ICAP - The Internet Content Adaption Protocol

Enno Davids

<enno.davids@metva.com.au>

ABSTRACT

As the Internet has matured the so has the need for organizations to apply policy to the content that comes in off the net to our various domains. This policy takes the form of admonitions about what we may surf, what is safe to surf or even substitution of content for what we might otherwise see. It may involve compression of content, automated translation from one language to another or indeed one markup style to another or even something as prosaic as merely suppressing annoying ads. ICAP is a means to allow standard web proxy servers to offer such services without compromising their basic caching mission.

1. ICAP defined

ICAP, the Internet Content Adaption Protocol is the formalization of a protocol which has been in use for some time by a number of vendors. It was first standardized in an informal group called the ICAP Forum which consists of vendors with an interest in products that could leverage such a standard.

In more recent times the OPES (Open Pluggable Edge Services) working group who are working to define mechanisms for extensions to be plugged into existing proxies, caches and servers, recognized the prior art in ICAP by documenting it as a preliminary step to their larger task. This document is RFC 3507 and serves as a neutral standard in the manner of other internet RFCs of the past.

So what is ICAP? Simply put ICAP allows an administrator to define some policy to be applied to internet content. Internet content in this case is specifically Web content as ICAP is intended primarily for use in web proxies at the network boundary of an organization, although there is no reason why other services such as email or instant messaging couldn't use it too.

The policy that ICAP implements can be applied either before content is acquired (i.e. before the request passed out of your network) or after the content arrives (i.e. examining/ altering the payload in a server response). Content may be passed unaltered, modified or indeed blocked in its entirety.

The model of an ICAP installation is shown in Figure 1 below.

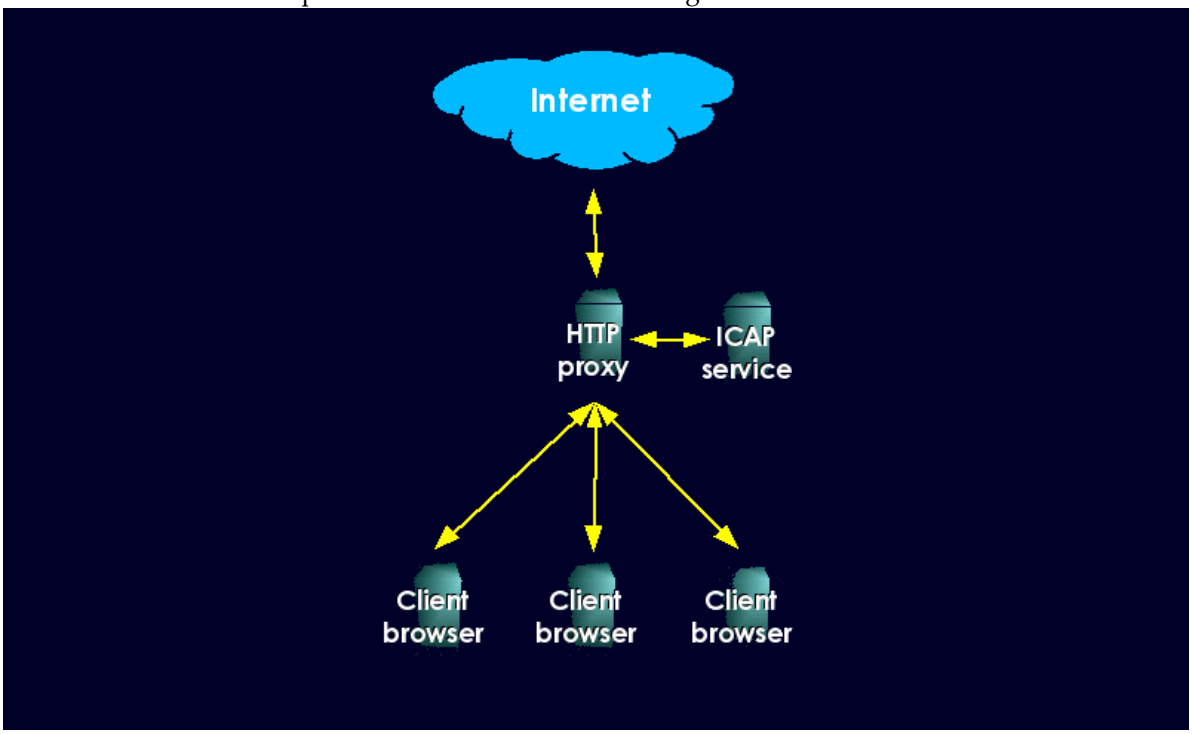

Figure 1: Model of an ICAP installation.

A typical installation consists of an ICAP server operating in conjunction with a proxy server. The proxy remains in the role it has had for some time now and its configuration remains largely unchanged. (In particular this means that access control and authentication of proxy and client remain unchanged and thus the disruption of adding ICAP to an existing infrastructure may be low.) As requests and responses pass through the proxy it hands them off to the nearby ICAP server which deals with them according to its function and configuration.

Note that ICAP is a largely stateless protocol and is passed across a socket connection. This means all the traditional load balancing and resourcing solutions for HTTP work equally well for ICAP. It means that ICAP installations can scale seamlessly from small models where the server is co-resident on the same host as the proxy to large installations where a proxy farm might rely on a load balanced ICAP server farm and any intermediate variations that make sense for the size of the load being offered.

It may be worthwhile briefly introducing a new piece of jargon which the ICAP RFC uses here, namely that of "*origin server*". With the plethora of servers in these installations it becomes important to differentiate them in some simple manner. Notably then we have the proxy or cache server which the user's browser interacts with, the ICAP server which the proxy deals with and the remote web site or *origin server* which the user is actually trying to access. ICAP also deals with systems it refers to as *surrogates*, essentially reverse proxies or accelerators which have some affinity with the origin server(s), but for most purposes the presence or absence of these systems is essentially invisible to the systems we care about.

Having seen how a web proxy might access an ICAP server leads to the natural question of what does 'adaption' mean in this context. Simply put, the ICAP protocol allows for the inspection and subsequent acceptance, modification or complete rejection of internet content both before and after the request takes place.

Adaption thus may encompass any number of operations such as:

- content substitution. The content is completely replaced by some alternate and presumably more preferable content. This may be complete replacement or merely some partial re-working of what's already there. Examples of this might include:
    - natural language translation. Content may be subject to automated translation. This may be an alternative when web-sites themselves fail to offer content in a manner that is useful to your population. We can presuppose some form of higher quality automated translation than those we've seen online to date!
    - banner ad suppression? Banner ads, indeed most forms of ad can be suppressed in a variety of ways based on where they are served from, the size and shape of them, whether they are static or dynamic images and so forth.
- re-coding markup and content. Content is unaltered from the human perspective but has been changed in its internal representation perhaps. Examples of this might be:
    - HTML --> WAP. An automated procedure might be developed to translate 'plain' HTML from a web site into something more suitable for portable or handheld devices.
    - GIF --> PNG. Modern browsers all support the PNG graphics format but few web sites avail themselves of it in spite of superior performance and file size. An ICAP server could do this conversion automatically and save bandwidth on an internal network.
- access control
    - rated content. Content which identifies itself as for an age restricted audience may be blocked in an automated manner.
    - work/non-work content. This is similar to the above but perhaps more arbitrary. Blocking cricket and football sites is often done for instance due to the high volumes of network traffic they often generate.
- virus scanning. Perhaps the most obvious and useful task on which ICAP is put to use today.
- compression. Most modern browsers are equipped to understand content which has been compressed with UNIX gzip. In networks where bandwidth is at a premium (say ocean cable to a remote station) it may make sense to compress any uncompressed content which is presented.

So having seen all these motherhood statements lets look at the protocol a little more closely. ICAP defines three types of transactions/ operations which it can perform. The first of these is the OPTIONS request. This is the only mandatory operation an ICAP server must

implement and serves principally as a mechanism for a proxy to determine which of the other ICAP services is available in an ICAP server.

Given that OPTIONS is asynchronous to other operations of the server and indeed as most servers are typical UNIX servers which can handle multiple requests on multiple sockets simultaneously, the OPTIONS packet can also be used by network monitoring systems as a 'ping' for checking the health of an ICAP server.

The next operation type is the REQMOD operation. This as its name implies is an opportunity for an ICAP server to examine a content request and optionally pass a modified form of it back to the requesting proxy server (or indeed quash the request entirely). REQMOD is passed the entire request from a browser and may examine it. As it has the entire request it may examine any or all of it in making its determination and indeed choose to modify any or all of the request. Examples here include simple things like stripping cookies from known data miners like doubleclick from the request to wholesale rewriting of the request such as changing a request for a forbidden piece of content to an error page alerting the user that their request was blocked.

ICAP responds with a set of return codes which are clearly inspired by HTTP:

- an ICAP protocol error status
- a HTTP error - primarily used for poorly formed requests
- a 204 'no modifications' response
- a 200 response (we presume here that some changes were made to the request)

Finally Figure 2 below pulls all this together for us.

A few quick final notes on REQMOD then. As we noted, the ICAP status bears a familial resemblance to HTTP. This is both good and bad as we will later see. Good in that the familiarity means we innately have some grasp of what is going on sooner than we otherwise might and bad because these statuses have nothing to do with the underlying HTTP transactions status. In REQMOD this is not a big deal but later it will be. Suffice it to say that the ICAP status should not be confused with the status of an encapsulated HTTP transaction which is being inspected and possibly adapted. An extreme example is an ICAP server could virus scan a HTTP response and deduce there were no viruses. ICAP would signal 200, all is well and squid might then accept an error response page from an origin server. (which was virus free of course!)

The final operation type is the RESPMOD. This is likely the workhorse of the ICAP protocol. It allows an ICAP server to examine the data returned from an origin server as its response to the HTTP request. This in fact is where the virus scanning of the last example takes place, on the body of the HTTP response. Once again, the server may respond with an error, a modified response body or indeed pass the data unchanged.

Specifically the statuses it may respond with are:

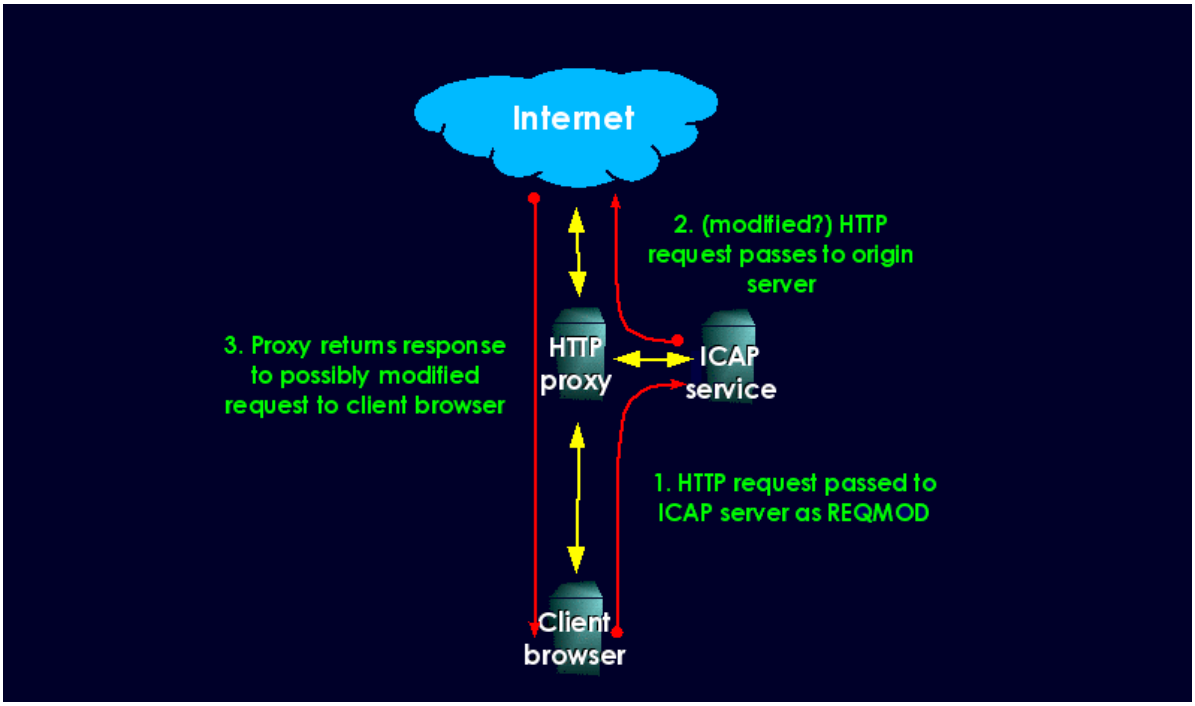

Figure 2: The REQMOD operation.

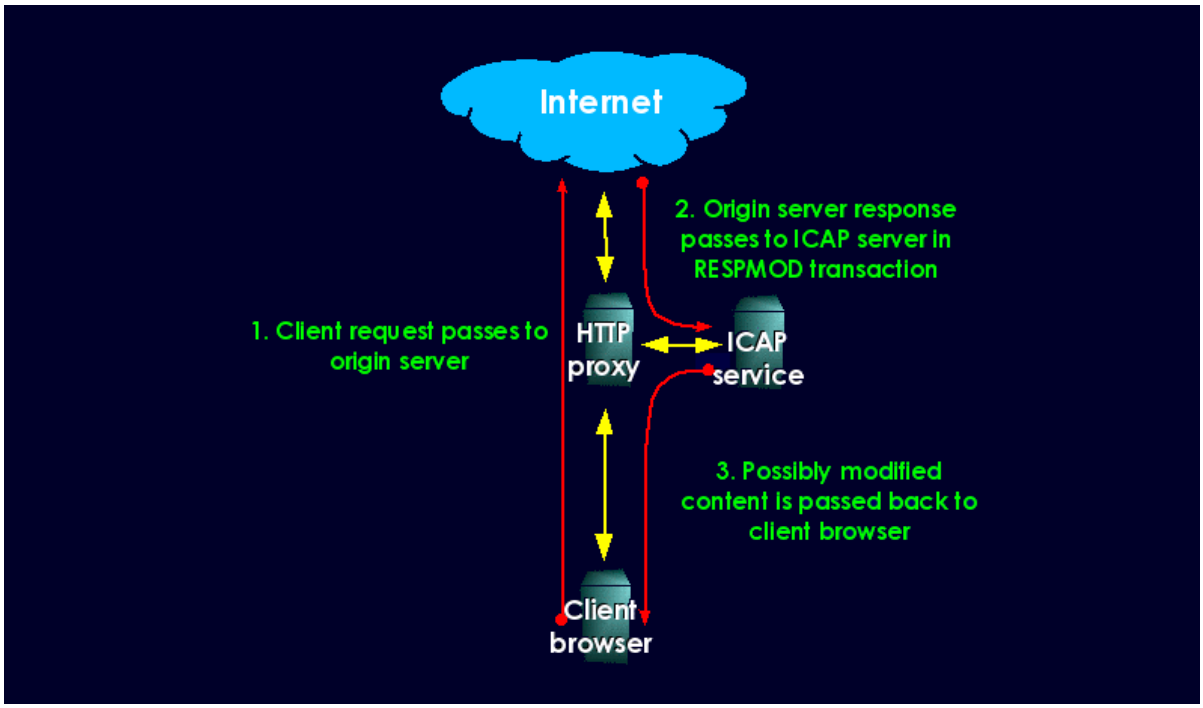

Figure 3: The RESPMOD operation.

- an ICAP error
- a 204 'no modifications' response
- a 200 response (which may or may not contain the original HTTP response data)

Once again we note that an ICAP 200 response may encapsulate an error response from an origin server and the two should not be confused. Also note that the ICAP server can also do some more subtle things like modifying cookies, or re-writing the caching parameters (to affect either your proxy, your browser or both).

Note that there is a significant potential for delay in this scenario. Typically this manifests as web surfers browsing large objects which must be completely downloaded before they can be passed to ICAP for a determination as to whether they can be passed on to the surfer. For jobs like compression there is little that can be done here. The entire object must be available before it can be compressed and passed on. This is significantly at odds with the behaviour we are used to seeing where our browser sees a steady stream of data as it arrives and squid for example only adds it to the cache when it has completely arrived.

Some ICAP operations, particularly inspection tasks like virus scanning may benefit from examining only some small portion of the returned content before passing judgment. To this end, ICAP has a piece of functionality called preview where an ICAP server can be presented with some small portion of an origin server response (as soon as it is available) and can perform its work on this previewed data). Thus if you're fetching a CD-ROM ISO image your virus scanner might only examine the first few Mb before declaring it is happy. A much better result than waiting for all 700Mb to be downloaded, scanned and then passed to you all at once. In fact, some browsers and most individuals will suffer a timeout in such circumstances! 🙂

There is of course no equivalent to this for REQMOD but then again your requests are unlikely to be this big to begin with.

So having looked at ICAP why should we choose to use it? Well firstly if you want functions like this, this is a standard way of getting them. It allows you get these functions without adding too much complexity into squid itself (following the tool model UNIX pioneered in some sense) and indeed because of its network communications model allows its load to spread to a separate host from the proxy when the need arises or indeed to more than one other host in high load environments. It is, as has been noted both open and scalable and can be efficiently load balanced (with the high availability benefits this can bring).

All of this is great stuff. So we then should ask why might I not choose to make use of ICAP? The simplest reason may be the most compelling. You've already solved all these problems. Its not broke and you choose not to fix it.

Or you may find you have a partial, but 'good enough' solution in place and the complexity, downtime or user re-training doesn't bear thinking about. Most sys admins have plenty to

do already and adding this may not be high on their list of things to try...

Or you may have a proprietary solution in place and are locked in, in some manner. Once again, if its working, this may not even be a bad thing.

Finally, you may prefer to wait for OPES. They will be the be-all and end-all in this space… according to them at least. (The cynics will note the difference in approach of course - ICAP is codifying an existing protocol so people can keep using what already works. Meanwhile off in a corner a committee is off designing the new replacement that will be all things to all men. Can anyone say X.400?)

The most compelling real reason you might choose to forgo ICAP at the moment sadly is a very simple one. Squid and ICAP together are not yet ready for the big time. There are BUGS! They are currently show stoppers, at least when I last tried them in March'04. As with all open source though we can expect that things will improve. My problems may have been related to the ICAP server side rather than the squid portion of the solution so squid with a commercial ICAP server may be robust and reliable. This is one of those areas where you have to evaluate a potential solution and move on from there…

For my part, I'm comfortable that if it doesn't work now, it soon will.

2. ICAP and Squid

For the most part if you want to use ICAP with squid you have to do it the hard way. This pretty much reflects the not yet ready for prime time state of things I suspect. It may also be due to people not having heard much about ICAP at the current time.

ICAP support is thus not yet widely available in the common squid source tarballs or indeed in the pre-built binaries you may be tempted to download. Happily, building squid yourself is fairly straightforward on most of the platforms you're likely to try it on.

Start by getting a copy of the squid sources with the ICAP support added in. Late last year the state of the art was to grab the squid source tree and apply the ICAP patches that are (still) available. Now things have improved as Duane has merged the patch into the official squid CVS repository at sourceforge. The recommended way to get ICAP enabled squid sources then is to fetch the whole shebang from sourceforge. A special tag is available of 'icap-2_5' which can be used in a checkout to grab the source you care about. Note that the squid 3 C++ rewrite is not yet considered stable so although squid 2.5 is officially described as somewhere between stable and moribund, it does seem to form the basis of most new feature work at this time.

So, to grab the source tree you'll be wanting something not unlike the command line shown here:

```
cvs -d \
 ":pserver:anonymous@cvs.sourceforge.net:/\
 cvsroot/squid" checkout -r icap-2_5 squid
```

The build process is a fairly straightforward GNU autoconf style build only slightly complicated by the fact that as it comes out of CVS you in fact need to generate the configure script yourself first. A configure.in is provided and indeed a bootstrap.sh file does most of the hard work for you. You may have to play with versions of the autoconf support tools though depending on your platform.

After this it pretty much a matter of using the standard configure and make commands and going off for a few leisurely coffees. When doing the configure remember to use --enable-icap-support. In fact you'll likely want a bunch of other options too depending on which squid features you are enamored of.

Having built an ICAP enabled squid you need to add some config to your squid.conf to tell it where to find its ICAP service. If you're installing from scratch the sample config file already has all the potential config in it in the usual manner for you to edit and customize to your needs. The brief list of config items looks like this:

```
icap_enable         <on|off>
icap_preview_enable  <on|off>
icap_preview_size    -1
icap_check_interval  300
icap_send_client_ip  <on|off>
icap_send_auth_user  <on|off>

icap_auth …
    icap_auth_scheme Local://%u
    icap_auth_scheme LDAP://ldap-server/\
        cn=%u,dc=company,dc=com
    icap_auth_scheme WinNT://nt-domain/%u
    icap_auth_scheme \
        Radius://radius-server/%u

icap_service …
    icap_service serv1 reqmod_precache 0 \
        icap://icap1.mydomain.net:1344/reqmod
    icap_service serv2 respmod_precache 0 \
        icap://icap2.mydomain.net:1344/respmod

icap_class classname serv1 serv2

icap_access classname allow|deny [!]aclname…
```

Some of this like icap_enable is self explanatory whilst other bits are more arcane. Rather than belabor it here I defer to the comments in the sample config file which are pretty clear.

Having built squid, we now need to find ourselves an ICAP server of some sort. Two choices are available. Open Source and Commercial. In the realm of Open Source the pickings are fairly slim in the same way that there are few if any OSS AV scanners out there. Happily the ICAP Forum does make a sample server available in source form for us to use, although in its default form it is little more than demo code.

In the arena of commercial offerings things are quite different though with many companies offering solutions for varying sorts of money. Some offer free limited run time downloads for those of you who'd like to try before you buy. Check the ICAP Forum's partners page for clues as to who might be worth talking to.

The sample server is a fairly straightforward beast. It comes as a tarball and implements a fairly straightforward UNIX socket based server, albeit one that makes use POSIX threads. (Which may be sticking point on some architectures).

The sample server implements some demonstration features, namely:

- banner ad removal based on popular banner image sizes
- a 'valley girl' filter c.f. the valspeak or chefspeak filters from comp.sources.misc a decade ago…

Unlike squid, the sample ICAP server is not GNU autoconf based, but follows the simpler (albeit much less flexible) older style of building binaries of just using make. Prior to doing so you should examine the headers to enable those demo features you'd like to try out and then crank the handle on the make process. On most modern machines the make will complete fairly rapidly as the are only a handful of source files to process. At this stage you're ready to enter the debug cycle!

It must be noted that the sample server's demos are not all that exciting. They are of course mostly present to serve as worked examples of functions that are attached to the framework that is the sample server. As such there is a sample REQMOD service, which mostly just echoes back the requests it is passed to process and a choice of two RESPMOD based services. The first is the banner stripper we mentioned which simply replaces any images it sees which are one of the standard sizes it recognizes as belonging to banner ads with a substitute image. The second sample is the valspeak filter which one imagines is meant to show what a natural(?!?) language translation feature might look like.

What is also in place of course are fairly simple set of hooks that you can hang any functionality you might be inclined to implement on. This is the real purpose of the sample server and in this it is at least easy to integrate into.

Once you've plugged in your code, it should just run right? Well *should* may be the operative word as some of the coding in the sample server is quite poor and may in fact offer you problems of its own. You may also want to check that the license is compatible with your intended use. Its an open license but its not GNU which may or may not concern you. (In fact its similar in style to the old style BSD license with the attribution clause for those who care…)

If all has gone well, run your sample server, update your squid configuration to point to it and see how well your luck is holding up. This roughly the point I came unstuck 6 months ago. This is one of those classic your mileage may vary deals though as quite clearly things have moved on since then. The new squid configuration seems to have some support for fail-soft operation now and automated recovery where it didn't when I tried it.

If all else fails, you can debug the squid ICAP server combo fairly well. Both ICAP and HTTP are protocols with useful amounts of conversation taking place in plain text. You can telnet to your ICAP server and fire in commands (more likely you'll use a tool like netcat to fire in 'canned' test messages) and watch what happens with packet sniffing tools. Ethereal is a great choice here as it can decode both HTTP and ICAP conversations in its most recent incarnations. Finally of course, both squid and the ICAP server can produce copious debug info if asked nicely.

3. Other resources

3.1 The ICAP Forum

As I noted early on, ICAP is owned by an industry body called The ICAP Forum who loosely are a bunch of vendors who want to offer such products in this space and convince customers they have choices by opening their interfaces. And from what little I've seen, this is working well for them so far.

Their web site is at:

http://www.i-cap.org/

They also run a mailing list for discussions mostly about the standard itself. Its fairly low volume and it hosted at yahoogroups.com with all the issues that may imply.

http://www.yahoogroups.com/group/icap-discussions/

3.2 Squid

Squid too has a couple of mailing lists that may be of help. These are mediated from the main Squid web site

http://www.squid-cache.org/

The two main lists of interest are

- squid-users
- squid-dev

Note that these list have high signal to noise ratios and fairly low volume so subscribing is no great hardship at the current time. Things may of course change though… Sign up is in the usual manner for such mailing lists.

3.3 Proprietary support

As noted previously, lots of vendors are offering proprietary solutions… both in parts and as a whole system approach. (i.e. proxy and ICAP servers as a bundle).

Some of these vendors are: (once again see the ICAP Forum membership page for more names)

- founders/hosts: NetApp, WebWasher, Akamai
- participants: 81 members (and counting) including most of the AV community, a few CMS vendors and even the ad vendors we might be trying to filter!

4. Conclusions

ICAP looks to be an excellent technology for adding subtle and perhaps useful policy enforcement to all sorts of networks. Having said this, the open source offerings are meagre and clearly parts of the solution still need work. Things are on the improve though, albeit slowly, and it can only be a matter of time till sufficient maturity is reached for production use and ICAP becomes the solution of choice.