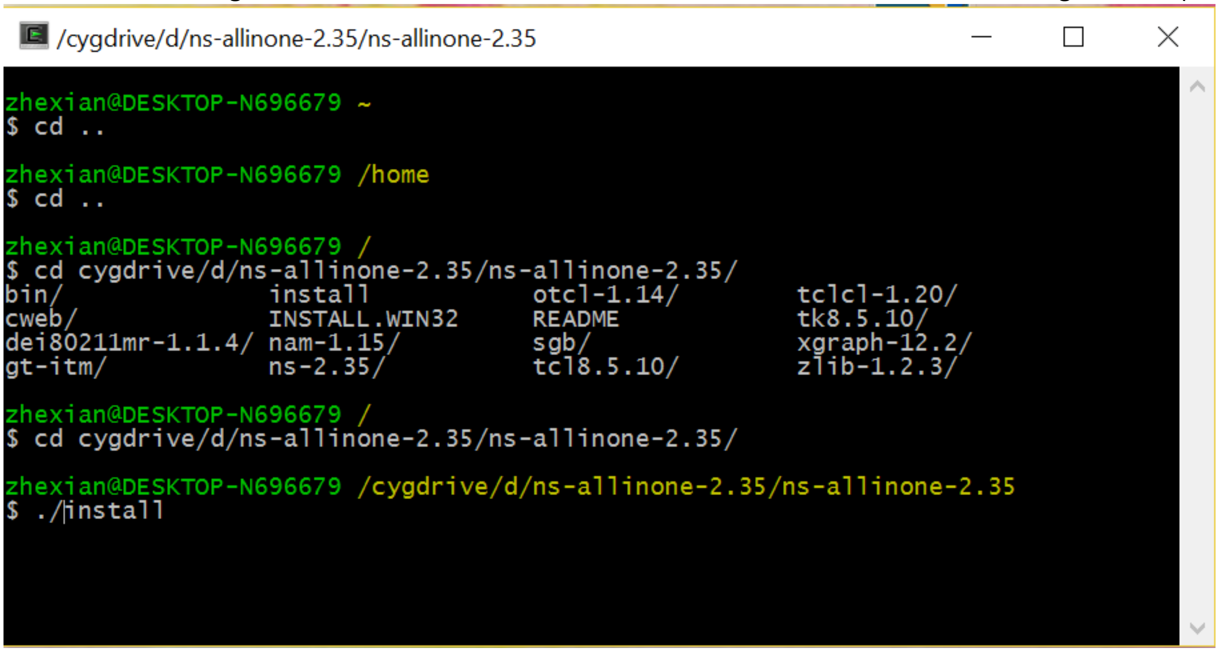
Computer Networks Assignment #5 Zhang Zhexian (0545080) zhangzhexian@outlook.com

1. [Chapter 5 Hands-on 1] Ns-2 is the most popular simulator for TCP research. It includes a package called NAM that can visually replay the whole simulation in all timescales. Many Web sites that introduce ns-2 can be found on the Internet. Use NAM to observe a TCP running from a source to its destination, with and without buffer overflow at one intermediate router.
    - Step 1: Search the ns-2 Web site and download a suitable version for your target platform.
    - Step 2: Follow the installation instructions to install all the packages.
    - Step 3: Build a scenario consisting of three cascaded nodes, one for the Reno TCP source, one for an intermediate gateway, and one for the destination. The links to connect them are full-duplex 1 Mbps.
    - Step 4: Configure the gateway to have a large buffer. Run a TCP source toward the destination.
    - Step 5: Configure the gateway as to have a small buffer. Run a TCP source towards the destination.

    For all the Reno TCP states that the Reno TCP source in the preceding two tests enters, screen dump them and indicate which state the TCP source is in. The figures should be correlated. For example, to represent the slow start behavior you may display it with two figures: (1) an ACK is coming back; (2) the ACK triggers out two new data segments. Carefully organize the figures so that the result of this exercise is no more than one A4 page. Only display necessary information in the screen dumps. Pre-process the figures so that no window decorations (window border, NAM buttons) are displayed.

    Steps I have performed:

    - Downloading the ns-2 NAM from http://www.isi.edu/nsnam/ns/ns-build.html
    - Downloading Cygwin as it is needed for Windows platform
    - Downloading gcc4-g++ packages as it is needed for ns-2 NAM installation
    - Modify the ./install file in ns-2 NAM to allow checkpoint to be passed
    - After countless tries, error still persists:


/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35
zhexian@DESKTOP-N696679 ~
$ cd ..
zhexian@DESKTOP-N696679 /home
$ cd ..
zhexian@DESKTOP-N696679 /
$ cd cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/
bin/ install otcl-1.14/ tclcl-1.20/
cweb/ INSTALL.WIN32 README tk8.5.10/
dei80211mr-1.1.4/ nam-1.15/ sgb/ xgraph-12.2/
gt-itm/ ns-2.35/ tcl8.5.10/ zlib-1.2.3/
zhexian@DESKTOP-N696679 /
$ cd cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/
zhexian@DESKTOP-N696679 /cygdrive/d/ns-allinone-2.35/ns-allinone-2.35
$ ./install

/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35

```
DHAVE_INTPTR_T=1 -DHAVE_UINTPTR_T=1 -DHAVE_SIGNED_CHAR=1 -DHAVE_LANGINFO=1 -DHAV
E_FTS=1 -DHAVE_SYS_IOCTL_H=1 -DTCL_UNLOAD_DLLS=1       /cygdrive/d/ns-allinone-2
.35/ns-allinone-2.35/tcl8.5.10/unix/../libtommath/bn_s_mp_sub.c
rm -f libtclstub8.5.a
ar cr libtclstub8.5.a tclStubLib.o  ; ranlib libtclstub8.5.a
rm -f libtcl8.5.a
ar cr libtcl8.5.a regcomp.o regexec.o regfree.o regerror.o tclAlloc.o tclAsync.o
 tclBasic.o tclBinary.o tclCkalloc.o tclClock.o tclCmdAH.o tclCmdIL.o tclCmdMZ.o
 tclCompCmds.o tclCompExpr.o tclCompile.o tclConfig.o tclDate.o tclDictObj.o tcl
Encoding.o tclEnv.o tclEvent.o tclExecute.o tclFCmd.o tclFileName.o tclGet.o tcl
Hash.o tclHistory.o tclIndexObj.o tclInterp.o tclIO.o tclIOCmd.o tclIORChan.o tc
lIOGT.o tclIOSock.o tclIOUtil.o tclLink.o tclListObj.o tclLiteral.o tclLoad.o tc
lMain.o tclNamesp.o tclNotify.o tclObj.o tclPanic.o tclParse.o tclPathObj.o tclP
ipe.o tclPkg.o tclPkgConfig.o tclPosixStr.o tclPreserve.o tclProc.o tclRegexp.o
tclResolve.o tclResult.o tclScan.o tclStringObj.o tclStrToD.o tclThread.o tclThr
eadAlloc.o tclThreadJoin.o tclThreadStorage.o tclStubInit.o tclStubLib.o tclTime
r.o tclTrace.o tclUtf.o tclUtil.o tclVar.o tclTomMathInterface.o tclUnixChan.o t
clUnixEvent.o tclUnixFCmd.o tclUnixFile.o tclUnixPipe.o tclUnixSock.o tclUnixTim
e.o tclUnixInit.o tclUnixThrd.o tclUnixCompat.o tclUnixNotfy.o  tclLoadDl.o  bnc
ore.o bn_reverse.o bn_fast_s_mp_mul_digs.o bn_fast_s_mp_sqr.o bn_mp_add.o bn_mp_
and.o bn_mp_add_d.o bn_mp_clamp.o bn_mp_clear.o bn_mp_clear_multi.o bn_mp_cmp.o
bn_mp_cmp_d.o bn_mp_cmp_mag.o bn_mp_copy.o bn_mp_count_bits.o bn_mp_div.o bn_mp_
div_d.o bn_mp_div_2.o bn_mp_div_2d.o bn_mp_div_3.o bn_mp_exch.o bn_mp_expt_d.o b
n_mp_grow.o bn_mp_init.o bn_mp_init_copy.o bn_mp_init_multi.o bn_mp_init_set.o b
n_mp_init_set_int.o bn_mp_init_size.o bn_mp_karatsuba_mul.o bn_mp_karatsuba_sqr.
o bn_mp_lshd.o bn_mp_mod.o bn_mp_mod_2d.o bn_mp_mul.o bn_mp_mul_2.o bn_mp_mul_2d
.o bn_mp_mul_d.o bn_mp_neg.o bn_mp_or.o bn_mp_radix_size.o bn_mp_radix_smap.o bn
_mp_read_radix.o bn_mp_rshd.o bn_mp_set.o bn_mp_set_int.o bn_mp_shrink.o bn_mp_s
qr.o bn_mp_sqrt.o bn_mp_sub.o bn_mp_sub_d.o bn_mp_to_unsigned_bin.o bn_mp_to_uns
igned_bin_n.o bn_mp_toom_mul.o bn_mp_toom_sqr.o bn_mp_toradix_n.o bn_mp_unsigned
_bin_size.o bn_mp_xor.o bn_mp_zero.o bn_s_mp_add.o bn_s_mp_mul_digs.o bn_s_mp_sq
r.o bn_s_mp_sub.o  ; ranlib libtcl8.5.a
gcc -c -O  -pipe  -Wall  -I"." -I/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/t
cl8.5.10/unix/../unix -I/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/tcl8.5.10/
unix/../generic -I/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/tcl8.5.10/unix/.
./libtommath -DPACKAGE_NAME=\"tcl\" -DPACKAGE_TARNAME=\"tcl\" -DPACKAGE_VERSION=
\"8.5\" -DPACKAGE_STRING=\"tcl\ 8.5\" -DPACKAGE_BUGREPORT=\"\" -DSTDC_HEADERS=1
-DHAVE_SYS_TYPES_H=1 -DHAVE_SYS_STAT_H=1 -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DH
AVE_MEMORY_H=1 -DHAVE_STRINGS_H=1 -DHAVE_INTTYPES_H=1 -DHAVE_STDINT_H=1 -DHAVE_U
NISTD_H=1 -DNO_VALUES_H=1 -DHAVE_LIMITS_H=1 -DHAVE_SYS_PARAM_H=1 -DTCL_CFGVAL_EN
CODING=\"iso8859-1\" -DSTATIC_BUILD=1 -DTCL_SHLIB_EXT=\".dll\" -DTCL_CFG_OPTIMIZ
ED=1 -DTCL_CFG_DEBUG=1 -DTCL_TOMMATH=1 -DMP_PREC=4 -DTCL_WIDE_INT_IS_LONG=1 -DHA
VE_GETCWD=1 -DHAVE_OPENDIR=1 -DHAVE_STRTOL=1 -DHAVE_WAITPID=1 -DHAVE_GETADDRINFO
=1 -DUSE_TERMIOS=1 -DHAVE_SYS_TIME_H=1 -DTIME_WITH_SYS_TIME=1 -DHAVE_GMTIME_R=1
-DHAVE_LOCALTIME_R=1 -DHAVE_MKTIME=1 -DHAVE_TM_GMTOFF=1 -DHAVE_TIMEZONE_VAR=1 -D
HAVE_STRUCT_STAT_ST_BLOCKS=1 -DHAVE_STRUCT_STAT_ST_BLKSIZE=1 -DHAVE_BLKCNT_T=1 -
DHAVE_INTPTR_T=1 -DHAVE_UINTPTR_T=1 -DHAVE_SIGNED_CHAR=1 -DHAVE_LANGINFO=1 -DHAV
E_FTS=1 -DHAVE_SYS_IOCTL_H=1 -DTCL_UNLOAD_DLLS=1        /cygdrive/d/ns-allinone-
2.35/ns-allinone-2.35/tcl8.5.10/unix/../unix/tclAppInit.c
gcc -O  -pipe     tclAppInit.o -L/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/t
cl8.5.10/unix -ltcl8.5 -ldl     \
          -o tclsh.exe
/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/tcl8.5.10/unix/libtcl8.5.a(tclEnv.
o):tclEnv.c:(.text+0x110): undefined reference to `cygwin_posix_to_win32_path_li
st_buf_size'
/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/tcl8.5.10/unix/libtcl8.5.a(tclEnv.
o):tclEnv.c:(.text+0x110): relocation truncated to fit: R_X86_64_PC32 against un
defined symbol `cygwin_posix_to_win32_path_list_buf_size'
/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/tcl8.5.10/unix/libtcl8.5.a(tclEnv.
o):tclEnv.c:(.text+0x132): undefined reference to `cygwin_posix_to_win32_path_li
st'
/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/tcl8.5.10/unix/libtcl8.5.a(tclEnv.
o):tclEnv.c:(.text+0x132): relocation truncated to fit: R_X86_64_PC32 against un
defined symbol `cygwin_posix_to_win32_path_list'
/cygdrive/d/ns-allinone-2.35/ns-allinone-2.35/tcl8.5.10/unix/libtcl8.5.a(tclEnv.
o):tclEnv.c:(.rdata$.refptr.__cygwin_environ[.refptr.__cygwin_environ]+0x0): und
efined reference to `__cygwin_environ'
collect2: error: ld returned 1 exit status
make: *** [Makefile:569: tclsh.exe] Error 1
tcl8.5.10 make failed! Exiting ...
For problems with Tcl/Tk see http://www.scriptics.com

zhexian@DESKTOP-N696679 /cygdrive/d/ns-allinone-2.35/ns-allinone-2.35
```

2. [Chapter 5 Hands-on 4] Linux Packet Socket is a useful tool when you want to generate arbitrary types of packets. Find and modify an example program to generate a packet and sniff the packet with the same program.

```
/*
    Raw TCP packets
*/
#include<stdio.h> //for printf
#include<string.h> //memset
#include<sys/socket.h>    //for socket ofcourse
#include<stdlib.h> //for exit(0);
#include<errno.h> //For errno - the error number
#include<netinet/tcp.h>   //Provides declarations for tcp header
#include<netinet/ip.h>    //Provides declarations for ip header

/*
    96 bit (12 bytes) pseudo header needed for tcp header checksum calculation
*/
struct pseudo_header
{
    u_int32_t source_address;
    u_int32_t dest_address;
    u_int8_t placeholder;
    u_int8_t protocol;
    u_int16_t tcp_length;
};

/*
    Generic checksum calculation function
*/
unsigned short csum(unsigned short *ptr,int nbytes)
{
    register long sum;
    unsigned short oddbyte;
    register short answer;

    sum=0;
    while(nbytes>1) {
        sum+=*ptr++;
        nbytes-=2;
    }
    if(nbytes==1) {
        oddbyte=0;
        *((u_char*)&oddbyte)=*(u_char*)ptr;
        sum+=oddbyte;
    }

    sum = (sum>>16)+(sum & 0xffff);
    sum = sum + (sum>>16);
    answer=(short)~sum;

    return(answer);
}

int main (void)
{
```

```
    //Create a raw socket
    int s = socket (PF_INET, SOCK_RAW, IPPROTO_TCP);

    if(s == -1)
    {
        //socket creation failed, may be because of non-root privileges
        perror("Failed to create socket");
        exit(1);
    }

    //Datagram to represent the packet
    char datagram[4096] , source_ip[32] , *data , *pseudogram;

    //zero out the packet buffer
    memset (datagram, 0, 4096);

    //IP header
    struct iphdr *iph = (struct iphdr *) datagram;

    //TCP header
    struct tcphdr *tcph = (struct tcphdr *) (datagram + sizeof (struct ip));
    struct sockaddr_in sin;
    struct pseudo_header psh;

    //Data part
    data = datagram + sizeof(struct iphdr) + sizeof(struct tcphdr);
    strcpy(data , "ABCDEFGHIJKLMNOPQRSTUVWXYZ");

    //some address resolution
    strcpy(source_ip , "192.168.1.2");
    sin.sin_family = AF_INET;
    sin.sin_port = htons(80);
    sin.sin_addr.s_addr = inet_addr ("1.2.3.4");

    //Fill in the IP Header
    iph->ihl = 5;
    iph->version = 4;
    iph->tos = 0;
    iph->tot_len = sizeof (struct iphdr) + sizeof (struct tcphdr) + strlen(data);
    iph->id = htonl (54321); //Id of this packet
    iph->frag_off = 0;
    iph->ttl = 255;
    iph->protocol = IPPROTO_TCP;
    iph->check = 0;      //Set to 0 before calculating checksum
    iph->saddr = inet_addr ( source_ip );    //Spoof the source ip address
    iph->daddr = sin.sin_addr.s_addr;

    //Ip checksum
    iph->check = csum ((unsigned short *) datagram, iph->tot_len);

    //TCP Header
    tcph->source = htons (1234);
    tcph->dest = htons (80);
    tcph->seq = 0;
    tcph->ack_seq = 0;
    tcph->doff = 5;  //tcp header size
    tcph->fin=0;
    tcph->syn=1;
    tcph->rst=0;
```

```
    tcph->psh=0;
    tcph->ack=0;
    tcph->urg=0;
    tcph->window = htons (5840); /* maximum allowed window size */
    tcph->check = 0; //leave checksum 0 now, filled later by pseudo header
    tcph->urg_ptr = 0;

    //Now the TCP checksum
    psh.source_address = inet_addr( source_ip );
    psh.dest_address = sin.sin_addr.s_addr;
    psh.placeholder = 0;
    psh.protocol = IPPROTO_TCP;
    psh.tcp_length = htons(sizeof(struct tcphdr) + strlen(data) );

    int psize = sizeof(struct pseudo_header) + sizeof(struct tcphdr) + strlen(data);
    pseudogram = malloc(psize);

    memcpy(pseudogram , (char*) &psh , sizeof (struct pseudo_header));
    memcpy(pseudogram + sizeof(struct pseudo_header) , tcph , sizeof(struct tcphdr) +
strlen(data));

    tcph->check = csum( (unsigned short*) pseudogram , psize);

    //IP_HDRINCL to tell the kernel that headers are included in the packet
    int one = 1;
    const int *val = &one;

    if (setsockopt (s, IPPROTO_IP, IP_HDRINCL, val, sizeof (one)) < 0)
    {
        perror("Error setting IP_HDRINCL");
        exit(0);
    }

    //loop if you want to flood :)
    while (1)
    {
        //Send the packet
        if (sendto (s, datagram, iph->tot_len ,  0, (struct sockaddr *) &sin, sizeof
(sin)) < 0)
        {
            perror("sendto failed");
        }
        //Data send successfully
        else
        {
            printf ("Packet Send. Length : %d \n" , iph->tot_len);
        }
    }

    return 0;
}

//Complete
```

Compile by program by doing a gcc raw_socket.c at the terminal.

Use a packet sniffer like wireshark to check the output and verify that the packets have actually been generated and send over the network.

3. [Chapter 5 Hands-on 8] What transport protocols are used in MS Media Player or RealMedia? Please use wireshark to observe and find out the answer.

The wireshark capture I obtained is from http://www.pcapr.net/, and here is the exact URL: http://www.pcapr.net/view/bwilkerson/2008/10/3/14/Viewpoint_Media_Player_for_IE_3.2_Remote_Stack_Overflow_PoC.pcap.html

1.	192.168.0.22	»	192.168.0.15	tcp	3712 > 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
2.	192.168.0.15	»	192.168.0.22	tcp	80 > 3712 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3.	192.168.0.22	»	192.168.0.15	tcp	3712 > 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
4.	192.168.0.22	»	192.168.0.15	http	GET /spoil/4610.htm HTTP/1.1
5.	192.168.0.15	»	192.168.0.22	tcp	[TCP segment of a reassembled PDU]

As shown in the capture above, the transport protocol used in media player is TCP.

4. [Chapter 5 Written 4] A mobile TCP receiver is receiving data from its TCP sender. What will the RTT and the RTO evolve when the receiver gets farther away and then nearer? Assume the moving speed is very fast so that the propagation delay ranges from 100ms to 300ms within 1 second.

When the receiver gets farther away, the RTT will increase (from 200ms minimum to 600ms maximum), as RTT is proportional to propagation delay which is affected by physical distance. The RTO will increase as well to prevent wrongly recovering packets that are not lost, but just take a long time to transmit.

Similarly, as the receiver gets nearer, both RTT and RTO will become smaller.

Since RTT is affected in real time as the physical distance changes, but RTO needs to be adjusted by algorithm, the RTT will change at a faster rate than RTO.

5. [Chapter 5 Written 6] Given that the throughput of a TCP connection is inversely proportional to its RTT, connections with heterogeneous RTTs sharing the same queue will get different bandwidth shares. What will be the eventual proportion of the bandwidth sharing among three connections if their propagation delays are 10ms, 100ms, and 150ms, and the service rate of the shared queue is 200 kbps? Assume that the queue size is infinite without buffer overflow (no packet loss), and the maximum window of the TCP sender is 20 packets, with each packet having 1500 bytes.

Since the throughput of a TCP connection is inversely proportional to its RTT, and the RTT is proportional to their propagation delay, the eventual bandwidth sharing is inversely proportional to their RTT too.

With delay of 150 ms, a sender transmits 1 packet every 300 ms (RTT is twice of delay); the sender with 100ms delay transmits 1.5 packets every 300 ms. A sender with 10ms delay transmits 15 packets every 300ms.

Total packets = 1 + 1.5 + 15 = 17.5;

150ms sender: 1/17.5 = 6%;
100ms sender: 1.5/17.5 = 8%;
10ms sender: 15/17.5 = 86%.

6. [Chapter 5 Written 8] If the smoothed RTT kept by the TCP sender is currently 30ms and the following measured RTTs are 26, 32, and 24ms, respectively, what is the new RTT estimate?

   By using dynamic RTT estimator, which adopts the exponential weighted moving average (EWMA), which takes 1/8 of the new RTT measure plus 7/8 of the old smoothed RTT value to form the new estimate of the RTT.

   (1/8)*26+(7/8)*30 = 29.5
   (1/8)*32+(7/8)* 29.5 = 29.813
   (1/8)*24+(7/8)* 29.813 = 29.086

   Thus, the new RTT estimate is 29.086ms.

7. [Chapter 5 Written 13] Suppose you are going to design a real-time streaming application over the Internet that employs RTP on top of TCP instead of UDP. What situations will the sender and the receiver encounter in each TCP congestion control state shown in Figure 5.21? Compare your expected situations with those designed on top of UDP in a table format.

State	RTP over TCP	RTP over UDP
Slow start	Slow start aims at quickly probing available bandwidth within a few rounds of RTTs. Since three-way handshake is required in TCP, it will take a longer time to complete the probing. Once bandwidth probing is done, several bandwidths can be utilized simultaneously for efficient RTP transmission.	UDP does not require connection between sender and receiver, so the proving time will be shorter. However, since no ACK is sent from receiver to sender, cwnd is always set to 1. Due to the same reason, bandwidth sharing is not possible in RTP over UDP so the transmission may not be as efficient.
Fast retransmit	Fast retransmit targets transmitting the lost packet immediately without waiting for the retransmission timer to expire. Triple duplicate ACK will trigger retransmission, thus resulting in faster resending of loss packet. The packet loss rate will be lower resulting in better quality streaming. Nevertheless, the time taken for retransmission affects streaming speed.	RTP itself does not address resource management and reservation and does not guarantee quality-of-service for real-time services. RTP assumes that these properties, if available, are provided by the underlying network. However, such retransmission mechanism is missing in UDP, so quality-of-service may be severely affected in RTP over UDP.

Congestion avoidance	Congestion avoidance aims at slowly probing available bandwidth but rapidly responding to congestion events Once congestion is detected, the cwnd will be quickly reset to 1 to trigger the fast-transmit state, and the real time streaming speed will suddenly be much slower.	No formal congestion control mechanism is implemented. Thus, in times of network congestion, RTP packets may be dropped, affecting streaming quality. However, since no cwnd size change, the streaming speed will not fluctuate much during streaming.
Retransmission timeout	Retransmission timeout provides the last and slowest resort to retransmit the lost packet. It is implemented to TCP to ensure a higher quality-of-service for RTP. Since the timer is designed with RTT, it minimizes the possibility of resending already-transmitted packets that wastes network resources.	No retransmission timer is maintained in UDP sender, because no acknowledgement will be sent from the receiver.
Fast recovery	After a packet loss, fast recovery state preserves enough outstanding packets in the network pipe to retain TCP's self-clocking behavior. It allows more packets to be sent even when packet loss is detected, thus increasing recovery speed for RTP.	RTP itself does not address resource management and reservation, as such it does not retransmit lost packets.

8. [Chapter 5 Written 16] The text goes into some detail introducing the different versions of TCP. Find three more TCP versions. Itemize them and highlight their contributions within three lines of words for each TCP version.

    a. TCP Hybla: it aims to eliminate penalization of TCP connections that incorporate a high-latency terrestrial or satellite radio link, due to their longer round-trip times. It implements the necessary modifications to remove the performance dependence on RTT.
    b. TCP Westwood+: it is a sender-side only modification of the TCP Reno protocol stack that optimizes the performance of TCP congestion control over both wireline and wireless networks. It significantly increases throughput over wireless links and fairness compared to TCP Reno/New Reno in wired networks.
    c. Compound TCP: it is a Microsoft implementation of TCP which maintains two different congestion windows simultaneously, with the goal of achieving good performance on LFNs while not impairing fairness.