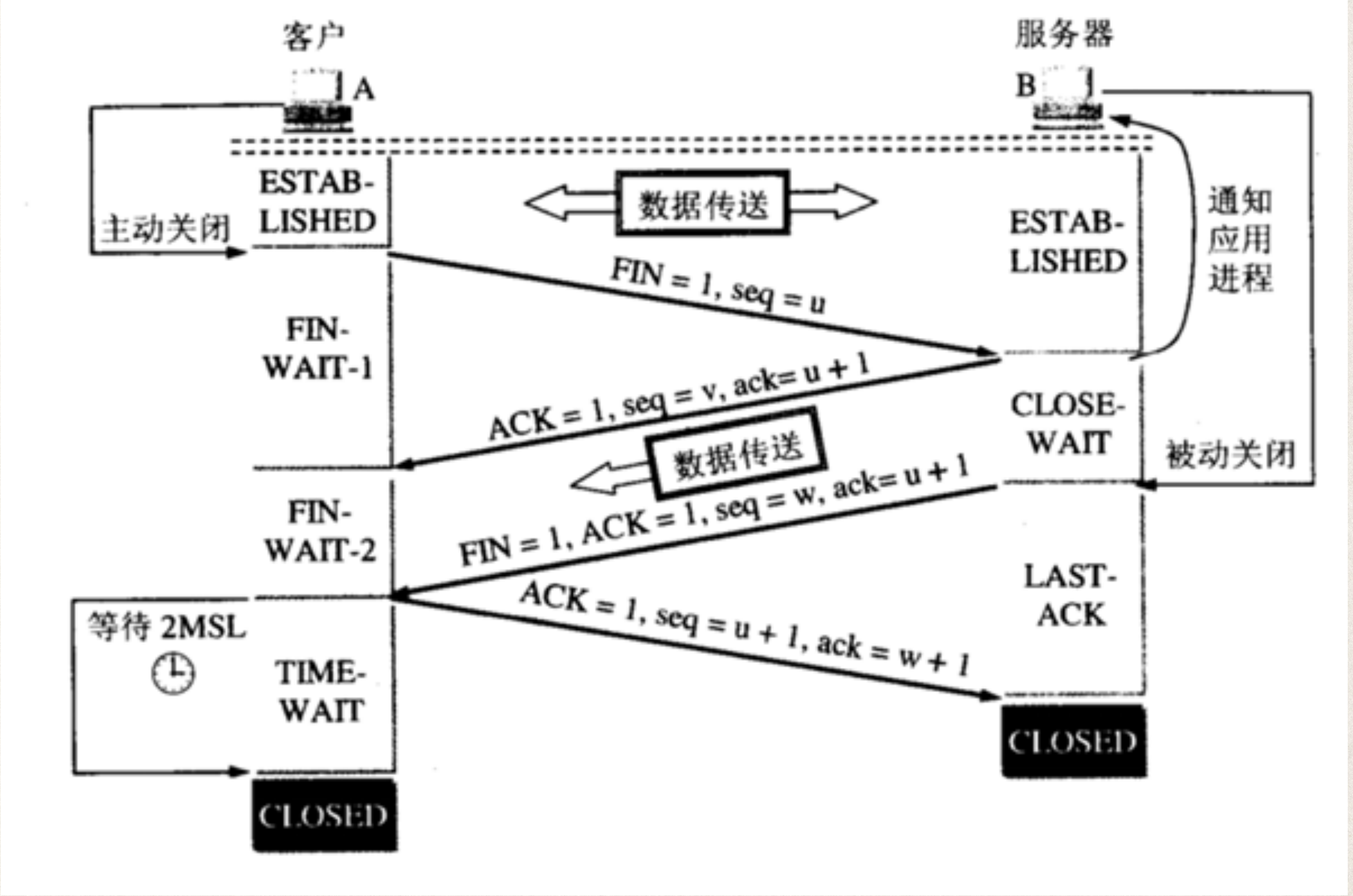


TCP 数据传输

- 可以看到，TCP 连接建立时，通信双方分别会为自己准备一个随机的包序号。它是 TCP 能够实现可靠传输的关键。
- 传输数据时，发送方发送的每一个数据包 seq 都为上一个数据包的 seq + 1。同时，对于收到的每一个数据包，接收方都会返回一个 ack = seq + 1 的确认包。
- 这样显而易见的好处在于：
  - 超时未收到确认包时可以视为包丢失并启动重传。
  - 通过进行数据包的编号可以避免数据乱序，保证其流式传输的特性。
- 总的来说，数据包编号机制是为 TCP 的可靠传输服务的。

TCP 连接的释放

- 连接释放时，首先客户端向服务端发送一个 FIN (finish) 包，对客户端而言连接进入 FIN-WAIT 状态。
- 服务端收到 FIN 后，向客户端返回一个 ACK 包。但并不急于关闭连接，而是等待确认数据全部传输完成。
- 确认数据全部传输完成后，服务端返回一个 FIN-ACK 包。客户端收到 FIN-ACK 包后，向服务器返回一个 ACK 包，连接关闭。



TCP 的其它机制简介

- TCP 是一个十分复杂的传输层协议。受限于课时安排本节课无法将它的面貌完整地呈现。
- TCP 的字节流传输基于“滑动窗口”机制。我们知道 TCP 有流量控制的功能，这一功能就是基于滑动窗口实现。
- TCP 还具有拥塞控制的功能。
  - 网络拥塞：用户对网络资源的需求总量大于网络可用资源。
  - 网络拥塞时，数据包延迟上升，丢包率增加。而 TCP 的确认和超时重传会进一步加剧拥塞。因此，好的拥塞控制算法会对传输质量有极大的提升。

TCP 小结

- TCP 是有连接的，因此只能实现单播的传输方式。在没有建立连接的情况下向 TCP 端口进行数据传输会直接被 RST (reset) 包重置连接。
- TCP 的应用场景主要是需要可靠传输，而对延时敏感性不高的场景。
- 常见的 TCP 应用：
  - 文件下载
  - 远程登录
  - 电子邮件
  - 在线聊天

Part V. Application Layer

应用层

应用层简介

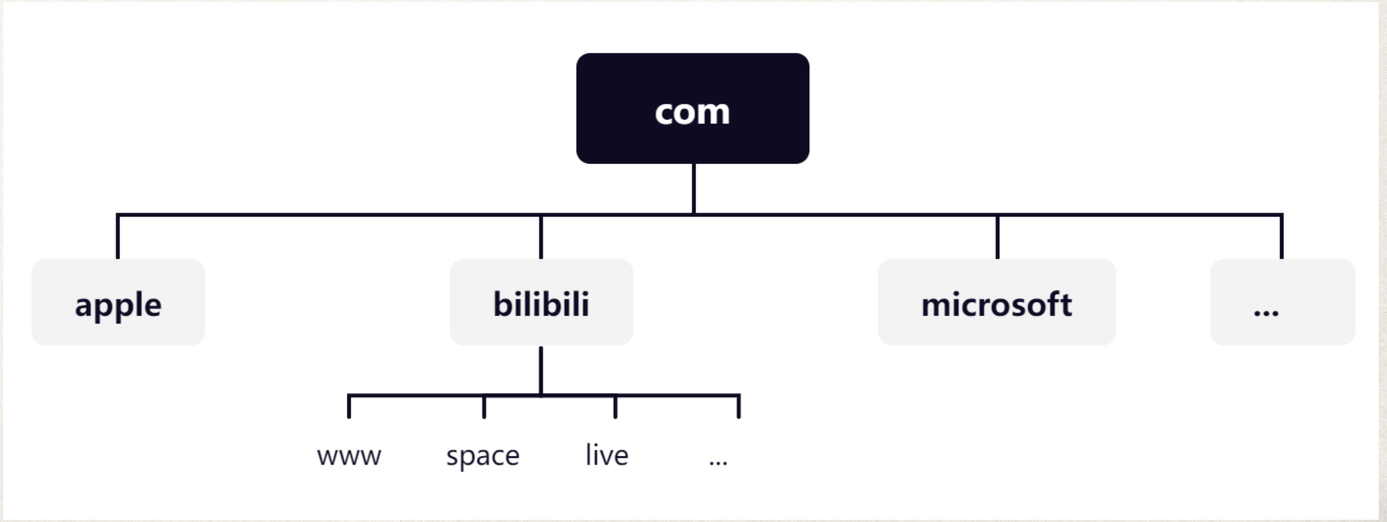
- OSI 模型只是一个理论模型，在工程实践上，会话层和表示层实质上并没有独立实现。
- 本节中我们按照 TCP/IP 模型，将这三层视为一层应用层进行讲解。

DHCP

- 动态主机配置协议 (Dynamic Host Configuration Protocol, DHCP) 是一个应用层协议，它能够动态地为本地主机分配 IP 地址。
- DHCP 也通过广播工作，基于 UDP 67 端口。
- 通过 DHCP，网络管理员可以方便地管理网络主机的 IP 地址，并为它们下发网关、DNS 等配置。
- DHCP 有许多 option 可供网络管理员配置。

域名与 DNS

- 从之前的学习中我们知道，计算机之间通过 IP 地址进行通信。
- 但 IP 地址较难以记忆，为了方便使用，人们提出了域名的概念。
- 域名是人类便于记忆的表现形式，访问时需要通过 DNS 解析将域名转换为 IP 地址以供计算机连接。
- 域名的格式是点分字符串，例如 www.bilibili.com。域名使用树形结构组织。



DNS

- DNS 是域名系统 (Domain Name System) 的缩写，同时 DNS 也是域名解析协议的名称。
- DNS 协议同时支持 TCP 和 UDP，使用 53 端口。通常为了较低的延迟，操作系统会选择使用 UDP。
- DNS 通过解析记录定义域名和 IP 的对应关系，常用的记录类型有：
  - A 记录 / AAAA 记录：定义域名到 IPv4 地址 / IPv6 地址的映射关系。
  - CNAME 记录：定义域名作为另一个域名的别名，对该域名进行解析请求得到的是另一个域名的解析结果。
  - MX 记录：定义域名的邮件服务器地址。

HTTP

- 超文本传输协议 (HyperText Transfer Protocol, HTTP) 是 Web 互联网应用最为广泛的协议。我们平时的网页浏览都离不开 HTTP。
- HTTP 经过了诸多版本的演进，从最早的 HTTP/1.0 到 HTTP/3 发生了天翻地覆的变化。本节以最为经典的 HTTP/1.1 版本进行讲解。
- 在 HTTP/3 之前，HTTP 基于 TCP，默认端口号为 80。
- HTTP/1.1 是基于文本的协议，也就是说它实际上是人类可读的。
- HTTP 分为请求和响应。同时 HTTP 是无状态的，每一次的请求都是彼此独立的。
- HTTP 通过“头”的方式传递必要的信息，HTTP 头可以认为是字符串键值对。

HTTP 的基本性质

- 无状态，有会话
  - HTTP 本身是无状态协议，会话通过 Cookie 保持
- 简单而可扩展
  - HTTP/2 之前的 HTTP 为纯文本协议
  - 通过 HTTP 头的机制，实现各种扩展功能

HTTP 请求

- HTTP 请求的格式如下：
  - <方法> <路径> <HTTP 版本>
  - <请求头>
  - [空行] (HTTP 中换行符均为 "\r\n")
  - <请求体>
- 下面我们来看一个请求 redrock.team 的请求实例：
  - GET / HTTP/1.1
  - Host: redrock.team
  - User-Agent: curl/7.55.1
  - Accept: */*

HTTP 方法

- HTTP 约定了几种“方法”以供客户端操作服务器上的资源，常用的有：
  - GET：向服务器请求获取指定的资源，通常 GET 请求不带请求体。
  - POST：向服务器请求向指定资源提交数据。
  - HEAD：要求服务器只返回响应头部而不返回内容。
  - PUT：向指定资源位置上传其最新内容。
  - DELETE：请求删除指定资源。
- 在不同的场景下，我们应当选择相应符合语义的方法进行请求。

HTTP 响应

- HTTP 响应的格式如下：
  - <HTTP 版本> <状态码> <状态信息>
  - <响应头>
  - [空行] (HTTP 中换行符均为 "\r\n")
  - <响应体>
- 下面我们来看一个请求 redrock.team 的响应实例：
  - HTTP/1.1 302 Found
  - Location: https://redrock.team/
  - Date: Thu, 25 Nov 2021 07:21:25 GMT
  - Content-Length: 5
  - Content-Type: text/plain; charset=utf-8

HTTP 状态码

- 状态码主要有以下几种分类：
  - 1xx：特殊消息。
  - 2xx：表示成功，例如 200 OK。
  - 3xx：需要进一步操作，通常是重定向，例如 302 Found。
  - 4xx：客户端错误，例如 404 Not Found。
  - 5xx：服务端错误，例如 503 Service Unavailable。

HTTP 重定向

- 301、302、303、307、308 是重定向状态码。

	缓存（永久重定向）	不缓存（临时重定向）
转GET	301	302、303
方法保持	308	307

- 使用 Location 响应头来返回目标地址。
  - 一种返回格式是完整 URL，例如 `Location: https://redrock.team/`
  - 另一种是返回路径，例如 `Location: /foo/`

HTTP 连接复用

- HTTP/1.1 加入了长连接。
  - Connection 头配置为 keep-alive 时，HTTP 进入长连接状态。服务端发送完响应后不关闭连接。客户端可以在同一个连接内发送请求。
  - 结束连接或明确表示不启用长连接，Connection 头配置为 close。
- HTTP/2 在多个连接的域名可被同一 HTTPS 证书验证时，会自动合并连接来减少连接数，但这在某些情况下可能导致问题。
  - https://www.zhihu.com/question/310263956/answer/601458852

HTTP 会话

- HTTP 通过 Cookie 保持会话。
- Cookie 是一种 Key-Value 集合，并附加了一些配置项目。接受 Cookie 的客户端每次请求时带着 Cookie 头发给服务器。
- 服务端可以通过 Set-Cookie 头来设置 Cookie。
  - 例如 `Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly`
- Cookie 有许多属性，例如上面的：
  - Expires：设置了 Cookie 的过期时间
  - Secure：只允许在 HTTPS 下传输
  - HttpOnly：不允许通过 JavaScript 脚本读取 Cookie。

HTTP 会话

- Cookie 可以通过属性限制它被发往哪里：
  - Domain 属性指定了哪些域名可以被发送 Cookie，它包括子域名。即 Domain=redrock.team 允许向 app.redrock.team 发送 Cookie。
  - Path 属性指定了哪些路径可以被发送 Cookie。
  - SameSite 限制跨站发送 Cookie。

HTTP 内容传输

- 众所周知，一个网站有大量的资源，对于不同的资源，浏览器的处理方式也应当不同。
- HTTP 头 Content-Type 则表示了资源的类型。例如：
  - HTML 网页是 text/html
  - PNG 图片是 image/png
- 这种文件类型我们称之为 MIME 类型。它的结构就是：
  - 类型/具体类型
- 具体的 MIME 类型介绍：https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Basics_of_HTTP/MIME_types

HTTP 内容传输

- Content-Type 头既可以出现在请求中又可以出现在响应中。
  - 对于请求，它表示 POST 数据时请求体的数据格式。
    - 对于表单，通常有两种格式，一种是 application/x-www-form-urlencoded，另一种是 multipart/form-data。
      - 第一种:

```
POST / HTTP/1.1
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

say=Hi&to=Mom
```

      - 第二种:

```
POST /test.html HTTP/1.1
Host: example.org
Content-Type: multipart/form-data;boundary="boundary"

--boundary
Content-Disposition: form-data; name="field1"

value1
--boundary
Content-Disposition: form-data; name="field2"; filename="example.txt"

value2
```

HTTP 内容传输

- 对于响应，它表示响应体的格式。
- 有时，在 MIME 类型后面，可能会跟着它的编码。

- 我们在下载时可以看到文件的大小，这是由 Content-Length 头控制的。Content-Length 头的值是一个数字，表示传输文件的字节数。
  - Content-Length 头也会在携带 Body 的请求体出现。
- 偶尔，我们希望改变内容的呈现方式，这时 Content-Disposition 头则处理这类问题。
  - 作为响应时，Content-Disposition: attachment 表示强制作为下载，inline 则期望浏览器尽量不用下载弹框
  - Content-Disposition 还有 filename 属性，可以指定文件名。

```
Content-Disposition: inline; filename=equation-' . $random . '.emf'
```

HTTP 内容传输

- HTTP 有一系列 Accept 的头供客户端传给服务端，以声明其想要的是怎样的数据。
  - Accept-Encoding 头控制期望的数据编码，因此它可以请求是否使用压缩。例如使用 `Accept-Encoding: gzip, deflate, br` 则告诉服务端可以使用 gzip、deflate、br 等压缩算法。
  - Accept 头说明期望的数据格式，例如 Accept: image/png 则期望服务端发送 PNG 图片。

HTTP 内容传输

- HTTP 具有断点续传的能力。
  - Range 头可以供客户端指定请求的数据范围，格式是 `Range: (单位=起始)-[终止(闭区间)]`
  - 服务器收到后，断点续传的内容会返回状态码 206 Partial Content，并携带 Content-Range 头表示当前传输的范围。`Content-Range: 单位 起始-终止(闭区间)/总长度`
  - 如果终止范围超出文件长度，则认为请求不合法，返回 416 Requested Range Not Satisfiable。
  - 如果终止范围小于起始范围，传输整个文件返回 200。
  - 例如 Range=-114 这一类请求头，表示获取文件最后的 114 个字节。

HTTP 内容传输

- HTTP 还具有条件请求的能力。为了减少不必要的数据传输，客户端可以携带对本地缓存资源的描述，若资源没有更改则无需传输。
  - 服务端返回资源时，可以在头部附带 Last-Modified 头表示最后一次修改的时间，或者 ETag 来表示文件的“实体值”，作为资源当前状态的唯一描述。
  - 客户端请求时，可以带上 If-Modified-Since 头或 If-Not-Match 头，分别对应 Last-Modified 和 ETag。
  - 如果服务端的资源发生变更，则 200 返回新的资源并更新上述头。否则，返回 304 Not Modified 告诉客户端无需更改并不传输资源。
- HTTP 也支持控制客户端缓存。
  - Cache-Control 头决定了缓存如何进行控制。
    - https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Cache-Control

HTTP Referer

- 在浏览器中，用户从链接打开网页以及页面内引入资源时，会带上 Referer 表明来源。
- 通常，可以通过 Referer 统计用户的来源，也可以通过 Referer 限制来防盗链。
- Referer 其实是一个错误拼写，正确的拼写是 Referrer。但 Referer 这个拼写错误在被应用需求后才被发现，为了兼容现有实现，Referer 被将错就错地纳入了标准。
  - 但在 Web 浏览器的 JavaScript 中，访问 Referer 采用的是 document.referrer，一定程度上避免了该问题的进一步传播。

HTTP 验证

- 访问被保护的资源时，对于尚未授权的客户端服务器返回 401 Unauthorized，同时返回 WWW-Authenticate 头表示支持的验证方式。

```
WWW-Authenticate: 类型 额外参数
```

- 客户端携带 Authorization 头，Authorization 头中包含授权信息。

```
Authorization: 类型 授权字符串
```

- 目前最常见的验证方式有两种：
  - HTTP 基本验证
    - Authorization 头携带 Basic “用户名:密码"的base64编码 格式
  - Bearer 令牌验证
    - Authorization 头携带 Bearer Token 格式

同源策略和跨域资源共享

- 随着 Web 技术的发展，网页可以通过 JavaScript 进行异步请求访问各种资源。但是如果允许发送任意请求可能会存在安全问题。
- 因此浏览器进行了”同源策略“。
  - https://developer.mozilla.org/zh-CN/docs/Web/Security/Same-origin_policy
- 但现实业务中，我们不太可能只从同源加载资源。因此为了让资源可以安全地跨域加载，跨域资源共享 (CORS) 被提出。
  - https://developer.mozilla.org/zh-CN/docs/Web/HTTP/CORS

同源策略和跨域资源共享

- CORS 预检请求：
  - 使用 OPTIONS 方法。
  - 携带 Origin。
  - 携带 Access-Control-Request-Method 和 Access-Control-Request-Headers
- CORS 预检响应：
  - 返回 204 No Content。
  - Access-Control-Allow-Origin: 允许的源
  - Access-Control-Allow-Methods: 允许的方法
  - Access-Control-Allow-Headers: 允许携带的头
  - Access-Control-Max-Age: 以上信息的客户端缓存时间

WebSocket

- HTTP 归根结底还是必须由客户端发起请求，这就导致服务器难以主动向客户端推送消息。
- 因此如果遇到类似聊天室或在线游戏等状态实时变化的场景，单纯的 HTTP 便有些难过，常用方式有：
  - 轮询：不断发起请求向服务器询问是否有新的内容，代价是带来了巨大的并发压力。
  - 长连接：发起连接后服务器不返回数据，直至有新的内容。但这种方式仍然不能做到连续推送数据。
- 因此，为了解决这个问题，WebSocket 出现了。WebSocket 是一个类似 TCP 的全双工协议，双方可以自由收发数据。

WebSocket

- WebSocket 在握手阶段使用 HTTP 协议，但完成握手后就仅仅是使用 HTTP 所建立的连接来传输数据，和 HTTP 协议再无关联。
- WebSocket 连接建立的过程：
  - 客户端请求服务器，携带头部如下：
    - Connection: upgrade　　// 表示升级协议
    - Upgrade: websocket　　// 表示升级到 WebSocket
    - Sec-WebSocket-Version: 13　　// 表示使用的 WebSocket 版本
    - Sec-WebSocket-Key: 一段随机生成的密钥

WebSocket

- 服务器返回 101 Switching Protocols，并返回以下头。
  - Connection: Upgrade
  - Upgrade: websocket
  - Sec-WebSocket-Accept: 从客户端 Sec-WebSocket-Key 计算得
- 之后便建立起了全双工通信。

HTTPS

- HTTP 是一个没有加密验证的文本协议，在 HTTP 的基础上添加一个加密层，就形成了 HTTPS。
- HTTPS 通常使用 SSL 或 TLS 进行加密和证书验证，其默认端口为 443。
- SSL/TLS 可以认为它工作在会话层和表示层，为承载于上的其他协议提供透明的加密传输服务。
- 证书置信的知识较为庞大但十分有趣，本节课不过多讲解。

其它常用的应用层协议

- 应用层协议的数量浩如烟海，我们的课程不可能一一讲解，对 DNS、HTTP 进行较为细致的讲解是因为它们十分常用。
- 常用的应用层协议还有：
  - Linux 服务器进行远程登录的 SSH 协议
  - 邮件收取使用的 IMAP 和 POP3 协议
  - 邮件发送使用的 SMTP 协议
  - 文件共享使用的 FTP 协议
  - ……

小结：打开 B 站看视频时发生了什么

- 首先，浏览器根据输入的 https://www.bilibili.com/，向 DNS 服务器发送 UDP 请求得到了 www.bilibili.com 的 IP 地址。在这个过程中，数据包经过路由选择算法选择的最优道路走出你的家门。
- 然后，浏览器和对方的 HTTP 服务器进行了 TCP 三次握手，从而在 TCP 连接中进行了 TLS 握手建立加密连接，然后在 TLS 安全隧道中进行 HTTP 通信。
- 你的视频加载完了，你看得很开心，终于看完了。
- 浏览器和服务器进行了 TCP 四次挥手，释放了 TCP 连接。
- 事实上，可能有许多包在网络中丢失或出现各种意外状况，可能传输过程中有无数个错误被纠正，但计算机网络的设计者们通过各种精妙的机制让你感受不到这些惊心动魄。

Part VI. Network Configuration on Linux

Linux 下的网络配置

Linux 网络接口

- 本地环回接口
- 以太网接口
- 点对点接口

共同点：

- 都在系统中有一个设备名
- 都有 UP 或 DOWN 两种状态，即活跃与不活跃状态
- 都有 MTU 属性和 flags 属性，在 flags 属性中定义该接口的类型与所支持的协议

本地环回接口

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
 group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

- 通常接口名为 lo
- MAC 地址为 00:00:00:00:00:00
- IPv4 通常为 127.0.0.1/8，IPv6 通常为 ::1/128
- 特别的，所有目标地址为 127.0.0.0/8 内的包均会被 lo 接口捕获
- 适用于本机程序之间的网络访问，流量不会经过任何网卡

以太网接口

```
3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ether 00:15:5d:00:32:00
    inet 100.98.22.33/24 brd 100.98.22.255 scope global dynamic
       valid_lft forever preferred_lft forever
    inet6 fe80::1507:455b:bcf3:796c/64 scope link dynamic
       valid_lft forever preferred_lft forever
```

- 接口名没有特定的规则
- 数据链路层确保使用 MAC 地址
- 支持 IPv4/IPv6 网络层协议，通常有一个 fe80:: 开头的 IPv6 链路本地地址
- 以太网接口的实例：物理网卡、TAP 设备、Bridge、VLAN 设备、Bond、VETH 等

点对点接口

```
5: sit1@NONE: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1480
qdisc noqueue state UNKNOWN group default qlen 1000
    link/sit 172.16.6.231 peer 166.111.21.1
    inet6 2402:f000:1:1501:200:5efe:3b6e:f2ab/64 scope
global
       valid_lft forever preferred_lft forever
    inet6 fe80::ac10:6e7/64 scope link
       valid_lft forever preferred_lft forever
```

- 接口名没有特定的规则
- 数据链路层实现不定
- 网络层协议支持不定
- 点对点接口的实例：许多 VPN (如 PPTP、L2TP、WireGuard)、TUN 设备以及一些隧道 (如 IPIP、ISATAP/SIT)

iproute2 的使用

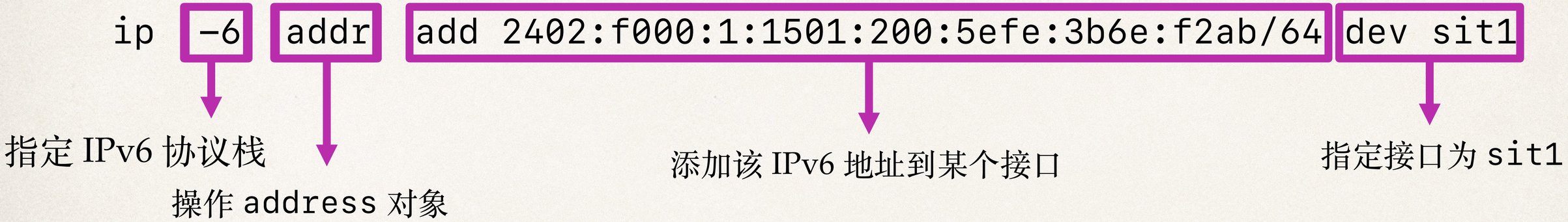
- 现代 Linux 使用 iproute2 取代了在 Unix-like 操作系统中广泛使用的 net-tools 系列工具。
- 我们主要使用 iproute2 包中的 ip 命令来操作网络。

iproute2 的使用

- `ip` 命令非常强大，常用操作对象有：
  - `address`：对网络接口上的 IP 地址进行操作
  - `route`：对系统路由表进行操作
  - `link`：对网卡接口链路进行操作
  - `rule`：对系统处理数据包的策略进行操作
- `ip` 命令可以通过 `-4` 和 `-6` 参数指定使用的是 IPv4 还是 IPv6 协议栈。
- `ip` 命令支持操作对象的前缀缩写，出现歧义时字典序靠前的对象被执行：
  - 例：`ip r`。由于 route 比 rule 字典序靠前，因此等价于执行 `ip route`。
- `ip` 命令的操作对象不带任何命令执行时即表示为输出。

iproute2 的使用

例子：



可以看出，一条 ip 命令的构成主要有 ip 命令本身的参数、操作对象和应用于操作对象的数条子命令构成。而这些子命令可能需要参数，如 add 和 dev 命令就需要传入参数。

其中，有些子命令类似“谓语”，表示动作，如 add。有些则类似“定语”，起限定、修饰的作用，如 dev。

net-tools 概述

- 在早期 Linux 发行版和其它 Unix-like 操作系统如 FreeBSD、macOS 中被广泛使用。
- 常用组件有 ifconfig、route 等。
- 尽管已经不再推荐在现代 Linux 中使用，但 net-tools 中有些组件因为某些优点或历史原因在现在仍被使用。例如 ifconfig 输出接口的信息更加详尽。
- 本节课不过多讲解 net-tools 的使用。

Netplan 的使用

- 在服务器版的 Ubuntu 20.04 中，netplan 是默认的配置工具。
- 它的配置文件编写使用 YAML 格式。对人类较为友好。
- 实际上 netplan 只是一个 renderer，它的后端可以是 system-networkd 或者 NetworkManager。

Netplan 的使用

- 将 ens18 接口配置为：
  - IPv4 地址 172.16.9.240/24
  - IPv4 默认网关 172.16.9.250
  - DNS 为 114.114.114.114 和 1.2.4.8
- 将 ens19 接口配置为：
  - IPv4 地址 30.0.0.103/24

实例：

```
victor@Victor-VM:~ » cat /etc/netplan/50-cloud-init.yaml
network:
    ethernets:
        ens18:
            addresses:
            - 172.16.9.240/24
            gateway4: 172.16.9.250
            nameservers:
                addresses:
                - 114.114.114.114
                - 1.2.4.8
        ens19:
            addresses:
            - 30.0.0.103/24

    version: 2
```

Netplan 的使用

- 使用 netplan 命令来管理配置，这里讲两个常用的子命令：
  - apply：直接执行配置。
  - try：先尝试配置，一定时间内没有收到反馈就将之前的配置恢复回去。
    有效避免远程调网调炸了直接失联的情况（
- 完整的使用方法可以 netplan help 或者 man netplan 查看。

Ping

- Ping 是最常用的网络调试工具，它通过发送 ICMP Echo 消息探测目标主机是否可以连通。
- Ping 在各种系统中都有实现。其基本语法为 ping destination。
- Windows 中的 ping 默认发送 4 个包，Linux 中的 ping 会持续发包直到手动中断。使用 -c 参数可以指定发包数量。
- Ping 的结果并不能保证有意义。Ping 通不意味着目标存活，Ping 不通也不意味着目标不存活。
  - 例如：开启 Windows 防火墙的主机不响应 ICMP 消息，因此表现为 ping 不通。某些特殊的网络环境中 ICMP 消息可能被伪造，因此即使表现为 ping 通也不意味着存活。

Traceroute

- Traceroute 是进行路由跟踪的工具，它能够探测出本机和对方主机之间的路由链路上经过的路由节点。
- 路由追踪并不能保证一定能够追踪到，某些路由器不响应 ICMP 消息或不遵守规范进行 TTL -1，则无法探测到对应节点。

扩展阅读

- https://zhuanlan.zhihu.com/p/34740683
- https://juejin.cn/post/6844903490595061767
- https://zhuanlan.zhihu.com/p/87869088
- https://zhuanlan.zhihu.com/p/61987654
- https://www.ruanyifeng.com/blog/2016/07/yaml.html
- https://www.debugger.wiki/article/html/1608694560288382

"All problems in computer science can be solved by another level of indirection."

– Butler Lampson