CS425: Computer Networks
Project-3: STCP Layer

Shubham Agrawal
13674, agshubh191@gmail.com

September 30, 2016

1. Design Choices

- As size of entire packet and size of payload both are mentioned to be 536 bytes in Project Spec, I used Maximum Payload Size as 536 - 20(size of Header).
- Maximum Size of packet is mentioned as 536 bytes for STCP in Project Report but in TCP, maximum segment size is much larger than this. So to make STCP compatible with TCP, I used maximum segment size as 3072(maximum of STCP windows).
- Network layer is assumed to be perfect so following things are not handled
  - Re-transmissions
  - Timeouts
  - Amount of data sent by stcp_network_send is not cross-checked .
- To handle states and various state variables, I've defined following variables in *ctx context* structure:
  - Next seq number to use (this -1 bytes are already sent)
  - Latest acknowledgement number received
  - Latest advertised window size
  - Latest acknowledgement number sent (Will tell us what sequence number we are expecting)
- I've added *127.0.0.1 Server name* to */etc/hosts* file to handle assertion failure in *stcp_api.c* due to incorrect check-sum

2. Testing Results

For testing, I've checked that various types of files (c, text, binary, pdf) are sent exactly. Following are the results in different scenarios:

- Both client and server on same laptop: All files were sent correctly
- Different student's client program, mine server program and vice versa over network: All files were sent.

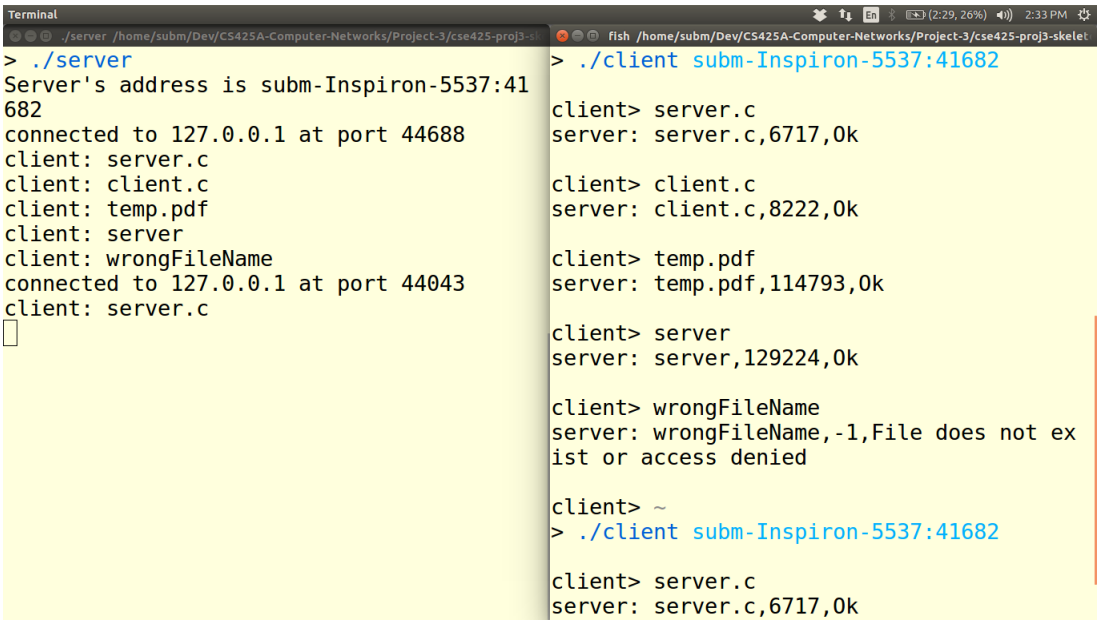

Figure 1: All types of file are exchanged properly

3. Summary

- All the mentioned features seems to work

4. Appendix

4.1 Code *transport.c*

```
/*
 * transport.c
 *
 *      Project 3
 *
 * This file implements the STCP layer that sits between the
 * mysocket and network layers. You are required to fill in the STCP
 * functionality in this file.
 *
 */


#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <arpa/inet.h>
#include "mysock.h"
#include "stcp_api.h"
#include "transport.h"
#include "network_io.h"
#include <limits.h>


enum {
        CSTATE_ESTABLISHED,
};    /* you should have more states */


/* this structure is global to a mysocket descriptor */
typedef struct
{
    bool_t done;    /* TRUE once connection is closed */
```

```
    int connection_state;   /* state of the connection (established, etc.) */

    /* any other connection-wide global variables go here */

    tcp_seq initial_sequence_num;
        tcp_seq next_seq; // Next seq number to use (this -1 bytes are already sent
        tcp_seq ack_seq;  // Latest ack received
        int     ad_win;           // Latest advertised window size
        int y_ack_seq;    // Latest ack number sent (Will tell us what seq number w

        int fin;                  // Fin state
} context_t;


static void generate_initial_seq_num(context_t *ctx);
static void control_loop(mysocket_t sd, context_t *ctx);

void our_dprintf(const char *format,...);

/* initialise the transport layer, and start the main loop, handling
 * any data from the peer or the application.  this function should not
 * return until the connection is closed.
 */
void transport_init(mysocket_t sd, bool_t is_active)
{
    context_t *ctx;

    ctx = (context_t *) calloc(1, sizeof(context_t));
    assert(ctx);

    generate_initial_seq_num(ctx);
        ctx->next_seq = ctx->initial_sequence_num;

        //ctx->rem_win_size = TH_Initial_Win; // Set Initial window size

        ctx->fin = 0; // 0 fin exchanged

    /* XXX: you should send a SYN packet here if is_active, or wait for one
     * to arrive if !is_active.  after the handshake completes, unblock the
     * application with stcp_unblock_application(sd).  you may also use
     * this to communicate an error condition back to the application, e.g.
     * if connection fails; to do so, just set errno appropriately (e.g. to
     * ECONNREFUSED, etc.) before calling the function.
```

```
*/
    char buffer[maxBufferSize];
    bzero(buffer, maxBufferSize);
    if(is_active){
            // create a tcp syn packet
            our_dprintf("ACTIVE: Initiating Handshake\n");
            our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx->ack_seq, ctx->a
            STCPHeader* initHeader = (STCPHeader*) malloc (sizeof(STCPHeader));
            initHeader->th_seq          = htonl(ctx->next_seq);
            initHeader->th_ack          = htonl(0); // This does not matter as o
            initHeader->th_flags    = TH_SYN;
            initHeader->th_off      = 5;
            initHeader->th_win      = htons(TH_Initial_Win);
            ctx->next_seq++; //First data byte is starting from isn+1

            // send packet
            stcp_network_send(sd, initHeader, sizeof(STCPHeader), NULL );
            our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx->ack_seq, ctx->a
            our_dprintf("SYN packet sent, waiting for synAck to arrive\n");

            // wait for syn-ack to arrive
            int flag = 0;
            while(!flag){
                    stcp_wait_for_event(sd, NETWORK_DATA, NULL);

                    // Read headers
                    // Set initial seq number of sender in context
                    stcp_network_recv(sd, buffer, maxBufferSize);
                    STCPHeader* synAckHeader = (STCPHeader*)buffer;
                    if(synAckHeader->th_flags != (TH_SYN|TH_ACK))
                            continue;
                    flag = 1;       // SYN-ACK arrived. Get out of loop
                    our_dprintf("SYN-ACK arrived\n");
                    ctx->ack_seq = ntohl(synAckHeader->th_ack);
                    ctx->ad_win = ntohs(synAckHeader->th_win);
                    int y_seq_number = ntohl(synAckHeader->th_seq);
                    our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx->ack_seq

                    // #send ack back
                    // ##create a tcp ack packet
                    our_dprintf("Sending ack packet\n");
                    STCPHeader* ackHeader = (STCPHeader*) malloc(sizeof(STCPHea
                    ackHeader->th_seq       = htonl(ctx->next_seq);
```

```
                ackHeader->th_ack       = htonl(y_seq_number+1);
                ackHeader->th_flags = TH_ACK;
                ackHeader->th_off   = 5;
                ackHeader->th_win   = htons(TH_Initial_Win);
                ctx->y_ack_seq = y_seq_number+1;

                // ##send packet
                stcp_network_send(sd, ackHeader, sizeof(STCPHeader), NULL);
                our_dprintf("Ack Sent\n");
                our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx->ack_seq
        }
}else{
        // wait for syn packet to arrive
        our_dprintf("PASSIVE\n");
        our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx->ack_seq, ctx->a
        our_dprintf("Waiting for Syn to arrive\n");
        int flag = 0;
        while(!flag){

                // Read headers
                // Set initial seq number of sender in context
                stcp_wait_for_event(sd, NETWORK_DATA, NULL );
                stcp_network_recv(sd, buffer, maxBufferSize);

                STCPHeader* synHeader = (STCPHeader*)buffer;
                our_dprintf("SYN Packet arrived\n");
                if(synHeader->th_flags != TH_SYN)
                        continue;
                flag=1;
                int y_seq_number = ntohl(synHeader->th_seq);
                our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx->ack_seq

                // #send syn-ack back
                // ##create a tcp syn-ack packet
                our_dprintf("SENDING SYN ACK\n");
                STCPHeader* synAckHeader = (STCPHeader*) malloc(sizeof(STCP
                synAckHeader->th_seq     = htonl(ctx->next_seq);
                synAckHeader->th_ack     = ntohl(y_seq_number+1);
                synAckHeader->th_flags   = TH_SYN|TH_ACK;
                synAckHeader->th_off     = 5;
                synAckHeader->th_win     = ntohs(TH_Initial_Win);
                ctx->next_seq++;
                ctx->y_ack_seq = y_seq_number+1;
```

```
                        our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx->ack_seq

                        // ##send synack packet
                        stcp_network_send(sd, synAckHeader, sizeof(STCPHeader), NUL

                        // #receive ack packet
                        our_dprintf("WAITING FOR ACK PACKET TO ARRIVE\n");
                        int flag2 = 0;
                        while(!flag2){
                                stcp_wait_for_event(sd, NETWORK_DATA, NULL );
                                our_dprintf("some data\n");

                                // Read headers
                                bzero(buffer, sizeof(buffer));
                                stcp_network_recv(sd, buffer, maxBufferSize);
                                STCPHeader* ackHeader = (STCPHeader*)buffer;
                                if(ackHeader->th_flags != TH_ACK)
                                        continue;
                                flag2 = 1;
                                our_dprintf("ACK ARRIVED\n");
                                ctx->ack_seq = ntohl(ackHeader->th_ack);
                                ctx->ad_win = ntohs(ackHeader->th_win);
                                our_dprintf("%lu %lu %d %lu\n", ctx->next_seq, ctx-
                        }
                }
        }
    ctx->connection_state = CSTATE_ESTABLISHED;
        our_dprintf("%ld %ld %d %ld\n", ctx->next_seq, ctx->ack_seq, ctx->ad_win, c
    stcp_unblock_application(sd);

    control_loop(sd, ctx);

    /* do any cleanup here */
    free(ctx);
}


/* generate random initial sequence number for an STCP connection */
static void generate_initial_seq_num(context_t *ctx)
{
    assert(ctx);

#ifdef FIXED_INITNUM
```

```
    /* please don't change this! */
    ctx->initial_sequence_num = 1;
#else
    /* you have to fill this up */
    /*ctx->initial_sequence_num =;*/
#endif
}


/* control_loop() is the main STCP loop; it repeatedly waits for one of the
 * following to happen:
 *   - incoming data from the peer
 *   - new data from the application (via mywrite())
 *   - the socket to be closed (via myclose())
 *   - a timeout
 */

static void control_loop(mysocket_t sd, context_t *ctx)
{
    assert(ctx);
    assert(!ctx->done);

        char buffer[maxBufferSize];
        unsigned int desired_event=NETWORK_DATA|APP_DATA|APP_CLOSE_REQUESTED;
    while (!ctx->done)
    {
        unsigned int event;

        /* see stcp_api.h or stcp_api.c for details of this function */
        /* XXX: you will need to change some of these arguments! */
                our_dprintf("waiting for some event \n");
        event = stcp_wait_for_event(sd, desired_event, NULL);

        /* check whether it was the network, app, or a close request */
                bzero(buffer, maxBufferSize);
        if (event & APP_DATA)
        {
            /* the application has requested that data be sent */
            /* see stcp_app_recv() */
                        //Get data from application
                        //Prepare Header for data to be sent
                        //Check available window. If data to be sent > window size
                        //send data
```

```
                STCPHeader* dataPacket = (STCPHeader*) malloc(sizeof(STCPHe
                dataPacket->th_flags = 0; // Payload packet, no flags
                dataPacket->th_ack      = htonl(ctx->y_ack_seq); // No ackn
                dataPacket->th_seq      = htonl(ctx->next_seq);
                dataPacket->th_off = 5;
                dataPacket->th_win = htons(TH_Initial_Win);
                int appData = stcp_app_recv(sd, buffer, MIN(maxPayloadSize,
                our_dprintf("Data from app received, Size-%d\n------------\
                if(appData>0){
                        int sent = stcp_network_send(sd, dataPacket, sizeof
                        assert(sent == appData + (int)sizeof(STCPHeader));
                        our_dprintf("Data %d sent to network\n", sent);
                        ctx->next_seq = ctx->next_seq + appData;
                }
}
        if(event & NETWORK_DATA){
                int networkData = stcp_network_recv(sd, buffer, maxBufferSi
                our_dprintf("Data(%d) from network received\n", networkData
                STCPHeader* packet = (STCPHeader *)buffer;
                if(packet->th_flags == TH_ACK){
                        if(ctx->fin > 0){
                                // If already fin received or sent (either
                                ctx->fin++;
                                ctx->ack_seq = MAX(ctx->ack_seq, ntohl(pack
                                if(ctx->fin >2){
                                        ctx->done = TRUE;
                                        break;
                                }
                                continue;
                        }
                        ctx->ad_win = ntohs(packet->th_win);
                        ctx->ack_seq = MAX(ntohl(packet->th_ack), ctx->ack_
                        our_dprintf("\nACK Received: th_win = %d, th_ack = 
                        our_dprintf(">>> %lu %lu", ctx->next_seq, ctx->ack_
                        assert(ctx->next_seq >= ctx->ack_seq );
                }else if(packet->th_flags == TH_FIN){
                        our_dprintf("Fin Packet Received\n");
                        ctx->fin++;
                        // send FIN-ACK
                        STCPHeader* finAckPacket = (STCPHeader*)malloc(size
                        finAckPacket->th_flags    = TH_ACK;
                        finAckPacket->th_ack      = htonl(ctx->y_ack_seq);
                        finAckPacket->th_seq      = htonl(ctx->next_seq);
```

```
                finAckPacket->th_off      = 5;
                finAckPacket->th_win      = htons(TH_Initial_Win);
                stcp_network_send(sd, finAckPacket, sizeof(STCPHead
                our_dprintf("Fin-Ack Packet Sent\n");
                stcp_fin_received(sd);
                if(ctx->fin >2){
                        ctx->done = TRUE;
                        break;
                }
                continue;
        }
        int offSet = packet->th_off * 4;
        int payloadSize = networkData - offSet;
        int early = 0;
        // Send Acknowledgement
        if(payloadSize > 0 || (payloadSize==0 && packet->th_flags !=
                if(ctx->y_ack_seq > (tcp_seq)ntohl(packet->th_seq))
                        int early = ctx->y_ack_seq - ntohl(packet->
                        our_dprintf("Already acknowledged data is r
                }
                if(payloadSize - early > 0){
                        stcp_app_send(sd, buffer+offSet+early, payl
                        our_dprintf("payload data(%d) sent to app \
                }
                // send acknowledgement of this data for other side
                STCPHeader* ackPacket = (STCPHeader*)malloc(sizeof(
                ackPacket->th_flags = TH_ACK;
                ackPacket->th_seq       = htonl(ctx->next_seq);
                ackPacket->th_ack       = htonl(MAX(ntohl(packet->t
                ackPacket->th_off = 5;
                ackPacket->th_win = htons(TH_Initial_Win);
                our_dprintf("Ack sent %d, %d\n", MAX(ntohl(packet->
                ctx->y_ack_seq =  ntohl(ackPacket->th_ack);
                stcp_network_send(sd, ackPacket, sizeof(STCPHeader)
        }
}
if(event & APP_CLOSE_REQUESTED){
        our_dprintf("App close demanded\n");
        // send FIN
        STCPHeader* finPacket = (STCPHeader*)malloc(sizeof(STCPHead
        finPacket->th_flags = TH_FIN;
        finPacket->th_ack       = htonl(ctx->y_ack_seq);
        finPacket->th_seq       = htonl(ctx->next_seq);
```

```
                        finPacket->th_off = 5;
                        finPacket->th_win = htons(TH_Initial_Win);
                        ctx->next_seq++;
                        ctx->fin++;
                        stcp_network_send(sd, finPacket, sizeof(STCPHeader), NULL);
                        our_dprintf("Fin Packet Sent\n");
                }
                our_dprintf("+++++++++++++++++++++One Event Completed+++++++++++++++
                desired_event=NETWORK_DATA|APP_DATA|APP_CLOSE_REQUESTED;
        /* etc. */
    }
        our_dprintf("Connection Closed\n");
}


/*********************************************************************/
/* our_dprintf
 *
 * Send a formatted message to stdout.
 *
 * format               A printf-style format string.
 *
 * This function is equivalent to a printf, but may be
 * changed to log errors to a file if desired.
 *
 * Calls to this function are generated by the dprintf amd
 * dperror macros in transport.h
 */
void our_dprintf(const char *format,...)
{
    //va_list argptr;
    //char buffer[1024];

    //assert(format);
    //va_start(argptr, format);
    //vsnprintf(buffer, sizeof(buffer), format, argptr);
    //va_end(argptr);
    //fputs(buffer, stdout);
    //fflush(stdout);
}
```

4.2 Code *transport.h*

```
/*HEADER.H*/
/* header file for the transport layer */

#ifndef __TRANSPORT_H__
#define __TRANSPORT_H__

#ifdef _NETINET_TCP_H
    #error <netinet/tcp.h> conflicts with STCP definitions.
    #error Include only transport.h in the STCP project.
#endif

#include <stdio.h>  /* for perror */
#include <errno.h>
#include "mysock.h"


/* For some reason, Linux redefines tcphdr unless one compiles with only
 * _BSD_SOURCE defined--but doing this causes problems with some of the
 * other system headers, which require other conflicting defines (such as
 * _POSIX_SOURCE and _XOPEN_SOURCE).  For simplicity, since the TCP header
 * format is well-defined, we just define this again here.
 *
 * You can ignore the following fields in tcphdr:  th_sport, th_dport,
 * th_sum, th_urp.  stcp_network_send() will take care of filling those
 * in.
 */

/* XXX: ugh, clean this up some time */
#if defined(SOLARIS)
    #define __LITTLE_ENDIAN 1234
    #define __BIG_ENDIAN 4321
    #define __BYTE_ORDER __BIG_ENDIAN
#elif defined(LINUX)    /* cppp replaced: elif */
    #ifndef __BYTE_ORDER
        #error huh?  Linux has not defined endianness.
    #endif
#else
    #error Unrecognised system type.
#endif
```

```
typedef uint32_t tcp_seq;

typedef struct tcphdr
{
    uint16_t th_sport;  /* source port */
    uint16_t th_dport;  /* destination port */
    tcp_seq  th_seq;    /* sequence number */
    tcp_seq  th_ack;    /* acknowledgement number */
#if __BYTE_ORDER == __LITTLE_ENDIAN
    uint8_t  th_x2:4;   /* unused */
    uint8_t  th_off:4;  /* data offset */
#elif __BYTE_ORDER == __BIG_ENDIAN
    uint8_t  th_off:4;  /* data offset */
    uint8_t  th_x2:4;   /* unused */
#else
#error __BYTE_ORDER must be defined as __LITTLE_ENDIAN or __BIG_ENDIAN!
#endif
    uint8_t  th_flags;
#define TH_FIN  0x01
#define TH_SYN  0x02
#define TH_RST  0x04    /* you don't have to handle this */
#define TH_PUSH 0x08    /* ...or this */
#define TH_ACK  0x10
#define TH_URG  0x20    /* ...or this */
    uint16_t th_win;    /* window */
#define TH_Initial_Win 3072
    uint16_t th_sum;    /* checksum */
    uint16_t th_urp;    /* urgent pointer (unused in STCP) */
} __attribute__ ((packed)) STCPHeader;


/* starting byte position of data in TCP packet p */
#define TCP_DATA_START(p) (((STCPHeader *) p)->th_off * sizeof(uint32_t))

/* length of options (in bytes) in TCP packet p */
#define TCP_OPTIONS_LEN(p) (TCP_DATA_START(p) - sizeof(struct tcphdr))

/* STCP maximum segment size */
#define STCP_MSS 536


#ifndef MIN
    #define MIN(x,y)  ((x) <= (y) ? (x) : (y))
```

```
#endif

#ifndef MAX
    #define MAX(x,y)  ((x) >= (y) ? (x) : (y))
#endif


#ifdef DEBUG
    #ifdef LINUX
        #include <string.h> /* Linux, for strerror_r() */
    #else
        extern char *sys_errlist[];
        #define strerror_r(num,buf,len) strncpy(buf, sys_errlist[num], len)
    #endif

    extern void our_dprintf(const char *format, ...);

    #define dprintf our_dprintf
    #define dperror(head) \
        { \
            if (errno >= 0) \
            { \
                char err_buf[255]; \
                dprintf("%s: %s\n", (head), \
                        strerror_r(errno, err_buf, sizeof(err_buf))); \
            } \
        }
#else
    #ifdef __GNUC__
        #define dprintf(fmt, ...)
        #define dperror(head)
    #else
        #define dprintf (void)
        #define dperror (void)
    #endif
#endif

extern void transport_init(mysocket_t sd, bool_t is_active);
#define maxBufferSize 536 // As in STCP, max packet size can be 536
#define maxPayloadSize 516 // As in STCP, max payload size can be 536. But header s

#endif  /* __TRANSPORT_H__ */
```