Squid 中文权威指南

（第 10 章）

译者序：

本人在工作中维护着数台 Squid 服务器，多次参阅 Duane Wessels（他也是 Squid 的创始人）的这本书，原书名是"Squid: The Definitive Guide"，由 O'Reilly 出版。我在业余时间把它翻译成中文，希望对中文 Squid 用户有所帮助。对普通的单位上网用户，Squid 可充当代理服务器；而对 Sina,NetEase 这样的大型站点，Squid 又充当 WEB 加速器。这两个角色它都扮演得异常优秀。窗外繁星点点，开源的世界亦如这星空般美丽，而 Squid 是其中耀眼的一颗星。

对本译版有任何问题，请跟我联系，我的Email是：yonghua_peng@yahoo.com.cn

彭勇华

目 录

第 10 章 与其他 Squid 会话

10.1 某些术语

通常把一组互相转发请求的 cache（或代理）叫做 cache 堆叠。把 cache 堆叠的成员叫做邻居或对等伙伴。

邻居 cache 有 2 种关系：父子或姐妹。从拓扑上看，父 cache 在堆叠里位于顶层，而姐妹 cache 位于同一层。两者真正的不同在于，父 cache 能为子 cache 转发 cache 丢失，然而姐妹 cache 之间不允许转发 cache 丢失。这意味着，在发送请求到姐妹 cache 前，发起者应该知道这是个 cache 命中。类似于 ICP,HTCP 和 Cache Digests 之类的堆叠协议，能预知邻居的 cache 命中。然而 CARP 不能这样。

某些时候，cache 堆叠并非真正层次性的。例如，考虑 1 组 5 个姐妹 cache。因为它们没有父子关系，故而不存在上或下。在这种情况下，你可以叫它 cache 结网，或甚至 cache 编队，而不是堆叠。

10.2 为何要（或不要）使用堆叠？

邻居 cache 通过提供某些额外的 cache 命中而改进了执行性能。换句话说，某些请求的 cache 丢失，在邻居 cache 里可能会命中。假如你的 cache 从邻居 cache 里下载这些命中，比从原始服务器快，那 cache 堆叠可全面改善性能。底线是邻居 cache 能提供一小部分的请求命中。大约 5%或 10%（假如幸运的话）的 cache 丢失请求，会在邻居 cache 中命中。在某些情况下，这点小作用价值不高。然而有的情形下，例如网络连接不尽人意时，cache 堆叠非常明显的改进了终端用户的访问性能。

假如在有防火墙的网络中使用 squid，就有必要配置防火墙作为父代理。在该情形中，squid 转发每个请求到防火墙，因为它不直接连接到外部原始服务器。假如有某些原始服务器在防火墙里面，就可以配置 squid 直接连接到它们。

也可以使用堆叠来在不同方向上传送 web 数据。这点有时候叫做应用层路由，或更近的叫法：内容路由。例如，考虑某个大公司，它有 2 个 Internet 连接。也许第二个连接比第一个成本更低。该公司可能想利用第二个连接作为次优先性传输，例如下载游戏、音频或视频，或其他形式的大文件传输。或者，也许他们想利用某条链路进行 HTTP 访问，而在另一条链路上进行非 HTTP 访问。再或者，也许某些用户的数据通过低优先级链路传输，而付费用户的数据通过更贵的链路传输。可以使用 cache 代理堆叠来完成上述设想。

信任机制对 cache 堆叠来说，是最重要的因素。你必须信任邻居 cache 会服务正确的、未修改的响应。必须在敏感信息上信任它们，例如用户请求的 URI。它们会维护一个安全的，不断更新的系统，将未授权访问和拒绝服务的机会降至最低。

关于堆叠的另一个问题，在于它们正常传播错误的途径。当邻居 cache 经历了某个错误，例如服务不可到达，它产生一个 HTML 页面来解释该错误及错误源头。假如邻居 cache 位于公司之外，它返回错误时，用户会觉得迷惑。假如该问题继续，用户将很难找到帮它们解决问题的管理员。

姐妹关系也有一个已知的问题，叫做假命中。假如 squid 发送某个请求到它的姐妹，它认为这是个 cache 命中，然而它的姐妹没有联系过原始服务器，因此不能满足该请求，这时就会发生假命中。有很多情况导致假命中，但通常可能性很低。甚至，squid 和其他 HTTP 代理有自动重新请求假命中的功能，不会让用户感觉到已发生的问题。

转发循环有时候是 cache 堆叠的另一个问题。当 squid 转发某个请求到某处，但该请求又被转发回 squid，这就发生了转发循环。请见图 10-1（略图）。

转发循环典型的发生在 2 个 cache 互相把对方当作父 cache 的情况。假如你遇到这样的问题，请确认使用 cache_peer_access 指令来阻止这类循环。例如，假如邻居 cache 的 IP 地址是 192.168.1.1，下面的行让 squid 不会产生转发循环：

```
acl FromNeighbor src 192.168.1.1
cache_peer_access the.neighbor.name deny FromNeighbor
```

转发循环在 HTTP 拦截里也能发生，特别是当拦截设备位于 squid 和原始服务器之间的路径上时。

Squid 通过检查 Via 头部里的主机名，来检测转发循环。假如 2 个协作 cache 有相同的主机名，实际上就会得到假转发循环。在该情形下，unique_hostname 指令很有用。注意，假如 Via 头部被过滤掉了（例如使用 headers_access 指令），squid 就不能检测到转发循环。

10.3 配置 Squid 与邻居通信

cache_peer 指令定义邻居 cache，并告诉 squid 如何与它的邻居通信：

```
cache_peer hostname type http-port icp-port [options]
```

第 1 个参数是邻居的主机名，或 IP 地址。可以安全的在这里使用主机名，因为 squid 会解析它们。在 squid 运行期间，主机名对应的 IP 地址可能会改变，所以实际上 squid 会周期性的解析主机名。邻居主机名必须唯一：不能在 2 个邻居 cache 上使用同样的主机名，即使它们有不同的端口。

第 2 个参数指定邻居 cache 的类型。有 3 个选择：父亲，姐妹，或广播。父亲和姐妹关系容易理解。我在 10.6.3 节里会详细谈到广播。

第 3 个参数是邻居 HTTP 端口号。它应该等同于邻居的 http_port 设置。总是应该指定 1 个

非零的 HTTP 端口号。

第 4 个参数指定 ICP 或 HTCP 端口号。squid 默认使用 ICP 来查询其他 cache。也就是说，squid 发送 ICP 查询到邻居 cache 的指定端口。假如你增加了 htcp 选项，squid 就会发送 HTCP 查询到这个端口。默认的 ICP 端口是 3130，默认的 HTCP 端口是 4827。假如增加了 htcp 选项，请记得改变它的端口号。将端口号设为 0，会禁止 ICP 和 HTCP。然而，应该使用 no-query 选项来禁止这些协议。

10.3.1 cache_peer 选项

cache_peer 指令有很多选项。我在这里描述其中的一些，其他的参数在与之相关的协议的章节里有描述。

proxy-only
该选项指示 squid 不存储它接受自邻居的任何响应。如果你有 cache 集群，并且不希望资源存储在多个 cache 上，那么该选项有用。

weight=n
该选项指定给 ICP/HTCP。请见 10.6.2.1 章节。

ttl=n
该选项指定给广播 ICP。见 10.6.3 节。

no-query
该选项指定给 ICP/HTCP。见 10.6.2.1 节。

default
在缺少其他线索时，该选项指定邻居 cache 作为适当的选择。正常情况下，squid 将 cache 丢失转发给父 cache，父 cache 看起来有特定资源的缓存拷贝。有时候 squid 没有任何线索（例如使用 no-query 禁用了 ICP/HTCP），这时 squid 会寻找父 cache，并将其作为默认选择。

round-robin
该选项是简单的负载共享技术。仅仅当你指定了 2 个或多个父 cache 作为轮转时，它才有用。squid 对每个父 cache 维持一个计数器。当需要转发 cache 丢失时，squid 选择计数器值最低的父 cache。

multicast-responder
该选项指定给广播 ICP。见 10.6.3 节。

closest-only
该选项指定给 ICP/HTCP。见 10.6.2.1 节。

no-digest
该选项指定给 cache digest，见 10.7 章。

no-netdb-exchange
该选项告诉 squid，不要请求邻居 cache 的 netdb 数据库（见 10.5 节）。

no-delay
该选项告诉 squid，忽略任何对邻居 cache 请求的延迟池设置。见附录 C 关于延迟池的更多信息。

login= credentials
该选项指示 squid 发送 HTTP 验证信息到邻居 cache。它有 3 个不同的格式：

login =user:password
这是最通用的形式。它导致 squid 在每个到邻居 cache 的请求里，增加相同的用户名和密码。用户无须填写任何验证信息。

login=PASS
将值设为 PASS，导致 squid 让用户的验证信息直接 pass 到邻居 cache。它仅仅工作在 HTTP 基础的验证机制下。squid 不会增加或修改任何验证信息。

假如 squid 被配置成需要代理验证（例如使用 proxy_auth ACL），邻居 cache 就必须使用同样的用户名和密码数据库。换句话说，应该只在被同一组织拥有和操作的一组 cache 中使用 PASS 形式。该功能是危险的，因为 squid 不会从转发请求里移除验证信息。

login =* :password
使用该形式，squid 改变它转发的请求里的密码，但不改变用户名。它允许邻居 cache 来验证独立的用户，但不暴露其密码。该形式比使用 PASS 危险少一些，但确实有一些隐私牵连。

使用该功能需要额外小心。即使你不管隐私条目，该功能也可能对上级代理产生不良的副作用。例如，我知道至少 1 个其他的 cache 产品，它仅仅查看持续连接里第一个请求的信用信息。它看起来（不正确的）假设在单个连接里的所有请求，都来自同一用户。

connect-timeout= n
该选项指定，在 squid 建立 TCP 连接到邻居 cache 时，等待的超时时间。若没有该选项，超时值就取自全局 connect_timeout 指令，它的默认值是 120 秒。使用低的超时值，squid 会迅速放弃到邻居 cache 的会话，并且试着发送请求到其他邻居 cache，或直接请求原始服务器。

digest-url= url
该选项指定给 cache digest。见 10.7 章。

allow-miss
该选项指示 squid 忽略 Cache-Control: only-if-cached 指令，该指令是针对发送给姐妹 cache 的请求的。仅在如下情况下使用它：邻居 cache 激活了 icp_hit_stale 指令，并且没有使用 miss_access 列表。

max-conn= n
该选项对 squid 打开到邻居 cache 的同时连接的数量进行限制。当抵达了该限制时，squid 从它的选择算法里排除掉该邻居 cache。

htcp
该选项指定邻居 cache 为 HTCP 服务器。换句话说，squid 发送 HTCP 查询，而不是 ICP 查询，到邻居 cache。注意 squid 不会在同一端口上接受 ICP 和 HTCP 查询。若你增加了该选项，不要忘了改变 icp-port 的值。见 10.8.1 节。HTCP 支持需要在运行./configure 时，使用 --enable-htcp 选项。

carp-load-factor= f
该选项让邻居 cache（它必须是父 cache）成为 CARP 的成员。对所有父 cache 的 f 值的总和，必须等于 1。我在 10.9 章里讲到 CARP。CARP 支持需要在运行./configure 时，使用--enable-carp 选项。

10.3.2 邻居状态

Squid 对其每个邻居 cache，维持着多种统计和状态信息。最重要的信息之一，是其邻居 cache 的存活状态。邻居 cache 的存活/死亡状态影响了 squid 选择算法的许多方面。确定邻居 cache 的存活/死亡状态的机制有点复杂，所以我在这里不讲它。假如你想通过阅读源代码来了解这点，请查看 neighborUp()函数。

Squid 使用 TCP(HTTP)和 UDP(ICP/HTCP)通讯来确定邻居 cache 的状态。TCP 状态默认是存活的，但如果连续 10 个 TCP 连接失败，状态会改变为死亡。如果发生这种事，squid 会发起探查连接，在每个 connect_timeout 期间内（全局指令，并非 cache_peer 选项），连接不会超过 1 次。邻居 cache 会保持死亡状态，直到探查连接之一成功。

假如 no-query 选项没有设置（意味着 squid 正发送 ICP/HTCP 查询到邻居 cache），UDP 层通讯也参与到存活/死亡算法里来。UDP 状态默认是存活的，但假如 squid 在超时范围内（dead_peer_timeout 指令的值），没有接受到任何 ICP/HTCP 响应，UDP 状态就会改变为死亡。

假如邻居 cache 的主机名不可解析，squid 也会将其标记为死亡。当 squid 确定其邻居 cache 死亡时，它在 cache.log 里增加类似如下的日志记录：

2003/09/29 01:13:46| Detected DEAD Sibling: bo2.us.ircache.net/3128/3130

当与邻居 cache 的通信重新建立起来时，squid 记录类似如下的日志：

2003/09/29 01:13:49| Detected REVIVED Sibling: bo2.us.ircache.net/3128/3130

邻居 cache 的状态，在下述几个方面影响邻居选择算法：

1)Squid 不希望接受来自死亡邻居的 ICP/HTCP 响应。squid 在每个 dead_peer_timeout 期间内，

只发送 1 次 ICP 查询到死亡的邻居。见附录 A。

2)死亡的父 cache 从下列算法里排除掉:Digests, round-robin parent, first up parent, default parent, 和 closest parent。

3)CARP 特殊：任何失败的 TCP 连接，会从 CARP 算法里排除掉父 cache。

没有办法强迫 squid 发送 HTTP 请求到死亡的邻居 cache。假如所有的邻居 cache 都死亡，squid 会试图连接到原始服务器。假如你不允许 squid 与原始服务器会话（使用 never_direct 指令），squid 会返回 cannot forward 的错误消息：

This request could not be forwarded to the origin server or to any
parent caches. The most likely cause for this error is that:

* The cache administrator does not allow this cache to make
  direct connections to origin servers, and
* All configured parent caches are currently unreachable

10.3.3 改变关系

neighbor_type_domain 指令允许你改变与基于原始服务器主机名的邻居 cache 的关系。假如邻居 cache 期望服务任何请求的 cache 命中，但仅仅服务某些临近域的 cache 丢失，那么这个指令就很有用。语法是：

neighbor_type_domain neighbor.host.name relationship [!]domain ...

例如：

cache_peer squid.uk.web-cache.net sibling 3128 3130
neighbor_type_domain squid.uk.web-cache.net parent .uk

当然，该示例里的 squid.uk.web-cache.net 缓存，会利用相应的 miss_access 规则来强迫姐妹关系给 non-UK 请求。注意域名会按照 6.1.1.2 章节里描述的那样，匹配主机名。

10.4 对邻居的请求限制

许多使用 cache 堆叠的用户，想控制或限制 squid 发送到邻居 cache 的请求。squid 有 7 个不同的指令可以影响请求路由： cache_peer_access, cache_peer_domain, never_direct, always_direct, hierarchy_stoplist,nonhierarchical_direct, 和 prefer_direct。

10.4.1 cache_peer_access

cache_peer_access 指令定义对邻居 cache 的访问列表。也就是说，它决定哪个请求允许或不允许发送到邻居 cache。

例如，可以使用这点来对 FTP 和 HTTP 请求进行分流。你可以发送所有的 FTP URI 到某个父 cache，所有的 HTTP URI 到另一个父 cache：

```
cache_peer A-parent.my.org parent 3128 3130
cache_peer B-parent.my.org parent 3128 3130
acl FTP proto FTP
acl HTTP proto HTTP

cache_peer_access A-parent allow FTP
cache_peer_access B-parent allow HTTP
```

改配置确保父 cache A 仅接受 FTP URI 请求，而父 cache B 仅接受 HTTP URI 请求。当然也包括 ICP/HTCP 请求。

也可以使用 cache_peer_access，在一天中的某段时间来激活或禁止邻居 cache：

```
cache_peer A-parent.my.org parent 3128 3130
acl DayTime time 07:00-18:00
cache_peer_access A-parent.my.org deny DayTime
```

10.4.2 cache_peer_domain

cache_peer_domain 指令是 cache_peer_access 指令的早期形式。相对于使用完整的访问控制特性，它仅使用 URI 里的域名。它常用于通过域名区分一组父 cache。例如，假如你有一个遍布全球的内部网，你也许想发送请求到位于各自大陆的 cache：

```
cache_peer europe-cache.my.org parent 3128 3130
cache_peer asia-cache.my.org      parent 3128 3130
cache_peer aust-cache.my.org      parent 3128 3130
cache_peer africa-cache.my.org parent 3128 3130
cache_peer na-cache.my.org         parent 3128 313
cache_peer sa-cache.my.org         parent 3128 3130

cache_peer_domain europe-cache.my.org parent .ch .dk .fr .uk .nl .de .fi ...
cache_peer_domain asia-cache.my.org parent      .jp .kr .cn .sg .tw .vn .hk ...
cache_peer_domain aust-cache.my.org parent      .nz .au .aq ...
cache_peer_domain africa-cache.my.org parent .dz .ly .ke .mz .ma .mg ...
cache_peer_domain na-cache.my.org parent         .mx .ca .us ...
```

cache_peer_domain sa-cache.my.org parent .br .cl .ar .co .ve ...

当然，该机制不匹配流行的全球顶级域名，例如.com。

10.4.3 never_direct

never_direct 指令是对从来不必直接发送到原始服务器的请求的访问列表。当请求匹配该访问列表，它必须被发送到邻居 cache（通常是父 cache）。

例如，假如 squid 位于防火墙之后，它也许能够直接与内部服务器会话，但必须通过防火墙代理（父 cache）发送所有的请求到外部服务器。你可以告诉 squid：“永不要直接连到防火墙之外的站点”。为了做到这点，请告诉 squid 有什么在防火墙之内：

```
acl InternalSites dstdomain .my.org
never_direct allow !InternalSites
```

语法有点奇怪。never_direct allow foo 意味着 squid 不会直接放过匹配 foo 的请求。既然内部站点容易指定，我使用否操作符(!)来匹配外部站点，这些站点 squid 不会直接联系。

注意该示例不会强迫 squid 直接连接到匹配 InternalSites ACL 的站点。never_direct 访问规则仅仅强迫 squid 不直接联系某些原始服务器。你必须使用 always_direct 规则来强迫直接连接到原始服务器。

在结合其他控制请求路由的指令来使用 never_direct 时，必须谨慎。可能很容易就建立一件不可能做到的事件。如下是个示例：

```
cache_peer A-parent.my.org parent 3128 3130
acl COM dstdomain .com
cache_peer_access A-parent.my.org deny COM
never_direct allow COM
```

该配置自相矛盾，因为任何域名以.com 结尾的请求必须通过邻居 cache。然而仅仅定义了一个邻居 cache，并且不允许.com 请求通过它。若发生这种情况，squid 会发布“cannot forward”的错误消息。

10.4.4 always_direct

也许你已猜到，always_direct 规则列表告诉 squid 某些请求必须直接转发到原始服务器。例如，许多单位想保持他们的内网通信本地化。容易做到的方法是，定义基于 IP 地址的 ACL，并将它放入 always_direct 规则列表：

```
acl OurNetwork src 172.16.3.0/24
always_direct allow OurNetwork
```

10.4.5 hierarchy_stoplist

Squid 内在的将每个客户端请求标记为层叠或不可层叠。不可层叠的请求看起来不会导致 cache 命中。例如，POST 请求的响应几乎从不会被 cache。在 squid 能简单的连接到原始服务器时，转发不可 cache 目标的请求到邻居 cache，纯粹是浪费资源。

某些区分层叠和不可层叠请求的规则，在 squid 里难于编码。例如，POST 和 PUT 方式总是不可层叠的。然而，hierarchy_stoplist 指令允许你定制这种算法。它包含一个字符串列表，当在 URI 里发现它们时，squid 将请求标记为不可层叠。默认的该列表是：

hierarchy_stoplist ? cgi-bin

这样，任何包含问号或 cgi-bin 字符串的请求匹配该列表，变成不可层叠。

默认的，squid 直接发送不可层叠的请求到原始服务器。因为这些请求不会导致 cache 命中，它们通常是邻居 cache 的额外负担。然而，never_direct 访问控制规则凌驾于 hierarchy_stoplist 规则之上。具体而言，squid 这样做：

1)从不对不可层叠的请求发送 ICP/HTCP 查询，除非该请求匹配 never_direct 规则；
2)从不对不可层叠的请求发送 ICP/HTCP 查询到姐妹 cache；
3)从不对不可层叠的请求查询邻居的 cache digest。

10.4.6 nonhierarchical_direct

该指令控制 squid 转发不可层叠的请求的方法。squid 默认直接发送不可层叠的请求到原始服务器。这是因为这些请求不会导致 cache 命中。我觉得直接从原始服务器获取它们总是好的，而不要浪费时间在邻居 cache 上查询它们。假如因为某些理由，你想路由这些请求通过邻居 cache，请禁止该指令：

nonhierarchical_direct off

10.4.7 prefer_direct

该指令控制 squid 转发层叠请求的方法。默认的，squid 首先发送这样的请求到邻居 cache，然后再到原始服务器。通过激活该指令，你能反转这种行为：

prefer_direct on

这样，假如 squid 与原始服务器的通信失败，邻居 cache 就变成了备份。

10.5 网络度量数据库（netdb）

Squid 的网络度量数据库（netdb）被设计来测量到原始服务器的远近。换句话说，通过查询该数据库，squid 知道它离原始服务器有多远。该数据库包含 ICMP 往返时间（RTT）测量值和路由跳计数。正常情况下，squid 仅使用 RTT 测量值，但在某些情形下也使用路由跳计数。

为了激活 netdb，必须使用--enable-icmp 选项来配置 squid。也必须以超级用户权限来安装 pinger 程序，请见 3.6 章的描述。当运行正确后，可以在 cache.log 里见到类似如下的消息：

2003/09/29 00:01:03| Pinger socket opened on FD 28

当激活了 nebdb 时，squid 发送 ICMP ping 到原始服务器。ICMP 消息实际上由 pinger 程序来发送和接受，pinger 以 root 运行。squid 会小心的不至于频繁发送 ping 消息，以避免骚扰 web 站点管理员。默认的，squid 在发送另一个 ping 到同一主机，或同一 24 位子网中主机时，会至少等待 5 分钟。可以使用 netdb_ping_period 指令来调整这个时间间隙。

ICMP ping 消息通常非常小（少于 100 字节）。squid 在 ICMP 消息的有效载荷里包含了原始服务器的主机名，紧跟一个时间戳。

为了减少内存需求，squid 以 24 位子网来合计 netdb 数据。squid 假设所有位于同一 24 位子网里的主机，有相似的 RTT 和路由跳计数。若某台新的原始主机所在子网的其他主机已被测量过，该机制也允许 squid 去估算这台新原始主机的远近。

随同 RTT 和路由跳计数，squid 也维护着一份主机名结合子网的列表。典型的记录看起来类似如下：

Subnet 140.98.193.0

RTT 76.5

Hops 20.0

Hosts services1.ieee.org
www.spectrum.ieee.org
www.ieee.org

netdb 度量值主要被 ICP 和 HTCP 使用。当你在 squid.conf 里激活 query_icmp 指令时，squid 在发送到邻居 cache 的 ICP/HTCP 查询里设置一个标记。该标记请求在 ICP/HTCP 响应里包含远近测量值。假如邻居 cache 也激活了 netdb，它们的响应将包含 RTT 和路由跳计数（假如可用）。注意 squid 总是立刻响应 ICP。在响应 ICP 查询前，它不会等待 ICMP 测量。见 10.6.2.2 节关于 ICP 如何使用 netdb 的细节。

Squid 会记住它从 ICP/HTCP 响应里学到的 RTT 值。这些值在随后的决定最佳转发途径时，也许会用到。通过叫做 netdb 交换的机制，squid 也支持 netdb 度量值的批量迁移。squid 周期性的通过 HTTP 请求邻居 cache 的 netdb 数据。在 cache_peer 行里设置 no-netdb-exchange 选项，就可以禁止这些请求。

netdb_low 和 netdb_high 指令控制度量数据库的大小。当存储子网的数量抵达 netdb_high 时，squid 删除最少近来使用的条目，直到数量低于 netdb_low。

minimum_direct_hops 和 minimum_direct_rtt 指令指示 squid 直接连接到低于一定数量路由跳计数，或少于一定毫秒的原始服务器。匹配该标准的请求在 access.log 里以 CLOSEST_DIRECT 形式记载。

cache 管理器的 netdb 页显示完整的网络度量数据库，包括来自邻居 cache 的值。例如：

Network DB Statistics:

```
Network              recv/sent     RTT  Hops Hostnames

63.241.84.0          1/    1      25.0   9.0 www.xyzzy.com
     sd.us.ircache.net             21.5  15.0
     bo1.us.ircache.net            27.0  13.0
     pb.us.ircache.net             70.0  11.0

206.100.24.0         5/    5      25.0   3.0 wcarchive.cdrom.com ftp.cdrom.com
     uc.us.ircache.net             23.5  11.0
     bo1.us.ircache.net            27.7   7.0
     pb.us.ircache.net             35.7  10.0
     sd.us.ircache.net             72.9  10.0

146.6.135.0          1/    1      25.0  13.0 www.cm.utexas.edu
     bo1.us.ircache.net            32.0  11.0
     sd.us.ircache.net             55.0   8.0

216.234.248.0        2/    2      25.0   8.0 postfuture.com www1.123india.com
     pb.us.ircache.net             44.0  14.0

216.148.242.0        1/    1      25.0   9.0 images.worldres.com
     sd.us.ircache.net             25.2  15.0
     bo1.us.ircache.net            27.0  13.0
     pb.us.ircache.net             69.5  11.0
```

这里，你可以见到 www.xyzzy.com 服务器有一个 IP 地址位于 63.241.84.0/24 子网。从 cache 到这台原始服务器的 RTT 值是 25 毫秒。邻居 cache: sd.us.ircache.net 更近一点，在 21.5 毫秒。

10.6 Internet Cache 协议（ICP）

ICP 是轻量级的目标定位协议，它作为 Harvest 项目的一部分而被发明。ICP 客户发送查询消息到一个或多个 ICP 服务器，询问它们是否缓存了某个 URI。每个服务器响应一个 ICP_HIT（ICP 命中），ICP_MISS（ICP 丢失），或其他类型的 ICP 消息。ICP 客户使用 ICP 响应里的信息来做转发决定。

除了指示 cache 命中，ICP 也用于提供关于 squid 和邻居 cache 之间网络条件的线索。从这点看，ICP 消息类似于 ICMP ping。通过计算查询/响应往返时间，squid 能估算网络拥塞情况。在极端情况下，ICP 消息可能丢失，这意味着两者之间的链路断掉，或线路拥塞。在这种情况下，squid 会避免请求这个邻居 cache。

增加延时可能是使用 ICP 的最显然的弊端。查询/响应交换要花费一些时间。cache 代理被设计来减少响应时间，而不是增加延时。如果 ICP 帮助我们发现在邻居 cache 上的 cache 命中，这样它应该全面减少响应时间。见 10.10 章关于 squid 的查询算法实现的描述。

ICP 也得承受某些设计不足带来的责难：安全性，伸缩性，假命中，和请求方式的缺乏。该协议不包括任何安全机制。通常 squid 不能确认某个 ICP 消息是可信的；它依赖于基于地址的访问控制来过滤掉不想要的 ICP 消息。

ICP 的伸缩性也很差。ICP 消息的数量（和带宽）增长，与邻居 cache 的数量成正比。除非使用某种隔离机制，这实际上限制了你能使用的邻居 cache 的数量。我不推荐拥有超过 5 或 6 个邻居 cache。

ICP 查询仅包含 URI，没有另外的请求头部。这让它很难精确的指示 cache 命中。某个 HTTP 请求可能包含附加的头部（例如 Cache-Control: max-stale=N）将 cache 命中转换为 cache 丢失。对姐妹关系的 cache 来说，这些假命中的弊端特别明显。

从 ICP 查询消息里丢失的还有请求方式。ICP 假设所有的查询采用 GET 请求。cache 代理不能使用非 GET 请求方式的 ICP 来查找缓存目标。

通过阅读如下资料，你可以找到其他的关于 ICP 的信息：

1)我写的书： Web Caching (O'Reilly)
2)RFCs 2186 and 2187
3)我的论文："ICP and the Squid Web Cache" in the IEEE Journal on Selected Areas in Communication, April 1998
4) http://icp.ircache.net/

10.6.1 成为 ICP 服务器

当你使用 icp_port 指令时，squid 自动成为 ICP 服务器。也就是说，它在你指定的端口侦听 ICP 消息，或默认的 3130 端口。假如决定使用非标准端口，别忘了告知姐妹和/或子 cache。

squid 默认拒绝所有 ICP 查询。必须使用 icp_access 规则列表，允许来自邻居 cache 的查询。使用 src ACL 容易做到这点。例如：

```
acl N1 src 192.168.0.1
acl N2 src 172.16.0.2
acl All src 0/0

icp_access allow N1
icp_access allow N2
icp_access deny All
```

注意仅仅 ICP_QUERY 消息从属于 icp_access 规则。ICP 客户端的功能，例如发送查询和接受响应，不需要任何特殊访问控制。我也推荐使用操作系统的包过滤功能（例如 ipfw,iptables, 和 pf）。允许来自可信任邻居的 UDP 消息到 ICP 端口，并拒绝所有其他的。

如果 squid 因为 icp_access 规则而拒绝某个 ICP 查询，它返回一个 ICP_DENIED 消息。然而，假如 squid 检测到超过 95%的近来查询被拒绝，它停止响应 1 个小时。若发生这种情况，squid 在 cache.log 里写如下消息：

```
WARNING: Probable misconfigured neighbor at foo.web-cache.com
WARNING: 150 of the last 150 ICP replies are DENIED
WARNING: No replies will be sent for the next 3600 seconds
```

假如你看到这样的消息，你该联系错误配置了 cache 的管理员。

squid 被设计为迅速响应 ICP 查询。也就是说，squid 通过检查内存索引，能告知它是否有更新的、缓存的响应。这也就是为什么 squid 如此耗内存的原因。当 ICP 查询进来时，squid 计算 URI 的 MD5 hash 值，并且在索引里查找它。假如没有找到，squid 返回 ICP_MISS 消息。假如找到了，squid 检查超时时间。假如目标没有刷新，squid 返回 ICP_MISS。对刷新的目标，squid 返回 ICP_HIT。

默认的，squid 记录所有的 ICP 查询（但非响应）到 access.log。假如拥有许多繁忙的邻居 cache，日志文件可能变得太大而难于管理。使用 log_icp_queries 指令来阻止记录这些查询。尽管会丢失 ICP 的详细日志，你仍可以通过 cache 管理器来获取一些合计信息。

假如有姐妹邻居，你也许想使用 miss_access 指令来强迫其关系。它指定 cache 丢失的访问规则。它与 http_access 相似，但仅检查必须被转发的请求。默认的规则是允许所有的 cache 丢失。除非增加一些 miss_access 规则，任何姐妹 cache 都可能变成子 cache，并通过你的网络连接来转发 cache 丢失，这样就盗用了带宽。

miss_access 规则相对简单。不要忘记包含本地客户端（例如 web 浏览器）。如下是个简单示例：

```
acl Browsers src 10.9.0.0/16
acl Child1 src 172.16.3.4
acl Child2 src 192.168.2.0/24
acl All src 0/0

miss_access allow Browsers
miss_access allow Child1
miss_access allow Child2
miss_access deny All
```

注意我没有在这里列举任何姐妹 cache。子 cache 允许通过我们请求 cache 丢失，但姐妹 cache 不允许。它们的 cache 丢失请求被 deny all 规则拒绝。

10.6.1.1 icp_hit_stale 指令

ICP 的问题之一是，它对 cache 住的，但陈旧的响应返回 ICP_MISS。这点确实存在，即使响应陈旧但有效。考虑一个简单的堆叠，有 1 个子 cache 和 2 个父 cache。某个目标缓存在其中 1 个父 cache 上。缓存的响应是陈旧的，但未改变，需要确认。子 cache 的 ICP 查询导致 2 条 ICP_MISS 响应。由于不知道陈旧的响应存在于第 1 个父 cache 中，子 cache 会转发它的请求到第 2 个父 cache。现在目标存储在 2 个父 cache 中，浪费资源。

在该情形下，icp_hit_stale 指令有用。它告诉 squid 对任何 cache 住的目标，即使它是陈旧的，都返回 ICP_HIT。这在父子关系的 cache 中很安全，但对姐妹关系的 cache 有问题。

回想一下姐妹关系，客户 cache 仅仅允许发送 cache 命中的请求。激活了 icp_hit_stale 指令，增加了假命中的数量，因为 squid 必须确认陈旧响应。正常情况下，squid 这样处理假命中：在发送给姐妹 cache 的 HTTP 请求里增加 Cache-Control: only-if-cached 指令。假如姐妹 cache 不能满足 HTTP 请求为 cache 命中，它返回一个 HTTP 504（网关超时）消息。当 squid 接受到 504 响应，它再次转发请求到父 cache 或原始服务器。

假如必须转发所有的假命中，激活 icp_hit_stale 就会给姐妹关系 cache 带来麻烦。这时 ICP 客户端 cache_peer 的 allow-miss 选项就变得有用。当设置了 allow-miss 选项时，squid 忽略它发送到姐妹 cache 的 HTTP 请求里的 only-if-cached 指令。

假如激活了 icp_hit_stale，必须确保 miss_access 不会拒绝来自姐妹 cache 的 cache 丢失请求。不幸的是，没有办法让 squid 只允许 cache 住的，但陈旧的目标的 cache 丢失。允许姐妹 cache 的 cache 丢失请求，也让 squid 容易被滥用。姐妹 cache 的管理员可能没经过你的同意，而擅自改变它为父 cache。

10.6.1.2 ICP_MISS_NOFETCH 功能

命令行-Y 选项的作用是：在 squid 重建内存索引时，它导致 squid 返回 ICP_MISS_NOFETCH，

而不是 ICP_MISS。接受到 ICP_MISS_NOFETCH 响应的 ICP 客户，不会对这些目标发送 HTTP 请求。这减少了 squid 的负载，并让重建过程快速完成。

10.6.1.3 test_reachability 指令

假如激活了 netdb 功能（见 10.5 章），你也许会对 test_reachability 指令感兴趣。它背后的目的是，squid 仅接受某些请求，这些请求的原始服务器 squid 能访问到。激活了 test_reachability 指令，在原始服务器不响应 ICMP ping 时，squid 返回 ICP_MISS_NOFETCH，而不是 ICP_MISS。这点有助于减少假 HTTP 请求的数量，并增加终端用户迅速接受数据的机会。然而，一定比率的原始服务器站点过滤掉了 ICMP 传输。对这些站点，即使 HTTP 连接成功，squid 也会返回 ICP_MISS_NOFETCH。

激活 test_reachability 也导致 squid 在 ICP 查询的响应里，发起 netdb 度量。假如 squid 没有请求中的原始服务器的任何 RTT 度量值，它会发出一个 ICMP ping 消息。

10.6.2 成为 ICP 客户

首先，必须使用 cache_peer 指令来定义邻居 cache。见 10.3 章。

接着，必须使用 icp_port 指令，即使 squid 仅作为 ICP 客户。这是因为 squid 使用同样的 socket 来发送和接受 ICP 消息。这也许是个不好的设计思路。假如仅作为 ICP 客户，请使用 icp_access 来阻塞查询。例如：

```
acl All src 0/0
icp_access deny All
```

Squid 默认的对大多数请求发送 ICP 查询到其邻居。见 10.10 章关于 squid 决定何时查询其邻居 cache 的方法的完整描述。

在发送一个或多个查询后，squid 等待一定数量的时间，等候 ICP 响应抵达。假如 squid 从某个邻居 cache 接受到 ICP_HIT，它会立刻转发请求到那里。会则，squid 会一直等待，直到所有的响应抵达，或超时发生。squid 基于下列算法，动态的计算超时。

Squid 从最近的 ICP 传输中，知道从它自己到每个邻居 cache 之间的平均往返时间。在查询一组邻居时，squid 计算所有邻居 ICP RTT 的平均数，然后将该数值翻倍。换句话说，查询超时值是每个邻居 cache 查询的 RTT 的平均值的 2 倍。在计算超时值时，squid 忽略看起来已 down 掉的邻居。

在某些情形下，该算法不会工作良好，特别在邻居 cache 的 RTT 变化很大的情况下。使用 maximum_icp_query_timeout 指令可以改变超时的上限。另外，也可以通过 icp_query_timeout 指令，让 squid 使用一个常量超时值。

10.6.2.1 ICP 客户的 cache_peer 选项

weight=n 允许你在使用 ICP/HTCP 时，手工加权父 cache。它仅当所有父 cache 报告 cache 丢失时，才变得有用。正常情况下，squid 选择响应首先抵达的父 cache。实际上，squid 会记住查询中哪个父 cache 有最好的 RTT。squid 实际上通过权重来区分 RTT，所以 weight=2 的父 cache，不管其实际距离如何，都会被对待为离 squid 很近。

no-query 禁止对邻居 cache 的 ICP/HTCP。也就是说，你的 cache 不会发送任何对 cache 丢失的查询到邻居。它通常以 default 选项被使用。

closest-only 指 squid 的 netdb 功能之一。它指示 squid 仅基于 netdb RTT 度量值来选择父 cache，而不是响应抵达的顺序。该选项要求 netdb 在两端都被激活。

10.6.2.2 ICP 和 netdb

就像 10.5 章里提到的一样，netdb 主要用于 ICP 查询。在本节里，我们跟随在这个过程中，所有的步骤调用。

1．作为 ICP 客户的 squid cache，准备发送查询到一个或多个邻居 cache。假如设置了 query_icmp，squid 在 ICP 查询里设置 SRC_RTT 标记。这通知 ICP 服务器，squid 想在 ICP 响应里接受 RTT 度量值。

2．邻居 cache 接受到带有 SRC_RTT 标记的查询。假如邻居 cache 配置了使用 netdb 度量，它会在数据库里搜索原始服务器的主机名。注意邻居 cache 不会查询 DNS 来解析原始服务器的 IP 地址。这样，假如指定的主机已经被度量过，它才能在 netdb 里找到相关条目。

3．假如主机存在于 netdb 数据库里，邻居 cache 在 ICP 响应里返回 RTT 和路由跳数。并在响应里设置 SRC_RTT 标记，指示度量值已提供。

4．当 squid 接受到设置了 SRC_RTT 标记的 ICP 响应，它剥离出 RTT 和路由跳数。这些被追加到本地 netdb，以便将来 squid 了解从邻居 cache 到原始服务器的近似 RTT。

5．ICP_HIT 响应导致 squid 立刻转发 HTTP 请求。然而假如 squid 仅接受到 ICP_MISS 应答，它就选择具有到原始服务器的最小（非零）RTT 的父 cache。该请求以 CLOSEST_PARENT_MISS 被记录到 access.log。

6．假如没有父 cache 的 ICP_MISS 响应中包含 RTT 值，squid 选择 ICP 应答最先到达的父 cache。这时，请求以 FIRST_PARENT_MISS 被记日志。然而，假如父 cache 设置了 closest-only 选项，squid 就从不会选择它作为第一个父 cache。换句话说，仅当该父 cache 离原始服务器最近时，才会被选中。

10.6.3 广播 ICP

你已知道，ICP 的伸缩性差。消息的数量与邻居 cache 的数量成正比。因为 squid 发送同样的 ICP_QUERY 消息到每个邻居，你可以使用广播来减少在网络中传输的消息数量。相对于发送 N 条消息到 N 个邻居，squid 仅发送一个消息到广播地址。广播路由结构确保每个邻居能接受到消息的复制品。请参阅书籍：Interdomain Multicast Routing: Practical Juniper Networks and Cisco Systems Solutions by Brian M. Edwards, Leonard A. Giuliano, and Brian R. Wright (Addison Wesley) 关于内部网络广播的更多信息。

注意 ICP 应答总是通过单点传播发送。这是因为 ICP 应答可能不一样，并且单点传播和广播的路由拓扑不同。因为 ICP 也用于指示网络条件，故 ICP 应答应走 HTTP 应答所采取的同样路径。广播仅减少了查询的消息数量。

历史上，我发现广播结构不稳定和不可信。许多 ISP 以低优先级来对待广播。广播工作一天，某些事情可能在数天或数周后崩溃。在你自己的网络里使用广播可能是安全的，但我不推荐在公共 Internet 上使用广播 ICP。

10.6.3.1 广播 ICP 服务器

广播 ICP 服务器加入 1 个或多个广播组地址来接受消息。mcast_groups 指令指定这些组地址。它的值必须是广播 IP 地址，或可被解析到广播地址的主机名。IPv4 广播地址的范围是：224.0.0.0-239.255.255.255。例如：

```
mcast_groups 224.11.22.45
```

关于广播有意思的事情是主机，而不是应用，加入一个组。当某台主机加入广播组时，它接受到发送给该组的所有数据包。这意味着，在选择一个广播组用于 ICP 时，你必须小心一点。不能选择某个已被其他应用所使用的地址。若这种组交迭情况发生，两个组会串起来，并接受彼此的数据传输。

10.6.3.2 广播 ICP 客户

广播 ICP 客户发送查询到某个或多个广播组地址。这样，cache_peer 行的主机名参数，必须是（或能解析到）一个广播地址。例如：

```
cache_peer 224.0.14.1 multicast 3128 3130 ttl=32
```

HTTP 端口号（例如 3128）在该情形下是无关的，因为 squid 从来不会发起 HTTP 连接到广播邻居。

应该认识到，广播组没有任何访问控制。任何主机能加入任何的广播组地址。这意味着，除非足够小心，其他人也能接受到你的 squid 发送的广播 ICP 查询。IP 广播有 2 个方法来阻止

包传得太远：TTL 和管理范围。因为 ICP 查询可能携带敏感信息（例如用户访问的 URI），我推荐使用管理范围地址，并正确的配置路由器。见 RFC 2365 的更多信息。

ttl=n 选项仅针对广播邻居。它是用于 ICP 查询的广播 TTL 值。它控制 ICP 查询能传送多远。有效值范围是 0-128。较大的取值允许广播查询传送更远，并有可能被外面的人截获。使用较低的 TTL 值，可以保证查询不会离源头太远，并位于自己网络中。

广播 ICP 客户也必须告诉 squid 关于响应查询的邻居的情况。squid 不盲目信任任何发送 ICP 应答的 cache。你必须告诉 squid 合法的，可信任的邻居。cache_peer 指令的 multicast-responder 选项用以确定这样的邻居。例如，假如知道 172.16.2.3 是位于广播组里的可信任的邻居，你可以在 squid.conf 行里增加如下行：

cache_peer 172.16.3.2 parent 3128 3130 multicast-responder

当然也能使用主机名代替 IP 地址。来自外部邻居的 ICP 应答被忽略了，但记录在 cache.log 里。

正常情况下，squid 希望对其发送的每个请求都接受到 ICP 应答。在广播机制下情况会不同，因为每个查询会导致多个响应。为了统计这点，squid 周期性的对广播组地址发送探测消息。这些探测告诉 squid，多少服务器在侦听。squid 统计特定数量时间内抵达的应答数量。该时间数量由 mcast_icp_query_timeout 指令给定。然后，当 squid 发送真正的 ICP 查询到广播组时，它期望返回前述数量的 ICP 应答。

10.6.3.3 广播 ICP 示例

既然广播 ICP 有点棘手，如下是另一个示例。假设你的 ISP 有 3 个父 cache 侦听在广播地址，等待 ICP 查询。ISP 在其配置文件里仅需要一行：

mcast_groups 224.0.14.255

子 cache 的配置有点复杂。首先，必须列出 squid 能发送查询的广播邻居。也必须列出 3 个父 cache 的单点传送地址，以便 squid 能接受它们的应答：

```
cache_peer 224.0.14.225 multicast 3128 3130 ttl=16
cache_peer parent1.yourisp.net parent 3128 3130 multicast-responder
cache_peer parent2.yourisp.net parent 3128 3130 multicast-responder
cache_peer parent3.yourisp.net parent 3128 3130 multicast-responder
mcast_icp_query_timeout 2 sec
```

请记住，squid 从不会发起 HTTP 请求到广播邻居，也从不发送 ICP 查询到广播应答邻居。

10.7 Cache 摘要(Cache Digest)

关于 ICP 的最常见的抱怨在于它增加了每个请求的延时。许多情况下，在 squid 作出转发决定前，它会等待所有的 ICP 响应抵达。squid 的 cache 摘要提供了类似的功能，但没有每个请求的网络延时。

Cache 摘要基于由 Pei Cao 首先发布的一项技术，叫做摘要缓存。基本思路是用一个 Bloom filter 来表现 cache 内容。邻居 cache 下载其他每个 cache 的 Bloom filter（也即摘要）。然后，通过查询摘要来决定某个 URI 是否在邻居的 cache 里。

相对于 ICP，cache 摘要以空间交换时间。ICP 查询浪费时间（延时），cache 摘要浪费空间（内存，磁盘）。在 squid 中，邻居的摘要完全存放在内存里。在一个典型的摘要里，每百万目标需要大约 625KB 的内存。

Bloom filter 是一种有趣的数据结构，它提供对条目集合的有损耗编码。Bloom filter 自身简单的是一个大的位数组。给定某个 Bloom filter（和用于产生它的参数），你会发现，不能确定是否某个特定条目位于集合中。在 squid 中，条目就是 URI，摘要大小是每个 cache 目标 5 位。例如，要呈现 1,000,000 个 cache 目标的集合，需要用到 5,000,000 位或 625,000 字节的 filter。

因为它们的天性，Bloom filter 不能完整的呈现条目集合。它们有时候不正确的指示某个条目位于集合中，因为 2 个或多个条目可能在同一位上打开。换句话说，filter 可能显示目标 X 位于 cache 中，即使 X 从来没被缓存或请求过。通过调整 filter 的参数，你可以在一定程度上控制这种假情况。例如，增加每个目标的位数量，可以减少这种假情况。请见我在 O'Reilly 出版的书:Web Caching，可以找到关于 Caceh 摘要的更多细节。

10.7.1 配置 squid 的 cache 摘要

首先，必须在编译 squid 时激活 Cache Digest 代码。在运行./configure 时简单的增加 --enable-cache-digests 选项。采用这步导致在运行 squid 时发生 2 件事：

1)Squid cache 产生它自己内容的摘要。邻居 cache 如果也配置了使用 cache 摘要，那可能就会请求这个摘要。

2)Squid 请求每个邻居的 cache 摘要。

假如不想请求某个邻居的 cache 摘要，就在 cache_peer 行里使用 no-digest 选项，例如：

cache_peer neighbor.host.name parent 3128 3130 no-digest

Squid 在下述 URL 中保存它自己的摘要：
http://my.host.name:port/squid-internal-periodic/store_digest 当squid请求邻居的摘要时，它简单请求：http://neighbor.host.name:port/squid-internal-periodic/store_digest 明显的，这种命名机

制特指squid。假如邻居cache支持cache摘要，但它不是squid，你必须告诉squid邻居摘要的不同地址。cache_peer指令的digest-url=url选项允许你配置邻居的cache摘要URL。例如：

cache_peer neighbor.host.name parent 3128 3130 digest-url=http://blah/digest

squid.conf 有许多指令控制 squid 产生它自己的 cache 摘要的方法。首先，digest_generation 指令控制 squid 是否产生自己的 cache 摘要。假如你的 cache 是一个子 cache，而不是其他任何 cache 的父或姐妹 cache，那么你也许想禁止产生摘要。其他指令控制产生摘要的低层次细节。只有当你完全理解了 cache 摘要的执行原理，你才能改变它们。

digest_bits_per_entry 决定摘要的大小。默认值是 5。增加该值导致更大的摘要（浪费更多内存和带宽）和更低的假命中可能性。降低该值导致更小的摘要和更多的假命中。我觉得默认值非常合理。等于或低于 3 的设置导致太多的假命中而不可用，等于或高于 8 的设置简单的浪费带宽。

Squid 使用 2 个步骤来创建 cache 摘要。首先，它建立 cache 摘要数据结构。这基本上是 1 个大的 Bloom filter 和包含了摘要参数的小头部。一旦写入了数据结构，squid 就会对摘要创建缓存的 HTTP 响应。它预先包含某些 HTTP 头部，并和其他缓存响应一起，存储该响应到磁盘。

Cache 摘要对某时刻的 cache 内容维护一个快照。 digest_rebuild_period 控制 squid 重建摘要数据结构（并非 HTTP 响应）的频率。默认是每个小时 1 次。更频繁的重建意味着 squid 的摘要更新更快，但会浪费更多的 CPU 时间。重建过程对 CPU 影响相对较大。在 squid 重建摘要过程中，用户会感觉到速度降低。

digest_rebuild_chunk_percentage 指令控制每次调用重建函数时，多少 cache 增加到摘要里。默认是 10%。在每次调用重建函数过程中，squid 增加一定百分比的 cache 到摘要里。在该函数运行时，squid 不处理用户请求。在增加了指定的百分比后，该函数重新安排它自己的时间表并退出，以便 squid 能正常处理 HTTP 请求。在处理完等待请求后，squid 返回重建函数，并增加另外的 cache 块到摘要里。减少该值将给予用户更多的响应时间，然而增加了重建摘要所需的总时间。

digest_rewrite_period 指令控制 squid 从摘要数据结构创建 HTTP 响应的频率。大部分情形下，这个频率应该匹配 digest_rebuild_period 值。默认是 1 小时 1 次。重写过程由对某个函数的数次调用组成，该函数简单的添加一定数量的摘要数据结构到 cache 条目里（就象 squid 正在从网络中读取原始服务器的响应）。每次调用该函数， squid 添加 appends digest_swapout_chunk_size 字节的摘要。

10.8 超文本 cache 协议（HTCP）

HTCP 和 ICP 有许多共同的特征，尽管 HTCP 范围更广，并且通常更复杂。两者都使用 UDP 传输，并且两者都是请求执行的协议。然而，HTCP 也有很多与 ICP 不同之处，即：
1）ICP 查询仅包括 URI，甚至没有请求方式。HTTP 查询包括完整的 HTTP 请求头部。

2）ICP 不提供安全保证。HTCP 通过共享密钥，提供附加的消息验证，尽管它还没有在 squid 中实现。两者都不支持加密消息。
3）ICP 使用简单的，修正大小的二进制消息格式，难于扩展。HTCP 使用复杂的，可变大小的二进制消息格式。

HTCP 支持 4 种基本的编码：

TST
测试缓存响应是否存在

SET
告诉邻居更新缓存目标头部

CLR
告诉邻居从其 cache 里删除一个目标

MON
监视邻居 cache 的活动

在 squid 里，当前仅实现了 TST 编码。本书不讨论其他方面。

使用 HTCP 相对于 ICP 的主要优势在于更少的假命中。HTCP 有更少的假命中，因为查询消息包含了完整的 HTTP 请求头部，包含了来自客户端的任何 Cache-Control 要求。使用 HTCP 的主要不足在于 HTCP 查询更大，要求更多的 CPU 来处理产生和解析消息。测量显示，HTCP 查询大约是 ICP 查询的 6 倍大，这归咎于 HTTP 请求头部的存在。然而，squid 的 HTCP 响应典型的比 ICP 响应小。

HTCP的文档在RFC 2756 里，它被作为实验性协议。关系消息格式的更多信息，请见RFC：http://www.htcp.org　或者我的O'Reilly的书，Web Cacheing

10.8.1 配置 Squid 使用 HTCP

为了使用 HTCP，必须配置 squid 使用--enable-htcp 选项。当该选项激活时，squid 默认变成 HTCP 服务器。htcp_port 指定 HTCP 端口号，默认是 4827。将端口号设为 0 禁止了 HTCP 服务模式。

要成为 HTCP 客户端，必须在 cache_peer 行里增加 htcp 选项。当你增加该选项时，squid 总是发送 HTCP 消息，而不是 ICP，到邻居 cache。不能对单一邻居既使用 HTCP 又使用 ICP。放置 ICP 端口号的那个域实际上变成了 HTCP 端口号，所以你必须也要改变它。例如，假如你想将某个 ICP 邻居变成 HTCP 的，如下是邻居 cache 的 ICP 配置：

cache_peer neighbor.host.name parent 3128 3130

为了转换到 HTCP，该行变成：

cache_peer neighbor.host.name parent 3128 4827 htcp

有时候人们忘记改变端口号，这样在发送 HTCP 消息到 ICP 端口时，就会犯致命错误。若这点发生，squid 在 cache.log 里写警告日志：

2003/09/29 02:28:55| WARNING: Unused ICP version 23 received from 64.216.111.20:4827

与对待 ICP 查询不同，Squid 当前不记录 HTCP 查询。HTCP 查询也不可在 client_list 页面跟踪到。然而，若为对等 cache 激活了 HTCP，cache 管理器的 server_list 页面就（见 14.2.1.50 节）会显示命中和丢失的 HTCP 响应的计数和百分比。

Histogram of PINGS ACKED:

Misses	5085	98%
Hits	92	2%

注意，当前还没有哪个 squid 版本支持 HTCP 验证。

10.9 Cache 数组路由协议(CARP)

CARP 是一种在一组缓存代理中，区分开 URI 的算法。换句话说，每个 URI 被分配到 cache 中的一个。CARP 在一组 cache 中，最大化了命中率，最小化了目标重复。该协议由 2 个主要的成分组成：路由函数和代理数组成员表。不象 ICP,HTCP,和 Cache 摘要，CARP 不会预示某个请求是否 cache 命中。这样，你不能在姐妹关系中使用 CARP--仅在父 cache 中有效。

CARP 背后的基本想法是，你有一组（或一个数组）的父 cache，它们处理所有来自用户或子 cache 的负载。cache 数组是用来处理日益增长的负载的方法之一。无论何时你需要更高的性能，只需增加更多的数组成员。CARP 是一个确定性的算法。也就是说，同样的请求总是走向同一数组成员（只要数组大小不改变）。不象 ICP 和 HTCP，CARP 不使用查询消息。

关于 CARP 另一个有趣的事情是，你可以选择用多种不同方法来扩展它。例如，方法之一是让所有用户代理(user-agent)执行 CARP 算法。使用一个 JavaScript 写的 Proxy Auto-Configuration(PAC)函数（附录 F），你也许能做到这点。然而，某些用户代理可能不能执行或支持 PAC 文件。方法之二是使用二级 cache 堆叠。子 cache 接受来自所有用户代理的请求，然后它们执行 CARP 算法，为每个请求选择父 cache。然而，除非你的网络够大，否则太多的 cache 带来的负担可能大过受益。最后一种方法，你也能在数组自身内执行 CARP。也就是说，用户代理连接到 cache 数组中的随机成员，但每个成员转发 cache 丢失到基于 CARP 算法的数组的其他成员。

CARP 被设计为比简单的哈希算法更好，后者典型的通过应用某个哈希函数（例如 MD5）到 URI 来工作。然后该算法计算数组成员数量的系数。它可能象如下这段伪代码一样简单：

N = MD5(URI) % num_caches;
next_hop = Caches[N];

该技术在所有 cache 中均一的分摊 URI。它也提供兼容性映射（最大化 cache 命中），只要 cache 的数量保持不变。然而，当增加或删除 cache 时，该算法改变大部分 URI 的映射。

CARP 的路由函数在该技术中以 2 种方式得以改进。首先，它允许不均衡共享负载。例如，可以配置一个父 cache 接受的请求数量是另一个的 2 倍。第二，增或删数组成员最小化了被再分配的 URI 的片断。

CARP 的负面影响是它相对消耗 CPU 时间。对每个请求，squid 为每个父 cache 打分。该请求被发送到分数最高的父 cache。该算法的复杂性与父 cache 的数量成正比。换句话说，CPU 负载的增加，与 CARP 父 cache 的数量成正比。然而，CARP 里的算术已被设计成比 MD5 和其他加密哈希函数更快。

除了负载共享算法，CARP 也有个协议成分。成员表有定义良好的结构和语法，以便单一数组里的所有客户端能有相同的配置。假如某些客户端配置得不同，CARP 变得没什么用，因为并非所有客户端发送同样的请求到同一父 cache。注意 squid 当前没有实现成员表功能。

Squid 的 CARP 实现在另外一方面也有缺陷。协议认为，假如某个请求不能被转发到最高分的父 cache，它就被发送到次高分的成员。假如又失败，应用就会放弃。squid 当前仅使用最高分的父 cache。

CARP 的原始文档是一份 1998 年的 Internet 草稿，现在已经过期。它由下述 2 人开发：Vinod Valloppillil of Microsoft and Keith W. Ross of the University of Pennsylvania。查询一下，你还可以在 Internet 上找到老文档。甚至能在 Microsoft 的站点上找到一些文档。在我的 O'Reilly 的书 Web Caching 里，你能找到更多信息。

10.9.1 配置 Squid 使用 CARP

为了在 squid 里使用 CARP，必须首先在运行./configure 脚本时使用--enable-carp 选项。接着，必须为属于数组成员的父 cache，增加 carp-load-factor 选项到 cache_peer 行。如下是示例：

cache_peer neighbor1.host.name parent 3128 0 carp-load-factor=0.3
cache_peer neighbor2.host.name parent 3128 0 carp-load-factor=0.3
cache_peer neighbor3.host.name parent 3128 0 carp-load-factor=0.4

注意，所有的 carp-load-factor 值必须合计为 1.0。Squid 会检查该条件，假如有差错，它会抱怨。另外，cache_peer 行必须以负载因素值的增量来列举。仅仅近来的 squid 版本会检查该条件是否真。

记住，在关于邻居 cache 的存活/死亡状态方面，CARP 在一定程度上有些特殊。正常情况下，在 10 次失败连接后，squid 宣布邻居死亡（并中止发送请求到它）。然而在 CARP 情况中，

squid 会跳过某个有 1 次或多次失败连接的父 cache。一旦 squid 使用 CARP 工作，就可以使用 cache 管理器的 carp 页面来监视它。请见 14.2.1.49 的更多信息。

10.10 归纳所有

现在你可能意识到，squid 有许多不同的方法，来决定请求如何转发，和转发到哪里。许多情况下，你可能同时使用不止一种的协议或技术。然而，仅仅通过观察配置文件，难以发现 squid 如何联合使用这些不同技术。在本节我将解释，squid 实际上怎样做出转发决定。

显而易见，一切从 cache 丢失开始。任何作为未确认的 cache 命中而得到满足的请求，不会在下述节里描述。

选择程序的目的是，创建适当的下一跳位置列表。下一跳位置可能是邻居 cache，或原始服务器。依赖于配置不同，squid 可能选择 3 个下一跳。假如请求不能首先满足，squid 接着尝试第 2 个，然后是第 3 个。

10.10.1 步骤 1：直接决定选项

第一步决定某个请求：它可能，必须，或不必直接发送到原始服务器。squid 会参考该请求的 never_direct 和 always_direct 访问规则列表。目的是为了给下面 3 个值设置 1 个标记：DIRECT_YES, DIRECT_MAYBE, 或 DIRECT_NO。该标记随后决定 squid 是否为该请求选择 1 个邻居 cache。squid 按顺序检查下述条件。假如任何一个条件为真，它设置 direct 标记，并跳到步骤 2。假如你在跟进源代码，该步相应于 peerSelectFoo( )函数的开始：

1．Squid 首先查找 always_direct 列表。假如请求匹配该列表，direct 标记设为 DIRECT_YES。

2．Squid 接着查找 never_direct 列表。假如请求匹配该列表，direct 标记设为 DIRECT_NO。

3．对看起来陷入转发死循环的请求，squid 有特殊的检查方法。当 squid 检查到 1 个转发死循环，它将 direct 标记设为 DIRECT_YES 来打破循环。

4．仅在激活了 netdb 情况下，squid 才检查 minimum_direct_hops 和 minimum_direct_rtt 的设置。假如测量跳计数，或往返时间低于配置的值，squid 设置 direct 标记为 DIRECT_YES。

5．假如前述条件没有真的，squid 设置 direct 标记为 DIRECT_MAYBE。

10.10.2 步骤 2：邻居选择协议

在这里，squid 使用堆叠协议之一来选择邻居 cache。象以前一样，一旦 squid 在本步里选择了邻居，它直接跳到步骤 3。该步粗略的相当于 peerGetSomeNeighbor( )函数：

1． Squid 检查邻居的 Cache 摘要。假如摘要指示 cache 命中，该邻居就放到下一跳列表。

2．Squid 尝试 CARP（假如激活了）。CARP 总是成功（例如选择一个父 cache），除非 cache_peer_access 或 cache_peer_domain 规则，禁止某条特殊请求与任何父 cache 通信。

3．Squid 检查 netdb 度量（假如激活了），选择最近父 cache。假如 squid 了解到，从 1 台或多台父 cache 到原始服务器的往返时间，少于它自己到原始服务器的 RTT，squid 就选择最少 RTT 的父 cache。这点若发生，必须符合下列条件：

1）Squid 和父 cache 都必须激活 netdb 功能；
2）query_icmp 必须在配置文件里激活；
3）原始服务器必须响应 ICMP ping 消息；
4）父 cache 以前必须测量过到原始服务器的 RTT，并在 ICP/HTCP 应答里（或通过 netdb 交换）返回这些测量值。

4．作为最后的手段，Squid 发送 ICP/HTCP 查询。squid 遍历所有的邻居，并检查许多条件。squid 在如下情况下不查询邻居：

1）direct 标记是 DIRECT_MAYBE，请求不可层叠（见 10.4.5 节）。因为 squid 能直接到达原始服务器，它不会将这个请求转发到邻居，该请求看起来不可缓存。
2）direct 标记是 DIRECT_NO，邻居是姐妹关系，并且请求不可层叠。因为 squid 被强迫要求使用邻居 cache，它就仅查询父 cache，后者总能处理 cache 丢失。
3）cache_peer_access 或 cache_peer_domain 规则禁止发送该请求到邻居。
4）邻居设置了 no-query 标记，或者其 ICP/HTCP 端口号是 0。
5）邻居是广播应答者。

5．Squid 计算它发出了多少查询，并估计该收到多少应答。假如它期望至少一个应答，下一跳选择过程会延时，直到应答到达，或超时发生。Squid 期望从生存着的邻居 cache 那里接受到应答，而不是死亡的邻居（见 10.3.2 节）。

10.10.3 步骤 2a:ICP/HTCP 应答处理

假如 Squid 发出任何 ICP 或 HTCP 查询，它会等待一定数量的应答。在发送出查询后，Squid 知道它期待多少应答，和等待应答的最长时间。Squid 期待来自被查询的生存的邻居的 1 个应答。假如使用广播，squid 将当前组大小的估计值增加到期待的应答计数里。在等待应答时，有可能 1 个或多个应答迟迟不来，这样 squid 会安排 1 个超时时间表。

当 squid 接受到来自邻居 cache 的 ICP/HTCP 应答时，它采取如下动作：

1．假如应答是 1 个命中，squid 立即转发请求到那个邻居。在该点后抵达的任何应答都被忽略。

2．假如应答是 1 个丢失，并且它来自姐妹 cache，那么就被忽略。

3．Squid 不立刻对来自父 cache 的 ICP/HTCP 丢失采取动作。代替的，它记住哪个父 cache 符合下列标准：

closest-parent miss
假如应答包含 netdb RTT 值，squid 会记住到原始服务器的 RTT 最少的父 cache。

first-parent miss
Squid 会记住第一应答的父 cache。换句话说，就是到你的 cache 的 RTT 最少的父 cache。2 个 cache_peer 选项影响这种选择算法：weight=N 和 closest-only。

weight=N 选项让父 cache 比它实际更近。在估算 RTT 时，squid 使用该权重来划分实际 RTT。通过增加父 cache 的权重，可以给予它们更高的优先级。

closest-only 禁止邻居的第一父 cache 丢失特性。换句话说，squid 仅在父 cache 离原始服务器最近时，才会选择这个父 cache（基于 ICP/HTCP 丢失应答）。

4．假如 squid 接受到期待数量的应答（所有丢失），或超时发生，它选择最近父 cache（closest-only）丢失邻居（假如设置了）。否则，它选择第一父 cache（first-parent）丢失邻居（假如设置了）。

Squid 也许不会接受到任何来自父 cache 的 ICP/HTCP 应答，因为父 cache 有可能没被查询到，或因为网络丢包。在该情形下，squid 信赖从父 cache 选择算法，在下面节里描述。

假如在接受到期待数量的应答前，ICP/HTCP 查询超时，squid 预先考虑将 access.log 里的字符串 TIMEOUT_的值设为结果码。

10.10.4 步骤 3：从父 cache 选择

该步有点棘手。记住假如 direct 标记是 DIRECT_YES，squid 就不会执行这步。假如这个标记是 DIRECT_NO，并且第 2 步选择失败，squid 会调用 getSomeParent( )函数（随后描述）来选择备份父 cache。接着，squid 将所有它认为是存活的父 cache 追加到列表。这样，在返回错误消息到用户前，它会尝试所有可能的父 cache。

在 DIRECT_MAYBE 情形下，squid 会同时增加父 cache 和原始服务器到列表。顺序依赖于 prefer_direct 设置。假如激活了 prefer_direct，squid 首先将原始服务器插入列表中。接着，假如请求是可层叠的，或假如禁用了 nonhierarchical_direct 指令，squid 会调用 getSomeParent()函数。最终，假如禁止了 prefer_direct，squid 才最后将原始服务器加到列表中。

getSomeParent()函数基于下列标准来选择父 cache 之一。在每种情形下，父 cache 必须是存活的，并允许依照 cache_peer_access 和 cache_peer_domain 规则来处理请求：

1）第一个父 cache，有默认的 cache_peer 选项。
2）父 cache 的 round-robin cache_peer 选项，有最低的请求数量。
3）第一个父 cache，已知是存活的。

10.10.5 重试

偶然情况下，因为某些理由，squid 试图转发请求到原始服务器或邻居 cache 可能会失败。这就是为什么 squid 在邻居选择过程中，要创建一个下一跳位置列表的原因。当下列类型的错误之一发生时，squid 重新尝试请求列表里的下一个服务器：

1）网络拥塞或其他错误可能导致“连接超时”。
2）原始服务器或邻居 cache 可能临时不可到达，导致“连接拒绝”错误。
3）假如请求导致 cache 丢失，姐妹 cache 可能返回 504（网关超时）错误。
4）假如 2 个 cache 在访问控制策略里配错了，邻居 cache 可能返回“访问拒绝”错误消息。
5）在 squid 读 HTTP 消息主体前，可能在已建立的连接上发生读错误。
6）持久连接可能存在紊乱情况。

Squid 的重试失败请求的算法相对强健。与其让 squid 对用户返回错误，不如让它再试一次（当然会有延时）。

10.11 该怎么做？

Squid 新手经常问同样的或相似的问题，关于如何让 squid 正确的转发请求。这里我将告诉你，在普通这种情况下，如何配置 Squid。

10.11.1 通过另外的代理发送所有请求？

简单的只需定义父 cache，并告诉 squid 它不允许直接连接到原始服务器。例如：

```
cache_peer parent.host.name parent 3128 0
acl All src 0/0
never_direct allow All
```

该配置的弊端是，假如父 cache down 掉，squid 不能转发 cache 丢失。假如这种情况发生，用户会接受到“不能转发”的错误消息。

10.11.2 通过另外的代理发送所有请求，除非它 down 了？

试试这个配置：

```
nonhierarchical_direct off
prefer_direct off
cache_peer parent.host.name parent 3128 0 default no-query
```

或者，假如你喜欢对其他代理使用 ICP：

```
nonhierarchical_direct off
prefer_direct off
cache_peer parent.host.name parent 3128 3130 default
```

在该配置下，只要父 cache 存活，squid 就会将所有 cache 丢失转发给它。使用 ICP 可以让 squid 快速检测到死亡的父 cache，但同时在某些情形下，可能不正确的宣称父 cache 死亡。

10.11.3 确认 squid 对某些请求，不使用邻居 cache 吗？

定义 1 条 ACL 来匹配特殊的请求：

```
cache_peer parent.host.name parent 3128 0
acl Special dstdomain special.server.name
always_direct allow Special
```

在该情形下，对 special.server.name 域的请求的 cache 丢失，总是发送到原始服务器。其他请求也许，或也许不，通过父 cache。

10.11.4 通过父 cache 发送某些请求来绕过本地过滤器？

某些 ISP（或其他组织）有上级服务提供者，他们强迫 HTTP 传输通过包过滤代理（也许使用 HTTP 拦截）。假如你能在他们的网络之外使用不同的代理，那就能绕过其过滤器。这里显示你怎样仅发送特殊的请求到远端的代理：

```
cache_peer far-away-parent.host.name parent 3128 0
acl BlockedSites dstdomain www.censored.com
cache_peer_access far-away-parent.host.name allow BlockedSites
never_direct allow BlockedSites
```