10.1.2 分析

我们知道，Pete 的问题在于，他不能浏览某一个网站，所以我们首先考虑 HTTP 协议。如果你阅读了之前的章节，那么你会对客户端和服务器之间的 HTTP 通信有基本的理解。我们建议，从客户端向远程服务器发送 HTTP 请求的地方开始分析。你可以使用过滤器筛选 GET 请求数据包（使用`http.request.method == "GET"`），也可以使用主选单中的**Statics > HTTP > Requests**（见图 10-2）进行过滤。

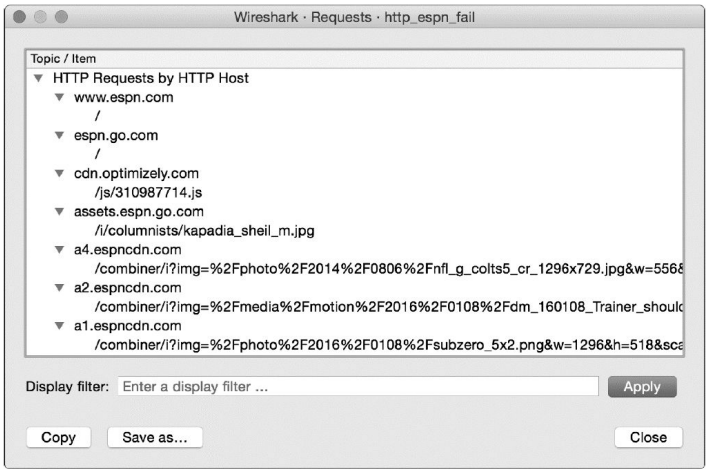


图 10-2　查看访问 ESPN 网站的 HTTP 请求

从这个概览中看到，抓取的数据包只包含 7 次不同的 HTTP 请求，这些请求都与 ESPN 网站有关。除了一个为大量网站投放广告的内容分发网络（CDN）服务，每个请求都在域名部分包含字符串espn。在浏览含有广告或其他外部内容的网站时，通常会看到大量对 CDN 服务的请求。

在没有明确目标会话的情况下，下一个步骤是，选择**Statistics > Protocol Hierachy**对抓包文件进行协议分层统计。这将帮助我们定位非预期的协议和通信过程中各种协议分布情况的异常（见图 10-3）。请注意，协议分层基于当前的筛选条件进行。为了对整个抓包文件进行分析以获得期望的结果，请先清除之前的过滤器设置。

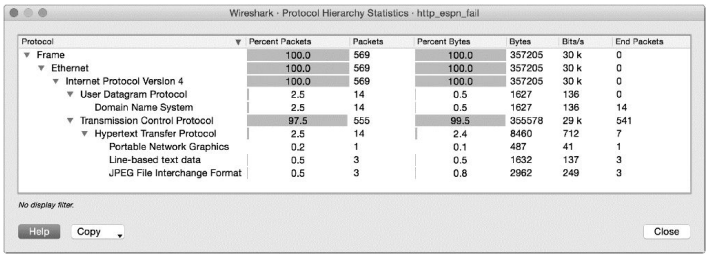


图 10-3　查看网页浏览会话的协议分层统计

协议分层视图并不复杂，我们能够很快地辨别出，会话只涉及两个应用层协议：HTTP 和 DNS。正如在第 9 章中学习到的，DNS 用于将域名转换为 IP 地址。因此，当访问一个网站，如 ESPN 时，如果你的系统没有缓存这个站点的 DNS 记录，则系统将发送一个 DNS 请求来查询远程网站服务器的 IP 地址。当 DNS 服务返回了一个可用的 IP 地址时，则域名解析信息将被添加到本地缓存中。此时，HTTP 通信（使用 TCP 协议）可以开始了。

虽然分层统计结果看上去并没有什么异常，但是 14 个 DNS 数据包是值得注意的。一个查询单个域名的 DNS 请求通常只包含一个数据包，请求响应同样由一个数据包构成（除非响应数据包很大，这种情况下，DNS 数据包将使用 TCP 传输）。会话中包含 14 个 DNS 数据包，这说明可能发生了多达 7 次的 DNS 请求（7 个请求数据包 + 7 个响应数据包 = 14 个数据包）。如图 10-2 所示，Pete 向 7 个不同的域名发起了请求，但是他在浏览器中只输入了一个 URL。其他请求是如何产生的呢？

在理想状况下，浏览网页只需要查询到一个服务器地址，然后在一次 HTTP 会话中即可获取所有的内容。但在实际情况中，一个网页可能提供多个服务器上的内容。可能的场景是，全部基于文本的内容在一处，图像内容在另一处，内嵌的视频则在第三个位置。这还不包括广告，广告可能由位于很多独立服务器上的多个提供商发放。当 HTTP 客户端解析 HTML 代码时，发现了对其他服务器上资源的引用，为了获取引用的内容，客户端将尝试查询相关的服务器，这导致了额外的 DNS 查询和 HTTP 请求。这就是在 Pete 访问 ESPN 时发生的事情。在他想要查看来自于单个源的内容时，HTML 代码中引用了其他源的内容，之后，他的浏览器自动从其他多个域名请求内容。

明白了这些额外的请求是如何产生的，接下来，我们将逐个检查与每个请求相关的会话（**Statistics** > **Conversations**）。图 10-4 所示的会话窗口为我们提供了重要线索。

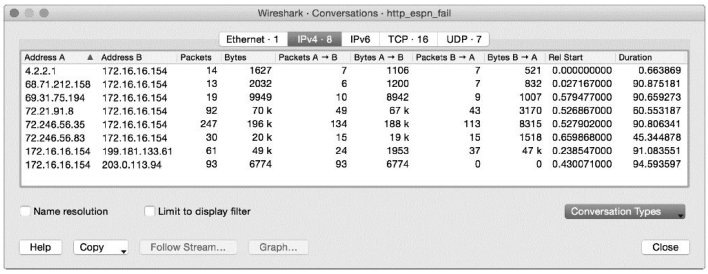


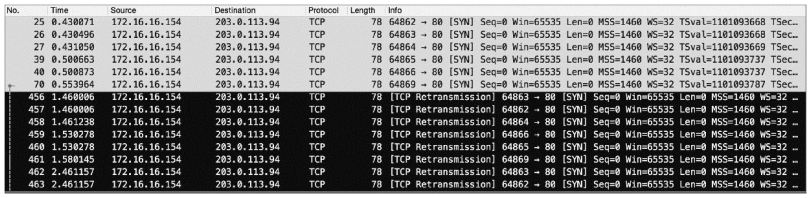
图 10-4　查看 IP 间会话

我们之前发现浏览器进行了 7 次 DNS 请求，以及相应的 7 次 HTTP 请求。在这个前提下，我们可以推断，有 7 个对应的 IP 间会话，然而，实际情况并非如此。这里出现了 8 个会话。应该如何解释这个情况？

一种可能性是，抓包文件被一个与本例中问题无关的会话「污染」。确保分析工作不被无关的通信扰乱是很重要的，但是这并不是这个会话中出现问题的原因。如果仔细检查每一个 HTTP 请求，并留意请求的目的 IP 地址，你会发现存在一个会话，没有对应的 HTTP 请求。这个会话的双方为 Pete 的计算机（172.16.16.154）和远程 IP 203.0.113.94。这个会话在图 10-4 的最后一行被列出。我们注意到，Pete 的计算机发送了 6774 字节至未知服务器（203.0.113.94），但是收到了 0 字节，这值得我们去深究。

现在，筛选并查看这个会话（在会话上右击，并选择**Apply As Filter** > **Selected** > **A<->B**），然后，运用 TCP 的知识来确定错误位置。

在正常的 TCP 通信中，我们期望看到一个标准的 SYNSYN/ACK-ACK 握手序列。在本例中，Pete 的计算机向 203.0.113.94 发送了一个 SYN 数据包（见图 10-5），但是我们没有看到 SYN/ACK 响应。不仅如此，Pete 的计算机还向不可用的目的服务器发送了多个 SYN 数据包，最终导致了他的机器发送 TCP 重传数据包。我们将在第 11 章中具体讨论 TCP 重传，此处的关键是，一个主机发送了数据包，但是从未接收到响应。查看时间栏，我们发现，重传持续了 95s，并且没有任何响应。在网络通信中，这堪称龟速。



No.	Time	Source	Destination	Protocol	Length	Info
25	0.430071	172.16.16.154	203.0.113.94	TCP	78	64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093668 TSec…
26	0.430496	172.16.16.154	203.0.113.94	TCP	78	64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093668 TSec…
27	0.431050	172.16.16.154	203.0.113.94	TCP	78	64864 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093669 TSec…
39	0.500663	172.16.16.154	203.0.113.94	TCP	78	64865 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093737 TSec…
40	0.500873	172.16.16.154	203.0.113.94	TCP	78	64866 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093737 TSec…
70	0.553964	172.16.16.154	203.0.113.94	TCP	78	64869 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093787 TSec…
456	1.460006	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
457	1.460006	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
458	1.461238	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64864 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
459	1.530278	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64866 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
460	1.530278	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64865 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
461	1.580145	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64869 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
462	2.461157	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
463	2.461157	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …

图 10-5　查看非预期的连接

我们识别了 7 次 DNS 请求、7 次 HTTP 请求和 8 个 IP 间会话。在确定抓包文件并没有被额外数据污染的情况下，我们有理由相信，神秘的第 8 个 IP 间会话可能是 Pete 加载网页缓慢并最终失败的原因。由于某些原因，Pete 的计算机试图与一个不存在或未提供服务的设备进行通信。为了了解这个情况出现的原因，我们不会查看抓包文件中的数据包；相反，我们会考虑文件之外的内容。

当 Pete 浏览 ESPN 网页时，他的浏览器识别了其他域名上的内容。为了获取这些内容数据，他的计算机发起 DNS 查询来获得这些域名的 IP 地址，之后，向这些目标 IP 发起 TCP 连接，从而发送 HTTP 请求并获取网页内容。对于与 203.0.113.94 的会话，并没有对应的 DNS 请求。那么，Pete 的计算机是怎样获得这个地址的呢？

如果你记得我们在第 9 章中关于 DNS 的讨论，或是熟悉 DNS，那么你就会明白，大部分系统使用一种 DNS 缓存机制。这一机制在系统内建立本地的域名与 IP 地址间的映射缓存，当访问已经缓存 DNS 记录、经常访问的域名时，系统将使用本地的映射缓存，而不发起 DNS 请求。直至这些域名

与 IP 地址间的映射过期，才进行新的 DNS 请求以获得域名的 IP 地址。然而，如果域名与 IP 的映射关系变更了，但是某个设备并没有发起新的 DNS 请求以获得新的地址，那么，在下次访问这一域名时，该设备将尝试连接一个无效的地址。

在 Pete 的例子中，就出现了这种情况。Pete 的计算机缓存了某一个为 ESPN 网站提供内容的主机的域名解析信息。由于缓存信息的存在，因此获取页面内容时没有进行 DNS 请求，这导致他的系统试图继续连接旧的地址。然而，这个地址已经不再为提供 HTTP 服务。作为结果，请求超时，并且内容加载失败。

对于 Pete 而言，值得庆幸的是，手动清除 DNS 缓存，只需要在命令行中敲击很短的命令即可。或者，他可以等待几分钟，在 DNS 缓存记录过期、新的 DNS 请求完成后，再尝试访问 ESPN 网站。