COMP3310/etc – IP Addressing

This tutorial assumes you have completed the previous UDP and TCP client server exercises.

Outline

In this tutorial you will

• Discover the network addresses of computers and understand two special values

• Study how port numbers can be used for transport layer multiplexing

1 Network addresses

For this part of the exercise you will need to work with another student, with both your computers connected to the same network. This could be two PCs in the same CSIT lab, or two laptops connected to the same wireless network.

The TCP client and server programs are the same.

1. In a shell on each computer, type **`ifconfig`** on a Linux or Mac, **`ipconfig`** on MS Windows.

This command shows the network interfaces on your computer and a lot of technical details about each. For now the only information we need is the `inet` (Internet Address) or `IPv4 Address`. This is four decimal numbers separated by periods – the same format as the `127.0.0.1` address shown in the server and client log messages.

> *Once Upon A Time a computer would have only one or two network interfaces. Modern 21st century computers may have lots of different network interfaces to search through.*
>
> *You may or may not see different addresses in* IPv6 *format as well. Some future version of this course will use IPv6 only, but not yet.*

Both addresses should start with the same first one to three numbers, indicating a shared network. For the examples my two home computers have IPv4 addresses `10.1.1.113` and `10.1.1.217`. (You will need to replace these with the addresses you found.)

(It is *possible* that this tutorial will run with two computers anywhere in the world, since that is why we use the Internet, but there are a lot of potential problems with no easy solutions. Both computers on the same network is more likely to work.)

Pick one computer to run the server and one for the client. The client will need the IP address of the computer running the server, here `10.1.1.113`. First verify that the client computer can connect to the server computer over the network with the **`ping`** command:

```
/Users/hugh/Desktop% ping 10.1.1.113
PING 10.1.1.113 (10.1.1.113): 56 data bytes
64 bytes from 10.1.1.113: icmp_seq=0 ttl=64 time=0.756 ms
64 bytes from 10.1.1.113: icmp_seq=1 ttl=64 time=0.363 ms
^C
```

If the client cannot ping the server, you may have mistyped the address. If not, repeat the **`ifconfig`** or **`ipconfig`** on the server computer and try different addresses until you get the right one.

For Wireshark, you will need to capture packets on the external interface, not the loopback as in previous tutorials. Since computers today are often very 'chatty', a filter

```
tcp and host 10.1.1.113
```

will show only TCP packets to or from the host in question.

2 TCP client and server

2. Run the server program on the server computer (in the example above, the computer with IP address `10.1.1.113`) with no command line arguments.

```
python tcpServer.py
```

or

```
java TcpServer
```

Run the client program on the other computer with the IP address of the server as the command line argument:

```
python tcpClient.py 10.1.1.113
```

or

```
java TcpClient 10.1.1.113
```

The client will crash with some kind of connection exception. But if you open a second shell on the server computer and run a client there, it will be able to send a request and get a reply. This is because the default server IP address `127.0.0.1` only works for programs on the same computer.

3. Control-C the server and restart with the IP address of the computer it is running on:

```
python tcpServer.py 10.1.1.113
```

or

```
java TcpServer 10.1.1.113
```

Run the client again on the other computer, with the server IP address, and this time it should work.

4. Control-C the server and start it again with this special address:

```
python tcpServer.py 0.0.0.0
```

or

```
java TcpServer 0.0.0.0
```

Run the client again: it should still work.

> *`127.0.0.1` means 'inside this computer'.*
>
> *`0.0.0.0` for a server means 'Whatever interface this computer has'. It allows the server program to be moved from computer to computer without needing to change the IP address.*

5. Swap the computer roles, with the server running on the former client computer and the client on what used to be the server.

3 Port numbers

Shut down any servers still running.

6. The client and server programs, UDP and TCP, all have a second CLI argument which is a port number. Start a TCP server with both network address and a new port number,

```
python tcpServer.py 0.0.0.0 6331
```

or

```
java TcpServer 0.0.0.0 6331
```

Run the client with just the IP address of the server: it will crash again with a connection exception. The IP address is correct, but transport layer connections are also identified by port number.

Run the client again with the right port number as a second CLI argument:

```
python tcpServer.py 10.1.1.113 6331
```

or

```
java TcpServer 10.1.1.113 6331
```

Now it should work.

How do clients know which port to use? This will be covered later in the course, but most common in the Internet are **`well known`** ports. A HTTP web server will be TCP port 80, HTTPS TCP port 443. A web server can be run on a different port, but then there has to be some way to tell the clients (web browsers) what it is.

In Wireshark, which TCP packets have port numbers?

4 Optional

Can you start a TCP (or UDP) server with a port number between 1 and 1023? Once Upon A Time, before personal computers were common, only root/administrator accounts could run programs using port numbers in this range. Some modern systems still have this limitation.

Try using an IPv6 address for server and client. What is the IPv6 equivalent to `0.0.0.0`?

Do the ICMP packets for pings and replies have port numbers?
