计网

2026-04-01

计网常见知识

常见含义

ISP：互联网**服务提供商**，如中国电信、中国联通等

MAC： **网络接口卡（网卡）** 在制造时固化的一个**全球唯一标识符**

RTT：**最小往返时延**（考虑网络拥堵）

MSS：TCP数据部分的**最大长度**

TCP/IP模型

OSI和TCP/IP

image-20251013214454674

如上图所示：

- 我们使用的 `HTTPS` 、 `FTP` 、 `DHCP` 、以及 `HTTP` 都属于**应用层**
- `TCP` 、 `UDP` 都属于**传输层**
- `IP` 则属于**网络层**

一个应用场景，完美契合TCP/IP模型

image-20251016115152094

Socket

套接字，就是ip地址+端口号

TCP

核心：三握四挥

TCP首部会用掉20个字节

image-20251013121751180

TCP报文里有SYN、ACK和FIN标识

- 设置为1就是开启这些标识
- 设置为0就是关闭这些标识

三次握手

流程

- 客户端发送SYN报文，并设置好序号
- 服务端发送SYN+ACK报文，设置序号，将确认号的值设置为客户端SYN报文的序号+1
- 客户端发送ACK报文，序号用服务端报文的确认号，确认号用服务端报文的序号+1

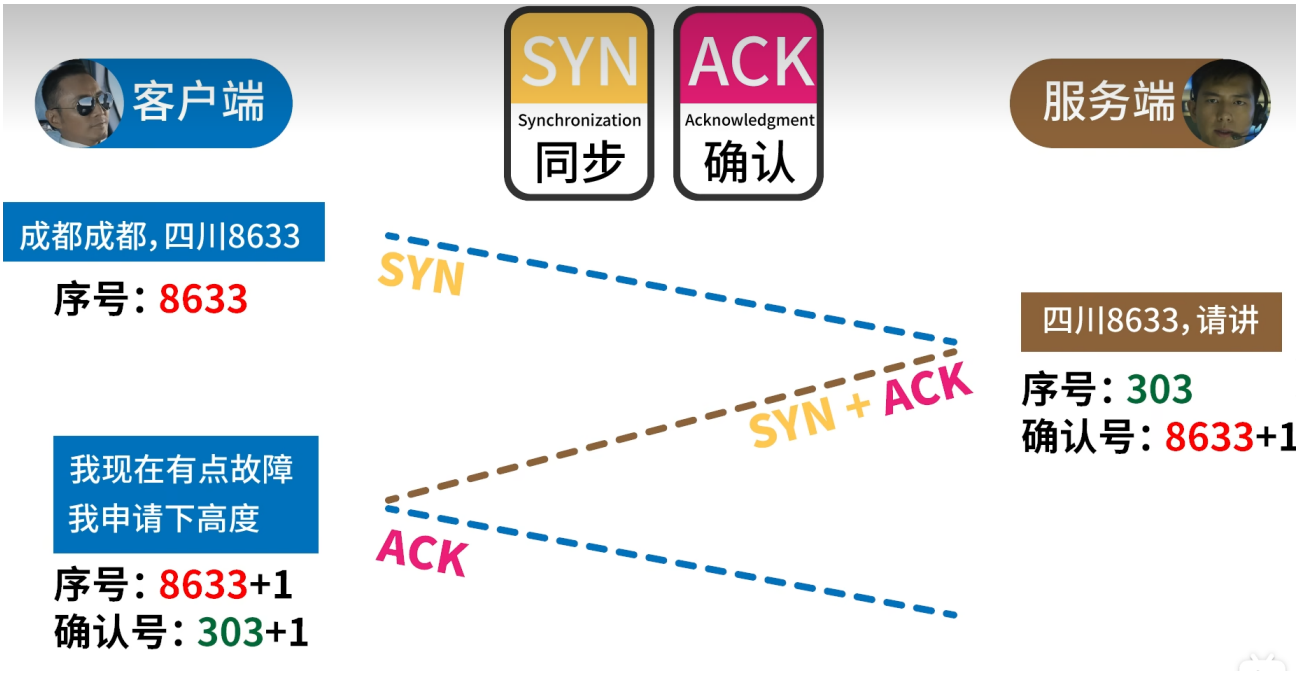

image-20251013130002349

四次挥手

流程

- 主动断开方（可以是客户端，也可以是服务端）发送一个FIN和ACK报文，并设置好**序号和确认号**
- 被动断开方发送一个ACK报文，报文的序号为断开请求的确认号，**报文的确认号为断开请求的序号+1**
- 被动断开方还可以进行数据的发送，剩余数据发送完后，被动断开方会向主动断开方发送一个FIN+ACK结束响应报文
- 主动断开方在收到FIN+ACK断开响应报文后，还需要进行最后的确认，向被动断开方发送一个ACK确认报文，**序号为被动断开方的确认号，确认号为被动断开方的序号+1**

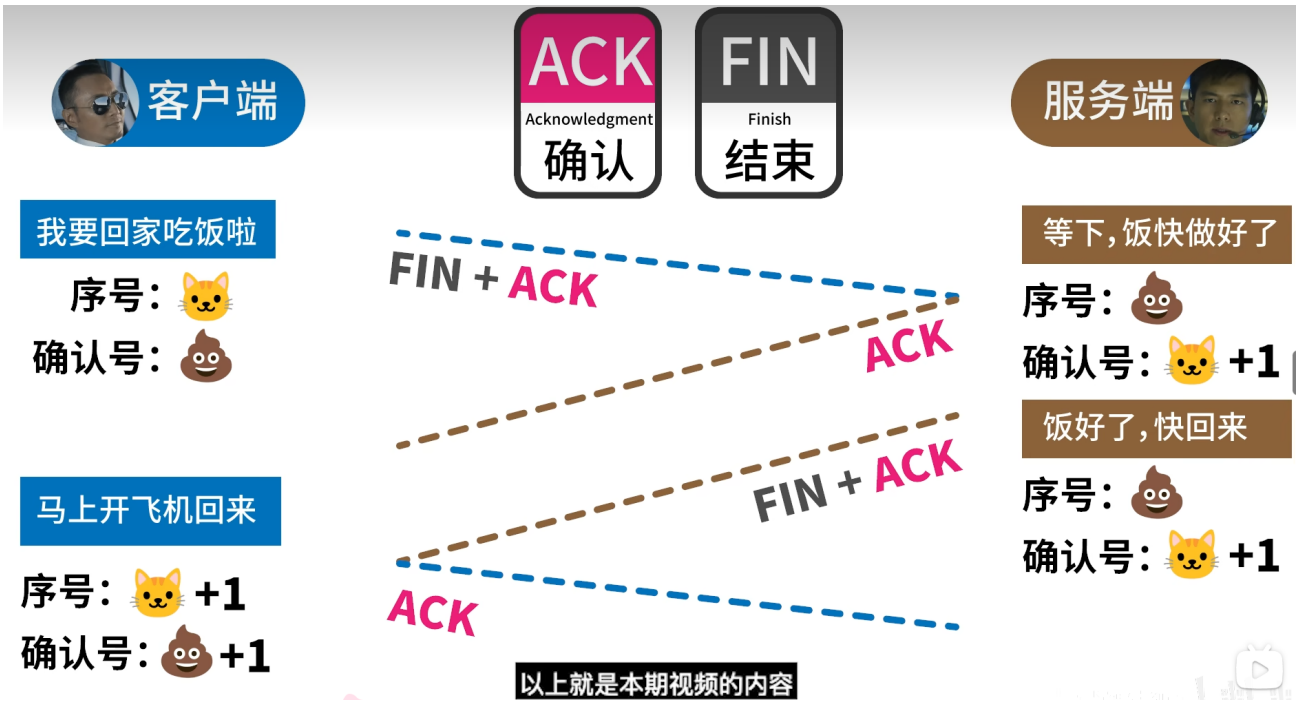

image-20251013124408713

为什么要进行四次挥手？

因为服务端可能还有数据需要发送

TCP流量控制

目的：主要是为了解决**发送方发送数据过快**导致接收方缓冲区溢出的问题

核心思想：接收方通过告知发送方自己还有多少**剩余的缓冲区空间**来主动控制**发送方的发送速率**

核心机制：滑动窗口

工作流程

1. **连接建立时**
   - 双方都会维护一个接收缓冲区
   - TCP首部中的 `Window` 字段表示当前可接收的窗口大小（ `rwnd` ），最大为65535字节
2. **数据传输中**
   - 发送方发送数据后会等待ACK
   - 接收方每次ACK时，会带上**当前的接收窗口大小**
   - 发送方根据ACK中的 `rwnd` 调整自己的发送速率
3. **窗口滑动**
   - 当接收方应用程序从接收缓存中取走部分数据时，空出来的空间就意味着窗口可以“滑动”
   - 接收方在下一次ACK中通知发送方新的窗口大小

窗口结构

- 发送方和接收方都有**自己的窗口结构**
- 发送方需要将数据分为**发送+已确认，发送窗口，不能发送**
  - 发送窗口还细分为**可用窗口**和**已发送+未确认**
    - **已发送+未确认**可用于重传
    - **可用窗口**则是表示可以发送的数据
- 接收方需要将数据分为**接收+已确认，接收窗口，不能接收**

image-20251016123303300

拥塞控制

网络拥塞：网络中的链路或者路由器**过载**时，会**丢弃数据包**

核心：采用一系列方法控制**拥塞窗口**的大小，进而控制**发送窗口**的大小，`发送窗口=min(拥塞窗口,接收窗口)`

流程

- 初始时设置`ssthresh`（慢启动阈值），采用**慢启动**，初始化**拥塞窗口**`cwnd`为1MSS（TCP数据部分的**最大长度**）
- 初始拥塞窗口为1，呈现**指数增长**
- 到达`ssthresh`后呈**线性增长**，进入**拥塞避免**阶段
- 当发生**超时重传**时
  - 说明网络**拥塞严重**，重新回到**慢启动**
  - `ssthresh = cwnd / 2`
  - `cwnd`被重置为1MSS
- 当收到3个重复的ACK时
  - 说明网络拥塞**不那么严重**，触发**快速重传**和**快速恢复**
  - `ssthresh = cwnd / 2`
  - `cwnd`被重置为`ssthresh`+3MSS
  - 然后进入**拥塞避免**阶段（线性增长）

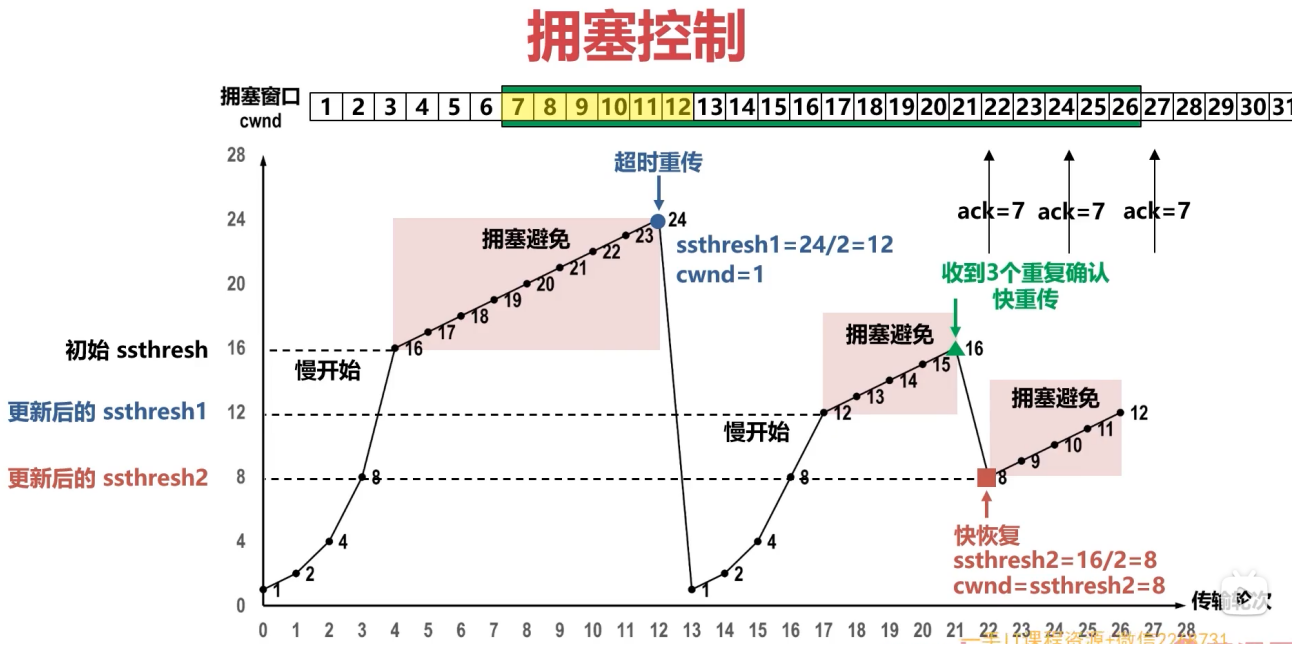

image-20251016000006624

BBR

是Google提出的**新一代TC 拥塞控制算法**

- 不靠丢包来判断网络拥堵情况
- 靠实时估算网络能提供的**最大带宽（BtlBw）**和**最小往返时延（RTprop）**来控制发送速率
- 发送速率 $\approx BtlBw \times RTprop$

UDP

无连接、不可靠的传输协议，常用在**端口寻址、实施在线游戏、实时音视频传输等**

特点

- **无连接**：通信前不需要建立连接，直接发送数据包即可
- **不可靠交付**：不提供确认、重传等机制
- **无拥塞控制**：不管网络状况多差，UDP都会以恒定的速率发送数据
- **支持广播**

UDP首部只用掉8个字节

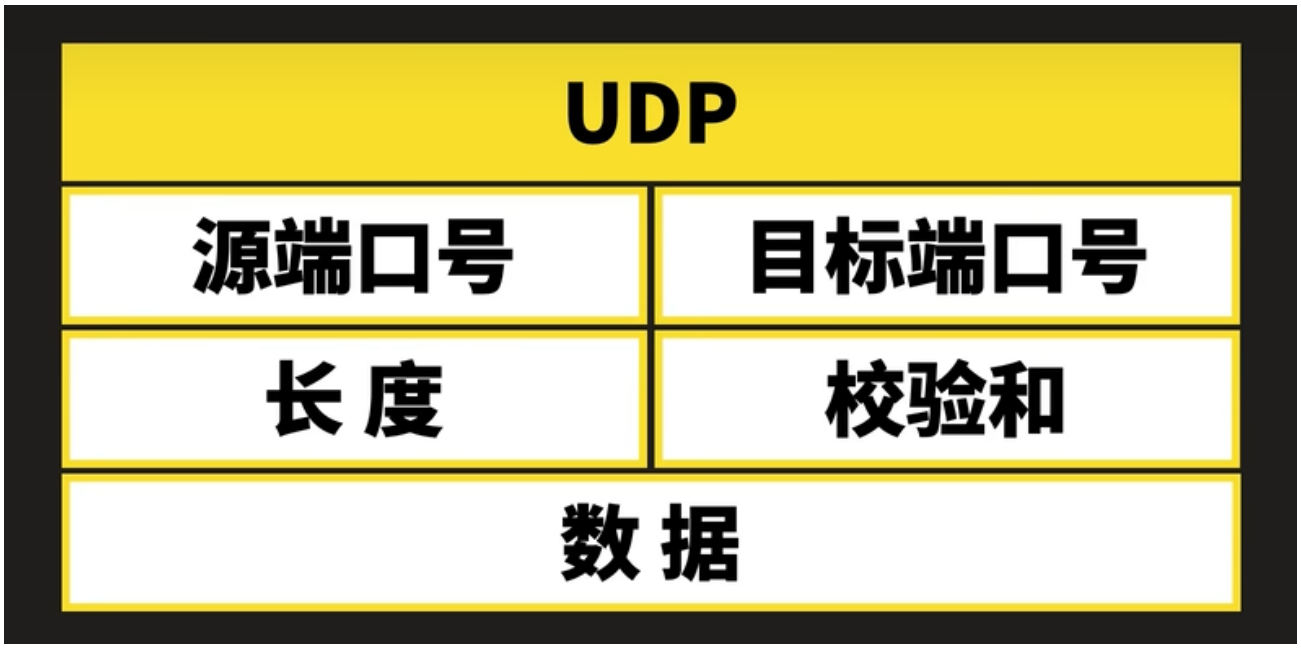

image-20251013121843576

IPV4和子网掩码

IPV4

`ipv4`是由**4组8位二进制组成的**，组之间用`.`隔开

`ip`地址 = 网络号 + 主机号

- **网络号**：同一个物理网络的所有设备，**网络号是相同的**
  - **路由寻址**：路由器只关心**目标IP地址的网络号**，从而实现数据包的转发
- **主机号**：IP地址中在特定网络内用于**标识唯一设备**的一部分
  - **最终交付**：数据包**到达目标网络**后，路由器会查看**目标IP的主机号**，从而将数据包准确地发送给正确的设备

ip地址类型

- A类：网络数为**128**，主机数为**16777216**
- B类：网络数为**16384**，主机数为**65536**
- C类：网络数为**2097152**，主机数为**256**

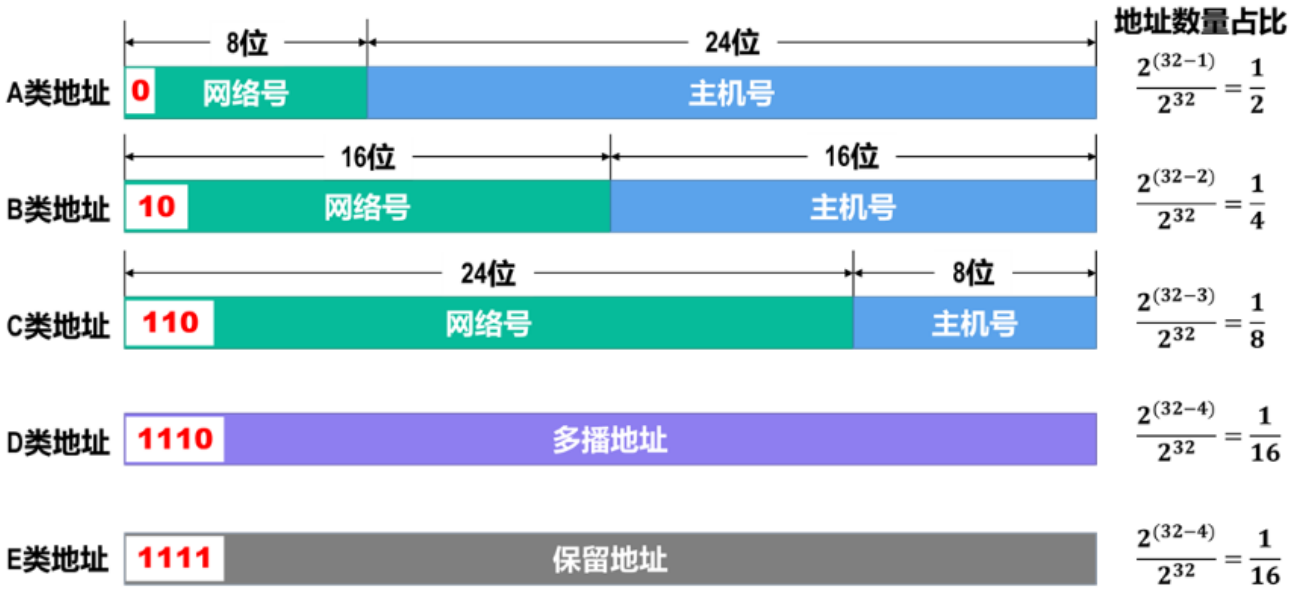

image-20251013112742698

注意

- **网络地址**的**主机位**全部为0，会占用1个ip，是整个网络的唯一标识
- **广播地址**的**主机位**全部位1，会占用1个ip，向网络中所有设备发送广播
- 因此，C类网络只能分配 `256-2=254` 个ip地址

子网掩码

一个32位二进制数，为了划分网路号和主机号而产生的

- **相同的网络号**会用**子网掩码的1**进行锁定
- 主机号**为0的二进制位**也会用**子网掩码的1**来进行锁定

也就是说，在子网掩码中

- **1对应的位是网络位**：标识一个子网。同一个子网内的所有IP地址，其网络位必须完全相同
- **0对应的位是主机位**：标识子网内的具体设备。主机位在子网内可变，且必须唯一

image-20251013113529447

CIDR表示方法

看子网掩码中有多少个1，在IP地址后加 `/1的个数` 即可

IPV6

`ipv6` 地址是由**128位二进制数**组成，通常以**十六进制**形式表示，分为**8组**，每组**16位二进制数（4个十六进制数字）**用**冒号**分隔

地址压缩

- **零压缩**：连续的零组可以用双冒号（ `::` ）表示，但在一个地址中只能使用一次
- **前导零压缩**：每组中的前导零可以省略，例如0001可以表示为1

地址的组成部分

前缀：前缀用于标识网络部分，类似于IPv4中的网络地址，前缀长度通常以斜杠后跟数字的形式表示

接口标识符：用于标识网络中的具体接口，通常为后64位

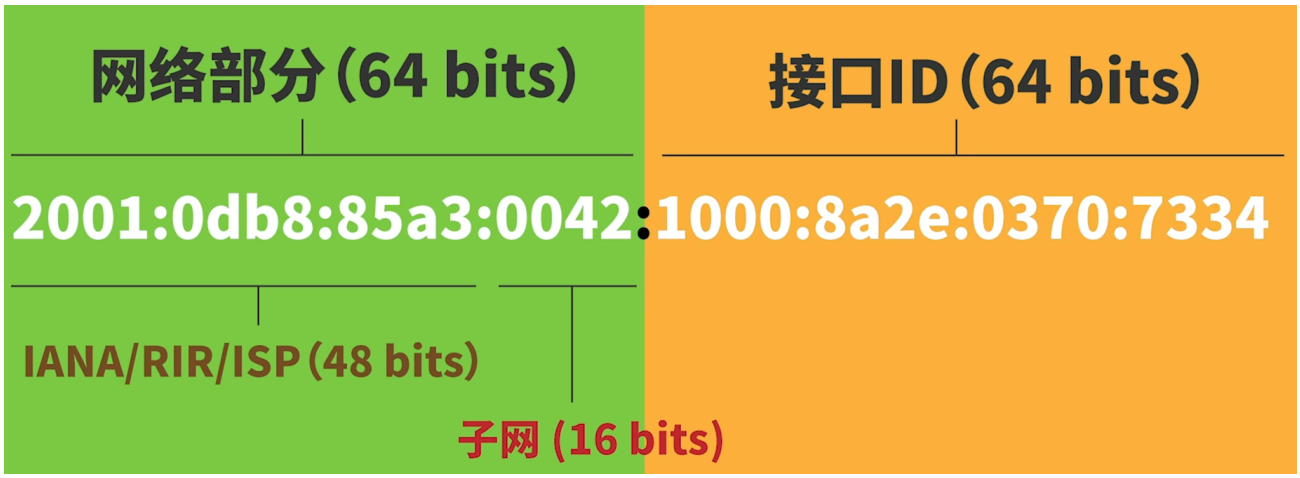

image-20251013121120752

地址类型

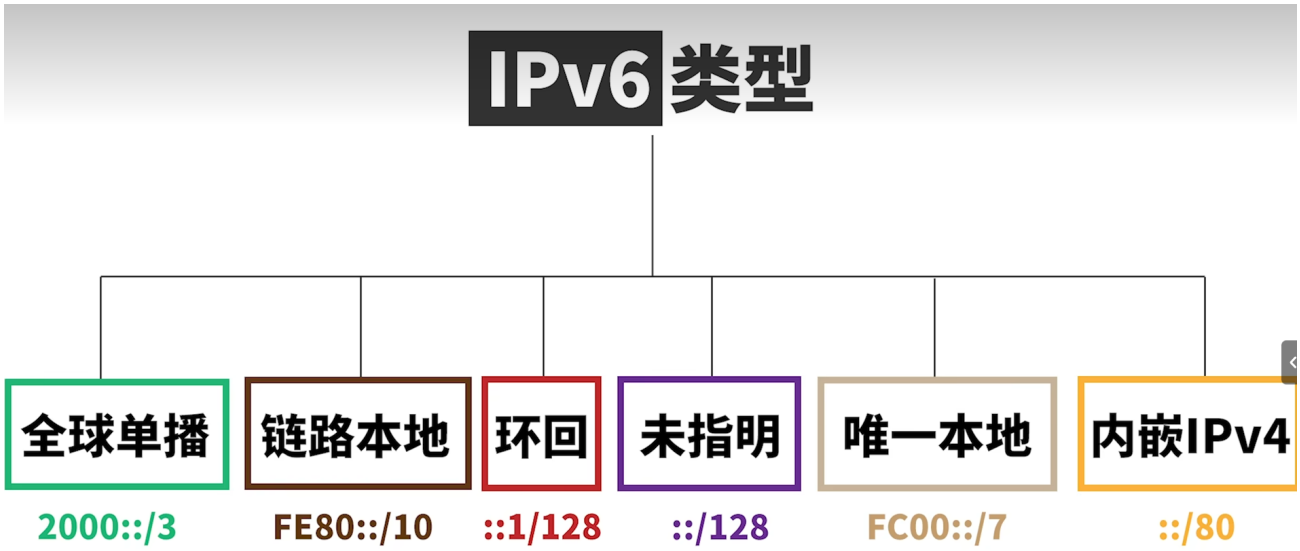

image-20251013121537369

NAT

网络地址转换

image-20251013222829174

原理：

- 内网访问外网通过出口路由时，源地址会转换成特定公有地址，并且将两个ip映射关系加到NAT映射表上
- 在外网向内网通信时，目的地址还是特定公有地址，但是到达出口路由器后，查看NAT映射表，从而转换为私有地址

问题：

- **破坏端到端通信**：两个都在NAT后的设备难以直接建立P2P连接
- **服务暴露困难**：外部网络无法直接主动访问NAT后的内部服务

虚拟机网络

NAT

图解

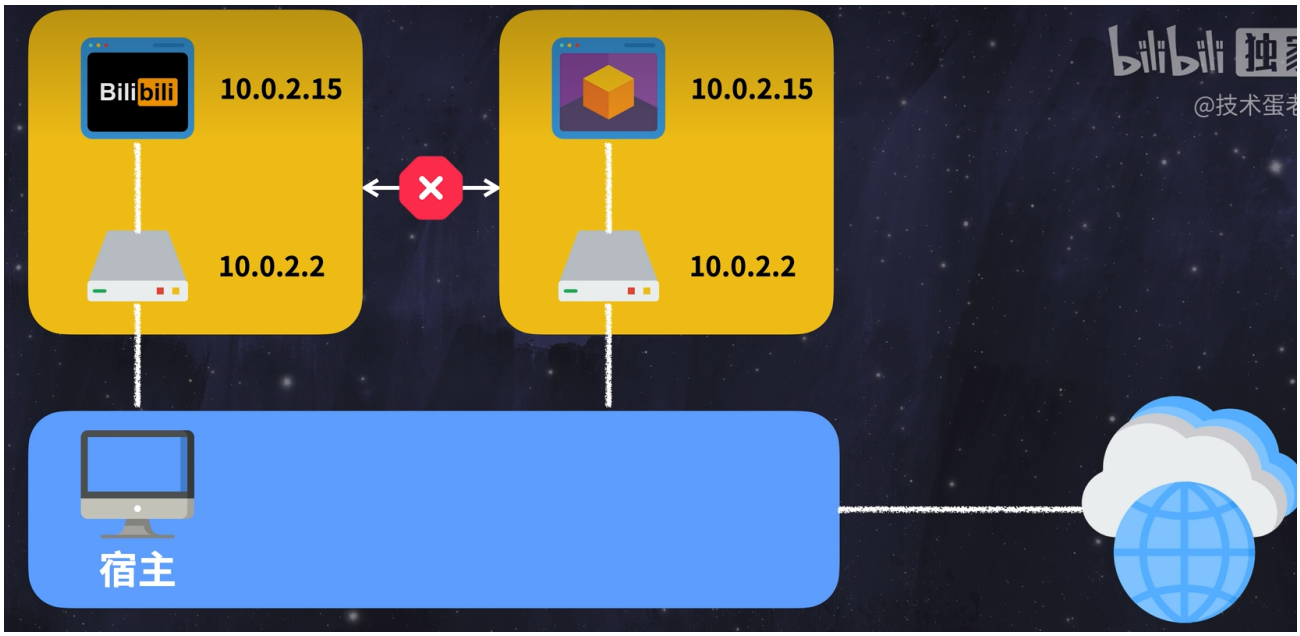

image-20251013221831854

- 虚拟机之间无法互相访问
- 宿主机、局域网设备无法访问虚拟机
- 虚拟机可以通过宿主机访问互联网

NAT网络

图解

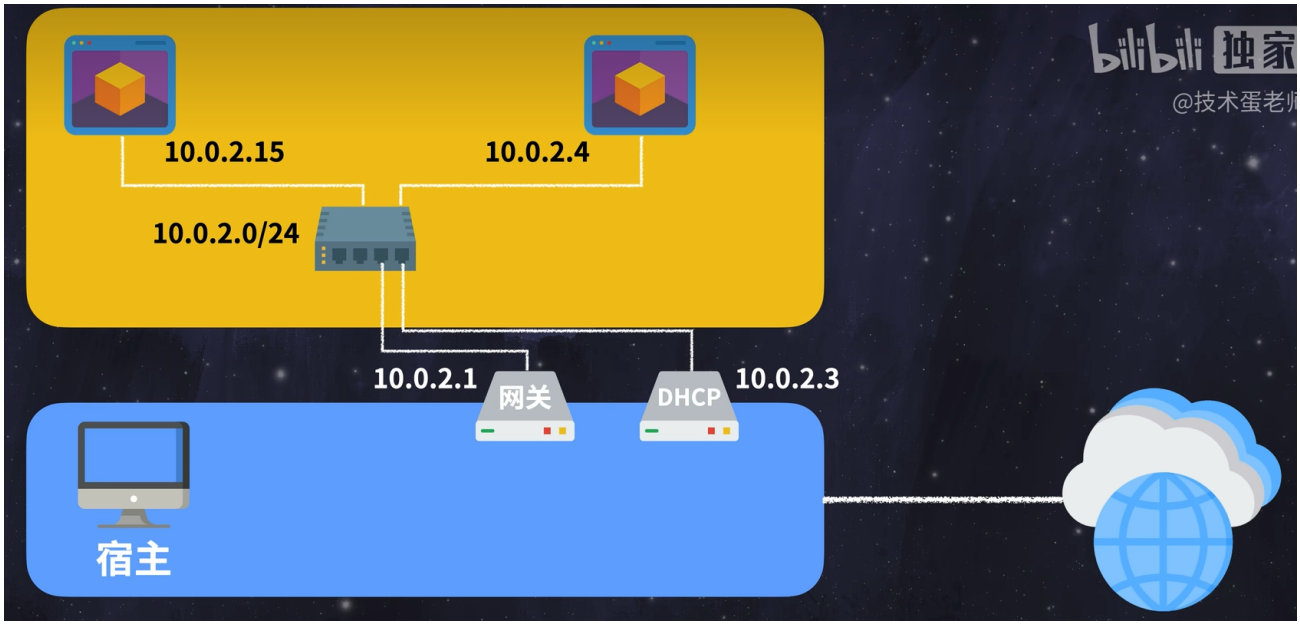

image-20251013222040076

- 在虚拟机前加了一台虚拟的交换机
- 加上了网关和DHCP服务

桥接

图解

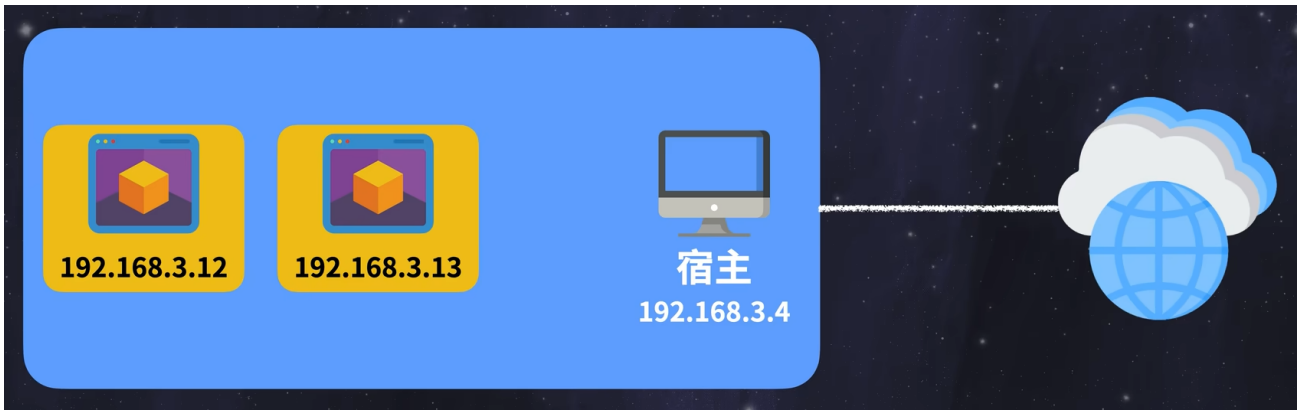

image-20251013222333237

- 虚拟机和宿主机同级，在同一个网络里
- 宿主机和虚拟机在**同一个DHCP服务**获取私有IP地址，因此虚拟机会**消耗**宿主机所在局域网的IP地址

DHCP

动态主机配置协议，是处于**应用层**的协议

作用：自动为网络中的电脑、手机等设备分配IP地址

动态配置

可以在路由器中配置IP池，增加私有IP的数量，从而增加联网设备的数量

DHCP握手

流程

- 客户端发送 `DHCP Discover`
  - **传输层**使用UDP进行数据传输，客户端使用68端口，服务端使用67端口
  - **网络层**中，不知道源IP地址填写 `0.0.0.0` ，不知道目标IP地址填写 `255.255.255.255` ，这样交换机接受到之后就会进行**广播**
  - **数据链路层**与IP地址道理相同

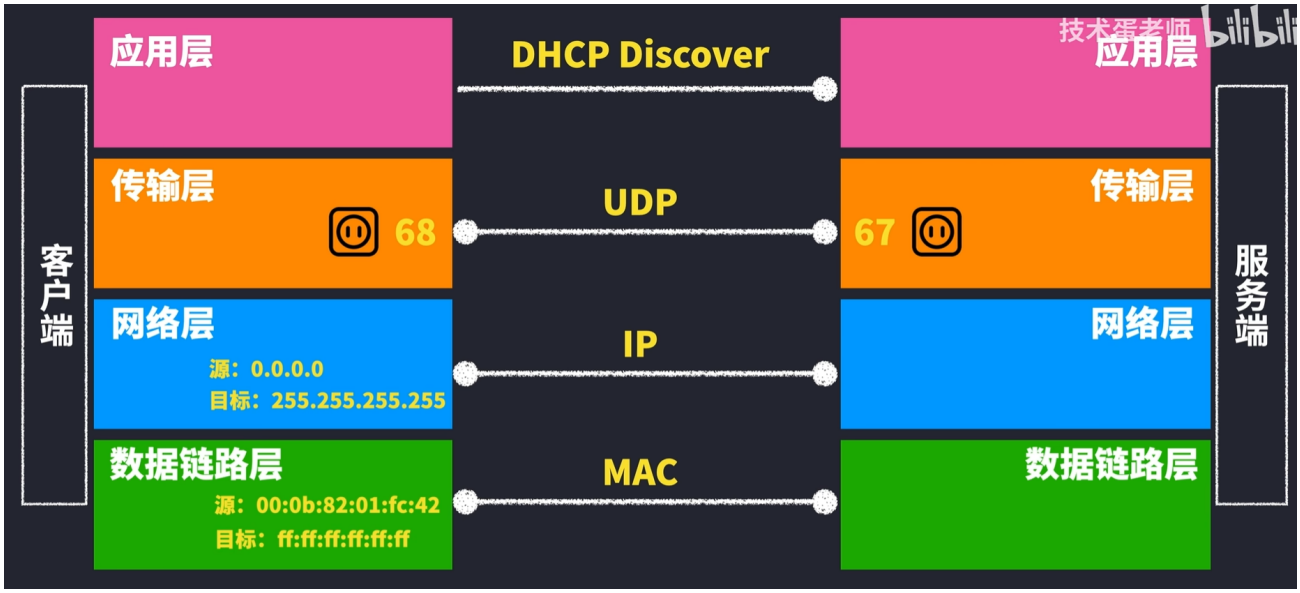

image-20251014122953171

- DHCP服务器发送`DHCP Offer`，给客户端提供私有IP地址、子网掩码、网关、DNS
  - **网络层**中DHCP服务器会发送确定的源IP地址和目标IP地址供客户端选择
  - **数据链路层**中的MAC地址也同理

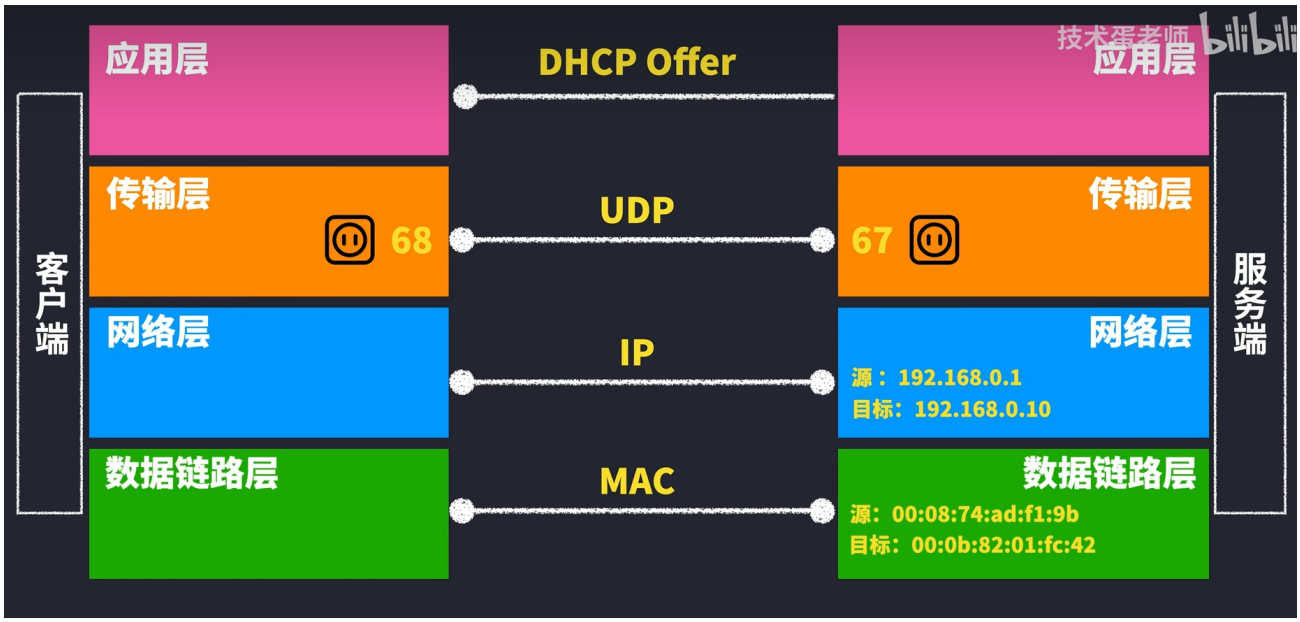

image-20251014123604469

- 客户端进行`DHCP Request`，广播所有DHCP服务器客户端选择了哪个IP

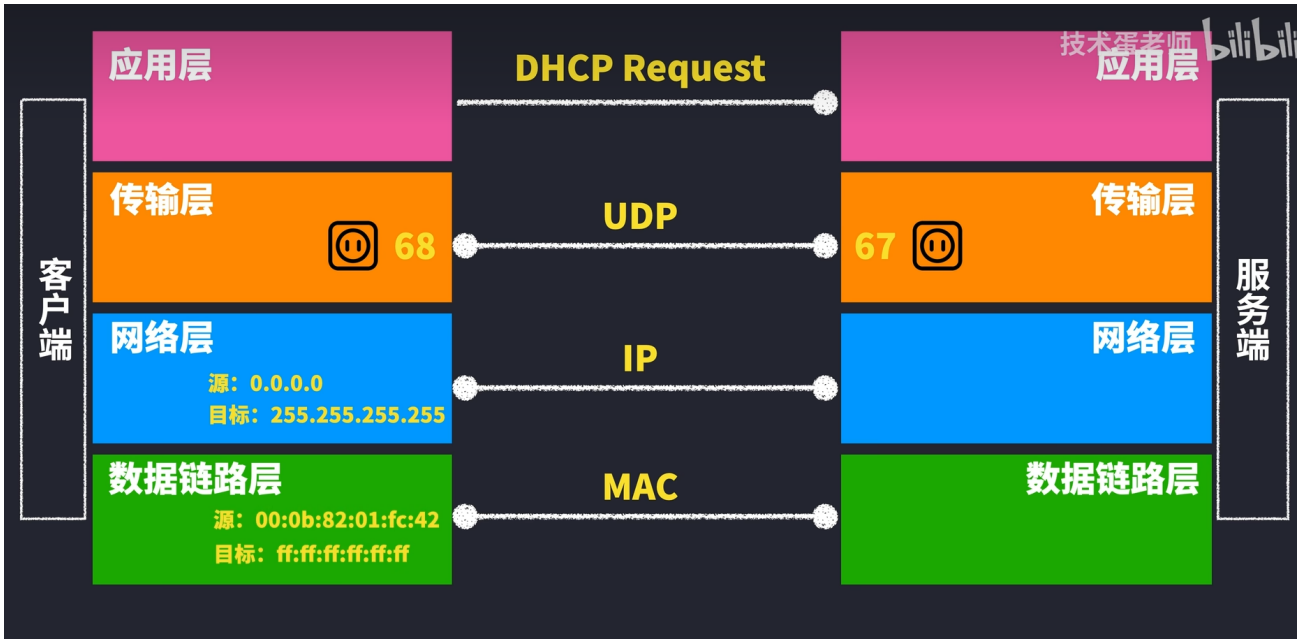

image-20251014123732540

- 服务端发送`DHCP ACK`，新设备可以开始上网

image-20251014123747948

DNS

DNS默认使用UDP协议

- 不出现**分片**情况下，UDP协议最大有效载荷是**512字节**以内
- 根服务器地址需要塞进一个**UDP包里**，最多只能放下**13组**记录

域名结构树

- 顶层的根 `.` 是由一群服务器组成的，**这群服务器只用了13个域名**

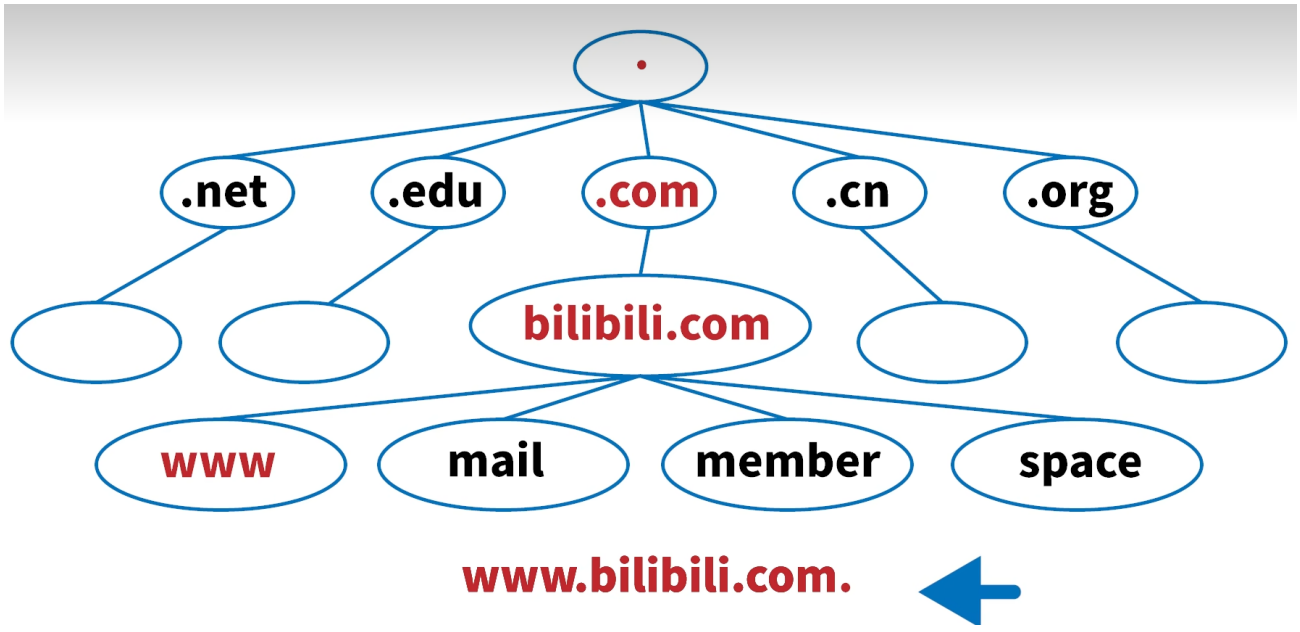

image-20251013200644701

域名服务器类型

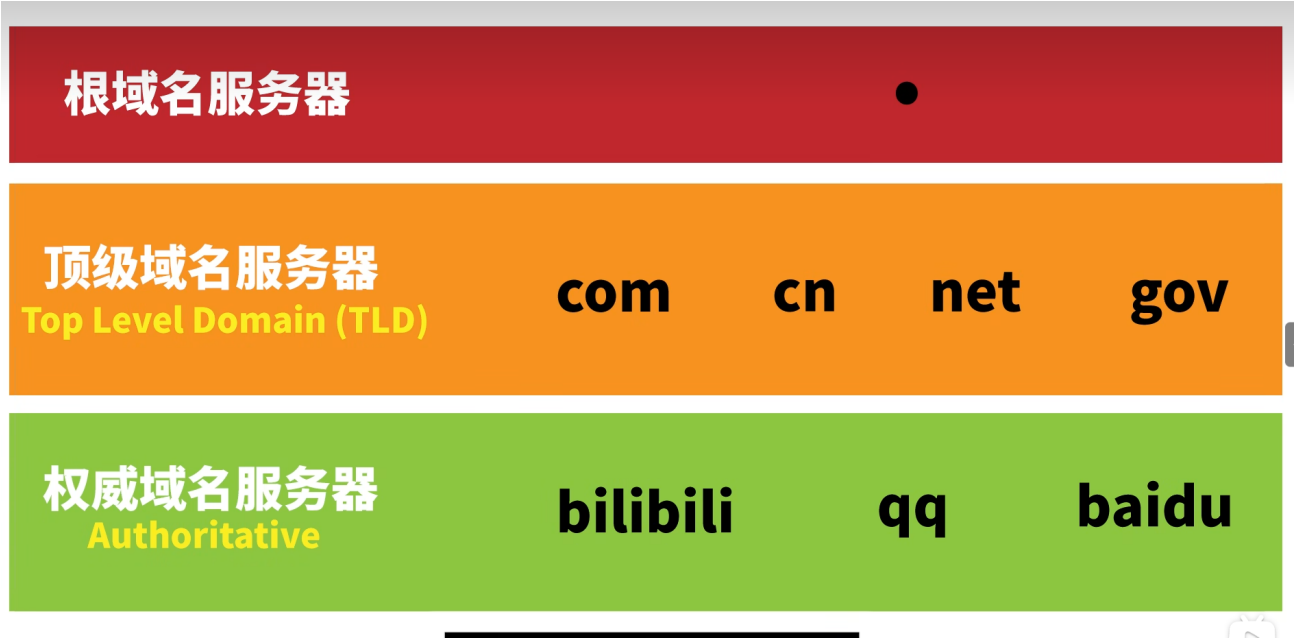

image-20251013201120277

DNS解析过程

1. **浏览器缓存**：首先在浏览器检查是否有该域名对应的IP
2. **操作系统缓存**：如果没有，浏览器会调用操作系统（如通过 `gethostbyname` 系统调用），检查本地的Hosts文件和操作系统DNS缓存
3. **本地DNS解析器**：如果本地没有，请求会发送到配置的**本地DNS服务器**
4. **根域名服务器**：若本地DNS解析器没有缓存，会向**根域名服务器**发起查询，**根域名服务器**只会返回负责你输入的域名的TLD（如 `.com` 、 `.cn` ）的**顶级域服务器地址**
5. **顶级域名服务器（TLD）**：本地DNS服务器再向TLD服务器查询，得到**权威域名服务器的**地址
6. **权威域名服务器**：最后，本地DNS服务器向权威域名服务器查询你输入的域名的IP
7. **返回并缓存**：本地DNS服务器将IP地址返回给操作系统，并缓存该记录。操作系统再返回给浏览器，并缓存

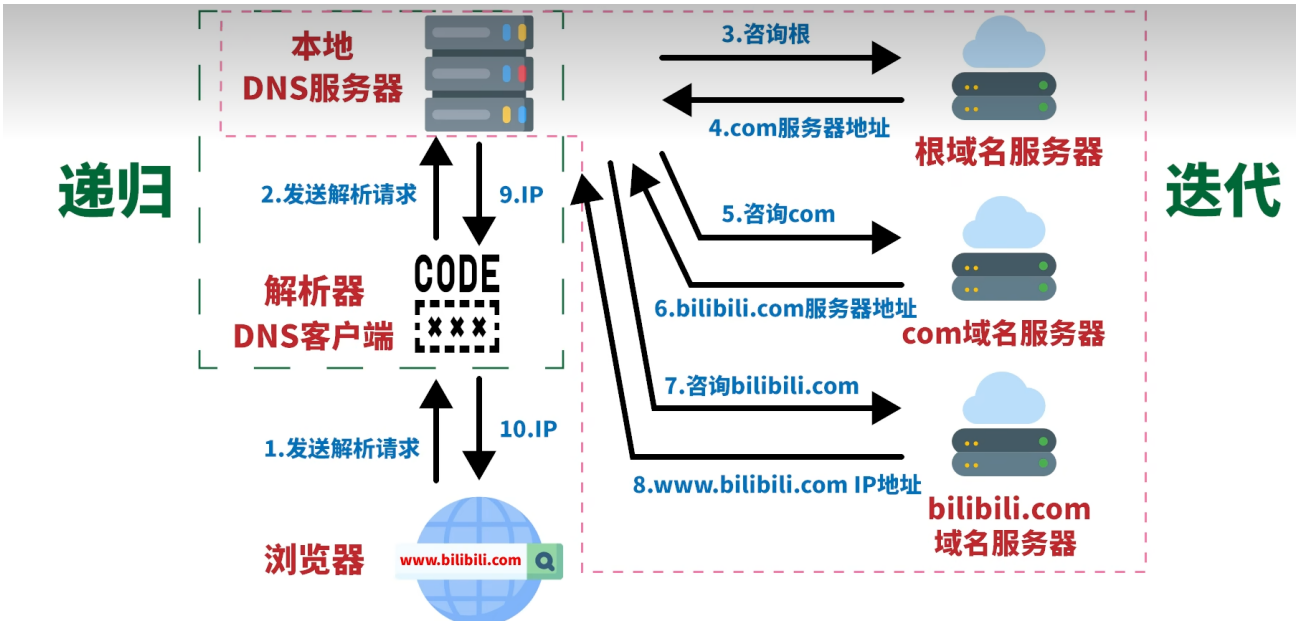

image-20251013201551042

常见DNS记录类型

- `A` **记录**：将域名指向一个**IPv4**地址
- `AAAA` **记录**：将域名指向一个**IPv6**地址
- `CNAME` **记录**： 域名别名，将一个域名指向另一个域名
- `MX` **记录**： 邮件交换记录，指定负责接收邮件的服务器

SSH

一种**加密的通信方式**，在SSH握手过程**使用非对称加密获得对称密钥**

连接流程

- 进行TCP连接
- 进行SSH握手
- 客户端和服务端协商**SSH协议版本**
- 进行**密钥交换初始化**，协商应该使用什么**算法**
- 客户端生成**临时私钥**和**临时公钥**，将**临时公钥**发送给服务端
- 服务端生成**临时私钥**和**临时公钥**，将**客户端的临时公钥**和自己的**临时私钥**和**临时公钥**生成**共享安全密钥**
- 服务端将自己的**临时公钥**发送给客户端，客户端将**服务端的临时公钥**和自己的**临时私钥**和**临时公钥**生成**共享安全密钥**
- 服务端生成一对 `host` 公钥和 `host` 私钥，生成**交换哈希值**，并使用 `host` 私钥对交换哈希值进行加密，生成交换哈希值的**数字签名**，将**数字签名和** `host` **公钥**发送给客户端
- 客户端拿到服务端的 `host` 公钥对**数字签名**进行解密，并生成自己的**交换哈希值（客户端和服务端算出来的交换哈希值是一样的）**，比较二者是否一样

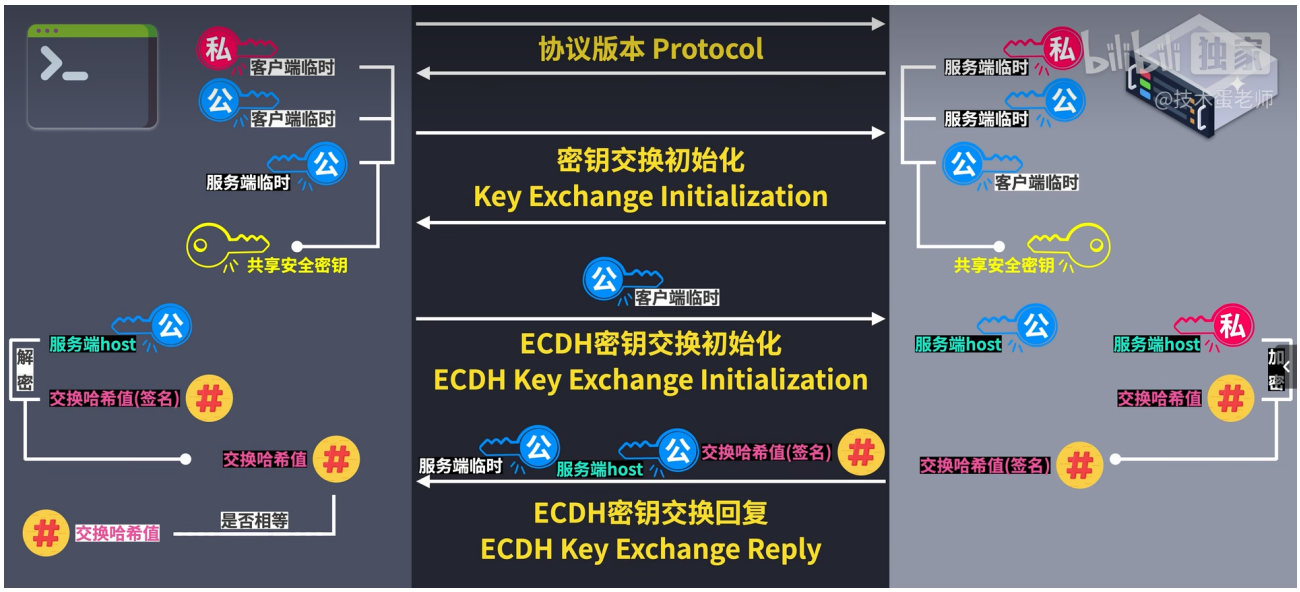

image-20251013220614655

交换哈希值

构成：

- 客户端和服务端的**版本号字符串**
- 客户端和服务端密钥交换初始化负载（**算法名称的字符串**）
- 服务端的 `host` 公钥
- 客户端临时公钥和服务端临时公钥
- 共享安全密钥

使用SSH证书

在客户端使用`ssh-keygen`命令生成密钥对，**私钥放在本地，使用`ssh-copy-id`将公钥发送给服务器**，可以实现免密登录，**提高登录安全性**

HTTP

超文本传输协议

- **无状态**（Stateless）
- **明文传输**（HTTP1.1以前）
- **可扩展**（Header可以自定义）
- 灵活（支持文本、图片、视频等多种资源类型）
- 请求-响应模型
- 默认端口是**80**

核心概括

- **HTTP/1.1**：持久连接、明文文本、队头阻塞
- **HTTP/2**：二进制分帧、多路复用、头部压缩
- **HTTP/3**：基于QUIC/UDP、解决TCP队头阻塞、集成TLS

HTTP/1.1

默认是**持久连接（Keep-Alive）**，且是**明文发送**

核心：发送一次HTTP请求，得到响应后才能进行下一次HTTP请求

- **报文格式**：使用**纯文本协议**，明文传输，（请求/响应是ASCII文本，头部和Body分界），**可读性高**
    - 每次请求/响应都发送完整的头部，存在大量重复开销
    - 报文首部不压缩，报文主体压缩
- 靠**多个TCP连接**并发加载资源，数据包丢失时容易造成**队头阻塞**

HTTP2

实现多路复用：

- 在**一个** TCP 连接上同时传输多个请求与响应，解决HTTP/1.1的队头阻塞问题（但仍存在TCP层的队头阻塞）

首部压缩：

- 使用动态表与静态表减少冗余，降低带宽占用

二进制分帧：

- 将报文拆分为**首部帧**和**数据帧**等类型
- 每个帧包含流标识符（`Stream_ID`），可按流独立组装

HTTP3

核心：整合

- 把传输层从**TCP**改为**QUIC（基于UDP）**，解决**TCP队头阻塞**问题
- 保留HTTP2的**二进制分帧**
- **握手**：在**QUIC中集成TLS1.3握手**

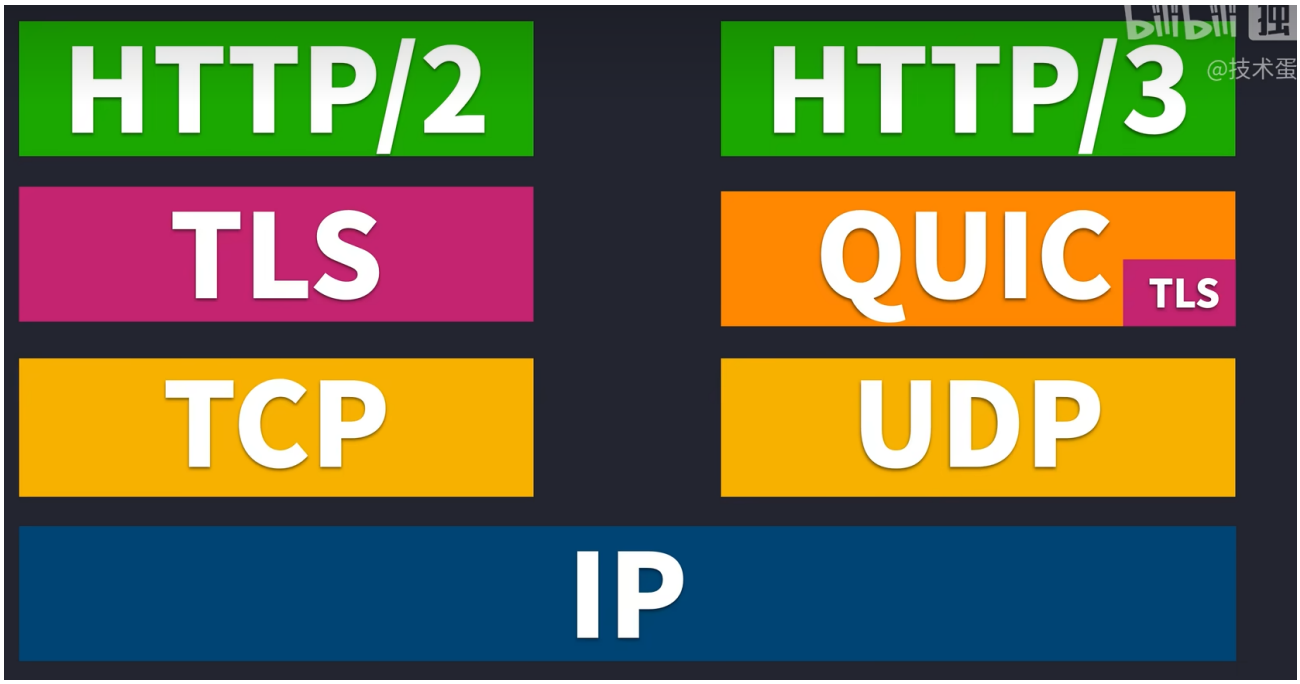

image-20251014171229965

HTTPS

核心：通过**非对称加密**安全的交换一个**对称加密的会话密钥**

默认端口是**443**

是对 `Http` 的升级，后面的 `S` 指的是 `SSL/TLS`

- `Https` = `Http` + `SSL/TLS`
- `SSL/TLS` 是一种加密安全协议，可以对发起http请求的请求和响应进行加密
- `SSL` 是 `TLS` 的前身，现在很多浏览器都支持 `TLS`

对称加密

加密和解密用的是同一个密钥

- 发送方用密钥和加密算法**对明文进行加密**
- 接收方用密钥和加密算法**对密文进行还原**

分发密钥时就会遇到**挑战**，通过网络传输的话**密钥容易被黑客截取**，黑客可以很轻松对密文进行还原

非对称加密

客户端和服务端各使用一把公钥和一把私钥，公钥可进行传递，用于加密，私钥不可泄漏，用于解密

- 发送方在网络上获取公钥，用自己的**私钥**和公钥对信息进行加密
- 接收方同样用**私钥**和公钥对信息进行解密

- 黑客无法知道**私钥**是什么，也就无法解密信息

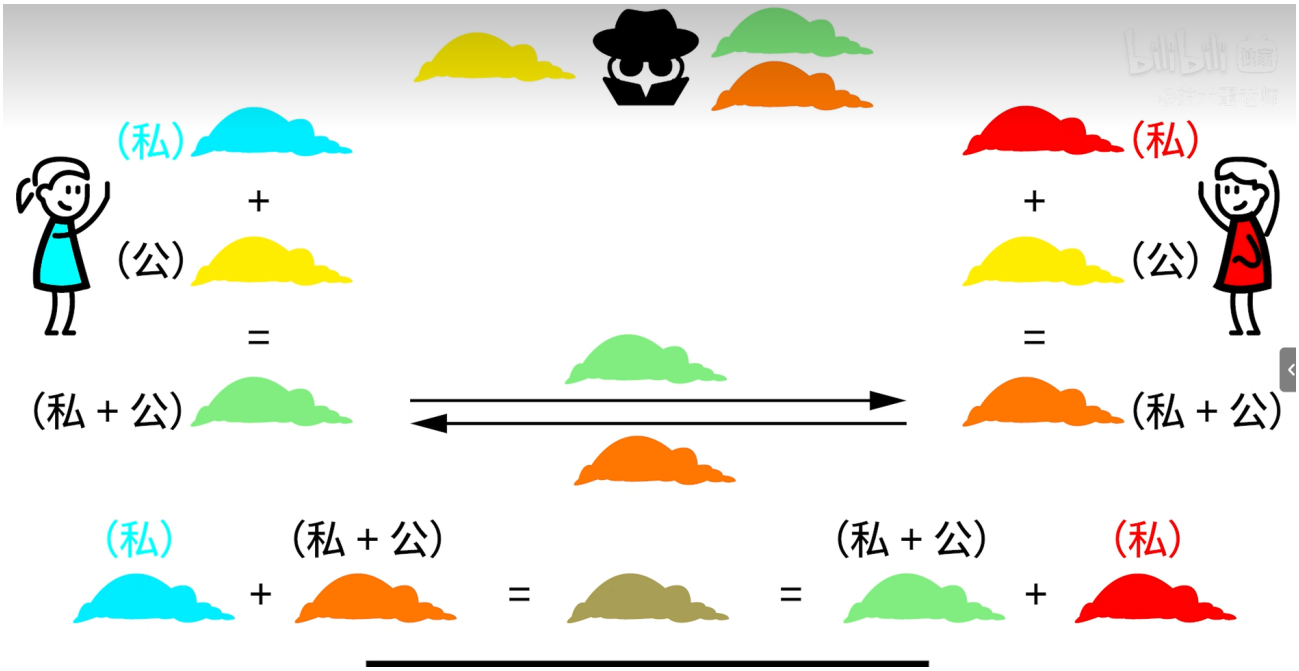

image-20251012193353851

SSL证书

将网站的**身份信息与一个公钥**进行绑定，并由权威的**证书颁发机构（CA）**进行数字签名，证明其真实性

TLS1.2握手流程（在这之前，服务端和客户端会进行TCP连接）

1. 客户端发送 `Client Hello` ，TLS版本，加密套件和**第1随机数**（ `Client Random` ）给服务端
   - 第1随机数为一个由**客户端**生成的随机字符串
   - 客户端发送**它支持的加密套件列表**供服务端选择
2. 服务端发送 `Server Hello` ，TLS版本，加密套件和**第2随机数**（ `Server Random` ）给客户端
   - 第2随机数为一个由**服务端**生成的随机字符串
   - 此时返回给客户端服务端所选择的**加密套件**
3. 服务端发送**证书**给客户端
   - 证书包含了服务器的**公钥、域名、颁发机构、有效期**等信息
4. 服务端进行 `Server Key Exchange` ，发送自己的**临时公钥**
5. 服务端发送 `Server Hello Done` ，告诉客户端**信息发送完毕**
6. 客户端**验证证书**，如果证书不通过，连接中止
7. 客户端生成一个**预主密钥**，使用**服务端提供的公钥**进行加密，然后发送给**服务端**
8. 服务端利用自己的**私钥解密**得到**预主密钥**
   - 客户端和服务端按照**约定好的算法对这三个随机数生成相同的会话密钥**
9. 客户端告诉服务端**切换密码规范**，使用刚刚生成的**会话密钥**对会话内容进行加密
10. 客户端发送 `Client Finished` 完成消息
11. 服务端发送 `Finished` 消息

TLS1.3将握手往返次数从2次减少到1次，显著降低了延迟，会话密钥都是由对方公钥加上自己的私钥和公钥生成的

握手总结

- **预主密钥**是客户端生成的第3个随机数
- 后续的信息传输使用**会话密钥**

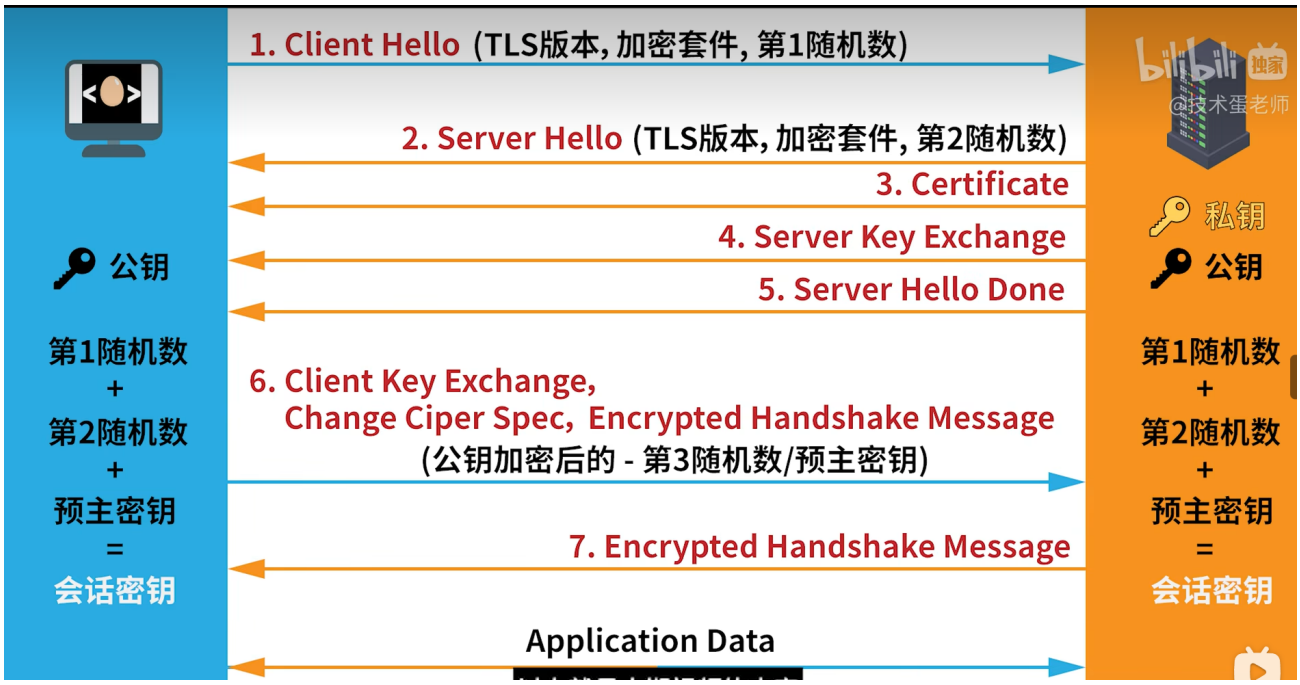

image-20251012195424252

QUIC

对协议层进行重构

- 融合了 `HTTP2` 、 `TLS` 、 `TCP` 的特征
- 顶层使用 `HTTP3`
- 传输层使用 `UDP`
- 使用 `QPACK` 进行数据压缩

image-20251016120726087

QUIC数据包

- 应用数据**拆分**成QUIC流
- QUIC流**组合**成QUIC帧
- QUIC帧**连接**成QUIC包

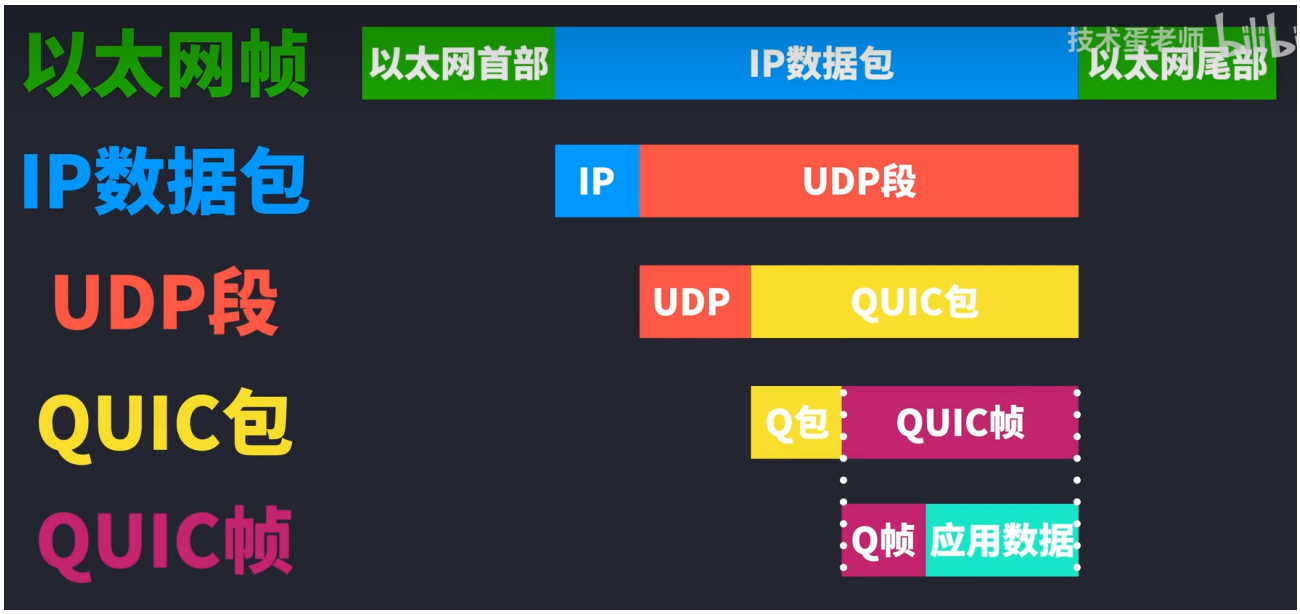

image-20251016122413482

传输时延

- 首次通信是1RTT，此时**首部数据会较长**，后续加密通信**首部会较短**
- 通信的恢复是0RTT

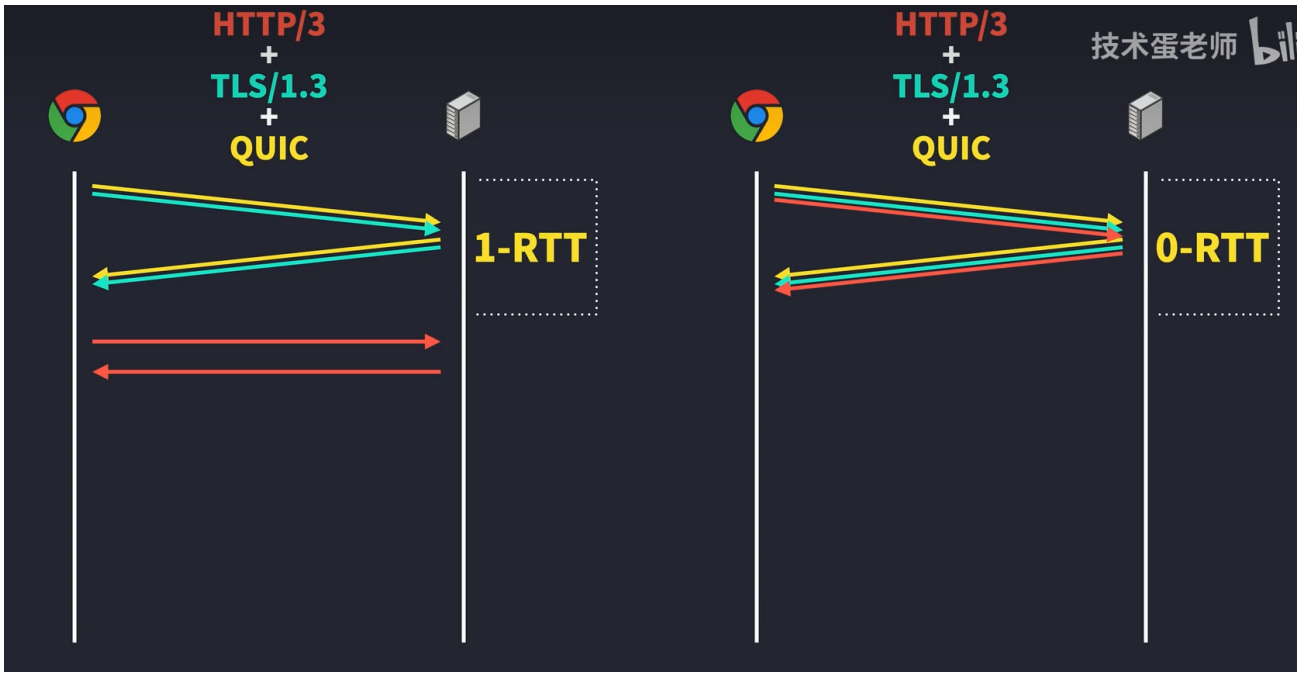

image-20251016121348576

加密

- QUIC对连接ID等信息进行了加密
- 导致ISP难以根据连接ID进行流量监测

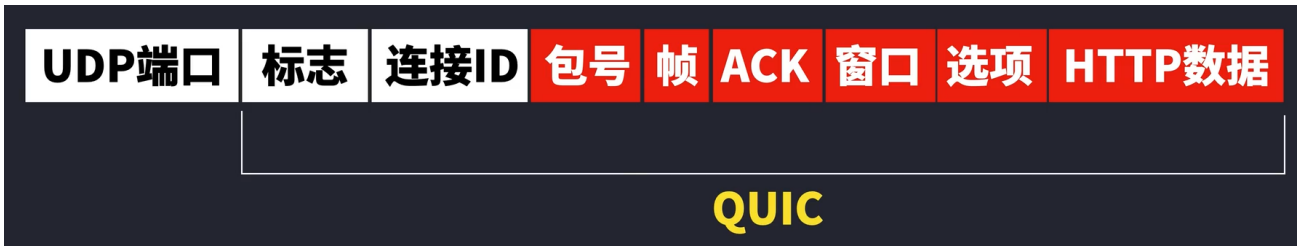

image-20251016122723629

RPC

远程过程调用： `Remote Procudure Call`

- 基于UDP/TCP等传输层协议，也可以配合HTTP等应用层协议
- 在分布式系统中，让一个应用像调用本地函数一样去调用另一台机器上的方法，拥有**跨语言能力**
- 有多种实现，如 `grpc` 、 `Dubbo` 等
- **序列化方法**也有多种实现，常见有 `protobuf` 等

流程

- 客户端发起调用
- 客户端存根（ `stub` ）处理，对**函数名、数据等**进行**序列化**，封装成网络信息
- 网络传输
- 服务端接收
- 服务端存根（ `skeleton` ），对请求进行**反序列化**，还原出函数名和参数
- 服务端执行业务代码
- 服务端存根将结果进行**序列化**并发送给客户端
- 客户端对传输数据**反序列化**出结果

grpc

rpc的一种实现方法，由Google实现

- 支持多种语言
- 基于IDL文件定义服务
- 基于HTTP2设计
- 序列化支持 `Protocol Buffer` 和 `JSON`
  - `PB` 使用 `.proto` 文件定义**数据结构和接口规范**，使用自带的编译器为**多种编程语言**生成对应的代码
    - 包括**消息结构体、客户端和服务端接口代码**
  - `protobuf` 比 `JSON` 小3-10倍，解析速度快5-20倍左右
- 支持拦截器：**Unary interceptor**（一元）和**Stream interceptor**（流式）
- `metadata` ，元数据，类似HTTP的header，可以存放 `token` 、 `rid` 等

通信模式

- 一元RPC（ `Unary RPC` ）：客户端发送**单个请求**，服务器返回**单个响应**
- 服务端流式RPC（ `Server Streaming RPC` ）：客户端发送**单个请求**，服务器返回**多个响应（数据流）**
- 客户端流式RPC（ `Client Streaming RPC` ）：客户端发送**多个请求（数据流）**，服务端返回**单个响应**
- 双向流式RPC（ `Bidrectional Streaming RPC` ）：客户端和服务端都可以发送**多个请求/响应**