2. 逐跳使用期的计算

这样就可以去除时钟偏差造成的负数使用期了，但对时钟偏差给精确性带来的整体偏差，我们能做的工作很少。文档经过代理和缓存时，HTTP/1.1 会让每台设备都将相对使用期累加到 Age 首部中去，以此来解决缺乏通用同步时钟的问题。这种方式并不需要进行跨服务器的、端到端的时钟对比。

文档经过代理时，Age 首部值会随之增加。使用 HTTP/1.1 的应用程序应该在 Age
190 首部值中加上文档在每个应用程序和网络传输过程中停留的时间。每个中间应用程序都可以很容易地用本地时钟计算出文档的停留时间。

但响应链中所有的非 HTTP/1.1 设备都无法识别 Age 首部，它们会将首部未经修改地转发出去，或者将其删除掉。因此，在 HTTP/1.1 得到普遍应用之前，Age 首部都将是低估了的相对使用期。

除了基于 Date 计算出来的 Age 之外，还使用了相对 Age 值，而且不论是跨服务器的 Date 值，还是计算出来的 Age 值都可能被低估，所以会选择使用估计出的两个 Age 值中最保守的那个（最保守的值就是最老的 Age 值）。使用这种方式，HTTP 就能容忍 Age 首部存在的错误，尽管这样可能会搞错究竟哪边更新鲜：

```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$corrected_apparent_age = max($apparent_age, $Age_header_value);
$age_when_document_arrived_at_our_cache = $corrected_apparent_age;
```

3. 对网络时延的补偿

事务处理可能会很慢。这是使用缓存的主要动因。但对速度非常慢的网络，或者那些过载的服务器来说，如果文档在网络或服务器中阻塞了很长时间，相对使用期的计算可能会极大地低估文档的使用期。

Date 首部说明了文档是在什么时候离开原始服务器的，[21] 但并没有说明文档在到缓存的传输过程中花费了多长时间。如果文档的传输经过了一长串的代理和父缓存，网络时延可能会相当大。[22]

没有什么简便的方法可以用来测量从服务器到缓存的单向网络时延，但往返时延则比较容易测量。缓存知道它请求文档的时间，以及文档抵达的时间。HTTP/1.1 会在这些网络时延上加上整个往返时延，以便对其进行保守地校正。这个从缓存到服务

注 21：注意，如果文档来自一个父缓存，而不是原始服务器，Date 首部反映的仍是原始服务器，而不是父缓存上的日期。

注 22：实际上，这个时延不会高于几十分之一秒（不然用户就会放弃），但即使是对生存期很短的对象来说，HTTP 的设计者也希望使用尽可能精确的过期时间。