EE323 Assignment 2

Building an HTTP Proxy

Introduction

- **HyperText Transfer Protocol (HTTP)**
  - The protocol used for communication on the web
  - How the web browser **requests** resources from a web server
  - How the server **responds**

- For simplicity, we will be dealing only with **HTTP version 1.0**
  - RFC 1945 (http://www.ietf.org/rfc/rfc1945.txt)

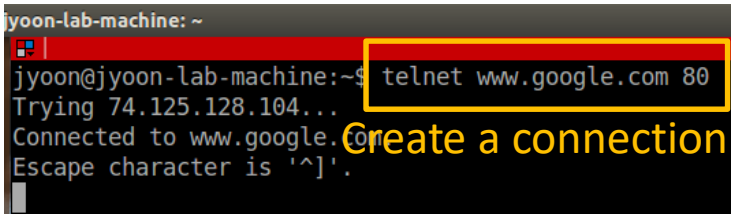
- Common basic format of request and response messages
  - An initial line
  - Zero or more header lines
  - A blank line (CRLF)
  - An optional message body

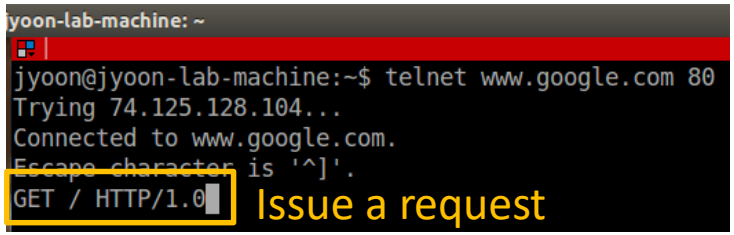
Common HTTP Transactions

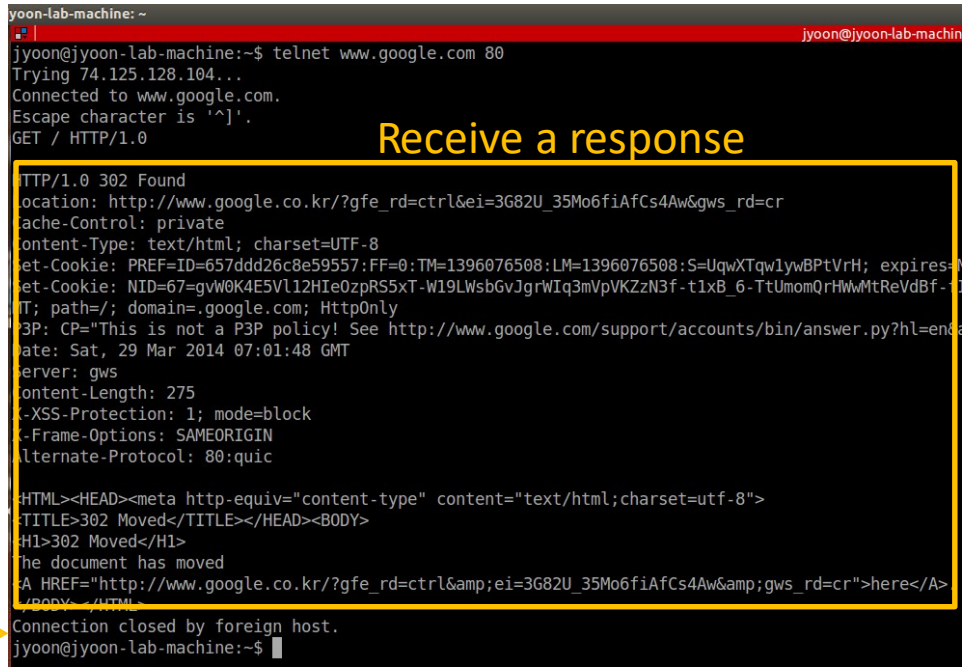
- A client **creates a connection** to the server
- The client issues a **request** by sending a line of text to the server
  - An HTTP method: GET, POST, PUT, etc.
  - A request URI: URL, etc.
  - The protocol version: HTTP/1.0, etc.
  - The message body of the initial request is typically empty
- The server sends a **response** message
  - Initial line consisting of a status line (request success/fail)
  - The protocol version: HTTP/1.0, etc.
  - A response status code: success, fail
  - A reason phrase: description of the status code
  - Data requested by the client at the message body
- **Connection is closed**

Let's Try!

- Open an UNIX prompt (Linux terminal is also okay)
- Type "telnet www.google.com 80"
- Type "GET / HTTP/1.0"
- Hit the enter twice and see what's happening

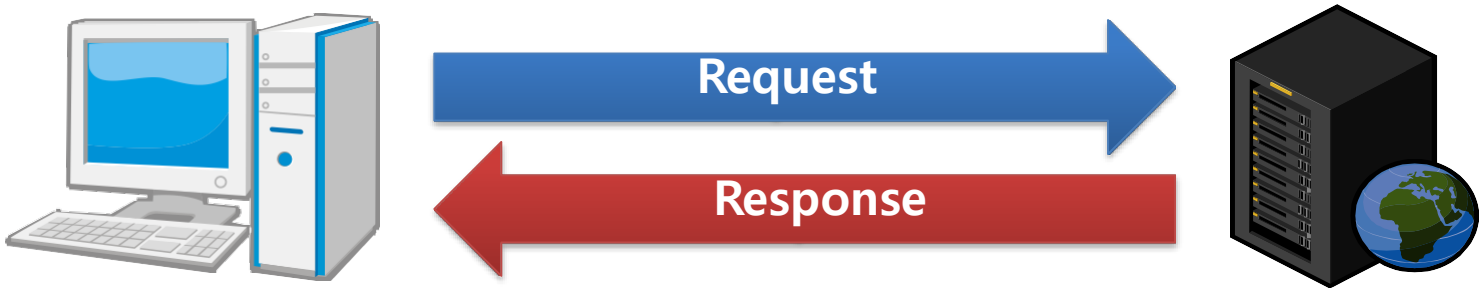




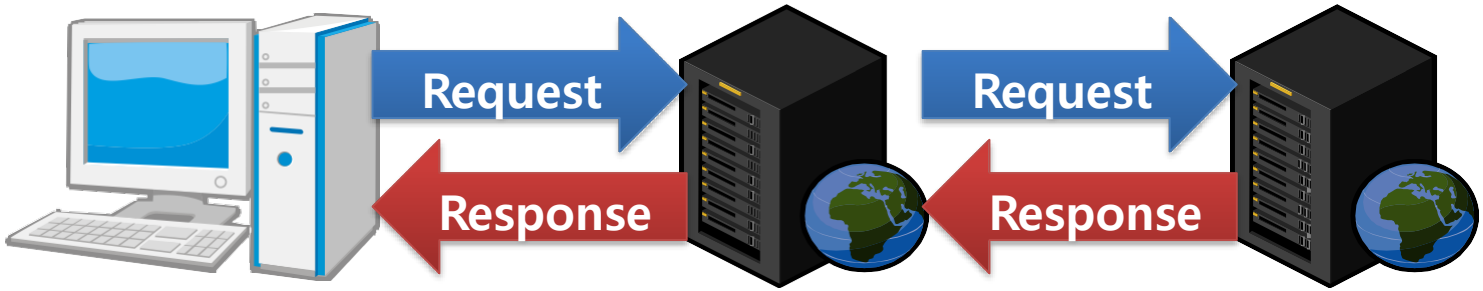


HTTP Proxies

- Communication between the client and the server



- Communication with proxy



Advantages

- **Performance**
  - Reduce the overall delay by saving a copy of the pages
- **Content filtering**
  - Inspect the requested URL and selectively block access to certain domains
- **Page Compression**
  - Make better use of available bandwidth, and provide greater transmission speeds by compressing HTTP data before it is sent
- **Transformation**
  - Reformat web pages
- **Privacy**
  - A client can hide its personally identifiable information

Basic Requirement

- Build a basic web proxy capable of
  - **Accepting HTTP requests**
  - **Making requests from remote servers**
  - **Returning data to a client**
  - **Serving multiple clients**

- Use either ANSI C or C++ to write "**proxy.c**"
  - It should be compiled and run without errors from the Haedong lounge machine
  - Producing a binary called "**proxy**" that takes as its **first argument** a **port** to listen on
    - ./proxy 12345
  - Do **NOT** use a hard-coded port number
  - Do **NOT** assume that your server will be running on a particular IP address, or that clients will be coming from a pre-determined IP

Listening

- When your proxy starts, the first thing that it will need to do is to establish a socket connection that it can use to **listen** for incoming connections
- Your proxy should listen on the port specified from the command line, and **wait** for incoming client connections
- Once a client has connected, the proxy should **read** data from the client and then check for a properly-formatted **HTTP request**
- An **invalid request** from the client should be answered with an appropriate **error code**

Parsing the URL

- Once the proxy sees a valid HTTP request, it will need to parse the requested URL

- The proxy needs at most three pieces of information
  - The requested **host**
  - The requested **port**
  - The requested **path**

Parsing the URL

- String Parsing in C
  - strtok()
  - strcmp() & strncmp()
  - strlen()
  - strchr()
  - strncpy() & strcpy() - memcpy()

- Reference:
  - http://forum.falinux.com/zbxe/index.php?mid=C_LIB&category=520875

Request and Response Transfer

- Once the proxy has parsed the URL, it can make a **connection** to the requested host using the appropriate **remote port**, or the **default of 80** if none is specified for the connection
- The proxy then sends the **HTTP request** that it received from the client to the remote server
- After the response from the remote server is received, the proxy should send the **response** message to the client via the **appropriate socket**
- Once the transaction is complete, the proxy should **close** the connection

Testing Your Proxy

- Run your client with the following command:
  - **./proxy <port>**
  - port: the port number that the proxy should listen on

- As a basic test of functionality, try requesting a page using telnet:

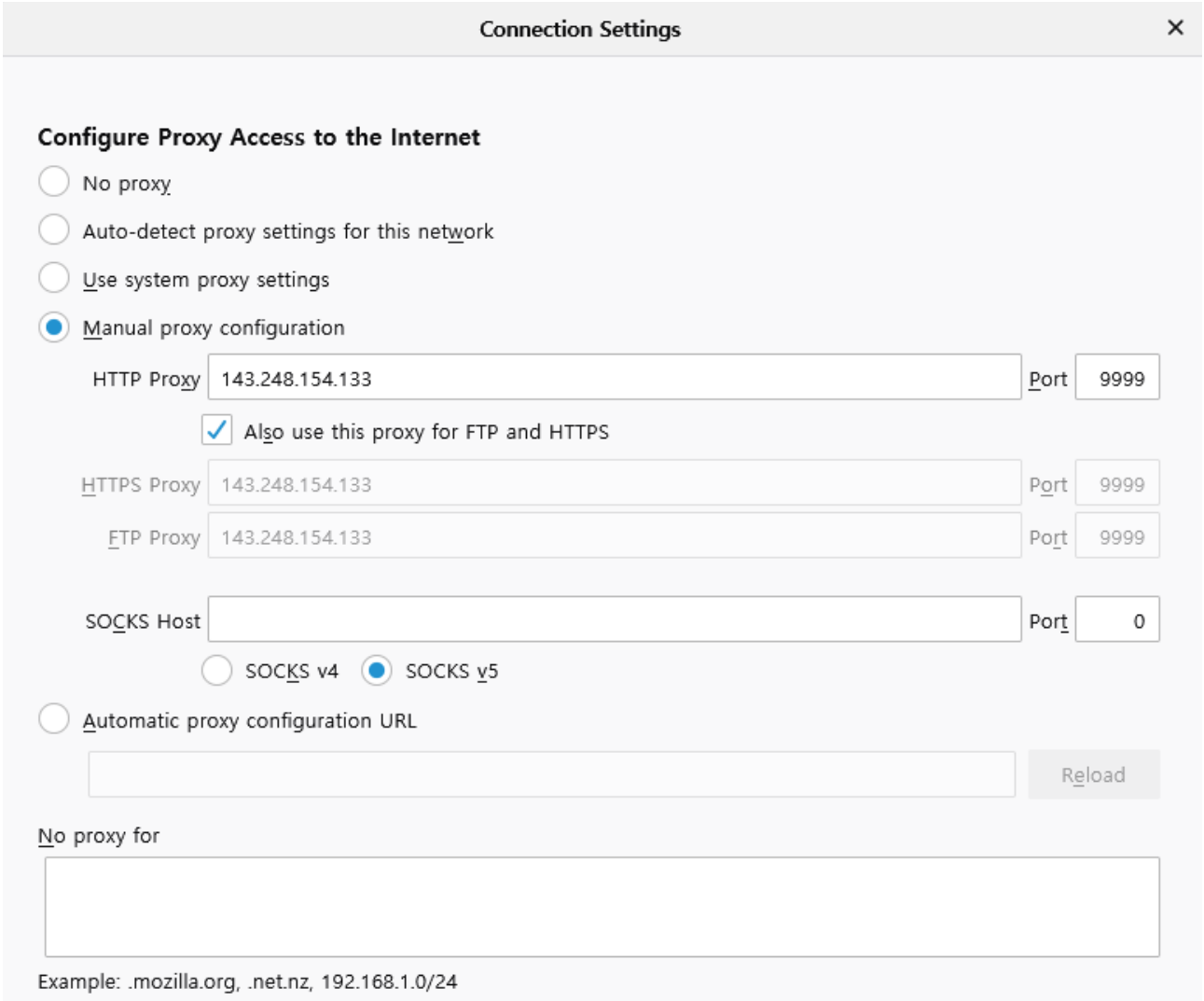
```
telnet localhost <port>
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
GET http://www.google.co.kr/ HTTP/1.0
Host: www.google.co.kr
```

- If your proxy is working correctly, the headers and HTML of the Google homepage should be displayed on your terminal screen

Test Using Web Browser

- Turn on your proxy with a port number on the Haedoung lounge machine or your local machine
- Set your web browser to use your proxy with appropriate port number that is used at your proxy
  - Please see the assignment documentation for details
- Try to access non-SSL web pages and see what's happening
- As long as your proxy works correctly for a simple HTML docu ment (for instance, the web page for this assignment) and follo ws the RFC, you can still receive all the points for this assignm ent

Example of proxy setting in Firefox



Test Using Python Script

- You can download a Python script for testing on KLMS

- **Testing sequence**
  1. Compile your proxy.c to proxy
     - $ gcc proxy.c –o proxy
  2. Download proxy_tester.py
  3. Give executable permission to the script using chmod
     - $ chmod +x proxy_tester.py
  4. Run the script
     - $ ./proxy_tester.py [PROXY_BINARY_PATH] [PORT]
     - Ex) $ ./proxy_tester.py ./proxy 45678

HTTP Host Header

- Some web servers require the "**Host**" HTTP header
  - Required in HTTP 1.1 but some 1.0 servers may complain if the request is missing the header

- Make sure to **add this header** whenever you (or client) make a request

- Your proxy should return an **error message** (400 Bad Request) w hen the request from a client does not have the "Host" header f ield

HTTP Redirection

- Your proxy also should support **redirection**

- Once the proxy get black list pages as a standard input (ex. ./proxy 9999 < blacklist.txt), it should **block** the request for those pages and **redirect** to warning site (http://warning.or.kr).

- Thus, proxy should not send requests to the pages on the black list file. It send a request to the warning site, and return the con tent of warning site to the client.

HTTP Redirection

Socket Programming

- Parsing addresses
  - inet_addr
  - gethostbyname
  - getservbyname
- Setting up a connection
  - socket
  - connect
  - getsockname
- Creating a server socket
  - bind
  - listen
  - accept
- Communicating over the connection
  - read/write
  - htons, htonl / ntohs, ntohl

Submission

- Due: 4/27 23:59 PM
- One tar file which contains “proxy.c” and “Makefile”
- Use KLMS to submit your assignments
- Your submission should be one gzipped tar file whose name is YourStudentID_assign#.tar.gz
- How to make the gzipped tar file
  - $ tar cvzf 20179999_assign2.tar.gz proxy.c Makefile

Late Policy

- Late penalty
  - 10% late penalty per day
  - Can't submit after 48 hours were elapsed from due date.

- Example
  - After 16 hours -> your grade = original point * 0.9
  - After 1day & 3hours -> your grade = original point * 0.8
  - After 2days & 1 second -> your grade = 0

- Token
  - Every student will get 3 tokens for deadline extension on this semester
  - One token can offset one day delay
  - If you submitted your assignment later than the deadline, your token automatically used for the delay

Others

- Do NOT copy and paste someone else's code including publicly available source code
- Start the assignment as quickly as possible
- **Please read assignment document carefully**
- Please ask questions via Piazza