HTTPS如何优化

HTTPS比HTTP多了TLS协议握手过程（非对称加密交换对称加密密钥），最长花费2RTT

分析性能损耗：

产生性能损耗的环节：1.TLS握手（主要）2.握手后的对称加密报文传输（损耗很小，主流对称加密算法AES、ChaCha20性能不错，有CPU厂商做了硬件级别的优化）

TLS握手产生损耗的环节：

1.客户端和服务端临时生成椭圆曲线公私钥；

2.客户端验证证书时访问CA

3.双方计算对称加密密钥

硬件优化：HTTPS是计算密集型，不是I/O密集型，需要升级CPU，可以选择支持AES-NI的CPU

软件优化：一种是软件升级、一种是协议优化

软件升级：如Linux内核从2.x升级到4.x

协议优化：

密钥交换算法优化：使用ECDHE代替RSA（2RTT->1RTT），ECDHE选择x25519曲线

TLS升级：TLS1.2升级到1.3，TLS1.3把Hello和公钥交换合并了，只需要1RTT就能完成TLS握手，只支持ECDHE

证书优化：

两个方向：

证书传输（使用椭圆曲线（ECDSA）证书而不是RSA证书）

证书验证：

CRL（Certificate Revocation List），由CA定期更新，撤销的证书列表，实时性差，随着吊销证书增加列表会越来越大。

OCSP（Online Certificate Status Protoco）在线证书状态协议，向CA发送查询请求，由CA返回证书的有效状态

OCSP Staping：服务器向CA周期性查询OCSP，获得一个带签名的响应缓存，把这个响应结果发给客户端证明证书，避免了服务器的查询

会话复用：将对称加密密钥缓存，下次直接复用

Session ID（1RTT）：首次建立TLS握手连接后双方在内存缓存会话密钥，用唯一的Session ID标识（定期失效）缺点：随着缓存增加服务器内存压力大；现在网站服务一般由多台服务器负载均衡提供服务，客户端不一定能命中上次访问的服务器

Session Ticket（1RTT）：服务器不再缓存客户端的会话密钥，把缓存工作交给客户端，类似HTTP的Cookie。首次建立连接服务器加密会话密钥作为Ticket发给客户端缓存，客户端再次连接时发送Ticket，服务器解密后获取上次的会话密钥。缺点：集群服务器要保证每台的加密会话密钥都一致；不具备前向安全性；应对重放攻击困难

Pree-shared Key：0RTT，把Session ID 或 Ticket 和HTTP请求一同发送，也有重放攻击问题

重放攻击：中间人截取身份证明后获得访问权限

总结：

硬件方向：选择支持AES-NI的CPU

协议方向：密钥交换算法选择ECDHE、TLS1.2升级到1.3（只需1RTT）

证书方向：选择ECDSA证书（密钥长度更短），服务器开启OCSP Stapling功能

会话复用：Session ID，Session Ticket， Pre-shared Ket，都存在重放攻击风险，不具备前向安全性，会话密钥应设置合理的过期时间