PB17121706 ZhangDongquan

Philip Chang

November 18, 2019

Experiment Contents

Implement reliable transmission using GO-BACK-N on an unreliable channel.

Environment

Ubuntu 16.04

Steps and Codes

sender

Firstly we must unblock the receiving procedure otherwise receiving ack packets will block the process of sending packets.
Codes are as below.

```
int UNBLOCK(int fd)
{
    int flags = fcntl (fd, F_GETFL, 0);
    if (flags == -1)
    {
        return -1;
    }
    if(fcntl (fd, F_SETFL, flags | O_NONBLOCK) == -1)
    {
        return -1;
    }
    return 0;
}// make receive procedure unblocking, from Internet
```

Then the GO-BACK-N parts.

```
if((double)(time - TIME) / CLOCKS_PER_SEC >= 0.5)
{
    int j = base;
    while(j <= i)
    {
        packet[j].HEADER.flag = 1;
        if(NUM != 64)
            send(sockfd, packet + j, 7 + NUM, 0);
        else
            send(sockfd, packet + j, 7 + 64, 0);
        j++;
    }
    TIME = time;
}// resend file
```

Notice that the send() is spilit into two conditions because the final packets may contain less than 64 bits data. If we send 71 bits it may send unknown data which we do not want.

receiver

```
int NUM = read(clientfd, &packet, 64+7);
if(packet.HEADER.btcp_seq == ACK)
{
    packet.HEADER.btcp_ack = ACK;
    send(clientfd, &packet,7,0);// overwrite ACK and send the header back
    ACK = (ACK + 1) % 256;
    int j = 0;
    while(j < NUM - 7)
    {
        fwrite(&packet.DATA[j],sizeof(unsigned char),1,file);
        j++;
    }
    if(NUM != 71)
    {
        fclose(file);
        close(sockfd);
        return;
    }
}
```

When receiving the packets, we change the ack bit of the header and send the header back as the ACK.

Experiment Result

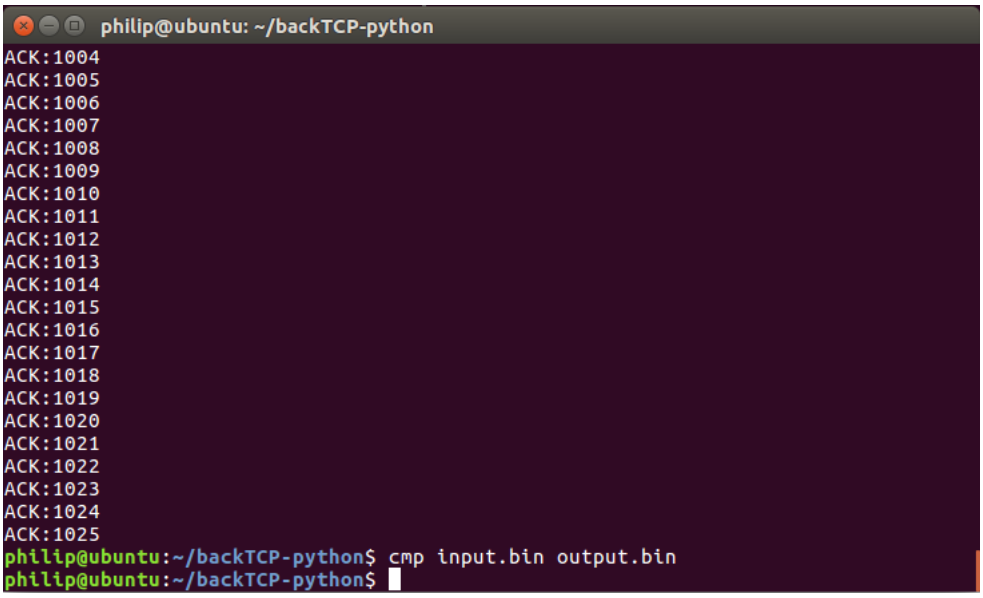
recv-terminal

```
philip@ubuntu: ~/backTCP-python
philip@ubuntu:~/backTCP-python$ gcc send.c -o send
philip@ubuntu:~/backTCP-python$ ./recv
philip@ubuntu:~/backTCP-python$ gcc send.c -o send
philip@ubuntu:~/backTCP-python$ ./recv
^C
philip@ubuntu:~/backTCP-python$ gcc send.c -o send
philip@ubuntu:~/backTCP-python$ ./recv
philip@ubuntu:~/backTCP-python$
```

channel

```
philip@ubuntu: ~/backTCP-python
Traceback (most recent call last):
  File "/usr/lib/command-not-found", line 27, in <module>
    from CommandNotFound.util import crash_guard
  File "/usr/lib/python3/dist-packages/CommandNotFound/__init__.py", line 3, in
<module>
    from CommandNotFound.CommandNotFound import CommandNotFound
  File "/usr/lib/python3/dist-packages/CommandNotFound/CommandNotFound.py", line
 9, in <module>
    import gdbm
ModuleNotFoundError: No module named 'gdbm'
philip@ubuntu:~/backTCP-python$ python3 testch.py
^CTraceback (most recent call last):
  File "testch.py", line 122, in <module>
    main()
  File "testch.py", line 118, in main
    btMITM(args.out_addr, args.out_port, args.in_addr, args.in_port)
  File "testch.py", line 44, in btMITM
    in_sock = backTCP.BTcpConnection('recv', in_addr, in_port)
  File "/home/philip/backTCP-python/backTCP.py", line 23, in __init__
    self.conn, self.remote_addr = self.sock.accept()
  File "/usr/lib/python3.6/socket.py", line 205, in accept
    fd, addr = self._accept()
KeyboardInterrupt
philip@ubuntu:~/backTCP-python$
```

send-terminal



In the sender terminal, we do a little check after closing the socket. Using cmp instruction we know that input and output files are the same. The actual file is not shown because .bin document is not visible for people.

P.S.

To run the send and recv program, the input file must be named ”input.bin” and it will create a output file ”output.bin”