1、 QUIC 的提出

SPDY 是个目前基于 TCP(经常使用 SSL)实现的多路复用流协议。此外，它可以通过尽可能快地 (而不是等待前面的确认返回)发送所有请求来减少延迟，并且可以通过压缩一些冗余流量来减少带宽使用。 尽管其特性和成功， 当提供一个延迟减少时， 它在请求有效地利用资源方面遇到了一些问题。

a) 单个包延迟导致一个流的头阻塞

b) 由 TCP 处理、导致额外带宽减少和序列化延迟开销的不适宜的拥塞避免

c) TLS(SSL) 会话恢复延迟

d) TLS 往往引发一个解密依赖，先前的包必须在后来的包可以被解密之前被解密

我们希望减少整个英特网的延迟， 提供一个响应性更好的用户交互环境。 随着时间的推移，整个世界的带宽将会提升， 但是受光速支配的往返时间不会减少。 我们需要一个协议用更少的延迟和更少的重传时间消耗去传递整个互联网的请求、 响应和交互， 并且， 我们相信现今的方法在阻碍我们。这部分指出我们希望解决的潜在问题。

我们想要开发一个支持以下目标的传输： 减少因包丢失造成的头阻塞， 低延迟， 隐私保证堪比 TLS，等等。

现今， 可行性的头号目标显然是这个协议发展的主要驱动力。 中间盒和防火墙会代表性地阻塞或明显地降低基于除了 TCP 或 UDP 的格式的任何传输， 明白这个以后， 我们甚至不会考虑革命性的协议。所以，只有开发基于 TCP 或 UDP 的协议，用来解决我们遇到的问题以及实现我们的目标。

由于基于 DTLS(数据包传输层安全)的 SCTP 在建立连接时需要的延时太长，大约为 4 个 RTT，所以 SCTP 是不合适的。因此，我们开发了基于 UDP 的 QUIC 协议。

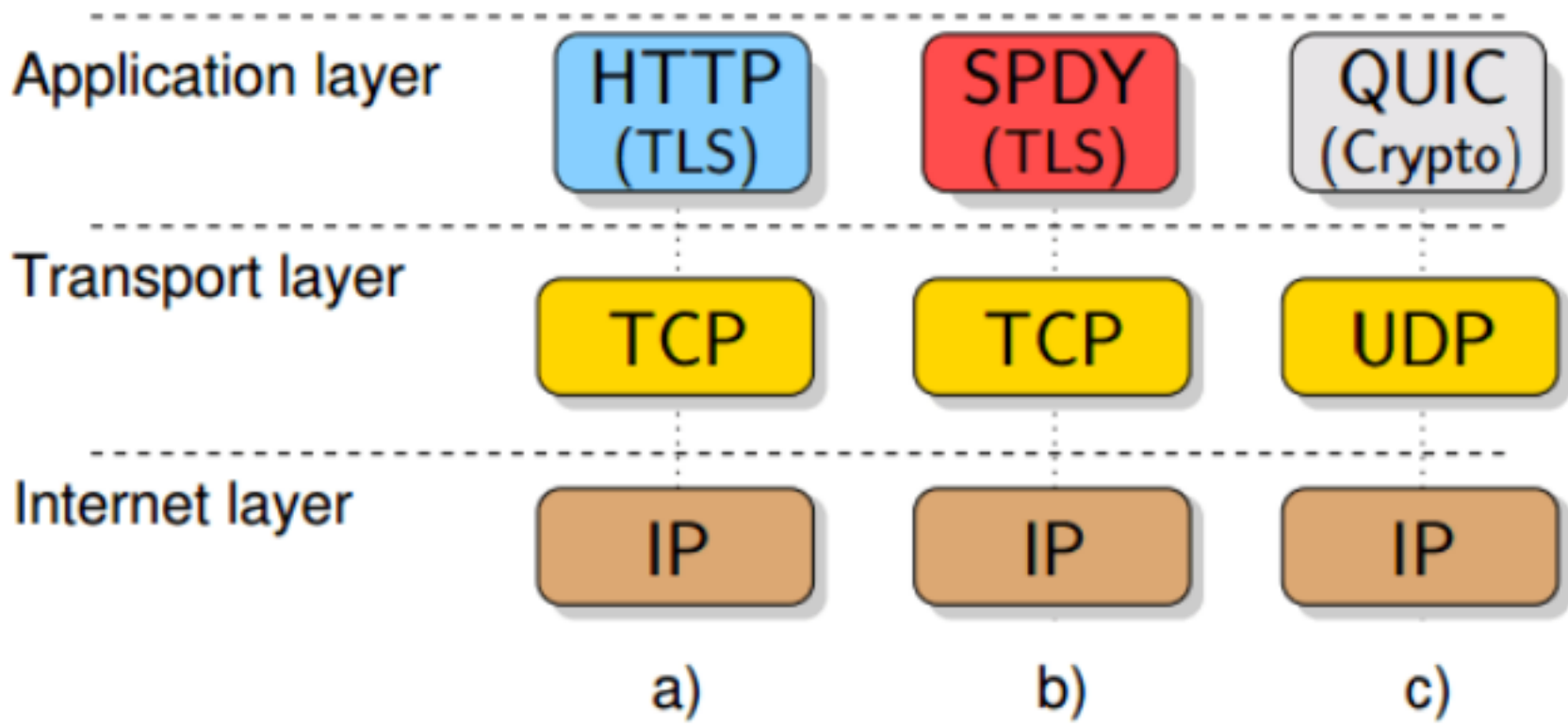
2、 QUIC 协议的层次

可以认为 QUIC 是为了解决 SPDY 在 TCP 遇到的瓶颈而在 UDP 上做探索所设计的方案。参考 SPDY 来理解，可认为 QUIC 的传输内容分两层，高层类似 SPDY，低层是在 UDP 上模仿实现 TCP 的面向连接特性和可靠性并加入类似 TLS 的加密过程。

QUIC 提供基于 UDP 的多路复用、有序、 可靠的流传输。 QUIC 是和 HTTP 同一层的应用层协议，其核心是将丢包控制工作转移到应用层。

这是一个对小组讨论和编辑来说是可行的文档，我们希望它发展成一个充实的设计文档。希望设计成一个运行在 UDP 上的可以在两个终端 (一个初始化整个连接的客户端和一个服务器端)上多路复用大量流的隧道协议。例如，每个流几乎相当于一个独立的 TCP 连接。

最终的协议可能非常像运行在 UDP 上的 SCTP，在使用加密上非常像 DTLS 。



3、流

流：在一个连接中传递数据的潜在的许多数据传输信道中的一个。 一个流是双向的。 如果流被客户端首先创建 (连接发起人)，那么它将有一个奇数编号的流标识符。如果流被服务器创建 (连接应答者)，那么它将有一个偶数编号的流标识符。 在流中的数据自动被分解成帧，然后在接收端重新组装。

在为一个多路复用流建立一个 API 时有一些复杂性需要解决。在最高层次上，我们需要有一个机制增加新流到一个连接中，以及分别独立地读和写不同的流。

对每个流，我们需要一个方法来访问流，指定流应该使用的特性。特性包括，例如，可靠性和性能权衡 (例如通过增加冗余减少抖动，通过减少冗余来减少带宽)。

我们期望不同的流将有明显的传输特征， 它们或许会被应用设置或修改。 这些包括明显的特征设置：

* 可调冗余水平 (对延迟储蓄的贸易带宽)
* 可调优先级水平 (仿照 SPDY 的演变优先级方案)

我们期望一些或许被视为输出流的控制通道将总是有用的， 且可能被用于标识剩下流的状态改变。

对加密协商来说，控制通道可能包含特殊目的帧 (控制帧)和一个保留的流。

在 QUIC 连接中的每个流将有一个独特的相关联的流标识符。

基于数据传输的字节流将仿照 TCP，具有有效负载数据提供的字节范围。字节范围将选择每个字节流中的定位。

流将被划分成帧放入 (UDP) 包中。只要有可能，任何特定的数据包的有效载荷应只来自

于一个流。 这将减少这种概率， 一个包丢失将阻碍不止一个流的进步。 当没有足够的数据流来填充一个数据包， 然后来自不止一个流的帧可能被打包进一个包。 这种包装应减少数据包计数开销，减少序列化延迟。

4、 QUIC 包格式

传输的基本单元将是一个标准的 UDP 包(又名，包)。注意确保所有的数据传输将会被分解成刚好放入一个包的块。

包括用来协商一个连接的第一包在内的所有包，将利用 AEAD 加密。第一包将利用一个默认的空加密密钥， 并且 AEAD 将只是用来排除意外干预 (作为一个高质量的校验和)。加密协商将发生在流 1，由客户端产生。

header	payload

QUIC 包

QUIC 包由头 (header)和有效载荷 (payload)组成。其中， payload 是 AEAD 算法认证的密文。

每个包的头包括：

*1 字节的公共标识，详述头的剩下部分的设计

* 全局唯一标识符

*QUIC 版本

* 包序列号

公共标识	全局唯一标识符	QUIC 版本	包序列号

QUIC 包的头部

数据包有效载荷由一系列帧组成：

头	一系列帧

FEC 包有效载荷由冗余数据组成：

头	冗余数据

在解密之后，我们将有一个明文有效载荷块，它包括：

*1 字节的私有标识

*FEC 组号

* 一系列自我标识帧

私有标识	FEC 组号
一系列自我标识帧	

解密之后的 QUIC 包的有效载荷

4.1 QUIC 包详解

1. 头：公共标识

在一个给定连接和给定包中，公共标识提供给每个包的其他区域的大小的说明。

2. 头：全局唯一标识符

全局唯一标识符的长度被指定为 64 比特，所以，客户端可以随机地选择一个全局唯一标识符，在一个固定的端口联系一个服务器，有很小的可能性会与其他连接碰撞。

然而，全局唯一标识符对于一个已经创建了一个专用的短暂的联系服务器的端口的客户端来说完全是冗余的。客户端将在那个端口收到的包将会是单一的 QUIC 出口连接的一部分。结果， 客户端可能请求，一个服务器不必费心的包括每个包的全局唯一标识符。一旦一个服务器收到一个请求 (例如在连接协商期间)，服务器可以使用公共标识表明，全局唯一标识符被省略 (在头中的长度为 0)。

对一些服务器和服务来说， 并行连接关系的编号是非常有限的。 例如， 一个服务器可能和一个客户端协商在一个特别的可选的 IP 和端口上继续那个连接。在那种情况下，服务器也可能对客户端表明，一个更小的全局唯一标识符或许是可接受的。

3. 头： **QUIC** 版本

我们知道， 协议发展的很快，并且需要发展。 这个区域出现在第一包中，用来确保服务器可以理解客户端将提供的相同的版本。 一旦连接建立， 这个是冗余的， 并且公共标识将会表明这个区域被省略了 (长度为 0)。如果一个服务器需要区别版本，它将记住，被全局唯一标识符定义的连接有一个与众不同的版本。

4. 头：包序列号

除了给包定序、留意副本、交流什么包丢失了，这个编号是加密的一个关键组成部分。这个编号组成了用来解密每个包的初始化向量的基础。结果，从概念上讲，它必须是大的，因为在连接期间，它必须绝不重复。那个需求强迫这个序列号概念上是大的，大约 2^64(比连接还要多的包等着发送)，但是我们通常不需要提供每个包的 8 个字节！

在任何给定的时间， 仅有一些有限的包序列号没有被确认。 这个限制是由这个事实的自然结果， 发送者必须缓存那些未决定的包中的内部数据， 并且发送者的内存是有限的。 另外，我们选择不去重传被声明为丢失的包， 而是把他们的内容放入后来的数据包中。 结果， 接收者常常通知，它没有收到一个包，并且发送者将通知接收者“ 停止等待 ”那个包，因此没有被确认的包的窗口将总是恰好有界的，并且不确定性 (讨论的可能的最大和最小值)范围可能被发送者知道。给予那个限制，发送者可能大大地减少需要表示包序列号 (使用公共标识)的字节数。

例如，假设正在用一个像拥塞控制的 TCP 传输，并且目前的拥塞窗口是 20 个包。即使包丢失，在 1 个 RTT 或大约 20 个额外包内，接收者将知道一个丢失的包不再是未判定的。结果，发送者可以侥幸逃脱在线路上仅发送 64 比特包序列号的低序号字节。接收者给予那些比特可以轻易地决定较高的 56 比特必须是什么，可以使用那个去解密包。注意，如果一个非常老的包到达接收方，并且老的包序列号是被曲解的，那么 AEAD 认证会失败，并且老的 (并且适合丢弃的)包甚至会被忽略。

5. 有效载荷：私有标识

目前 1 字节大小的私有标识表示 “ 私有 ”， 因为他们被加密覆盖，并且对窃听者是不可见的。与公共标识一样，在标识中编码的一个值是 FEC 组号的大小，和有效载荷到帧的开始的隐式偏移。

私有标识还有另外 2 个独特的比特。 第一个比特是一个随机加密比特, 第二个比特用来鉴定 FEC 组中的最后一个包。 FEC 组中的最后一个包是冗余 FEC 包(包含组中的明文有效载荷的异或)。

私有标识：加密比特

加密比特由发送者随机选择, 并且放入每一个包中。 这个信息用来对抗任何潜在的乐观确认攻击。 当一个接收器发送一系列包的确认时, 它需要证明它收到了那个集合, 通过提供它宣称已经看见的加密比特哈希表宣称的那个集合。

加密比特的一个问题是, 当一个包丢失时, 它的加密比特是不可见的。 为了解决这个问题，并且不需要接收器创建无止尽的丢失包列表 (来自哈希表的异常)，发送者常常提供一个更新，表明哈希表相对于一个特定的数据包来说应该是什么。

例如，假设包 1，2，3，4 被发送了。 假设包 3 丢失了。 接收器将表明 (在一个确认帧里)，它收到了从 1 到 4 的所有包，除了包 3。确认帧也会提供一个打包 1，2，3，4 包的加密比特的哈希表。发送器可以确认哈希表是正确的，并且声明包 3 丢失了。 然后，发送器可以传输(包括一个确认帧)它不再关注包 4 或者更老的包, 并且它可以提供到包 4 为止的所有加密比特哈希表。 结果, 接收器终于知道到包 4 为止的包的哈希表, 并且当它收到下一步的数据包时可以提供额外的增值哈希表结果。

私有标识： FEC 最后的比特

这个比特用来标记最后一个包是一个 FEC 组。我们现在的 FEC 方法在一系列受 FEC 保护的包的末尾恰好有一个 FEC 包。这个比特表明，整个有效载荷应该单独处理，并且被视作在这个组中的其他有效载荷的异或和。

注意，当这个比特被设置时，包必须是一些 FEC 组的一部分，因此 FEC 组号也必须总是存在的。

6. 有效载荷： **FEC** 组号

假定，我们大约看见在互联网上 1%以上的包丢失, 100 以上且不是 200 个包都无损失的被接收是完全不可能的。因为这个原因，我们决定限制 FEC 组不大于 255 个包。我们的实验暗示，当 FEC 丢失时，一个 10 到 20 的受一个冗余 FEC 包保护的包数据序列或许是最有益的。每 20 个包，有大约 5%的带宽消耗，并且有一个完全可见的延迟权衡。我们还注意到，对 TCP 来说，看见拥塞窗口正好低于 50 个包是最常见的。因此，和重传相比，对于更大的组使用一个 FEC 无益于减少延迟。

因为以上分析，我们目前支持一系列连续的包的简单异或 FEC，并且一字节足以鉴定一组受保护的包。 如果发送器决定保护一系列包, 它使用私有标识去表明, 确实有一个嵌入式的 FEC 组号字节。 每个 FEC 组的参与者有一个偏移 FEC 组号，它可以鉴定组中的第一个包(即，它是减去目前包序列号的一个总数)。这种方法考虑到就什么时候结束那个 FEC 组的一个懒惰的决定 (即，包的准确编号不需要知道，最后的 FEC 包可以在任何时间被加入，例如当没有数据发送时)。

7. 有效载荷：自我标识帧

每个 QUIC 包的块有望成为一个称为帧的数据的串行列表。 每个帧都有标识帧的主要字节，和关于帧格式与内容的信息。 例如，有些携带确认信息的帧， 也有携带纯粹流数据的帧。还有一些其他的帧。

包通常被包装成充满一个或者更多帧的数据的包然后发送出去。 帧在某种意义上来说是协议的载体 .

所有的帧都以一个帧类型字节开始， 但是我们希望在那个字节中包装一些额外的关于具体的每种类型的标识。 结果，它实际上是用于鉴定帧类型的一个帧类型字节的开始几个比特，其他几个比特用来作为编码帧格式的标识。 QUIC 的初始实现版本可能使用非常固定的大小区域表示一些帧类型，但是它应该随着我们优化和压缩那个编码而改变。

帧分为：流帧，确认帧，拥塞控制帧，重置流帧，连接关闭帧，离开流帧，

流帧

每个 QUIC 连接可以多路复用一些流， 每个流帧承载一个流的应用数据。 帧的头在概念上讲必须体现流编号， 流编号由帧和独立数据 (就像 TCP 为段提供字节偏移一样)的流内开始偏移组成。 一个流帧还用来隐式地打开或创建一个流， 帧目前包含一个表明它是上述流的最后一部分的标志比特。

当一个大文件传输在一个流上时效率可能是至关重要的， 为了提高效率， 我们希望在这个帧的头部利用不同长度的区域。 例如， 即使我们希望支持非常大的文件传输， 但是一个流使用许多对现在来说是最大偏移的 64 比特来传输数据这种情况还是很少发生的。同样的，当大的流被传输时， 用一个有效载荷填充所有剩下的空间对一个数据流来说是很常见的， 因此，它的大小 (在有效载荷之内)可以是隐式的而不是显式地。

正如 SPDY，我们希望压缩 HTTP 请求流的头部。 这应该在连接建立和加密握手期间协商。这个方法的一个代价就是流头部必须按序编码， 因此， 一个新的流必须在前面的流的头部被编码后才能处理。我们期望，在许多情况下， HTTP 头的压缩效果是非常显著的，以至于大量的 HTTP 请求被很好地填充进一个 QUIC 包，并且这个头阻塞将是无关紧要的。 我们同样期望， HTTP/2.0 将解决一个有更大提高的有更好的序列化编码控制的压缩算法，并且能够在 QUIC 中被使用。

确认帧

确认帧用来调整包丢失恢复和其他一些类似作用，但是和 TCP 中的一个确认包不是完全相同的。 QUIC 中的确认总是累积的，新的确认包含的信息表明任何先前的确认都应该被丢弃。结果，如果包含一个确认帧的一个包丢失了，不需要重传带有确认的帧。

第一个且最清晰的概念性内容， 一个帧包含的确认是一个接收器已经决定为遗失的包的否定确认列表。 结果， 一个确认帧有一个丢失包数量的计数和包列表。 就像在 QUIC 头中呈现的包序列号，丢失的包可以很简洁的表示。即使有一个简洁的表示，有一个关于多少 (数百) 否定确认可以在一个包内被枚举的限制。结果，在非常不可能的情况下，有太多否定确认以至于不能填充进一个包，仅仅最早的 (数字上最低的)一些丢失包被枚举了。由 1%的包

丢失造成的这种不太可能的情况，可能限制有效的流动窗口大约为 1000 个包 (不是一个很大的限制，但是任何情况下都应该能工作)。我们把这种情况视为一个“ 缩短了的确认帧 ”。

第二个呈现在确认帧中概念性元素通常是已经接收了的最大包序列号，并且我们把它视为“ 最大的观测的 ”包。那个结构使一个确认帧类似于一个 TCP 选择性的确认，后者列出了最大的接收到的偏移量和 (如果有的话)一个小的否定确认编位号列表 (多大一对)。

QUIC 的一个并发症是，一个确认帧可以如上述的缩短。在那种确认帧被缩短的情况下，接收器丢失在这个压缩边界的几个包是有可能的。为了恰当的处理那种独特的情况，呈现在确认包中的“ 最大的观察的 ”包可能在否定确认包列表中！

为了使上述的问题更加清晰，考虑一个被发送从 1 到 1000 的数据包的接收器，但是仅仅收到包 1 和包 1000(即，它丢失了从 2 到 999 的所有包)。在那个例子中，我们假设，在一个确认帧中只有 200 个否定确认的空间。结果，一个确认帧列出了 (显式地)从包 2 到包 201 的 200 个否定确认。奇怪的是，“ 最大的接收到的 ”包是包 1000，但是这不可能在确认帧中陈述，因为它将看起来好像包 202 到 999 都被收到！在这个缩短的确认帧情况下，“ 最大的观察到的 ”包会被列为包 201。当发送器用一个不考虑否定确认包 201 或更早包的请求响应这个否定确认列表时，上述情况会自行快速解决。按钮行： 即使在这种情况下，也会向前进步。

确认帧的第二个成分承载与普通情况相反的信息，因为它携带从发送器到接收器的确认信息。 QUIC 不重传包，但是在他们仍然有用时把他们的内容装入一个后来的包中 (即，它将倾向于重传丢失包的流帧，但是不重传丢失包的确认帧)。结果，当发送器通过一个丢失包的否定确定学习时，它需要通知接收器“ 停止否定确认 ”那个包。为了支持这个，一个确认帧也有一个鉴定一个 “ 最小的未确认的 ” 包的区域。这个包序列号鉴定一个点，发送器不想要任何更早的否定确认。

在确认帧中提供的最后的两个元素与用来排除一个积极确认攻击的加密哈希表有关。就像更早提到过的，在有效私有标识中每个包中提供一个加密比特。当接收器发送一个确认帧，潜在地指定知道的包和丢失的包，也会为每一个知道的包发送一个已接收加密哈希表。同样地，当发送器更新最小的否定确认的包编号时，它也提供哈希表，当累积加密到达那个最小的否定确认点时接收器应该用这个哈希表。

通常，只包含一个确认帧的一个包是未确认的 (这反过来会导致一个连续的确认！)。跟着一个只包含一个确认帧的包的一个丢失包一定是确认过的，因为接收器不知道丢失包包含什么。

当发送器收到一个包含否定确认 (特别是以前没有听说过的新否定确认)的确认帧时，它可能提供一个应答的确认帧，要求接收器“ 停止否定确认 ”上述的包。

正如 TCP，当一个有效的时间长度 (和 RTT 相关)过去了，没有收到一个确认，发送器可能重传一个包的内容 (即，即使提供了一个明确的否定确认)。这样的重传将继续使用一个指数的备值，但是就像 TCP 一样，假设当一个包正在重传时，他们没有被拥塞窗口限制 (即，推测的损失允许有重传)。

拥塞控制帧

拥塞控制帧用来传递关于一个特定的拥塞控制算法的信息。 在连接的开始， 加密认证协商也包括适合的拥塞控制算法的协商。 作为一个基线， 所有的实现将支持一个类 TCP 算法，但是由于相应的不同数据的需求，我们期望使用其他的算法。

例如，在一个类 TCP 算法中，一个拥塞控制帧可能包含关于一个包什么时候收到的信息，或者同样地，在收到后多久确认被发送出去。这种反馈统计可能在计算 RTT 时有所帮助，就像 TCP 使用一个 TCP 确认的到达来估计一个 RTT(当假定在一个包的收到和一个相应的确认之间有微不足道的延迟时)。目前一个拥塞反馈帧的类 TCP 实现包括累积的包丢失数。

就一个注重测量逗留延迟的算法来说， 一个拥塞反馈帧可能提供包之间的相对到达时间(允许发送器推断包传播)，或者关于接收时间的统计。例如，它可能包括周期性的更新，最短的可观察到的包的往返时间是什么。 也可能包括周期性的声明， 对于一个特定的最近的包，用接收器时钟测量的到达时间是什么。 在那种情况下， 接收器时钟和发送器时钟是不同步的，但是发送器可以探测一个变化， 它象征着一个增加或减少的中间包队列。 这样的一个统计可能是一个更加精确的单程传输时间的估计 (虽然它的精确性受到端点时钟差距影响)。

呈现在拥塞控制帧中的精确范围明显基于所使用的算法而变化。

重置流帧

重置流帧用来反常地终止一个流。 例如， 当接收器希望比预期早的阻止发送器发送额外的数据时或当数据源不再供应数据时可以使用它。 为了帮助排出故障， 一个动机短语和一个错误代码被包括在这些帧中。

连接关闭帧

连接关闭帧用来攻击性的和快速的终止一个连接， 隐式地关闭所有流。 为了试图支持这种比普通在 TCP 中还快的快速拆卸，这个帧后面跟着一个确认帧，为了更好地交流拆卸时的状态 (即，在拆卸之前收到的最后一个包序列号是什么)。

离开流帧

离开流帧请求一个连接的完美终止，没有新的流产生，只有已存在的流存留 (有希望立刻结束)。这个帧也包括一个动机短语和一个错误代码。

4.2 连接重置可选项

一个 QUIC 连接可以被连接的任意一端重置。 连接重置意味着， 连接永久地被丢弃或者拆除。 没有使用与连接的当前全局唯一标识符相关的包进行的额外通信。 一个 QUIC 连接不能通过注入一个通用包而被拆除。 当足够多的流量丢失并且没有收到确认时， 一个连接将 (最终？)被丢弃并且有效地重置。

有两种方式交流一个 QUIC 连接重置。两种方法都是经过认证的 (排除第三方出入)。第一种方法是经由一个连接中所有其他已认证的同一认证的标准包， 通过包括一个消失的帧或一个连接重置帧 (以上所有描述的)。第二种重置的方式使用一个单向的预计划的重置，这是

一个共享密钥。

如果收到一个带有不再与一个活跃 QUIC 连接相关的全局唯一标识符的包 (不是一个启动包)，这时需要一个迅速的重置。 这个未被承认的全局唯一标识符 (和相关的包)可能选择性地被忽略 (丢弃)，但是那只会导致一个缓慢的由发送者引起的相关连接的丢弃。

现时标志通常可能被构造为一个全局唯一标识符的 MAC 。

4.3 恶意的返回地址重写

一个恶意的中间盒对手可能重写源 IP 或端口，造成接收器迷惑于发送响应流到哪里。

避免这样一个靠不住的重置的一个方法就是继续指挥包流到先前的源 IP，直到一个“充分”多的来自新源 IP 的包被响应。
另外一个客户端 (或中间盒)可能制造的攻击是故意篡改源 IP 地址，产生一个流放大攻击。
阻止这样一个攻击的比较自然的一个方式是避免广泛使用一个替换源 IP，直到一个包进行了至少一个往返。

中间盒可能有恶意的这个事实暗示，可能在这个协议能够多好 /多块适应源地址的非固定改变上会受到限制。 如果源了解到它的地址正在改变， 在没能够认证细节时， 发送一个已经过认证的改变正在临近的暗示是有可能的， 这样一个暗示是非常有帮助的， 例如， 当一个移动设备切换到从 WiFi，因此可以向服务器发送保证，传输正在发生。

4.4 加密启动概览

有一个单独的 QUIC 加密文档提供细节和精确加密问候协议的理由，密钥交换，等等。这节仅意味着概念性的概述涉及连接建立的事件的序列。 QUIC 加密文档是权威的。

一个典型的零 RTT 刷新连接建立将被客户端围绕推测建立，服务器仍然使用一个先前已知的公钥。基于那个推测，客户端可能传输一个被提议的会话密钥， 等， 用预测的公钥加密它。客户端可以立即跟踪具有加密数据且被初始会话密钥加密的那个连接建立传输。 这个提供基础的高概率的 0-RTT 启动。

为了减少一个意外连接损失的可能性，连接的每一方都应该表明，一旦连接变成闲置，它期望维持会话状态多久。

1 个 RTT 回退

如果在一个连接建立中的最初的推测性的公钥对服务器来说是不可接受的， 或者服务器想要额外验证请求客户端的地址，服务器或许会拒绝连接请求并在开始前进行一个往返验证。期望往返元素包括等值于一个现时标志的价值 (与 TCP 同步 cookie 同类)，等值于一个完整的如 TLS 演示的问候交流。在客户端响应服务器问候之后，在完整的协商会话密钥保护下，客户端可能重传任何数据请求 (可能被一个服务器斜面丢弃)。在这个例子中，服务器可能选择性地丢弃所有先前的数据包，可能在推测性的公钥保护下发送的那些数据包。

2 个 RTT 的最糟糕的回退情况

服务器需要响应一个广泛的证书链这个事件中， 服务器或许会拒绝发送一个多包有效负

载给一个客户端，直到客户端可以证明它拥有了一个特定的源地址。在那种情况下，问候交流或许会花费多达 2 个 RTT。

在这个最糟糕的情况下，第一个客户端问候会被拒绝，但是服务器会提供一个令牌来证明客户端地址的所有权和服务器证书，不包括证书链(如果它太大以至于不能填充进单个包)。

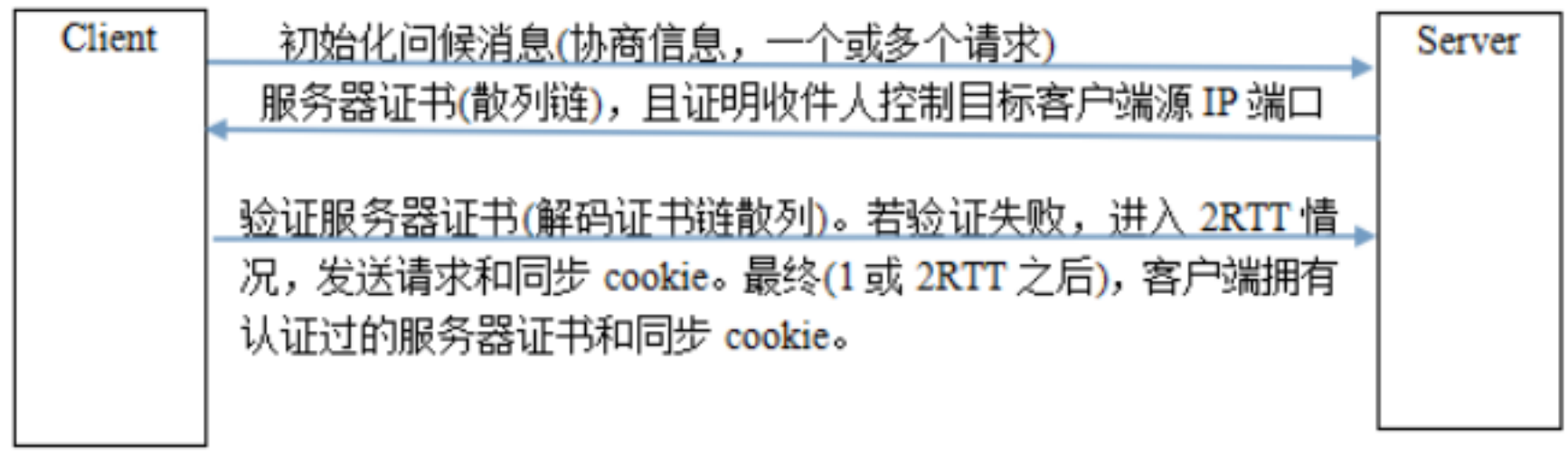
客户端将会在第二个问候请求中使用源地址令牌，但是可能直到它收到完整的证书链时它才会愿意相信那个证书。在那个例子中，服务器将会拒绝第二个问候(因为对一个共享密钥来说缺乏任何实际的提议)，但是将至少回复一个完全充实的证书链，需要发送几个包。

最后，拥有一个值得信任的服务器证书和一个可信任的源地址令牌在手，客户端可能就像在那个 0-RTT 例子中建议的一样提供一个客户端问候，连接将开始。

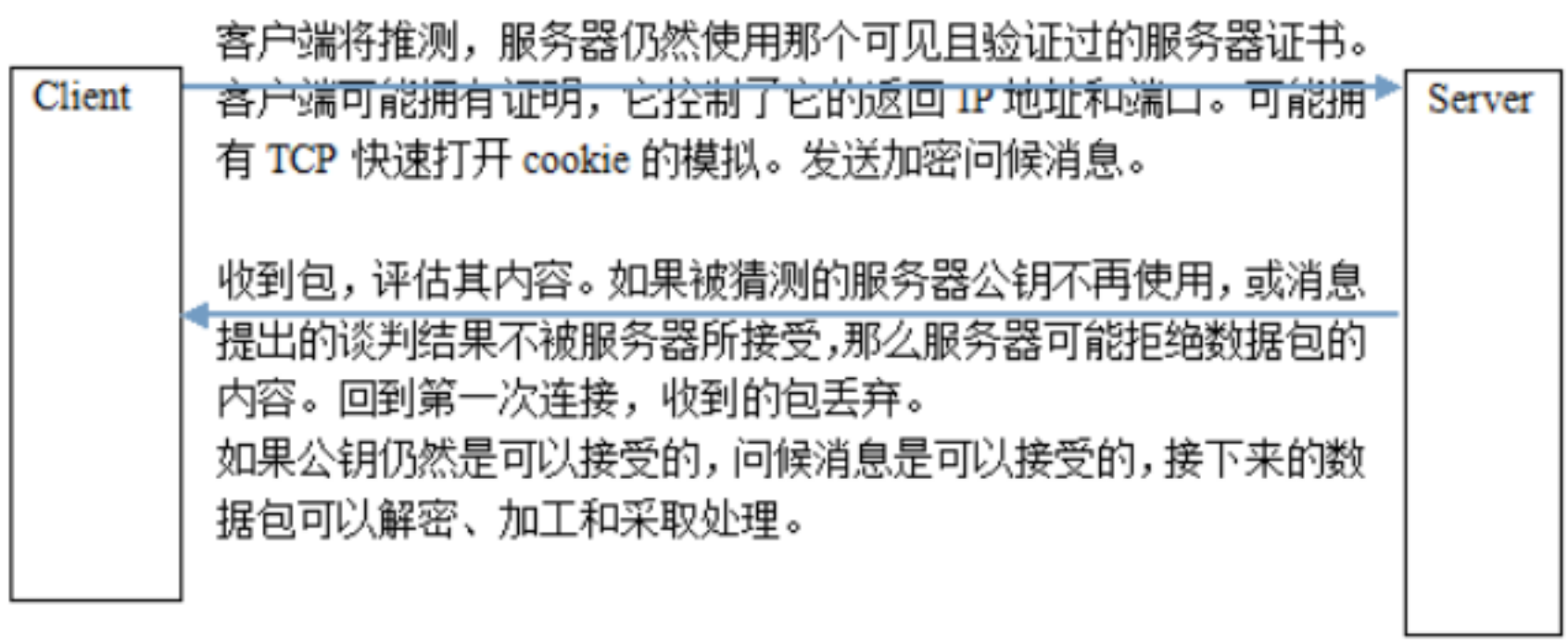
5、 QUIC 连接

协议有四个阶段，我们需要考虑性能效率：启动，稳定状态，闲置进入，闲置离开。第一个挑战是减少启动(连接建立)期间的延迟。第二个挑战是确认在稳定状态下的有效的和低延迟的性能。第三个挑战是有效地且低延迟地确认平稳过渡到一个空闲状态。过渡是明显的，TCP 包丢失的尾部包丢失历来发生显著延迟，特别当最后一个包丢失时，发送端超时需要重新发送丢失的数据。

启动：

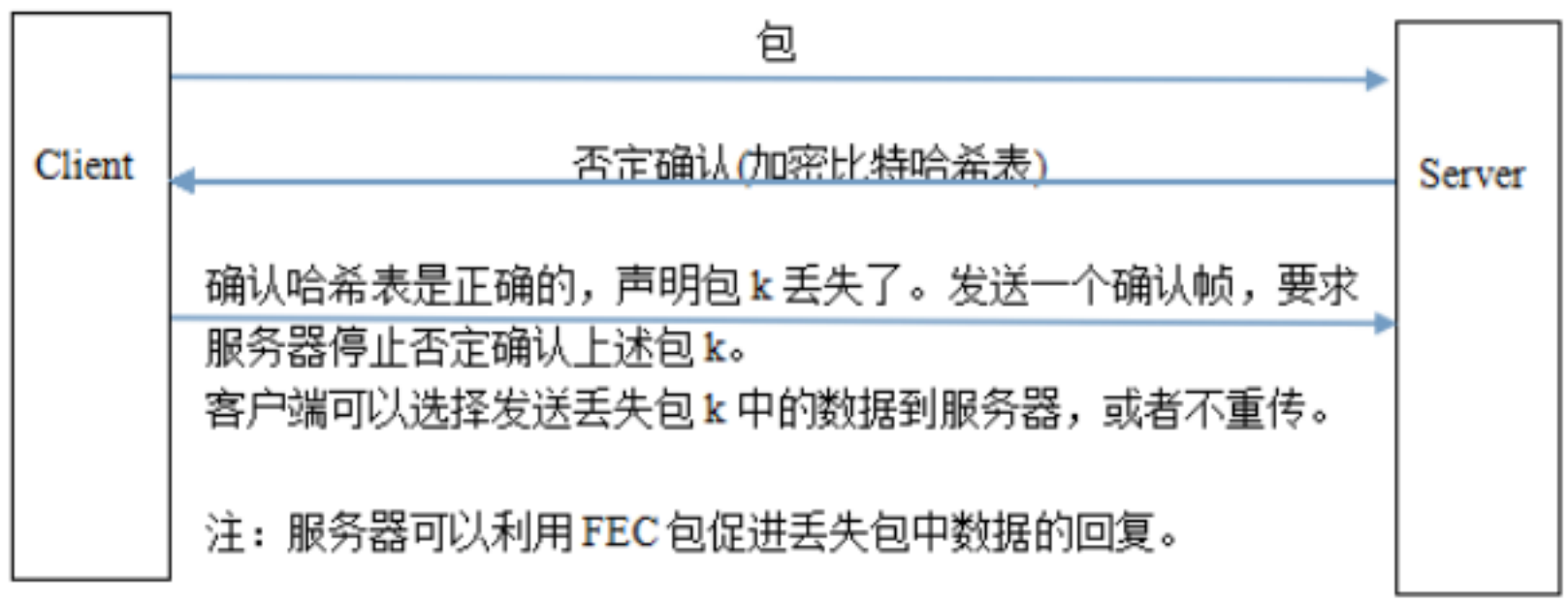


第一次连接：1RTT，有时 2RTT

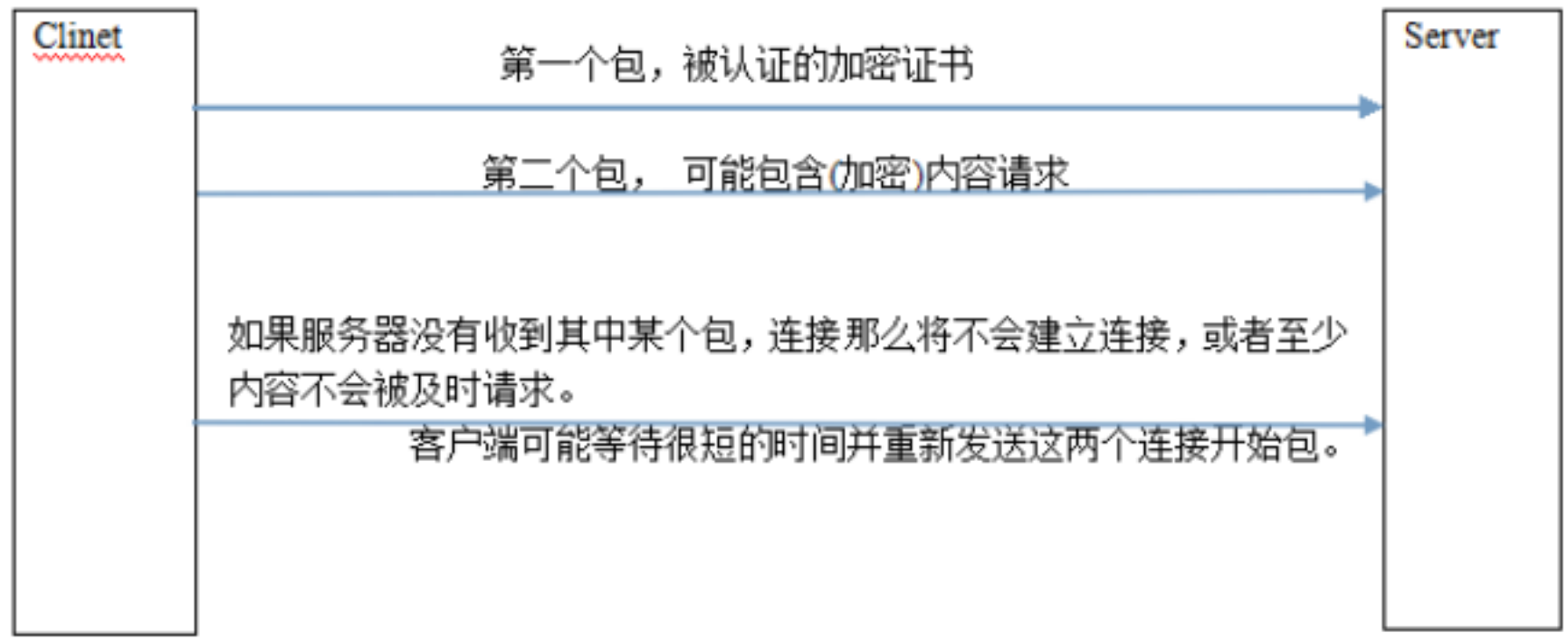


重复连接：通常 0RTT，有时 1RTT，很少 2RTT

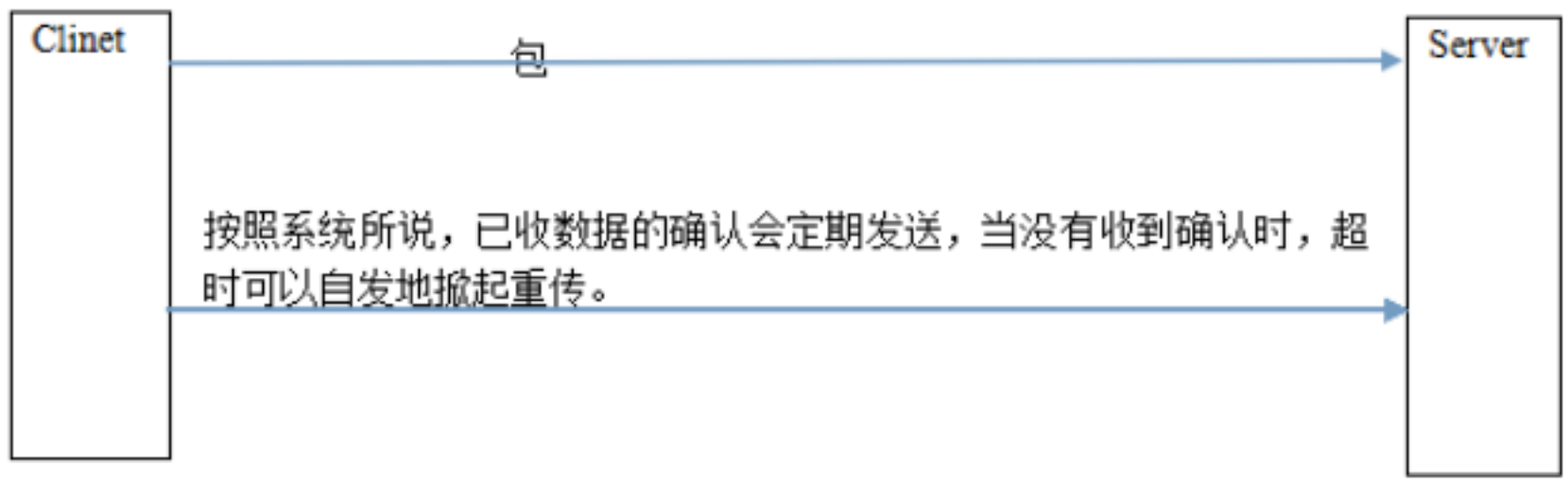
稳定：



丢包

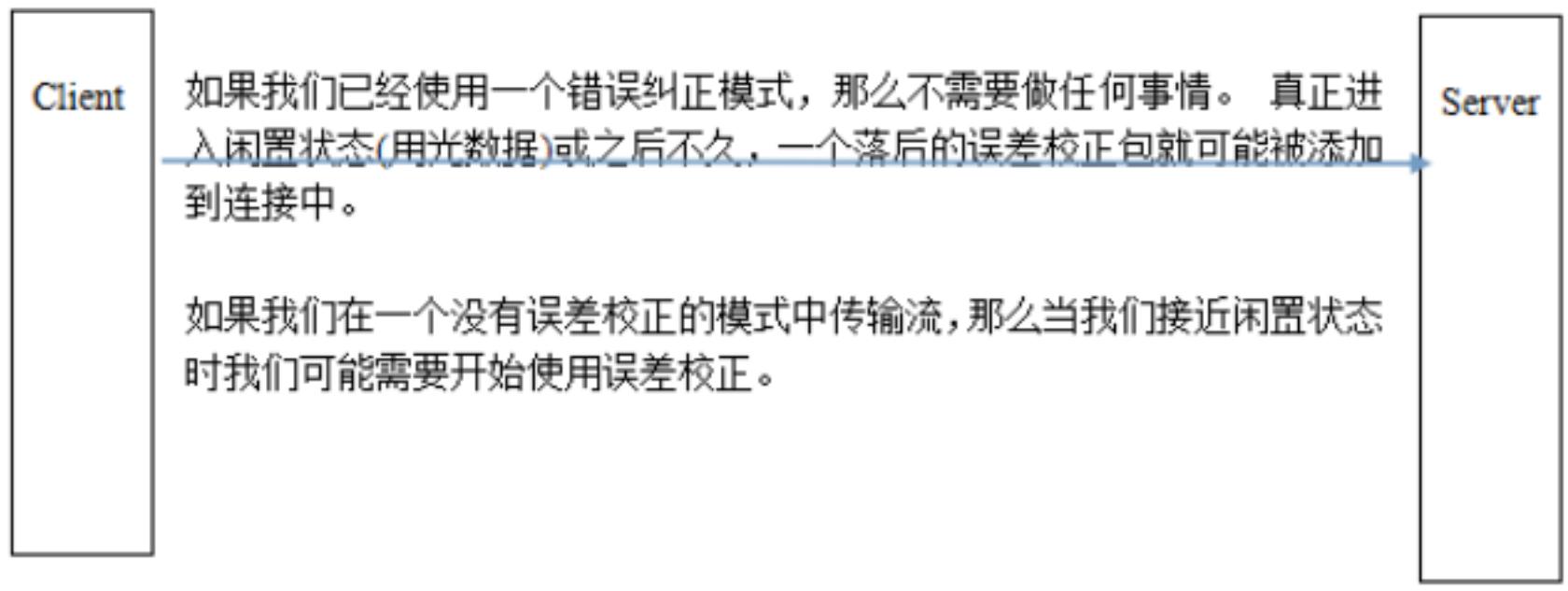


积极地推测重传

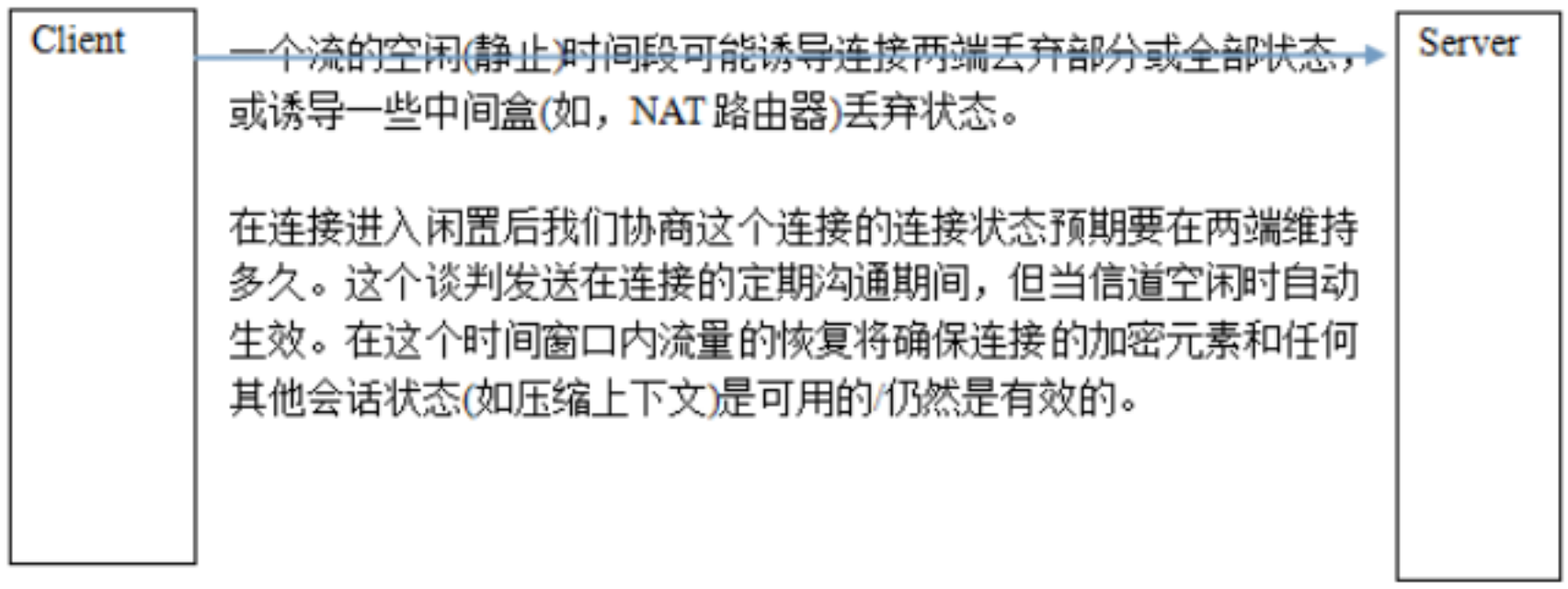


包丢失的重传恢复

空闲进入：



空闲离开：



5.1 通过无连接的 **UDP** 建立连接：克服 **NAT**

一个最基本的问题是怎样将 UDP 转换成一个基于连接的协议。

例如，考虑一个中间框的情况下，如防火墙和 NAT，决定不再支持一个特定的 TCP 连接。中间框可以发送一个 TCP 重置 (连接重置)到两端作为一个断开连接通知的一部分。相反，当一个 NAT 服务决定丢弃一个使用 UDP 流量的绑定，没有通知，因为一个 UDP 数据报不希望是一个“ 连接 ”。一旦一个 UDP NAT 释放发生，外部端点 (通常是一个服务器)无法发送流量到客户端。 更糟的是， 来自服务器的流量通常是黑洞，没有否定确认响应。对 TCP，试图使用一个未绑定端口可能导致至少一个堪比否定确认的重置。 作为进一步的并发症，如果一个 NAT 端口未绑定发生，然后客户端发送额外的 UDP 流量， NAT 服务可能创建一个新绑定。新端口绑定可能有不同的源端口 (在服务器看来），尽管 QUIC 需要把流量看作一个现存连接的持续。

目前的估计表明，对现在已部署的 NAT 盒来说，在 30 到 120 秒的空闲时间之后释放一个 UDP 端口映射是非常常见的，并且释放可能会很快发生。

所以需要一个 GUID(全局唯一标识符)来标识一个连接。

UDP 包分片

单个链路上大于 MTU 的 UDP 数据包有在 IP 层被分片的风险。我们是否把我们所有的数据包标记为“ 不分片 ”是待定的，如果超过 MTU 的话，这将导致包损失。但随着包分片的普遍，这可能不再是问题。

当一个包在 IP 层分片时，初始包将继续持有 UDP 源和目标端口说明符，以及一个全局唯一标识符，但所有后面的分片将缺乏这样的识别信息。重新组装的惟一手段将是 IP 层的“ 识别 ”字段， 它只有 16 位长度。 因此， 如果超过 (大约)2 ^ 16 的平方根的包同时在传输，碰撞 (和破碎的重新组装)的概率将是巨大的。

收集可能一次轻易有超过 256 包在传输，争夺唯一的 IP 标识符，造成重组碰撞 (如果分片)。

碰撞，可能基于相同的 MTU 边界，将 (面对包重新排序)，几乎不可能重新组装正确。因此，接收方要么放弃重新组装工作 (在一些分片包的一部分)，要么它将提供一个错误的重新组装。 错误组装的包将被探测为被一个认证过的哈希表歪曲了。 因此， 重新组装错误不会导致比丢弃可能分片了的包更坏的协议错误。

操作系统 API 不允许一个应用层代码探测 IP 分片是否已经发生。我们可能试图增强我们的服务器操作系统支持，以便检测分片。有这样一个 API，我们可以尝试在任何可见的 MTU 限制内运行。

5.2 连接建立和恢复

最小化启动时的延迟， 加快第一次接触时的数据响应， 第一次通过 QUIC 发送的数据包通常会包括会话协商信息以及一个或多个请求。 QUIC 目的是推测性地假设客户端有可接受的至少需要一个初步加密请求的加密证书， 它有足够的不需要反分布式拒绝服务挑战的连接凭证，而且它有足够的新证据表明重播攻击可以排除。如果服务器拒绝接受凭证，比得上 TCP 会话建立或 SSL 问候协商的额外的反复协商就会接踵而来。

启动 **DDOS** 攻击

关于快速启动的一个已知的问题是拒绝服务攻击。 在这种攻击中， 恶意客户端可能会提供一个错误的返回 (响应) 地址，并可能会导致服务器耗尽重要的计算资源， 和 / 或发送重要的流量给一个第三方地址。从历史观点上说，这些对 TCP 服务器的攻击已通过要求一个额外的往返 (确认返回地址)和同步 cookie 的利用被避免。最近围绕具有 TCP 快开放的 TCP 进行的扩展工作试图答应改良 TCP 的方案，使 TCP 包括在初始同步包中的数据，具有对 DOS 攻击的合理的可接受的控制。

QUIC 将不得不解决连接证书期满的问题，和 / 或其他机制来阻止重放攻击使用这些凭证。我们希望设计的协议，包括密码恢复，像 TCP 快速打开协议一样强健并且能处理这样的攻击。 当一个服务器被攻击或出现过载时， 论文包括源 IP(基于以前的连接历史)所有权的证明和到 3 次握手式连接的自动回退 (re: 增加一个额外的 RTT) 。我们的协议始终包括端到端加密的这个事实使我们确信，所有权凭证不可能简单地通过偷听被偷走。

安全认证

客户机应该在与上述服务器的先前联系中保留关于一个服务器的信息。 保留的信息应该至少足以支持 TLS 会话恢复的模拟， 还应该包括服务器信息， 比如服务器最近使用的公钥 (历史性地保存在一个证书里，用一个到可信根的关联链)。客户应该推测性的假设，最后的已知服务器公钥仍在使用 (除非有证据表明撤销或更换)，并尝试利用这些信息实现加密初始载荷转移 (请求)的一个零 RTT 传输。

支持加密的没有往返的服务器将随机性添加到连接中， 要求服务器维护状态以避免重放攻击。该状态可以通过假设一定数量的时钟同步及时被限制，在太空，给客户端一个 ID 标识碎片 (称为一个快速启动的轨道)。服务器可能无法建立连接的唯一性，在这种情况下，客户机必须准备再加密和重传它乐观发送的每件事。

同样地， 如果服务器密钥已经改变， 客户端将不得不重加密和重传乐观的初始的数据流量。

在一个零 RTT 设置中的初始流量可能没有想象中安全，所以我们应该假设连接将升级到一个真正的在同一连接上的后续流的临时密钥。尽管支持 0- RTT 连接建立的默认设置应该是默认的，服务器应该可能坚持，只有完全向前安全加密用于甚至最初的流量。

高水平连接情况的概述

本节旨在概述 QUIC 连接通常会如何进行。 我们将着重概述导致每种情况的 RTT 数量。我们还将提供一些用来减轻可能发生的袭击的细节的动机。

第一次连接：通常 1RTT，有时 2RTT

第一次连接，客户端联系服务器，它的初始化问候消息表明，客户端从未访问服务器，因此它不能推测公钥。来自客户端的最初的消息可能包括一些加快一个会话协商的随机性。整个客户端消息将适合一个包， 并且应该填补 /填充这包。 通过填写第一个客户端生成的包，我们保证服务器会发送一个完整的响应数据包，而且它通常不会超过客户端的第一个包。

当客户端收到上面的包， 它可以尝试验证服务器证书。 验证证书的第一步是解码证书链散列。 如果客户无法充分地解码证书链散列， 或证书链似乎没有验证服务器证书， 然后客户端进入 2 RTT 的情况，并为了全面阐述证书链而被迫发送一个请求到服务器。随着请求发送的还有同步 Cookie，它允许服务器安全地将额外的数据包发送到客户机 (没有误导放大攻击的风险)。

最终 (1 或 2 个往返之后)，客户端将有一个经过身份验证的服务器证书，也有一个短暂生存的同步 Cookie 模拟。有了这些信息，客户端可以进入到下面描述的 0 RTT 情况，并被立即作为可信的客户端返回的 IP 地址和端口所有者。

重复连接：通常 0RTT，有时 1RTT，很少 2RTT

在一个重复连接中，客户端将推测，服务器仍然使用那个可见且验证过的服务器证书。此外， 客户端可能拥有证明， 它控制了它的返回 IP 地址和端口。 证明可能包括在上面的 “ 首次连接 ” 一节中描述过的 SYN Cookie 的模拟，也可能包括一个 TCP 快速打开 cookie 的模拟，这是生存更长的 (有效期在更长的时间内)。

为了继续此连接，客户端会建造一个消息的 TLS 会话主秘钥的模拟。在 SSL 快速启动中使用的技术将用于这个结构，以减轻重播攻击。也包括在这个构造中的 (加密)消息将是控制客户机的 IP 地址的 “ 证据 ”，以及任何 QUIC 可转让项目，例如提出的拥塞避免算法等。加密的证据 (如 TCP 的快速打开 cookie 模拟)将确保一个偷听者不能偷和滥用这样的证据。

上面的加密问候消息将发送一个包到服务器。 发送之后， 客户端可以加密、 验证和发送额外的数据包。

当服务器获取加密问候包时， 它将评估其内容。 如果被猜测的服务器公钥不再使用， 或消息提出的谈判结果不被服务器所接受， 那么服务器可能拒绝数据包的内容， 而把它作为上面所描述的相同的 “ 第一次连接 ”。 在这种情况下， 利用提议的主秘钥后到达的数据包将被丢弃 (没有办法在服务器端解密)。

如果公钥仍然是可以接受的，问候消息是可以接受的，接下来的数据包可以解密、加工和采取行动。

客户端 IP 地址所有权的证明

客户端 IP 的所有权证明是不够的几个原因。许多这些原因与服务器多忙以及在资源利用攻击下的可能性有多大有关，这可能会提高或降低证明的标准。同样，如果证据更旧，它可能被视为不值得信赖的。最后，如果证据没有直接适用而是一个“附近 IP”的证明 (实际 IP 证明备用 IP)，那么它可能被视为比一个更加远的相关 IP 的证明更加值得信赖。如果证明不足够信任，那么服务器可能发送一个拒绝 (探测)包，以确保客户端返回的 IP 和端口是真实的。

生成客户端 IP 地址所有者 / 控制的证明

在一个连接中，一个 QUIC 服务器可能创建并发送一个声明，客户正在一个特定的时间使用一个特定的 IP 地址和端口。这样的声明将包括一个由服务器密钥生成的 MAC，这样服务器就可以验证这一说法的 0-RTT“重复连接”证明。服务器可能会推动这个语句 (具有最近的时间戳)在这个连接中的更新。

客户端可以组装一个或多个这样的不同的 IP 地址证明列表。例如，它可能获得被家庭 WIFI 防火墙使用的 IP 所有权的证明，以及在一个 3G 移动连接中使用的 IP 地址。在漫游的情况下似乎是合理的，或可能的，客户端可能为服务器提供一个带有多个预期未来漫游的所有权证明。当服务器提前意识到可能漫游 (和可能发挥作用的特定 IP 地址)，它通常可以立即回复客户返回的 IP 地址的改变，没有拖延执行调查的过渡。

5.3 稳定状态

在 QUIC 连接中的每个流将有一个独特的相关联的流标识符。

连接结构

流将被划分成帧放入 (UDP) 包中。只要有可能，任何特定的数据包的有效载荷应只来自于一个流。这样的奉献将减少这种概率，一个包丢失将阻碍不止一个流的进步。当没有足够的数据流来填充一个数据包，然后来自不止一个流的帧可能被打包进一个包。这种包装应减少数据包计数开销，减少序列化延迟。

安全：抗干扰、保密、真实性

给定一个包，我们需要一种方法来识别加密上下文。上下文将被关联到出现在每一个 (UDP) 数据包中的全局唯一标识符。

数据包将被在加密上下文中使用共享密钥的 AEAD(与相关数据验证加密)保护。我们预计在每个数据包大约需要 8 个字节的认证开销，但细节待定，基于艺术标准的状态。

每个数据包需要一些用于加密的初始化向量字节。现时标志需要每一个数据包是唯一的。 AEAD 算法将被选择用来做一个简单的计数器 (即，必须是唯一的，但初始化向量的可预测性是可以接受的)，并将基于数据包序列号。

我们应该提供支持分组打包以减少流量分析的易损性。 在这一点上， 反流量分析可能没有被充分理解怎样设计使用这种能力的对策。

安全协议应该旨在往返。 我们对零 RTT 安全协议有经验， 我们明显地可以做的比 (D)TLS 更好。

孤立包加密：

在任何孤立包加密模式中， 我们避免加密增加延迟， 除此之外， 都是被一个延时包强制的。例如，如果一个数据包的解密需要其他包的明文或密文 (如，一些密码块链接模式)，那么我们就不会有一个孤立的包加密属性。没有那个属性，数据包的延迟 (损失 ?)将推迟另一个包的解码，并且是不可接受的。

包丢失

包丢失的主要原因被认为是拥塞，拥塞地点的路由器执行切换操作且输出缓冲区大小超标了。这个问题是 TCP 和互联网设计的根本，包丢失在那里被用作拥塞信号，协议需要通过减少经过拥塞路径的流来响应。

包丢失将由两个机制处理： 包级别的纠错码和数据丢失重传。 最终退却发生在其他所有失败都是丢失数据的重传。 当以数据在 QUIC 转播来响应数据包丢失时， 原包不转播。 相反，封装数据被放置在一个新的包里，那个新包被发送。

为了减少短流的延迟， 为了一些 (所有？)流的终端， 按照协议添加冗余信息来促进纠错，并减少重传的需要。对于较大的流，序列化延迟被认为是主导因素 (通过构造和发送流的应用程序)，对大多数的流数据包来说前向纠错冗余的使用可能被减少或消除。

1) 拥塞避免

包丢失将会被假设成连接中一些包经过的路径拥塞的标志，并将近似地处理目前 TCP 处理它的方式。接收器判定包丢失时，一个显式的否定确认 (NACK) 将被传送到拥塞连接的来源处。

在响应一个否定确认时，发送器应该改变它的传输速率，如在 UDP 包匀速传输时通过减少拥塞窗口或添加额外的等待时间。调整了发送方的传输速率应该类似于 TCP 传达的精神。在总体影响上，如果所有多路复用流是单独的 TCP 连接且这些假设的 TCP 流中的一个收到包丢失通知，响应应该大致相当在 TCP 中发生的。例如，我们希望这些假想的 TCP 连接的拥塞窗口总和的改变与我们的单个 QUIC 拥塞窗口的变化有相似的方式。 这种相似性的目的是“ TCP 友好 ”，既不支配流 (阻碍 TCP 活动)，也不放弃控制 (允许 TCP 控制利用)。

使用同样对 TCP 友好的算法，仅当收到数据确认时拥塞控制窗口扩大之时，带宽利用率才可能会增加。

2) 对乐观确认攻击的攻击缓解

一个已知的对 QUIC 来说可能更糟的 TCP 攻击是乐观确认攻击。在这种攻击中，一个畸形的或潜在的恶意客户端始终确认包，甚至包括那些没有真正收到的包。在 TCP 中，如果没有数据包 (报告)丢失， 然后发送方 (服务器 ?)使用的带宽可能没有界限地增长。这种攻击的结果是，服务器可能无意中洪泛 ISP，当一个流氓客户端误导服务器时创建一个 DOS 攻击。对于 QUIC 来说，依赖一个接收机来评估潜在的可用带宽可能导致服务器更快 DOS 攻击一个 ISP。

防御这种攻击是相当简单的： 客户端必须证明它已经收到了包， 在服务器应该交付高层带宽传输之前并没有遇到包丢失。 只要客户端证明正在接收所有的包， 没有理由使服务器节流。一旦如实报告包丢失， QUIC 服务器将压制数据包发送速率 (或未偿付的包窗口)。

为了支持收据的概率证明，每个数据包将包括大约一个比特的熵 (存在于加密的有效负载中的一个不可预测的比特)。包的一个至少稍稍不可预测地因素是包的传输时间 (实际上是包含在 MAC 区)，而在每个数据包中的一个显式随机比特简化了证明确认。

实际接收证明将伴随每个 QUIC 确认，它将表现为一个可交换的校验和的形式 (即，一个校验和不依赖于校验和被创建的顺序)。校验和将是 8 位长度，并将包括来自接收方声称它已经收到了所有数据包的熵。 如果一个服务器检测到一个错误的校验和， 它将推测原因是一个畸形的接收器，并将终止连接。

这种缓解的一个剩下的问题是，确认必须包括丢失的所有数据包的一个显式列表 (因此不包括校验和)。

3) 合理的误差校正模式

与传统的连续流误差校正不同， 包丢失时的误差校正应该利用整个包通常是丢失或完好无损到达的这一事实。

关于一个简单的阻塞式误差校正方案的例子，发送 N 个数据包的有效载荷，然后发送所有有效载荷的奇偶性 (按位求和)。那个方案可以视作支持 1 个包丢失的错误校正。虽然奇偶性在传统上只相当于 1 位错误检测能力 (在这种情况下一个包丢失)，事实上，丢失的比特 (如果有的话)在一个已知的位置，这允许恢复丢失的数据包 (假设不超过 1 包丢了)。这样一个简单的奇偶校验计划的优点是可伸缩的冗余，因为一个小值 N 意味着高冗余 (额外的带宽使用)，而一个较大的值 N 意味着减少冗余和任意小的超额带宽。这样一个简单的方案的另一个优点是， N 值的大小可以晚一些做决定， 也就是说当没有更多数据要发送或到了流的尾部时，发送方可以决定使用一个更小的值 N(并且发送一个奇偶校验包)。

[使用这种大 N 值的误差校正需要谨慎的一点是，包丢失的延迟影响及其相关的错误恢复可能与 N 成比例增长。因此，当需要有界延迟和 /或抖动时， N 的大小应该受到 RTT * unidirectional_utilized_bandwidth / avg_packet_size 左右。方程中，“ 利用带宽 ”是预计使用的数据率，并可能小于最大可用带宽。从本质上讲，如果需要接收方超过 1 个 RTT 得到 N 个数据包块中第一个包的奇偶性比特，就没有减少延迟 (在重传请求中)。]

上面的示例类似于 RAID 3 到 RAID 5 的配置。使用奇偶校验系统类似于基于里德所罗门码的 RAID 6 ，两个奇偶校验数据包发送后发送 N 个数据包， N + 2 个发送的数据包中有 2

个包丢失应该有可能纠正的 (不需要一个数据包重传)。

在 Chrome 上进行的发送 UDP 数据包的实验表明，当数据包是快速的，损失足够去除相关，上述简单的异或 FEC 策略对组包可能是有价值的。不幸的是，有一些相关损失，并试图从大爆炸损失 (2 个以上的包)，对一个大的消耗来说收益递减。结果，我们将重点讨论基于 QUIC 中的 FEC 的异或。

当误差校正被使用时， 收集明文有效负载， 计算相应的误差校正有效负载， 误差校正包传输加密有效负载。 因此， 当一个包含误差校正的包在传输时， 不应该对一个偷听者来说是显而易见的。

4) 逐步减少包丢失

Chrome 的实验已经表明，通常可以通过快速包来减少包丢失。

关于拥塞避免的当前计划是基于带宽估计和速度， 并因此减速损失可能放弃设计。 我们可能需要更好地深入操作系统，以更好地促进在操作系统级别的缓冲中准确的速度。

5) 包丢失的重传恢复

按照系统， 已收数据的确认会定期发送， 当没有收到确认时， 超时可以自发地掀起重传。

6) 积极地推测重传

积极推测重传相比于 FEC 组(例如： 拥有一个数据包和一个 FEC/冗余包的 FEC 组)的优势在于， QUIC 中的 FEC 组将按顺序传送， 虽然投机传输可能容易发生故障， 进一步减少了机会，任何相关的包丢失将影响不止一组冗余数据包中的一个。

初始传输和加密数据包重传有这么一个差距，两者都丢失是非常不可能的。

缓冲溢出

缓冲溢出已经证明始终是一个困难的问题， 不仅对 TCP 而且对等更高层的协议如 SPDY 同样如此。我们需要尝试整合控制，它将减少成为解决这个问题的贡献者的可能。

缓冲溢出可能对于要求及时交付的流来说更加是一个问题。

拥塞时， 在任何包含我们流量的臃肿缓冲区减少流量比例， 会最终导致我们都饿死。 通过比例填充膨胀的队列，我们会是缓冲区膨胀的一个贡献者。所以，这是一个难题。

1）本地缓存控制

缓冲可能是可控的、 可观察的一个地方是在与发送方端点相关联的机器上。 发送方机器通常可能有非常高的本地带宽， 作为衡量填充输出缓冲区的能力， 这可能明显高于链路层出口带宽，如有线以太网或 WiFi 。我们需要努力使这个协议了解到本地当前队列大小 (硬件和操作系统将允许的范围内)，努力将实际尺寸减小到最低，以避免本地出口链路由于缺乏有效的缓冲数据而饥饿。

基于流的流量控制

流量控制为接收端状态提供了一个基本能力，如过度使用 (缓冲区)内存，诱导一个发送

端减少 (或在极端情况下，停止)传输的请求。用一个多路复用协议，接收端的流的消费者可能是缓慢的，或不消耗数据，而其他来源水槽对可能“ 正常 ”运行。我们乱序交付的事实保证我们不会阻止其他流， 但我们可能因为未耗尽的流数据包而面临接收端 (缓冲 /内存)资源枯竭。

5.4 空闲进入

某时，所有连接是闲置的，因为没有任何数据流等待传输。在 TCP 中，当我们进入空闲状态 (又名，尾下降)，落后的丢失数据包可能引发基于计时器的重传。这样的重发有可能添加显著的延迟，而且纠错冗余应该用于减少这样的重发的概率。

当我们接近一个闲置状态时， 如果我们已经使用一个错误纠正模式， 那么不需要做任何事情，因为一旦我们真正进入闲置状态 (用光数据)或之后不久，一个落后的误差校正包就可能被添加到连接中。 如果我们在一个没有误差校正的模式中传输流， 那么当我们接近闲置状态时我们可能需要开始使用误差校正。

从概念上讲， 当我们接近任何流的闲置状态时我们应该添加误差校正冗余， 即使相关流一般不会使用误差恢复。一旦我们预料我们可能在不到一 RTT 进入空闲状态，我们应该以最佳方式开始这个阶段。

5.5 闲置离开

1） NAT 表重置

NAT 盒可能有他们的超时或相反的到期条目，导致底层 UDP 传输的崩溃。如今，通常在静止 30-120 秒后到期。

2）全连接状态的延续

当客户端试图在之前建立的连接上继续传输数据时，第二个闲置离开的例子可能会出现。对于 TCP，在重置已经发送前这类延续是容许的，且目前许多服务器在大约 5 分钟静止后重置 TCP 连接。 对于 QUIC，鉴于这样一个事实， 通常可以在先前的连接中用获得的凭证建立一个 0RTT 的再连接， 我们最初是打算只保持一个连接活着 (没有流开放)约 30 秒。这也应该允许我们更普通地避免 NAT 重新绑定的复杂性。

考虑到担心电池寿命且无用地唤醒接收机说 “ 我们做完了 ” ，在连接进入闲置后我们协商这个连接的连接状态预期要在两端维持多久。 这个谈判发送在连接的定期沟通期间， 但当信道空闲时自动生效。 在这个时间窗口内流量的恢复将确保连接的加密元素和任何其他会话状态 (如压缩上下文)是可用的 /仍然是有效的。

5.6 加密元素

一个加密安全和认证信道的初始化协商，在之前从来没有被一个客户端访问过的网站上，将明显地模仿 TLS，且一个传统的 TLS 问候消息交换。同样地，在客户端的任何数据传输之前，绝对第一的交互需要一个完整的 RTT。首先介绍后，包括加密参数的协商，与证书链的交换等，客户端可以记录这些活动的结果，并使用它们来执行更快速的未来连接。

1）保留的加密通信流

所有密码交流倾向于使用一组消息的有序交付， 这个事实对分析和修正协议证明是至关

重要的。因此，我们将保留一个流 ID = 1 的加密流，作为加密信息元通信信道。

减少头阻塞可能性的第一种方法是投机/冗余关键包的传输。

第二种方法特别注重密码规范改变的时间点(如，加密密钥更改)，类似于 TLS 变化密码信息被利用时的时间点。

2）加密和认证

通过为每个包得到一个来自 UDP 包序列号的初始化向量，我们将避免 QUIC 的序列化解码依赖(这将损害我们在包丢失时提供乱序传输的能力)。

3）会话密钥升级

一个连接的生命周期中，有几个改变会话密钥的原因。第一个变化是一个无效的加密(仅用一个已知的密钥验证)传输。客户端发送的包含加密客户端问候的初始包必须无加密传输，但后续数据包的数据可能在每次协商后加密。第二个示例以投机连接为中心，这可能取决于一个完美前向安全是不保证的会话密钥。在推测性的连接建立之后，我们将改变成一个完美前向安全被保证的会话密钥。