Speed up your internet access using Squid's refresh patterns

Bandwidth limitation is still a problem for a lot of people who connect to the Internet. You can improve your available bandwidth by installing Squid caching proxy server on your network with configuration parameters that will increase your byte hit rate, giving you about 30-60% more bandwidth.

Squid can be fine-tuned to satisfy a host of needs. The stable version has at least 249 configurable parameters. The heavily commented configuration file, usually found in /etc/squid.conf, is more than 4,600 lines long. This can be intimidating to even experienced administrators. All settings are to be modified in this file.

You need a big cache that will not fill up in less than a week, and preferably should take more than a month to fill up. The actual size will be dependent on the volume of traffic on your network. The bigger the size of your storage, the greater the probability that the object someone is requesting for will already be in your cache.

In addition to the memory required for your operating system and Squid to run, you will need memory of about 1% of your cache size to keep the database of your cache in memory. That is, for a cache of 100GB disk space, you will need about 1GB RAM, in addition to about 100MB for the OS and Squid.

The default maximum size of objects that may be cached by Squid is 4MB. Nowadays, this is too low for the media-rich Internet. If your clients download a lot of video and software packages, you can increase this to a figure more representative of the maximum size of files that your clients normally download -- say 100MB.

Refresh patterns determine what is saved and served from the cache. Ideally, you would want your squid to follow the directions of the Web servers serving the content to determine what is cacheable and for how long. These directions are set as HTTP headers that are processed and understood by Squid. Unfortunately, the directions given by most servers are the Web servers' defaults, and do not produce significant bandwidth savings.

Refresh patterns are of the format:

refresh_pattern [-i] regex min percent max [options]

where min and max are time values in minutes and percent is a percentage figure. The options are:

- override-expire -- ignores the expire header from the Web server.
- override-lastmod -- ignores the last modified date header from the Web server.
- reload-into-ims -- a reload request from a client is converted into an If-Modified-Since request.
- ignore-reload -- a client's no-cache or "reload from origin server" directive is ignored. The request can therefore be satisfied from the cache if available.
- ignore-no-cache -- a no-cache directive from the Web server which makes an object non-cacheable is ignored.
- ignore-no-store -- a no-store directive from the Web server which makes an object non-cacheable is ignored.
- ignore-private -- a private directive from the Web server which makes an object non-cacheable is ignored.
- ignore-auth -- objects requiring authorisation are non-cacheable. This option overrides this limitation.
- refresh-ims -- a refresh request from a client is converted into an If-Modified-Since request.

Consult your configuration file to see which of these options are available in your version of Squid.

Refresh patterns are effective if there is no expire header from the origin server, or your refresh pattern has an override-expire option. Example:

refresh_pattern -i \.gif$ 1440 20% 10080.

This says:

- If there is no expire header for all objects whose names end in .gif or .GIF (that is, image files) then:
- if the age (that is how long the object has been on your cache server) is less than 1,440 minutes, then consider it fresh and serve it and stop
- else if the age is greater than 10,080 minutes, consider it stale and go to the origin server for a fresh copy and stop
- else if the age is in between the min and max values, use the lm-factor to determine freshness. lm-factor is the ratio of the age on your cache server to the period since creation or modification of the object on the origin server as a percentage. So if the object was created 10,000 minutes ago on the origin server and it has been on my cache server for 1,800 minutes (that is the age) the lm-factor is 1,800/10,000 = 18%.
- If the lm factor is less than the percent in our refresh pattern (20%) then the object is considered fresh; serve it and stop
- else the object is stale, go for a fresh copy from the origin server.

For objects that scarcely change under the same file name, such as video, images, sound, executables, and archives, you can modify the refresh pattern to consider them fresh on your Squid for a longer time, increasing the probability of having hits. For example, you could modify our refresh pattern above to:

```
refresh_pattern ^ftp: 1440 20% 10080
refresh_pattern ^gopher: 1440 0% 1440
refresh_pattern -i \.(gif|png|jpg|jpeg|ico)$ 10080 90% 43200 override-expire
ignore-no-cache ignore-no-store ignore-private
refresh_pattern -i \.(iso|avi|wav|mp3|mp4|mpeg|swf|flv|x-flv)$ 43200 90%
432000 override-expire ignore-no-cache ignore-no-store ignore-private
refresh_pattern -i \.(deb|rpm|exe|zip|tar|tgz|ram|rar|bin|ppt|doc|tiff)$
10080 90% 43200 override-expire ignore-no-cache ignore-no-store ignore-private
refresh_pattern -i \.index.(html|htm)$ 0 40% 10080
refresh_pattern -i \.(html|htm|css|js)$ 1440 40% 40320
refresh_pattern . 0 40% 40320
```

Sometimes, for no good reason, at least from our perspective, origin servers, such as youtube.com, do everything to make it difficult or impossible for you to cache content. The

options above should help you to overcome some of these limitations.

Refresh patterns are matched against all requests in order from the top until there is a matching rule. The last rule is a catch-all and will match any request that is not satisfied by any of the rules above it. There are normally separate catch-all default rules for other protocols like FTP and gopher at the very top of the list so as to exempt those protocols from the patterns below them.

By default, Squid will not cache dynamic content. Dynamic content is determined by matching against either "cgi-bin" or "?". This feature used to be activated via the "hierarchy_stoplist" and "cache deny" settings in older versions of Squid. In recent versions, starting with 3.1, this feature is activated via a refresh pattern such as `refresh_pattern (/cgi-bin/|\?) 0 0% 0`. This enables you to specify sites that serve dynamic content that could be made cacheable in bypass rules. For example, you could set up a refresh pattern such as:

```
refresh_pattern -i movies.com/.* 10080 90% 43200
refresh_pattern (/cgi-bin/|\?) 0 0% 0
```

Then, even if content from movies.com is served with "?" in their URL, the content will still be cached if all other conditions are met.

For the older versions of Squid, you will have to define an access control list (ACL) for the content providers you wish to make exceptions for, and use `cache accept` to exempt it before the `cache deny` rule. The following example is from the Squid wiki:

```
# Let the client's favourite video site through
acl youtube dstdomain .youtube.com
cache allow youtube
# Now stop other dynamic stuff being cached
hierarchy_stoplist cgi-bin ?
acl QUERY urlpath_regex cgi-bin \?
cache deny QUERY
```

Squid makes a lot of DNS requests, one dns request for each http request. Install a caching DNS server on your server and have Squid use it so as to cut down on your DNS requests. This how-to may be helpful.

Sites like Microsoft's windowsupdate.com, which virtually all Windows PCs update their OS from, are among the most bandwidth-intensive sites on some networks. Unfortunately, they are

not cacheable because they offer partial responses (http return code 206), which Squid presently does not cache. Where you have control over the client machines, you can install Microsoft's Update Server to handle caching for windowsupdate. If you cannot use the Update Server, you can use Squid's delay pools -- a bandwidth management technique -- to limit the portion of bandwidth that windowsupdate consumes during your peak periods. The clients will then have to be online during off-peak periods to complete their updates.

Below, we configure one global delay pool at 64Kbps (8KBps). Traffic for which the ACL of destination domain is windowsupdate.com during the peak period of 10:00-16:00 will be limited to 64Kbps.

```
acl winupdate dstdomain .windowsupdate.com
acl peakperiod time 10:00-16:00
delay_pools 1
delay_class 1 1
# 64 Kbit/s
delay_parameters 1 8000/8000
delay_access 1 allow winupdate peakperiod
```

After making changes like the ones above, my Squid's byte hit rate increased from about 8% to between 26-37%. If you are doing 33%, it means a third of all traffic is coming from your cache, and not from slower links across the Internet. For monitoring and log analysis to determine the performance of your Squid, you can use squid3-client and calamaris.

Solomon Asare is the developer of DelXy*, an HTTP compression service*