COMP90007 Internet Technologies: Lab 2

1 Objectives

- To learn about using the telnet to establish transport layer connection-oriented communication with a remote host.
- To examine TCP (Transmission Control Protocol) in detail. TCP is the main transport layer protocol used in the Internet.
- To examine HTTP (HyperText Transfer Protocol) in detail and how requests are made in a web browser and how responses are returned.

2 Requirements

There are two software requirements for this lab:

- **Wireshark**

  Wireshark is a sniffing tool that allows the user to examine a packet trace, that is, a record of network traffic over time. It can be downloaded at `http://www.wireshark.org/download.html`.

- **PuTTY or an equivalent telnet client**

  PuTTY is Windows telnet and SSH client which is commonly used to connect and login to remote machines. We will be using PuTTY as a telnet client in this lab. PuTTY can be downloaded from `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.

  If you are running OS X or Linux, a telnet client should be pre-installed on your operating systems. It can be run by typing `telnet` on your command line terminal.

The software resources mentioned above should be pre-installed on the laboratory computers. If not, however, you may download them at the URLs provided above. Also, please feel free to try and install and run the software on your own personal machines at home.

3 Lab tasks

Telnet is an application layer protocol running on TCP. This means it is a connection-oriented protocol. It is used to provide text-based communication between two hosts. It was one of the first standardised protocols used on the internet. In the past, telnet was used for logging into remote computers, but because of the lack of security, other protocols such as SSH are widely used instead.

In this lab, we will investigate the use of HTTP and using telnet to request web pages, similar to what your favourite web browser does. HTTP is an application layer protocol used to deliver web pages across the web. We will be crafting requests to a remote web server according to the HTTP specification RFC 2616 and observe what is sent back in response.

Usually, crafting the requests for web pages is all done by your web browser. Leveraging telnet to request web pages is usually only done when testing if a web server is working correctly or not. For most intents and purposes, this is considered a "hack" and is not the proper way to send requests over HTTP.

Step 1

Open PuTTY. There are several options you need to set to get PuTTY working as a proper telnet client. These are detailed as follows:

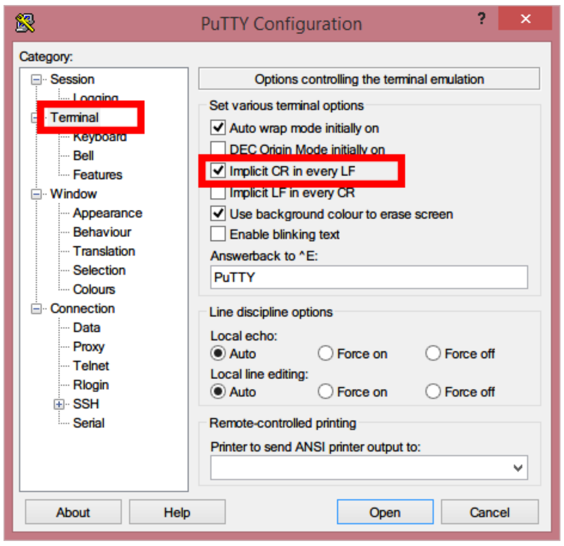
- Under *Connection type*, select *Telnet*.
- Under *Close window on exit*, select *Never*. This will make sure you can read the output after the web server closes the connection.
- Select the *Terminal* category, with reference to Figure 1.
- Check *Implicit CR in every LF*. This will ensure proper line breaks are displayed, stemming from differences with Unix (`\n`) and Windows (`\r\n`) line break conventions.
- Select the *Telnet* sub-category, which is under the *Connection* category, with reference to Figure 2.
- Under *Telnet negotiation mode*, select *Passive*. The HTTP requests don't seem to work if you don't set this option.

If you are not running on Windows, then you don't need do to all of the above and you can get started by typing `telnet` in a command-line prompt.

Step 2

- Prepare a Wireshark capture with the capture filter '`tcp port http`', or alternatively the display filter '`tcp.port eq 80`'. Try to minimise noise in your capture by closing any unnecessary network applications.
- In PuTTY, enter `cis.unimelb.edu.au` as the host name and `80` as the port number. HTTP applications are traditionally run over port 80. We are going to connect to the CIS department's web server.

  If you are not using PuTTY, enter '`o cis.unimelb.edu.au 80`' at your '`telnet> `' prompt.
- Before hitting *Enter* and starting the connection, start your Wireshark capture first!



Options controlling the terminal emulation

Set various terminal options

- [✓] Auto wrap mode initially on
- [ ] DEC Origin Mode initially on
- [✓] Implicit CR in every LF
- [ ] Implicit LF in every CR
- [✓] Use background colour to erase screen
- [ ] Enable blinking text

Answerback to ^E: PuTTY

Line discipline options

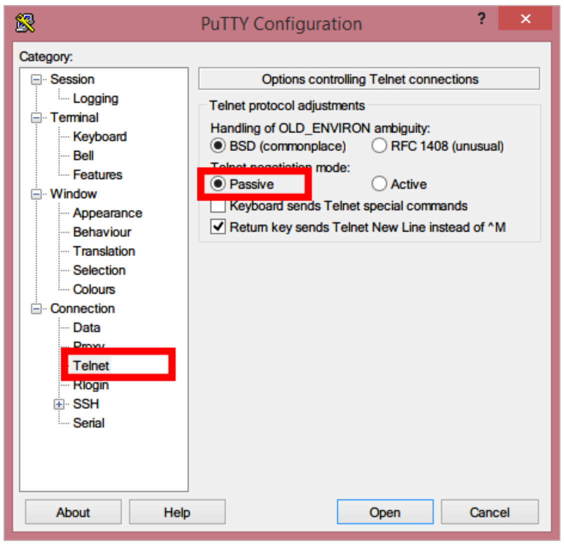
Local echo: [✓ Auto] Force on Force off

Local line editing: [✓ Auto] Force on Force off

Remote-controlled printing

Printer to send ANSI printer output to:

Figure 1: Setting *Terminal* options in PuTTY



Options controlling Telnet connections

Telnet protocol adjustments

Handling of OLD_ENVIRON ambiguity: [✓ BSD (commonplace)] RFC 1408 (unusual)

Telnet negotiation mode: [✓ Passive] Active

- [ ] Keyboard sends Telnet special commands
- [✓] Return key sends Telnet New Line instead of ^M

Figure 2: Setting *Telnet* options in PuTTY

Step 3

- After hitting *Enter* in your telnet client, you should see a black screen and a flashing cursor prompt. Confirm that a connection has been established by observing TCP traffic to `128.250.37.164` in Wireshark. (IP Address may be different)
- Craft a HTTP request by entering '`GET / HTTP/1.0`' and hit *Enter* twice. The HTTP/1.0 specification as detailed in RFC 1945 requires an extra blank line after the GET request.
- If you see a response may it be a success or an error/ found Stop the Wireshark trace.
- **Top tip:** The connection would have been closed after the response was sent back. To make another connection to the same host, you can right-click on the title bar in PuTTY and click *Restart Session*. See Figure 4. Currently, all HTTP 1.0 requests will be denied by uni. HTTP 1.1 is the de-facto standard now because HTTP 1.0 could not manage persistent connections.

4 Questions

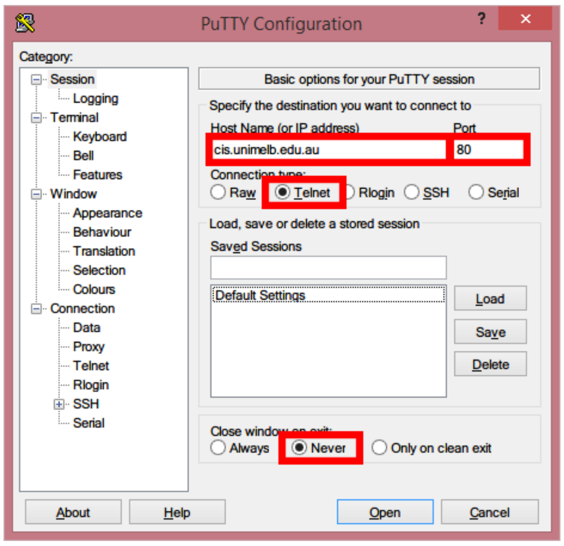
Discuss the following questions and record your answers while working with the people on your table.

1. What is the complete ASCII response from the web server after you sent the request '`GET / HTTP/1.0`'?
2. Open up the specification on HTTP/1.0 at `http://tools.ietf.org/html/rfc1945`. Under section 9, what do the different categories of status codes 2xx, 3xx, 4xx and 5xx refer to?
3. In Wireshark, right click on a HTTP packet and click *Follow TCP stream*. Here you can observe the connection-oriented nature of the communication between the host and your computer. Play around and explore with this window, and when you're ready click *Close*.

   A display filter would have been automatically applied to filter the TCP connection associated with the HTTP request. Click any packet and expand the *Internet Protocol Version 4* field in the middle section of Wireshark. Correlate the fields with Figure 5.

   On the menu bar on top, click *Statistics* $\Rightarrow$ *Flow Graph*. Select *Displayed packets* under *Choose packets* and click *OK*. This makes it easier to see the packets being transferred back and forth. Can you identify the packets corresponding to the TCP 3-way handshake and connection teardown? How do the `Seq` numbers and `Ack` numbers work? Confirm with your answer with your tutor.
4. Apart from GET, what are the other two methods in HTTP/1.0? Write a one or two sentence description for each of the three methods, with reference to RFC 1945.
5. Try requesting the CIS web page with HTTP/1.1 over telnet.

   Refer to Section 5.1.2 of RFC 2616 at `http://tools.ietf.org/html/rfc2616` for the correct request format.
6. What is the difference between HTTP/1.0 and HTTP/1.1? (Both from your observations in this lab and your own research.)
7. What security implications does HTTP have, relating your answer to what you have captured over Wireshark.
8. **Bonus:** What advantages does the new HTTP/2 standard bring?


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

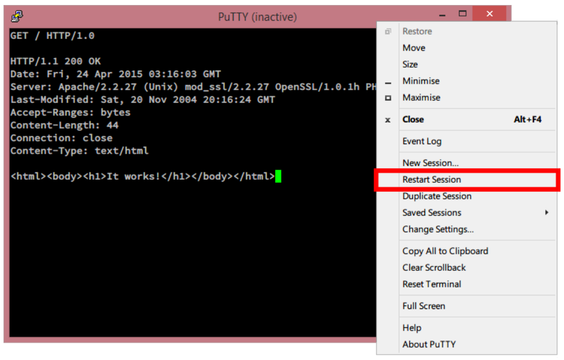
Host Name (or IP address): cis.unimelb.edu.au

Port: 80

Connection type: Raw [✓ Telnet] Rlogin SSH Serial

Close window on exit: Always [✓ Never] Only on clean exit

Figure 3: Setting host name and *Session* options in PuTTY


PuTTY (inactive)

```
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Fri, 24 Apr 2015 03:16:03 GMT
Server: Apache/2.2.27 (Unix) mod_ssl/2.2.27 OpenSSL/1.0.1h PH
Last-Modified: Sat, 20 Nov 2004 20:16:24 GMT
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html

<html><body><h1>It works!</h1></body></html>
```

Figure 4: Expected response from web server and how to restart the session (From 2015)

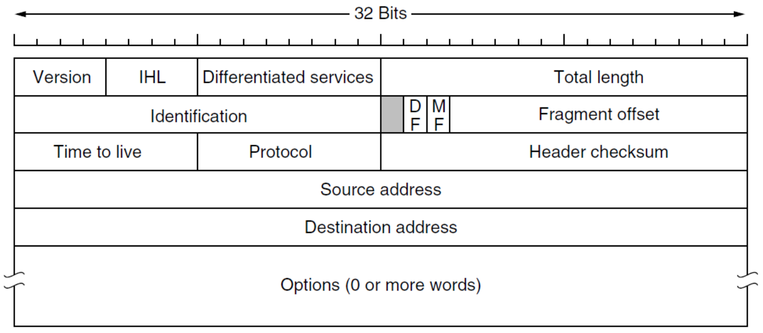


Figure 5: Structure of the IPv4 header