INASP: Effective Network Management Workshops

1 Unit 11: Web Caching Manual

1.1 About these workshops

Authors:

- Dick Elleray, AfriConnect
    - delleray@africonnect.com
- Chris Wilson, Aptivate
    - chris+inaspbmo2013@aptivate.org

Date: 2013-04-29

Contents

1.2 Caching

The term cache literally means to store. In computing terms caching is the act of storing information on a local system, where the act of retrieving the information from the local cache is less than the cost of retrieving the information from the original source.

There is a principle known as the “locality of reference” in which there are two flavours, temporal and spatial. Temporal refers to the popularity or degrees of reference to, in this case, an item of information. Certain sites for example google are referenced many more times than other sites. Spatial locality of reference refers to the phenomenon that requests for certain pieces of information are likely to occur together. We know for example that when a request for google is processed we should expect back the google image, in conjunction to the search input page. We will learn a little about HTTP requests later, this will help us understand how caches can be so helpful.

If we now imagine a situation where the image for google is transferred from the local cache instead of traversing an expensive (in data terms) WAN link we can see the benefit. This is easily transposed into other areas of computing. Caches are used for memory, display cards, drive controllers etc. In general any area where the cost of getting data from the local cache rather than from the source is much cheaper than from the original, a cache is useful.

When an item is transferred from the local cache instead of from the source this is termed a hit when it is not cached it is termed a miss.

An important concept of cache is that of stale data. Data or information is considered stale when it is not up to date with the source after validation. Fresh data is data that is generally used without validation (as it has not yet expired) and so is quicker.

Strong consistency is when data is always validated from the source before it is used, with weak consistency out of date data can sometimes be returned.

1.2.1 Web Architecture

The web is essentially client server type architecture: a client will typically make a small request to the web server, the web server will respond with a response to the request. The latest version of the HTTP protocol is vrsion HTTP/1.1 (version 3) and this determines the format of the request and the response. It is useful to look at this when we consider caching.

A HTTP header consists of a name followed by a colon and one or more values separated by commas. An example of a header request to google would be:

```
GET /index.html HTTP/1.1
Host: www.google.com
Accept: */*
```

HTTP defines four categories of headers: entity, request, response and general. Entity headers describe something about the data in the message body, for example the content-length length entity header describes the size of the data.

Request headers are sent by the client to the server, along with the request. `Host` is an example of a request header. Response headers are sent by the server to the client. `Server` is an example of a response header. General headers can be used in both `POST` requests and responses.

The first line of an HTTP message is important; it is called the request line and contains the request method for requests and response status, URI and HTTP version for responses. If we consider the example above of the request to google, and suitable response may be:

```
HTTP/1.1 200 Ok
Date: Mon, 11 Feb 2005 13:50:01 GMT
Last-Modified: Sun, 10 Feb 2005 13:00:26 GMT
Server: Apache/2.0.5
Content-Length: 20
Content-Type: text/plain
```

A HTTP request can contain several different methods as specified in RFC 2616. For the purposes of this course the methods we are interested in are `GET`, `POST` and `HEAD` methods.

Other protocols besides HTTP are used to get content from servers, two relevant ones are FTP and HTTP/TLS or HTTP over TLS commonly referred to as HTTPS.

1.3 Caching of Web Traffic

1.3.1 When and how should we cache web traffic?

For obvious reasons caching helps conserve bandwidth, the more content that is fetched from a local cache than direct from the destination will save bandwidth on the local Internet link. Caching can also reduce the load on destination webserver. Many content providers (eg CNN) use content providers such as Akamai to deliver content to users; these content caches are situated at key points throughout the internet. If this was not done the load on the CNN web server would increase exponentially. This is also an example of how cache servers that are close to the user will result in low latency (or transmission delays). An example of how latent demand for information can kill systems was reported from the South African 1998 elections. All live the election results where made available via the web. In the first few days while election results where being released the web servers frequently crashed. This was primarily because the content of the pages was dynamic and was fetched from the database every time a request was received. The system was quickly changed to generate static html pages every so often. This meant that the database was not queried for every page but it also had the side effect that the pages could be cached by cache servers very easily.

Dynamic pages can be difficult to cache as the content on the web pages can typically change frequently and the cache server can only cache at the expense of containing some stale data.

1.3.2 Types of Cache

Web content can be cached at different places, there is several different types of caches in existence, typically these are;

1.3.2.1 Caching Proxy Servers

These are servers that cache for multiple users; they typically sit between the client (browser) and server (web server). They usually are situated at an internet gateway. The more a Shared Proxy Server is used the better the cache hit rate will be where the cache hit rate is defined as the rate of hits vs. the rate of requests. The proxy server will effectively split the request to the web server into two parts. There will be a connection between the browser and the proxy server and between the proxy server and the web server.

1.3.2.2 Browser Caches

Many browsers have built in caches; these caches will store content on disk for a specified period of time. Often this cache is used when the user hits the back button. This is often known as a private or single user cache.

1.3.2.3 Reverse Proxies/ accelerators

These servers sit in front of a web server and intercept requests destined to the web server, they then reply with the content instead of the web server. As in the example given above with akamai, this service conserves resources on the web server.

1.3.2.4 Meshes and Hierarchies

Caches can talk to each other and share cached data and this can have pleasing benefits. We go into this in more depth later. For now it is convenient to know that a hierarchy and mesh can exist. These can further increase the efficiency of the cache system as a whole.

1.3.3 Web Caching: How does it Work?

1.3.3.1 The HTTP connection

When your browser of user agent makes a connection to the proxy server, it talks to the proxy server in a slightly different way than if it was talking straight to the web server. The browsers (or user agents) know they are talking to a proxy server because this has generally been configured in the agent's settings. The exception to this is a transparent proxy. The agent in a non- transparent setup will talk to the proxy server in HTTP language, even if the URL that the user is going to is an FTP URL. The agent will pass the request on to the proxy server; the proxy server will then communicate to the server in the correct protocol.

An example of this:

A user agent that is not configured to connect to a proxy server when connecting to www.google.com will have the following request structure (headers are shortened for convenience sake)

```
GET /index.html HTTP/1.1
Host: www.google.net
Accept: */*
Connection: Keep-alive
```

A user agent connecting to the same server via a proxy server will make the following request to the proxy server:

```
GET http://www.google.com/index.html HTTP/1.1
Host: www.google.com
Accept: */*
Proxy-connection: Keep-alive
```

A FTP request through a proxy server would have the following format:

```
GET ftp://ftp.is.co.za/pub/ HTTP/1.1
Host: ftp.is.co.za
Accept: */*
```

The difference between the direct request and the proxy request is that the first line in the request to the proxy contains the full URI, this is important as it enables the proxy to quickly ask for the page. In a transparent proxy the proxy server will only see the normal http header. It then builds the complete URI by combining the Host portion of the header with the first line.

1.3.4 Cacheability

A proxy will only determine if a response is cacheable once it has received that response from a server, there are several things that the proxy will look at before it determines if the response is cacheable.

1.3.4.1 Response codes or Status codes

When a user agent asks for a URL from a webserver the webserver will respond with a response that includes a status code, these status codes are defined in RFC 2616 and are broadly defined into five groups:

1xx

An informational, intermediate status. The transaction is still being processed.

2xx

The request was successfully received and processed.

3xx

The server is redirecting the client to a different location.

4xx

There is an error or problem with the client's request. For example, authentication is required, or the resource does not exist.

5xx

An error occurred on the server for a valid request.

The most common and one we are mainly interested in is the 200 (OK) status which means that the server has processed the request and is returning the response. The 200 code is cachable by default. However this is not enough, other responses in the response from the server may also have an influence on the cachability of the response as we shall see below.

1.3.4.2 Cache-Control

The Cache-Control feature of HTTP/1.1 is used to tell caches how to handle requests and responses.

The possible values of the Cache-Control header are:

private

This allows private caches to store responses but prevents a shared proxy from doing so.

public

This allows both private and shared caches to cache the response.

max-age

This will determine the expiration date/time of the response

s-maxage

This is the same as Max-age except it only applies to shared caches.

must-revalidate

This allows the cache to store the response subject to validation.

proxy-revalidate

Same as must-revalidate but only applies to shared caches.

no-store

Causes a response to become un-cachable.

1.3.4.3 Expiration/Validation

RFC 2616 states that a URL with neither an expiration time nor a cache validator should not be cached. A cache needs these pieces of information to tell if a copy of a piece of data is stale or not. A page that is expired can still be validated. A typical header would contain:

```
Date: Sun, 01 Apr 2001 18:32:48 GMT
Expires: Sun, 01 Apr 2001 18:32:48 GMT
```

1.3.4.4 Cookies

If you want all the response data apart from the cookie to be cached, you could use this setting:

```
Cache-control: no-cache="Set-cookie"
```

1.3.4.5 Dynamic Content

Typically dynamic content is content that changes frequently and so should not be cached. There are no header options specifically to control this but the cache can be told not to cache pages with certain string matches, for example cgi-bin or asp. URLs that contain query terms, typically denoted by a question mark “?” are also likely candidates for not caching.

1.3.5 Cache replacement

Cache replacement is the process that takes place once the cache is “full”. When this happens old cache objects must be removed to make way for new cache objects, there are several algorithms that cache servers use to do this.

1.3.5.1 LRU – least recently used

LRU is very popular and is as the name suggests based on the idea that the objects that have not been used for the longest time are removed from the cache. This is typically done using some sort of list; whenever an object in cache is accessed the object is moved to the top of the list. When an object needs to be removed from the cache the object at the end of the list is removed. This algorithm gives good performance.

1.3.5.2 LFU – Least frequently used

Objects are ordered according to the number of accesses. The algorithm does not usually take time in account but simply keeps a sorted list of the objects by number of accesses.

1.3.5.3 FIFO-First in first out

Objects are purged in the same order that they are added. This gives a very low hit rate as there is no accounting for popularity of the object.

1.3.5.4 Size

The largest objects are moved out first; typically an aging mechanism is needed as well otherwise the cache may fill up with old files.

1.4 Cache Hierarchies

1.4.1 What and How

When talking about cache hierarchies we are talking about relationships between caches. Much like a human family these relationships are described in terms of parent, child and sibling. A parent and sibling cache can also be referred to as a neighbor or peer.

A child cache will forward all requests that did not result in a local hit to the parent; the parent will then either reply with a hit from its own cache or will go and fetch the object from its source and pass it back to the child. A sibling cache will only respond with a hit, if it does not have the object in it’s local cache then it will respond with a miss to the cache that asked, the original cache will then go and fetch the object from the source.

Cache hierarchies can be very useful in large networks where WAN links are prevalent. If there are 5 sites each with its own proxy these proxies can be setup to peer to each other, this will result an object being fetched from the neighbor cache instead of via the gateway. Cache hierarchies need to be evaluated carefully and a clear understanding of what is involved is needed. A debate should then ensure to determine the optimal setup taking all factors into account.

Cache hierarchies can be very complex and can be setup in a mesh or in a hierarchy. To realize improved performance with a cache hierarchy, the following points should all be true:

- Some of the objects not found in your cache will be found in your neighbor caches. In other words, you can get cache hits from your neighbors.
- Cache hits from neighbors are delivered more quickly than misses from origin servers.
- Cache misses from parent caches are not significantly slower than responses from origin servers.

If any of these are false cache performance might actually suffer.

Caches use inter cache protocols to query each other, the most common of these are CARP, ICP and HTCP.

1.4.2 CARP

For a given request, CARP calculates a score for every proxy cache. The request is forwarded to the proxy with the highest score. If this fails, then the second-highest scoring cache is tried. The score is a calculation based on a hash of the URL, a hash of the cache's name, and weights assigned to each cache. The important characteristic is that adding a new cache to the array does not change the relative ranking of the scores for the other caches; instead, the new cache creates new scores. Statistically, the new scores will be higher than the existing caches' scores for a fraction of the URLs that is proportional to the cache's weight within the array.

CARP also specifies a file format for a Proxy Array Membership Table. This table allows clients to figure out which caches belong to a group. The table may be accessed via web protocols (HTTP) so many clients can easily retrieve the information.

The CARP algorithm may be used by any web client, such as a browser or proxy cache, that needs to choose among a set of caches. Note, however, that it only works for parent relationships because CARP does not predict cache hits. Another minor problem with CARP is related to persistent HTTP connections. A client that makes four requests may use more TCP connections with CARP than it would without. Finally, also note that CARP has linear scaling properties (similar to ICP) because a score must be calculated for every group member. CARP can be used by Squid and Microsoft proxy servers

1.4.3 ICP

ICP is the most useful intercache protocol, I quote this section from the book “Web Caching” by Duane Wessels.

> ICP is the original intercache protocol. Its primary purpose is to discover whether any neighbor caches have a fresh copy of a particular object. The neighbor caches answer with either yes (HIT) or no (MISS). The querying cache collects a particular number of ICP replies and then makes a forwarding decision. Even if all neighbors reply with MISS, ICP may provide additional hints that help to choose the best parent cache.

As already mentioned, the primary purpose of ICP is to find out which, if any, of your neighbor caches have a specific object. This feature is fundamental to the idea of a sibling relationship. We can request only cache hits from a sibling, so we need some way to predict whether a given request will be a hit or a miss. ICP's hit predictions are useful for parent relationships as well. In most cases, a hit from one parent should be faster than a miss from another.

ICP messages are normally sent as UDP packets, which have some useful side effects. The delay that an ICP transaction experiences tells us something about the status of the network and the remote cache. For example, if we don't receive a reply message within a certain period of time, we can conclude that the remote cache is offline or the network is severely congested. ICP can be used as a tiebreaker when choosing between two or more equivalent parents. A parent that is heavily loaded takes longer to process an ICP query, and the transaction experiences some delay. By selecting the first cache that responds, we do some basic load balancing.

One of the benefits of caching is reduced wait time for web pages. A strange thing about ICP is that cache misses actually incur an additional delay during an ICP query/reply phase. The

ICP transaction is very similar to gambling. If you get a HIT reply, then the gamble probably paid off. If, on the other hand, you get all MISS replies, then you lost the bet. The big question is: does ICP increase or decrease your overall object retrieval times in the long run? The answer depends on a number of factors, including the network distances to your neighbor caches and the speeds at which web objects are transferred from neighbors and origin servers.

If your neighbors are relatively close compared to the rest of the Internet, then the ICP delays represent a small gamble with a big payoff. If you win, you can perhaps download an object in 0.5 seconds instead of 5 seconds. If you lose, you've wasted perhaps only 0.05 seconds. Conversely, if your neighbor caches are not much closer than the rest of the Internet, ICP probably doesn't make sense. All downloads may still take 5 seconds on average, and the ICP misses probably add an additional 0.1- or 0.2-second delay.

Another important factor is the percentage of replies that are hits. In essence, this represents the probability of the gamble paying off. In practice, the ICP hit ratio is quite low—about 5%. Some caches achieve up to 10%, and some only get around 1%. The reason for this low percentage is that neighbor caches tend to store the same set of popular objects over time. The users of one cache request the same web pages as the users of another cache.

Speaking of bandwidth, you may be wondering how much of it ICP consumes. It is quite easy to calculate the additional bandwidth ICP uses. The size of an ICP query is 24 bytes plus the length of the URL. The average URL length seems to be about 55 characters, so ICP queries average about 80 bytes. ICP replies are four bytes smaller than queries.

Taken together, an ICP query/reply transaction uses about 160 bytes per neighbor before including UDP and IP headers. A cache with 3 neighbors uses 480 bytes for each cache miss. To put this in perspective, consider that the average web object size is about 10KB. Thus, a cache with three ICP neighbors increases its bandwidth consumption by about 5% on average. Note that this corresponds to a measurement made at the cache's network interface. Whether ICP increases your wide area bandwidth depends on the location of your neighbor caches.

Notice that an ICP query contains only a URL and none of the other HTTP headers from a client request. This is problematic because whether a given HTTP request is a hit or a miss depends on additional headers such as the Cache-control: max-age, Accept-language. ICP doesn't include enough information for a cache to make an accurate determination, which leads to false hits.

1.4.4 HTCP

HCTP addresses the problem of false hits with ICP by sending the full http headers with the request. HCTP is a very complex protocol and as such uses extra processing power.

1.4.5 Cache Digests

A cache digest is essentially a database of cache contents, this is shared between caches and this removes the reliance on the ICP query. An algorithm called the Bloom Filter is used to encode the cache contents into a database that describes the contents of the cache. A digest protocol will typically use more bandwidth during idle time but will utilize the same bandwidth as ICP in busy times. Digest also require more CPU and memory as the digest lists need to be kept in memory.

1.4.6 Which protocol to use?

The four protocols and algorithms presented here each have unique features and characteristics. Which one you should use depends on numerous factors. The following guidelines may help to determine the best protocol for your particular situation.

ICP may be a good choice if you need to interoperate with products from different vendors. Since ICP has been around longer than the others, it is supported in most caching products. It is also a reasonable choice if you want to build or connect into a small mesh of caches.

You want to avoid having too many neighbors with ICP; try to limit yourself to no more than five or six. ICP may not be a good choice if you are very concerned about security, and it may be all but useless on networks with high delays and/or a large amount of congestion. The protocol has no authentication mechanisms and may be susceptible to address spoofing. Finally, you probably cannot use ICP if there is a firewall between you and your neighbor caches because firewalls typically block UDP traffic by default.

CARP is a logical choice if you have multiple parent caches administered by a single organization. For example, some large service providers may have a cluster of proxy caches located where they connect to the rest of the Internet. If this applies to you, make sure you always have up-to-date configuration information. CARP is the only protocol that does not allow you to create sibling relationships.

HTCP has characteristics similar to ICP. You can use HTCP for small cache meshes and where network conditions are good. Unlike ICP, HTCP has relatively strong authentication. This may be particularly important if you need the object deletion features. HTCP should cause fewer false hits than ICP because hit/miss decisions are based on full HTTP headers rather than only the URI. Note that HTCP messages are about five times larger than ICP, so it uses more bandwidth. You should use Cache Digests if you can afford to trade increased memory usage for lower forwarding delays. The bandwidth tradeoffs of Cache Digests versus ICP depend on many factors. You cannot really use Cache Digests over slow network connections because the transfer of a large digests saturates the link. Cache Digests probably result in a higher percentage of false hits compared to ICP. Even so, overall client response times should be lower, and false hits are only a concern for sibling relationships.

1.4.7 Why not to have a hierarchy?

1.4.7.1 Privacy

You will have to trust caches. Log file are stored on cache servers, this might have privacy concerns. You will need to trust your cache and all its neighbors. There also is no guaranteed method for authenticating the authenticity of web pages end to end with the web server.

1.4.7.2 Hit ratios

On a lot of occasions similar data is contained on different proxies due to similar browsing patterns, this can result in lower hit ratios between cache servers.

1.4.7.3 Freshness

Different versions of a file can be contained across different cache servers.

1.4.7.4 Performance issues

In a large hierarchy the top levels can become performance bottlenecks. Management of this can become problematic

1.4.7.5 Legal issues

Proxy servers help users to become anonymous, it makes it difficult for service providers to track abuse.

1.4.7.6 Loops

A loop can occur when two proxies are parented to each other.

1.4.7.7 Points of failure

A parent cache in particular is a single point of failure, this is not optimal. Care must be taken when the parent cache is not under the control of your organization.

1.5 Transparent caching

1.5.1 The process

Transparent caching simply involves the process where the client request is intercepted at a router or switch and is passed onto a cache server. The cache server will then service the request and pass it back to the switch/router and the client.

This has the benefit that the clients do not need to be configured to use the cache. As the client does not know it is connected to a cache the HTTP request is a little different than when it is talking to a proxy server directly.

A switch or router will recognize a packet as being HTTP. This is usually done by identifying the port. There are various methods that are commonly used do the interception.

Layer four switches:

A layer 4 switch can make decisions based on IP addresses and port numbers, because of this they can be used for HTTP re-direction. These switches are often capable of other advanced features such as load balancing.

WCCP:

This is a Cisco method that is used between the switch/router and the cache, most implementations of this are by Cisco. WCCP supports load balancing and clusters.

CPR:

Cisco policy routing is when a router makes decisions based on some characteristic of the IP packet. Used by Cisco Routers and some switches.

In Line:

This is a device that combines caching and routing in one device. This is an obvious point of failure and care should be taken.

Other Router/Switch:

There are various software implementations of router/bandwidth management systems that will intercept HTTP traffic and pass this on to a cache. ipchains/iptables can be used to do this.

1.5.2 Issues

1.5.2.1 Port Issues

Not all HTTP traffic sits on port 80, the major problem comes when someone runs something other than HTTP on port 80, and a typical transparent cache will attempt to do something with it.

1.5.2.2 Browser and HTTP Issues

As the browser does not know it is using a proxy server when you issue a reload on the browser some browsers will neglect to issue the correct reload header and so the cache will serve up a page from its cache, this is because Internet Explorer in particular does not include the Cache- control: no-cache header.

1.5.2.3 It affects all HTTP traffic

There are times when you wish some traffic to not go via the cache; this traffic might be from robots or scripts. It is also difficult for users to bypass

1.5.2.4 Routing/IP Changes

The request that comes from the transparent cache will have the cache's IP address as the source. This might cause issues with services that use the original IP address for authentication or authorization.

1.5.2.5 Flexibility

You loose some measure of flexibility that you might have with multiple proxy servers, you can have groups of users serviced by proxy servers that have different configurations. This can be hard to do when a transparent cache is used.

1.5.2.6 Authentication

Authentication cannot be done when using interception.

1.6 Caching Clusters and fault tolerance

A single cache server can be a point of failure, when a single cache server fails your users can fail to connect to web sites. This can present a problem; the obvious solution is to run multiple cache servers. The most popular way for doing this is via DNS Round Robin. However it can sometimes take a long time for the browser to recognize the failure (up to two minutes) and switch over. Another technique is to use the proxy.pac or autoconfig scripts to do this; this will be fully explained in the next chapter.

Another obvious approach is to use a hardware or software cluster, however this can be expensive and difficult to implement.

1.7 Configuring cache clients

1.7.1 Manual Configuration

Proxies can be manually configured in most browsers, typically this is done by inserting the address and port of the proxy server in the appropriate place in the browser. (See below)

Both provide a place where you can enter a list of domains or IP addresses which bypass the cache. It is recommended to add all domains that you serve locally to these lists.

1.7.2 Auto Config scripts

You can imagine that configuring and maintaining the proxy cache settings on hundreds or thousands of clients is a difficult and monotonous task. Fortunately there are automated configuration mechanims to ease the burden.

A Proxy PAC file or autoconfig file is a very convenient way of setting proxy settings in the browser. If the browser is configured to use a proxy auto config script it will execute a function for every request. This function will return a list of proxy addresses.

The script must contain this function (FindProxyForURL) and return a valid proxy server. The nice thing about the proxy autoconfig script is that you have the flexibility of JavaScript at your control, for example you can forward different requests to different proxies based on the source address, destination or other info.

An example proxy PAC is:

```
function FindProxyForURL(url, host)
{
        var proxy1 = "proxy1.abc.com:8080; proxy2.abc.com:8080";
        var myip = myIpAddress();
        var ipbits = myip.split(".");
        var myseg = parseInt(ipbits[3]);

        if(myseg == Math.floor(myseg/2) * 2)
        {
                var proxy1 = " proxy1.abc.com:8080; proxy2.abc.com ";
        }
        else
        {
                var proxy1 = " proxy2.abc.com; proxy1.abc.com:8080 ";
        }

        if (shExpMatch(host, "*.abc.com"))
                return "DIRECT";
        else if (shExpMatch(host, "127.*"))
                return "DIRECT";
        else if (shExpMatch(host, "localhost"))
                return "DIRECT";
        else if (shExpMatch(host, "*.jstor.org"))
                return "PROXY 146.230.128.27:8080";
        else if (isInNet(myIpAddress(), "192.168.0.0", "255.255.0.0"))
                return "PROXY "siteproxy.abc.com:8080";
        else
                return proxy1;
}
```

What does this script do?

1. Sets a variable to point to two proxies that we want returned by default if there are no other matches.
2. Calculates whether the client (workstation) IP address is odd or even, and swaps the order of the proxies depending on the outcome. This is a simple and cheap form of load balancing across multiple proxy servers.
3. Checks the destination website address (using the function shExpMatch). In certain cases it returns `DIRECT`, telling the client to bypass the proxy server for these hostnames.
4. If the client is accessing the domain `jstor.org`, then the request will be sent to a different proxy server (perhaps in order to fix the public IP address).
5. If the client's IP address is in the range 192.168.0.0/16 then it will use a different proxy server (perhaps a physically closer one).

As you can see, this relatively simple script can tell the client to use one of four different proxy servers, or none, depending on the identity of the client.

1.7.3 Auto Discovery

Manually configuring all clients to use a Proxy PAC file is still a significant burden, even though it only has to be done once. Web Proxy Auto Discovery (WPAD) allows even this configuration step to be automated for many clients.

WPAD allows the administrator to configure the network DHCP, SLP or DNS server to hand out to clients the address of the Proxy PAC file that they should use.

For the DNS method, the client simply attempts to resolves the domain name `WPAD.domain`, where `domain` is the domain name configured on the client, possibly by the DHCP server. The administrator can configure this domain name to point to a web server hosting the PAC file. If the web address http://wpad.my.domain.edu/wpad.dat serves the PAC file to the client, then it will configure itself.

1.8 Political and organization aspects of caching

There are some issues that need to be considered when implementing a proxy server.

1.8.1 Privacy

The privacy of individuals needs to be respected in conjunction with the local law and any applicable organisational policies. If proxy logs are kept there should be accountability with regard to these logs. Caching servers can also be configured in a way that strengthens privacy, by hiding the identity of the end user.

1.8.2 Authentication

This issue is closely connected to privacy. Normally authentication is done for legal reasons, management or network security reasons. You need to make users aware of the consequences of their actions and where the boundaries between private and corporate actions lie. When you authenticate you are placing an identity against a request.

1.8.3 Site restrictions

It is possible to block content at the proxy server. This is a dangerous exercise and the cache administrator should ensure that he/she is operating within the boundaries of the organizational culture and policies when doing this.

1.8.4 Copyright

Technically, keeping a copy of protected information without permission of the copyright holder could be regarded as copyright infringement. In most jurisdictions it is considered to be completely legal, provided that the copy is only kept for functional reasons (it is necessary for fast and efficient network operation). However if in doubt, please check with a lawyer.

1.8.5 Content Integrity

How can you trust the information returned by the cache? Is it up to date, or has it been altered by the cache administrator? In some cases the user has no choice but to trust the cache administrator. The cache is only used for unsecured communications (non-SSL) and many attacks are possible by untrusted parties on such communications, so the additional threat posed by a proxy cache is usually negligible. But as usual it is important to properly secure the proxy server.

1.8.6 Uncachable Content

If you don’t wish to cache certain content there are a few options available to you including bypass lists and Proxy PAC files that ignore the cache servers for some URLs.

1.8.7 Free bandwidth

Care must be taken in the access rules of the proxy otherwise you could find yourself supplying free bandwidth to Internet users, and your servers and bandwidth could be used in Denial Of Service (DOS) attacks or other illegal activity. As a minimum, ensure you only allow internal connections to the proxy server.

1.9 Case Study: University of KwaZulu-Natal

This will be done via instructor PowerPoint

2 SECTION 2 - Squid

2.1 Squid Introduction

With apologies to the author this section on squid is based unashamedly on the book “Squid the Definitive Guide” by Duane Wessels. Since this book is to be given out to each participant I have tried to keep the structure along the same lines so it can be used as a sort of abbreviated study guide. I encourage everyone to get a copy of this excellent book if you consider running squid. (See Reference section)

2.1.1 What

Squid is an open source caching engine with many features. Squid runs on many platforms and operating systems including flavors of UNIX, Linux, BSD and Windows. Squid is a proxy and a cache; it is a proxy between the client and the server and will cache replies. Squid only supports HTTP requests from browsers but can talk other protocols such as FTP to servers. Undoubtedly squid is most famous as an extremely stable proxy cache server running under Linux.

2.1.2 Where from

Squid is available from www.squid-cache.org, however most Linux distributions contain a stable version of squid in their distributions. Usually you do not need to download the source and compile the software, however often it is necessary if you wish to enable or compile in new features or if you wish for performance enhancements.

The Squid developers make periodic releases of the source code. Each release has a version number, such as 2.5.STABLE4. The third component starts either with STABLE or DEVEL (short for development).

As you can probably guess, the DEVEL releases tend to have newer, experimental features. They are also more likely to have bugs. Inexperienced users should not run DEVEL releases. After spending some time in the development state, the version number changes to STABLE. These releases are suitable for all users. Of course, even the stable releases may have some bugs. The higher-numbered stable versions (e.g., STABLE3, STABLE4) are likely to have fewer bugs. If you are really concerned about stability, you may want to wait for one of these later releases.

2.2 Installation

As we mentioned before squid is available in most distributions, so why compile? The primary reason is that the code needs to know about certain operating system parameters. In particular, the most important parameter is the maximum number of open file descriptors. Squid’s. /configure script probes for these values before compiling. If you take a Squid binary built for one value and run it on a system with a different value, you may encounter problems. Another reason is that many of Squid’s features must be enabled at compile time. If you take a binary that somebody else compiled, and it doesn’t include the code for the features that you want, you’ll need to compile your own version anyway.

If downloading and compiling Chapter 3 of the book provides a good guide on how to do this. I have found in my own experience that a precompiled version is not good enough, especially on a busy server. The file descriptors usually end up being set too low. I have ended up re- compiling it on all of our existing boxes.

2.3 Basic Configuration

The squid conf file is very similar to other UNIX type config files, consisting of a directive and a value. Things to watch out for in particular are case sensitivity and sometimes order can be important when one directive needs to take heed of another that was before it, access rules are an example of this. The squid config file is generally stored in `/etc/squid` or `/etc` and is called `squid.conf`.

2.3.1 Userid

A userid must be set for the directive `cache_effective_user`, this is usually squid. Squid will startup as root and switch to this id. This user must be made owner of the cache directory.

2.3.2 Port

A port number should be given for the directive `http_port`, this will tell squid on what port to listen on. An `Ipaddress` can be included in this directive if your server has multiple interfaces.

2.3.3 Log Files

Make sure the log files are stored on a partition with enough space. It is handy to keep the log files on a separate partition to the cache files, this will help performance.

2.3.4 Access Controls

By default squid will deny all access to all clients. To get things working the simplest approach to take is to add a line for your pc or network such as:

```
acl mynet 192.168.0.0/16
```

and then to add a rule that allows your PC access such as:

```
http_access allow mynet
```

The order of the http_access is very important, You should put the `http_access` line just before:

```
http_access Deny All
```

2.3.5 Visible hostname

The `visible_hostname` directive should be set to the hostname of your machine. This is important as it is sometimes used in headers.

2.3.6 Cache_mgr

You should set the `cache_mgr` directive to a valid contact address; this is displayed to users when a squid error occurs.

2.3.7 Cache_mem

Set this to the amount of memory to use to store hot objects, memory is your friend in this case. As a rule of thumb on Squid uses approximately 10 MB of RAM per GB of the total of all cache_dirs (more on 64 bit servers such as Alpha), plus your `cache_mem` setting and about an additional 10-20MB. It is recommended to have at least twice this amount of physical RAM available on your Squid server.

2.4 Running Squid

2.4.1 Command Line options

There are many command line options for squid, the books goes into some detail here; useful ones to remember are:

`-v` Prints the version string.

`-z` Initializes cache, or swap, directories. You must use this option when running Squid for the first time or whenever you add a new cache directory.

2.4.2 First time

When you start squid for the first time you should do so with squid `–z` to create the cache directories then with squid `–N –d1`, `-N` for foreground operation and `–d1` for debug to check that everything is ok.

Once you see the Ready to serve requests message, test Squid with a few HTTP requests. You can do this by configuring your browser to use Squid as a proxy and then open a web page. If Squid is working correctly, the page should load as quickly as it would without using Squid. Alternatively, you can use the `squidclient` program that comes with Squid:

```
% squidclient http://www.squid-cache.org/
```

2.4.3 Starting squid in scripts and as a daemon

Most prebuilt systems that come with distributions have all the startup scripts in place.

2.4.4 Reconfiguring a running squid

When you change the config files on a running system you will need to tell squid to re-read the config file and re-configure itself. This can happen more often than you might think. For example an ACL that controls who can do what based on a charging model will change often and squid must re-read the config file each time. This is done with the command:

```
squid –k reconfigure
```

2.4.5 Rotating log files

The squid log files can get very big, it is very important to rotate these logfiles at periodic intervals. This can be done with a:

```
squid –k rotate
```

2.5 Access controls

Access controls are a very important part of the squid system, they are very flexible and can be used to great effect in all manner of ways.

The basic format of the access rule is:

```
acl name type value1 value2 ..
```

Squid knows about the following types of ACL elements:

src

Source (client) IP addresses.

dst

Destination (server) IP addresses.

myip

The local IP address of a client's connection.

srcdomain

Source (client) domain name.

dstdomain

Destination (server) domain name.

srcdom_regex

Source (client) regular expression pattern matching.

dstdom_regex

Destination (server) regular expression pattern matching.

time

Time of day, and day of week.

url_regex

URL regular expression pattern matching.

urlpath_regex

URL-path regular expression pattern matching, leaves out the protocol and hostname.

port

Destination (server) port number.

myport

Local port number that client connected to.

proto

Transfer protocol (http, ftp, etc).

method

HTTP request method (get, post, etc).

browser

Regular expression pattern matching on the request's user-agent header.

ident

String matching on the user's name.

ident_regex

Regular expression pattern matching on the user's name.

src_as

Source (client) Autonomous System number.

dst_as

Destination (server) Autonomous System number.

proxy_auth

User authentication via external processes.

proxy_auth_regex

User authentication via external processes.

snmp_community

SNMP community string matching.

maxconn

A limit on the maximum number of connections from a single client IP address.

req_mime_type

Regular expression pattern matching on the request content-type header.

arp

Ethernet (MAC) address matching.

rep_mime_type

Regular expression pattern matching on the reply (downloaded content) content-type header. This is only usable in the `http_reply_access` directive, not `http_access`.

external

Lookup via external acl helper defined by `external_acl_type`.

2.5.1 Notes

Not all of the ACL elements can be used with all types of access lists (described below). For example, `snmp_community` is only meaningful when used with `snmp_access`. The `src_as` and `dst_as` types are only used in `cache_peer_access` access lists.

The arp ACL requires the special configure option –`enable-arp-acl`. Furthermore, the ARP ACL code is not portable to all operating systems. It works on Linux, Solaris, and some BSD variants. The SNMP ACL element and access list require the –`enable-snmp` configure option.

Some ACL elements can cause processing delays. For example, use of `src_domain` and `srcdom_regex` require a reverse DNS lookup on the client's IP address. This lookup adds some delay to the request.

Each ACL element is assigned a unique name. A named ACL element consists of a list of values. When checking for a match, the multiple values use OR logic. In other words, an ACL element is matched when any one of its values is a match.

You can't give the same name to two different types of ACL elements. It will generate a syntax error.

You can put different values for the same ACL name on different lines. Squid combines them into one list.

2.6 Access Lists

There are a number of different access lists:

http_access

Allows HTTP clients (browsers) to access the HTTP port. This is the primary access control list.

http_reply_access

Allows HTTP clients (browsers) to receive the reply to their request. This further restricts permissions given by `http_access`, and is primarily intended to be used together with the `rep_mime_type` acl type for blocking different content types.

icp_access

Allows neighbor caches to query your cache with ICP.

miss_access

Allows certain clients to forward cache misses through your cache. This further restricts permissions given by `http_access`, and is primarily intended to be used for enforcing sibling relations by denying siblings from forwarding cache misses through your cache.

no_cache

Defines responses that should not be cached.

redirector_access

Controls which requests are sent through the redirector pool.

ident_lookup_access

Controls which requests need an Ident lookup.

always_direct

Controls which requests should always be forwarded directly to origin servers.

never_direct

Controls which requests should never be forwarded directly to origin servers.

snmp_access

Controls SNMP client access to the cache.

broken_posts

Defines requests for which squid appends an extra `CRLF` after `POST` message bodies as required by some broken origin servers.

cache_peer_access

Controls which requests can be forwarded to a given neighbor (peer).

2.6.1 Notes

An access list rule consists of an allow or deny keyword, followed by a list of ACL element names.

An access list consists of one or more access list rules.

Access list rules are checked in the order they are written. List searching terminates as soon as one of the rules is a match. If a rule has multiple ACL elements, it uses AND logic. In other words, all ACL elements of the rule must be a match in order for the rule to be a match. This means that it is possible to write a rule that can never be matched. For example, a port number can never be equal to both 80 AND 8000 at the same time. To summarise the acl logics can be described as: http_access allow|deny acl AND acl AND ... OR http_access allow|deny acl AND acl AND ... OR ... If none of the rules are matched, then the default action is the opposite of the last rule in the list. Its a good idea to be explicit with the default action. The best way is to use the all ACL. For example: acl all src 0/0 http_access deny all See http://www.squid-cache.org/Doc/FAQ/FAQ-10.html for common senarios. Testing acls squid -k parse will check your acl's and report any problems. Section 14.01 Regular expressions A regular expression (regex) is an incredibly useful way of matching strings in other strings. Like all powefull things they can get a little complicated. Regexes are often used in languages like Perl, where they make processing of large text files easy. Regular expressions in squid are case-sensitive by default. If you want to match both upper or lower-case text, you can prefix the regular expression with a –i . Using regular expressions allows you to create very flexible access lists. Since regular expressions are used to match text strings, you can use them to match words, partial words or patterns in URLs or domains. The url_regex acl type is used to match any word in the URL. Here is an example: Denying access to sites with the word sex in them acl porno url_regex -i sex 30 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs http_access deny porno http_access allow all the line below will match sex and drugs acl badURL url_regex -i sex.*.drugs$ Section 14.02

Always direct A good rule to have in your proxy server is one that forces the cache to go direct to local servers. # acls for my network addresses acl myinternal src 192.168.1.0/24 http_access allow myinternal http_access deny all always_direct allow myinternal You use the no_cache option to specify uncachable requests. For example, this makes all responses from origin servers in the 10.1.1.0/24 network uncachable: acl Local dst 10.1.1.0/24 no_cache deny Local Section 14.03 Custom error messages You can customize the existing error messages, you can also create new error messages and use these in conjunction with the deny_info option. For example, lets say you want your users to see a special message when they request something that matches your pornography list. First, create a file named ERR_NO_PORNO in the /usr/local/squid/etc/errors directory. That file might contain something like this: <p> Our company policy is to deny requests to known porno sites. If you feel you've received this message in error, please contact the support staff (support@this.company.com, 555-1234). Next, set up your access controls as follows: acl porn url_regex "/usr/local/squid/etc/porno.txt" deny_info ERR_NO_PORNO porn http_access deny porn (additional http_access lines ...) 31 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs XV. The disk subsystem and cache parameters Section 15.01 Cache_dir The cache_dir directive is one of the most important in squid.conf. It tells Squid where and how to store cache files on disk. The cache_dir directive takes the following arguments: cache_dir scheme directory size L1 L2 [options] (a) Scheme Squid supports a number of different storage schemes. The default (and original) is ufs. Depending on your operating system, you may be able to select other schemes. You must use the —enable-storeio=LIST option with ./configure to compile the optional code for other storage schemes. (b) Directory The directory argument is a filesystem directory, under which Squid stores cached objects. Normally, a cache_dir corresponds to a whole filesystem or disk partition. It usually doesn't make sense to put more than one cache directory on a single filesystem partition. Furthermore, I also recommend putting only one cache directory on each physical disk drive. (c) Size The third cache_dir argument specifies the size of the cache directory. This is an upper limit on the amount of disk space that Squid can use for the cache_dir (d) L1 L2 For the ufs, aufs, and diskd schemes, Squid creates a two-level directory tree underneath the cache directory. The L1 and L2 arguments specify the number of first- and second-level directories. The defaults are 16 and 256, respectively Section 15.02 Disk Space Watermarks The cache_swap_low and cache_swap_high directives control the replacement of objects stored on disk. Their values are a percentage of the maximum cache size, which comes from the sum of all cache_dir sizes. For example: cache_swap_low 90 cache_swap_high 95 32 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs As long as the total disk usage is below cache_swap_low, Squid doesn't remove cached objects. As the cache size increases, Squid becomes more aggressive about removing objects. Under steady-state conditions, you should find that disk usage stays relatively close to the cache_swap_low value. XVI. Hierarchies As discussed before hierarchies can be very useful, they can also be dangerous if proper care is not taken. Section 16.01 Squid.conf settings for hierarchies The cache_peer directive has quite a few options. proxy-only This option instructs Squid not to store any responses it receives from the neighbor. This is often useful when you have a cluster and don't want a resource to be stored on more than one cache. weight= n This option is specific to ICP/HTCP. ttl= n This option is specific to multicast ICP. no-query This option is specific to ICP/HTCP. default This option specifies the neighbor as a suitable choice in the absence of other hints. Squid normally prefers to forward a cache miss to a parent that is likely to have a cached copy of the particular resource. Sometimes Squid won't have any clues (e.g., if you disable ICP/HTCP with no-query). In these cases, Squid looks for a parent that has been marked as a default choice. round-robin This option is a simple load-sharing technique. It makes sense only when you mark two or more parent caches as round-robin. Squid keeps a counter for each parent. When it needs to forward a cache miss, Squid selects the parent with the lowest counter. 33 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs multicast-responder This option is specific to multicast ICP. closest-only This option is specific to ICP/HTCP. no-digest This option is specific to Cache Digests. no-netdb-exchange This option tells Squid not to request the neighbor's netdb database. Note, this refers to the bulk transfer of the RTT measurements, not the inclusion of these measurements in ICP miss replies. no-delay This option tells Squid to ignore any delay pools settings for requests to the neighbor. login= credentials This option instructs Squid to send HTTP authentication credentials to the neighbor. It has three different formats: login =user:password This is the most commonly used form. It causes Squid to add the same username and password in every request going to the neighbor. Your users don't need to enter any authentication information. login=PASS Setting the value to PASS causes Squid to pass the user's authentication credentials to the neighbor cache. It works only for HTTP basic authentication. Squid doesn't add or modify any authentication information. If your Squid is configured to require proxy authentication (i.e., with a proxy_auth ACL), the neighbor cache must use the same username and password database. In other words, you should use the PASS form only for a group of caches owned and operated by a single organization. This feature is dangerous because Squid doesn't remove the authentication credentials from forwarded requests. 34 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs login =* :password With this form, Squid changes the password, but not the username, in requests that it forwards. It allows the neighbor cache to identify individual users, but doesn't expose their passwords. This form is less dangerous than using PASS, but does have some privacy implications. Use this feature with extreme caution. Even if you ignore the privacy issues, this feature may cause undesirable side effects with upstream proxies. For example, I know of at least one other caching product that only looks at the credentials of the first request on a persistent connection. It apparently assumes (incorrectly) that all requests on a single connection come from the same user. connect-timeout= n This option specifies how long Squid should wait when establishing a TCP connection to the neighbor. Without this option, the timeout is taken from the global connect_timeout directive, which has a default value of 120 seconds. By using a lower timeout, Squid gives up on the neighbor quickly and may try to send the request to another neighbor or directly to the origin server. digest-url= url This option is specific to Cache Digests. allow-miss This option instructs Squid to omit the Cache-Control: only-if-cached directive for requests sent to a sibling. You should use this only if the neighbor has enabled the icp_hit_stale directive and isn't using a miss_access list. max-conn= n This option places a limit on the number of simultaneous connections that Squid can open to the neighbor. When this limit is reached, Squid excludes the neighbor from its selection algorithm. htcp This option designates the neighbor as an HTCP server. In other words, Squid sends HTCP queries, instead of ICP, to the neighbor. Note that Squid doesn't accept ICP and HTCP queries on the same port. When you add this option, don't forget to change the icp-port value as well. HTCP support requires the —enable-htcp option when running ./configure. 35 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs carp-load-factor= f This option makes the neighbor, which must be a parent, a member of a CARP array. The sum of all f values, for all parents, must equal 1. CARP support requires the —enable-carp option when running ./configure. Section 16.02 Up/Down Squid uses various techniques to see if a neighbor/peer is up or down, this can involve DNS query (does the hostname resolve), TCP and UDP queries. One must be careful if the never_direct directive is used as if the neighbors/peers are all down the cache will not contact the origin directly. Section 16.03 Selection Often you wish to direct queries to certain neighbors only, squid provides great facilities to do this. Cache_peer This directive will define an access cache for a neighbor 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 cache_peer_domain This directive will allow you to specify domains that must be forwarded cache_peer sa-cache.my.org parent 3128 3130 cache_peer_domain sa-cache.my.org parent .br .cl .ar .co .ve … never_direct This will tell squid to never go directly to the sites but to go via the hierarchy. never_direct allow acl always_direct 36 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs This tells squid to always direct through the origin server Hierarchy_stoplist This is a list of strings that when found in a URI will result in a direct fetch. hierarchy_stoplist ? cgi-bin nonhierarchical_direct and prefer_direct These two control the way squid handles nonhierarchical and hierarchical requests, are they send direct or not. Section 16.04 ICP and Cache digests This will be done from the book, chapter 10.6 and 10.7 XVII. Authentication Authentication is something most Universities find very useful. This enables you to track usage and perform proper bandwidth management. There are many authentication helpers available. Section 17.01 Config The auth_param directive controls every aspect of configuring Squid's authentication helpers. The different methods (Basic, Digest, NTLM) have some things in common, and some unique parameters. The first argument following auth_param must be one of basic, digest, or ntlm. I'll cover this directive in detail for each authentication scheme later. In addition to auth_param, Squid has two more directives that affect proxy authentication. You can use the max_user_ip ACL to prevent users from sharing their username and password with others. If Squid detects the same username coming from too many different IP addresses, the ACL is a match and you can deny the request. For example: acl FOO max_user_ip 2 acl BAR proxy_auth REQUIRED http_access deny FOO http_access allow BAR Section 17.02 Basic Authentication 37 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs Basic authentication is a mechanism for client and server authentication and is simple yet insecure. It transmits the user's username and password to the squid server as a base64 encoded string. If the network is not a switched network someone could intercept a username and password using a packet sniffer. In a switched network the threat is much less, however it is still there. Even so because of the simplicity and compatibility with a wide range of browsers and systems, I know many companies and Universities that to this day use Basic Authentication. There are various helper program that can be used with Basic authentication, we will cover some in the exercises. Examples are LDAP, NCSA, MSNT and PAM. The way squid implements authentication helpers is simple and new helpers can be developed easily. Section 17.03 HTTP Digest Authentication HTTP digest authentication is much more secure than Basic but more complicated. The user agent instead of sending the clear test password sends a message digest (MD5) of the user's password and other information. The squid server then generates a hash using the same information and compares it. A problem with Digest Authentication is that it is not supported by 100% of browsers. Section 17.04 NTLM Authentication NTLM authentication is a Microsoft authentication method, this protocol has been reversed engineered and is supported by squid. Section 17.05 External ACLs Squid has a feature called external ACL's. This can be used to write your own helpers to provide authentication. Certain information is written to the helper who then issues an OK or ERR. The helper can be written in any scripting language. XVIII. Log Files The most useful log to most administrators is the access.log, this log lists the activity of the users accessing the squid cache. The file contains various fields: 1: timestamp The completion time of the request, expressed as the number of seconds since the Unix epoch (Thu Jan 1 00:00:00 UTC 1970), with millisecond resolution. Squid uses this format, instead of something more human-friendly, to simplify the work of various log file processing programs. 38 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs You can use a simple Perl command to convert the Unix timestamps into local time. For example: perl -pe 's/^d+.d+/localtime($&)/e;' access.log 2: response time For HTTP transactions, this field indicates how much time it took to process the request. The timer starts when Squid receives the HTTP request and stops when the response has been fully delivered. The response time is given in milliseconds. The response time is usually 0 for ICP queries. This is because Squid answers ICP queries very quickly. Furthermore, Squid doesn't update the process clock between receiving an ICP query and sending the reply. While time values are reported with millisecond resolution, the precision of those entries is probably about 10 milliseconds. Timing becomes even less precise when Squid is heavily loaded. 3: client address This field contains the client's IP address, or hostname if you enable log_fqdn. For security or privacy reasons, you may want to mask a part of client's address out using the client_netmask directive. However, that also makes it impossible to group requests coming from the same client. 4: result/status codes This field consists of two tokens separated by a slash. The first token, result code, classifies the protocol and the result of a transaction (e.g., TCP_HIT or UDP_DENIED). These are Squid-specific codes. The codes that begin with TCP_ refer to HTTP requests, while UDP_ refers to ICP queries. The second token is the HTTP response status code (e.g, 200, 304, 404, etc.). The status code normally comes from the origin server. In some cases, however, Squid may be responsible for selecting the status code. These codes, defined by the HTTP RFC. 5: transfer size This field indicates the number of bytes transferred to the client. Strictly speaking, it is the number of bytes that Squid told the TCP/IP stack to send to the client. Thus, it doesn't include overheads from TCP/IP headers. Also note that the transfer size is normally larger than the response's

Content-Length. This value includes the HTTP response headers, while Content- Length does not. 39 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs These properties make the transfer size field useful for approximate bandwidth usage analysis but not for exact HTTP entity size calculations. If you need to know a response's Content- Length, you can find it in the store.log file. 6: request method This field contains the request method. Because Squid clients may use ICP or HTTP, the request method is either HTTP- or ICP-specific. The most common HTTP request method is GET. ICP queries are always logged with ICP_QUERY. See Section 6.1.2.8 for a list of HTTP methods Squid knows about. 7: URI This field contains the URI from the client's request. The vast majority of logged URIs are actually URLs (i.e., they have hostnames). Squid uses a special format for certain failures. These are cases when Squid can't parse the HTTP request or otherwise determine the URI. Instead of a URI/URL, you'll see a string such as "error:invalid-request." For example: 1066036250.603 310 192.0.34.70 NONE/400 1203 GET error:invalid-request - NONE/- - Also in this field look out for whitespace characters in the URI. Depending on your uri_whitespace setting, Squid may print the URI in the log file with whitespace characters. When this happens, the tools that read access.log files may become confused by the extra fields. When logging, Squid strips all URI characters after the first question mark unless the strip_query_terms directive is disabled. 8: client identity Squid can determine a user's identity in two different ways. One is with the RFC 1413 ident protocol; the other is from HTTP authentication headers. Squid attempts ident lookups based on the ident_lookup_access rules, if any. Alternatively, if you use proxy authentication (or regular server authentication in surrogate mode), Squid places the given username in this field. If both methods provide Squid with a username, and you're using the native access.log format, the HTTP authentication name is logged, and the RFC 1413 name is ignored. The common log file format has separate fields for both names. 9: peering code/peerhost The peering information consists of two tokens, separated by a slash. It is relevant only for requests that are cache misses. The first token indicates how the next hop was chosen. The 40 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs second token is the address of that next hop. When Squid sends a request to a neighbor cache, the peerhost address is the neighbor's hostname. If the request is sent directly to the origin server, however, Squid writes the origin server's IP address or its hostname if log_ip_on_direct is disabled. The value NONE/- indicates that Squid didn't forward this request to any other servers. 10: content type The final field of the default, native access.log is the content type of the HTTP response. Squid obtains the content type value from the response's Content-Type header. If that header is missing, Squid uses a hyphen (-). If you enable the log_mime_headers directive, Squid appends two additional fields to each line: 11: HTTP request headers Squid encodes the HTTP request headers and prints them between a pair of square brackets. The brackets are necessary because Squid doesn't encode space characters. The encoding scheme is a little strange. Carriage return (ASCII 13) and newline (ASCII 10) are printed as r and n, respectively. Other non-printable characters are encoded with the RFC 1738 style, such that Tab (ASCII 9) becomes %09. 12: HTTP response headers Squid encodes the HTTP response headers and prints them between a pair of square brackets. Note that these are the headers sent to the client, which may be different from headers received from the origin server. This file can be used for statistics and billing, in the exercises I will show you how to import this data into a mysql database for reporting purposes. Many scripts can be found at http://www.squid-cache.org/Scripts/ XIX. Monitoring Squid Section 19.01 The Cachemgr Interface The cachemgr interface is a very useful program that can be used to get various pieces of information from a squid server. There is a command line component called squidclient. If you run squidclient mgr:info it will supply informative information on the current status of your squid. 41 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs There are many options to this program, running squidclient –p (port of your squid) mgr will display them. There is also a web cgi script that will display the squidclient info to a web page. The book goes into detail about the various information returned. The cachmgr info can be very useful in solving problems with your squid. XX. Server accelerator Mode As discussed before, server accelerator mode is usually used in conjunction with a router or switch that will forward requests through to the squid server. XXI. Delay Pools Delay pools can be very useful to manage bandwidth. They work by limiting the amount of bandwidth returned for cache-misses. Delay pools work by using ‘buckets’. There are three types of pools, class1, class2 and class3. Class 1 pools have a single bucket for all clients; the clients will be limited by the settings of these pools. This is useful when you wish to restrict a single network. Class 2 pools have a single aggregate bucket and then 256 individual buckets; this means that you can restrict the entire network and each of the clients on a class C network Class 3 pools have a single aggregate bucket and then 65536 (Class B). A bucket has a bucket size and a refill rate. The bucket size will determine how much the user gets at wire speed, when the bucket is empty the user will be supplied the page at the refill rate speed. If there is no activity and the bucket is not full the bucket will fill up at the refill rate. With a little configuration buckets can be very powerful; I have scripts that adjust the bucket size according to the line utilization at our site. The nice thing is that the buckets give some measure of fairness to the internet connection. If the current rate is 4Kb/s a user running a download manager with 5 sessions open will still only get 4Kb/s across his 5 sessions, so on saturated links you do not have the situation where a single user will kill the link for other users. 42 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs You can have different users assigned to different buckets so some users could have fast access while some have slow. This can also be done on a site basis, so some sites could have preference over others. Section 21.01 Conf File Options delay_pools The delay_pools directive tells Squid how many pools you want to define. It should go before any other delay pool-configuration directives in squid.conf. For example, if you want to have five delay pools: delay_pools 5 delay_class You must use this directive to define the class for each pool. For example, if the first pool is class 3: delay_class 1 3 Similarly, if the fourth pool is class 2: delay_class 4 2 In theory, you should have one delay_class line for each pool. However, if you skip or omit a particular pool, Squid doesn’t complain. delay_parameters Finally, this is where you define the interesting delay pool parameters. For each pool, you must tell Squid the fill rate and maximum size for each type of bucket. The syntax is: delay_parameters N rate/size [rate/size [rate/size]] The rate value is given in bytes per second, and size in total bytes. If you think of rate in terms of bits per second, you must remember to divide by 8. Note that if you divide the size by the rate, you’ll know how long it takes (number of seconds) the bucket to go from empty to full when there are no clients using it. A class 1 pool has just one bucket and might look like this: delay_class 2 1 delay_parameters 2 2000/8000 For a class 2 pool, the first bucket is the aggregate, and the second is the group of individual buckets. For example: delay_class 4 2 43 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs delay_parameters 4 7000/15000 3000/4000 Similarly, for a class 3 pool, the aggregate bucket is first, the network buckets are second, and the individual buckets are third: delay_class 1 3 delay_parameters 1 7000/15000 3000/4000 1000/2000 delay_initial_bucket_level This directive sets the initial level for all buckets when Squid first starts or is reconfigured. It also applies to individual and network buckets, which aren’t created until first referenced. The value is a percentage. For example: delay_initial_bucket_level 75% In this case, each newly created bucket is initially filled to 75% of its maximum size. delay_access This list of access rules determines which requests go through which delay pools. Requests that are allowed go through the delay pools, while those that are denied aren’t delayed at all. If you don’t have any delay_access rules, Squid doesn’t delay any requests. The syntax for delay_access is similar to the other access rule lists except that you must put a pool number before the allow or deny keyword. For example: delay_access 1 allow TheseUsers delay_access 2 allow OtherUsers Internally, Squid stores a separate access rule list for each delay pool. If a request is allowed by a pool’s rules, Squid uses that pool and stops searching. If a request is denied, however, Squid continues examining the rules for remaining pools. In other words, a deny rule causes Squid to stop searching the rules for a single pool but not for all pools. cache_peer no-delay Option The cache_peer directive has a no-delay option. If set, it makes Squid bypass the delay pools for any requests sent to that neighbor. Section 21.02 Issues Fairness: There are no guarantees about fairness when users are sharing buckets. Accuracy: As the rate limiting is done at the application layer it will not be as exact as rate limiting done at the protocol layer. 44 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs Subnets: The current implementation assumes that you are using a /24 class C subnets. 45 Caching Workshop , Kenya , Feb/March 2006 by Richard Stubbs Links Autoconfig reference http://wp.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html Squid Site www.squid-cache.org Squid FAQ http://www.linux-faqs.com/faq/squid-faq/FAQ.php

References Squid the Definitive Guide, Duane Wessels, O'Reilly ISBN 0-596-00162-2 Web Caching, Duane Wessels, O'Reilly ISBN 1-56592-536-X Squid Documentation – www-squid.org/Doc http://squid-docs.sourceforge.net/latest/book-full.html