Introduction to computer networking

Second part of the assignment

Academic year 2013-2014

Abstract

In this assignment, students will have to implement a software that allows the network administrator to automatically filter the HTML content of pages from remote websites, based on the restricted keywords stored in the memory. In the most extreme case, the web page is replaced by another one, warning the user that he cannot access the given resource. Students will work in teams of 2 students, using the Java Language. The geomatics/geometrology sections are exempted from this part.

The software is a *gateway*, meaning that users explicitly connect to it and provide the information about the desired remote Web site to access. Unlike a regular web server, the gateway relies on external servers to provide content present in most of its replies.

Sections 1 and 2 define your assignment objectives. Sections 3 and further are an executive summary of the technologies we use.

1 Filtering application gateway

The context of this second part of the assignment is identical to the context of the first part, i.e. the design of an application gateway for web pages filtering. While the first part focused on the configuration of the gateway, the second part will emphasize on the filtering process.

The HTTP/HTTP gateway provides security at the application layer by denying access to the forbidden (content of) web pages. Every request to the gateway encodes the URL of a front page F on the target website that the user wants to have access to. The corresponding HTML content $H(F)$ is retrieved through another HTTP request.

$H(F)$ is a formatted document that contains text elements $T = \{t_1, t_2, \dots, t_n\}$ separated by HTML tags. Considering that the gateway keeps a list of restricted keywords $K = \{k_1, k_2, \dots, k_m\}$, the gateway screens T[1] and F[2] in search for elements of K, i.e. $\forall k_i \in K$, it counts the number of occurrences of k_i in $T \cup F$.

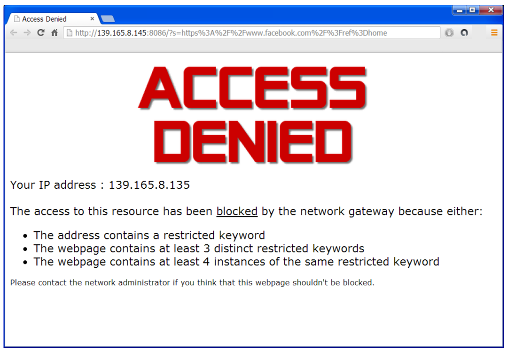
[1] The set of text elements

[2] The URL of the web page

Given that $O = \{o_1, o_2, \ldots, o_m\}$ is the list of occurrences of each k_i in T and o_F is the number of occurrences of any k_i in F, the screening process is described as follows:

- if $\forall i, 1 \leq i \leq m : o_i = 0$ and $o_F = 0$, then $H(F)$ is returned to the client without any modification,
- if $(o_F > 0) \vee (\exists j, 1 \leq j \leq m : o_j \geq 4) \vee (\exists r, s, t, 1 \leq r \leq m, 1 \leq s \leq m, 1 \leq t \leq m, r \neq s \neq t : o_r > 0 \wedge o_s > 0 \wedge o_t > 0)$ then $H(F)$ is replaced by another web page E, generated by the gateway, and returned to the client,
- For any other cases, $H(F)$ is replaced by $H'(F)$ where each occurrence of a restricted keyword $k_i \in K$ in T has been replaced by a word w_i containing only '*' characters and with $|w_i| = |k_i|$. In other words, the text content of $H(F)$ can be altered but its structure (the HTML tags) must remain intact.

Figure 1 provides two examples of filtered or replaced content.



Access Denied

http://139.165.8.145:8086/?s=https%3A%2F%2Fwww.facebook.com%2F%3Fref%3Dhome

ACCESS DENIED

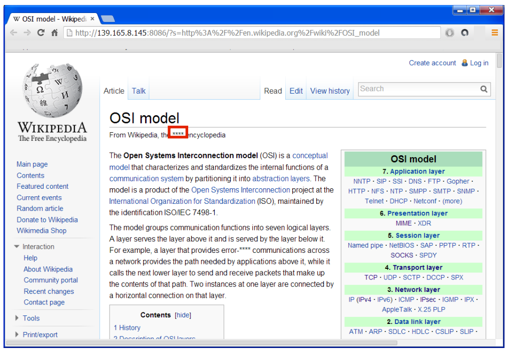
Your IP address : 139.165.8.135

The access to this resource has been blocked by the network gateway because either:

- The address contains a restricted keyword
- The webpage contains at least 3 distinct restricted keywords
- The webpage contains at least 4 instances of the same restricted keyword

Please contact the network administrator if you think that this webpage shouldn't be blocked.

(a) Blocked web page



OSI model - Wikipedi

http://139.165.8.145:8086/?s=http%3A%2F%2Fen.wikipedia.org%2Fwiki%2FOSI_model

OSI model

From Wikipedia, the **** encyclopedia

The **Open Systems Interconnection model** (OSI) is a conceptual model that characterizes and standardizes the internal functions of a communication system by partitioning it into abstraction layers. The model is a product of the Open Systems Interconnection project at the International Organization for Standardization (ISO), maintained by the identification ISO/IEC 7498-1.

The model groups communication functions into seven logical layers. A layer serves the layer above it and is served by the layer below it. For example, a layer that provides error-**** communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that make up the contents of that path. Two instances at one layer are connected by a horizontal connection on that layer.

Contents [hide]
1 History

OSI model
7. Application layer
NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · SMPP · SMTP · SNMP · Telnet · DHCP · Netconf · (more)
6. Presentation layer
MIME · XDR
5. Session layer
Named pipe · NetBIOS · SAP · PPTP · RTP · SOCKS · SPDY
4. Transport layer
TCP · UDP · SCTP · DCCP · SPX
3. Network layer
IP (IPv4 · IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk · X.25 PLP
2. Data link layer
ATM · ARP · SDLC · HDLC · CSLIP · SLIP ·

(b) Filtered web page (keyword = "free")

Figure 1: Example of displayed web pages

1.1 Launch

The software is invoked on the command line with 1 additional numerical arguments:
`java Server` *maxThreads*
where *maxThreads* is the maximum number of java Threads that can be run in a concurrent way to handle the requests. One can also see this arguments as the maximum number of requests that can be treated "simultaneously" (see section 5).

The gateway listens to port 80*xx* – where *xx* is your group number on two digits – for requests over the HTTP protocol and to port 90*xx* for configuration. See the first part of this assignment to get the information about the configuration protocol. Please note that the software will thus require at least two concurrent threads : one that handles configuration and one that handles HTTP requests.

1.2 Gateway requests

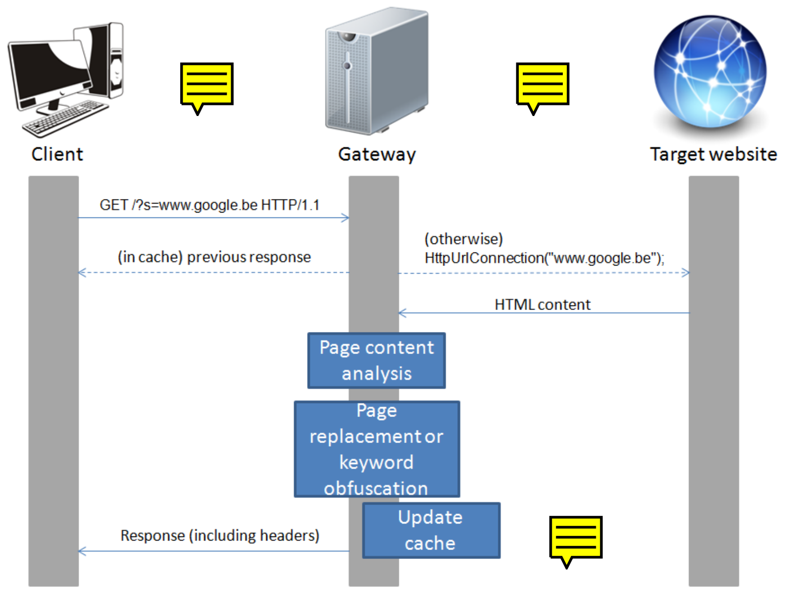

Client
Gateway
Target website
GET /?s=www.google.be HTTP/1.1
(in cache) previous response
(otherwise)
HttpUrlConnection("www.google.be");
HTML content
Page content analysis
Page replacement or keyword obfuscation
Update cache
Response (including headers)

Figure 2: Example of network exchange when contacting the Google™ website through the gateway

When a client wants to surf on a given site (e.g. `http://www.site.com/main.php`), it connects to the gateway using a request that encodes this front page URL. If the gateway is running locally on port 8086, the URL would be:
`http://localhost:8086/?s=http%3A%2F%2Fwww.site.com%2Fmain.php`,
meaning that the HTTP request actually carries the string
`/?s=http%3A%2F%2Fwww.site.com%2Fmain.php`.

The gateway will first consult its cache in search for the requested document (see section 1.6). Either the document can be found in the cache with a recent associated timestamp, in that case the gateway immediately responds with this document. Or nothing recent w.r.t the document can be found, and thus the gateway then connects to the corresponding server (on www.site.com, port 80) and retrieves the desired resource (`/main.php`). For this last connection, you are allowed to use `HttpURLConnection()` (be sure to read **carefully** the content of Section 4).

Note that the request could also contain an optional `forceRefresh` parameter, forcing the cache to be refreshed if set to true. For instance, the request
`http://localhost:8086/?s=www.site.com&forceRefresh=true` would make the gateway fetch the content of the index file of `www.site.com` even if the associated content was recently put in the cache.

When the web page is retrieved, the gateway filters or replaces the content, updates its cache, then sends the response back to the client. Figure 2 provides an example of a possible exchange between the client, the gateway and the target website.

1.3 Link redirection

The retrieved web page $H(F)$ is likely to contain links to other pages. Because we are assuming that the LAN is protected by a filtering firewall, accepting only HTTP requests from and to the gateway, it means that following these direct links will lead to a blockage, and the user would have to explicitly update the gateway request to fetch the new website content.

In order to counter this problem, you will screen $H(F)$ in search for the `HREF` parameter of the `<A>` tag. Each of these URLs must be encoded (see Section 1.5) and inserted into a gateway request that will replace the original URL in the `HREF` parameter.

In a production scheme, we would also apply the same approach to handle images correctly (`<IMG SRC=...>`), but you are allowed to leave their address unchanged. For a bonus point, you can implement the image address redirection, but bear in mind that this implies binary file transfer over HTTP, which might be tricky.

1.4 The `<script>` tag

Apart from `<A HREF=...>`, you are asked to neglect the content of a HTML tag and only consider the information (text) in between.

We remind you that HTML allows the embedding of JavaScript programs and that, obviously, these programs shouldn't be altered. Particular attention should thus be taken for the handling of `<SCRIPT>` tags.

1.5 URL encoding

So far, we considered **encoded** values for F to be inserted in the `s` parameter of a gateway request. That is, the example we provided was
`http://localhost:8086/?s=http%3A%2F%2Fwww.site.com%2Fmain.php`, and not
`http://localhost:8086/?s=http://www.site.com/main.php`

The reason is that some characters, in a URL request, have a particular meaning:

- `/` indicates a change of directory
- `?` announces the beginning of the parameters
- `&` separates the different parameters
- and so on...

If the different Fs were not encoded, then with a gateway request like `http://localhost:8086/?s=http://www.example.com/?category=dress&color=red`, we face different problems (http protocol specified twice, is `?s=http://www.example.com` a directory?, does the `color` parameter apply to the URL or to the gateway request?, ...).

To cope with this issue, we will encode every F using percent-encoding (RFC 1738, §2.2; RFC 2396, §2.4; RFC 3986, §1.2.1, 2.1, 2.5), i.e. the special characters in F will be replaced by a group of characters (e.g. / is replaced by %3A).

When the gateway receives a request, it can then identify the value of parameter `s` and decode this value (replace the percent notations by the corresponding special characters) to obtain the complete URL of the target resource.

1.6 Caching

It is expected that most of the $H(F)$ will stay the same between two requests to the gateway, for a given website. As a result, retrieving and re-parsing all the $H(F)$ is a waste of resources for the target web server and hinders performance of the gateway itself.

To alleviate from this excessive load, the gateway will use caching of the result retrieved from previous F. As long as two requests are close in time, we can generally safely assume that the content hasn't changed, since the probability that it happened is quite low.

The caching process can be circumvented by the use of the `forceRefresh` parameter of the request. In that case, the gateway will still update its cache accordingly.

2 Guidelines

- You will implement the gateway using Java 1.7, with packages `java.lang`, `java.io`, `java.net` and `java.util`,
- You will ensure that your program can be terminated at any time simply using CTRL+C, and avoid the use of ShutdownHooks
- You will not manipulate any file on the local file system.
- You will ensure your main class is named Server, located in Server.java at the root of the archive, and does not contain any `package` instruction.

Submissions that do not observe these guidelines could be ignored during evaluation.

Your commented source code will be delivered no later than 16th of May to *S.Hiard@ulg.ac.be* as a .zip package.

Your program will be completed with a .pdf report (out of the zip package) addressing the following points:

Software architecture: How have you broken down the problem to come to the solution? Name the major classes responsible of requests processing, map them onto the activity diagram of Fig. 2 and depict the dependencies between them.

Multi-thread coordination: How have you synchronized the activity of the different threads esp. to guarantee coherent access to the cache.

Limits: How does your program recover from a loss of connectivity, esp. towards remote web servers ?

Possible Improvements: This is a place where you're welcome to describe missing features or revisions of these specifications that you think would make the gateway richer or more user-friendly.

Reference Websites: mention 3 URLs (and the corresponding keyword set) on which you have successfully tested your program. If any, also mention 3 URLs for which you think the gateway could not work and provide a possible reason for that failure.

3 The HTTP Protocol

HTTP is an client/server applicative protocol that enables *resources* to be shared between machines based on their symbolic names (the *URL*s). "Resource" is a generic term that can encompass both a file or content generated dynamically from a database (e.g. using CGI or PHP).

Main methods (i.e. actions that a client can invoke on an HTTP server) retrieve content or meta-data (i.e. content type, last modification timestamp) for a specific resource. The `GET` method retrieves both content and meta-data, while `HEAD` only retrieves meta-data.

The protocol also defines additional methods to upload content (`POST`) as well as ways to `PUT` and `DELETE` resources on a server. In this work, only `GET` and `HEAD` methods will be used.

As many applicative protocols defined by IETF, HTTP is a *text-oriented* protocol, meaning that all exchanges are intended to be human-readable (as opposed to "binary" protocols such as IP, TCP and BitTorrent that feature custom information packing). Each request and reply is made of multiple lines of text, usually with only one chunk of information per line.

HTTP Request Overall Syntax

```
<http-request>  ::=  <method-name> <url> HTTP/<http-version> CRLF
                     <http-headers> CRLF
                     CRLF
```

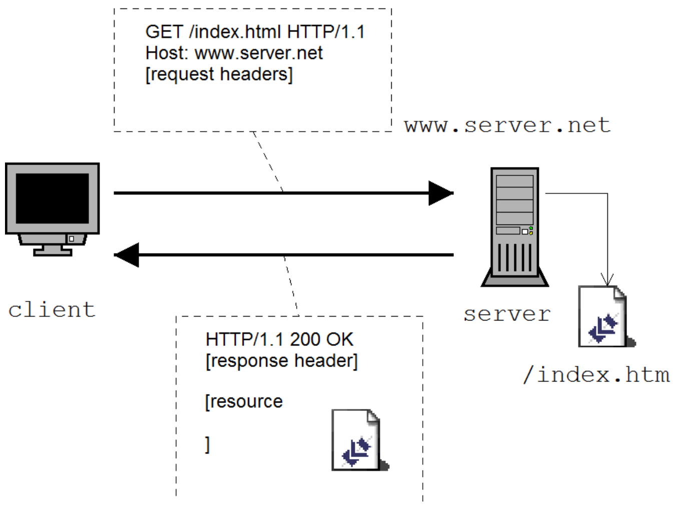


Figure 3: *A typical HTTP Request/Response cycle*

`<method-name>`	names the operation to take place
`<url>`	indicates which resource is to be manipulated. Note that it usually only provides the *path* and *query* part of the resource's location[3].
`<http-version>`	is the protocol's version. We will exclusively work with version 1.1, as described in RFC2616.
`<CRLF>`	are two control characters used as line terminators (\r\n in Java). Watch out: software exists that do not obey the standards and solely use the `LF` character as line terminator.
`<http-headers>`	Any number of optional header lines to define preferences over content type, required freshness, etc. They follow the generic format `<option-name>: <option-value>`.

Providing extra information "one line at a time" allows fairly simple extension of the protocol, as options that are not recognized by an entity (either client or server) can easily be discarded. The precise syntax and semantic of the HTTP headers are defined in section 14 of the RFC 2616 describing the HTTP/1.1 protocol[4].

[3]unless a Web proxy is involved
[4]available at http://www.ietf.org/rfc/rfc2616.txt

Overall HTTP reply syntax

```
<http-response>  ::=  HTTP/<http-version> <ret-code> <status> CRLF
                      <http-headers> CRLF
                      CRLF
                      [response-body]
```

`<ret-code>`	is a numerical value defining whether the request could be handled properly, that is intended to be used by the client software. The first digit indicates the success level (1=ongoing, 2=successful, 3=reiterate, 4=client-side error, 5=server-side error) while the last two digits further refine the type of error or to what extent the request could be fulfilled.
`<status>`	is a human-readable message matching the `ret-code` that could be displayed to the end-user.
`<http-headers>`	provide additional information on the content type, size, the encoding used to deliver it over the channel, etc.
`<response-body>`	is the actual resource's content (when found). Note that the body may be empty for some methods.

4 Useful classes

While a good programmer is able to develop any application that responds to a certain need, a good programmer is also lazy and hates to reinvent the wheel. This section will present two classes (namely `HttpUrlConnection` and `URLEncoder`) that you are allowed to use in this assignment.

4.1 HttpUrlConnection

The `java.net.HttpUrlConnection` is a class that handles HTTP requests and responses automatically. In the most simple usage, you would link a `HttpURLConnection` object with an `URL`[5] object, call the `connect()` method, get the response from the `InputStream` and the work is done.

This class is thus very handy, but its usage for this assignment is **restricted to the communications between the gateway and the target website**.
You are **NOT ALLOWED** to use this class to handle the communications between the client and the gateway.

[5]the `URL` class has a constructor that takes a String as argument.

4.2 URLEncoder

The `java.net.URLEncoder` (or `java.net.URLDecoder`) is a class that can encode (or decode) any string to (from) a percent-encoded representation. You are allowed to use these classes for this assignment. Note that the `URLEncoder` is not perfectly fitted for the job, as it would encode the space character into `+` instead of `%20` and would leave characters such as `!`, `'`, `(`, `)` and `~` unchanged. So extra processing might be required after the execution of `URLEncoder.encode(s, "UTF-8")`.

5 Thread Pool

When a server accepts a connection, it usually invokes a new *thread* that will handle that connection, such that the server can go back to listening to the port. This is very convenient to guarantee a certain level of accessibility but also has a flaw.

The *(Distributed) Denial of Service* (or(D)Dos) is an attack that targets servers with this kind of behaviour. In this attack, one (for DoS) or several (for DDoS) machines initiate many bogus connections. If the server launches a new thread for each of these connections, it will soon encounter performance problems or even crash.

To circumvent this problem, one can use a *Thread Pool* that limits the number of threads that can be executed concurrently, while keeping the other jobs on hold until new threads become available. This thread pool can be implemented through the use of `java.util.concurrent.Executors` by calling the `newFixedThreadPool(int maxThreads)` method to create a fixed-size pool of *maxThreads* threads and calling the `execute(Runnable worker)` method to assign the work represented by *worker* to one of a thread in the pool, when available.

You can (and are encouraged to) use a thread pool in your assignment.

Good work...