Lappeenranta University of Technology
Department of Information Technology

Delivery system for location based information in wireless IP networks

The topic of the Thesis has been confirmed by the Departmental Council of the Department of Information Technology on April 10, 2002.

Examiner: Professor JAN VORÁČEK
Supervisor: JOUNI IKONEN, PhD.

Lappeenranta, May 10, 2002

Vladislav Kurz
Karankokatu 4 C 11
53810 Lappeenranta

Abstract

Lappeenranta University of Technology
Department of Information Technology

VLADISLAV KURZ
Delivery system for location based information in wireless IP networks

Thesis for the Degree of Master of Science in Technology
2002

This thesis presents a new way for delivering location based information to users of wireless computer networks. Information is delivered to every user in network, without any knowledge of the user's identity. As application layer protocol was chosen HTTP, which allows this system to deliver information to most users using wide variety of terminal devices.

The system works as an extension to interception web proxy, that consults various databases to decide whether some information needs to be delivered or not. The system also includes simple software for locationing of users, with precision of single access point range. Even though the presented solution aims at delivery of location based advertisements it can be easily modified to deliver any kind of information to users.

Thesis contains 52 pages, 16 figures, 1 table and 2 appendices.
Examiner: Professor JAN VORÁČEK
Supervisor: JOUNI IKONEN, PhD.
Keywords: wireless networks, location based information, content delivery, world wide web, interception proxy, cache

Tiivistelmä

Lappeenrannan teknillinen korkeakoulu
Tietotekniikan osasto

VLADISLAV KURZ

Paikkariippuvaisen tiedon jakelujärjestelmä langattomissa IP verkoissa

Diplomityö
2002

Tämä työ esittelee uuden tarjota paikasta riippuvaa tietoa langattomien tietoverkkojen käyttäjille. Tieto välitetään jokaiselle käyttäjälle tietämättä mitään käyttäjän henkilöllisyydestä. Sovellustason protokollaksi valittiin HTTP, joka mahdollistaa tämän järjestelmän saattaa tietoa perille useimmille käyttäjille, jotka käyttävät hyvinkin erilaisia päätelaitteita.

Tämä järjestelmä toimii sieppaavan www-liikenteen välityspalvelimen jatkeena. Erilaisten tietokantojen sisällön perusteella järjestelmä päättää välitetäänkö tietoa vai ei. Järjestelmä sisältää myös yksinkertaisen ohjelmiston käyttäjien paikantamiseksi yksittäisen tukiaseman tarkkudella. Vaikka esitetty ratkaisu tähtääkin paikkaan perustuvien mainosten tarjoamiseen, se on helposti muunnettavissa minkä tahansa tyyppisen tiedon välittämiseen käyttäjille.

Työ sisältää 52 sivua, 16 kuvaa, 1 taulun ja 2 liitettä.
Tarkastaja: Professori JAN VORÁČEK
Ohjaaja: TkT. JOUNI IKONEN
Hakusanat: Langattomat verkot, paikkatietoon perustuva tieto, sisällön perillesaattaminen, world wide web, sieppaava välityspalvelin, välimuisti

Abstrakt

Lappeenranta University of Technology
Department of Information Technology

VLADISLAV KURZ
Systém pro doručování pozičně závislých informací v bezdrátových počítačových sítích

Diplomová práce
2002

Tato diplomová práce presentuje nový způsob doručování pozičně závislých informací uživatelům bezdrátových počítačových sítí. Informace jsou doručovány každému uživateli v síti, bez jakékoli znalosti uživatelovy identity. Jako aplikační vrstva byl vybrán protokol HTTP, což umožňuje doručovat informace většině uživatelů s nejrůznějšími koncovými zařízeními.

Tento systém funguje jako rozšíření transparentní webové proxy, které čte data z různých databází a na jejich základě rozhoduje zda a co bude uživateli doručeno. Systém obsahuje také jednoduchý software na detekci polohy uživatelů, s přesností na buňku přístupového bodu. Presentované řešení je orientováno na doručovaní pozičně závislých reklam, ale může být snadno rozšířeno o doručování libovolných informací.

Práce obsahuje 52 stran, 16 obrázků, 1 tabulku a 2 přílohy.
Zkoušející: Profesor JAN VORÁČEK
Vedoucí diplomové práce: JOUNI IKONEN, PhD.
Klíčová slova: bezdrátové sítě, pozičně závislé informace, doručování informací, world wide web, transparentní proxy, cache

Contents

List of Figures

List of Tables

List of Abbreviations

AP Access Point

CARP Cache Array Routing Protocol

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

FTP File Transfer Protocol

GNU GNU's Not Unix

GPL General Public License

GRE Generic Routing Encapsulation

HTCP Hyper-Text Caching Protocol

HTML Hyper-Text Markup Language

HTTP Hyper-Text Transfer Protocol

ICP Internet Cache Protocol

IP Internet Protocol

ISP Internet Service Provider

LAN Local Area Network

LDAP Lightweight Directory Access Protocol

MAC Media Access Control

NAT Network Address Translation

PAC Proxy Auto Configuration

PHP PHP: Hypertext Preprocessor

RADIUS Remote Authentication Dial In User Service

RFC Request For Comments

RPC Remote Procedure Call

SDL Specification and Description Language

SMS Short Message Service

SQL Structured Query Language

SSL Secure Sockets Layer

TCP Transmission Control Protocol

UDP User Datagram Protocol

URI Unified Resource Identifier

URL Unified Resource Locator

URN Unified Resource Name

WAN Wide Area Network

WCCP Web Cache Communication Protocol

WLAN Wireless Local Area Network

WLPR Wireless Lappeenranta

WPAD Web Proxy Auto Discovery

WWW World-Wide Web

XML Extensible Markup Language

Chapter 1

Introduction

Wireless local area networks (WLAN), based on IEEE 802.11b standard, are commonly used for several years already. The most common use is to extend or replace wired networks in offices or homes. Wireless networks are preferred over wires for their easy installation and moving to new premises. Transmission speed (11 Mbit/s) is good for end user stations, price affordable, and coverage (few hundred meters) is sufficient for office use. Details about wireless LANs and IEEE 802.11 standards can be found in [1].

Another area of common use for WLAN is connecting two separated networks via wireless point-to-point link. With directional antenna, WLAN can build links even few kilometres long. Of course, direct visibility is necessary for good connection. WLAN operates in 2.4 GHz band that does not handle obstacles very well.

However most wireless networks are private, not public. Public wireless networks have appeared in last few years. Most of them at airports, hotels and shopping malls. These so-called hotspots offer Internet connectivity for visitors. Connection is paid either as a part of other service (hotel room, or frequent flyer service) or by extra payment at the place. Prices are clearly targeted at business users. (Tourists do not carry notebooks with WLAN card around much nowadays.)

Other approach are community networks [2] built by volunteers or non-profit organisation. These networks offer local connectivity, but Internet access must be arranged by other means, either individually or by some third party connected to community network and providing Internet access for a fee. Such networks may need to have a way to inform all users about important network related issues, like upcoming network

maintenance, temporary outages or new services. Also location based information such as interactive maps, timetables and advertisements may be beneficial for the network. As the network can be open to many users, administrators may not know each users e-mail to inform them, moreover users may decide not to give administrators their e-mail address in fear of spam.

Research presented in this thesis, provides a way how to deliver, or "push", information to network users. Example implementation uses HTTP and interception web cache [3] to deliver location based advertisements. Users moving around the network get localised information according to their position. For example in vicinity of theatre, tonight programme is shown, nearby restaurant may advertise its menu of the day, railway station can display actual train timetables, etc. Deploying of this technology in public wireless networks may attract more companies to join the network and sponsor building of more access points, as they will have more customers attracted to their business. In addition to location based information, also user authentication for Internet access is addressed in this thesis.

Chapter 2

Internet and push technologies

At first place let's define what is considered a push technology in the scope of this thesis. Push technology is method to send some information to the user without user's explicit request.

Most of Internet protocols are pull technologies. User sends a request to server and gets requested data in response. Typical examples can be HTTP, Gopher, DNS and FTP. FTP allows also sending of files from user to server, but it is always initiated by the user sending or receiving data, not the server.

2.1 E-mail

The oldest and most popular push technology is e-mail. Users receive e-mail without explicitly requesting them. In other words, the user receiving e-mail has no means of choosing what and when will be delivered to him in e-mail. Unfortunately this feature of e-mail has been soon discovered by many people and abused to send unsolicited e-mail known as *spam* or junk mail.

This behaviour led to implementation of e-mail filters and mail blacklists. Filters can automatically discard e-mail coming from known spammers or perform some kind of heuristic to determine whether particular e-mail may be spam or not. Filters can also

protect against e-mail worms, but that is out of the scope of this thesis. However best protection against spam is keeping the e-mail addresses as secret as possible.

E-mail is very useful, even though it is abused a lot. It is one of the few ways how to send messages to other people when the sender wants. And the sender can be person as well as machine. It is quite common that Internet servers inform their administrators about problems, attacks or failures by e-mail or other push technology, for example by sending short message (SMS) to administrator's mobile phone. E-mail is also sometimes used by Internet Service Providers (ISP) to inform their users about important news concerning their networks.

2.2 Instant messaging

In last few years several *instant messaging* services appeared on Internet. The first and perhaps the most popular is Mirabilis ICQ (pronounced as "I seek you"). Others are AOL Instant Messenger (America OnLine), MSN Messenger (Microsoft), Yahoo! Messenger, and Jabber.

Unlike e-mail, these technologies are proprietary. Jabber is the only exception being developed as free software under GNU General Public License. These technologies allow users to send messages and files to each other in real time. Online users see received messages on their screens immediately and can respond. Also real time chat is possible, where messages are transfered immediately as they are written, char by char or line by line. Some technologies allow also voice communication (a kind of Voice over IP).

Users can search in "white pages" to find if their friends are registered and can watch when other users are online. Many of these messengers have gateways to other messaging technologies like e-mail, SMS or competing instant messengers.

Unfortunately spammers are abusing these technologies as well as e-mail. In my opinion any future public push technology will be abused by spammers. These technologies also invade user's privacy because other people can watch if they are online.

2.3 WLAN hot-spot differences

Now, let's think about a WLAN hot-spot open for public use. (Of course with some limitation or payment for connection.) In such a network, the operator does not necessarily

know the e-mail address or ICQ number of the users. Moreover, user's may not even use e-mail or ICQ or anything similar.

Using instant messengers is also very unlikely because of wide variety of incompatible and proprietary protocols. Special client software is necessary for instant messaging. Network operator cannot rely on fact, that users will use some of these technologies. Also users cannot be forced to use instant messaging at all.

In this situation, push technology must be based on protocol that is used by most people using standard clients on their devices. The most popular protocol used on Internet is without doubt HTTP, the core of World-Wide Web. But HTTP is traditionally a pull technology, not push.

To push some content to users via HTTP, it is necessary to intercept their communication and modify it. So users will not need any special setup of their software, operator does not need any information about users. Of course, this technology will require special network infrastructure to intercept the traffic and deliver some content.

The next chapter analyses different methods of traffic modification and delivery of push information.

Chapter 3

Analysis of methods of content delivery

In this chapter several methods of content delivery are analysed. Demo versions of most of them were created for evaluation purposes.

3.1 Response rewrite

In first demo of push technology all HTTP responses were *rewritten* on the fly to include some additional information. Of course, modification was possible only for messages with content-type: text/html.

Demo version was created using free WWW-proxy software FilterProxy [4], which was originally designed as a personal proxy. Its main purpose is to filter out unwanted stuff like advertisement banners, JavaScript [5], cookies or to improve the appearance of poorly designed webpages by removing small fonts or blinking headings. FilterProxy is written in Perl [6] and takes full advantage of Perl regular expressions to find out where to apply particular rules.

For the purpose of content delivery a simple rule was written. It is adding a piece of HTML [7] code right behind the `<BODY>` tag. This code included banner image and link to project's webpage. The FilterProxy rule follows:

```
rewrite tag <body> as
<body>
  <center>
    <a href="http://www.it.lut.fi/comlab/wlan/indexes.html">
      <img src="http://www.it.lut.fi/comlab/wlan/wlan1.jpg">
    </a>
  </center>
  <hr>
```

The problem is that FilterProxy does not have an *insert* rule so it is necessary to rewrite the whole `<BODY>` tag, thus losing all its attributes like background colour, image or text colour, unless they are defined in style sheet.

After setting the browser to use FilterProxy, some unexpected behaviour was experienced. Due to the way the filter works, the banner appears in every frame shown by browser. This is fine when there are just a few clearly determined frames. Unfortunately many web designers (including our project webmaster) use frames in abundance for visual aligning of text pieces. Normally these frames are not seen but delivered content shows where they are and totally destroys the precise page layout. As an example of this unwanted effect, two screenshots of home page of town of Lappeenranta are included as figures 3.1 and 3.2.

Another strange effect was that many advertising banners have been replaced by our own. Quick look into the source code of affected pages revealed that they were included as inline frames using `<IFRAME>` tag. Therefore they come in form of a short HTML page that was, of course, rewritten by the FilterProxy rule.

It is almost impossible for proxy to know whether particular page is a part of multiframe web page, so this technique cannot be much improved. Moreover modifying other people's web pages and distributing this modified content might violate copyright law.

3.2 Pop-up windows

JavaScript offers the possibility to create additional browser windows with arbitrary content and configurable appearance. Many web-servers use JavaScript to create "pop-up" windows with advertisements. This was considered as another possibly useful technique.

Figure 3.1: Original web page

Figure 3.2: Rewritten web page with pushed information

This method is very similar to the previous one. It was needed to *rewrite* the HTTP responses to include a piece of JavaScript code that shows our pop-up window. This was accomplished using this FilterProxy rule:

```
rewrite tag <head> as
<head>
  <script>
    banner=window.open("http://www.lut.fi/~kurz/school/DT/Banner.html",
                       "banner", "width=400, height=100");
    banner.focus();
  </script>
```

The JavaScript code is then executed when pages are loaded by web browser. If the pop-up window already exists, it is just focused, i.e. made visible on top of all other windows. It must be noted that the file *Banner.html* must be accessed without filtering. Otherwise the browser could run into infinite recursion.

This approach has several advantages when compared to previous method of rewriting the HTML code:

- It does not modify the appearance of downloaded web pages. Even if the page consists of multiple frames, only one pop-up window is shown.
- Rewriting the `<HEAD>` tag is much less problematic than rewriting `<BODY>` tag, because it is commonly used without any attributes.

However there are also disadvantages:

- Users can easily get rid of the pop-up window by switching off JavaScript support in their browser.
- There are still browsers that do not support JavaScript at all, especially those on PDA devices cannot show pop-up windows.
- Different browsers have different bugs that make JavaScript less portable.

At last the same legal problem appears as with the previous method – distributing modified content, that can be considered as copyright violation in some circumstances.

3.3 Redirect with frames

This method was inspired by several free web hosting services. They do not require users to put banners on their sites. Instead they add a small frame at the top or bottom of the page that includes the advertisements.

Usually they do it so that the users' home pages have official address in the form of `http://user.free.web/` where a page defining two frames is stored. One of the frames then contains advertisements, the other one points to real home page address that is something like `http://pages.free.web/~user/`. By direct access to the later address one can easily get rid of the advertisements.

As we cannot affect the addresses of all pages on Internet, we have to find another way. We need to redirect only the first page to the frame definition. Then all browsing continues inside the frame, and the information frame is left intact. First idea was to use `Referer` header to detect the beginning of user's browsing.

3.3.1 `Referer` triggered redirect

RFC 2616 [8] defines `Referer` header like this:

> The Referer[sic] request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained (the "referrer", although the header field is misspelled.) The Referer request-header allows a server to generate lists of back-links to resources for interest, logging, optimised caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referer field MUST NOT be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.

According to the above mentioned specification, a FilterProxy module was developed, that checked whether incoming request had a `Referer` header. If it had none, the request was redirected to a frame definition page that included pushed information and originally requested page. As the pages included in a frame are referred by the frame definition page, they should have a `Referer` header and the proxy module would forward them unmodified.

Requests without `Referer` header were redirected to *Framer.shtml* with originally requested address as query string. That page defines two frames. One points to our info-page, *Banner.html*, the other to originally requested page. The resulting page is shown on figure 3.3.

Testing this proxy settings revealed that different browsers are not consistent in sending the `Referer` header. As specified in RFC 2616 it is not sent when the URI is typed on the keyboard or selected from user's bookmarks. However RFC does not *require* sending this header at all. It suggests to give users an option whether this header should be sent or not. Current practice is that most browsers send it. Some lightweight browsers do not send it at all. Only Opera allows user to switch `Referer` header sending on and off.

Problem is that some browsers do not send `Referer` header for some special cases when it could be sent. Example situations of this behaviour are requests for style-sheet files, background images, flash animations and redirects. Some browsers even did not send `Referer` when loading framed pages thus getting recursively redirected by our filter. This inconsistencies forced us to drop the idea of using the `Referer` header as a trigger for redirect.

3.3.2 Access time triggered redirect

As HTTP headers are not reliable and they can be also easily forged by users, another independent way of finding whether pushed information is shown to user, must be used. One way is to make the pushed information to reload regularly and to mark the time to some database. Self reloading of web page (for example every minute) can be accomplished by including this tag into HTML header: `<META HTTP-EQUIV="Refresh" CONTENT="60">` Then this time can be checked and user's browsing redirected to framer page if the time is longer than the reload period.

Using a database for storing reload times can be extended to store also authentication, user location and other necessary stuff. In WLAN hot-spot network the proxy module could do processing like this:

1. Check if the user is authenticated. If not, show login page.
2. Check if the self-reloading frame reloaded in last minute. If not, show framer page.

3. Frame was shown recently, so leave the request unmodified.

However framed approach has also some disadvantages. One is that the browser's title bar and URL input box show title and address of the framer page instead of the one the user is browsing. Thus bookmarking and navigation is complicated. Other issue is that the banner always occupies some space. For usual laptops or desktops this is not a problem but handheld devices have really small screens. Some web pages render really bad on such screens and frame with banner will make the situation even worse. Furthermore not all browsers support the self-reloading `<META>` tag.

3.4 Full page redirect

Another easy way of forcing users to see some information is to redirect their HTTP requests to some other page that contains the information we want them to see. Of course, we have to allow users to browse WWW, so we cannot redirect every request. Good place to put the push information is for example a login page (if it exists). In case of our test network, users are already being redirected to login page where they have to authenticate before using Internet connection, so this seems easy to do.

This may not be feasible solution, as pushed information has to appear independently of user authentication. User authentication is valid usually for several hours, while we may want pushed information to show with period of several minutes or irregularly. This is also because we want the pushed information to be location sensitive. For location based information, the redirection can be triggered also by sensing the location of users and redirecting the traffic only if their location has changed.

For this method popular open-source web cache Squid [9] was used. It is widely used, actively developed, freely available and freely modifiable. Squid has implemented interface to *redirector*. Redirector is a program that takes URL, source IP and HTTP request method on input and returns (possibly different) URL. Squid can be configured to consult a redirector prior to satisfying a request. Redirectors are commonly used to extend Squid's access control lists. Anyone can write redirector that does arbitrary processing to determine the return value.

Squid usually fetches documents from the changed URL and returns them to clients instead of originally requested data. FilterProxy module from previous section did the same. This lead to confusion of some browsers, when they had cached the framer page,

believing it to be the requested data, and ended up in infinite recursion. Squid redirectors can indicate permanent or temporary redirection to clients by returning HTTP response 301 (Moved Permanently) or 302 (Found) instead of just fetching different document.

This method was implemented as a special redirector for Squid that checks users last and current position and determines which information page shall be shown to the user. Location of users was detected using locationing software developed by Antti Seppänen as a part of his MSc. thesis [10]. The redirector process works as follows:

1. Read and parse the HTTP request
2. Ask locationing server (using XML-RPC) for user's location
3. Query SQL database for last user's location
4. If the two locations are different, write new location to database and redirect user's browser to information (advertisement) page.
5. If the locations are same or user's location is unknown let the request intact.

SDL diagram for the above algorithm is shown on figure 3.5. The data flow is described on figure 3.4. Ovals denote processes, disk icon is database table, thin lines show the direction of data flow, thick dashed lines show the redirection of HTTP requests.

This redirector could also access database of authenticated users and redirect to login pages if necessary. For non-mobile users that are using only one access point or wired connection, redirector can also check the last time of redirection and push the information page occasionally, for example once a day or so.

Drawback of this method is that user's location is being searched during every single request, i.e. when one Web page is downloaded, the locationing will be done several times, once for each included image. Thus for this method it is absolutely necessary to have fast locationing service, preferably in form of database where current locations are stored and updated when users move and reassociate with different access points.

Another problem might be user that is moving around while accessing Internet, i.e. traveller in a bus or taxi. Such user can be changing access points so often that the redirection to info pages would make web browsing practically impossible. On the other hand this method gives us the opportunity to make customised information pages for different browsers, especially with respect to limited display size. After considering all methods from this chapter, decision was made to implement this one, because it is the most portable and flexible method.

Figure 3.3: Framed web page with pushed information

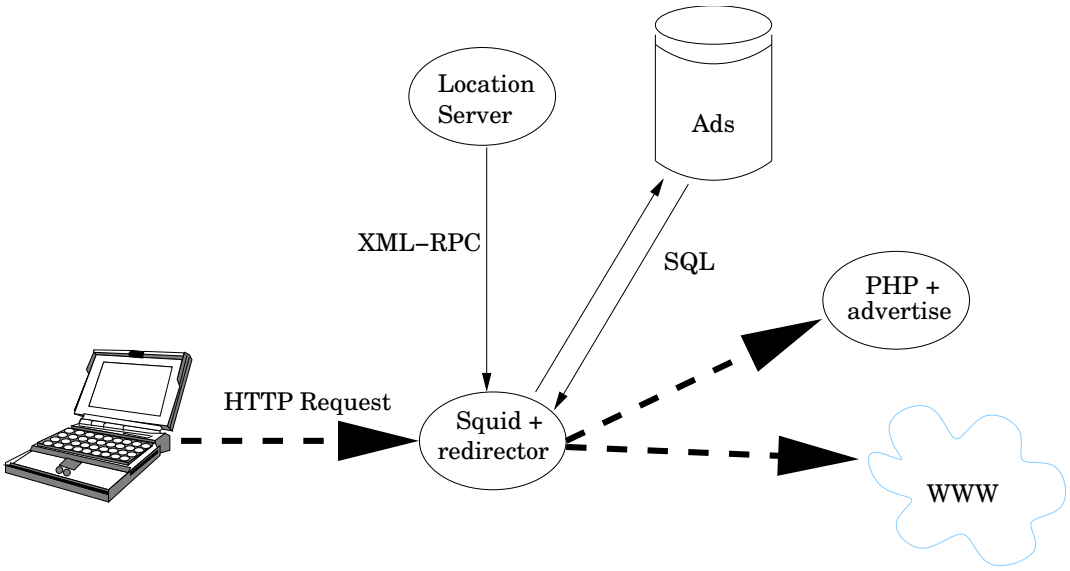

Figure 3.4: Squid redirector (demo) and related data flow

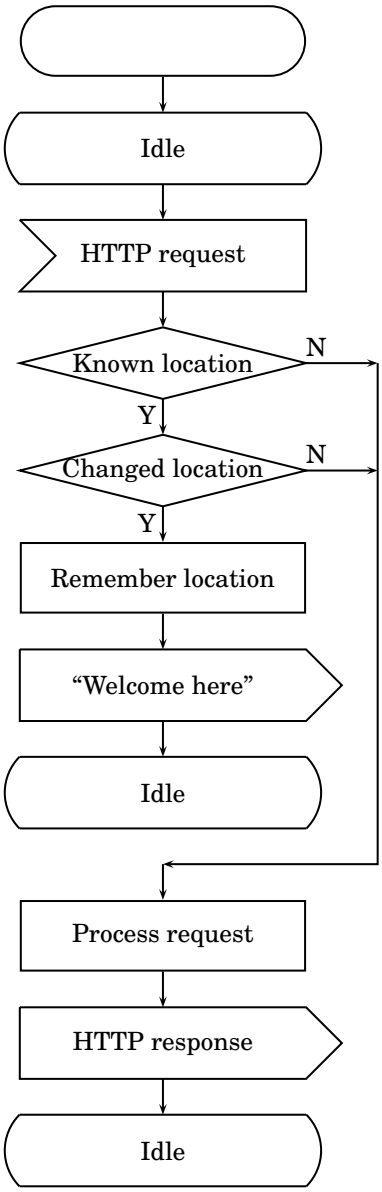

Figure 3.5: Squid redirector (demo) – SDL diagram

Chapter 4

Implementation of content delivery system

This chapter describes the implementation details of location-based content delivery system. The system consists of three main components:

1. Interception proxy.
2. Locationing service.
3. Location sensitive content.

4.1 Interception proxy

To implement any of the content delivery methods mentioned in the previous chapter, it is necessary to intercept the HTTP traffic and pass it to a proxy. Then proxy can perform some modifications in requests or responses. Interception proxy (also known as transparent cache) is quite commonly used by corporate or school networks. Interception proxying is such network configuration, where router or some other smart network device redirects all HTTP traffic to a proxy.

This approach has great benefit for system administrators because users use the cache while browsing World-Wide Web without explicitly configuring their browsers. Drawback is that browsers behave differently when communicating with proxy than when

communicating with origin servers. This may cause problems with some browsers. Also any authentication based on IP address will not work, because all requests will have the source IP address of the cache.

A rather questionable feature is, that users have almost no chance to bypass the proxy. The only way is to find some open proxy on Internet, communicating on port that is not blocked or redirected by router, and willing to forward requests from strangers. For the purposes of content delivery this is a benefit. More discussion on interception proxy issues can be found in chapter 5 of excellent book about Web Caching [3] from Duane Wessels, one of the Squid authors.

4.1.1 Router configuration

Interception proxy exists in two typical arrangements: inline and stand-alone. Inline cache configuration is shown on figure 4.1. In this configuration, router is a normal computer, e.g. some UNIX machine, and the proxy software runs directly on the router. IP stack on router must be configured so, that all HTTP traffic (TCP port 80) coming from local network is redirected to the proxy software instead of normal routing.

According to Transparent Proxy mini-HOWTO [11], on Linux system with iptables (kernel version 2.4) this configuration can be done using this command:

```
iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 80 -j REDIRECT
```

With this setting all TCP connections coming from interface wlan0 and having destination port 80 (HTTP) are redirected to localhost. Proxy must listen to requests on port 80, of course. If Web server runs on the same machine for whatever reason, it is necessary to run cache on different port (e.g. 8080). Then the command will be like this:

```
iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 80 -d ! 10.0.0.1 \
-j REDIRECT --to-port 8080
```

10.0.0.1 is the IP address of the router's wlan0 interface. This setting will process normally all requests that go to the web server running on the router, others will be redirected to proxy running on port 8080.

In-line proxy is easier to set up, but it is a single point of failure. It is better to have the router as simple as possible so that there are not many things that can fail. If stand-alone proxy (see figure 4.2) fails, smart router can detect it and pass HTTP traffic directly as if there is no cache at all.

Some routers and layer four switches can make forwarding decisions based not only on IP addresses but also on TCP or UDP port numbers. These devices can forward HTTP connections to a proxy server using two different methods:

1. If the router and cache are on the same subnet, it is enough to send the packets to cache's Ethernet (layer two) address.
2. If the router and cache are on different networks, it is necessary to encapsulate redirected packets into another IP packets and route them normally.

Cisco developed special protocol for their routers to improve interception caching capabilities – Web Cache Communication Protocol (WCCP). WCCP [12] is capable of detecting cache status and allows load balancing among more proxies and routers. Packets are forwarded to cache using Generic Routing Encapsulation protocol (GRE).

If the router is a UNIX box, it is possible to use Network Address Translation (NAT) to forward certain packets to different machine. With Linux and iptables, these commands are necessary:

```
iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 80 -d ! 10.0.0.1 \
-s ! 10.0.0.2 -j DNAT --to 10.0.0.2:8080
iptables -t nat -A POSTROUTING -o wlan0 -p tcp --dport 8080 -d 10.0.0.2 \
-s 10.0.0.0/8 -j SNAT --to 10.0.0.1
```

This configuration will send all HTTP packets from router 10.0.0.1 to proxy 10.0.0.2 listening on port 8080. There can be web server running on both the router and proxy if desired. The second line is necessary for routing the packets back from proxy to users.

This configuration has an important side effect. Using Network Address Translation effectively hides the user's IP address. As a consequence, proxy cannot do any IP based authentication or any other processing. All requests seem to be coming from the router itself. If proxy uses IP address only for access control, this is not a problem. Access can be limited by the router and it's firewalling software. Unfortunately in this project, the

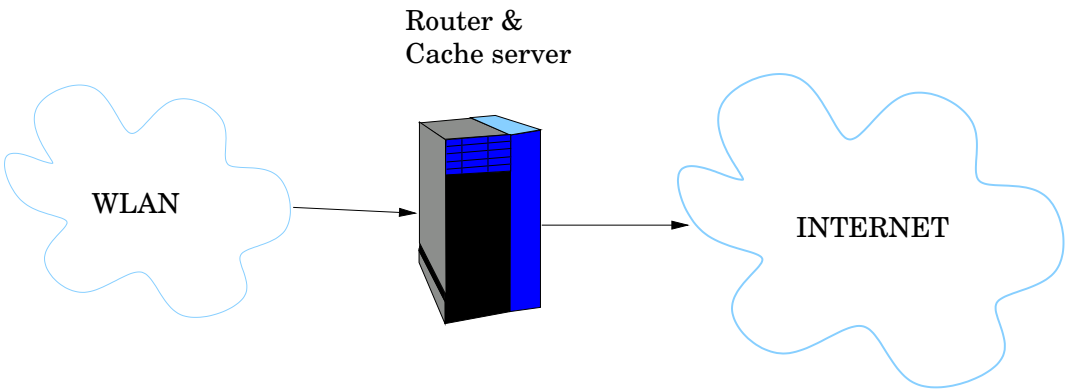

Figure 4.1: In-line interception cache

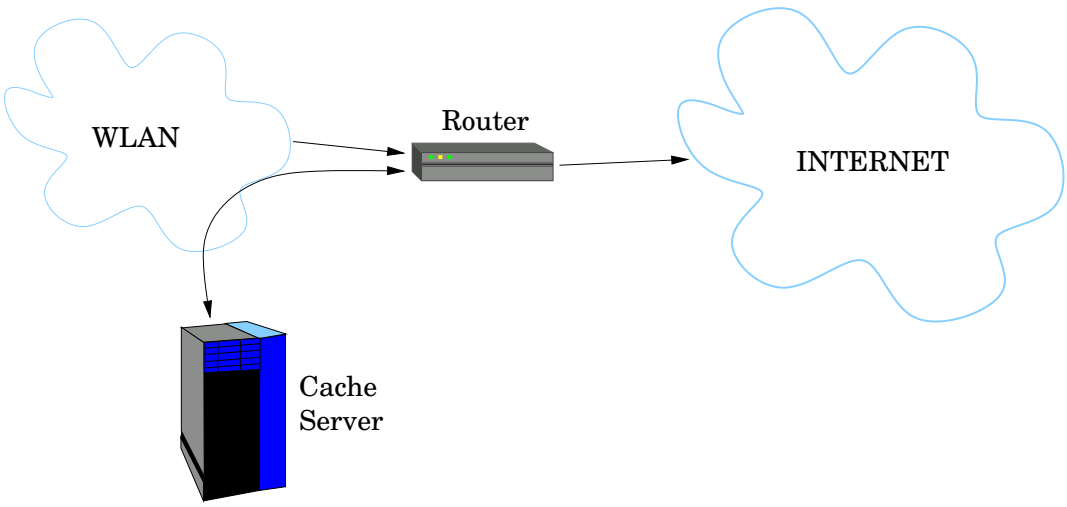

Figure 4.2: Stand-alone interception cache

IP address is used to find user's location that determines *if* something will be pushed, and *what* will be pushed.

Of course, it is possible to route packets from router to proxy preserving the source IP address, but more advanced routing techniques are necessary. In case of interest please see Linux Advanced Routing & Traffic Control HOWTO [13]. For the purpose of development and testing the technology, in-line cache is considered sufficient.

4.1.2 Squid settings

HTTP requests sent to origin servers are different than those sent to proxies. Normal HTTP/1.1 request has this form:

```
GET /index.html HTTP/1.1
Host: www.wlpr.net
```

The same request sent to proxy has a little different form:

```
GET http://www.wlpr.net/index.html HTTP/1.1
```

The `Host` header is important when single web server acts as more virtual servers with different DNS names, and serves different content according to name that is in the `Host` header. However HTTP/1.0 does not require `Host` header to be present.

Proxy server must be aware of the fact it acts as interception proxy because it will get different format of requests. It has to assemble the whole URL from the headers. If the `Host` header is missing, it must consult the lower layer information to find the original destination.

Squid [9] can be configured for interception proxying by adding these lines to *squid.conf*:

```
httpd_accel_host virtual
httpd_accel_port 80
httpd_accel_with_proxy on
httpd_accel_uses_host_header on
ie_refresh on
```

This configuration allows Squid to act as both normal and interception proxy. The last line enables a workaround for Microsoft Internet Explorer version 5.5 and older, which had problems with transparent proxies. Without this option it is impossible to reload the page. However enabling this option reduces the hit ratio of the cache.

4.1.3 Avoiding interception

As mentioned previously, interception proxies may cause strange problems. First of all, browsers send different requests if set up to talk to proxy. Also in case of stand-alone proxy there is one more hop of the HTTP packets introducing some additional delay. Therefore it is better to set up browsers to use the proxy explicitly.

In small corporate networks it is possible to set up all browsers by hand, or to tell users to do that themselves. For strong enforcement of proxy usage, port 80 can be blocked by firewall. However this is not the way for public network.

Registered users can be advised to set up the proxy by themselves, but this must not be a requirement. Interception proxy must be in use. Web Proxy Auto Discovery (WPAD) [14] is a great benefit in this scheme. This protocol is implemented in Microsoft Internet Explorer version 5.5 and later (maybe in little older versions as well). This protocol uses DHCP [15, 16] and DNS to discover file with proxy configuration. DHCP method issues DHCPINFORM message with option code 252. The response should contain the URL of proxy configuration script. DHCP server (ISC DHCPd v 3.0) can be configured for WPAD by adding these lines into *dhcpd.conf*:

```
option wpad-curl code 252 = text;
option wpad-curl "http://www.wlpr.net/wlan.pac\000";
```

The final null character is a workaround for a bug in Internet Explorer, that was adding some garbage at the end of URL. If DHCP is not used in some network, DNS allows WPAD as well. It is enough to set up another domain name alias for the web server: *wpad*. Browser then tries to download proxy configuration from URL: `http://wpad.wlpr.net/wpad.dat`.

The downloaded file has to be Proxy Auto Configuration (PAC) [17] script defined by Netscape. This file must implement JavaScript [5] function `FindProxyForURL`. This function returns string with address and port of one or more proxies to use. If more

proxies are returned, browser will try each of them and use the first one that is functional. Proxy names are separated by semicolons. Return string has the following syntax:

PROXY host:port Use WWW proxy *host* at *port*

SOCKS host:port Use SOCKS proxy *host* at *port*

DIRECT Do not use any proxy

Example PAC script for use in a test network is this:

```
function FindProxyForURL(url, host)
{
  if (isPlainHostName(host))             return "DIRECT";
  if (dnsDomainIs(host, "wlpr.net"))     return "DIRECT";
  if (url.substring(0, 8) == "https://") return "DIRECT";
  return "PROXY cache1.wlpr.net:8080; PROXY cache2.wlpr.net:8080; DIRECT";
}
```

This script allows direct connection to local servers (in domain *wlpr.net*) and to servers without explicit domain (like *localhost*). Also encrypted connections (HTTP over SSL) are allowed to go direct as there is no reason to pass them through proxy. All other requests (HTTP, FTP and Gopher) are sent to `cache1.wlpr.net:8080`. If the cache1 is broken, cache2 is used and if that fails too, direct connections are established.

Many current web browsers support PAC scripts and it is quite easy to configure them to use it. Users only enter URL of the script in appropriate dialog box. Users should be informed about this option so they can configure their browsers and improve the web performance.

PAC scripts have also one big advantage for system administrators. If cache configuration is changed, the only thing is to change the script and no user intervention is necessary. As PAC is JavaScript code, it can implement also some nice load balancing methods. One example is Super Proxy Script [18] that uses URL hash to determine which proxy to use.

4.2 Detection of user location

The demo presented in section 3.4 used locationing system written by Antti Seppänen [10]. This system used XML-RPC and proved to be too slow for redirector needs. Resolving user's location for every request caused significant delays and unnecessary network traffic. Thus new system was developed.

Previous system was based on active searching for user's location. The new system passively collects location information and writes it into database. Redirector then uses simple SQL query to find user's location almost immediately.

Locationing information is extracted from RADIUS [19] server logs. All access points in project network are set up to authenticate users via RADIUS. Access points send MAC address instead of username and the same password for every request. RADIUS server logs all login attempts to a file. This file is checked for new lines continuously and successful logins are written into database. The database consists of user's MAC address, access point's IP address and a timestamp.

However proxy identifies users by IP address, not MAC address. Thus another database is created in similar way. This time DHCP [15] server logs are being watched and written into database. This allows proxy to find IP and MAC address pairs of every user. Users that do not use DHCP (and violate the network usage policy) can be identified by proxy and notified of this fact.

Source code of the scripts that parse the RADIUS and DHCP logs, together with their startup scripts and SQL database structure are in appendix A.

4.3 Squid redirector

Squid redirectors read one line per request on standard input and write one line per request on standard output. The input line has the following format:

URL source_IP/domain_name username HTTP_method

For example:

```
http://www.lut.fi/index.html 10.1.10.1/pc1.wlpr.net - GET
```

Output is the URL that has to be downloaded or empty line if original URL shall be served. URL can be prefixed by `301:` or `302:` if HTTP redirect message is more appropriate as an answer. To use redirectors, these lines are necessary in *squid.conf*:

```
redirect_program /usr/local/bin/redir.pl
redirect_children 5
```

First line tells Squid which program to use as redirector. Second line tells how many redirector processes shall be spawned. If the proxy is busy more redirectors may be necessary.

Redirector from section 3.4 has been modified to use new locationing service described in previous section. All necessary information is now read from SQL database. Three tables are used:

leases: This table contains IP – MAC address pairs taken from DHCP server log.

location: This table contains MAC address - Access Point pairs taken from RADIUS server log.

ads: This table contains IP address - Access Point pairs defining what was the last location of user when information delivery occurred.

Redirector combines information from *leases* and *location* to find the current location. If it is different than in table *ads*, i.e. user has moved to different location, browsing is redirected to location-sensitive pages. These pages can contain any information, for example map, or advertisement of nearby shop, restaurant, etc. At last *ads* table is updated by script embedded into shown web page. As redirection is temporary, redirector uses `302:` prefix to generate HTTP Temporary Redirect response.

The algorithm and its SDL description is practically the same as in the demo version shown on figure 3.5. The difference is in the data flow (figure 4.3), where locationing server has been changed by database and log parsing scripts, and table *ads* is not updated by redirector, but web server with location-sensitive pages.

4.4 Location based Web pages

Location sensitive information pages or advertisements have to search user's location in the database created by the locationing service. This information is then stored into

database table to announce that user has seen this page recently and should not be redirected again. For the purposes of test network, the info pages are stored one per access point and all of them include this piece of PHP [20] code.

```
<?php
    mysql_connect("10.1.3.18", "php", "****");
    mysql_select_db("WLPR");
    mysql_query("REPLACE INTO ads (IP, AP) SELECT IP, AP FROM leases \
                        NATURAL JOIN location WHERE IP=\"$REMOTE_ADDR\"");
    mysql_close();
    echo "<p>Continue your browsing <a href=\"$url\">here</a></p>"
?>
```

The SQL table *ads* has the following structure.

```
CREATE TABLE ads (
     IP CHAR(15) NOT NULL,
     AP CHAR(15) NOT NULL,
     shown TIMESTAMP,
     PRIMARY KEY (IP)
);
```

Timestamp value can be used in future to trigger time based redirects, if the user does not move from one location to another.

The location is implicitly given in the file name of those pages. This makes it easy to make customised pages for every access point. The *url* parameter is used to show "continuation link", i.e. the page user originally requested, and was redirected from. Example URL is http://www.wlpr.net/show.php?addfile=ads/10.1.1.8.php&url=http://www.lut.fi/. The actual look of such pages is on figure 4.4. There are only links to on-line maps of Lappeenranta with indicated location of the nearest access point.

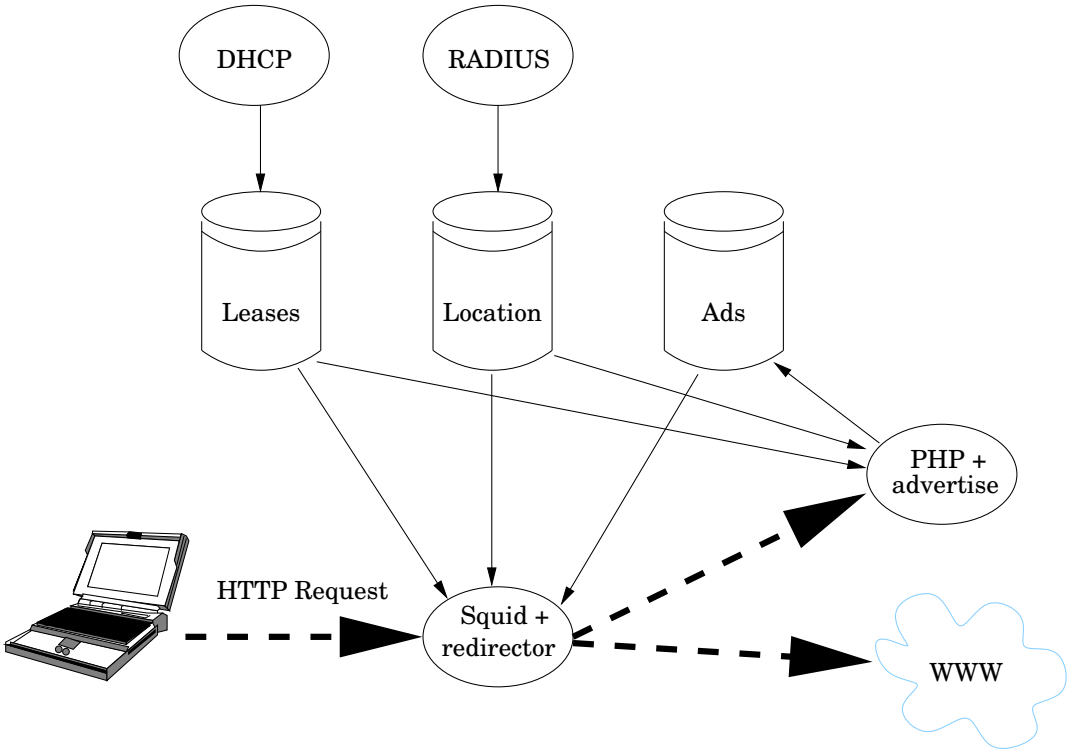

Figure 4.3: Squid redirector and related data flow

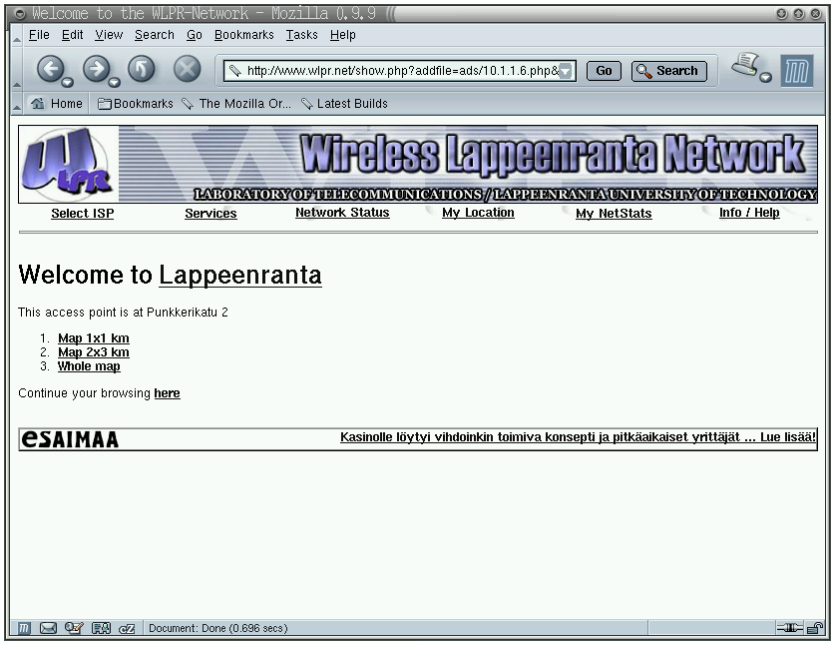

Figure 4.4: Example of location sensitive page

Chapter 5

Test network specific problems

Wireless Lappeenranta (WLPR) project network aims to provide city-wide wireless network for public use. Internet connection can be provided by multiple Internet Service Providers (ISP). User can freely choose their ISP according to their preferences or budget. Also private corporations can act as an ISP for their employees, thus extending their office network and allowing employees to work at home. This multiple ISP environment brings two main problems: user authentication and routing.

5.1 Routing

Current practice in test network is that users choose their ISP using web form. Their selection is written into DHCP [15] server configuration file, and they are issued new IP address and other configuration items (e.g. default router, DNS server). There is one router per ISP and client computers are configured to use their's ISP router. ISP selection is permanent, i.e. there is no need to specify ISP again, unless the user decides to change ISP and submits new selection. Network architecture with two ISPs is shown on figure 5.1.

As we already have one router per ISP, and these routers are Linux boxes, it is easy to make every router act as a proxy for its ISP. Then the router is set up as an in-line interception proxy according to section 4.1. This solution is easy to set up, but has some drawbacks:

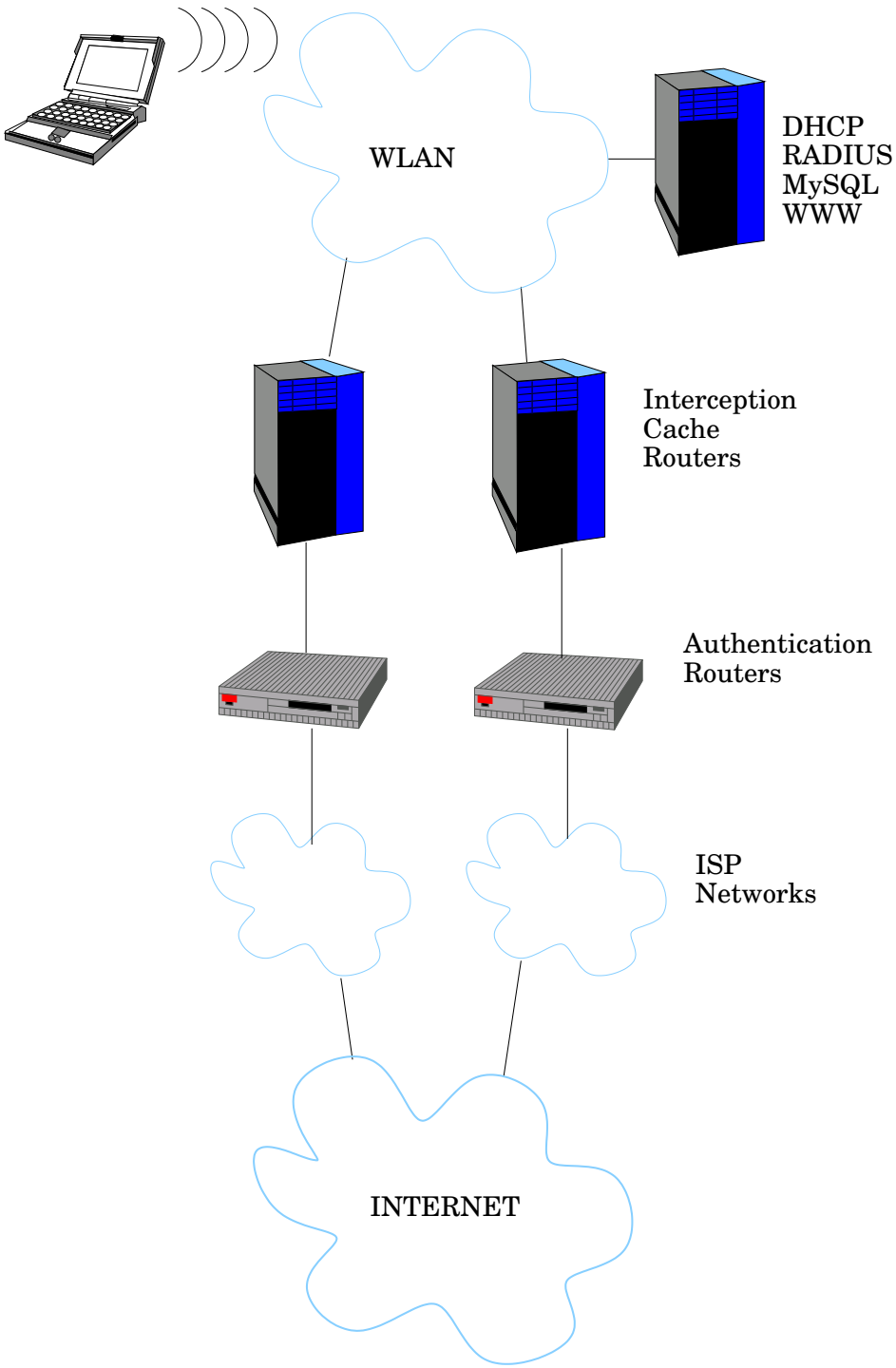

Figure 5.1: WLPR project network architecture

1. additional process running on the router is a security risk. Higher load or failure of proxy software can reduce the availability of system, and a process listening to requests can open system to potential remote network attacks.
2. several computers with a little differing configurations are more difficult to maintain, than a single proxy (or array of identical proxies).
3. web content is duplicated in every cache as downloaded by each ISP separately. However it is possible to set up proxies to share already cached content so pages are downloaded only once.

Using a single proxy common for all users will improve security and efficiency, but introduces a routing problem: It is impossible to tell the original source of the request after being processed by proxy. Moreover routing is usually done according to destination address, not source.

Web proxies can forward requests to upstream proxies, forming a hierarchy of proxies. If every ISP will have its own web proxy, it will be possible to set up such rules, that requests from users will be forwarded only to their ISP's proxy. Unfortunately we cannot rely on the fact that every ISP will have a proxy.

In the upcoming version of Squid (2.5) [9] it is possible to select outgoing address for forwarded requests according to access control lists. This allows for source IP based routing, but requires the proxy to have one network interface for every ISP.

Both of the above solutions will probably need very clever routing setup. As network routing is out of the scope of this theses, implementation using multiple in-line proxies was used.

The fact, that there is on proxy per ISP, offers us the possibility to provide "whitelists" of pages that can be viewed without authentication. Each ISP can have also different whitelist than others. Such a whitelist can be used for access to ISP's web pages that reside behind the authentication router. With a little configuration hacking it is also possible to forward requests for particular whitelist to the correct proxy from all the others. Squid redirector was extended to provide access to whitelisted web servers.

5.2 Authentication

In multiple ISP environment, using the correct router is not enough. Users have to authenticate with their ISP, to prove that they are authorised to use the connection.

(Using the local network is allowed to everyone.) ISP routers block everything by default. After successful authentication, router will allow packets from user's IP address. Single login has usually a limited period of validity, after which users have to log-in again. Different ways of authenticating users are currently under development.

Unfortunately using a proxy makes things complicated. Proxy, by definition, acts on behalf of its clients. That means that for the outside world, all requests come from the same IP address (proxy) and original clients are unknown.

If proxy is to be used it must authenticate users by itself. Squid has interface to authenticate users using HTTP basic and digest authentication [21], and to verify supplied login and password in many ways (e.g. LDAP, Samba, */etc/passwd*, etc). It is possible to write any authenticator for Squid in similar fashion to redirectors. Unfortunately proxies cannot authenticate users in interception mode. Furthermore, proxies are controlled by WLAN operator and ISPs may decide to set up their own authentication router right behind the proxy to have full control over the authentication process.

In this case it is necessary to have some communication between the authentication software and web proxy. Our solution was to create a database where authenticated IP addresses and other useful information (user name, login time, etc) are stored. Authentication software can write to this database, while proxy can only read stored data. The following steps were necessary to accomplish this task:

1. Database used by Squid redirector was extended to include this table:

   ```
   CREATE TABLE auth (
        IP CHAR(15) NOT NULL,
        user VARCHAR(20),
        login TIMESTAMP,
        logout TIMESTAMP,
        auth_soft VARCHAR(20),
        conn_sec VARCHAR(20),
        PRIMARY KEY (IP)
   );
   ```

   The last four fields are provided for future extensions.

2. Authentication software was extended to write successful authentications (IP, and username) into above defined table. Logouts are indicated by setting the *user* field to NULL or empty string or by deleting the appropriate line from database. Example PHP [20] code to be executed after successful login is provided here:

```
<?php
  mysql_connect("10.1.3.18", "auth", "****");
  mysql_select_db("WLPR");
  mysql_query("REPLACE INTO auth (IP, user, auth_soft)
               VALUES (\"$REMOTE_ADDR\", \"$login\", \"PHP\")");
  mysql_close();
?>
```

3. Squid redirector was extended to query the database if there was a valid authentication record for given IP address.

As locationing information is collected using DHCP [15] and RADIUS [19] logs, redirector has been also extended to check if the IP address was issued by DHCP server. If not, Internet access is denied and warning page is shown instead. The DHCP check was added by Radek Spáčil as part of his thesis [22] and shows a way how to enforce some of network usage rules. The algorithm of extended version of Squid is show here:

1. Read and parse the HTTP request
2. Search the source IP address in DHCP leases
3. If corresponding MAC is not found, redirect user to warning page.
4. Search user's last and current location in database
5. If the locations are different, redirect user to information (advertisement) page.
6. If the requested URL is on the whitelist, process the request unmodified.
7. Search for user's authentication
8. If user is authenticated, process his request, otherwise redirect him to login page.

Formal description of the above algorithm, using SDL, is on figure 5.2. The source code of the final version of redirector, implemented in Perl [6], is in the appendix B. Data flow diagram related to Squid redirector and related software is shown on figure 5.3. (Thin lines are SQL requests, thick dashed lines are HTTP requests, ovals are processes and "disks" are database tables.)

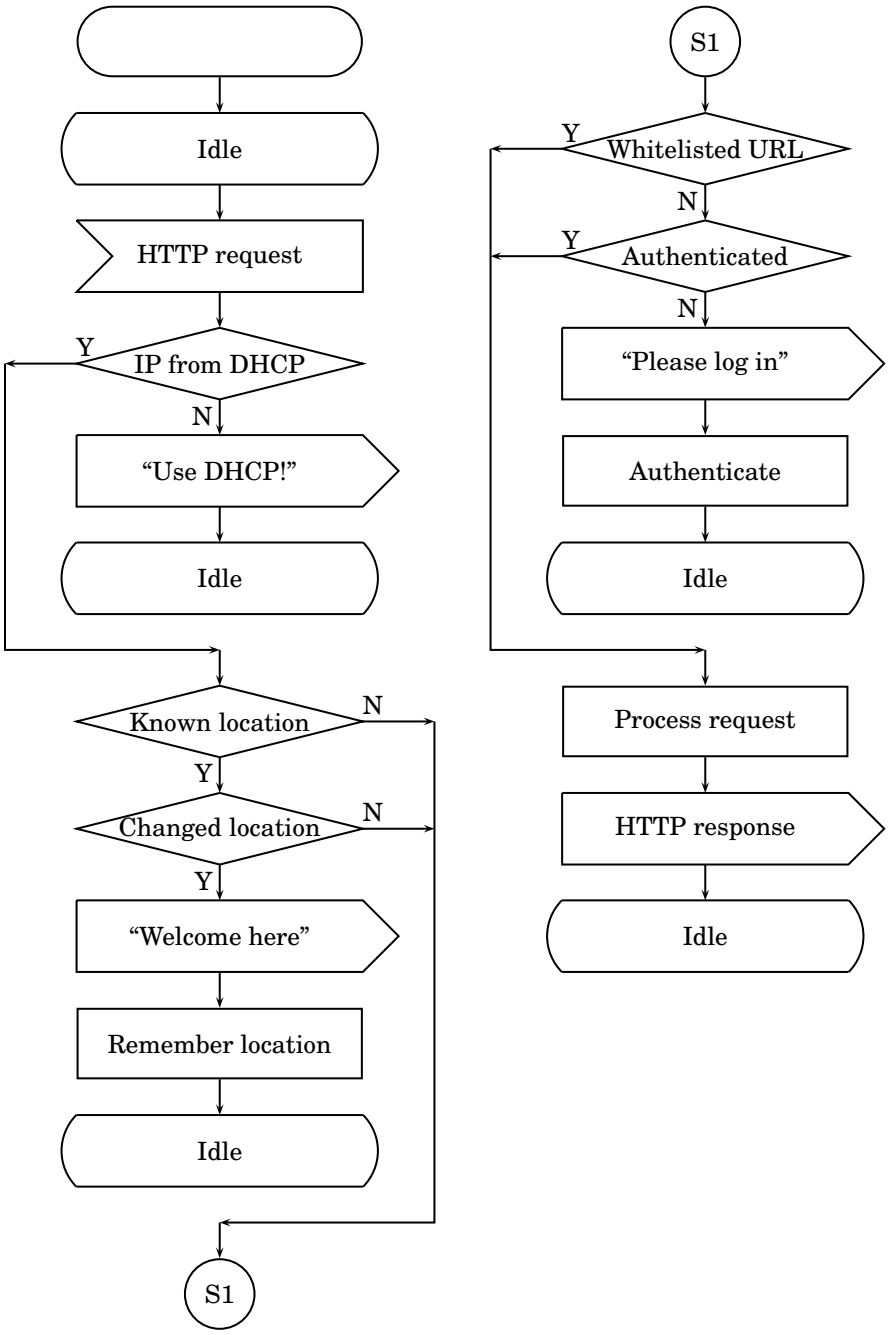

Figure 5.2: Squid redirector (final) SDL

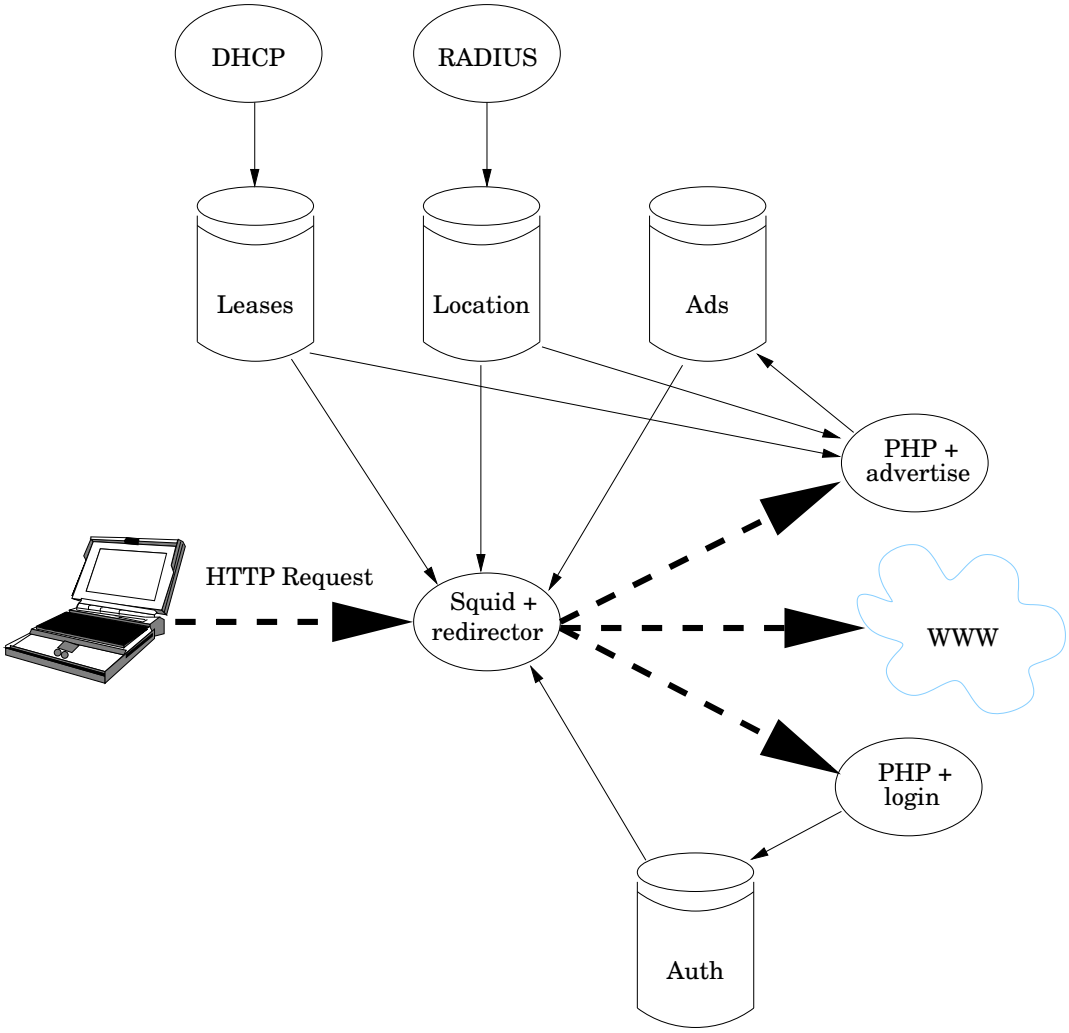

Figure 5.3: Squid redirector (final) and it's data sources

Chapter 6

Operational issues

In production environment one has to care about several issues. Not only functionality is required, but also some quality and robustness of service is necessary. In this chapter scalability of the proposed system is discussed as well as service's impact on speed of Internet (WWW) services.

6.1 Scalability and availability

This section offers solutions how to make caching service more robust and scalable. It is quite clear that in-line interception cache is a point of failure and a potential bottleneck. If the proxy goes down, users will be unable to browse web. Expanding user base can overload currently sufficient server in the future.

The solution is to create *cache cluster* – multiple caches that share the load and can absorb load from broken one. There are several ways to accomplish this goal. Very good discussion of this problem can be found in chapters 9 and 10 of Web Caching book [3]. First let's discuss possibilities of configuring cache clusters that are used by explicit configuration of clients (non-interception cache).

The easiest way is to set up DNS name like *cache.wlpr.net* to resolve to more IP addresses. DNS servers usually alter the order of addresses in response in round-robin fashion. However the most frequently accessed web pages will be downloaded and cached by all caches. This can be avoided by setting caches in cluster to ask each

other about already cached data and to fetch hits from neighbours rather than from origin servers. Furthermore, if disk space is more valuable than latency, proxies may be configured not to cache requests obtained from neighbours. There are several protocols for inter-cache communications: ICP [23, 24], cache digests [25], HTCP [26] and CARP [27]. Comprehensive comparison can be found in chapter 8 of Web Caching [3] by Duanne Wessels.

Better solution is to use deterministic load-balancing. That means, that certain document will be always requested from the same member of the cluster. Proxy autoconfiguration scripts (PAC [17]) offer this possibility. Not only that JavaScript [5] can do arbitrary processing, PAC has also built-in failover capability: If the return value of script contains more proxy addresses, browser will try them one after another until one works.

Simple load-balancing approach is to split the load according to domain names of origin servers. Much better is to compute a hash function from the requested URL and use that to determine the proxy to be used. In such cluster no duplication occurs (except for the time of failure of some members) so no inter-cache communication is necessary. Examples of such load-balancing PAC scripts are Super Proxy Script [18] and possibly also CARP [27].

Load balancing of interception proxies is much more different. Possible solutions are to use specialised smart (read as expensive) switches (so called Layer 4 or Layer 7 switches) that can make routing decisions not only on IP addresses but also according to TCP port numbers or even HTTP headers. These switches can check for "health" of caches and stop using them if they fail. The load balancing can be done either by simple round-robin or by measuring the responsiveness of caches or by destination address or URL hashing. Cisco's WCCP [12] is a special protocol for interception caching that allows better cooperation between several caches and routers.

For Linux-based (read as cheap) routers, there exists cache enforcer that understands PAC scripts [28] and thus is capable of load balancing. It is a simple program that runs as in-line interception proxy, but the only processing it does is forwarding requests to one or more caching proxies. If this cache enforcer will be used in future enhancements of presented system, it may be useful to combine Squid redirector with load balancing PAC script. Then it will run as part of cache-enforcer and not on the caching proxy, which could be then change for something else than Squid. Other interesting solution may be Linux Virtual Server [29].

In our case, load-balancing the proxies is not enough because the SQL database could

eventually become also a bottleneck. With MySQL [30], replication is possible, so this problem can be solved as well.

Our current solution using in-line caches is not scalable too much. The only improvement possible is to fetch hits from other ISP caches. For real scalability it is necessary to establish a central cache cluster that will serve users of all ISP's. Unfortunately, as mentioned in the previous chapter, routing will be much more complicated, as well as the interception.

6.2 Response delay

Web caches are used to reduce the load of expensive and limited-bandwidth Internet connections. However nothing is for free. Uncached objects take slightly longer to load because of additional processing done by the cache. As a result too small cache will not provide much bandwidth savings, and will slow down users browsing the web. This project implements special extension to proxy that is executed while processing every request. The proxy will not only look for the requested object in the cache, but will also query database server for users location and authentication. Thus it is critical that the extension is as fast as possible. For production use, the extra delays should be compensated by serving documents form cache. If the cache cannot compensate the delays, users will not use the cache, if they have the choice, or will demand the operator to stop using interception cache.

To find out if final version of delivery system, causes long delays, benchmarking test was done. The benchmarking software used was Web Polygraph [31]. This test suite is developed and used by The Measurement Factory for commercial cache benchmarks known as Cache-Offs. Web Polygraph is highly configurable and the official Cache-Off test suite configuration PolyMix-4 is freely available. This test suite is capable of simulating real web traffic, including these features:

- mixture of content types (HTML, Images, etc.)
- mixture of cache hits and misses
- mixture of cachable and uncachable responses
- origin server latency
- network packet loss

- varying request rate
- realistic content (for testing filtering proxies)
- DNS names in URL
- varying reply size
- unique URL set for every run (no need to erase cache)
- etc, etc

The main components of Web Polygraph are server and client emulators. They can be running on many machines to generate higher request rates. Client generates requests with configurable request rate, cache hit distribution, cache validation requests or forced reloads. Requests may be sent to the proxy, or directly to the server (if interception proxy is being tested). Server generates replies with real (but nonsense) content with configurable cachability ratio, and real server processing delays.

Our benchmark was done using the following hardware:

Server: 300 MHz Pentium II, 128 MB RAM, 4 GB Hard Disk, Intel 82557 [Ethernet-Pro 100] PCI network card.

Client: 200 MHz Pentium MMX, 64 MB RAM, 6 GB Hard Disk, 3Com 3c905 100BaseTX [Boomerang] PCI network card.

Proxy: HP OmniBook 500: 800 MHz Pentium III, 128 MB RAM, 20 GB Hard Disk, 3Com 3c556 Hurricane CardBus network card. (Originally we intended to use 1.3 GHz Athlon, 512 MB RAM, 30 GB HD, 3Com 3c905C-TX [Fast Etherlink], but the machine did not sustain the test load and crashed with mysterious kernel panic errors.)

Hub: WiseCom 10/100 Mbit

All machines were running Debian GNU/Linux 3.0. Client and Server were running Web Polygraph version 2.7.6 on Linux kernel 2.4.18. Proxy was running Squid 2.4.6, Bind 9.2.0, MySQL 3.23.49 and Perl 5.6.1 on Linux kernel 2.4.17. All the proxy software was available as precompiled Debian packages. Processes not related to the benchmark were just monitoring tools like *top*, *tail* for viewing the cache log, *ssh* for remote access to headless machines. Also usual Linux daemon *syslog* was running as it does on production servers.

One pair of Web Polygraph client and server is theoretically capable of 500 requests per second. Our hardware was able to do only 64 requests per second without proxy involved. Network throughput measurements, using *netperf*, between server and client showed 68.7 Mbit/s for uni-directional traffic and 35.0 Mbit/s for bi-directional traffic. Some device was operating in half duplex mode, but we did not find which. Even though, it was enough for the traffic generated by the benchmark suite.

During the first run of benchmark, it was discovered that the load is still too high so it was decreased to 32 requests per second. Web cache Squid was configured to use 2 GB of disk cache, and 8 MB of memory cache. Number of open file descriptors was increased to 4096. Other performance affecting features were left on their default values.

The test workload is described in detail on Web Polygraph pages, but the main features are described here. PolyMix-4 servers emulate real server latency by introducing random delay with normal distribution (mean 2.5 s, deviation 1 s). We were unable to emulate WAN latency and packet loss as the recommended tool (dummynet) was available only for FreeBSD system. PolyMix-4 client revisits some of previously visited URLs, offering cache hits. PolyMix-4 workload consists of 10 phases, but only the *top2* phase is considered in official Cache-Off benchmark.

framp: 20 minutes, load increases from 0 % to 100 %.

fill: load stable at 100 %, phase ends when cache is full. Fill load is configurable at 10-100 %. Hit ratio is minimal in this phase.

fexit: 20 minutes, load decreases form 100 % to 0 %.

inc1: 20 minutes, load increases from 0 % to 100 %.

top1: 4 hours, load stable at 100 %. Hit ratio is similar to phase *top2*

dec1: 20 minutes, load decreases from 100 % to 10 %.

idle: 20 minutes, load stable at 10 %.

inc2: 20 minutes, load increases from 10 % to 100 %.

top2: 4 hours, load stable at 100 %. 55 % of requests were cache hits.

dec2: 20 minutes, load decreases from 100 % to 0 %.

Graph of the client generated load (request rate), clearly showing the phases is on figure Figure 6.1. The same graph generated from server load, showing the effect of cache on decreasing outbound bandwidth is on figure Figure 6.2.

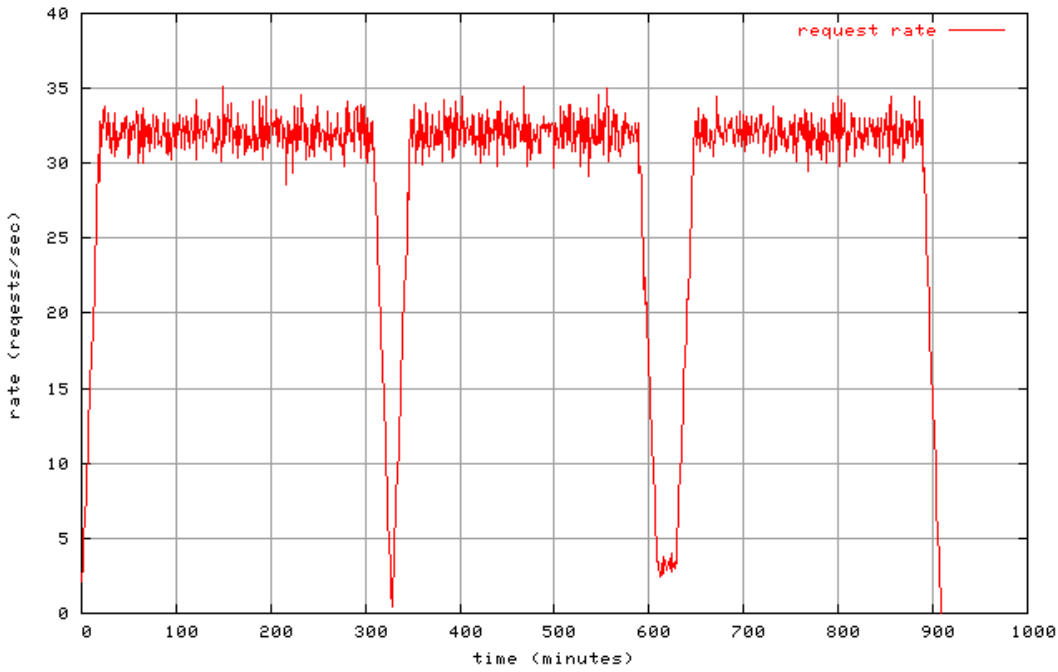

Figure 6.1: Client load trace

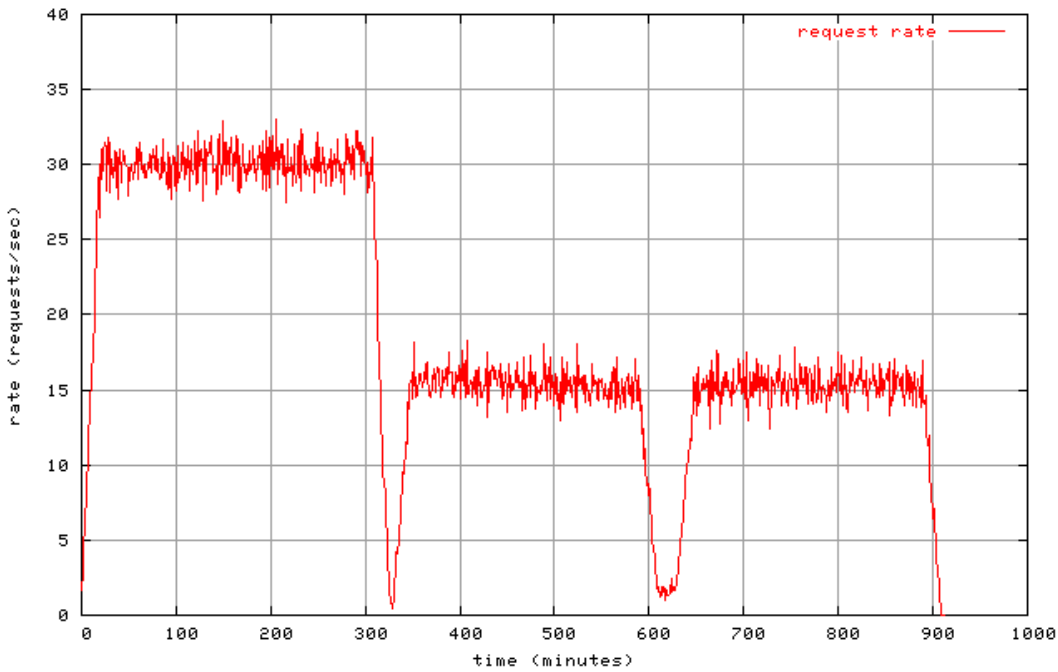

Figure 6.2: Server load trace

To find out the impact of content delivery system, the benchmark was run twice. One run with normal proxy, second with the same proxy but using redirector described in section 5.2 and appendix B performing the content delivery. Proxy used 15 redirectors to process requests in parallel. If more than 15 requests had to be served at the same time, they were queued and delayed until some redirector finished processing its request. No special content was delivered, because the client was already authenticated and in the same location during the test. That's also normal situation when users are browsing. The redirector was searching the database during every request, so the response time was, of course, longer than in the first run. Figures Figure 6.3 and Figure 6.4 show the response times during the test for normal proxy and proxy with redirector. Upper line shows response times for cache misses, lower line for cache hits and middle line is average of all requests. Response times measured during the *top2* phase are summarised in table 6.1. Improvement is calculated from mean response times as *(nocache – cache) / nocache*.

Table 6.1: Response time measurement

	normal cache	cache with redirector	no cache
hit	124.32 ms	127.40 ms	–
miss	2742.28 ms	2720.56 ms	2500.00 ms
mean	1375.14 ms	1376.69 ms	2500.00 ms
improvement	44.99 %	44.93 %	0.00 %

Results of the benchmark show that our implementation of delivery technology slows down web browsing only a little. The measured difference between normal proxy and proxy with redirector was minimal. Effect of caching itself was much more significant than effect of content delivering redirector. In general introducing content delivery system into network brings in positive side effects of web caching.

Of course, the effect of cache depends on many factors including speed of Internet connection, number of users, size of cache, hardware and software of cache. During the benchmark, the database was running on the same host as proxy so it was using some resources that could be used by cacher. On the other hand, it reduced network overheads that could cause much longer delays, if database would have its own server. Also size of database tables (proportional to number of users) is important factor. In the test the size was 1050 lines in each table.

Full test results are not published as they do not tell much more about the effect of content delivery system. However they may be valuable for web caching research and cache performance tuning.

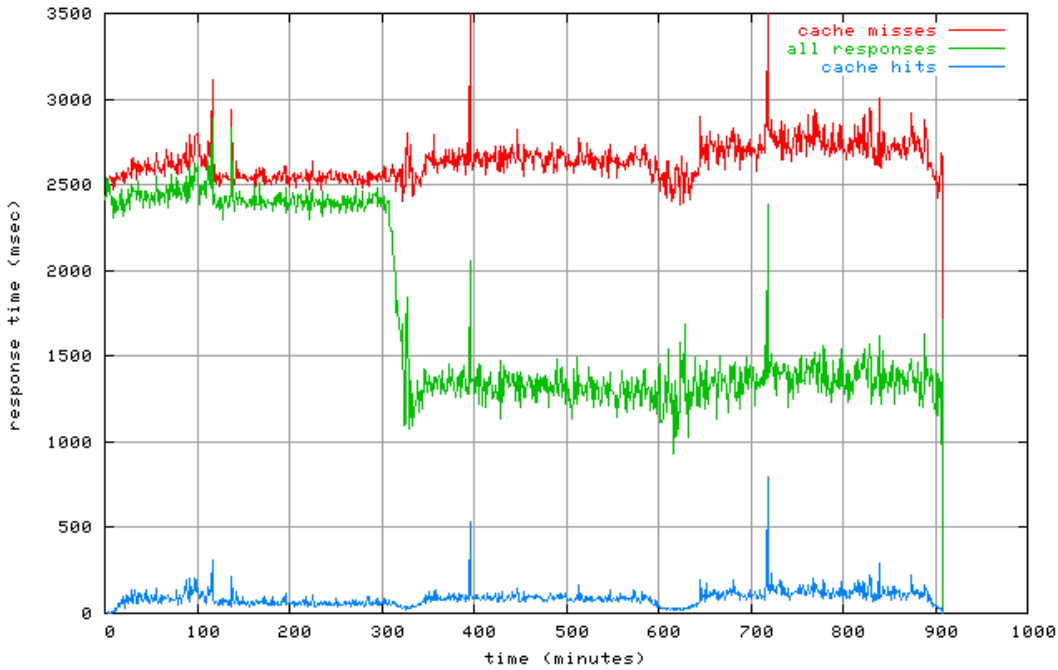

Figure 6.3: Response time trace (normal cache)

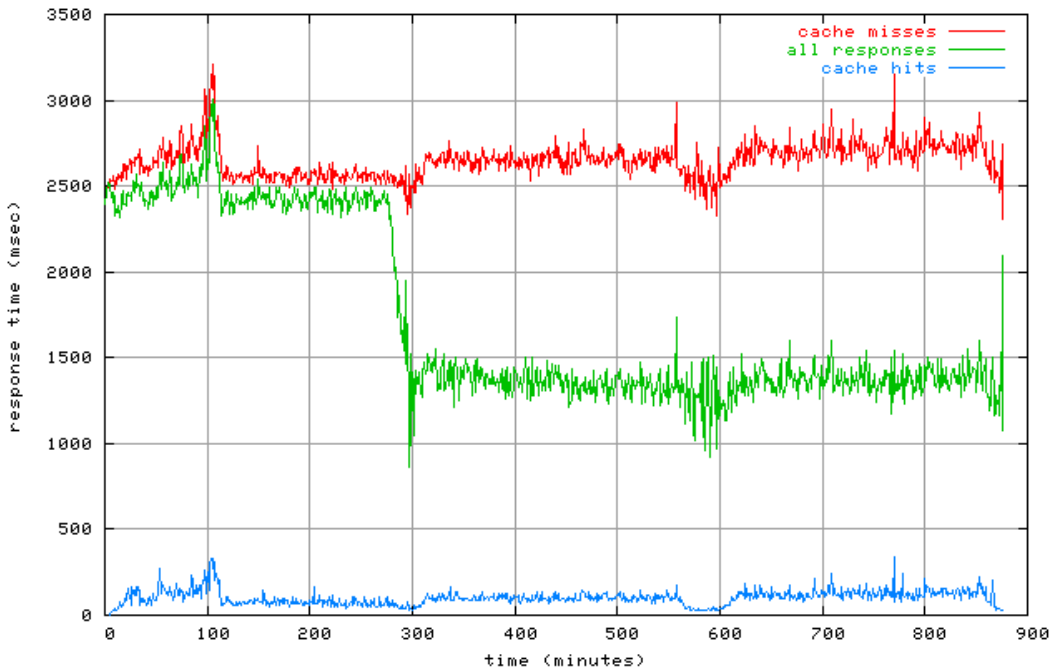

Figure 6.4: Response time trace (cache with redirector)

Chapter 7

Conclusion

Content delivery system implemented in this thesis, shows alternative way how to send more or less important information to users. Data is delivered using the most popular Internet protocol of this time – HTTP. Data is delivered to users without any prior knowledge about who they are, unlike with e-mail where knowledge of users e-mail address is necessary.

Delivery is implemented using interception web proxy cache, that limits the delivery to happen only when user is browsing World-Wide Web. Using local resources and non-HTTP traffic is not affected by this system. However it is possible to put local web servers behind a proxy as well.

Web cache is extended using redirector, that reads the requested URL and user's IP address, consults databases to find user's location, authentication and possibly other information to determine whether some special information is to be delivered. If not request is processed as normal proxy does, otherwise HTTP redirect message is generated and a needed information is shown to user. This page usually contains a hyper-link to user's original destination, so user can continue browsing the web after reading the "pushed" information.

Current implementation offers delivery of location based advertisements, user authentication, "whitelist" of web pages, and forcing simple network usage rules. This system can be extended to deliver also other content. For example periodic news, non-periodic news (network enhancements, new services, new access points), temporary information (network maintenance, outages and other problems), checking more usage rules and

blocking users violating them. Also paid subscription services may be implemented for authenticated users, like weather information, financial market quotes or payment against getting advertisements.

The systems impact on web browsing was found to be minimal. Furthermore introducing web cache into network brings the benefit of reducing outbound traffic and response times, so overall effect is positive. Of course, performance depends on the hardware and software configuration and too small or slow cache can be worse than no cache at all. For increasing demands on bandwidth and processing power, using of cache clusters was discussed in the thesis and several existing solutions were referred.

Presented software was installed in project test network and used for almost one month at the time of publication of the thesis. System was running without any problems and users did not notice any negative performance changes. It is expected that the system will be successful also in production environment.

References

[1] JIM GEIER. *Wireless LANs, Implementing Interoperable Networks.* Macmillan Technical Publishing, 1999 ISBN: 1-57870-081-7

[2] ROB FLICKENGER. *Building Wireless Community Networks.* O'Reilly, 2002 ISBN: 0-596-00204-1

[3] DUANE WESSELS. *Web Caching.* O'Reilly, 2001 ISBN: 1-56592-536-X

[4] BOB MCELRATH. *FilterProxy.* http://filterproxy.sourceforge.net/

[5] DAVID FLANAGAN. *JavaScript, The Definitive Guide.* O'Reilly, 1997 ISBN: 1-56592-234-4

[6] LARRY WALL, TOM CHRISTIANSEN & JON ORWANT. *Programming Perl.* O'Reilly, 2000 ISBN: 0-596-00027-8

[7] DAVE RAGGETT, ARNAUD LE HORS & IAN JACOBS. *HTML 4.01 Specification.* http://www.w3.org/TR/html4/

[8] ROY T. FIELDING, JAMES GETTYS, JEFFREY C. MOGUL, HENRIK FRYSTYK NIELSEN, LARRY MASINTER, PAUL J. LEACH & TIM BERNERS-LEE. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1.* IETF, 1999

[9] ADRIAN CHADD, ROBERT COLLINS, HENRIK NORDSTROM, ALEX ROUSSKOV & DUANE WESSELS. *Squid Web Proxy Cache.* http://www.squid-cache.org/

[10] ANTTI SEPPÄNEN. *Gathering and using location information in WLAN.* Diploma thesis, Lappeenranta University of Technology, 2002

[11] DANIEL KIRACOFE. *Transparent Proxy with Linux and Squid mini-HOWTO.* http://www.linux.org/docs/ldp/howto/mini/TransparentProxy.html

[12] MARTIN CIESLAK, DAVID FORSTER, GURUMUKH TIWANA & ROB WILSON. *Web Cache Communication Protocol V2.0.* Work in progress. http://www.ietf.org/internet-drafts/draft-wilson-wrec-wccp-v2-01.txt

[13] BERT HUBERT, GREG MAXWELL, REMCO VAN MOOK, MARTIJN VAN OOSTERHOUT, PAUL B. SCHROEDER & JASPER SPAANS. *Linux Advanced Routing & Traffic Control HOWTO.* http://www.lartc.org/

[14] IAN COOPER, PAUL GAUTHIER, JOSH COHEN, MARTIN DUNSMUIR & CHARLES PERKINS. *Web Proxy Auto-Discovery Protocol.* Work in progress. http://www.wrec.org/Drafts/draft-cooper-webi-wpad-00.txt

[15] RALPH DROMS. *RFC 2131: Dynamic Host Configuration Protocol.* IETF, 1997

[16] STEVE ALEXANDER & RALPH DROMS. *RFC 2132: DHCP Options and BOOTP Vendor Extensions.* IETF, 1997

[17] NETSCAPE. *Navigator Proxy Auto-Config File Format.* http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html

[18] KATSUO DOI. *Super Proxy Script.* http://naragw.sharp.co.jp/sps/

[19] CARL RIGNEY, ALLAN C. RUBENS, WILLIAM ALLEN SIMPSON & STEVE WILLENS. *RFC 2865: Remote Authentication Dial In User Service.* IETF, 2000

[20] JESUS CASTAGNETTO, HARISH RAWAT, SASCHA SCHUMANN, CHRIS SCOLLO & DEEPAK VELIATH. *Professional PHP Programming.* Wrox Press Ltd, 1999 ISBN: 1-861002-96-3

[21] JOHN FRANKS, PHILLIP M. HALLAM-BAKER, JEFFERY L. HOSTETLER, SCOTT D. LAWRENCE, PAUL J. LEACH, ARI LUOTONEN & LAWRENCE C. STEWART. *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication.* IETF, 1999

[22] RADEK SPÁČIL. *Forcing Usage Rules in Public Wireless LANs.* Diploma thesis, Lappeenranta University of Technology, 2002

[23] DUANE WESSELS & K. CLAFFY. *RFC 2186: Internet Cache Protocol (ICP), version 2.* IETF, 1997

[24] DUANE WESSELS & K. CLAFFY. *RFC 2187: Application of Internet Cache Protocol (ICP), version 2.* IETF, 1997

[25] MARTIN HAMILTON, ALEX ROUSSKOV & DUANE WESSELS. *Cache Digest specification - version 5.* Work in progress. http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt

[26] PAUL VIXIE & DUANE WESSELS. *RFC 2756: Hyper Text Caching Protocol (HTCP/0.0).* IETF, 2000

[27] VINOD VALLOPPILLIL & KEITH W. ROSS. *Cache Array Routing Protocol v1.0.* Work in progress. http://www.wrec.org/Drafts/draft-vinod-carp-v1-03.txt

[28] GLEB NATAPOV, LEONID KOILIS & ESTER HERSHY. *JS Configurable Cache Enforcer.* http://www.cs.technion.ac.il/Courses/Computer-Networks-Lab/projects/winter98_99/proxy2/

[29] WENSONG ZHANG. *Linux Virtual Server Project – Linux Server Cluster.* http://www.linuxvirtualserver.org/

[30] DAVID AXMARK, ALLAN LARSSON & MICHAEL WIDENIUS. *MySQL: The World's Most Popular Open Source Database.* http://www.mysql.org/

[31] THE MEASUREMENT FACTORY. *Web Polygraph.* http://www.web-polygraph.org/

All on-line documents were last visited on May 8, 2002.

Appendix A

Locationing scripts

A.1 RADIUS log parser *radiustaild*

RADIUS log has the following format of lines for successful login. (Other lines are not interesting for locationing purposes.)

Day_of_Week Month Day Time Year: Auth: Login OK: [*MAC*] (from nas *IP*/S0)

For example:

```
Mon Mar 18 16:40:22 2002: Auth: Login OK: [00022d-28ca20] (from nas 10.1.1.8/S0)
```

The script parses the log, searches for successful logins and rewrites them to SQL requests. Those are sent to MySQL database. Errors are logged via syslog. Script runs in infinite loop so that it will not stop in case of temporary database failure.

```
#!/bin/bash
#
# radiustaild
#
# Copyright (c) 2002 Vladislav Kurz, Wireless Lappeenranta Network project,
# Lappeenranta University of Technology
#
# This script tails RADIUSd log and writes all AP, MAC pairs into MySQL
```

```
# database. The database is used by location-based advertisement redirector.

RADIUS_LOG=/var/log/radius.log
MySQL_USER=radius
MySQL_PASS='****'
MySQL_HOST=localhost
MySQL_DB=WLPR
SYSLOG_LEVEL=daemon.err

while true; do
  tail --pid=$$ --follow=name $RADIUS_LOG | \
  awk -F '[][/ -]+' '/Login OK/ {print "REPLACE INTO location (MAC, AP)   \
    VALUES (\""substr($9,1,2)":"substr($9,3,2)":"substr($9,5,2)":"        \
    substr($10,1,2)":"substr($10,3,2)":"substr($10,5,2)"\", \""$13"\");"; \
    fflush();}' | \
  mysql --user=$MySQL_USER --password=$MySQL_PASS --host=$MySQL_HOST \
    $MySQL_DB 2>&1 | \
  logger -t $0 -p $SYSLOG_LEVEL
done
```

There is rather intricate regular expression `[][/ -]+` used by *awk* to split the input into fields. This expression means: "Fields are separated by any combination of one or more right brackets, left brackets, slashes, spaces and hyphens."

A.2 DHCP log parser *dhcptaild*

DHCP log has the following format of lines with DHCPACK messages. (Others are not important for IP to MAC address translation.)

Month Day Time Host `dhcpd: DHCPACK on` *IP* `to` *MAC* `via` *interface*

For example:

```
Mar 18 17:41:12 ns1 dhcpd: DHCPACK on 10.0.3.3 to 08:00:20:04:f5:df via eth2
```

This script does practically the same as *radiustaild*. The difference is that it parses different log and writes into different database.

```
#!/bin/bash
#
# dhcptaild
#
# Copyright (c) 2002 Vladislav Kurz, Wireless Lappeenranta Network project,
# Lappeenranta University of Technology
#
# This script tails DHCPd log and writes all ACKed IP, MAC pairs into MySQL
# database. The database is used by location-based advertisement redirector.

DHCPD_LOG=/var/log/dhcpd.log
MySQL_USER=dhcp
MySQL_PASS='****'
MySQL_HOST=localhost
MySQL_DB=WLPR
SYSLOG_LEVEL=daemon.err

while true; do
  tail --pid=$$ --follow=name $DHCPD_LOG | \
  awk '/DHCPACK on/ {print "REPLACE INTO leases (IP, MAC) \
    VALUES (\""$8"\", \""$10"\");"; fflush();}' | \
  mysql --user=$MySQL_USER --password=$MySQL_PASS --host=$MySQL_HOST \
    $MySQL_DB 2>&1 | \
  logger -t $0 -p $SYSLOG_LEVEL
done
```

A.3 Init scripts for Red Hat Linux

This script starts *radiustaild* at the same runlevel as MySQL server. For *dhcptaild* can be used the same script just with script names properly changed. This script assumes Red Hat Linux distribution and needs adjustment for others. Proper symbolic links in */etc/rc.d/* directories can be created by command *chkconfig –add radiustaild*.

```
#!/bin/bash
#
# /etc/rc.d/init.d/radiustaild
```

```
#
# Copyright (c) 2002 Vladislav Kurz, Wireless Lappeenranta Network project,
# Lappeenranta University of Technology
#
# Based on example in /usr/share/doc/initscripts-5.83/sysvinitfiles
# Copyright (c) 2000 Red Hat Software, Inc.
#
# chkconfig: 3 78 12
# description: Tails radius.log and writes successful logins to SQL database
# processname: radiustaild
# pidfile: /var/run/radiustaild.pid

# Source function library.
. /etc/rc.d/init.d/functions

NAME=radiustaild
DAEMON=/usr/local/sbin/$NAME
PIDFILE=/var/run/$NAME.pid
USER=nobody

start() {
        echo -n "Starting $NAME: "
        if status $NAME >/dev/null; then
                failure $"$NAME startup"
                echo
                return 1
        fi
        su --command $DAEMON --login $USER &
        echo $! > $PIDFILE
        success $"$NAME startup"
        echo
        return 0
}

stop() {
        echo -n "Shutting down $NAME: "
        killproc $NAME
        RETVAL=$?
```

```
        echo
        return $RETVAL
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        status $NAME
        ;;
    *)
        echo "Usage: $0 {start|stop|status}"
        exit 1
        ;;
esac
exit $?
```

A.4 Database structure

The following commands create necessary tables in MySQL database.

```
CREATE TABLE leases (
     IP CHAR(15) NOT NULL,
     MAC CHAR(17) NOT NULL,
     start TIMESTAMP,
     expiry TIMESTAMP,
     PRIMARY KEY (IP)
);

CREATE TABLE location (
     MAC CHAR(17) NOT NULL,
     AP CHAR(15) NOT NULL,
```

```
    since TIMESTAMP,
    PRIMARY KEY (MAC)
);
```

Timestamp fields are updated automatically by MySQL. In current implementation they are not used. They can be used in future for determining the time when user entered the access point range or if the DHCP lease is still valid.

Appendix B

Squid redirector

```
#!/usr/bin/perl
#
# redir.pl
#
# Copyright (c) 2002 Vladislav Kurz & Radek Spacil,
# Wireless Lappeenranta Network project, Lappeenranta University of Technology
#
# This script redirects HTTP requests to advertisement pages if the user
# location has changed, checks user's authentication, URL's presence on
# whitelist and if the user's IP address was issued by DHCP server.

use DBI;

# Not buffered I/O
$|=1;

# Connection to MySQL server
$dbh = DBI->connect("DBI:mysql:database=wlan_adv;host=www.wlpr.net",
                    "squid", "****")
    || die $DBI::errstr;
$sth_location = $dbh->prepare("SELECT AP, leases.MAC FROM leases \
                               LEFT JOIN location USING(MAC) WHERE IP=?")
    || die $dbh->errstr;
$sth_ads = $dbh->prepare("SELECT AP FROM ads WHERE IP=?")
```

```
    || die $dbh->errstr;
$sth_auth = $dbh->prepare("SELECT user FROM lut_auth WHERE IP=?")
    || die $dbh->errstr;

# Whitelist definition
$whitelist = qr/
        \blappeenranta\.fi\b|
        \betela-karjala\.fi\b|
        \bekarjala\.fi\b|
        \besaimaa\.fi\b|
        \blut\.fi\b|
        ^lut-gw\.wlpr\.net:3128$/x;
# the last one allows access to icons for proxy generated FTP directory
# listings

while (<>) {
  # Parse the redirector input
  ($url, $addr, $ident, $method) = split();
  ($ip, $fqdn) = split('/', $addr);
  ($proto, $nil, $host) = split('/', $url);

  # Query the database to find current location
  $sth_location->execute($ip) || die $sth_location->errstr;
  @lease = $sth_location->fetchrow_array();

  # check an error (null response) -> !!! no lease = doesn't use DHCP !!!
  if(!defined $lease[1]) {
    # is not in DB -> redirect to show 'use dhcp!' page
    print "302:http://www.wlpr.net/show.php?addfile=use-dhcp.html\n";
    next;
  };

  if(defined $lease[0]) {
    $ap = $lease[0];
    # Query the database to find if location changed
    $sth_ads->execute($ip) || die $sth_location->errstr;
    if( !($ad = $sth_ads->fetchrow_array()) || ($ad ne $ap) ) {
```

```
      # Location changed -> show the advertisement
      print "302:http://www.wlpr.net/show.php?addfile=ads/$ap.php&url=$url\n";
      next;
    };
  };

  # Whitelist check
  if($host =~ $whitelist) {
    print "\n";
    next;
  }

  # Query the dbase to find authentication
  $sth_auth->execute($ip) || die $sth_auth->errstr;
  if( !($logname = $sth_auth->fetchrow_array()) || !($logname) ) {

    # Not authenticated -> show login page
    print "302:http://www.wlpr.net/?url=$url\n";
    next;
  }

  # Everything's OK -> let it go
  print "\n";

}

$sth_location->finish;
$sth_auth->finish;
$sth_ads->finish;
$dbh->disconnect;
```