

# 计网常见协议

xbZhong

2025-10-16

## Contents

计网常见知识 . . . . .	1
本页 PDF	

## 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

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

### 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

#### 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

## OSI参考模型

7	应用层
6	表示层
5	会话层
4	传输层
3	网络层
2	数据 链路层
1	物理层

## TCP/IP协议



Figure 1: image-20251013214454674



Figure 2: image-20251016115152094



Figure 3: image-20251013121751180

## 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

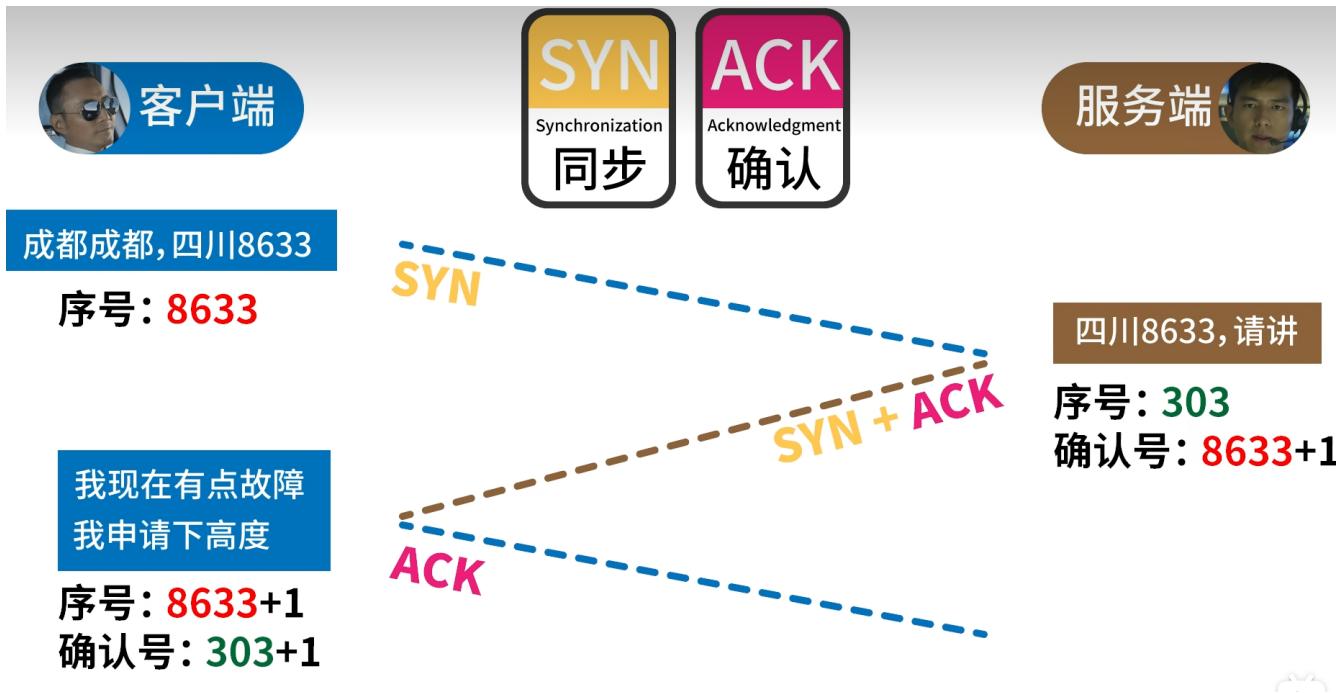


Figure 4: image-20251013130002349

## 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**

### 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



Figure 5: image-20251013124408713

## 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

**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， $\text{发送窗口} = \min(\text{拥塞窗口}, \text{接收窗口})$

### 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



Figure 6: image-20251016123303300

- 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)

**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
- $\approx BtlBw \times RTprop$

## 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

## 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

## 拥塞控制

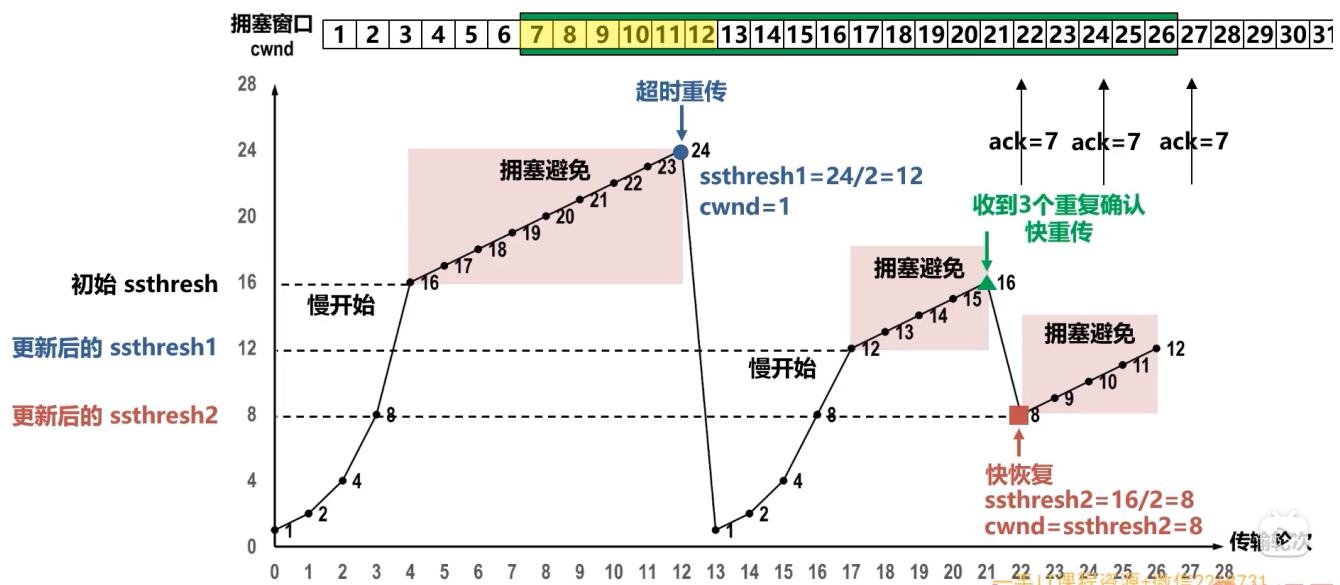


Figure 7: image-202510160000066624

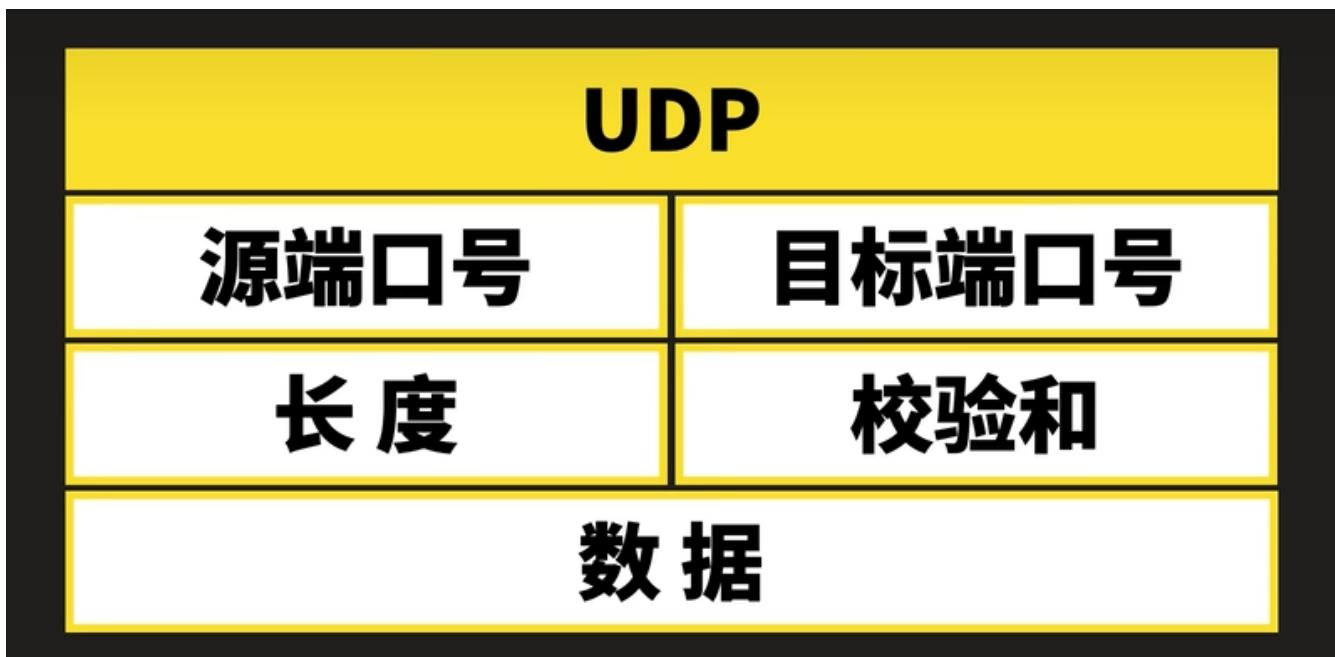


Figure 8: image-20251013121843576

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**

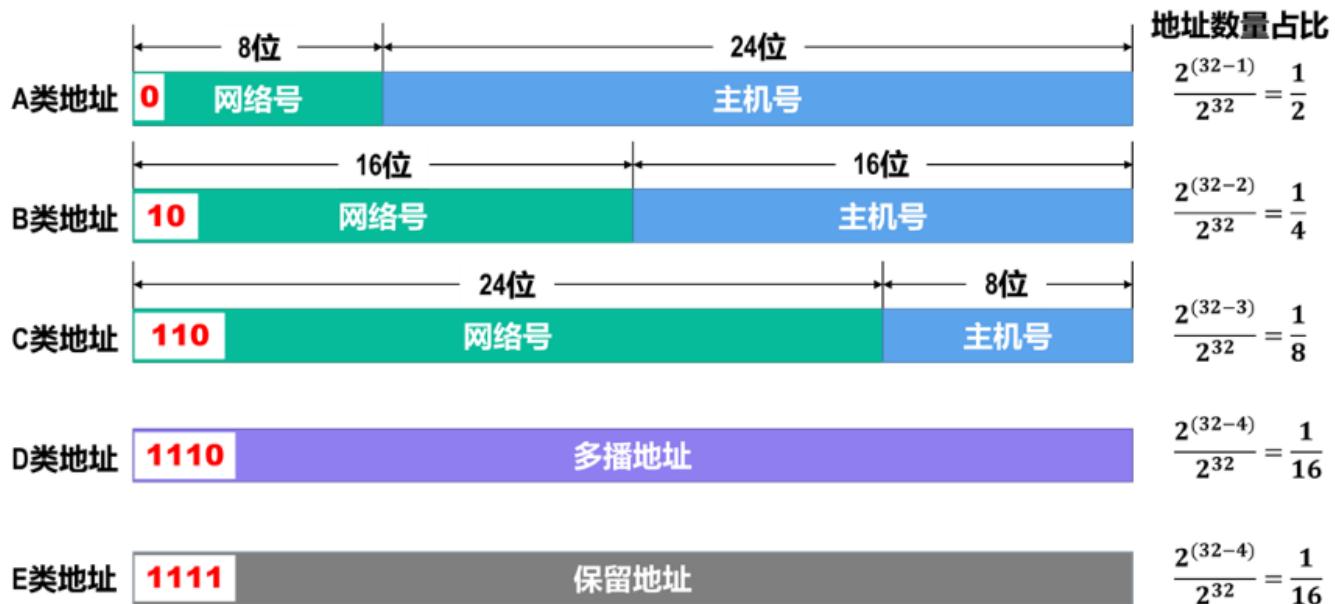


Figure 9: image-20251013112742698

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

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

**11000000.10101000.00000000.00000000**

**11000000.10101000.00000000.00000001**

**11000000.10101000.00000000.00000010**

**11000000.10101000.00000000.00000011**

**11111111.11111111.11111111.11111100**

Figure 10: image-20251013113529447

## 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



Figure 11: image-20251013121120752

### 地址类型

#### NAT

##### 网络地址转换

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

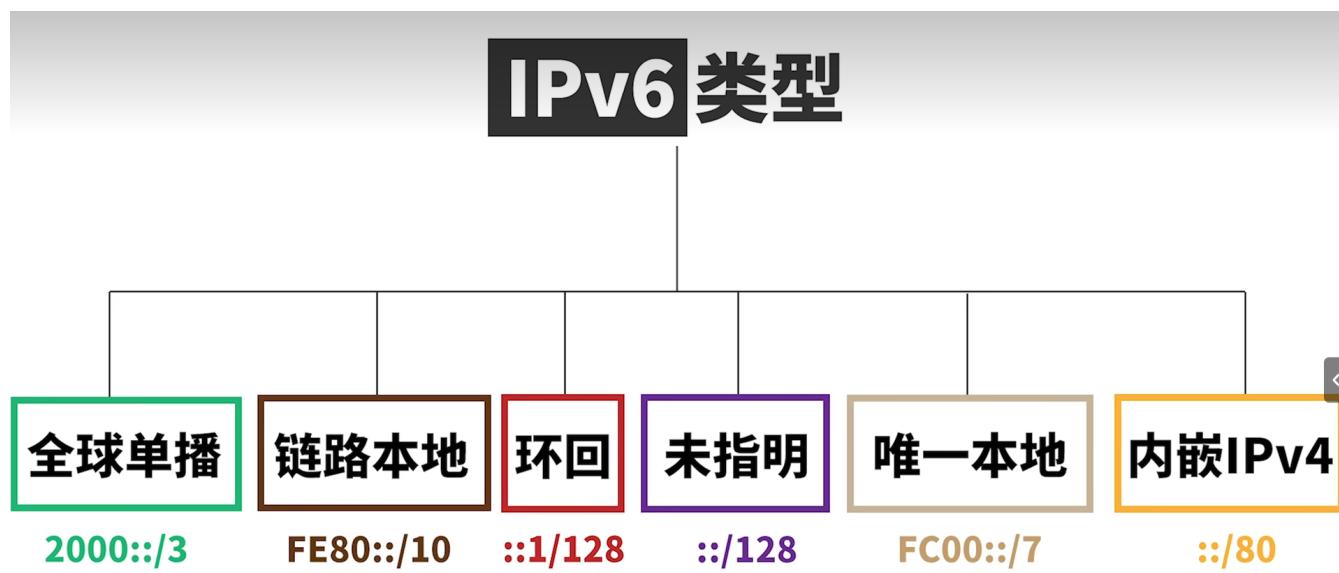


Figure 12: image-20251013121537369

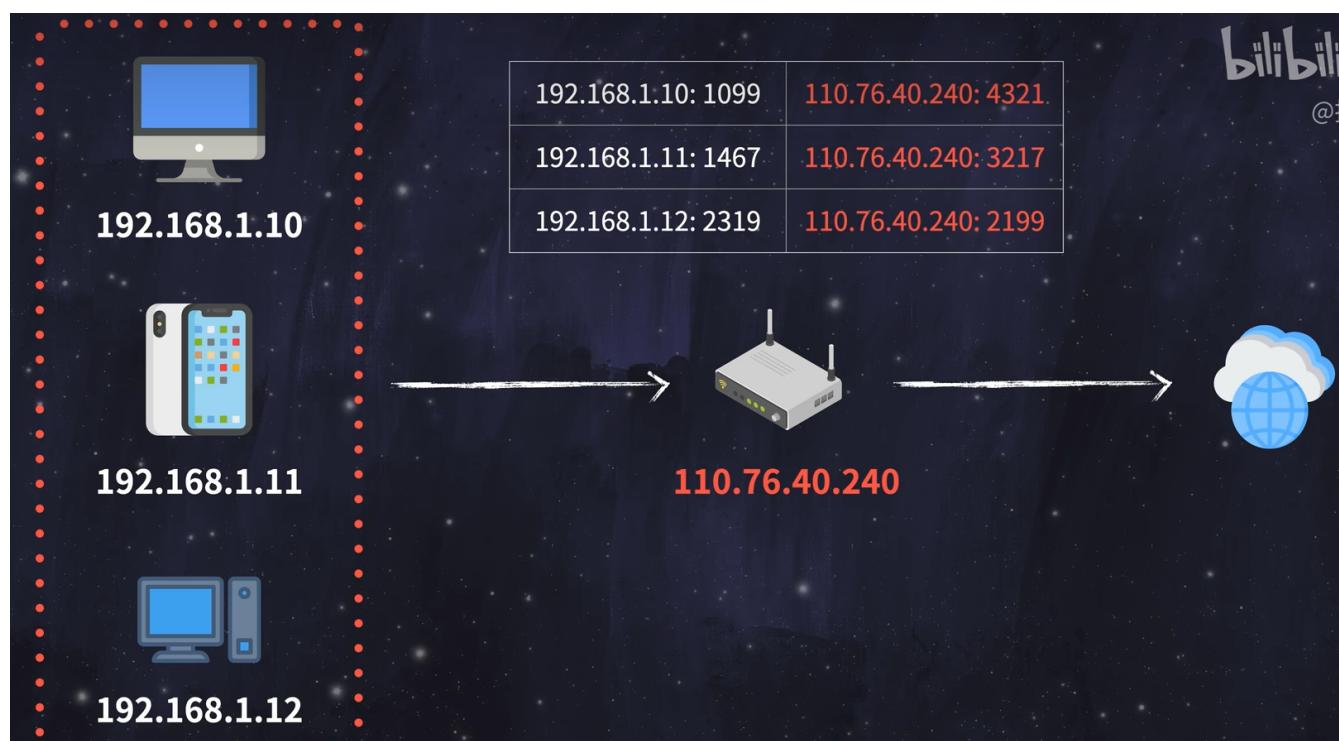


Figure 13: image-20251013222829174

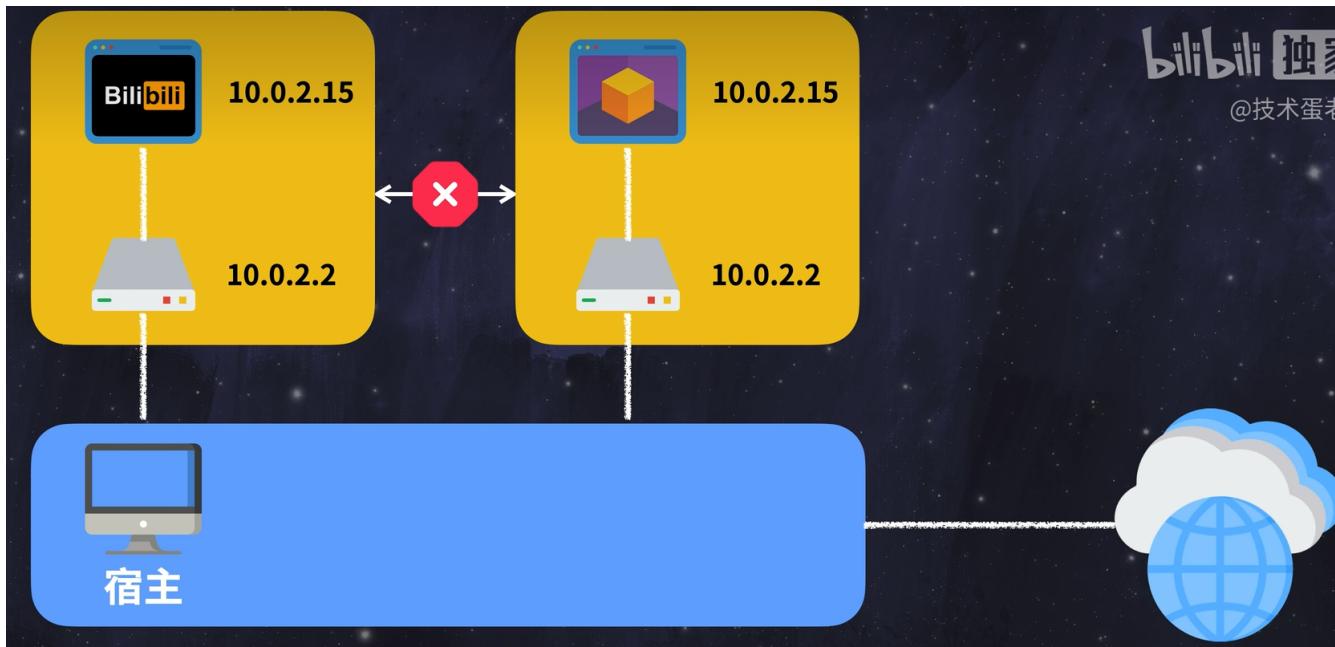


Figure 14: image-20251013221831854

- 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

- 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

- 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，这样交换机接收到之后就会进行广播

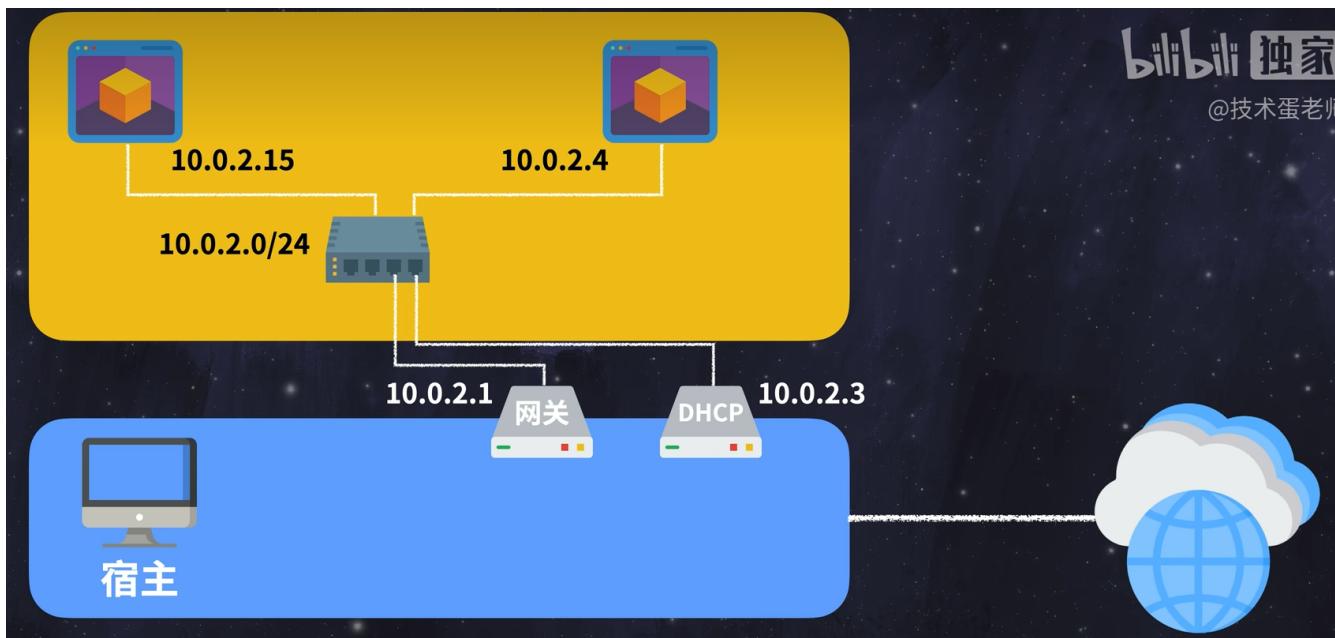


Figure 15: image-20251013222040076



Figure 16: image-20251013222333237

- 数据链路层与 IP 地址道理相同

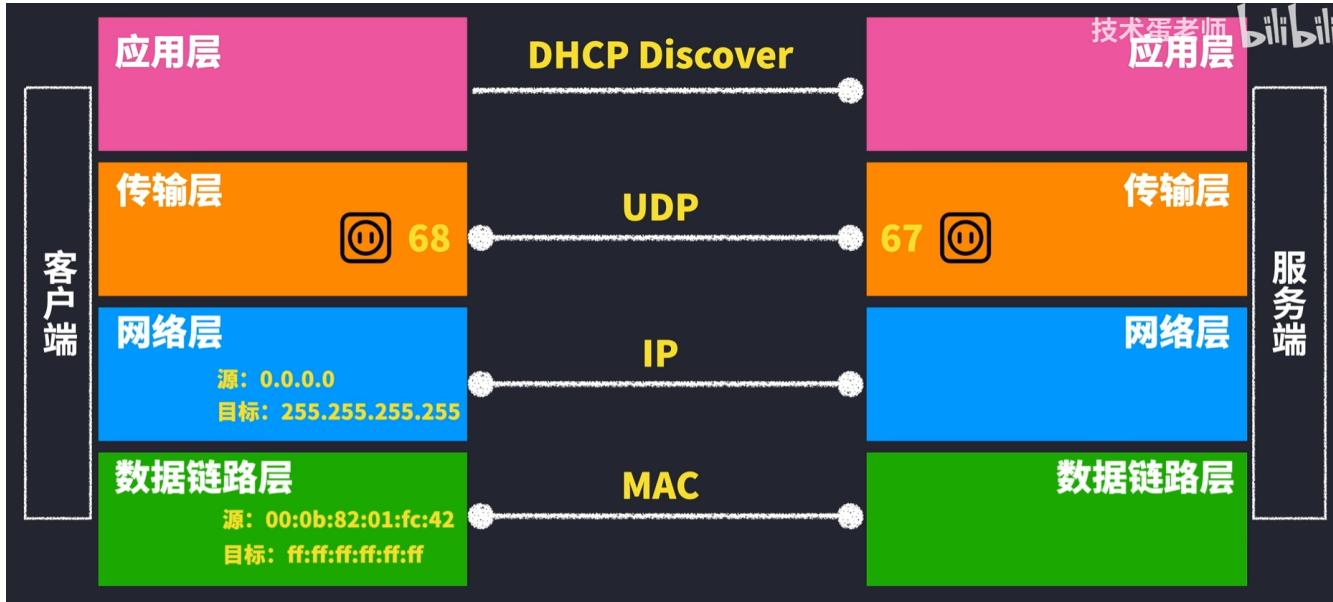


Figure 17: image-20251014122953171

- 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
- 客户端进行 **DHCP Request**, 广播所有 DHCP 服务器客户端选择了哪个 IP
- 服务端发送 **DHCP ACK**, 新设备可以开始上网

## 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**

### 域名服务器类型

#### 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**

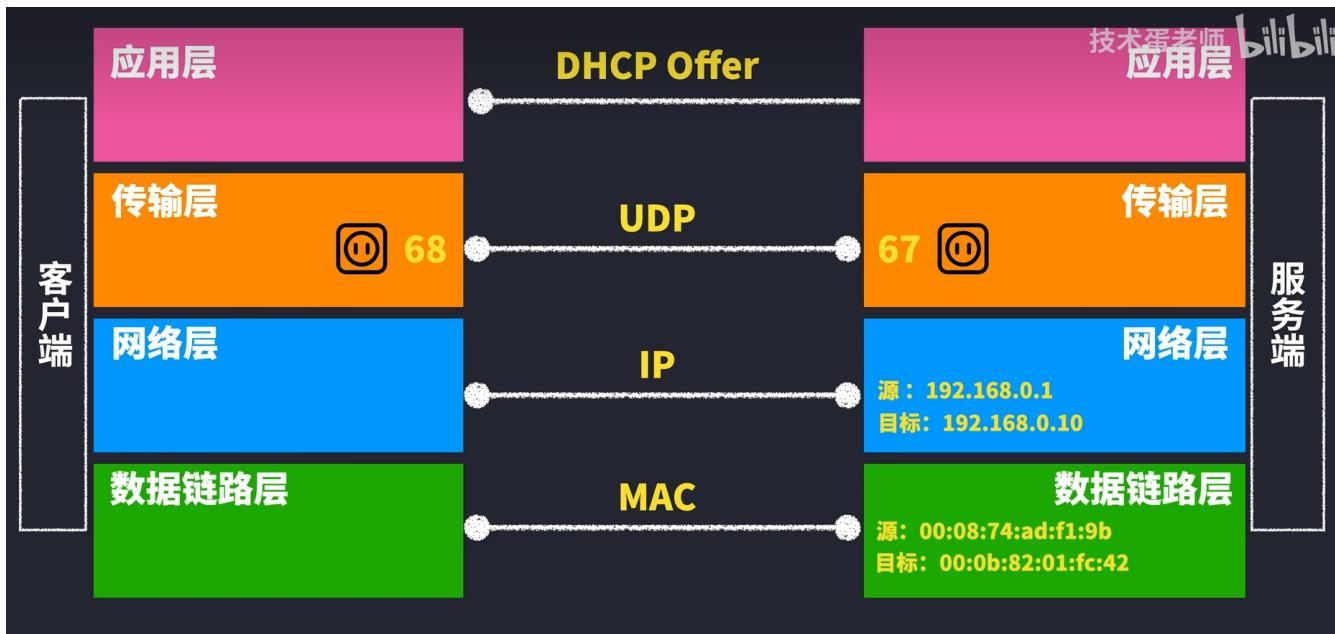


Figure 18: image-20251014123604469

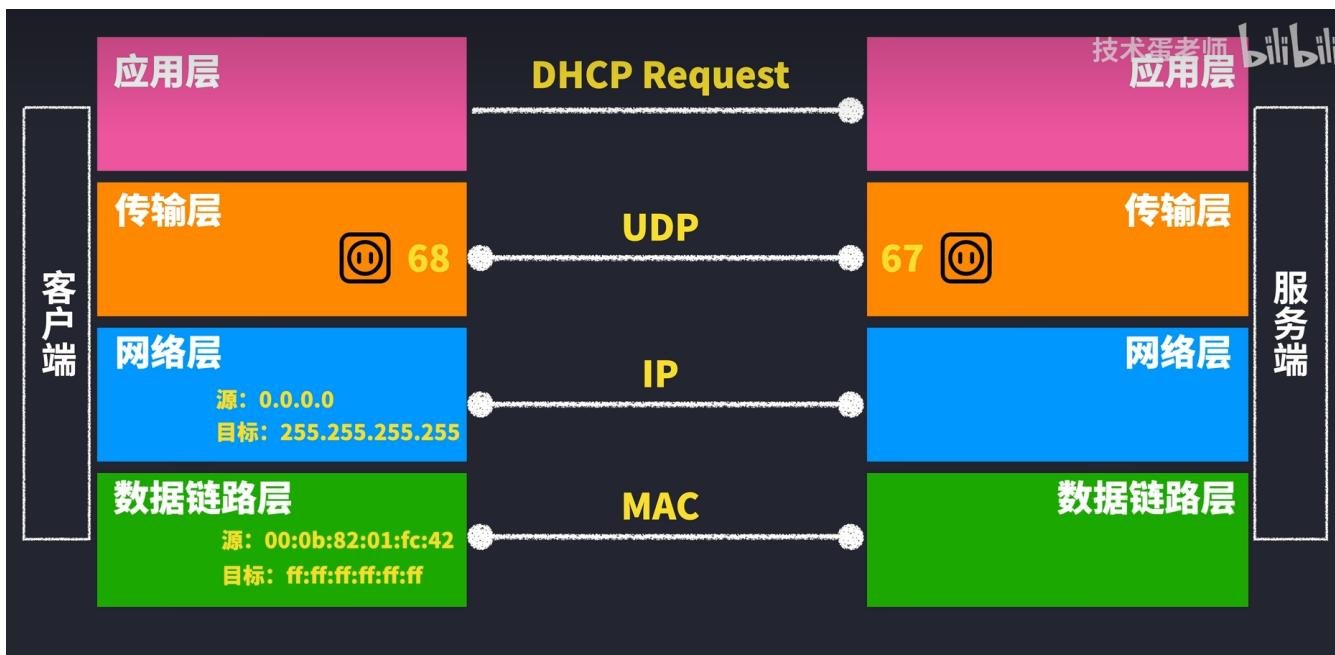


Figure 19: image-20251014123732540



Figure 20: image-20251014123747948

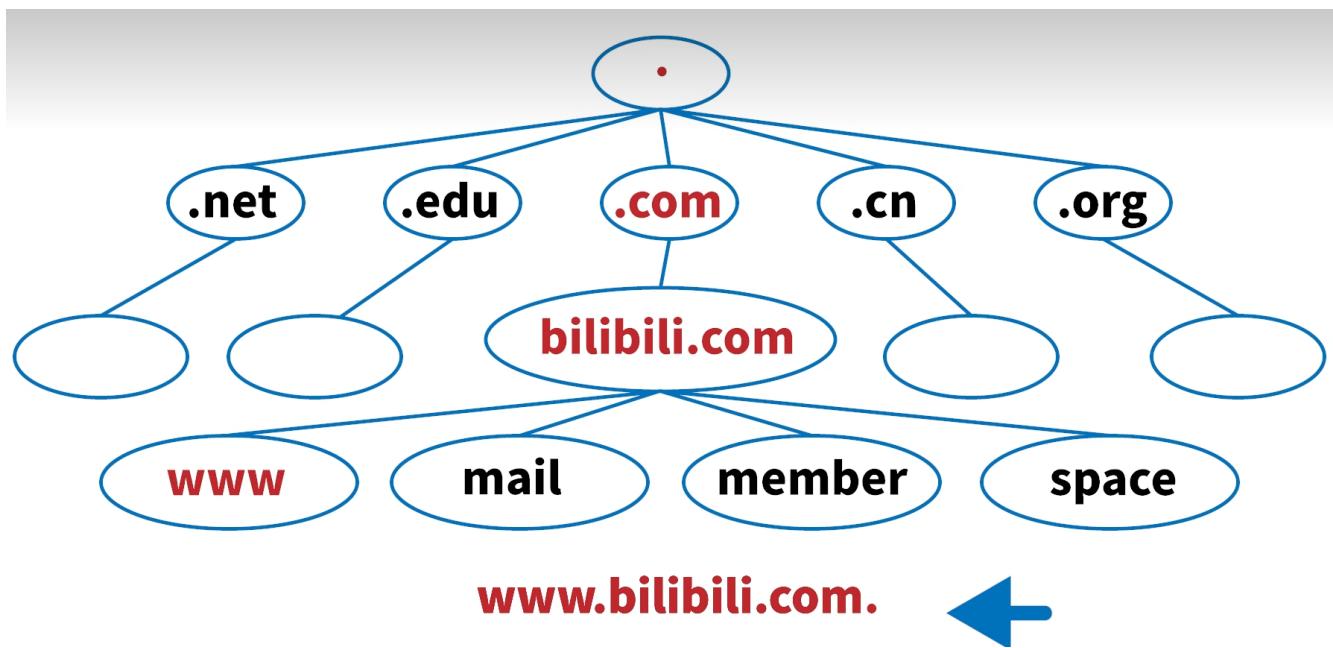


Figure 21: image-20251013200644701

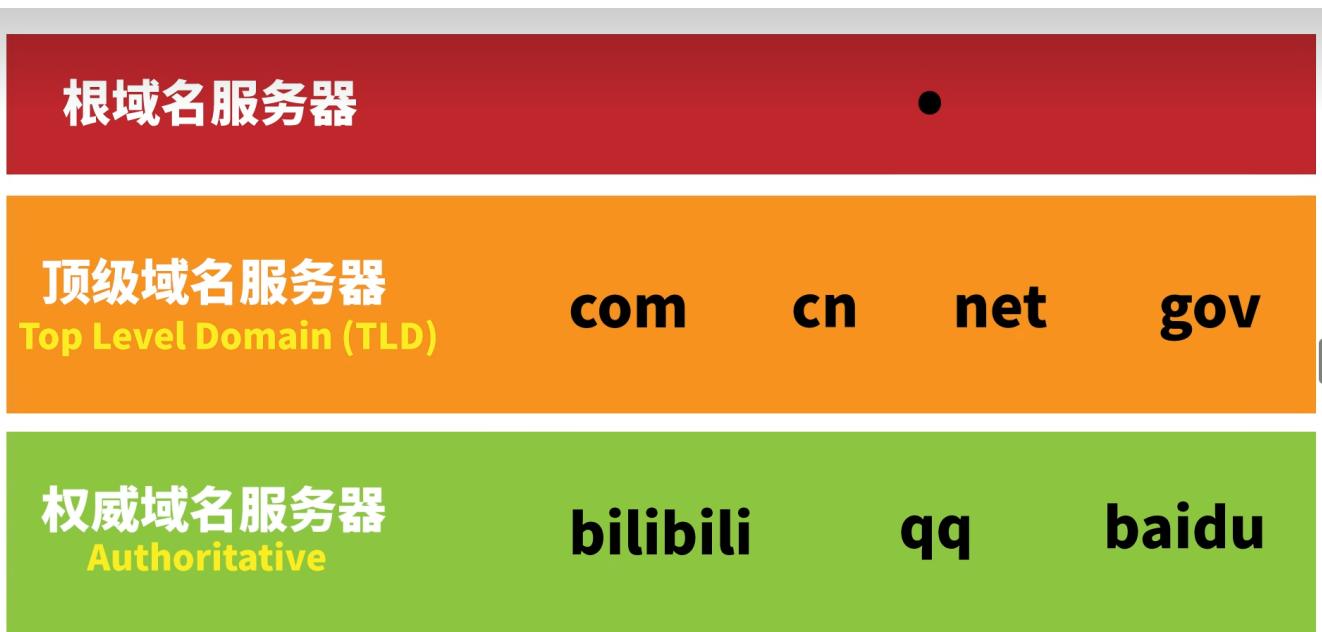


Figure 22: image-20251013201120277

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

### 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 握手过程中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

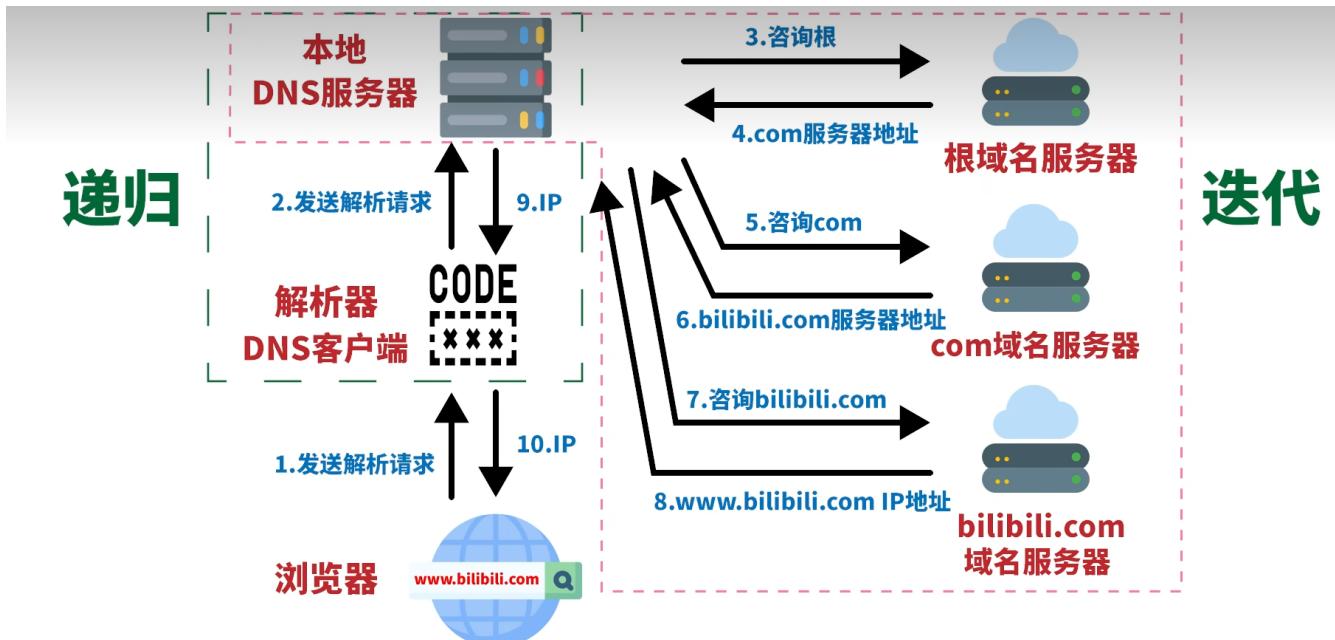


Figure 23: image-20251013201551042

- 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

**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



Figure 24: image-20251013220614655

- **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

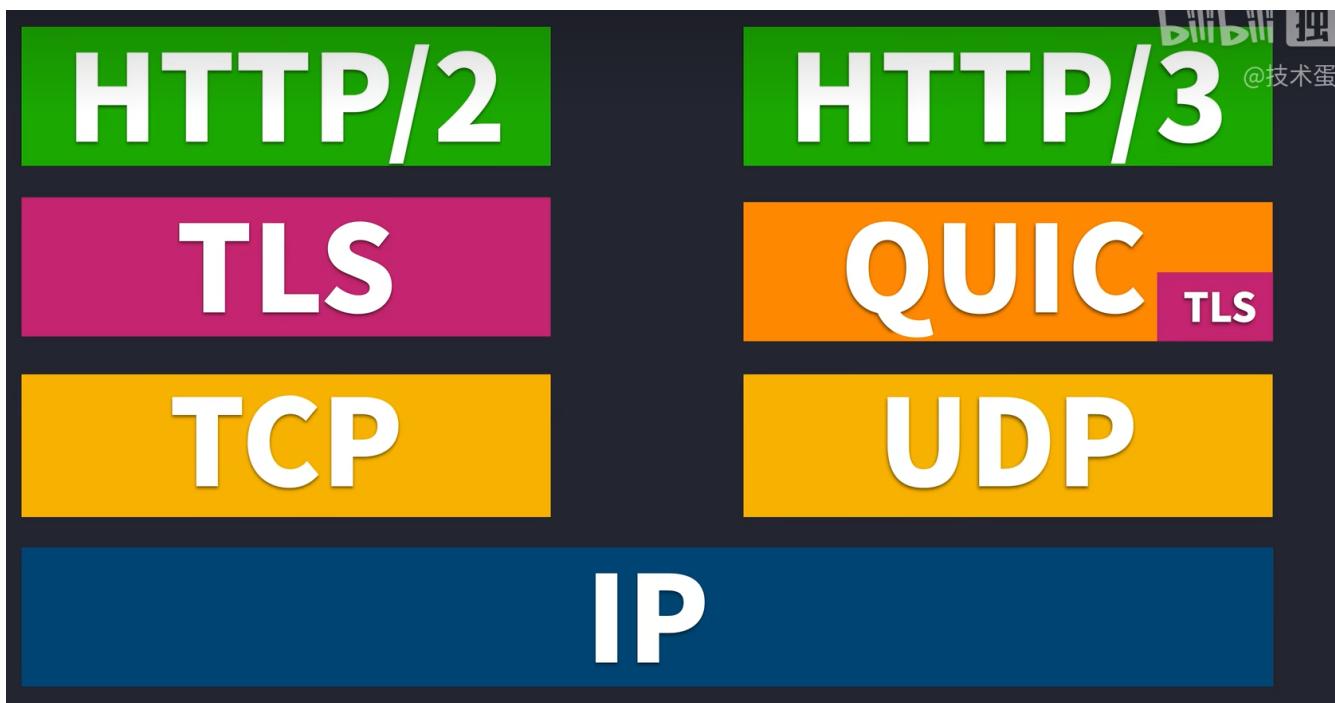


Figure 25: image-20251014171229965

## 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

### 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）

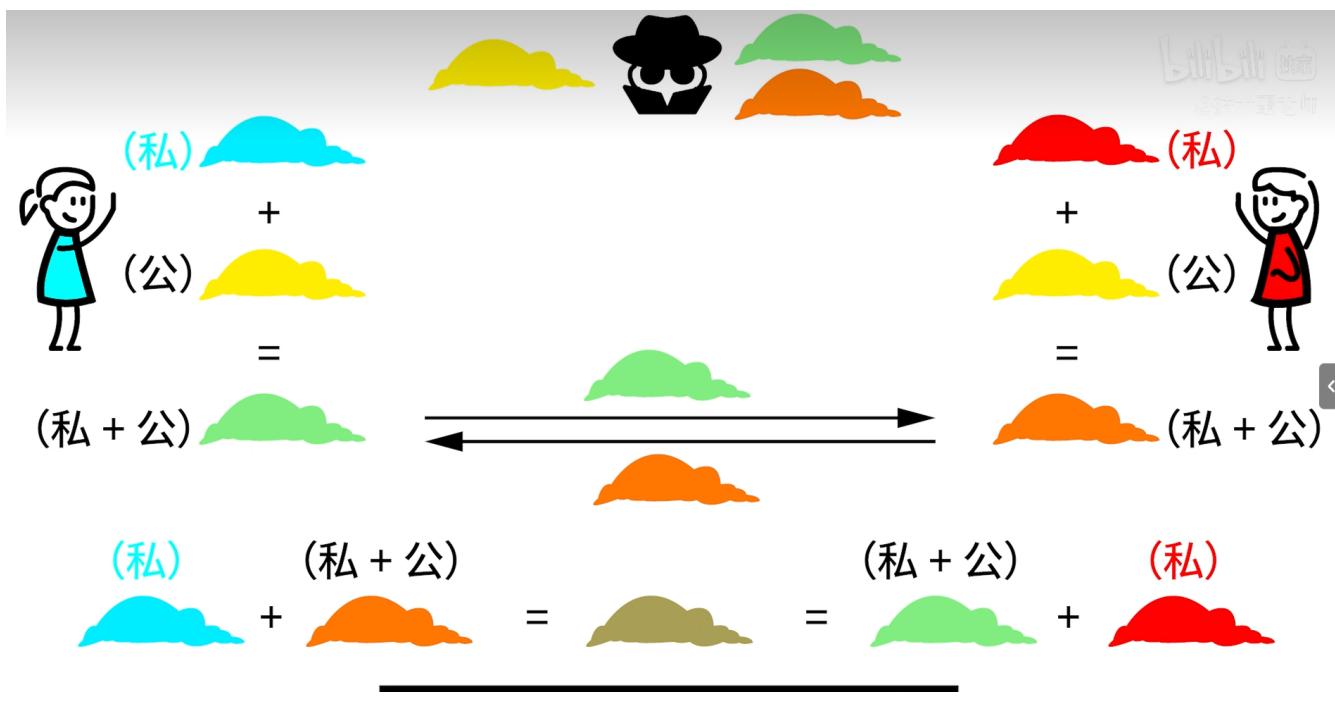


Figure 26: image-20251012193353851

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**

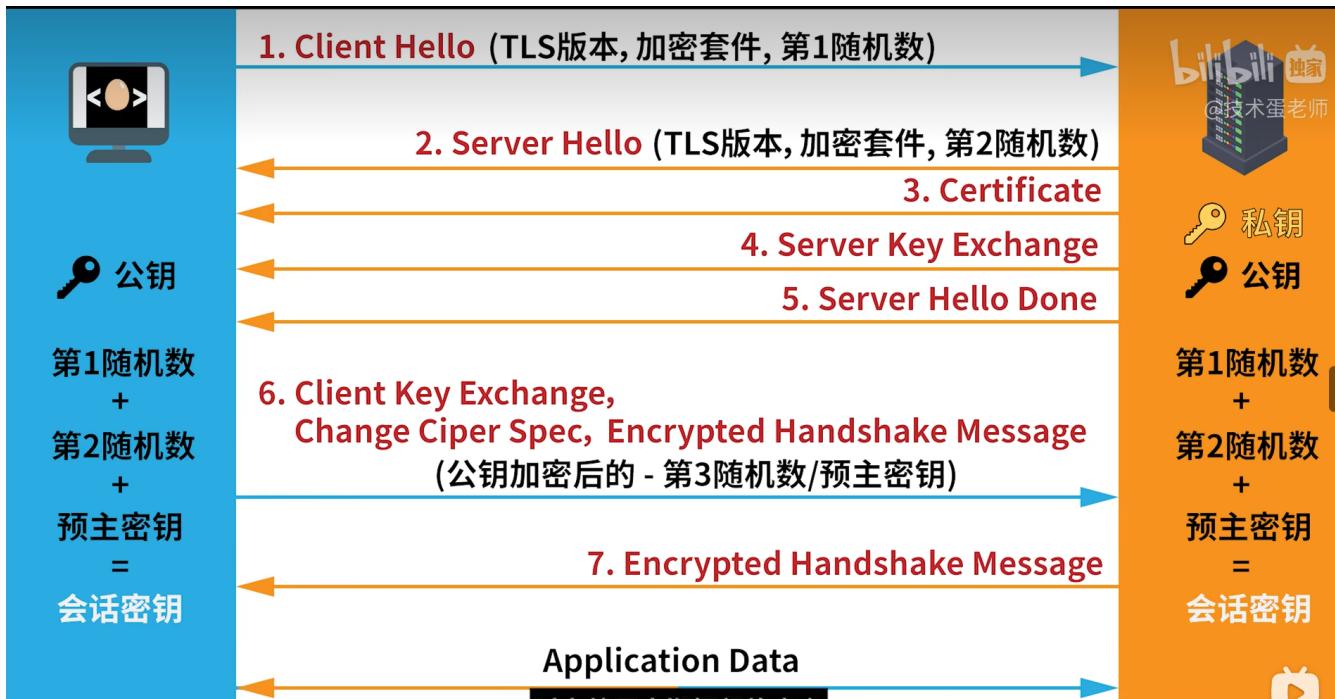


Figure 27: image-20251012195424252

## 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

### 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

### 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

### 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



Figure 28: image-20251016120726087

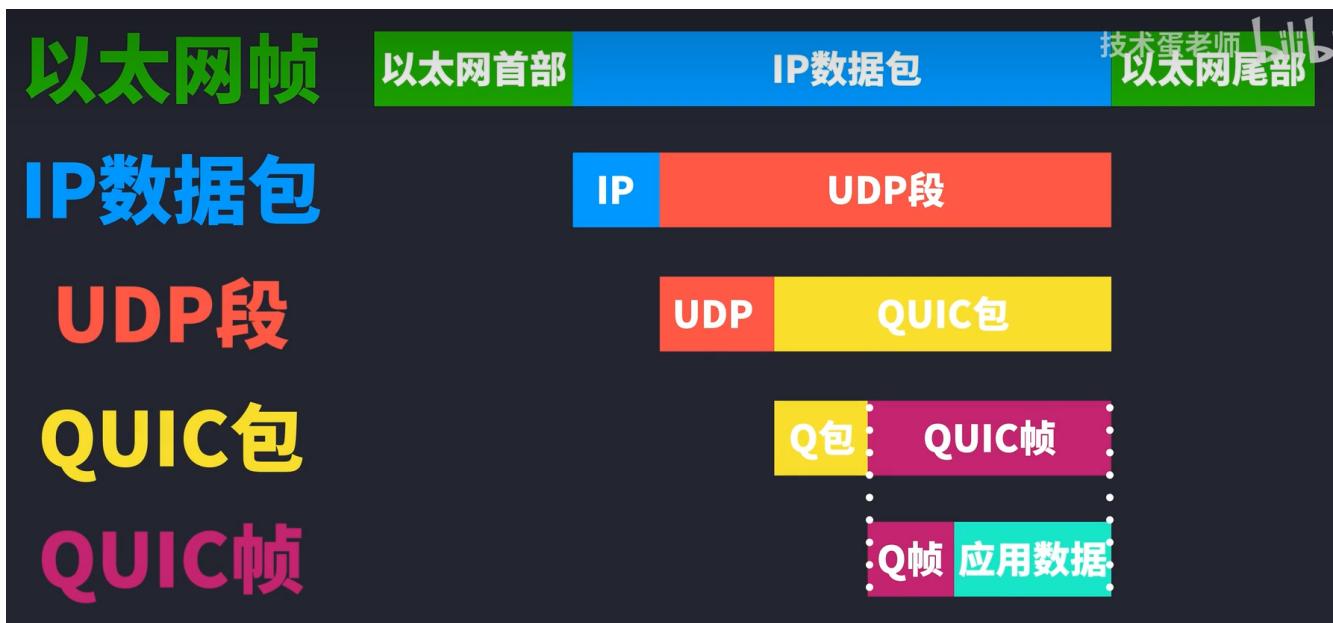


Figure 29: image-20251016122413482

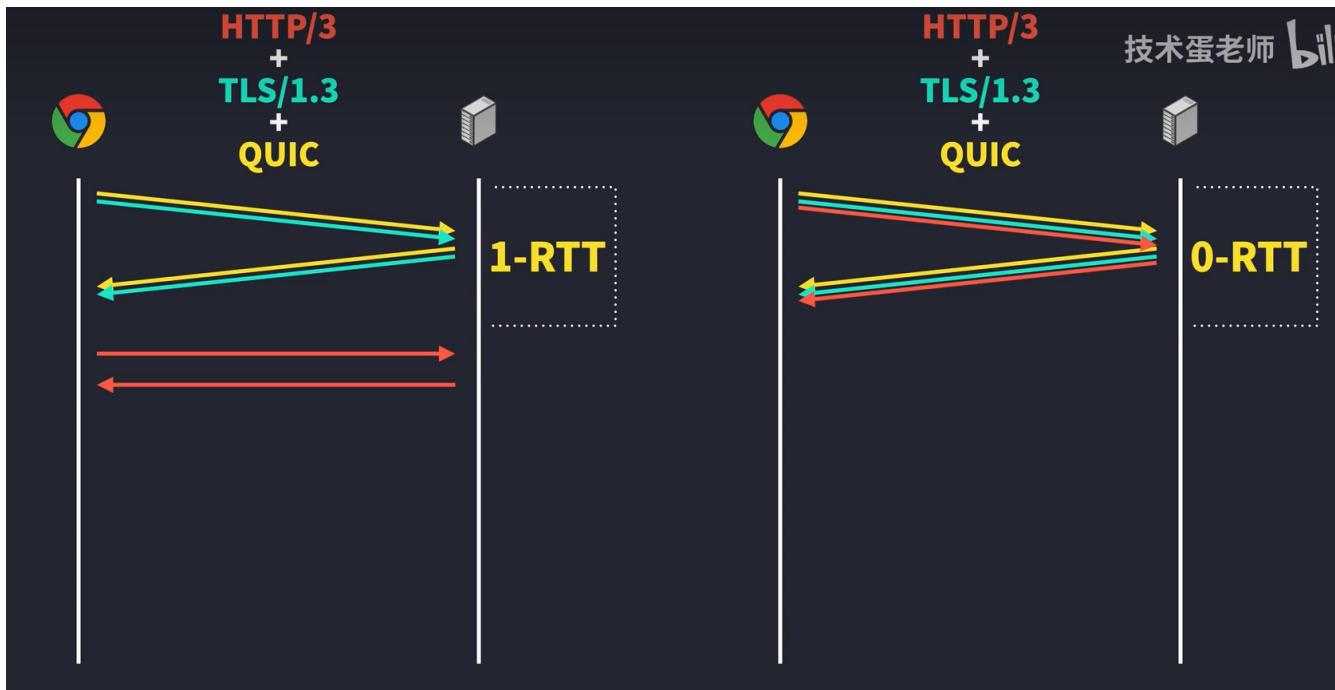


Figure 30: image-20251016121348576



Figure 31: image-20251016122723629