ÈCOLE NATIONALE SUPÉRIEURE DES TÉLÉCOMMUNICATIONS

PROJECT REPORT

Development of an open source HTTP proxy with ICAP support for the POESIA project.

Department : Computer Science and Networks

Students : Fares TRIKI and Riadh ELLOUMI

Monitors : Sylvie VIGNES (ENST) and Basile STARYNKEVITCH (CEA)

March - June 2003

Contents

Résumé (FR)

POESIA est un projet européen de filtrage de contenu Internet (Web, email et news) pour protéger les enfants des contenus violents, pornographiques ou racistes. Dans ce projet, on a besoin d'un proxy HTTP qui va servir les clients Web en pages passées au crible par les filtres de POESIA. Pour communiquer avec le module de filtrage de POESIA, Shweby utilise le protocole ICAP.

Shweby est donc un proxy HTTP et un client ICAP conçu dans le but d'être intégré dans le projet POESIA. Néanmoins, Shweby peut s'intégrer avec tout serveur ICAP pour d'autres applications comme la traduction de texte, le filtrage du contenu, la détection de virus...

Pour le développement de Shweby, nous sommes partis d'un proxy HTTP, nommé " Middleman " et distribué sous la licence GNU GPL. Dans nos développements nous nous sommes focalisés à avoir dans Shweby les caractéristiques pré-requises pour être intégré dans le projet POESIA. Il faut que Shweby permette de changer les niveaux de filtrage (filtrage pour les enfants, filtrage pour les adolescents, filtrage pour adultes), qu'il soit robuste, sécurisé et qu'il puisse éventuellement cacher les données.

La conception du noyau de Shweby est basée sur les threads POSIX et sur la mise en tampon de tous les messages HTTP. Cette conception s'adapte parfaitement au contexte du logiciel POESIA qui est conçu pour servir quelques dizaines d'ordinateurs et ne prévoit pas le passage à l'échelle.

Le projet POESIA est actuellement en libre diffusion dans la communauté du libre. Il est hébergé par la plate forme Sourceforge.net et son adresse web est `http://shweby.sourceforge.net`. La prochaine étape dans le projet est le développement d'une interface utilisateur pour la configuration de Shweby.

Introduction

Protecting the young people from the harmful content of the Internet is becoming a very important concern. The Internet is growing fast, and there are no or very few control mechanisms over harmful content. Moreover, this content is financed by powerful organizations like pornographic industry. As a result, it is now urging to find how we could protect efficiently our youth.

Nowadays, there are many proprietary filtering software solutions that can efficiently filter the harmful content. But these solutions are expensive and may not be affordable to little educational establishments. They are also mostly designed to filter English-speaking web sites. There is a need for an open source filter because we can afford easily such software and we can adapt it to specific languages and cultural differences between countries.

The POESIA project [1] (Public Opensource Environment for a Safer Internet Access) is an open source Internet content filter funded by the European Commission. This project aims to become the standard filtering solution deployed by educational institutions and for home use. Open extensible architecture enables POESIA software to filter harmful content in several channels (Web, Email, News) and to deploy many kinds of filters: natural language filters (English, Spanish and Italian), image filters, Javascript filters...

The web channel of POESIA needs an HTTP proxy server. That proxy server will be operate on behalf of the browser client and will download the web page before scanning its content by calling filter devices. The Shweby project aims to develop an open source stable HTTP proxy in order to be integrated in POESIA software set. We will try in this paper to describe the requirements of POESIA for an open source HTTP proxy server and to explain how Shweby was designed in order to obey these requirements.

The use of Shweby will not be restrained in POESIA software. The conformance to open Internet standards will enable Shweby to be an open source solution for content adaptation. This will be achieved by implementing the ICAP (Internet Content Adaptation Protocol) in Shweby proxy server. The ICAP protocol [2] is designed by the IETF [3] (Internet Engineering Task Force) in order to allow web proxies to contact ICAP servers and ask them for a content modification of the HTTP requests or the HTTP responses. The content adaptation has many useful applications such as text translation, virus scanning, adapting web pages to small devices (PDAs, Cell phones), advertising banner insertion or removal, etc.

Open source community has welcomed the announcement of Shweby's first releases. There were some interesting feedbacks about these releases. We will try to promote developing, test-

ing and documenting Shweby in the open source community. The future steps in developing Shweby will be testing interoperability with most ICAP servers, even open source or commercial, and to define a framework for developing proxylets.

Chapter 1

Presentation of POESIA project

1.1 Overview

POESIA software is designed to filter Internet content in three channels (web, email and news). In this project, we will be interested only in web channel. The architecture of POESIA is composed of some different devices: All the HTTP flow will go through the *HTTP proxy* which will ask the *POESIA monitor* for the content filtering. The communication between the proxy and the monitor is based on the ICAP protocol. The monitor communicates with some specific filters (Spanish, English, French; language detectors; image filters; Javascript; URL filters...) and decisors or decisions mechanisms.

POESIA is targeted for situations where groups of computers are used for Internet browsing, such as classrooms, libraries, computer centers and business. The number of computers will not exceed 20, because POESIA is installed on a single computer and CPU resources bound filtering capabilities.

1.2 ICAP communication

The ICAP protocol is fully explained in RFC 3507 [2]. When the proxy receives an HTTP request or an HTTP reply, it encapsulates it in an ICAP request and sends it to the monitor, which plays the role of the ICAP server. The monitor replies by an ICAP message that encapsulates a modified HTTP request or response.

The purpose of ICAP protocol is content adaptation, but we use it only in filtering purpose in POESIA software. The monitor will not "modify" the ICAP request, it will only accept or reject it. Here are two examples:

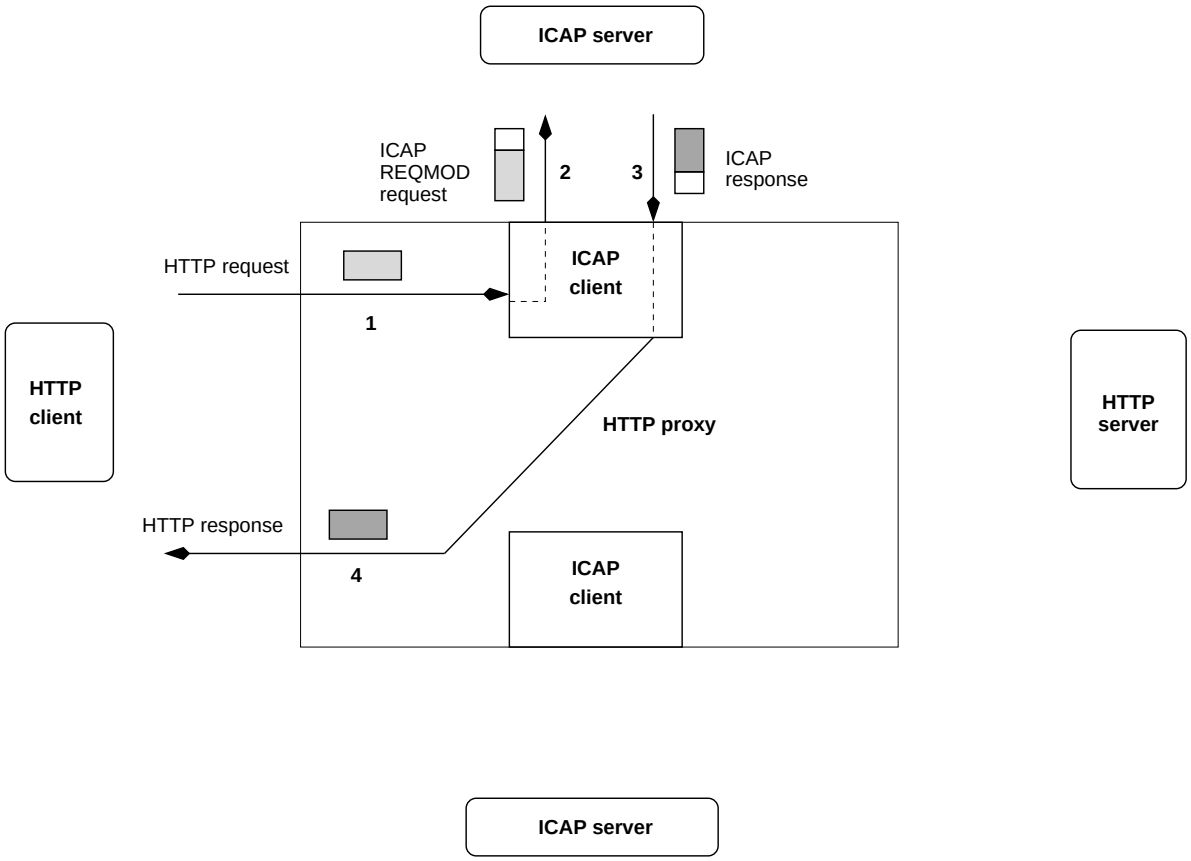

Figure 1.1: Example: blocking unauthorized content at the request time

1.2.1 Example: blocking unauthorized content at the request time, figure 1.1

The client requests a porno web page, for example `www.sex.com`. The client browser sends a request to the proxy (step 1).

```
GET /index.html HTTP/1.1
Host: www.sex.com
User-Agent: Mozilla
Connection: keep-alive
```

The proxy will encapsulate the HTTP request in an ICAP request and send it to the monitor (step 2):

```
REQMOD icap://monitor.school.com/filter ICAP/1.0
Host: monitor.school.com
Encapsulated: req-hdr=0, null-body=68

GET /index.html HTTP/1.1
Host: www.sex.com
User-Agent: Mozilla
```

The ICAP request begins with REQMOD, which is the ICAP method used. REQMOD means request modification and RESPMOD means response modification. There is a specific ICAP header "Encapsulated" which indicates what's the nature of the HTTP message (req for requests), if there is a body (null-body when no body exists) and the length of the HTTP headers (68). We assume that the monitor interrogates a database of unauthorized web servers and finds the url (`www.sex.com`) on it. Thus the ICAP reply will look like this (step 3):

```
ICAP/1.0 200 OK
Date: Mon, 10 Jan 2000 09:55:21 GMT
Server: POESIA-Monitor/1.0
Connection: close
ISTag: "W3E4R7U9-L2E4-2"
Encapsulated: res-hdr=0, res-body=(...)

HTTP/1.1 403 Forbidden
Date: Wed, 08 Nov 2000 16:02:10 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Thu, 02 Nov 2000 13:51:37 GMT
ETag: "63600-1989-3a017169"
Content-Length: 58
Content-Type: text/html
```

```
3a
Sorry, you are not allowed to access that naughty content.
0
```

According to the ICAP protocol, when the proxy receives an HTTP response for an HTTP request in an ICAP transaction, it will deliver the response to the web client (step 4):

```
HTTP/1.1 403 Forbidden
Date: Wed, 08 Nov 2000 16:02:10 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Thu, 02 Nov 2000 13:51:37 GMT
ETag: "63600-1989-3a017169"
Content-Length: 58
Content-Type: text/html
Connection: close

Sorry, you are not allowed to access that naughty content.
```

Notice in this example the use of chunked transfer-encoding in ICAP communication.

1.2.2 Example: blocking unauthorized content at the response time, figure 1.2

When the web client requests a web content that we cannot identify as forbidden or not from its URL, we must download the content before scanning it. In this example, the requested url is `www.unknown-server.com`. The client sends its request to the proxy like this (step 1):

```
GET /index.html HTTP/1.1
Host: www.unknow-server.com
User-Agent: Mozilla
```

The proxy forwards it to the monitor like in the previous example. The monitor will not block the request, and will reply like this (step 3):

```
ICAP/1.0 200 OK
Date: Mon, 10 Jan 2000  09:55:21 GMT
Server: POESIA-Monitor/1.0
Connection: close
ISTag: "W3E4R7U9-L2E4-2"
Encapsulated: res-hdr=0, null-body=(...)

GET /index.html HTTP/1.1
Host: www.unknow-server.com
User-Agent: Mozilla
```

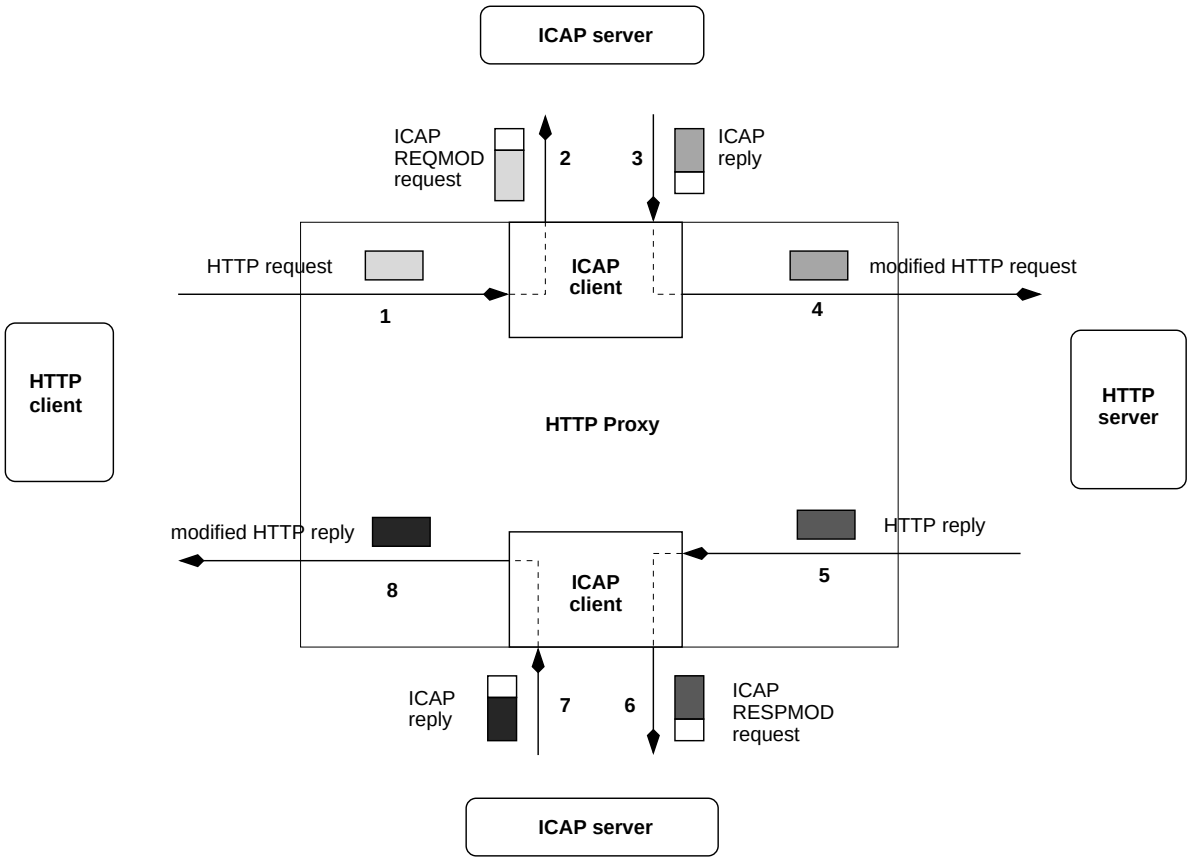

Figure 1.2: Example: blocking unauthorized content at the reply time

The monitor hasn't changed anything in the request. In the ICAP protocol, the ICAP server can say that he will not change the HTTP request (or the response), if the proxy asks him to do so, by providing a 204 "No content" response. In this case, the proxy must store the HTTP message in a buffer before asking the ICAP server for an adapatation. We suppose now that the proxy has contacted `www.unknown-server.com` and got some naughty content. The ICAP transaction will look like this:

From the proxy to the monitor (step 6):

```
RESPMOD icap://icap.example.org/satisf ICAP/1.0
Host: icap.example.org
Encapsulated: req-hdr=0, res-hdr=(...), res-body=(...)

GET /origin-resource HTTP/1.1
Host: www.origin-server.com
Accept: text/html, text/plain, image/gif
Accept-Encoding: gzip, compress

HTTP/1.1 200 OK
Date: Mon, 10 Jan 2000 09:52:22 GMT
Server: Apache/1.3.6 (Unix)
ETag: "63840-1ab7-378d415b"
Content-Type: text/html

1d
Here is some naughty content.
0
```

From the monitor to the proxy (step 7):

```
ICAP/1.0 200 OK
Date: Mon, 10 Jan 2000  09:55:21 GMT
Server: POESIA-Monitor/1.0
Connection: close
ISTag: "W3E4R7U9-L2E4-2"
Encapsulated: res-hdr=0, res-body=(...)

HTTP/1.1 403 Forbidden
Date: Wed, 08 Nov 2000 16:02:10 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Thu, 02 Nov 2000 13:51:37 GMT
ETag: "63600-1989-3a017169"
Content-Length: 58
Content-Type: text/html

3a
```

```
Sorry, you are not allowed to access that naughty content.
0
```

Thus the final response to the server will be (step 8):

```
HTTP/1.1 403 Forbidden
Date: Wed, 08 Nov 2000 16:02:10 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Thu, 02 Nov 2000 13:51:37 GMT
ETag: "63600-1989-3a017169"
Content-Length: 58
Content-Type: text/html
Connection: close

Sorry, you are not allowed to access that naughty content.
```

If the monitor accepts the content, it will reply with code 200 or 204. The origin content will be provided to the client.

1.3 POESIA requirements for an open proxy server

Before we develop Shweby proxy server, the proxy used in POESIA was Squid-ICAP [4]. Squid-ICAP is an ICAP enabled version of Squid [5]. Although Squid is a popular, stable and scalable HTTP proxy, Squid-ICAP has still some bugs and maintaining it is a quite hard work. The POESIA project requires a stable ICAP implementation, even not scalable. That's the reason why we have developed Shweby proxy server.

We will explain here what are the requirements of POESIA.

1.3.1 Configuration of the proxy

POESIA software can support few levels of filtering, like kid filter, teenager filter and adult filter. The filtering level is managed by the system administrator and depends on client IP addresses. We must be able to configure the proxy to a suitable filtering level and to change the configuration without restarting the proxy.

1.3.2 Caching

Because we are serving many computers with Internet content, caching will reduce the response time and will save outgoing bandwidth. We can cache the content before or after filtering. The POESIA filter will be called after the cache, because this provides more security

to users when they change. For example, when the kids take the place of the adults, we must simply change the POESIA filter and we don't need to empty the cache.

1.3.3 Buffering

The POESIA monitor requires buffering. It will wait for the whole ICAP request before it can make a decision, and the content is fully accepted or fully rejected. Given the fact that in HTTP protocol, servers may not provide the content-length header, we cannot know in advance what's the length of the delivered content, and the monitor buffers could be completely filled. It would be better to perform content buffering in the server-side of the proxy. If the received content exceeds a limit (10Mb), the proxy will reject it and send an error message to the client browser. This limitation is large enough for web browsing but it will forbid long file downloading.

1.3.4 Stability

The stability of the proxy server is an important POESIA requirement. The proxy must run for months without breaking down.

1.3.5 Security

The proxy must protect himself against hacking attacks (e.g. by Telnet). The POESIA end-user will not be allowed to bypass the proxy by changing its browser configuration. This will be achieved by transparent proxying, which means that the browser connects directly to the Internet, but POESIA gateway forwards all the requests to the proxy. Thus the proxy must support transparent requests (or direct requests). The only difference between a proxy request and a transparent request is that a proxy request contains the full URI in the first line and the transparent request contains only the path in the first line, the name of the server is provided by "Host" header in both requests.

Example of a proxy request:

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

Example of a transparent (direct) request:

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

1.3.6 Scalability

The POESIA software is targeted to a public of 20 PCs. We assume that each PC user requests a web page every 10 seconds and each web page includes 10 components, the generated load is 20 requests per second. Thus the web proxy will handle 20 request per second.

POESIA is not designed to scale to a huge public of PC clients. So it does not require a scalable web proxy.

Chapter 2

Design of the proxy server

One big advantage of developing under the GNU GPL licence is that we can start the development from another source code distributed under the same licence. In this project, called Shweby, we tried to enhance Middleman to ICAP support. Middleman [6] is an open proxy server with features designed to increase privacy and remove unwanted content. For example, Middleman enables the user to block unwanted content like banners, to remove cookies from request headers, to filter delivered content by simple scripts, etc. The first step in developing Shweby was removing these unwanted features and keeping wanted features like caching and transparent proxying.

2.1 Description of Middleman

2.1.1 Multithreading

The Middleman's design is based on multithreading. There is no thread pool in Middleman; Threads are created when clients connect to the proxy and are destroyed when clients are disconnected. Each thread serves a client connection. The proxy listens on some configurable ports. When a client makes a TCP connection to the proxy, the socket returned by `accept` is recorded in a structure called `connection`. The `connection` is attached as an attribute to a new created POSIX thread. This thread will serve the client requests coming from the TCP connection.

2.1.2 Caching

Middleman implements the HTTP caching [7]. HTTP caching requires two mechanisms: expiration model and validation model. When a client requests a cached file, the proxy must verify if the cached file is not expired. If it is not, the proxy delivers the cached file to the client as a response. If the cached file is expired, the proxy must send a conditional request to the origin web server (or to the next caching proxy). The conditional request is an HTTP request

with the header "If-Modified-Since" followed by the time when the proxy has downloaded the file. If the cached file is expired, the server replies by a fresh copy and the proxy updates its cached copy, else the server responds with a special status code (usually, 304 (Not Modified)) with no body and the proxy validates its cache entry.

2.1.3 Configuration

Middleman reads its configuration at runtime from an XML file. The structure of the XML file is fully explained in `README.html`. We can configure Middleman by its web interface: we simply request URL `http://mman` on a web browser connected to Middleman. We have changed the URL to `http://shweby`, but these types of URLs won't be accepted by system administrators because they break the addressing plan in the local network. This URL should be changed to the host machine address.

Middleman uses an embedded XML parser coded in `src/xml.c`.

2.2 Upgrading Middleman to Shweby

2.2.1 ICAP implementation

The ICAP RFC [2] introduces the concept of "ICAP services". An ICAP service is a specific resource of an ICAP server. It is identified by its URI (Uniform Resource Identifier), which is composed of the protocol name (icap), the name of the server and the path of the service. Here is an example of ICAP service URI:

```
icap://icap.net/services/tranlate?lang=fr
```

When we send an ICAP request to an ICAP service, we request a specific value added service on the encapsulated HTTP message. This service can take place at different points, called vectoring points:

- `reqmod_precache`: the request is modified by ICAP server "in its way to the cache". The request is modified before looking in the proxy cache.
- `reqmod_postcache`: the request is modified by ICAP server "in its way to origin server".
- `respmod_precache`: the origin server's reply is modified before it is stored in the cache.
- `respmod_postcache`: the reply is modified by ICAP server "in its way to the client".

We can call one or many ICAP services in each vectoring point, in a precise order. The ICAP services are grouped in ICAP classes. ICAP classes can include one or many ICAP services with their associated vectoring points. The ICAP class defines a global value added

service for the HTTP session. For example we define an ICAP class called “kids-filter” in table 2.1.

ICAP service	Vectoring point
`icap://monitor.school.net/req-filter?level=kids`	`reqmod_precache`
`icap://avirus.school.net/virus-scanning`	`respmod_precache`
`icap://monitor.school.net/res-filter?level=kids`	`respmod_postcache`

Table 2.1: Structure of the ICAP class “kids-filter”.

In the ICAP class "kids-filter", there is a filter service for requests, a filter service for responses and a virus-scanning service for responses. The virus scanning is performed before storing the file in the cache, so the cache will be not infected, but may contain porno images because response filtering is done after caching.

When a client connects to the proxy, the selected ICAP service depends on the client IP address, the proxy port to which the client is connected and the access rules stored in the XML configuration file. The selection of the ICAP class is coded in `src/main.c`.

The entity of the proxy that communicates with an ICAP service is called an ICAP client. ICAP clients are stored in structures of type `ICAP_CLIENT` and are coded in `src/icap.c`. The `connection` contains the references to theses structures in `connection->icap`

2.2.2 Adding of gzip encoding and decoding

In order to perform content adaptation, ICAP servers must systematically decompress any compressed file before analysing it and this needs CPU resources. However, the advantage of requesting compressed content from the Internet is to save the access bandwidth. Thus, it would be interesting to decompress downloaded content prior to sending them to ICAP servers.

2.2.3 Using keep-alive connections

Keep-alive connections increase performances. Middleman keeps the connection alive with HTTP servers, even if the client requests a closed connection. The alive connections are put in a socket pool in order to be used for other HTTP requests.

We have used the same mechanism in ICAP implementation. The alive ICAP connections are also put in the socket pool and reused for other ICAP sessions.

2.2.4 Testing and debugging the proxy

For the tests, we developed testing scripts with Python and asyncore library. These testing scripts are committed in `test` module under the CVS repository. Here are the test components:

- The HTTP client, coded in Python. It uses the asyncore Python library.
- The HTTP server, which based on Medusa Server [8].
- The ICAP server, which is a patched version of Network Appliance open source server. The ICAP server does not modify HTTP messages.

The HTTP client sends a request to the HTTP server through the proxy. Each request, identified by a number, makes the system work on a specific HTTP configuration. We can experiment POST requests, GET requests, chunking, keep-alive connections and closed connections.

We can test the thread synchronisation by running two HTTP clients in a loop.

Debugging is made by GDB (GNU Debugger). Memory leaks can be detected in debugging mode by logging all allocated memory addresses in the table `marray`. See `src/mem.c` for more details.

2.2.5 Configuration of Shweby and deployment scenario

We will explain here an example of deploying Shweby in POESIA environment. We consider the case of a school having two computer rooms A and B. We assume that we can identify if a computer is in room A or room B by its IP address. We assume that POESIA components (Shweby, monitor and filters) are running in a machine called "gateway" and this machine is located in the administrator office.

The room A will be occupied by kids and room B will be occupied by teenagers. The system administrator must configure POESIA to filter requests coming from room A by "kids-filter" ICAP class and requests coming from room B by "teenagers-filter" ICAP class. On his machine "gateway", the system administrator can surf the web without filtering and can also test "kids-filter" and "teenagers-filter". He can configure Shweby proxy directly from his machine by requesting the Shweby web configuration interface. POESIA filters can download web content through Shweby proxy without filtering, in order to know if a web pages contains any porn pictures.

To achieve these requirements, we configure Shweby with three listening ports. If a client connects to a port, given his IP address and the rules below (table 2.2), the proxy will perform suitable ICAP filtering. We suppose that we have already configured three ICAP services: "kids-filter", "teenagers-filter" and "bypass", which means no filtering.

Here is how these rules are coded in the XML file:

```
<!-- Access control description -->
        <access>
                <policy>deny</policy>
                <allow>
                        <enabled>true</enabled>
```

IP address	Port 4000	Port 4001	Port 4002
Gateway	configuration class = kids-filter	configuration class = teenagers-filter	configuration class = bypass
Room A	class = kids-filter	Not allowed	Not allowed
Room B	class = teenagers-filter	Not allowed	Not allowed

Table 2.2: Example of deploying POESIA: table of access rules.

```
        <comment>No filtering on Gateway:4000</comment>
        <class>bypass</class>
        <ip>127.0.0.1</ip>
        <port>4000</port>
        <access>config,proxy,connect,http,transparent</access>
</allow>
<allow>
        <enabled>true</enabled>
        <comment>to test teenagers-filter on Gateway:4001</comment>
        <class>teenagers-filter</class>
        <ip>127.0.0.1</ip>
        <port>4001</port>
        <access>config,proxy,connect,http,transparent</access>
</allow>
<allow>
        <enabled>true</enabled>
        <comment>to test kids-filter on Gateway:4002</comment>
        <class>kids-filter</class>
        <ip>127.0.0.1</ip>
        <port>4002</port>
        <access>config,proxy,connect,http,transparent</access>
</allow>
<allow>
        <enabled>true</enabled>
        <comment>Room A: filter for kids on Gateway:4000</comment>
        <class>kids-filter</class>
        <ip>137.194.34.5</ip>
        <iprange>137.194.34.8-137.194.34.50</iprange>
        <port>4000</port>
        <access>proxy,http,transparent</access>
</allow>
<allow>
        <enabled>true</enabled>
        <comment>Room B: filter for teenagers on Gateway:4000</comment>
        <class>teenagers-filter</class>
        <iprange>137.194.34.51-137.194.34.80</iprange>
        <ip>137.194.34.88</ip>
```

```
                <iprange>137.194.34.90-137.194.34.96</iprange>
                <port>4000</port>
                <access>proxy,http,transparent</access>
        </allow>
</access>
```

Chapter 3

Next Shweby developments

The Shweby proxy server is hosted on Sourceforge.net platform. Its web site is `http://shweby.sourceforge.net`. The Sourceforge platform will offer to our project a large diffusion in the open source community. We will be pleased to find people contributing to Shweby development and tests.

Shweby is not a finished product. The next step in developing Shweby is the configuration web interface.

3.1 Upgrading the Web interface to ICAP support

In release 0.9.1 of Shweby, there is a web interface obtained by typing `http://shweby` in the web browser. We must change this URL to the machine name, in order to respect the naming plan of network administrators. In addition, the web interface don't know anything about ICAP. Thus we must implement setting up ICAP classes and services, and selecting ICAP classes in the access lists.

The Middleman web interface is written on C. We think that this is not the best choice to implement a web interface. It would be simpler to write it with a scripting language (Python, Perl, ...). The web script can communicate with the proxy process by different manners (signals, CGI, embedded interpreter, etc)

3.2 Advanced ICAP features

The ICAP RFC [2] describes some mechanisms that upgrade performances and save bandwidth. The OPTIONS method allows the client to know what type of content the ICAP server is interested in. The preview mode enables the client to send a "preview" of the content, i.e. the HTTP headers and a small part of the beginning of the body. If the server needs to consider the entire file, it will ask the client to send it, else it will deliver the response. This

mode can be helpful for other types of applications like virus scanning, image conversion, etc.

3.3 Using Shweby in other applications

Our target is to run Shweby in many purposes. So we will test its interoperability with other products, even commercial products. But the non-scalability of Shweby is a handicap for its use in a large context. Thus we may upgrade the Shweby core design to more performance and scalability.

Conclusion

This project was a good introduction for us in the open source world. The open source community welcomed the announcement of Shweby and we had some interesting feedbacks about the software.

The Shweby and POESIA projects are part of a big project for enabling content services at the edge of the Networks, particularly for education and small businesses, which could not afford for expensive commercial licenses. The project of enabling content services at the edge is being standardized by the IETF working group OPES. This project is quite interesting because it allows different communities to adapt the web to their culture, for example by filtering undesired content and translating web pages. However, will this project increase web censure and limit the liberty of expression?

List of Figures

Bibliography

[1] Web site of POESIA Project, http://www.poesia-filter.org/

[2] ICAP RFC, http://www.ietf.org/rfc/rfc3507.txt

[3] IETF web site, http://www.ietf.org/

[4] Squid-ICAP web site, http://icap-server.sourceforge.net/squid.html

[5] Squid web site, http://www.squid-cache.org/

[6] Middleman project site, http://www.sourceforge.net/projects/middle-man/

[7] HTTP RFC, http://www.ietf.org/rfc/rfc2616.txt

[8] Medusa HTTP Server web site, http://www.nightmare.com/medusa/