Bureau d’étude Réseau

TD 2

By Thibault Eynard-Suarez and Jaehyung Cho

Introduction

We supposed in this report the existence of global variables bearing the information of the connection (CONNEXION = true / false), of the existence of a current demand of connection (DEMAND_OF_CONNEXION = true / false) as well as global information about the addresses and sockets of the emitting and receiving applications given by the functions mic_tcp_socket and mic_tcp_bind which we have not implemented (since not asked for here).

We implemented our functions in C like pseudo-code.

Part I. Data transfer without loss recovery

```
/*
* Enables to request the sending of an applicative data
* Returns the size of sent datas, -1 in case of failure
*/
int mic_tcp_send (int mic_sock, char* mesg, int mesg_size) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");
  // getting the socket from it’s id to get the structure mic_tcp_sock associated
  my_socket //← the structure
  // constructing the PDU
  mic_tcp_pdu pdu;
  pdu.payload.data = mesg;
  pdu.payload.size = mesg_size;
  Pdu.header.source_port //<- write the number of the local port associated with the socket
  pdu.header.dest_port //<- write the number in mon_socket.addr.port (the destination)
  //int sent_data = IP_send(pdu, // Get back the address associated with my_socket given to the connect );
  Return sent_data;
}

/*
* Enables the receiving application to request the recovery of datas stocked in the reception
* buffers of the socket
* Returns the number of read datas, -1 in case of failure
* NB : this function calls the function app_buffer_get()
*/
int mic_tcp_recv (int socket, char* mesg, int max_mesg_size) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");
  // constructing a payload to receive datas in
  Mic_tcp_payload payload;
  Payload.data = mesg;
  Payload.size = max_mesg_size;
  int written_size = app_buffer_get(payload); // Ask for the datas in the reception buffer
  return written_size;
}
```

```
/*
* Enables to treat a PDU MIC-TCP received (update sequence numbers and ack
* numbers, …) then inserts datas from the pdu (msg) into the reception buffer of the socket.
* This function calls the function app_buffer_put(). It is called by initialize_components().
*/
void process_received_PDU(mic_tcp_pdu pdu, mic_tcp_sock_addr addr) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");
  // checks that the destination port of the header of the pdu is the port of an actual local socket

  if (CONNEXION == true) { // data transfer in connected mode

    if(pdu.header.ack == 1 & pdu.header.syn == 1) {
      // Case in wich we receive a syn ack while already connected : means the final ack of the connection did not go
through

      // check the source is the one we established a connection with

      mic_tcp_pdu ack;
      ack.header.source_port = //<- write the number of the local port associated with the socket
      ack.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
      ack.header.ack = 1;
      int s = IP_send(ack, /* Get back the address associated with my_socket given to the connect */ );
    }

    if(pdu.header.ack == 0 && pdu.header.fin == 1) {
      // Case in which we are asked to end the connection
      mic_tcp_pdu ack_fin;
      ack_fin.header.source_port = //<- write the number of the local port associated with the socket
      ack_fin.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
      ack_fin.header.ack = 1;
      ack_fin.header.fin = 1;

      mic_tcp_pdu ack;
      mic_tcp_sock_addr ack_addr;

      int received = -1;
      while(received == -1) {
        int s = IP_send(ack_fin, /* address */);

        received = IP_recv(ack, ack_addr, timer);
        if (received != -1 ) {
          if(ack.header.ack == 1) {
            break;
          }
          received = -1;
        }
      }

    }

    app_buffer_put(pdu.payload);
  }

  else {

    if(pdu.header.ack == 1 && pdu.header.fin == 1) {
      // Case in which the final ack of the end of connection did not go though
      mic_tcp_pdu ack;
      ack.header.dest_port = //
      ack.header.source_port = //
      ack.header.ack = 1;
      int s = IP_send(ack, /* address*/ );
      return ;
    }
```

```
    // establishing the connection
    if (pdu.header.syn == 0){
      return ; // Not asking to connect
    }

    DEMAND_FOR_CONNEXION = true; //false by default
    // create a mic_tcp_sock_addr to contain the address of the appli asking for a connection
  }

}
```

Part II. Data transfer with loss recovery (Stop-and-Wait)

```
/*
* Enables to request the sending of an applicative data
* Returns the size of sent datas, -1 in case of failure
*/
int mic_tcp_send (int mic_sock, char* mesg, int mesg_size) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");
  // getting the socket from it's id to get the structure mic_tcp_sock associated
  my_socket //← the structure

  // constructing the PDU
  mic_tcp_pdu pdu;
  pdu.payload.data = mesg;
  pdu.payload.size = mesg_size;
  pdu.header.source_port //<- write the number of the local port associated with the socket
  pdu.header.dest_port //<- write the number in mon_socket.addr.port (the destination)
  pdu.header.seq_num //← PE +1 mod 2



  mic_tcp_pdu ack_pdu;
  mic_tcp_sock_addr ack_addr;

  int received = -1;
  while( received == -1 ) {
    //Int sent_data = IP_send(pdu, // Get back the address associated with my_socket given to the connect );
    received = IP_recv(&ack_pdu, &ack_addr, timer) // timer = 1000

    if(received != -1) {
      if(ack_pdu.header.ack == 1) {
        if(ack_pdu.header.ack_num == PE) {
          break;
        }
      }

    }
    received = -1;
  }
  return sent_data;
}



/*
* Enables the receiving application to request the recovery of datas stocked in the reception
* buffers of the socket
* Returns the number of read datas, -1 in case of failure
* NB : this function calls the function app_buffer_get()
*/
int mic_tcp_recv (int socket, char* mesg, int max_mesg_size) {
```

```
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");
  // constructing a payload to receive datas in
  mic_tcp_payload payload;
  payload.data = mesg;
  payload.size = max_mesg_size;
  int written_size = app_buffer_get(payload); // Ask for the datas in the reception buffer
  return written_size;
}



/*
* Enables to treat a PDU MIC-TCP received (update sequence numbers and ack
* numbers, …) then inserts datas from the pdu (msg) into the reception buffer of the socket.
* This function calls the function app_buffer_put(). It is called by initialize_components().
*/
void process_received_PDU(mic_tcp_pdu pdu, mic_tcp_sock_addr addr) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");
  // checks that the destination port of the header of the pdu is the port of an actual local socket

  if (CONNEXION == true) { // data transfer in connected mode

    if(pdu.header.ack == 1 & pdu.header.syn == 1) {
      // Case in wich we receive a syn ack while already connected : means the final ack of the connection did not go
through

      // check the source is the one we established a connection with

      mic_tcp_pdu ack;
      ack.header.source_port = //<- write the number of the local port associated with the socket
      ack.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
      ack.header.ack = 1;
      int s = IP_send(ack, /* Get back the address associated with my_socket given to the connect */ );
    }

    if(pdu.header.ack == 0 && pdu.header.fin == 1) {
      // Case in which we are asked to end the connection
      mic_tcp_pdu ack_fin;
      ack_fin.header.source_port = //<- write the number of the local port associated with the socket
      ack_fin.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
      ack_fin.header.ack = 1;
      ack_fin.header.fin = 1;

      mic_tcp_pdu ack;
      mic_tcp_sock_addr ack_addr;

      int received = -1;
      while(received == -1) {
        int s = IP_send(ack_fin, /* address */);

        received = IP_recv(ack, ack_addr, timer);
        if (received != -1 ) {
          if(ack.header.ack == 1) {
            break;
          }
          received = -1;
        }
      }

    }

    if(pdu.header.ack == 0) { // not an ack
      if(pdu.header.seq_num == PA) {
        app_buffer_put(pdu.payload);
        // PA = (PA + 1) mod 2 // update PA
```

```
      }

      mic_tcp_pdu ack_pdu;
      ack_pdu.header.dest_port = pdu.header.source_port;
      ack_pdu.header.source_port = pdu.header.dest_port;
      ack_pdu.header.ack = 1;
      ack_pdu.header.ack_num = PA; // the old or new PA
      int s = IP_send(ack_pdu, /* address from the connection */ );

    }

    app_buffer_put(pdu.payload);

  }

  else {

    if(pdu.header.ack == 1 && pdu.header.fin == 1) {
      // Case in which the final ack of the end of connection did not go though
      mic_tcp_pdu ack;
      ack.header.dest_port = //
      ack.header.source_port = //
      ack.header.ack = 1;
      int s = IP_send(ack, /* address*/ );
      return ;
    }



    // establishing the connection
    if (pdu.header.syn == 0){
      return ; // Not asking to connect
    }

    DEMAND_FOR_CONNEXION = true; //false by default
    // create a mic_tcp_sock_addr to contain the address of the appli asking for a connection
  }
}
```

Part III. Connection

```
/*
* Put the socket in the state of connexion acception
* Returns 0 if success, -1 if failure
*/
int mic_tcp_accept(int socket, mic_tcp_sock_addr* addr) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");

  if(DEMAND_FOR_CONNEXION == false) {
    return -1;
  }

  // we take the mic_tcp_sock_addr of the appli asking to connect
  mic_tcp_pdu syn_ack;
  syn_ack.header.source_port = // filled with
  syn_ack.header.dest_port = // filled with
  syn_ack.header.ack = 1;
  syn_ack.header.syn = 1;

  mic_tcp_pdu ack;
  mic_tcp_sock_addr ack_addr;
```

```
  int ack_received = -1;
  while(ack_received = -1) {
    int s = IP_send(syn_ack, /*...*/);

    ack_received = IP_recv(ack, ack_addr, timer);

    // after a certain time : fails
  }

  CONNEXION = true;

}



/*
* Enables to require the establishment of a connection
* Returns 0 if the connection was successfully established, -1 if failure
*/
int mic_tcp_connect (int socket, mic_tcp_sock_addr addr) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");
  // Stock the destination addr by associating it with the socket identified as int socket.

  mic_tcp_pdu syn;
  syn.header.source_port = //<- write the number of the local port associated with the socket
  syn.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
  syn.header.syn = 1;

  mic_tcp_pdu ack_syn;
  mic_tcp_sock_addr ack_syn_addr;



  int received = -1;
  while( received == -1 ) {
    int s = IP_send(syn, /* Get back the address associated with my_socket given to the connect */ );
    received = IP_recv(&ack_syn, &ack_syn_addr, timer) // timer = 1000

    if(received != -1) {
      if(ack_syn.header.ack == 1 && ack_syn.header.syn == 1) {
        break;
      }
    }
    received = -1;
  }

  mic_tcp_pdu ack;
  ack.header.source_port = //<- write the number of the local port associated with the socket
  ack.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
  ack.header.ack = 1;

  int s = IP_send(ack, /* Get back the address associated with my_socket given to the connect */ );

  CONNEXION = true ; // Global variable

  return 0;
}



/*
* Enables to require the destruction of a socket
* Leads to the closure of the connection according to the TCP model
```

```
* Returns 0 if success, -1 if failure
*/
int mic_tcp_close (int socket) {
  printf("[MIC-TCP] Call to the function: "); printf(__FUNCTION__); printf("\n");

  mic_tcp_pdu fin;
  fin.header.source_port = //<- write the number of the local port associated with the socket
  fin.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
  fin.header.fin = 1;

  mic_tcp_pdu ack_fin;
  mic_tcp_sock_addr ack_fin_addr;

  int received = -1;
  while( received == -1 ) {
    int s = IP_send(fin, /* Get back the address associated with my_socket given to the connect */ );
    received = IP_recv(&ack_fin, &ack_fin_addr, timer) // timer = 1000

    if(received != -1) {
      if(ack_fin.header.ack == 1 && ack_fin.header.fin == 1) {
        break;
      }
    }
    received = -1;
  }

  mic_tcp_pdu ack;
  ack.header.source_port = //<- write the number of the local port associated with the socket
  ack.header.dest_port = //<- write the number in mon_socket.addr.port (the destination)
  ack.header.ack = 1;

  int s = IP_send(ack, /* Get back the address associated with my_socket given to the connect */ );

  CONNEXION = false ; // Global variable

  return 0;
}
```