

CSIS0234B Computer and Communication Networks (Class B)

Tutorial 12

TCP operations in Linux

In this tutorial we will do various experiments to see how TCP operates in Linux. Form groups of 2 to perform the following steps, using 2 computers. In order to effectively test the TCP protocol, we prepared a pair of small programs (`tcp_svc` and `tcp_cli`) that perform TCP operations as instructed by the user, with three commands: `send`, `receive` and `close`. They are placed in the home directory of the `root` user, which you should log into during the whole tutorial. As the first step, you should type `make` to compile the programs. You will slightly modify the program later in the tutorial.

1. Connection management

- a. **Connection establishment.** Let's start by the first phase of the connection. Start ethereal in both computers, and start capturing packets of the two hosts (using `ip` and (`host <IP-addr1>` or `<IP-addr2>`)) **with the promiscuous mode off**. This capture should continue throughout the section. Then start the server and the client. Read the packets captured. Pay special attention to (1) the TCP flags, and (2) the TCP options set.
- b. Send 2 bytes of data from one side and receive it in the other. Read the sequence numbers used of each packet. You will find it somewhat hard to read the numbers because they are long. From the Edit->Preference->Protocols->TCP option pane, select "Analyze TCP sequence numbers" and "Use relative sequence numbers". They cause the sequence number to be displayed as an offset to the first TCP packet of the same connection, which makes it much easier to read. Understand the sequence numbers used in both the actual data and the acknowledgements.
- c. **Connection closing.** With the capturing still on, stop the server first (using Control-C), and then the client. Now try to start the server again. You should notice that it becomes impossible to start the server. To understand why, use `netstat -nt` to see the TCP states in both computers.
- d. Understand the sequence numbers used by the FIN packets.
- e. Now run the server and the client again. Repeat step (c) above, but this time stop the client before the server to see the difference.
- f. Stop both the client and the server. Modify the `cmd_close()` function in the `cmd_loop.c` file, so that it uses `shutdown()` instead of `close()`, taking the number the user provides (in `arg1`) as argument. Now restart both the server and the client. From the client side, close only the send (write) side of the connection. Confirm that the server can still send data to the client. From another terminal, observe what is the states of the connection in both computers using `netstat -nt`.
- g. Close the client completely and send from the server again twice. Look at the captured packets and the current TCP states.
- h. Devise a way to see a connection in the `FIN_WAIT1` and `LAST_ACK` state.

2. Window management

- a. **Nagle.** Capture packets while one side is sending 1 byte packets to the other for 200 times, without delay. See what are the sizes of the packets captured. Repeat the experiment when (a) a delay of 10ms and 50ms are used, and (b) when the sender sends 2000 bytes 10 times without delay. In the last case, notice that the kernel starts delaying acknowledgements when large packets are received repeatedly.
- b. **Window probing.** Restart the client and then the server. Start capturing again. Let the server send 150000 bytes of data once, without the receiver receiving. Look at the window field of the generated packets. You might find it necessary to colorize the display with the Display->Colorize Display option.
- c. **Clark.** From the client side, receive 300 bytes manually one after the another. See when a window advertisement is generated by the server.
- d. **Window scaling.** From the initial SYN packet, we know that no scaling is done in the current connections. We will enlarge the receiver buffer to trigger window scaling. First, setup the system to allow larger buffer: see the content of `/proc/sys/net/core/rmem_max` with `cat`. Change the value to 262144 with `echo 262144 > /proc/sys/net/core/rmem_max`. Then look at `/proc/sys/net/ipv4/tcp_rmem` to see its format (the last number is the maximum receive buffer size), and use similar format to set its maximum to 262144 as well. Now modify the server program to set the `SOL_SOCKET` level socket option `SO_RCVBUF` of the **listening** socket to 131072 (the kernel artificially would double it to 262144), and run the server and the client again when capturing. Confirm that a non-zero window scaling option is used in the SYN packet.
- e. From the client, send 500000 bytes of data to the server when capturing the packets. Observe how the window size of the packets change, and check the last phase of the connection when the receive window size starts decreasing to confirm that the size field is scaled.

3. PMTU discovery

We will use 4 computers to see how TCP deals with different MTU (Maximum Transfer Unit) across the network. Join with a group besides you, or find two more computers around.

- a. **Setting up the computers.** Call the 4 computers A, B, C and D. In A, B and C, add a host route (using `route add <dest> gw <nexthop>`) so that packets from A to D passes through B and C in that order. In the reverse direction, in B, C and D, add a host route so that packets from D to A passes through C and B in that order. In B, setup the MTU to 900 (using `ifconfig eth0 mtu 900`); and in C, setup the MTU to 700. (A and D uses the default MTU, 1500.) Also, stop the `iptables` service of all the 4 computers, and enable forwarding in B and C.
- b. **Discovery process.** Start capturing in all four computers (remember to change the filter to include all the four computers). Now start the server in A, and let D connect to it. Send 1400 bytes from A to D and see what packets can be seen in each of the computers. Take special note to the flags in the IP header, as well as the content of the ICMP message delivered.
- c. **Too small MTU.** Linux refuses to use a small MTU for TCP. Change C to use an MTU of 500, and repeat the last step. See what packets are received at D.