

## 28 | 连接太慢该怎么办：HTTPS的优化

2019-07-31 Chrono

《透视HTTP协议》

课程介绍 >



讲述：Chrono

时长 11:19 大小 12.96M



你可能或多或少听别人说过，“HTTPS 的连接很慢”。那么“慢”的原因是什么呢？

通过前两讲的学习，你可以看到，HTTPS 连接大致上可以划分为两个部分，第一个是建立连接时的**非对称加密握手**，第二个是握手后的**对称加密报文传输**。

由于目前流行的 AES、ChaCha20 性能都很好，还有硬件优化，报文传输的性能损耗可以说是非常地小，小到几乎可以忽略不计了。所以，通常所说的“HTTPS 连接慢”指的就是刚开始建立连接的那段时间。

领资料

在 TCP 建连之后，正式数据传输之前，HTTPS 比 HTTP 增加了一个 TLS 握手的步骤，这个步骤最长可以花费两个消息往返，也就是 2-RTT。而且在握手消息的网络耗时之外，还会有其他的一些“隐形”消耗，比如：



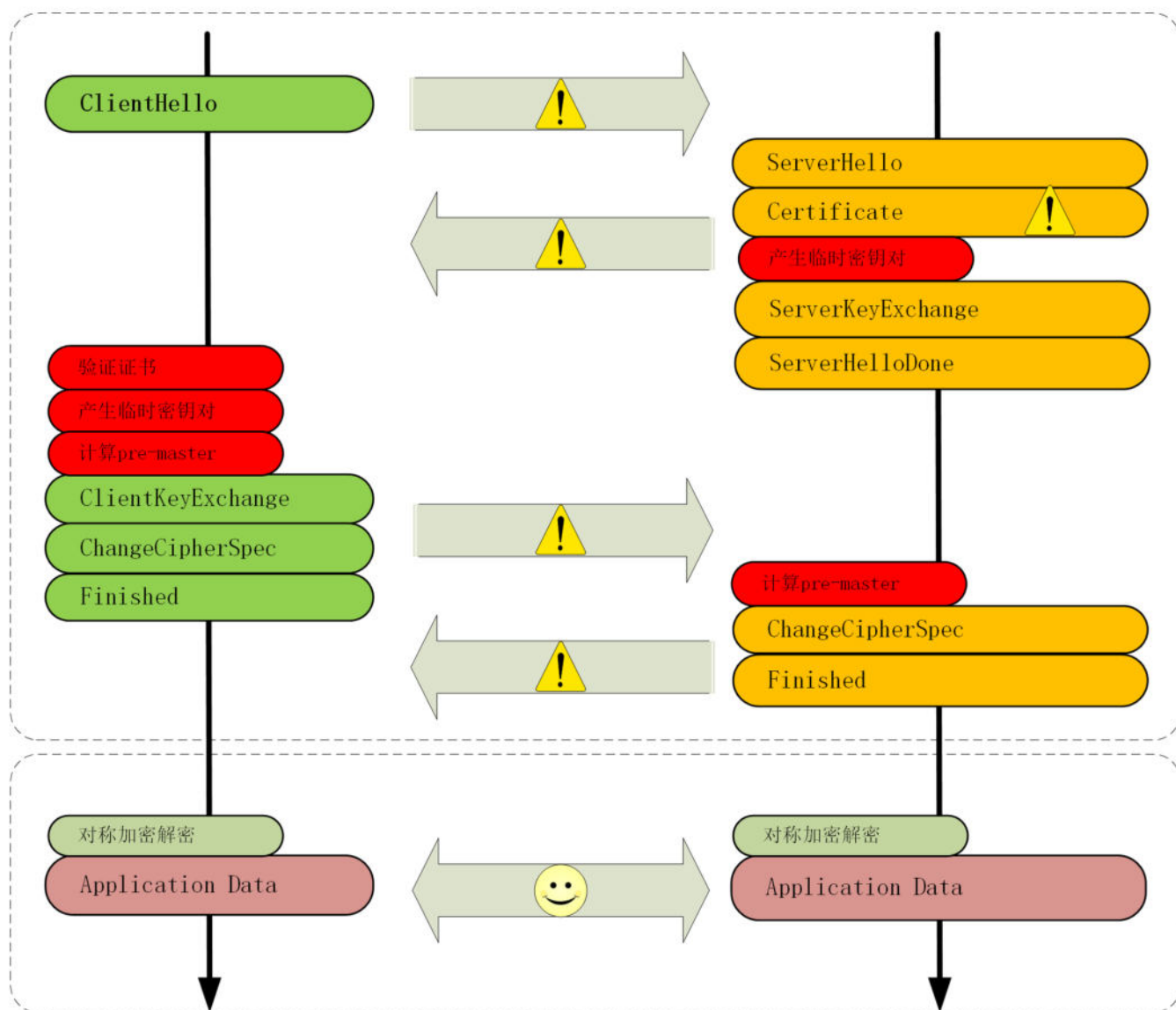
- 产生用于密钥交换的临时公私钥对（ECDHE）；

- 验证证书时访问 CA 获取 CRL 或者 OCSP;
- 非对称加密解密处理“Pre-Master”。

在最差的情况下，也就是不做任何的优化措施，HTTPS 建立连接可能会比 HTTP 慢上几百毫秒甚至几秒，这其中既有网络耗时，也有计算耗时，就会让人产生“打开一个 HTTPS 网站好慢啊”的感觉。

不过刚才说的情况早就是“过去时”了，现在已经有了很多行之有效的 HTTPS 优化手段，运用得好可以把连接的额外耗时降低到几十毫秒甚至是“零”。

我画了一张图，把 TLS 握手过程中影响性能的部分都标记了出来，对照着它就可以“有的放矢”地来优化 HTTPS。



领资料



## 硬件优化

在计算机世界里的“优化”可以分成“硬件优化”和“软件优化”两种方式，先来看看有哪些硬件的手段。

硬件优化，说白了就是“花钱”。但花钱也是有门道的，要“有钱用在刀刃上”，不能大把的银子撒出去“只听见响”。

HTTPS 连接是计算密集型，而不是 I/O 密集型。所以，如果你花大价钱去买网卡、带宽、SSD 存储就是“南辕北辙”了，起不到优化的效果。

那该用什么样的硬件来做优化呢？

首先，你可以选择**更快的 CPU**，最好还内建 AES 优化，这样即可以加速握手，也可以加速传输。

其次，你可以选择“**SSL 加速卡**”，加解密时调用它的 API，让专门的硬件来做非对称加解密，分担 CPU 的计算压力。

不过“SSL 加速卡”也有一些缺点，比如升级慢、支持算法有限，不能灵活定制解决方案等。

所以，就出现了第三种硬件加速方式：“**SSL 加速服务器**”，用专门的服务器集群来彻底“卸载”TLS 握手时的加密解密计算，性能自然要比单纯的“加速卡”要强大的多。

## 软件优化

不过硬件优化方式中除了 CPU，其他的通常可不是靠简单花钱就能买到的，还要有一些开发适配工作，有一定的实施难度。比如，“加速服务器”中关键的一点是通信必须是“异步”的，不能阻塞应用服务器，否则加速就没有意义了。

所以，软件优化的方式相对来说更可行一些，性价比高，能够“少花钱，多办事”。

软件方面的优化还可以再分成两部分：一个是**软件升级**，一个是**协议优化**。

软件升级实施起来比较简单，就是把现在正在使用的软件尽量升级到最新版本，比如把 Linux 内核由 2.x 升级到 4.x，把 Nginx 由 1.6 升级到 1.16，把 OpenSSL 由 1.0.1 升级到



1.1.0/1.1.1。

由于这些软件在更新版本的时候都会做性能优化、修复错误，只要运维能够主动配合，这种软件优化是最容易做的，也是最容易达成优化效果的。

但对于很多大中型公司来说，硬件升级或软件升级都是个棘手的问题，有成千上万台各种型号的机器遍布各个机房，逐一升级不仅需要大量人手，而且有较高的风险，可能会影响正常的线上服务。

所以，在软硬件升级都不可行的情况下，我们最常用的优化方式就是在现有的环境下挖掘协议自身的潜力。

## 协议优化

从刚才的 TLS 握手图中你可以看到影响性能的一些环节，协议优化就要从这些方面着手，先来看看核心的密钥交换过程。

如果有可能，应当尽量采用 TLS1.3，它大幅度简化了握手的过程，完全握手只要 1-RTT，而且更加安全。

如果暂时不能升级到 1.3，只能用 1.2，那么握手时使用的密钥交换协议应当尽量选用椭圆曲线的 ECDHE 算法。它不仅运算速度快，安全性高，还支持“False Start”，能够把握手的消息往返由 2-RTT 减少到 1-RTT，达到与 TLS1.3 类似的效果。

另外，椭圆曲线也要选择高性能的曲线，最好是 x25519，次优选择是 P-256。对称加密算法方面，也可以选用“AES\_128\_GCM”，它能比“AES\_256\_GCM”略快一点点。

在 Nginx 里可以用“ssl\_ciphers”“ssl\_ecdh\_curve”等指令配置服务器使用的密码套件和椭圆曲线，把优先使用的放在前面，例如：

```
1 ssl_ciphers TLS13-AES-256-GCM-SHA384:TLS13-CHACHA20-POLY1305-SHA256:EECDH+CHACHA20
2 ssl_ecdh_curve X25519:P-256;
```

复制代码

## 证书优化

领资料



除了密钥交换，握手过程中的证书验证也是一个比较耗时的操作，服务器需要把自己的证书链全发给客户端，然后客户端接收后再逐一验证。

这里就有两个优化点，一个是**证书传输**，一个是**证书验证**。

服务器的证书可以选择椭圆曲线（ECDSA）证书而不是 RSA 证书，因为 224 位的 ECC 相当于 2048 位的 RSA，所以椭圆曲线证书的“个头”要比 RSA 小很多，即能够节约带宽也能减少客户端的运算量，可谓“一举两得”。

客户端的证书验证其实是个很复杂的操作，除了要公钥解密验证多个证书签名外，因为证书还有可能会被撤销失效，客户端有时还会再去访问 CA，下载 CRL 或者 OCSP 数据，这又会产生 DNS 查询、建立连接、收发数据等一系列网络通信，增加好几个 RTT。

CRL（Certificate revocation list，证书吊销列表）由 CA 定期发布，里面是所有被撤销信任的证书序号，查询这个列表就可以知道证书是否有效。

但 CRL 因为是“定期”发布，就有“时间窗口”的安全隐患，而且随着吊销证书的增多，列表会越来越大，一个 CRL 经常会上 MB。想象一下，每次需要预先下载几 M 的“无用数据”才能连接网站，实用性实在是太低了。

所以，现在 CRL 基本上不用了，取而代之的是 OCSP（在线证书状态协议，Online Certificate Status Protocol），向 CA 发送查询请求，让 CA 返回证书的有效状态。

但 OCSP 也要多出一次网络请求的消耗，而且还依赖于 CA 服务器，如果 CA 服务器很忙，那响应延迟也是等不起的。

于是又出来了一个“补丁”，叫“OCSP Stapling”（OCSP 装订），它可以让服务器预先访问 CA 获取 OCSP 响应，然后在握手时随着证书一起发给客户端，免去了客户端连接 CA 服务器查询的时间。

## 会话复用

到这里，我们已经讨论了四种 HTTPS 优化手段（硬件优化、软件优化、协议优化、证书优化），那么，还有没有其他更好的方式呢？

领资料




我们再回想一下 HTTPS 建立连接的过程：先是 TCP 三次握手，然后是 TLS 一次握手。这后一次握手的重点是算出主密钥“Master Secret”，而主密钥每次连接都要重新计算，未免有点太浪费了，如果能够把“辛辛苦苦”算出来的主密钥缓存一下“重用”，不就可以免去了握手和计算的成本了吗？

这种做法就叫“**会话复用**”（TLS session resumption），和 HTTP Cache 一样，也是提高 HTTPS 性能的“大杀器”，被浏览器和服务端广泛应用。

会话复用分两种，第一种叫“**Session ID**”，就是客户端和服务端首次连接后各自保存一个会话的 ID 号，内存里存储主密钥和其他相关的信息。当客户端再次连接时发一个 ID 过来，服务器就在内存里找，找到就直接用主密钥恢复会话状态，跳过证书验证和密钥交换，只用一个消息往返就可以建立安全通信。

实验环境的端口 441 实现了“Session ID”的会话复用，你可以访问 URI

“<https://www.chrono.com:441/28-1>”，刷新几次，用 Wireshark 抓包看看实际的效果。

 复制代码

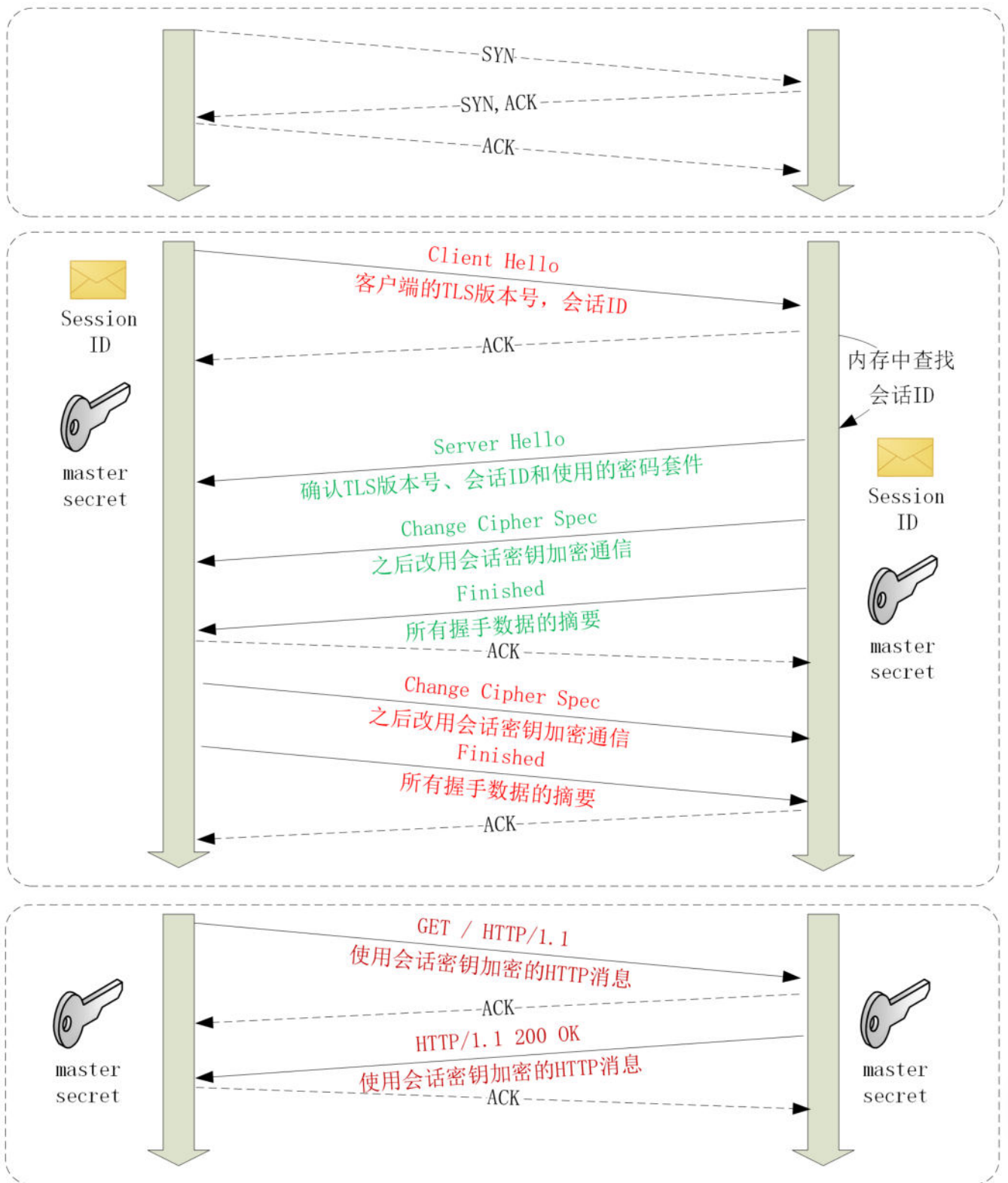
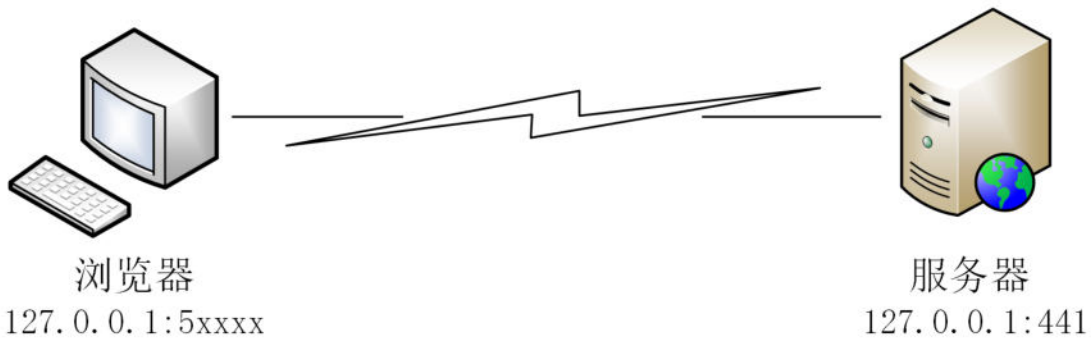
```
1 Handshake Protocol: Client Hello
2   Version: TLS 1.2 (0x0303)
3   Session ID: 13564734eeec0a658830cd...
4   Cipher Suites Length: 34
5
6
7 Handshake Protocol: Server Hello
8   Version: TLS 1.2 (0x0303)
9   Session ID: 13564734eeec0a658830cd...
10  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
```

通过抓包可以看到，服务器在“ServerHello”消息后直接发送了“Change Cipher Spec”和“Finished”消息，复用会话完成了握手。

领资料







领资料



## 会话票证

“Session ID”是最早出现的会话复用技术，也是应用最广的，但它也有缺点，服务器必须保存每一个客户端的会话数据，对于拥有百万、千万级别用户的网站来说存储量就成了大问题，加重了服务器的负担。

于是，又出现了第二种“**Session Ticket**”方案。

它有点类似 HTTP 的 Cookie，存储的责任由服务器转移到了客户端，服务器加密会话信息，用“New Session Ticket”消息发给客户端，让客户端保存。

重连的时候，客户端使用扩展“**session\_ticket**”发送“Ticket”而不是“Session ID”，服务器解密后验证有效期，就可以恢复会话，开始加密通信。

这个过程也可以在实验环境里测试，端口号是 442，URI 是“<https://www.chrono.com:442/28-1>”。

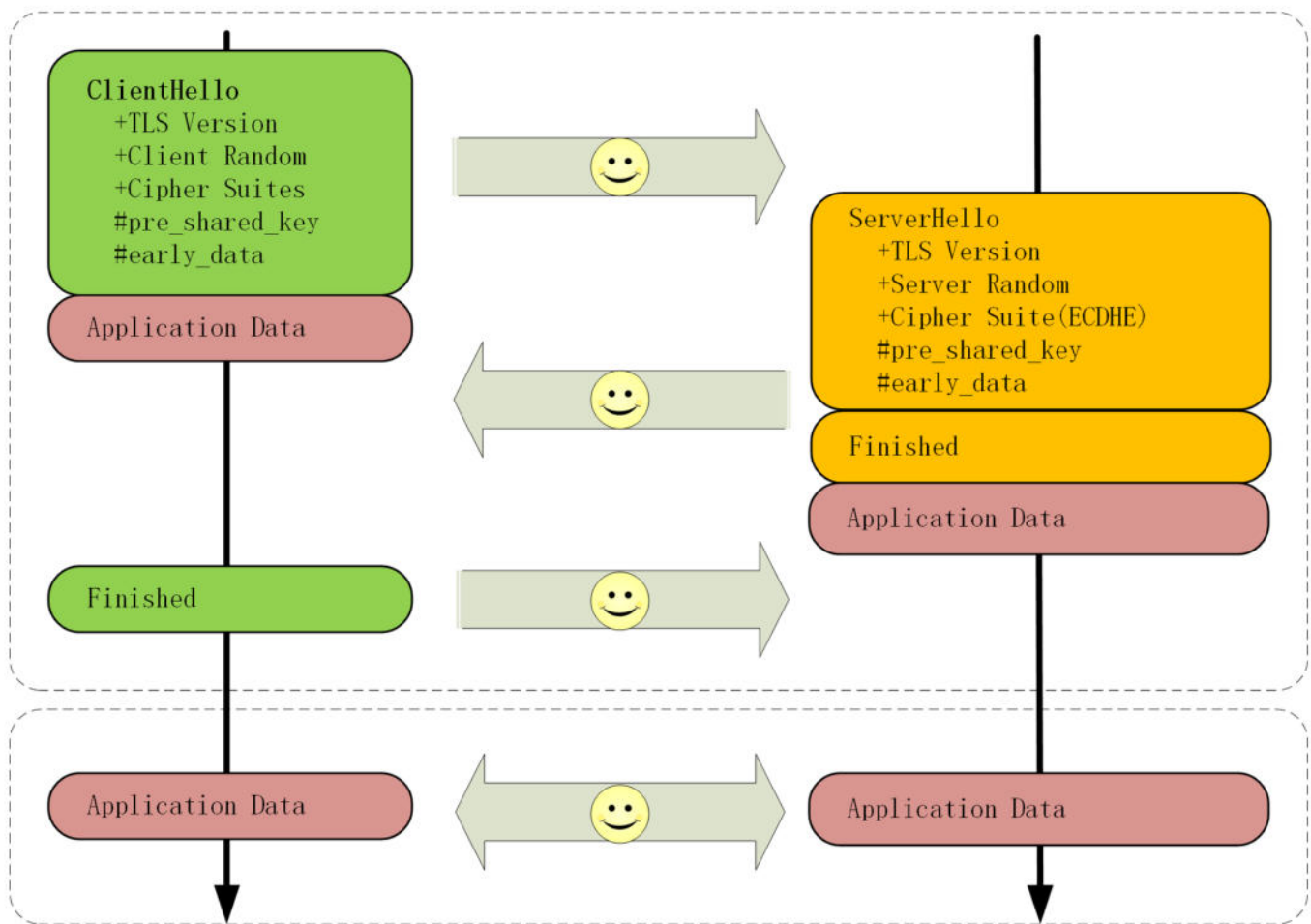
不过“Session Ticket”方案需要使用一个固定的密钥文件（ticket\_key）来加密 Ticket，为了防止密钥被破解，保证“前向安全”，密钥文件需要定期轮换，比如设置为一小时或者一天。

## 预共享密钥

“False Start”“Session ID”“Session Ticket”等方式只能实现 1-RTT，而 TLS1.3 更进一步实现了“**0-RTT**”，原理和“Session Ticket”差不多，但在发送 Ticket 的同时会带上应用数据（Early Data），免去了 1.2 里的服务器确认步骤，这种方式叫“**Pre-shared Key**”，简称为“PSK”。







但“PSK”也不是完美的，它为了追求效率而牺牲了一点安全性，容易受到“重放攻击”（Replay attack）的威胁。黑客可以截获“PSK”的数据，像复读机那样反复向服务器发送。

解决的办法是只允许安全的 GET/HEAD 方法（参见 [第 10 讲](#)），在消息里加入时间戳、“nonce”验证，或者“一次性票证”限制重放。

## 小结

1. 可以有多种硬件和软件手段减少网络耗时和计算耗时，让 HTTPS 变得和 HTTP 一样快，最可行的是软件优化；
2. 应当尽量使用 ECDHE 椭圆曲线密码套件，节约带宽和计算量，还能实现“False Start”；
3. 服务器端应当开启“OCSP Stapling”功能，避免客户端访问 CA 去验证证书；
4. 会话复用的效果类似 Cache，前提是客户端必须之前成功建立连接，后面就可以用“Session ID”“Session Ticket”等凭据跳过密钥交换、证书验证等步骤，直接开始加密通信。

领资料



## 课下作业

1. 你能比较一下“Session ID”“Session Ticket”“PSK”这三种会话复用手段的异同吗？
2. 你觉得哪些优化手段是你在实际工作中能用到的？应该怎样去用？

欢迎你把自己的学习体会写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。



## == 课外小贴士 ==

- 01 使用“SSL 加速卡”的一个案例是阿里的 Tengine，它基于 Intel QAT 加速卡，定制了 Nginx 和 OpenSSL。
- 02 因为 OCSP 会增加额外的网络连接成本，所以 Chrome 等浏览器的策略是只对 EV 证书使用 OCSP 检查有效性，普通网站使用 DV、OV 证书省略了这个操作，就会略微快一点。
- 03 在 Nginx 里可以用指令“ssl\_stapling on”开启“OCSP Stapling”，而在 OpenResty 里更可以编写 Lua 代码灵活定制。

领资料




04 “Session ID”和“Session Ticket”这两种会话

04 Session ID 和 Session Ticket 这两种会话复用技术在 TLS1.3 中均已经被废除，只能使用 PSK 实现会话复用。

05 常见的对信息安全系统的攻击手段有重放攻击 (Replay attack) 和中间人攻击 (Man-in-the-middle attack)，还有一种叫社会工程学 (Social engineering attack)，它不属于计算机科学或密码学，而是利用了“人性的弱点”。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 9  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 27 | 更好更快的握手：TLS1.3特性解析

下一篇 29 | 我应该迁移到HTTPS吗？

领资料



# JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



## 精选留言 (26)

写留言



-W.LI-

2019-07-31

Session ID:会话复用压力在服务端

Session Ticket:压力在客户端，客户端不安全所以要频繁换密钥文件

PSK:验证阶段把数据也带上，少一次请求

前两个都是缓存复用的思想，重用之前计算好的结果，达到降低CPU的目的。第三个就是少一次链接减少网络开销。

感觉都可以把开销大的东西缓存起来复用，缓存真个好东西，空间局部命中和时间局部命中定理太牛逼了。不过最关键的还是找性能瓶颈，精确定位性能瓶颈比较重要，然后针对瓶颈优化，空间换时间或者时间换空间。这个时候算法的价值就体现出来了。可惜这些我都不会

作者回复: psk实际上是Session Ticket的强化版，本身也是缓存，但它简化了Session Ticket的协商过程，省掉了一次RTT。

多在实践中学，多看些开源项目，就能逐渐掌握了。

共 2 条评论 >

13



Fstar

2019-08-01

领资料



1. 你能比较一下“Session ID”“Session Ticket”“PSK”这三种会话复用手段的异同吗？

答：

(1) Session ID 类似网站开发中用来验证用户的 cookie，服务器会保存 Session ID对应的主密钥，需要用到服务器的存储空间。

(2) Session Ticket 貌似有点类似网站开发中的 JWT (JSON Web Token)，JWT的做法是服务器将必要的信息（主密钥和过期时间）加上密钥进行 HMAC 加密，然后将生成的密文和原文相连得到 JWT 字符串，交给客户端。当客户端发送 JWT 给服务端后，服务器会取出其中的原文和自己的密钥进行 HMAC 运算，如果得到的结果和 JWT 中的密文一样，就说明是服务端颁发的 JWT，服务器就会认为 JWT 存储的主密钥和有效时间是有效的。另外，JWT 中不应该存放用户的敏感信息，明文部分任何人可见（不知道 Session Ticket 的实现是不是也是这样？）

(3) PSK 不是很懂，貌似是在 tcp 握手的时候，就直接给出了 Ticket（可是这样 Ticket 好像没有加密呢）。

总的来说，Session ID 需要服务器来存储会话；而 Session Ticket 则不需要服务器使用存储空间，但要保护好密钥。另外为了做到“前向安全”，需要经常更换密钥。PSK相比 Session Ticket，直接在第一次握手时，就将 ticket 发送过去了，减少了握手次数。

作者回复: 说的挺好，PSK其实就是Session Ticket的强化版，也有ticket，但应用数据随ticket一起发给服务器。



👍 5



俊伟

2020-01-17

session id 把会话信息存在服务端

session ticket 把会话信息存在客户端

psk 在session ticket 的基础上，使用early data顺便再发送一下服务端的数据

作者回复: 态度认真、积极，good。



👍 5

领资料



Wr

2020-01-14

1、相同点：都是会话复用技术

区别：

Session ID：会话数据缓存在服务端，如果服务器客户量大，对服务器会造成很大压力

Session Ticket：会话数据缓存在客户端

PSK：在Session Ticket的基础上，应用数据和Session Ticket一起发送给服务器，省去了中



## 2、暂无

作者回复: 居然是三连.....



👍 3



饭粒

2020-05-05

1.抓包看了下 442 , 复用会话时 Client Hello session\_ticket 确实有数据, Session Id 好像也还会复用。

Transport Layer Security

TLSv1.2 Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 512

Handshake Protocol: Client Hello

...

Random: 9474888cafdce89fd32eac247a8b464f842efbac706d8930...

Session ID Length: 32

Session ID: a4a0caef10dee7a6f44aa522a35f6c799101d5eced01eb32... # Sessi

on ID

...

Extension: session\_ticket (len=192) # session\_ticket

Type: session\_ticket (35)

Length: 192

Data (192 bytes)

...

Transport Layer Security

TLSv1.2 Record Layer: Handshake Protocol: Server Hello

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 100

Handshake Protocol: Server Hello

...

Random: e82519e9e8bfcbd40e1da7a202bd50ff993d5ef0cbc33378...

Session ID Length: 32

Session ID: a4a0caef10dee7a6f44aa522a35f6c799101d5eced01eb32... # Sessi

on ID

领资料





...

TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec

TLSv1.2 Record Layer: Handshake Protocol: Finished

2.有个疑问：会话复用技术，保存会话数据的一端使用对端传过来的 Session Id 查询到之前的 master secret，但它如何安全的把这个 master secret 传递给对端（对端应该只有 Session Id 吧）？

3.抓包过程用 Chrome 发请求，前三次 Client Hello, Server Hello 握手过程都因为 Alert (Level: Fatal, Description: Certificate Unknown) 失败了，第四次才成功。而使用 Firefox 发请求则不会出现。这是因为 Chrome 内置的证书更少吗？

作者回复：

1.有ticket就不会用id。

2.master secret都保存在两端各自的内存里，不需要，也不允许在网络上传递，两端用id对一下，一致就行了。

3.实验环境的证书是自签名证书，需要提前信任才行，不然就会告警。这个跟浏览器的安全策略有关，可能Firefox的处理逻辑不一样。

共 2 条评论 >



1



钱

2020-04-04

1：你能比较一下“Session ID”“Session Ticket”“PSK”这三种会话复用手段的异同吗？

1-1：回话复用

核心是缓存主密钥，为啥要缓存？因为计算出主密钥比较费劲，如果能重复利用，重复计算的活就免了，这是在拿空间换时间。

1-2：Session ID

可以认为是缓存主密钥的key，在客户端和服务端都有存储，通过传递这个key来获取和重用主密钥。

缺点是太费服务器的存储空间，因为每一个客户端的回话数据都需要保存，如果客户端有百万甚至千万基本，那存储空间使用的就有些多啦！

1-3：Session ticket

这个方案可以解决服务器存储空间压力山大的问题，核心是把信息放在客户端存储，当然是加密后的信息，服务器侧需要解码后再使用。

1-44：PSK

和Session ticket类似，为了效率加大了一些安全风险，ticket中带上了应用数据的信息，这样能省去服务器的确认步骤。为了加强安全性，使用上做了一些限制。

领资料



有此可见，没有完美的解决方案，具体想要什么需要自己权衡对待。想要实现多快好省，那要么是一句口号，要么必须付出其他的代价。

2：你觉得哪些优化手段是你在实际工作中能用到的？应该怎样去用？

软件优化这个估计最常用，也能用到，一些安全漏洞或性能优化常这么玩。

作者回复: psk理解的稍微有点偏差，ticket里是加密的会话密钥，应用数据在psk后面。



1



lesserror

2019-12-21

老师，以下问题麻烦请回答一下：

1.客户端有时还会再去访问 CA，下载 CRL 或者 OCSP 数据，这又会产生 DNS 查询、建立连接、收发数据等一系列网络通信，增加好几个 RTT。这个CRL 或者 OCSP是对应到某个网址上面的嘛？客户端根据网址访问？

2.它可以让服务器预先访问 CA 获取 OCSP 响应，然后在握手时随着证书一起发给客户端，免去了客户端连接 CA 服务器查询的时间。这里不是客户端自己去验证的会不会有问题？服务器自己代做了。

作者回复:

1.crl和ocsp是一个很大的列表，包含所有过期或者作废的证书序列号，不关联到具体的网址。客户端只要在这个列表里查一下证书序列号是否在这里面就行。而证书里面是包含网址的。

2.ocsp都是经过ca签名的，所以不会被篡改，保证肯定是ca发出的。



2



书生依旧

2019-10-16

PSK 在发送 Ticket 的同时会带上应用数据，免去了 1.2 里面的服务器确认步骤。

这句话有点不太理解，请问老师：

1. 看图上 pre\_shared\_key 是在 Hello 中发送的，Session Ticket 也是在 Hello 中发送的吗？
2. 带上应用数据是什么数据？
3. 1.2 里面的服务器验证指的是哪个步骤？

作者回复:

1.是的，hello消息里有个pre\_share\_key扩展，就是ticket。

2.在https里应用数据就是http报文了。



3.在tls1.2的会话复用里，必须在服务器发送server hello确认建立加密连接之后才能发送应用数据，而tls1.3就不需要这个确认步骤。



1



cake

2021-11-27

老师请问下 非对称加密解密处理“Pre-Master” 这计划的意思就是生成“Pre-Master”的意思么

作者回复: 对，因为交换pre-master需要非对称算法的参与。



路漫漫

2021-11-23

老师，文章里说，这后一次握手的重点是算出主密钥“Master Secret”，而主密钥每次连接都要重新计算，未免有点太浪费了。难道每次连接的主密钥是一样的？不是每次连接的ecdhe的公钥私钥都是不同的吗？怎么还可以缓存呢？

作者回复: ecdhe的公钥私钥每次都是临时生成的，但它只是用在握手过程中。

而master secret是用在会话过程中的，由client random&#47;server random&#47;pre-master生成的，而这三个都是随机数，所以master secret必然每次都不同。

但算master secret需要握手，成本高，所以为了提高效率，就可以用session id、session ticket来复用，并不是简单的缓存。



Geek\_5227ac

2021-03-24

罗老师，本节最后提到PSK有遭重放隐患，但为什么TLS开始握手阶段的明文传输没有重放危险呢，是因为有携带应用数据才谈重放吗？另外HTTPS是先有tcp连接了再来TLS握手，那有没有可能黑客通过不断地先进行TCP连接再TLS握手来很快消耗掉服务器提供正常服务的连接资源？

作者回复:

1.握手时每次都是随机生成的密钥，不重复，所以不会有重放的危险。而psk直接放行，就容易被冒充。

领资料



2.这个就是dos攻击了。



尿布

2020-09-10

服务器端应当开启“OCSP Stapling”功能，避免客户端访问CA去验证证书

在Nginx里可以用指令“ssl\_stapling on”开启“OCSP Stapling”，而在OpenResty里更可以编写Lua代码灵活定制



Geek\_e4a9c5

2020-08-26

接之前提的问题，可是session id和session ticket不也是一段时间内固定的吗，psk只是加上了https报文信息，还是不太理解“更容易”被攻击的原因，万分感谢老师解答。

作者回复: session id的数据在内存里，所以不会有危险。

session ticket同样会受到重放攻击，本质上和psk是一样的，所以需要一些手段来防止。

正文里没有说太清楚，可能造成了解，sorry。



Geek\_e4a9c5

2020-08-25

请问PSK 为什么容易受到重放攻击呢

作者回复: 因为PSK每次是固定的，这就容易被黑客截取，然后伪装成客户发送给服务器，如果没有鉴别手段就会被冒充。



领资料



Joker

2020-04-14

特来请教老师，第二个问题，有什么开源的项目里面有实现了这些吗？

作者回复: 现在流行的Apache、Nginx都支持，因为这都是TLS协议标准规定的，可以参考它们的文档。





**Amberlo**

2020-04-07

老师好，关于 证书优化，获取CRL列表的时候会不会存在中间人攻击呢

作者回复: crl有ca的签名，不会被伪造，所以是安全的。

不过如果有恶意ca，恶意签发假证书、假crl就是另外一回事了。



**qzmone**

2020-01-16

老师，<https://www.chrono.com:441/28-1> 这个我抓包看到client和server的session-ID不一样，而且每次也都是server发了变更密码规范消息后才发加密的数据，跟您说的不一致？不知什么原因

作者回复: 第一次握手肯定是没有会话复用的，到第二次就会会话复用。

第一次后实验环境在浏览器会显示出“reused? false”，这就是没有复用。然后wireshark重新开始抓包，刷新一下页面，会显示为“reused? true”，这个时候再看抓包。

我又做了一次试验，是可以的，你再操作看看。



**Wr**

2020-01-14

1、相同点：都是会话复用技术

区别：

Seesion ID：会话数据缓存在服务端，如果服务器客户量大，对服务器会造成很大压力

Seeion Ticket：会话数据缓存在客户端

PAK：在Seesion Ticket的基础上，应用数据和Session Ticket一起发送给服务器，省去了中间服务器与客户端的确认步骤

2、暂无

作者回复: good。

领资料





Wr

2020-01-14

1、相同点：都是会话复用技术

区别：

Session ID：会话数据缓存在服务端，如果服务器客户量大，对服务器会造成很大压力

Session Ticket：会话数据缓存在客户端

PAK：在Session Ticket的基础上，应用数据和Session Ticket一起发送给服务器，省去了中间服务器与客户端的确认步骤

2、暂无

作者回复: 好像二连了.....



Luke

2019-08-30

老是，如果使用服务器集群来做专门的加解密运算，建立TLS链接时，客户端将数据发送给服务器集群计算密钥，服务端又是如何安全的将密钥返回？或者是在解密报文时，又如何将解密后的报文安全返回？

作者回复: 由于都是内网，所以就不存在外网那么多的威胁，所以可以直接通信，无需其他安全手段。

共 2 条评论 >



领资料

