Project 3: Making TCP Applications Survive Address Changes

When a host moves to a new network and obtains a new IP address, its ongoing TCP conversation will not be able to continue because TCP uses source/destination addresses to identify packets belonging to the same conversation. While there are a number of solutions, including Mobile IP, in this project we will implement a simple solution that does not require any infrastructural support, OS kernel changes, or changes to applications.

We use **telnet** as the application example in this project. The goal is to make a telnet connection survive address changes. Telnet was the primary way to get remote shell access some years ago until ssh was introduced to replace it. The biggest difference between the two is that telnet sends everything in clear text, including username and password, while ssh encrypts all the traffic between client and server. We use telnet in this project for simplicity in order to focus on the networking aspect.

You will use the same virtual machines (VM) assigned in Project 1. You need to know the **eth1 IP address**, which is displayed as part of the system information when you login the VM.

Step 1: install and run telnet, demonstrate the problem

Between the two VMs assigned to you, pick one as the Server, and the other as the Client.

1. Login the Server, run the following command to install the telnet server daemon:
   sudo apt-get -y install xinetd telnetd

2. Use scp to transfer the two configuration files, inetd.conf and xinetd.conf, to your home directory on the Server. Then copy them into /etc:
   sudo cp inetd.conf /etc
   sudo cp xinetd.conf /etc

3. Start the telnet server:
   sudo /etc/init.d/xinetd restart

4. Test it by telnet from the Client to the Server. On the Client, run the following:
   telnet Server-eth1-Address

It prompts you for the username and password on the Server VM. After successful login, you get a regular Unix shell session.

5. While in the telnet session, run a ping command:
ping localhost

This is just a trivial way to generate some continuous traffic. It keeps displaying the result of each ping at one-second interval with increasing icmp_seq number. The command is executed on the Server, but the results are transmitted to and displayed on the Client via the telnet session. As long as the session is working, you should see the ping results keep coming and should have no gap in icmp_seq, i.e., no data loss.

6. In another Client terminal, remove the Client's eth1 address:
sudo ip addr del Client-eth1-Address/24 dev eth1

This removes the current address from the eth1 interface. You can use the following command to show the interface information before and after any change:
ip addr show eth1

Once the current address is removed, the ping results in the telnet session will stop immediately. Actually the ping program is still running on the Server, but because the TCP connection has been severed by the address removal, the results cannot be transferred to the Client. Similarly, no input from the Client can be transferred to the Server, so the telnet session appears hanging.

7. Add a new address to Client's eth1:
sudo ip addr add new-ip-address/24 dev eth1

Since your topology is assigned a /24 prefix, the new address must be in the same /24. In other words, it should share the same first 3 bytes as the old address, but the last byte can be any number between 1 and 254, except those that are already taken by the Server and the Client's old address.

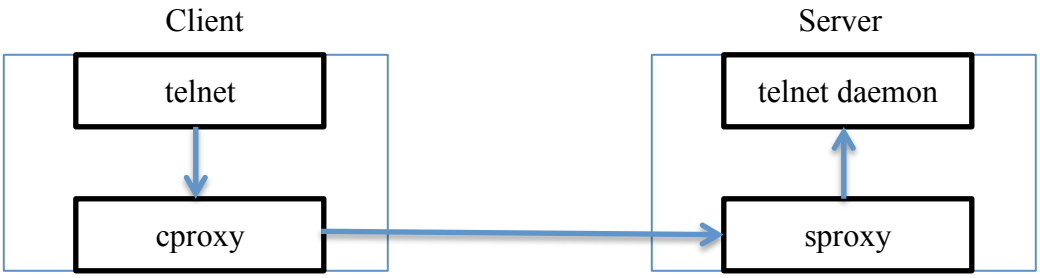
Because the existing telnet session was established using the old address, adding a new address doesn't resume the session, which will still be hanging, not responding to any key stroke. That's the problem we want to fix in this project.

To terminate the hanging telnet session, type "ctrl]" and then "ctrl d".

You should also kill the ping process at the Server.

Step 2: The basic proxy programs

Our goal is to be able to resume the telnet session after the address change, and do so without making changes to the telnet program or the TCP implementation in the OS kernel. The idea is to write two proxy programs: "cproxy" running on the Client, and "sproxy" running on the Server. The two customized proxy programs detect and handle address changes, while the telnet program only communicates with local proxy program without being affected by any address change.



1. After implementing the proxy programs, run them on the Client and Server respectively. The sproxy should listen to TCP port 6200, and takes no command-line argument. The cproxy should listen to TCP port 5200, and take one argument that is the eth1 IP address of the Server, i.e.,

cproxy w.x.y.z

2. Instead of telnet into the Server directly, on the Client you telnet into cproxy:

telnet localhost 5200

After accepting a connection from local telnet, cproxy should make a TCP connection to Server port 6200, i.e., connect to sproxy. Upon accepting this connection, sproxy should make a TCP connection to localhost (127.0.0.1) port 23, which is what the telnet daemon is listening on.

3. Now the cproxy has two connections and so does the sproxy. Their job is to relay packets between the two sockets. For example, cproxy calls select( ) on both of its sockets, and whenever there is data to read from one socket, it will read the data and write it to the other socket. This way any user input to the telnet terminal will be passed by cproxy to sproxy, which then passes it to the telnet daemon on Server. Similarly data sent by the telnet daemon will be passed by sproxy to cproxy, which then passes it to the telnet program on the Client.

This replaces the single direct TCP connection between telnet and telnet daemon by 3 separate TCP connections stitched together by the proxy programs. Run the ping program or any command in the telnet session and it should work just as fine as the single TCP connection.

In this way the telnet and telnet daemon connect to programs on the localhost, whose address (127.0.0.1) never changes. Thus we mask the issue of address changes from the applications, and only need to develop a solution between cproxy and sproxy.

Step 3 Detecting and Handling Address Changes

As the Client changes its address, cproxy needs to detect such a change, close the socket associated with the old address, reconnect to sproxy, and resume the application session.

1. cproxy exchanges periodic heartbeat messages with sproxy to detect the loss of connection. At every 1s cproxy and sproxy should send to each other a heartbeat message. Both proxy programs treat the missing of three consecutive heartbeats as the loss of connection. They will then close the failed socket. **You can use select( ) to implement the 1s timeout.**

2. cproxy should try connecting to sproxy again. When the new address is added, such a connection request will go through and the connection between croxy and sproxy is reestablished using new sockets. Application data should now flow again. In our example, ping results will show up again.

3. cproxy and sproxy should not lose any packets, e.g., no gap in icmp_seq, in the entire process. Therefore cproxy and sproxy need a mechanism similar to TCP's sequence number and acknowledge number between them. For example, some pending packets in the buffer may be gone when you close the old sockets. Upon reestablishing the session on the new address, the two proxies need to retransmit the missing data if any.

4. From sproxy's point of view, when a new connection with cproxy is established, there are two possibilities: it is either a brand new telnet session, or it is a continuation of the existing session after the Client got a new address. In the former case, sproxy should close the existing connection with telnet daemon, and start a new connection with the daemon for a new telnet session. In the latter case, sproxy should keep using the existing connection with the daemon to resume the hanging session. Therefore sproxy needs to be able to differentiate these two cases, which needs information from cproxy.

To address all these issues, you need to define a packet format for traffic between cproxy and sproxy. There are three types of packets: heartbeat, new connection initiation, and application data. All packets should have a header and a payload. The header needs to identify which type of packet it is, the length of payload, and the sequence and acknowledgement numbers for loss detection. For heartbeat and connection initiation, the payload can be empty.

Based on the packet format, you need define the protocol actions between cproxy and sproxy, i.e., how each packet is being processed.

Requirements and Deliverables

- A project report that documents the design of the packet format and protocol actions.
- Implement cproxy and sproxy. Together they should make a telnet session survive address changes without any loss of application data. E.g., after removing the old address, cproxy and sproxy should detect the failure after 3s and close the sockets. After adding the new address, cproxy should reconnect with sproxy, retransmit any data that got lost during the transition, and resume the session.

- You can assume only one telnet session is alive at any time, but need to differentiate the continuation of an existing session from a brand new session. E.g, after removing the old address, the Client terminates the telnet session; after the new address is added, the Client starts a new session. The proxies should be able to support this use case.

Extra Credit

There is a 10% extra credit for supporting multiple concurrent telnet sessions. You can achieve it by different means. One approach is to use one worker thread per session, and let the master thread accept new connections, spawn new threads for brand new sessions or passing the connected socket to an existing thread to resume communication. Another approach is to stick to single thread, but use select( ) to multiplex both accepting new connections and reading data from sockets. This may need to set non-blocking mode for the accept( ) call.

Select() or Poll():

You need **select( ) or poll( )** to multiplex I/O from multiple sockets in your programs. Which one to use is your choice.

http://beej.us/guide/bgnet/output/html/multipage/selectman.html
http://beej.us/guide/bgnet/output/html/multipage/pollman.html