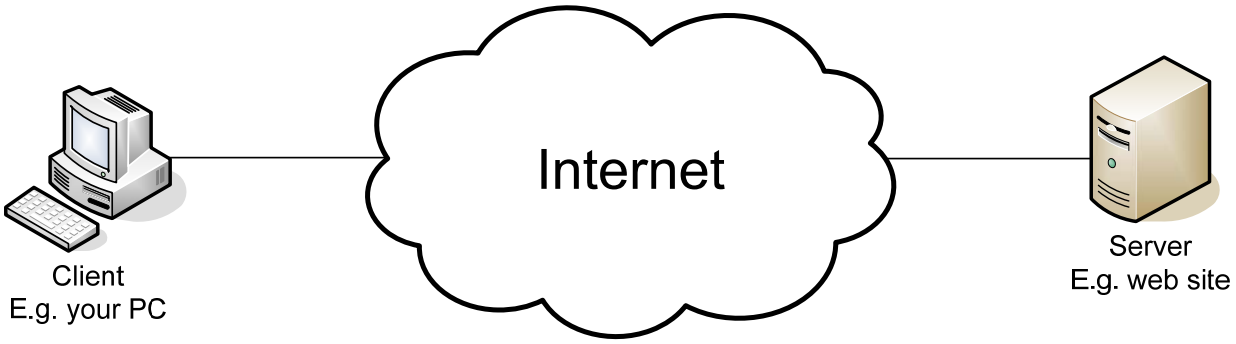
Client/Server Applications

First Name: _____ Last Name: _____

ID: _____

1 Introduction

Most network applications (e.g. file transfer, web access, email) are implemented as *Client/Server* applications.



The client/server model involves the server listening for new connections and the client initiating new connections. (A connection is usually needed each time we perform some operation, e.g. transfer a file, download a web page, send an email). We use IP addresses, as well as *ports*, to uniquely identify each connection.

1.1 Addresses and Ports

We know that IP addresses are used to identify computers on the Internet. This includes clients and servers. When sending data between a client and server, the source and destination IP addresses are carried in the IP datagram. These two addresses (source and destination) uniquely identify the connection between these two computers.

But what about different application programs (or processes) running on the computers? If you have one web browser connecting to a web server at www.google.com and a second web browser connected also to www.google.com, then how does your computer know which IP datagrams are destined for which instance of the web browser?

Client/Server applications also use *ports* to identify connections between applications. Your first web browser instance uses a different *port* number than your second web browser instance. So in fact all communications between client/server applications can be uniquely identified by both the source/destination IP addresses and the source/destination port numbers:

For example, connection 1 between browser 1 and web server www.google.com:

```
Source IP = 203.131.209.77    Destination IP = 66.249.89.99
Source Port = 47984           Destination Port = 80
```

And connection 2 between browser 2 and www.google.com:

```
Source IP = 203.131.209.77    Destination IP = 66.249.89.99
Source Port = 48032           Destination Port = 80
```

Note that the two connections between the same computers are uniquely identified, because the source ports are different.

While the source and destination IP addresses are carried in an IP datagram header, the source and destination ports are carried in the TCP (or UDP) packet header. Therefore every packet we send over the Internet has these four addresses.

1.2 Servers

The common structure of most network server applications is as follows:

1. The server is idle, listening (or waiting) for connection from clients on a *well known* port.
2. When a server receives and accepts a connection request (e.g. TCP SYN), it creates a child process to communicate with the client.
    a. The child process exchanges data with the client. When the exchange is finished, the child process is deleted.
3. The server returns to the idle state (step 1).

In this way, a server can typically handle many connections at a time. For example, the www.google.com web server can handle connections from 1000's of client hosts at a time.

An important aspect is a well known port. Since the client initiates the connection, it has to know what is the destination IP address and port number. The client can find the servers IP address through DNS (e.g. www.google.com maps to 66.249.89.99). It knows the port number because most common servers use a well known port number. These include:

Server	**Port number**
HTTP (Web)	80
FTP (File transfer)	20
SSH (Secure shell)	22
SMTP (Email)	25
Telnet (Remote login)	23
DNS	53
HTTPS (Secure web)	443

1.3 Clients

The common structure of most network client applications is as follows:

1. Send a connection request to a server. The client (in fact, the operating system) chooses an unused port number as the source port, and sends the connection request to the server.
2. Once connected with the server, the client and server exchange data.

So multiple instances (or processes) of one application can communicate at the same time – they just use different source port numbers.

2 Lab Tasks

In this class we are going to configure, run and test servers in Ubuntu Linux.

Note that when example commands are given in this set of instructions, anything that follows a # is a comment.

Also we will use the IP address of 192.168.0.1 for the web server in these examples. You must use the address of the web server in your group.

When creating new users, you should use your own ID – in this handout the example ID of 4812345678 is used (replace this with your ID number).

2.1 Groups and Initial Network Setup

You must split into the same groups as Lab 4, however the tasks must be completed in pairs. Use the same network address ranges as assigned in Lab 4.

Connect your two computers together in a peer-to-peer mode. One computer will act as the client and the other computer will act as the server.

2.2 Task 1 – Configuring Apache Web Server

The first server we are going to use is Apache Web Server. Apache is free (www.apache.org) and is the most commonly used web server in the Internet. Apache is already installed (and running) on the Ubuntu computers. In this task you will perform some simple configuration of Apache.

2.2.1 Apache Files

Apache (and many other Linux servers) are configured via one or more text files. You set the options in the files, and then restart the server, and then the server will run with those options.

The main configuration directory for Apache is:

```
/etc/apache2/
```

The main configuration file for Apache is:

```
/etc/apache2/apache2.conf
```

You can edit these file if you use `sudo` and your favourite text edit (e.g. `nano`, `vi`, `gedit`), e.g.:

```
cd /etc/apache2
# Make a backup of the apache2.conf file
sudo cp apache2.conf apache2.bkp
sudo nano apache2.conf
```

You should add the following line after the ServerRoot line:

```
ServerName localhost
```

This sets the name of your server (and prevents several warning messages when you start Apache). In Section 2.2.2 we will show how to apply this change.

In this course we do not try to explain all the details of the `apache2.conf` file. The default settings are suitable for a basic web server.

Another file specific to the web site is:

```
/etc/apache2/sites-available/default
```

This file contains configuration options specific to a site. (You can potentially host multiple sites on the one Apache server). You will need to edit this file in Task 2 to setup authentication.

The web server documents (e.g. the HTML pages that are available via the server) are stored in:

```
/var/www/
```

By default there is a directory called:

```
/var/www/apache2-default/
```

You can browse to the URL: `http://192.168.0.1/apache2-default/` to view the web page and test that your server is working.

You can create any files/directories in the `/var/www` directory which will then be accessible by the web server. (Remember you need to use `sudo` to write to the `/var/www` directory).

Finally, another important file is the log produced by Apache. Apache logs (records) all requests for content on this server. The log is a text file:

```
/var/log/apache2/access.log
```

The format of this log file is a space separated file with each line showing details of a single request for a web page on the server. Each line has the following fields:

- The IP address of the source
- - (not used)
- The user name of the user who requested the page (only present if HTTP authentication is used, otherwise is -)
- Date and time the request was made
- The GET request, showing the path/file requested
- The HTTP status code sent back to the client (e.g. 200 is OK)
- The size of the page/object sent back to the client
- The URL of the page that referred the request (e.g. the page that linked to the requested page)
- The user agent making the request, e.g. an identifier of the web browser

You should not edit the `access.log` file. Instead use `less` or `tail` to display its contents:

```
# Browse through the file, page by page:
less access.log
# View the last ten lines of the log – useful for viewing the most recent
# accesses to your web server:
tail access.log
```

2.2.2 Starting and Stopping Apache

You can use the `apache2ctl` command to start, stop and restart Apache. You must restart Apache if you want any changes in the configuration files to take effect. The commands are:

```
sudo apache2ctl restart       # stop then start Apache
sudo apache2ctl stop          # stop Apache
sudo apache2ctl start         # start Apache
```

Since you have made a change to `/etc/apache2/apache2.conf` (you added the `ServerName` line), you should now restart your web server.

2.2.3 Creating Web Pages

Basic web pages are written in HTML. In this course we do not explain the format of HTML (maybe you have covered it previously) but you should be able to create a basic web page in HTML using any text editor (e.g. `nano`, `vi`, or `gedit` in Ubuntu). With Apache, the web pages are stored in `/var/www`, and by default, if you browse to a directory (e.g. http://192.168.0.1/) then Apache will display the `index.html` file (if it exists).

You should create a simple HTML test page in `/var/www` called `index.html`. An example page is:

```
<html>
<head>
<title>Test Page for ITS 332</title>
</head>
<body>
<h1>Test Page for ITS 332</h1>
<p>
This is a test page for ITS 332.
</p>
</body>
</html>
```

You can of course create sub-directories and other files in `/var/www` if you want.

2.2.4 Using Apache

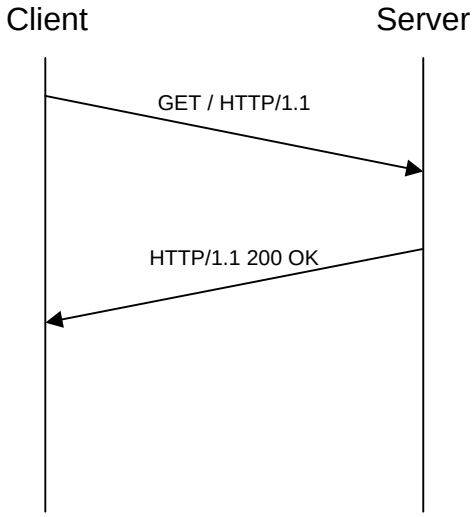
Now that you have restarted Apache and created the `index.html` file, complete the following steps:

1. On your client, start Wireshark/Ethereal and start capturing packets
2. On your client, browse to http://192.168.0.1/
3. Once the page is displayed, stop the Wireshark capture.
4. Check that your capture contains the messages similar to the time sequence diagram on the next page.
5. What are the values of the four parameters that uniquely identify the connection between client and server?

6. View and record the Apache log record that matches the exchange you captured:

Apache log record:

From the Wireshark capture, a simple time sequence diagram (showing only HTTP messages – we ignore the TCP ACKs and TCP handshaking) would look like the figure below. It shows the sequence of messages being sent. This is a useful and common method of visualising what is happening in a simple network exchange of information. It is quite simple for the web page request in this task, but may be more complex, as you will see in Tasks 2 and 3:



2.3 Task 2 – Authentication and Apache

It is common to protect some content on a web server using passwords. You should complete the following to setup a password protected directory on your website.

First you must create a username/password that Apache will use. Lets check whether a password file already exists (for some computers it will, others not):

```
# Enter the Apache directory
cd /etc/apache2
# check if the ‘passwd’ directory and ‘passwords’ file exist
ls passwd
```

There are two possible results from the above command (depending on the configuration of your computer):

Option 1 (Password file already exists): if you see the file passwords in the directory, then you can add a new user as follows:

```
# Use htpasswd to add a user based on your ID, e.g. u4812345678
sudo htpasswd /etc/apache2/passwd/passwords u4812345678
# You will be prompted for a password for user 'u4812345678'.
```

Option 2 (Password file does not yet exist): if you see a message ‘No such file or directory’, then you must create a passwords file:

```
# create a directory where the username/passwords will be saved
sudo mkdir passwd
# Use htpasswd to create (-c) a passwords file, and add a user based on ID
sudo htpasswd -c /etc/apache2/passwd/passwords u4812345678
# You will be prompted for a password for user 'u4812345678'.
```

Now you must configure Apache to use a password on a specific directory.

```
cd /etc/apache2/sites-available
# Make a backup of the default file
sudo cp default default.bkp
# Edit the default site configuration file
sudo nano default
```

The file `/etc/apache2/sites-available/default` contains several `<Directory>` directives. These directives set parameters for specific directories on the web server. You must create a new directive (for example, after the `"/usr/lib/cgi-bin"` directive) with the following text:

```
# Password protect the directory /var/www/protected
<Directory “/var/www/protected”>
      # Set authentication type to Basic (which requires a password)
      AuthType Basic
```

```
      # The name of the authentication domain
      AuthName "Restricted Access to Protected Content"
      # The file that stores the usernames/passwords (created with htpasswd)
      AuthUserFile /etc/apache2/passwd/passwords
      # The users that are allowed to access this directory
      Require user u4812345678
</Directory>
```

Now save and close the file.

The final thing you must do is create the protected directory and put some content in it for testing.

```
cd /var/www
sudo mkdir protected
cd protected
# Now you should put some example content in the index.html file
sudo nano index.html
```

Once the `/var/www/protected/index.html` file is created, you can test that your authentication works:

1. On your client (web browser), start Wireshark/Ethereal and start capturing packets.
2. On your client, browse to http://192.168.0.1/protected/ (remember to replace with the IP address of your web server) and enter the username/password. The `index.html` file you created should now be displayed.
3. Stop the Wireshark capture and draw a time sequence diagram of the HTTP exchange. (Hint: look for the HTTP messages by sorting by the Protocol column in Wireshark).

4. View the last record of your web server access log and record the information in the table:

IP address of source	
Username	
Time of access	
File/directory requested	
HTTP status code returned	
Size of page returned	
Referring page	
Browser that made request	

5. Answer the following questions:
    a. In which packet in your time sequence diagram does the web server request the browser for authentication? What does the WWW-Authenticate field refer to?

    b. In which packet in your time sequence diagrams does the web browser send the username and password? What do the Authorization and Credential fields refer to?

c. Assume your network consisted of a client connected to router, and the router connected to server (e.g. 3 computers instead of just 2 computers). If Wireshark/Ethereal was run on the router during the above authentication test, could the router see the password?

d. Explain or comment on the security of Apache authentication (considering your answer to part (c)). What are the problems of using it for protecting content on your web server?

2.4 Task 3 – OpenSSH Server

Secure shell (ssh) is a protocol for securely logging in to another computer. It is a replacement for telnet (which was insecure). OpenSSH is a free implementation of a SSH client and server. Both client and server should be installed on the Ubuntu computers (talk to me if the server is not installed).

Secure shell can be run from the command line using:

```
ssh 192.168.0.1
```

Optionally, you can include the username to log in as (otherwise it will default to the current username in use on the client):

```
ssh 192.168.0.1 -l student
```

You will be prompted for the password of that user on the server. (The first time you log in you may also be prompted about unknown authentication – enter Yes to continue).

Once you have logged in, you can run commands on the server. That is, it is the same as if you are using the command line on the server.

You can log out using the `exit` command.

Use the above steps to log in and out of the server, and then complete the following:

1. On your client computer, start Wireshark and start capturing packets.
2. On your client computer, log in to the server using `ssh`
3. Once logged in, execute a `ls` command to view the directory contents.
4. Now exit
5. Stop the Wireshark capture.
6. Draw a time sequence diagram of the SSH exchange. (Hint: only consider the packets that have Protocol column as SSH or SSHv2).

7. Answer the following questions:

   a) From the Wireshark capture, can you find or view the username and password used for the log in?

   b) If you ran Wireshark on a router in between the client and server, could you obtain the password? Explain your answer.

   c) If you ran Wireshark on a router in between the client and server, could you see the commands that the client user executes on the server?

   d) Bonus Question: Explain the steps that SSH goes through before data is exchanged between client and server.

e) Bonus Question: What algorithms are used to create the secure connection between client and server?

2.5 Cleaning Up

At the end of the Lab class you must reset your computer back to the default settings, which include:

- Replace `/etc/apache2/apache2.conf` with the original `/etc/apache2/apache2.bkp`
- Replace `/etc/apache2/sites-available/default` with the original `/etc/apache2/sites-available/default.bkp`
- Delete any files you created in `/var/www` (note the `apache2-default` directory must not be deleted)
- Delete the file `/etc/apache2/passwd/passwords` (you must use `sudo` to delete).
- Routing is off:

```
sudo su
echo “0” > /proc/sys/net/ipv4/ip_forward
exit
```

- Ethernet cable is connected to a one network interface, which is configured to use DHCP. You should be able to browse the Internet.
- The second network interface is disabled (turned off).
- All crossover cables are removed and returned to the TAs.
- Delete the routes that you manually added to routing tables.