Computer Networks Lab2 Evaluation

Member: Haodong Duan Muge Chen

Work:

- Muge Chen: implement TCP SYN&ACK; implement TCP stop & wait
- Haodong Duan: searching lab material, implement IPElement, implement 2 APP Elements(Client & Server) , design the TCPElement , implement TCP flow control, congestion control, FIN&ACK, stop&wait debug, debug & test

IP section

- click for testing of basic functions：

```
ip1 :: IPElement(MY_ADDRESS 1.2.3.4,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
ip2 :: IPElement(MY_ADDRESS 3.4.5.6,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
ip3 :: IPElement(MY_ADDRESS 5.6.7.8,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
Idle()->[0]ip1[1]->[1]ip3[2]->[1]ip2[0]->Discard();
Idle()->[0]ip2[1]->[2]ip3[1]->[1]ip1[0]->Discard();
Idle()->[0]ip3[0]->Discard();
```

- the args of IPElements are my_address, period of sending out hello (sec) , period of sending out routing table (sec)
- here is some log for debug

```
301: RENEW PAIR: MYIP 4030201,DEST 8070605
102:MYIP IS :4030201  AND I RECIEVE RESP FROM IP 8070605 PORT 1, RENEW MY DEST TABLE
101:MYIP IS :8070605  AND I RECIEVE HELLO FROM IP 4030201 PORT 1, AND SEND BACK A RESP
002: My IP is 4030201, I just broadcast Hello message to 1 ports
301: RENEW PAIR: MYIP 6050403,DEST 8070605
102:MYIP IS :6050403  AND I RECIEVE RESP FROM IP 8070605 PORT 1, RENEW MY DEST TABLE
101:MYIP IS :8070605  AND I RECIEVE HELLO FROM IP 6050403 PORT 2, AND SEND BACK A RESP
002: My IP is 6050403, I just broadcast Hello message to 1 ports
301: RENEW PAIR: MYIP 8070605,DEST 4030201
102:MYIP IS :8070605  AND I RECIEVE RESP FROM IP 4030201 PORT 1, RENEW MY DEST TABLE
101:MYIP IS :4030201  AND I RECIEVE HELLO FROM IP 8070605 PORT 1, AND SEND BACK A RESP
301: RENEW PAIR: MYIP 8070605,DEST 6050403
102:MYIP IS :8070605  AND I RECIEVE RESP FROM IP 6050403 PORT 2, RENEW MY DEST TABLE
101:MYIP IS :6050403  AND I RECIEVE HELLO FROM IP 8070605 PORT 1, AND SEND BACK A RESP
002: My IP is 8070605, I just broadcast Hello message to 2 ports
```

The IPElement did the HELLO & RESPONDE properly (between directly connected pair), we will omit these items in the following log

- here is log about exchanging routing table, we do this by carrying distance pair in the IP Packet:

```
003: My IP is 4030201, I just broadcast Exchange message to 1 ports
EXCHANGE PACKET CONTAINS: SRC: 4030201; DEST: 8070605; DIST: 1
103:MYIP IS :8070605  AND I RECIEVE EXCHANGE FROM IP 4030201 PORT 1
EXCHANGE PACKET RECV: LENGTH 1
EXCHANGE PACKET RECV: SRC 4030201, DEST: 8070605, DIST: 1
400: AFTER DIJK: WE(IP: 8070605) ARE CONNECTED TO THE FOLLOWING IP
IP: 4030201;  NEXTHOP: 1
IP: 6050403;  NEXTHOP: 2
103:MYIP IS :6050403  AND I RECIEVE EXCHANGE FROM IP 4030201 PORT 1
EXCHANGE PACKET RECV: LENGTH 1
EXCHANGE PACKET RECV: SRC 4030201, DEST: 8070605, DIST: 1
400: AFTER DIJK: WE(IP: 6050403) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 4030201;  NEXTHOP: 1
103:MYIP IS :8070605  AND I RECIEVE EXCHANGE FROM IP 4030201 PORT 1, FORWARD IT TO PORT
2
004: My IP is 4030201, I will check if I should erase port #1
003: My IP is 6050403, I just broadcast Exchange message to 1 ports
EXCHANGE PACKET CONTAINS: SRC: 6050403; DEST: 8070605; DIST: 1
EXCHANGE PACKET CONTAINS: SRC: 6050403; DEST: 4030201; DIST: 2
EXCHANGE PACKET CONTAINS: SRC: 8070605; DEST: 4030201; DIST: 1
103:MYIP IS :8070605  AND I RECIEVE EXCHANGE FROM IP 6050403 PORT 2
EXCHANGE PACKET RECV: LENGTH 3
EXCHANGE PACKET RECV: SRC 6050403, DEST: 8070605, DIST: 1
EXCHANGE PACKET RECV: SRC 6050403, DEST: 4030201, DIST: 2
EXCHANGE PACKET RECV: SRC 8070605, DEST: 4030201, DIST: 1
400: AFTER DIJK: WE(IP: 8070605) ARE CONNECTED TO THE FOLLOWING IP
IP: 4030201;  NEXTHOP: 1
IP: 6050403;  NEXTHOP: 2
103:MYIP IS :4030201  AND I RECIEVE EXCHANGE FROM IP 6050403 PORT 1
EXCHANGE PACKET RECV: LENGTH 3
EXCHANGE PACKET RECV: SRC 6050403, DEST: 8070605, DIST: 1
EXCHANGE PACKET RECV: SRC 6050403, DEST: 4030201, DIST: 2
EXCHANGE PACKET RECV: SRC 8070605, DEST: 4030201, DIST: 1
400: AFTER DIJK: WE(IP: 4030201) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 6050403;  NEXTHOP: 1
103:MYIP IS :8070605  AND I RECIEVE EXCHANGE FROM IP 6050403 PORT 2, FORWARD IT TO PORT
1
004: My IP is 6050403, I will check if I should erase port #1
003: My IP is 8070605, I just broadcast Exchange message to 2 ports
EXCHANGE PACKET CONTAINS: SRC: 8070605; DEST: 4030201; DIST: 1
EXCHANGE PACKET CONTAINS: SRC: 8070605; DEST: 6050403; DIST: 1
EXCHANGE PACKET CONTAINS: SRC: 4030201; DEST: 6050403; DIST: 2
103:MYIP IS :4030201  AND I RECIEVE EXCHANGE FROM IP 8070605 PORT 1
EXCHANGE PACKET RECV: LENGTH 3
EXCHANGE PACKET RECV: SRC 8070605, DEST: 4030201, DIST: 1
EXCHANGE PACKET RECV: SRC 8070605, DEST: 6050403, DIST: 1
EXCHANGE PACKET RECV: SRC 4030201, DEST: 6050403, DIST: 2
400: AFTER DIJK: WE(IP: 4030201) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 6050403;  NEXTHOP: 1
103:MYIP IS :6050403  AND I RECIEVE EXCHANGE FROM IP 8070605 PORT 1
```

```
EXCHANGE PACKET RECV: LENGTH 3
EXCHANGE PACKET RECV: SRC 8070605, DEST: 4030201, DIST: 1
EXCHANGE PACKET RECV: SRC 8070605, DEST: 6050403, DIST: 1
EXCHANGE PACKET RECV: SRC 4030201, DEST: 6050403, DIST: 2
400: AFTER DIJK: WE(IP: 6050403) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 4030201;  NEXTHOP: 1
004: My IP is 8070605, I will check if I should erase port #2
```

The function of exchanging routing table work well, to evaluate the DIJK part , we use a simple ring network with 5 nodes on the ring

```
rp1 :: RouterPort(DEV veth1, IN_MAC 76:49:97:6c:10:82 , OUT_MAC ce:7a:7b:4e:b5:b9 );
rp2 :: RouterPort(DEV veth2, IN_MAC ce:7a:7b:4e:b5:b9 , OUT_MAC 76:49:97:6c:10:82 );
ip1 :: IPElement(MY_ADDRESS 1.2.3.4,PERIOD_HELLO 1, PERIOD_EXCHANGE 5);
ip2 :: IPElement(MY_ADDRESS 3.4.5.6,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
ip3 :: IPElement(MY_ADDRESS 5.6.7.8,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
ip4 :: IPElement(MY_ADDRESS 7.8.9.10,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
ip5 :: IPElement(MY_ADDRESS 9.10.11.12,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
Idle()->[0]ip1[1]->[1]ip2[2]->rp1;
rp2->[2]ip3[1]->[1]ip4[0]->Discard();
Idle()->[0]ip4[1]->[1]ip3[2]->rp2;
rp1->[2]ip2[1]->[1]ip1[0]->Discard();
Idle()->[3]ip1[2]->[1]ip5[2]->[2]ip4[3]->Discard();
Idle()->[3]ip4[2]->[2]ip5[1]->[2]ip1[3]->Discard();
Idle()->[0]ip2[0]->Discard();
Idle()->[0]ip3[0]->Discard();
Idle()->[0]ip5[0]->Discard();
```

The output log shows that the DIJK part works well

```
103:MYIP IS :8070605  AND I RECIEVE EXCHANGE FROM IP a090807 PORT 1
400: AFTER DIJK: WE(IP: 8070605) ARE CONNECTED TO THE FOLLOWING IP
IP: a090807;  NEXTHOP: 1
IP: 6050403;  NEXTHOP: 2
IP: c0b0a09;  NEXTHOP: 1
IP: 4030201;  NEXTHOP: 2
103:MYIP IS :c0b0a09  AND I RECIEVE EXCHANGE FROM IP a090807 PORT 2
400: AFTER DIJK: WE(IP: c0b0a09) ARE CONNECTED TO THE FOLLOWING IP
IP: 4030201;  NEXTHOP: 1
IP: a090807;  NEXTHOP: 2
IP: 6050403;  NEXTHOP: 1
IP: 8070605;  NEXTHOP: 2
103:MYIP IS :4030201  AND I RECIEVE EXCHANGE FROM IP a090807 PORT 2
400: AFTER DIJK: WE(IP: 4030201) ARE CONNECTED TO THE FOLLOWING IP
IP: 6050403;  NEXTHOP: 1
IP: c0b0a09;  NEXTHOP: 2
IP: a090807;  NEXTHOP: 2
IP: 8070605;  NEXTHOP: 1
103:MYIP IS :6050403  AND I RECIEVE EXCHANGE FROM IP a090807 PORT 1
400: AFTER DIJK: WE(IP: 6050403) ARE CONNECTED TO THE FOLLOWING IP
IP: 4030201;  NEXTHOP: 1
IP: 8070605;  NEXTHOP: 2
IP: c0b0a09;  NEXTHOP: 1
IP: a090807;  NEXTHOP: 2
```

To evaluate whether our IPElement have the ability to add or delete IP address in its routing table, we use the following click file:

```
ip1 :: IPElement(MY_ADDRESS 1.2.3.4,PERIOD_HELLO 30,PERIOD_EXCHANGE 30);
ip2 :: IPElement(MY_ADDRESS 3.4.5.6,PERIOD_HELLO 1,PERIOD_EXCHANGE 5);
ip3 :: IPElement(MY_ADDRESS 5.6.7.8,PERIOD_HELLO 30,PERIOD_EXCHANGE 5);
Idle()->[0]ip1[1]->[1]ip3[2]->[1]ip2[0]->Discard();
Idle()->[0]ip2[1]->[2]ip3[1]->[1]ip1[0]->Discard();
Idle()->[0]ip3[0]->Discard();
```

We set the port time_out to be 20s, as in a 30-second period, there won't be any traffic between ip1 and ip2, we suppose at first, ip2 and ip3 will delete ip1 from their routing tables, until the second period begin, they will add ip1 to their routing table again

The log shows that at 20s, port1 of ip3 timeout, erase links through that port

```
IP: 4030201, PORT: 1, PORT_AGE: 5
004: My IP is 4030201, I will check if I should erase my 1 ports

103:MYIP IS :8070605  AND I RECIEVE EXCHANGE FROM IP 6050403 PORT 2
400: AFTER DIJK: WE(IP: 8070605) ARE CONNECTED TO THE FOLLOWING IP
IP: 6050403;  NEXTHOP: 2
IP: 4030201;  NEXTHOP: 1
IP: 6050403, PORT: 1, PORT_AGE: 5
004: My IP is 6050403, I will check if I should erase my 1 ports
103:MYIP IS :6050403  AND I RECIEVE EXCHANGE FROM IP 8070605 PORT 1
400: AFTER DIJK: WE(IP: 6050403) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 4030201;  NEXTHOP: 1
IP: 8070605, PORT: 1, PORT_AGE: 20
200: MY IP 8070605, ERASE PORT: 1
IP: 8070605, PORT: 2, PORT_AGE: 5
004: My IP is 8070605, I will check if I should erase my 2 ports
ToDevice(veth2): Message too long
103:MYIP IS :4030201  AND I RECIEVE EXCHANGE FROM IP 8070605 PORT 1
400: AFTER DIJK: WE(IP: 4030201) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 6050403;  NEXTHOP: 1
```

at 25s, ip2 and ip3 drop ip1 from their routing tables

```
IP: 4030201, PORT: 1, PORT_AGE: 5
004: My IP is 4030201, I will check if I should erase my 1 ports

103:MYIP IS :8070605  AND I RECIEVE EXCHANGE FROM IP 6050403 PORT 2
400: AFTER DIJK: WE(IP: 8070605) ARE CONNECTED TO THE FOLLOWING IP
IP: 6050403;  NEXTHOP: 2
IP: 6050403, PORT: 1, PORT_AGE: 5
004: My IP is 6050403, I will check if I should erase my 1 ports
103:MYIP IS :6050403  AND I RECIEVE EXCHANGE FROM IP 8070605 PORT 1
400: AFTER DIJK: WE(IP: 6050403) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 8070605, PORT: 1, PORT_AGE: 0
IP: 8070605, PORT: 2, PORT_AGE: 5
004: My IP is 8070605, I will check if I should erase my 2 ports
103:MYIP IS :4030201  AND I RECIEVE EXCHANGE FROM IP 8070605 PORT 1
400: AFTER DIJK: WE(IP: 4030201) ARE CONNECTED TO THE FOLLOWING IP
IP: 8070605;  NEXTHOP: 1
IP: 6050403;  NEXTHOP: 1
```

Then, at 30s, due to the new HELLO and RESP, the network returns to its original state

We don't need to verify the transportation ability of our IPElement, the following TCP and APP part will help us do that.

TCP Section

- For convenience, we did some assumption in this part of the lab, first, I devide TCPElement in 2 types: TCPElement for Client and TCPElement for server, so that each type only need to do certain part of work in connection establishment and connection release. In my setting, TCPElement for server is connected to a Server Element, while TCPElement for client is connected to a Client Element. TCPElement for server will passively establish and release a connection. (Our APP is, server send a file to a client)
- For convenience, we devide the four functions to implement in 3 files (obviously you can't put stop & wait and Tahoe Reno in a single file), in our folder, tcpv2 include TCP SYN/FIN, stop & wait, tcpv3 add flow control to tcpv2, tcpv4 implement simple TCP Reno.
- For convenience, our seq, ack, buffer are all on packet level.
- Here we list some test to our TCPElement:
- Test for TCPElement (tcpv2):
    - ```
      TCP: connection establishing..
       MY IP 5030201, MY PORT: 222, CO IP: 4030201, CO PORT: 111
      sent pack with seq: 0 ,ack 1
      before server:   32 | de006f00 00000000 01000000 2000066c 743a2030 01020305
      I'm server, I received packet from IP
      TCP: connection establishing..
       MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
      sent pack with seq: 1 ,ack 1
      after tcp:   32 | 6f00de00 01000000 01000000 20000200 00000000 01020304
      after ip:   52 | 01003400 00000000 1e000000 01020304 01020305 6f00de00
      TCP: connection establishing..
       MY IP 5030201, MY PORT: 222, CO IP: 4030201, CO PORT: 111
      sent pack with seq: 1 ,ack 2
      before server:   32 | de006f00 01000000 02000000 20000200 00000000 01020305
      I'm server, I received packet from IP
      Server received packet from client
      ```
    - the log of connection establishment period, here we are different from the traditional TCP: we need 4 packets, 3 for establishing a connection, 1 for trigger the Server side to send packet
      ```

- ```
  TCP: connection established..
   MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
  sent pack with seq: 6 ,ack 6
  after tcp:  544 | 6f00de00 06000000 06000000 20020100 00000000 01020304
  after ip:  564 | 01003402 00000000 1e000000 01020304 01020305 6f00de00
  TCP: connection established..
   MY IP 5030201, MY PORT: 222, CO IP: 4030201, CO PORT: 111
  sent pack with seq: 6 ,ack 7
  1:  512 | 01020305 de000002 0400690c 636f6e76 31220a09 746f703a
  I'm client, and I receive data from server
  packet 4 buffered
  I will quit
  before server:   32 | de006f00 06000000 07000000 20000200 00000000 01020305
  I'm server, I received packet from IP
  Server received packet from client
  TCP: connection established..
   MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
  sent pack with seq: 7 ,ack 7
  after tcp:  544 | 6f00de00 07000000 07000000 20020100 00000000 01020304
  after ip:  564 | 01003402 00000000 1e000000 01020304 01020305 6f00de00
  TCP: connection established..
   MY IP 5030201, MY PORT: 222, CO IP: 4030201, CO PORT: 111
  sent pack with seq: 7 ,ack 8
  1:  512 | 01020305 de000002 0500690c 643a2030 2e30310a 09097d0a
  I'm client, and I receive data from server
  packet 5 buffered
  I will quit
  ```
- Here we can see how packets are transported in our TCP connection, we can see stop & weight strategy (strictly paired DATA & ACK). (Some of the logs are from APP level)
- ```
  Server: data transportation finished, send FIN packet!
  after tcp:   32 | 6f00de00 6b0c0000 6b0c0000 20000800 00000000 01020304
  after ip:   52 | 01003400 00000000 1e000000 01020304 01020305 6f00de00
  TCP : recieved FIN from server!
  TCP: connection establishing..
   MY IP 5030201, MY PORT: 222, CO IP: 4030201, CO PORT: 111
  sent pack with seq: 3179 ,ack 3180
  before server:   32 | de006f00 6b0c0000 6c0c0000 20000a00 00000000 01020305
  I'm server, I received packet from IP
  Server in state fin_sent
  after tcp:   32 | 6f00de00 6b0c0000 6b0c0000 20000200 00000000 01020304
  after ip:   52 | 01003400 00000000 1e000000 01020304 01020305 6f00de00
  Client received ACK for FIN, connection down!
  chunk of FIN finished
  after ip:   20 | 10001400 00000000 01000000 01020304 ffffffff
  after ip:   20 | 20001400 00000000 01000000 01020304 01020305
  after ip:   20 | 10001400 00000000 01000000 01020304 ffffffff
  after ip:   20 | 20001400 00000000 01000000 01020304 01020305
  after ip:   20 | 10001400 00000000 01000000 01020304 ffffffff
  ```

    - after data transportation, we can see how the connection is released. After connection down, we can only see periodical IP packets on the wire.
- Test for TCPElement (tcpv3):
    - here we test our flow control in TCP
    - we suppose there is a buffer in our client point TCPElement, this Element periodically push packets to the APP level client, so that it's buffer can be full, we use this setting to simulate flow control
    - ```
      TCP: connection established..
       MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
      sent pack with seq: 21 ,ack 21
      after tcp:  544 | 6f00de00 15000000 15000000 20020100 00000000 01020304
      after ip:  564 | 01003402 00000000 1e000000 01020304 01020305 6f00de00
      TCP: connection established..
       MY IP 5030201, MY PORT: 222, CO IP: 4030201, CO PORT: 111
      sent pack with seq: 21 ,ack 22
      before server:   32 | de006f00 15000000 16000000 20000200 00000000 01020305
      I'm server, I received packet from IP
      Server received packet from client
      No more room in clients buffer, wait for a while
      ```
    - here we use 20 packet slot as a buffer, and we can see, when the available slots are all full, our sender will stop and wait for a while
    - in that situation, we should pay attention that as we push packet to client every x seconds, after connection closed, we should send left packets in the buffer to the client.
- Test for TCPElement (tcpv4):
    - here we test our congestion control, which include ACK clocking, slow-start, fast retransmit, fast recovery
    - brief introduction:
        - ACK clocking: ACK triggers every possible data packet sending;
        - slow-start: we start from a low RTT, after each successful cycle, multiply RTT by 2 (if timeout happens, RTT/=2, then each ACK will trigger RTT to increase by one);
            - PS: in our setting, we set max RTT to 1024 for convenience
        - fast retransmit: 3 same ACK cause a retransmit
        - fast recovery: after timeout, if later packets have been buffered in client point, send these ACKs to server to recover RTT fastly (here our implementation is tricky, we add an attribute count to TCPHeader to inform how many ACK packets one ACK packet actually means)
    - to simulate packet loss, in TCPElement client part, we randomly drop packet we received (at 1% possiblity), let's see our simulation result:
      ```

- ```
  after a successful transimission, cwnd is increased to 2
  TCP: connection established..
   MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
  sent pack with seq: 2 ,ack 2
  after tcp:  544 | 6f00de00 02000000 02000000 20020100 00010000 01020304
  after ip:  564 | 01003402 00000000 1e000000 01020304 01020305 6f00de00
  receiver received packet, with seq 2, ack 2
  .........
  after a successful transimission, cwnd is increased to 4
  TCP: connection established..
   MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
  sent pack with seq: 4 ,ack 4
  after tcp:  544 | 6f00de00 04000000 04000000 20020100 00010000 01020304
  after ip:  564 | 01003402 00000000 1e000000 01020304 01020305 6f00de00
  receiver received packet, with seq 4, ack 4
  .........
  after a successful transimission, cwnd is increased to 8
  TCP: connection established..
   MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
  sent pack with seq: 8 ,ack 8
  after tcp:  544 | 6f00de00 08000000 08000000 20020100 00010000 01020304
  after ip:  564 | 01003402 00000000 1e000000 01020304 01020305 6f00de00
  receiver received packet, with seq 8, ack 8
  ..........
  after a successful transimission, cwnd is increased to 128
  TCP: connection established..
   MY IP 4030201, MY PORT: 111, CO IP: 5030201, CO PORT: 222
  sent pack with seq: 130 ,ack 130
  after tcp:  544 | 6f00de00 82000000 82000000 20020173 00010000 01020304
  after ip:  564 | 01203402 30306465 1e302038 01020304 01020305 6f00de00
  ..........
  ```
- the slow start part
- ```
  I am 0,I lost packet with seq 130,ack 130
  ```
- Unfortuanately, our client lost a data packet, let's see what the server will do
- ```
  Server waiting for ACK, get wrong packet or not appropriate ACK
  Server received packet with seq 129,ack 130
  current last_ack 3
  three times, retransmission
  retransmission packet is with seq 130,ack 130
  already popped out 128 packets
  after tcp:  544 | 6f00de00 82000000 82000000 20020173 00010000 01020304
  after ip:  564 | 01703402 6f707269 1e746520 01020304 01020305 6f00de00
  receiver received packet, with seq 130, ack 130
  ```
- after three same ACK, our server know there is a packet lost, and do the retransmit
  ```

```
sent pack with seq: 130 ,ack 131
finally we got k to be 4
1:  512 | 01020305 de000002 80001c08 4e65742d 3138220a 6e616d65
I'm client, and I receive data from server
packet 128 buffered
I will quit
1:  512 | 01020305 de000002 81001c08 22526573 4e65742d 3138220a
I'm client, and I receive data from server
packet 129 buffered
I will quit
1:  512 | 01020305 de000002 82001c08 68617065 207b0a20 20202020
I'm client, and I receive data from server
packet 130 buffered
I will quit
1:  512 | 01020305 de000002 83001c08 696c6c65 72207b0a 09090974
I'm client, and I receive data from server
packet 131 buffered
I will quit
CLIENT DATA:send out ack with seq 130,ack 131,count 4
```

- after received right packet, client found it had buffered the following 3 packets, so it push them all to the APP level, and set the count to 4, which make this ACK actually represents 4 ACKs. Which can be useful in RTT recovery
- in this setting, as the loss rate is small, during the 2000 packets transportation, the cwnd finally reach 1024, in our further test, we add latency to the packet sending and found the cwnd varies at a stable level.

APP Section

- We have 2 APP Element, client and server. Server start transmission of data (here we implement 2 options: Message(string) and File). When we send file, we split it into 500 bytes size small packets, add our APPHeader to it (which includes seq number), then push it to TCP level. When our client received packets, it will reorgnize it by seq numbers, and write it to filesystem.
- There is no need to show any log in this part, you can run the iptcpapp.click to see the detailed log.