解读从HTTP1.1到HTTP3的进化过程

1. HTTP1.1的进化与缺陷

1.1 HTTP1.1的进化

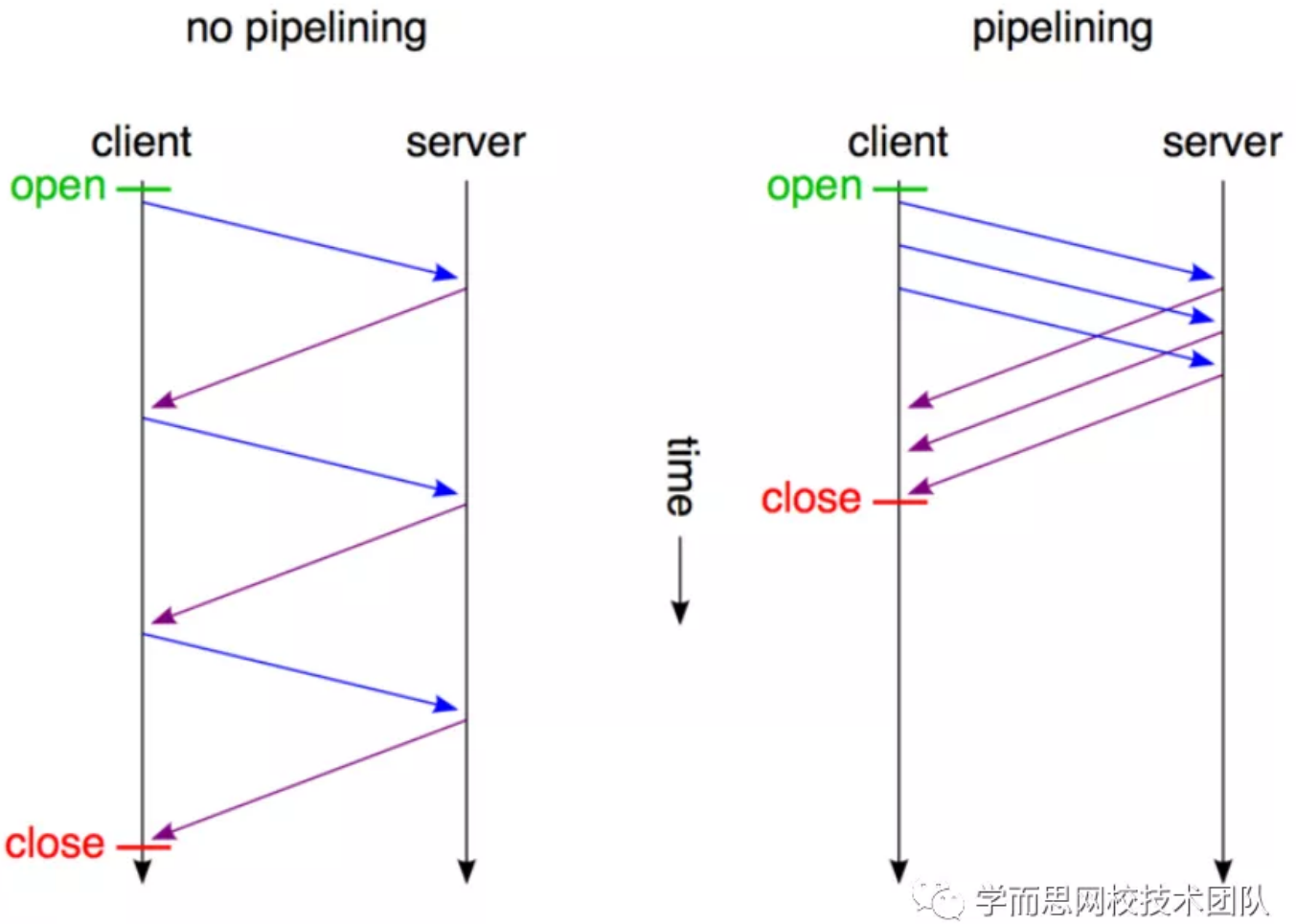
1.1.1 增加持久性连接

由于每一次TCP连接都是非常耗费资源的操作，所以在HTTP1.0版本增加持久性连接的特性。

```
Connection: keep-alive
```

1.1.2 增加管道机制

HTTP1.1版本改进了1.0版本的排队方式，增加了管道机制，请求可以同时发出，但是响应必须按照请求发出的顺序依次返回。



1.1.3 分块传输

在HTTP1.0版本中，如果在服务器端遇到较为耗费时间的操作。

那么需要等到这一操作全部完成后，才会向客户端发送数据，那么这一段等待时间是非常影响性能和客户体验。

HTTP1.1版本对上述问题进行了改进，可以没必要等待数据完全处理完毕再返回，服务器产生部分数据，那么就发送部分数据，可以节省很多等待时间。

1.2 HTTP1.1的缺陷

1.2.1 TCP 连接数限制

对于同一个域名，浏览器最多只能同时创建 6~8 个 TCP连接 。为了解决数量限制问题将资源放到不同域名下 ，但是也会造成其他问题。

例如每个 TCP 连接本身需要经过 DNS 查询、三步握手、慢启动等，对于服务器来说过多连接也容易造成网络拥挤、阻塞等

1.2.2 队头阻塞

每个 TCP 连接同时只能处理一个请求 - 响应，浏览器按 FIFO 原则处理请求，如果上一个响应没返回，后续请求 - 响应都会受阻。

为了解决此问题出现了 Pipelining技术，但是Pipelining存在诸多问题，比如第一个响应慢还是会阻塞后续响应、服务器为了按序返回相应需要缓存多个响应占用更多资源、

浏览器中途断连重试服务器可能得重新处理多个请求、还有必须客户端 - 代理 - 服务器都支持Pipelining

1.2.3 明文传输不安全

HTTP1.1使用明文传输内容，客户端和服务器端都无法验证对方的身份，数据的传输过程可被被篡改和劫持，出现了篡改注入广告和劫持进行大数据分析等恶性案例。

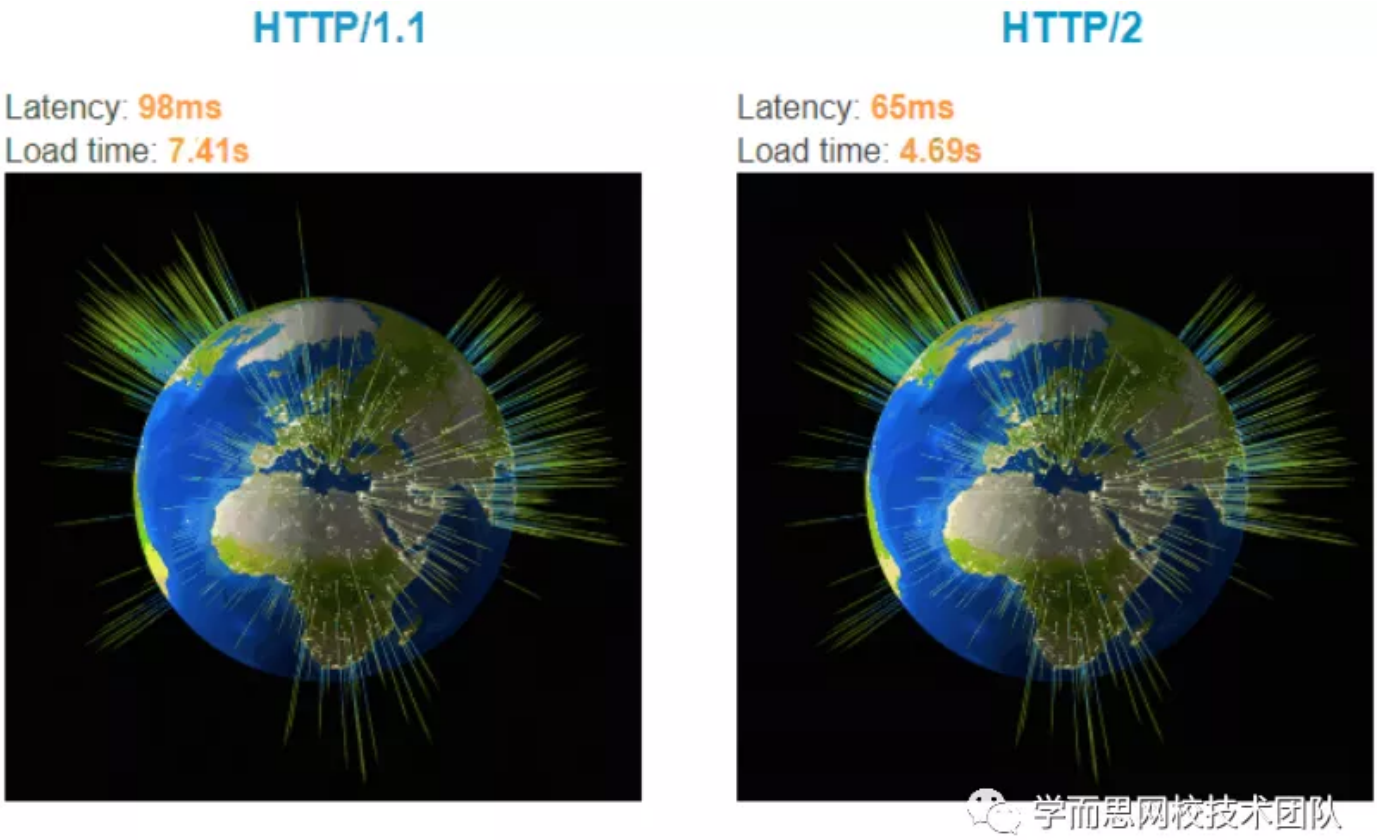
2. HTTP2的进化与缺陷

2.1 HTTP2的特性

HTTP2传输数据量较HTTP1.1大幅减少, 主要受惠于以二进制分帧传输和HPACK头部压缩，可以达到50%~90%的高压缩率。

HTTP2在客户端和服务器间只需用一个连接（connection），使用HTTP2能带来20%~60%的效率提升。

下图可以看到HTTP2比HTTP1.1快多少



2.1.1 二进制分帧

HTTP2 完全兼容 HTTP1.1的语义，在应用层使用二进制分帧方式传输。因此，也引入了新的通信单位：帧、消息、流。

HTTP2 将请求和响应数据分割为更小的帧，并且它们采用二进制编码。

把HTTP1.1的"Header+Body"的消息分为数个小片的二进制帧(Frame)，用HEADERS帧存放Header数据、DATA帧存放Body数据。

HTTP2数据分帧后"Header+Body"的报文结构就完全消失了，协议看到的只是一个个的"碎片"

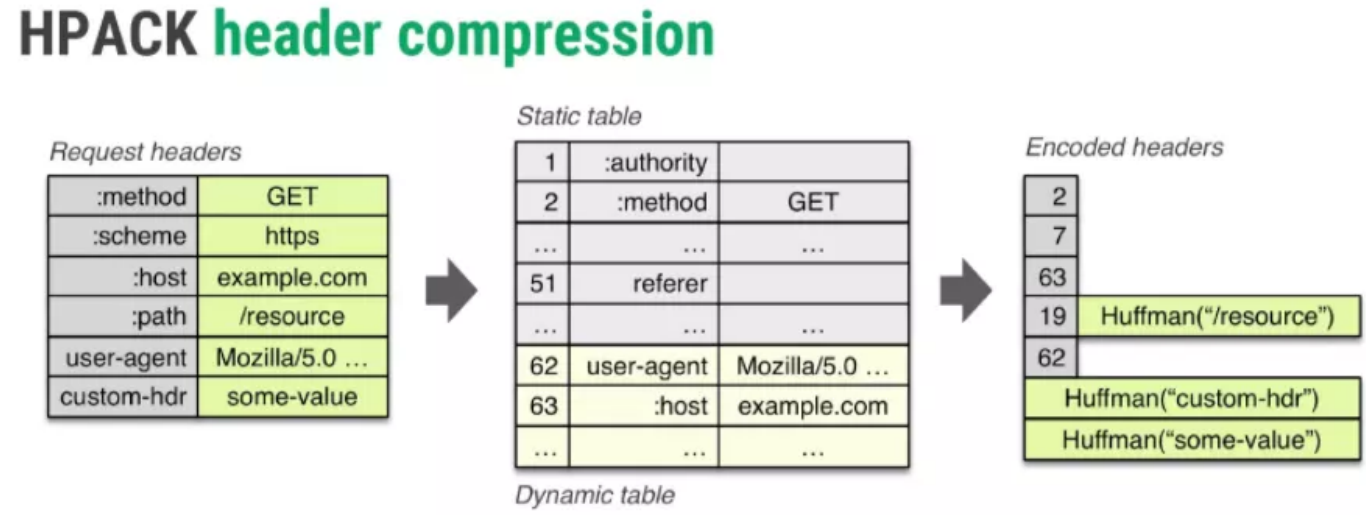
分帧有什么好处?

- 服务器单位时间接收到的请求数变多，可以提高并发数
- 为多路复用提供了底层支持

2.1.2 HPACK头部压缩

HTTP1.1使用文本的形式传输 header，在 header 携带 cookie 的情况下，每次都需要重复传输几百到几千的字节。

HTTP2并没有使用传统的压缩算法，基于Huffman编码开发了HPACK 头部压缩，可以达到50%~90%的高压缩率，在客户端和服务器两端建立字典，用索引号表示重复的字符串。



- *Literal values are (optionally) encoded with a static Huffman code*
- *Previously sent values are (optionally) indexed*
  - *e.g. "2" in above example expands to "method: GET"*

2.1.3 多路复用

在 HTTP2 中，实现了二进制分帧之后：

- 同域名下所有通信都在单个连接上完成；
- 单个连接可以承载任意数量的双向数据流；
- 数据流以消息的形式发送，而消息又由一个或多个帧组成，多个帧之间可以乱序发送，因为根据帧首部的流标识可以重新组装。

在HTTP1.1的协议中，request和response都是基本于文本明文传输的，所有的数据必须按顺序传输，所以并行传输在HTTP1.1是不能实现的。

HTTP2引入二进制数据帧和流的概念，其中帧对数据进行顺序标识，这样浏览器收到数据之后，就可以按照序列对数据进行合并，服务器就可以并行的传输数据，这就是流所做的事情。

HTTP2对同一域名下所有请求都是基于流，也只建立一个连接，因为有了这个新特性，提高了服务端最大的并发连接数

2.2 HTTP2的缺陷

2.2.1 TCP+TLS建连时延

HTTP1.1版本出现众多劫持和注入的事件，主流的浏览器Chrome、Firefox等都公开宣布只支持加密的HTTP2，

必须标配TLS完成HTTPS加密传输，以目前TLS1.2+TCP的建连过程需要2RTT完成。

RTT（Round-Trip Time）:往返时延。表示从发送端发送数据开始，到发送端收到来自接收端的确认（接收端收到数据后便立即发送确认），总共经历的时延。

一次请求的时间去哪了？

- DNS查询: 获取IP。DNS服务一般默认是由你的ISP提供，ISP 通常都会有缓存的，这部分时间能适当减少；
- TCP连接: 需要和服务器进行三次握手，即消耗完 1.5 个 RTT 之后才能进行数据传输；
- TLS握手: 以目前应用最广泛的TLS 1.2而言，需要2个RTT。对于非首次建连, 选择启用Session Resumption，则可缩小握手时间到1个RTT；
- HTTP业务数据交互:，假设服务端的数据在一次交互就能取回来。那么业务数据的交互需要1个RTT。

总耗时：新连接：4RTT + DNS，会话重用：3RTT + DNS

2.2.2 队头阻塞没有彻底解决

TCP 为了保证可靠传输，有一个超时重传机制，丢失的包必须等待重传确认。HTTP2 出现丢包时，整个 TCP 都要等待重传，那么就会阻塞该 TCP 连接中的所有请求。

2.2.3 单连接带来的问题

HTTP2 使用了多路复用，同一域名下只需要使用一个 TCP 连接。

当这个 TCP 连接中出现了丢包的情况, 为 TCP为了保证可靠传输，有个“丢包重传”机制，丢失的包必须要等待重新传输确认，

HTTP2出现丢包时，整个 TCP 都要开始等待重传，那么就会阻塞该TCP连接中的所有请求。

对于 HTTP/1.1 来说，可以开启多个 TCP 连接，出现这种情况反到只会影响其中一个连接，剩余的 TCP 连接还可以正常传输数据。

3. HTTP3的进化

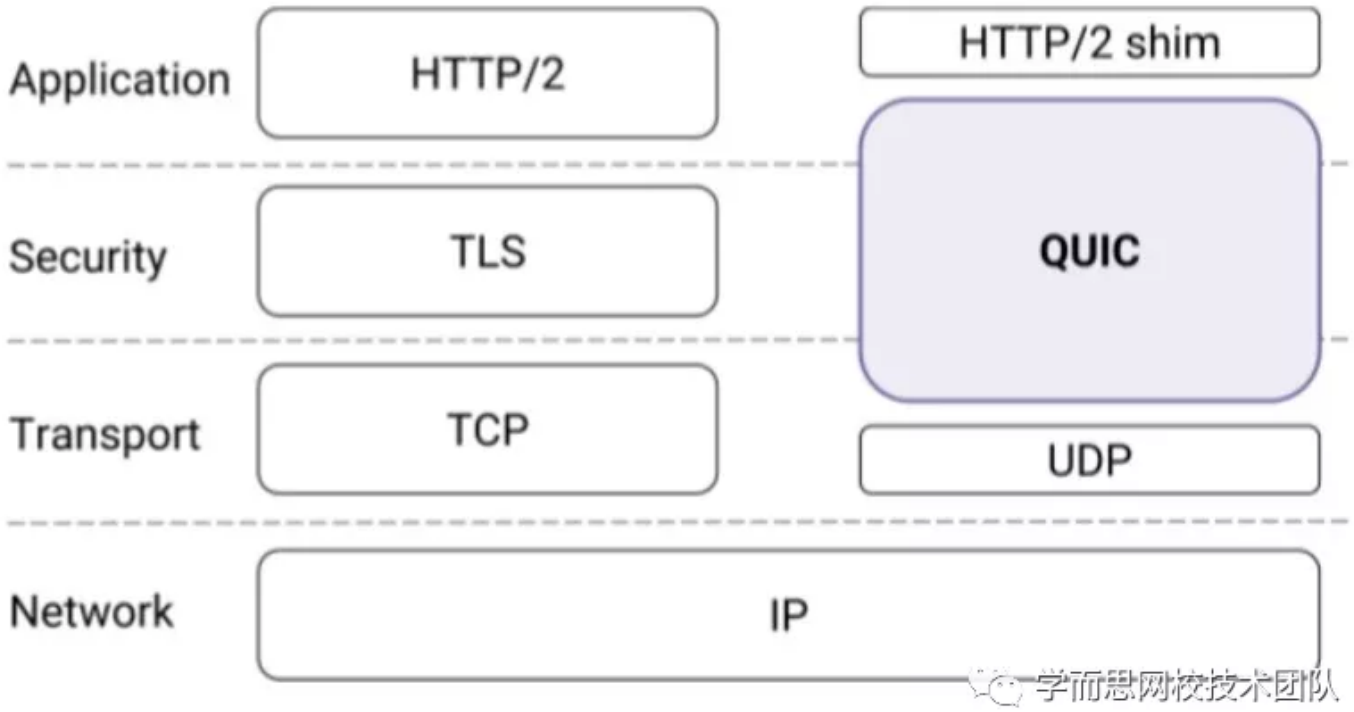
HTTP3 = HTTP2 over QUIC

HTTP3真正地解决了队头阻塞问题。QUIC实现了在同一物理连接上可以有多个独立的逻辑数据流。实现了数据流的单独传输，就解决了TCP中队头阻塞的问题。

3.1 QUIC的优势

QUIC取代了大多数传统的HTTPS堆栈：HTTP/2、TLS和TCP，图1显示了TCP和QUIC之间应用程序堆栈的区别。

QUIC的主要特点包括：0-RTT连接 减少丢包；前向纠错，减少重传延迟；自适应拥塞控制，连接迁移。



3.1.1 单调递增的 Packet Number

每个 Packet Number 都严格递增，使用 Packet Number 代替了 TCP 的 seq，也就是说就算 Packet N 丢失了，重传的 Packet N 的 Packet Number 已经不是 N，

而是一个比 N 大的值。而 TCP 重传策略存在二义性，比如客户端发送了一个请求，一个 RTO 后发起重传，而实际上服务器收到了第一次请求，

并且响应已经在路上了，当客户端收到响应后，得出的 RTT 将会比真实 RTT 要小。当 Packet N 唯一之后，就可以计算出正确的 RTT。

3.1.2 无队头阻塞的多路复用

HTTP2的最大特性就是多路复用，而HTTP2最大的问题就是队头阻塞。

HTTP2在一个TCP连接上同时发送3个stream，其中第2个stream丢了一个Packet，TCP为了保证数据可靠性，需要发送端重传丢失的数据包，虽然这时候第3个数据包已经到达接收端，但被阻塞了。这就是所谓的队头阻塞。

而QUIC多路复用可以避免这个问题，因为QUIC的丢包、流控都是基于stream的，所有stream是相互独立的，一条stream上的丢包不会影响其他stream的数据传输。

3.1.3 改进的拥塞控制

QUIC协议默认使用TCP协议的Cubic拥塞控制算法，QUIC协议在TCP拥塞算法基础上做了些改进：

可插拔：

- 应用程序层面就能实现不同的拥塞控制算法，不需要内核支持
- 单个应用程序的不同连接也能支持不同配置的拥塞控制
- 不需要停机和升级就能实现拥塞控制的变更

3.1.4 连接迁移

TCP 的连接标识是通过 “源IP + 源Port + 目标IP + 目标Port + 协议号“ 组成的五元组，一旦其中一个参数发生变化，则需要重新创建新的 TCP 连接。

而 QUIC 的连接标识是一个 64位的Connection ID，用户在 WiFi 和5G切换时，无论是 IP 或者端口（Port）发生变化，QUIC 连接中的连接 ID 保持不变，

因此不需要重新创建连接。这种用户无感知的网络切换特性，叫连接迁移。

3.1.5 向前纠错

每个数据包除了它本身的内容之外，还包括了部分其他数据包的数据，因此少量的丢包可以通过其他包的冗余数据直接组装而无需重传。

向前纠错牺牲了每个数据包可以发送数据的上限，但是减少了因为丢包导致的数据重传，因为数据重传将会消耗更多的时间（包括确认数据包丢失、请求重传、等待新数据包等步骤的时间消耗）

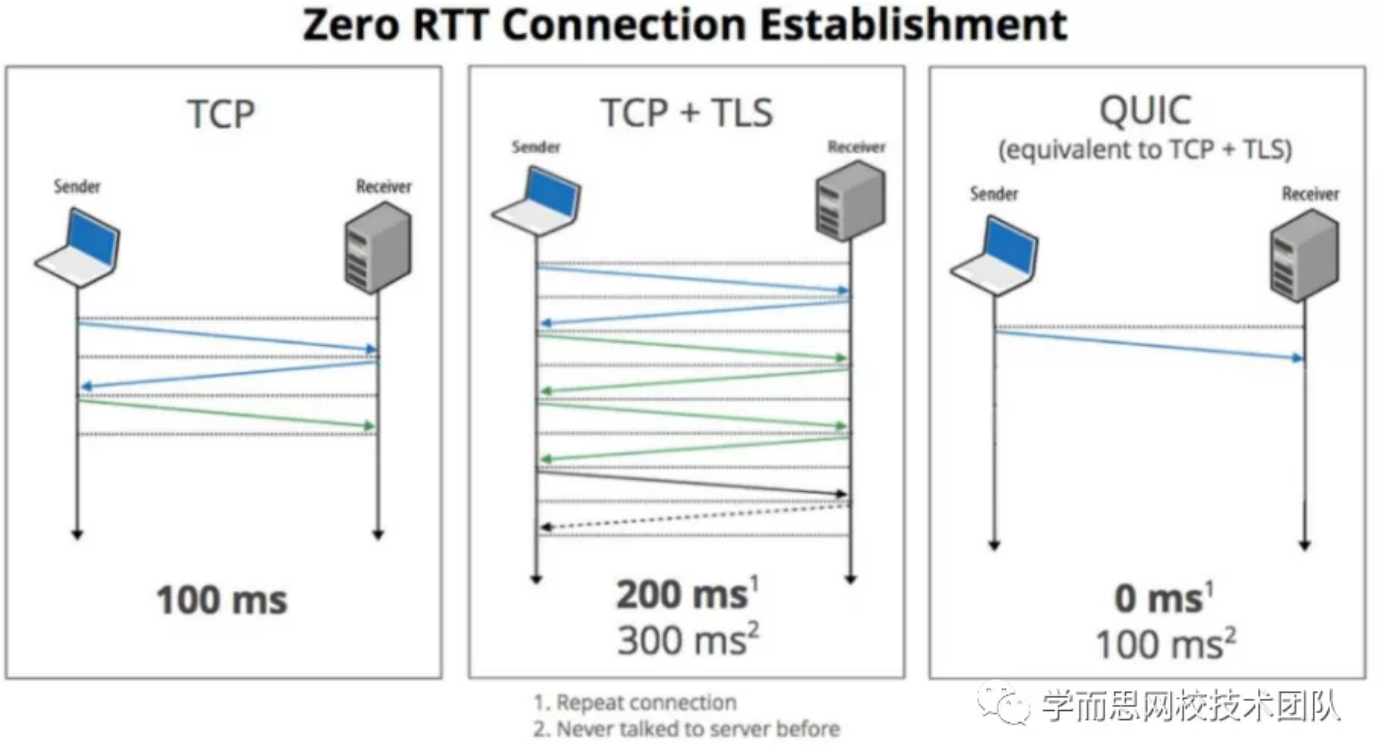
3.2 QUIC 握手

TCP需要三次握手来建立连接，还需要协商TLS连接。因此希望在建立新的联系时减少往返时间（RTT）的影响。

通过将TCP和TLS集成在一个协议中，QUIC可以避免两次顺序握手。QUIC可以更重要地完全避免往返，称为0-RTT连接延迟。

以前与服务器通信的客户机可以使用存储在客户机和服务器上的有限状态启动新会话，而无需三方握手。这将从连接建立中删除多个rtt。

有关TCP/TLS和QUIC之间建立连接的简要比较，请参见下图。



实际上，如果客户机和服务器在过去进行过通信，那么我们讨论的是零握手连接—75%的情况下都是这样。

3.3 Diffie-Hellman算法

而QUIC则使用了Diffie-Hellman算法(迪菲-赫尔曼算法)来保证数据交互的安全性并合并了它的加密和握手过程来减小连接建立过程中的往返次数，以此来达到0RTT的目的
QUIC 中的加密密钥有两个 initial key 和 forword-secure key。前者用于实现 0-RTT 的握手，后者则用于握手成功以后整个会话的数据加密；

我们知道 Diffe-Hellman 算法用于生成网络通信两端（假设为 Alice 和 Bob）的加密密钥 K。
使用该算法的前提：

```
Alice 和 Bob 都知道两个素数（g、p）的存在
Alice 有 a（private key）；Bob 有 b（private key）
```

有了上面的前提，Diffe-Hellman 的算法流程如下：

```
Alice 计算 A=g^a mod p，并发送 A 给 Bob；
Bob 计算 B=g^b mod p，并发送 B 给 Alice；
此时，Alice 计算 B^a mod p，Bob 计算 A^b mod p，

分解可得
B^a mod p = (g^b mod p)^a mod p = g^ab mod p = K；
A^b mod p= (g^a mod p)^b mod p = g^ab mod p = K；
```

于是，双方都有了一个共享密钥 K。

参考：

解读HTTP/2与HTTP/3 的新特性https://mp.weixin.qq.com/s/t0QTQr0kpjly0Cat2q3TLw

浅析HTTP/2的多路复用https://segmentfault.com/a/1190000011172823

跟坚哥学QUIC系列：连接迁移https://my.oschina.net/u/4703596/blog/4769057

详解QUIC加密握手共享密钥的生成过程https://blog.csdn.net/chuanglan/article/details/85106706