Sverres approach to a networking module for the ttk4145 project

Sverre Hendseth

January 24, 2017

Contents

1 Introduction

I do not attempt "perfect code" here. Rather I make a point out of:

- Getting a nice module interface, suited for most C lift projects.
- . . . including very straightforward error modes.
- brutally copying code from the internet :-)

Notice that the decisions

- "The networking of the lift system can be made into a module. (. . . encapsulating a significant complexity behind a narrow interface)", and
- "This given set of functions (see the design chapter under) is a complete representation of the responsibilities of the network module needed in this context."

while not trivial, *can* be made without a complete analysis of the system or even a detailed knowledge of the specification.

Be aware of, and encourage, these "intuitive" design processes.

It turns out that this mode of thinking is applicable also when solving other challenges in life than SW design.

2 Infrastructure

This document is generated from a text-file with some markup (emacs' org-mode). Also the document is written in a "literate programming" style where also the implementation files (all the c-code, header and the Makefile)

Combine this style with a macro processor and you have a *fabulous* expressive power!

Here are the structure of the module files.

2.1 The header file

```
/*** #file "sverresnetwork.h" ***/
#ifndef SVERRESNETWORK_H
#define SVERRESNETWORK_H

// Note that the callback functions you provide may be called by the
// threads created by this module. Synchronize the access to any
// resources.

typedef void (*TMessageCallback)(const char * ip, char * data, int datalength);
typedef void (*TTcpConnectionCallback)(const char * ip, int created);

/*** sverresnetwork header includes ***/

/*** sverresnetwork header contents ***/

#endif
/*** End of File ***/
```

2.2 The implementation file

```
/*** #file "sverresnetwork.c" ***/
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#include <assert.h>
#include <pthread.h>

/*** sverresnetwork implementation includes ***/

#include "sverresnetwork.h"

void
error(char *s)
{
  perror(s);
  exit(1);
}

/*** sverresnetwork implementation ***/

/*** End of File ***/
```

3 Specification

3.1 TCP

- Must be able to listen for connections
  - nonblocking; establishing a connection should per default continue listening - notification per callback of establishing a connection?
- Must be able to stop listening (?)
- Must be able to refuse a connection (?)
- Must be able to establish a connection towards a listening program.
- Must be able to send a string of bytes with a given length, in both directions.

- Must be able to close a connection and tidy up a lost connection.

Failure modes should be simple.

3.2 UDP

- Should be able to send a string of bytes with a given length to another listening node.
- Should be able to broadcast a string to all listening nodes.
- Should be able to listen for messages.

3.3 Misc

- IP addresses and port numbers are great adresses for referring to a node.
- Should be able to find the ip address of myself.

4 Design

4.1 Misc

This is just the getMyIpAddress function: char* -> char*

```
/*** sverresnetwork header contents ***/
// Misc
char * getMyIpAddress(char * interface); // NB: allocates the return string!

/*** End of sverresnetwork header contents ***/
```

4.2 UDP

What about the following functions:

```
/*** sverresnetwork header contents ***/
// UDP & Broadcast
void udp_startReceiving(int port,TMessageCallback callBack);
void udp_send(char * address,int port,char * data, int dataLength);
void udp_broadcast(int port,char * data, int dataLength);

/*** End of sverresnetwork header contents ***/
```

Skipping this for now: void udp_stopReceiving(int port);

Possibly udp_stopListening will not be needed (?), but in any way Code Complete recommend that "start" and "stop" should come in pairs :-)

4.3 TCP

We notify the user system of this module by callbacks: One for a message received and one for a connection that has been established or lost.

So wee need to keep track of all open connections, and handle that they are closed/disappears.

If we choose to listen for new connections we also need a thread for this.

What about these:

```
/*** sverresnetwork header contents ***/

// TCP
void tcp_init(TMessageCallback messageCallback, TTcpConnectionCallback connectionCallba
void tcp_startConnectionListening(int port);
void tcp_openConnection(char * ip,int port);
void tcp_send(char * ip,char * data, int datalength);

/*** End of sverresnetwork header contents ***/
```

void tcp_stopListening(...); Is this one needed?

void tcp_close(); This?

We will have one thread listening for connection per listening port, and one thread reading from each open connection.

5 Implementation

5.1 Controlling logging

...I do not export this now. Any students searching for it will find it.

```
/*** sverresnetwork implementation ***/
int m_log = 1;

void
setLogMode(int l){
  m_log = l;
}
/*** End of sverresnetwork implementation ***/
```

5.2 getMyIpAddress

First and best on google: ("C finding my ip adress" -> `http://stackoverflow.com/questions/20800319/how-do-i-get-my-ip-address-in-c-on-linux`)
hints at getifaddrs(&addrs)

Google "getifaddrs example linux" yields this nice example:
`http://stackoverflow.com/questions/4139405/how-can-i-get-to-know-the-ip-address-for`

But it seems we must know the name of the "interface" we are looking for (A pc might have more network cards or other networks like wireless).

```
/*** sverresnetwork implementation includes ***/
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <ifaddrs.h>
#include <string.h>
#include <assert.h>
#include <stdio.h>
#include "sverresnetwork.h"
/*** End of sverresnetwork implementation includes ***/

/*** sverresnetwork implementation ***/
char *
getMyIpAddress(char * interface)
{
    struct ifaddrs *ifap, *ifa;
    struct sockaddr_in *sa;
    char *addr;

    assert(strlen(interface) < 20);

    getifaddrs (&ifap);

    for (ifa = ifap; ifa; ifa = ifa->ifa_next) {
        if (ifa->ifa_addr->sa_family==AF_INET) {
            sa = (struct sockaddr_in *) ifa->ifa_addr;
            addr = inet_ntoa(sa->sin_addr);
            if(strncmp(ifa->ifa_name,interface,strlen(ifa->ifa_name)) == 0){
              char * res = strdup(addr);
              freeifaddrs(ifap);
```

```
              return res;
            }
//            printf("Interface: %s\tAddress: '%s' - '%s'\n", ifa->ifa_name, addr, inte
        }
    }
    fprintf(stderr,"\n\nERROR: getMyIpAddress: Could not find interface %s.\nAlternativ
    for (ifa = ifap; ifa; ifa = ifa->ifa_next) {
        if (ifa->ifa_addr->sa_family==AF_INET) {
            sa = (struct sockaddr_in *) ifa->ifa_addr;
            addr = inet_ntoa(sa->sin_addr);
            fprintf(stderr,"Error: Interface: %8s\tAddress: %s\n", ifa->ifa_name, addr)
        }
    }
    fprintf(stderr,"\n");

    freeifaddrs(ifap);
    assert(0);
    return 0;
}

/*** End of sverresnetwork implementation ***/
```

5.3 UDP

Second hit on "udp in C" on google yields

`https://www.abc.se/~m6695/udp.html`

Example code on sending and receiving udp messages.

I must add some header files to make them compile. Also I research UDP broadcast and mudify the client to broadcast rather than just send to one PC. I need to use "local broadcast address" 255.255.255.255 and call setsockopt with the SO_BROADCAST option.

Talking about setsockopt: I also added som "reusable" status, so that a crash does not occupy a bound socket for a long time.

5.3.1 Listening

This will be done with a separate thread.

1. Data structures

   ```
   /*** sverresnetwork implementation ***/
   ```

```
// Need to keep track of running listening threads.
typedef struct {
  int port;
  int socket;
  TMessageCallback callBack;
  pthread_t thread;
} TUdpThreadListItem;

#define NLISTENINGTHREADS 100
#define BUFLEN 512

TUdpThreadListItem * listeningThreads[NLISTENINGTHREADS];
int nOfListeningThreads = 0;

/*** End of sverresnetwork implementation ***/
```

2. thr_udpListen

```
/*** sverresnetwork implementation ***/
void *
thr_udpListen(void * arg){

  TUdpThreadListItem * threadListItem = (TUdpThreadListItem*) arg;

  struct sockaddr_in si_me, si_other;
  socklen_t slen = sizeof(si_other);
  char buf[BUFLEN];

  threadListItem->socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  assert(threadListItem->socket != -1);

  memset((char *) &si_me, 0, sizeof(si_me));
  si_me.sin_family = AF_INET;
  si_me.sin_port = htons(threadListItem->port);
  si_me.sin_addr.s_addr = htonl(INADDR_ANY);

  int optval = 1;
  setsockopt(threadListItem->socket,SOL_SOCKET,SO_REUSEADDR, &optval, sizeof(optva
```

```
  int res = bind(threadListItem->socket,(struct sockaddr *) &si_me, sizeof(si_me))
  if(res == -1) error("thr_udpListen:bind");

  if(m_log) printf("Sverresnetwork: udpListen: Binding to port %d\n",threadListIte

  while(1){
    res = recvfrom(threadListItem->socket, buf, BUFLEN, 0,(struct sockaddr *) &si_
    if(res == -1) error("thr_udpListen:recvfrom");
    if(res >= BUFLEN-1){
      fprintf(stderr,"recvfrom: Hmm, length of received message is larger than max
      assert(res < BUFLEN-1);
    }
    // printf("Received packet from %s:%d\nLength = %d, Data: <%s>\n\n", inet_ntoa
     (*(threadListItem->callBack))(inet_ntoa(si_other.sin_addr),buf,res);
  }

  // Never executed - this thread will be killed if it is not needed any more.
  close(threadListItem->socket);
  return 0;
}
/*** End of sverresnetwork implementation ***/
```

3. udp_startReceiving

```
/*** sverresnetwork implementation ***/

void udp_startReceiving(int port,TMessageCallback callBack){

  assert(nOfListeningThreads<NLISTENINGTHREADS);

  listeningThreads[nOfListeningThreads] = (TUdpThreadListItem*) malloc(sizeof(TUdp
  assert(listeningThreads[nOfListeningThreads] != NULL);

  listeningThreads[nOfListeningThreads]->port = port;
  listeningThreads[nOfListeningThreads]->callBack = callBack;

  int res = pthread_create(&(listeningThreads[nOfListeningThreads]->thread), NULL,
  if(res != 0) error("pthread_create failed");

  nOfListeningThreads++;
```

```
    }

    /*** End of sverresnetwork implementation ***/
```

5.3.2 Sending

Should possibly cache the sockets here rather than opening and closing all the time.

```
/*** sverresnetwork implementation ***/

void
udp_send(char * address,int port,char * data, int dataLength){
  struct sockaddr_in si_other;
  int s, slen=sizeof(si_other);

  if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
    error("udp_send: socket");

  memset((char *) &si_other, 0, sizeof(si_other));
  si_other.sin_family = AF_INET;
  si_other.sin_port = htons(port);
  int res = inet_aton(address, &si_other.sin_addr);
  if(res==0) error("inet_aton() failed\n");

  res = sendto(s, data, dataLength, 0, (struct sockaddr *) &si_other, slen);
  if(res==-1)  error("udp_send: sendto()");

  close(s);
}

/*** End of sverresnetwork implementation ***/
```

5.3.3 udp_brodacast

Should possibly cache the sockets here rather than opening and closing all the time.

```
/*** sverresnetwork implementation ***/

void
```

```
udp_broadcast(int port,char * data, int dataLength){
  struct sockaddr_in si_other;
  int s, slen=sizeof(si_other);

  if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
    error("udp_send: socket");

  int optval = 1;
  setsockopt(s,SOL_SOCKET,SO_BROADCAST, &optval, sizeof(optval));

  memset((char *) &si_other, 0, sizeof(si_other));
  si_other.sin_family = AF_INET;
  si_other.sin_port = htons(port);
  if (inet_aton("255.255.255.255", &si_other.sin_addr)==0) {
    fprintf(stderr, "inet_aton() failed\n");
    exit(1);
  }

  int res = sendto(s, data, dataLength, 0, (struct sockaddr *) &si_other, slen);
  if(res==-1)  error("udp_broadcast: sendto()");

  close(s);
}

/*** End of sverresnetwork implementation ***/
```

5.4 TCP

First hit on "opening a tcp connection C" on google yields the nice examples here `http://www.linuxhowtos.org/C_C++/socket.htm` that works, warning-free out of the box. (I should probably have found the udp code here also...)

5.4.1 tcp data

```
/*** sverresnetwork implementation ***/
static TMessageCallback m_messageCallback = NULL;
static TTcpConnectionCallback m_connectionCallback = NULL;

/*** End of sverresnetwork implementation ***/
```

5.4.2 Keeping track of open connections

This is a small module in itself:

- send must be able to look up the socket from the recipients ip
- Keeping track of open connections enables some consistency checking

```
/*** sverresnetwork implementation ***/
#define MAXTCPCONNECTIONS 100

typedef struct {
  char ip[32];
  int socket;
} TTcpConnection;

static TTcpConnection tcpConnections[MAXTCPCONNECTIONS];

void
conn_init(){
  int i;
  for(i=0;i<MAXTCPCONNECTIONS;i++) tcpConnections[i].socket=0;
}

int
conn_lookup(char * ip){
  int i;
  for(i=0;i<MAXTCPCONNECTIONS;i++){
    if(strncmp(tcpConnections[i].ip,ip,30) == 0 && tcpConnections[i].socket != 0){
      return tcpConnections[i].socket;
    }
  }
  return 0;
}

const char *
conn_findIp(int socket){
  int i;
  for(i=0;i<MAXTCPCONNECTIONS;i++){
    if(tcpConnections[i].socket == socket){
      return tcpConnections[i].ip;
```

```
    }
  }
  return NULL;
}


void
conn_add(char * ip,int s){
  int i;

  if(conn_lookup(ip))
    error("conn_add: Trying to add an already existing connection");

  for(i=0;i<MAXTCPCONNECTIONS;i++){
    if(tcpConnections[i].socket == 0){
      strncpy(tcpConnections[i].ip,ip,30);
      tcpConnections[i].socket = s;
      return;
    }
  }
  error("conn_add: No free connection slots?");
}

void
conn_remove(const char * ip){
  int i;
  for(i=0;i<MAXTCPCONNECTIONS;i++){
    if(strncmp(tcpConnections[i].ip,ip,30) == 0){
      tcpConnections[i].socket = 0;
    }
  }
}


/*** End of sverresnetwork implementation ***/
```

5.4.3 thr_tcpMessageListen

```
/*** sverresnetwork implementation ***/
void *
```

```
thr_tcpMessageListen(void * parameter){
  int socket = (long) parameter;
  char buffer[1024];

  if(m_log) printf("SverresNetwork: Starting reading messages from socket %d\n",socket)

  while(1){
    bzero(buffer,1024);
    int n = read(socket,buffer,1020);
    if(n <= 0){
      // Something wrong; close and give up
      close(socket);
      if(m_log) printf("SverresNetwork: Lost a connection to %s - socket %d\n",conn_fi
      m_connectionCallback(conn_findIp(socket),0);
      conn_remove(conn_findIp(socket));
      return NULL;
    }

    // Got a message, do the callback
    m_messageCallback(conn_findIp(socket),buffer,n);
  }
}

/*** End of sverresnetwork implementation ***/
```

5.4.4 void * thr_tcpConnectionListen(void *)

```
/*** sverresnetwork implementation ***/
void *
thr_tcpConnectionListen(void * parameter){
  int sockfd, newsockfd, port;
  socklen_t clilen;
  struct sockaddr_in serv_addr, cli_addr;

  pthread_t thread;

  port = (long) parameter;

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if(sockfd < 0)
```

```
    error("ERROR opening socket");

  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(port);

  int optval = 1;
  setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR, &optval, sizeof(optval));

  if(bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
    error("ERROR on binding");

  listen(sockfd,5);

  if(m_log) printf("SverresNetwork: Starting listening for connections on %d\n",port);

  clilen = sizeof(cli_addr);
  while(1){
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    if(newsockfd < 0)
      error("ERROR on accept");

    // Now: Have the connection!
    char * newConnectionIp = strdup(inet_ntoa(cli_addr.sin_addr));
    if(m_log) printf("SverresNetwork: Got a connection from %s: socket %d\n",newConnect

    conn_add(newConnectionIp,newsockfd);

    // Notify the system
    if(m_log) printf("SverresNetwork: Created a connection to %s - socket %d\n",newCon
    m_connectionCallback(newConnectionIp,1);

    // And create the thread that will receive the messages.
    long lfd = newsockfd;
    int res = pthread_create(&thread, NULL, thr_tcpMessageListen,(void *) lfd);
    if(res != 0) error("pthread_create failed");
  }

}
```

```
/*** End of sverresnetwork implementation ***/
```

5.4.5 tcp_init()

```
/*** sverresnetwork implementation ***/

void
tcp_init(TMessageCallback messageCallback, TTcpConnectionCallback connectionCallback){
  m_messageCallback = messageCallback;
  m_connectionCallback = connectionCallback;
  conn_init();
}

/*** End of sverresnetwork implementation ***/
```

5.4.6 tcp_StartConnectionListening(int port)

```
/*** sverresnetwork implementation ***/

void
tcp_startConnectionListening(int port){
  pthread_t thread;

  long lfd = port;
  int res = pthread_create(&thread, NULL, thr_tcpConnectionListen,(void *) lfd);
  if(res != 0) error("pthread_create failed");
}


/*** End of sverresnetwork implementation ***/
```

5.4.7 void tcp_openConnection(char * ip,int port)

```
/*** sverresnetwork implementation ***/
void
tcp_openConnection(char * ip,int port){
  int sockfd;
  struct sockaddr_in serv_addr;

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
  if(sockfd < 0)
    error("ERROR opening socket");

  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  int res = inet_aton(ip, &serv_addr.sin_addr);
  if(res==0) error("inet_aton() failed\n");
  serv_addr.sin_port = htons(port);

  res = connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr));
  if(res < 0){
    // error("ERROR connecting");
    if(m_log) fprintf(stderr,"WARNING: sverresnetwork: Tried to open connection to %s:%
    return;
  }

  // Have a connection; register it and make the callback.
  conn_add(ip,sockfd);

  // Notify the system
  if(m_log) printf("SverresNetwork: Created a connection to %s - socket %d\n",ip,sockfd
  m_connectionCallback(ip,1);

  // And create the thread that will receive the messages.
  long lfd = sockfd;
  pthread_t thread;
  res = pthread_create(&thread, NULL, thr_tcpMessageListen,(void *) lfd);
  if(res != 0) error("pthread_create failed");

}


/*** End of sverresnetwork implementation ***/
```

5.4.8 tcp_send

```
/*** sverresnetwork implementation ***/
void
tcp_send(char * ip,char * data, int datalength){
  int socket = conn_lookup(ip);
```

```
  if(socket==0){
    // The connection is nonexistent: Must notify
    if(m_log) printf("SverresNetwork: Tried to write to a nonexistant connection: %s\n"
    m_connectionCallback(ip,0);
  }

  int res = write(socket,data,datalength);
  if(res != datalength){
    if(m_log) printf("SverresNetwork: Writing failed to %s:%d Got %d/%d Closing\n",ip,s
//    error("write:");
    if(m_log) printf("SverresNetwork: Closed a connection to %s - socket %d\n",ip,socke
    m_connectionCallback(ip,0);
    conn_remove(ip);
  }
}

/*** End of sverresnetwork implementation ***/
```

6 Tests

6.1 test program for getMyIpAddress

```
/*** #file "test_getmyipaddress.c" ***/
#include <stdio.h>
#include <stdlib.h>
#include "sverresnetwork.h"

int main(){
  char * ip = getMyIpAddress("enp4s0");
  printf("My Ip is %s\n",ip);

  free(ip);

  return 0;
}

/*** End of File ***/
```

6.2 Testing udp

6.2.1 Udp Listening

```
/*** #file "test_udplisten.c" ***/
#include <stdio.h>
#include <unistd.h>
#include "sverresnetwork.h"

void
udpmessageReceived(const char * ip, char * data, int datalength){

  // Assuming an ascii string here - a binary blob (including '0's) will
  // be ugly/truncated.
  printf("Received UDP message from %s: '%s'\n",ip,data);

}

int main(){

  udp_startReceiving(4321,udpmessageReceived);

  sleep(100);
  return 0;
}

/*** End of File ***/
```

6.2.2 Udp Sending

```
/*** #file "test_udpsend.c" ***/
#include <stdio.h>
#include <unistd.h>
#include "sverresnetwork.h"

void
udpmessageReceived(const char * ip, int port, char * data, int datalength){

  printf("Received UDP message from %s:%d: '%s'\n",ip,port,data);

}
```

```
int main(){
//  char * ip = "129.241.10.74";
  char * ip = "192.168.10.99";

  udp_send(ip,4321,"Hello World",12);

  return 0;
}

/*** End of File ***/
```

6.2.3 Udp Broadcasting

```
/*** #file "test_udpbroadcast.c" ***/
#include <stdio.h>
#include <unistd.h>
#include "sverresnetwork.h"

void
udpmessageReceived(const char * ip, int port, char * data, int datalength){

  printf("Received UDP message from %s:%d: '%s'\n",ip,port,data);

}

int
main(){

  udp_broadcast(4321,"Hello all",10);

  return 0;
}

/*** End of File ***/
```

6.3 Testing tcp

6.3.1 A server

```
/*** #file "test_tcp_server.c" ***/
```

```
#include <stdio.h>
#include <unistd.h>
#include "sverresnetwork.h"

void
messageReceived(const char * ip, char * data, int datalength){

  printf("Received message from %s: '%s'\n",ip,data);
}

void
connectionStatus(const char * ip, int status){

  printf("A connection got updated %s: %d\n",ip,status);
}



int main(){

  tcp_init(messageReceived,connectionStatus);
  tcp_startConnectionListening(5555);

  sleep(100);
  return 0;
}

/*** End of File ***/
```

6.3.2 A client

```
/*** #file "test_tcp_client.c" ***/
#include <stdio.h>
#include <unistd.h>
#include "sverresnetwork.h"

void
messageReceived(const char * ip, char * data, int datalength){

  printf("Received message from %s: '%s'\n",ip,data);
```

```
}

void
connectionStatus(const char * ip, int status){

  printf("A connection got updated %s: %d\n",ip,status);
}



int main(){

  tcp_init(messageReceived,connectionStatus);

  tcp_openConnection("192.168.10.99",5555);

  tcp_send("192.168.10.99","Hello!",7);

  return 0;
}

/*** End of File ***/
```

6.4 Testing the complete system

```
/*** #file "test_sverresnetwork.c" ***/
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <assert.h>
#include "sverresnetwork.h"

char * myIp;

void
messageReceived(const char * ip, char * data, int datalength){
  printf("Received message from %s: '%s'\n",ip,data);
  if(strncmp("Hello",data,5) == 0 && strncmp(myIp,ip,strlen(myIp)) != 0 ){
    printf("Received Hello message: Connecting...\n");
```

```
    tcp_openConnection((char *)ip,5555); // Cast away the const...
  }
}

char ips[100][32];

void
initIps(){
  int i;
  for(i=0;i<100;i++){
    ips[i][0] = 0;
  }
}

void
addIp(const char * ip){
  int i;
  for(i=0;i<100;i++){
    if(ips[i][0] == 0){
      printf("Adding connection %s to index %d\n",ip,i);
      strncpy(ips[i],ip,30);
      return;
    }
  }
  assert(0);
}
void
removeIp(const char * ip){
  int i;
  for(i=0;i<100;i++){
    if(strncmp(ip,ips[i],strlen(ips[i])) == 0 && ips[i][0] != 0){
      printf("Removing connection %s from index %d\n",ip,i);
      ips[i][0] = 0;
      return;
    }
  }
}


void
```

```
connectionStatus(const char * ip, int status){
  printf("A connection got updated %s: %d\n",ip,status);
  if(status){
    addIp(ip);
  }else{
    removeIp(ip);
  }
}


int main(){
  int i;
  initIps();

  int myTcpPortNumber = 5555;

  myIp = getMyIpAddress("enp4s0");
//  myIp = getMyIpAddress("eth0");
  printf("My Ip is %s\n",myIp);

  udp_startReceiving(5555,messageReceived);

  char buffer[1024];

  sprintf(buffer,"Hello: Contact me at %s:%d",myIp,myTcpPortNumber);
  printf("Broadcasting my address: '%s'\n",buffer);
  udp_broadcast(5555,buffer,strlen(buffer)+1);

  tcp_init(messageReceived,connectionStatus);
  tcp_startConnectionListening(5555);

  printf("Going into send loop \n");

  while(1){
    sleep(4);
    printf("Slept: Going into send loop \n");
    for(i=0;i<100;i++){
      if(ips[i][0] != 0){
        printf("Sending a message to %s\n",ips[i]);
        tcp_send(ips[i],"Message",7);
```

```
      }
    }
  }

  sleep(100);
  return 0;
}

/*** End of File ***/
```

Some unused code from when I thought random port numbers were necessary.

```
  srand((unsigned) time(NULL));

//  int myTcpPortNumber = 10000 + rand()%10000;
//  printf("My Tcp Port Number is %ld\n",myTcpPortNumber);
```

7 The Makefile

It seems the TAB's get lost in the pdf export here.

```
/*** #file "Makefile" ***/

CFLAGS = -g -Wall

all: sverresnetwork.ok sverresnetwork.o test_getmyipaddress test_udplisten test_udpsend

sverresnetwork.c: sverresnetwork.ok

sverresnetwork.ok: ../sverresnetwork.org
clipprep ../sverresnetwork.org | tinyclip -va
touch $@

%.o: %.c
gcc -c $(CFLAGS) $<

sverresnetwork.o: sverresnetwork.h sverresnetwork.c

test_getmyipaddress: test_getmyipaddress.o sverresnetwork.o
```

```
gcc $(CFLAGS) test_getmyipaddress.o sverresnetwork.o -lpthread -o $@

test_udplisten: test_udplisten.o sverresnetwork.o
gcc $(CFLAGS) test_udplisten.o sverresnetwork.o -lpthread -o $@

test_udpsend: test_udpsend.o sverresnetwork.o
gcc $(CFLAGS) test_udpsend.o sverresnetwork.o -lpthread -o $@

test_udpbroadcast: test_udpbroadcast.o sverresnetwork.o
gcc $(CFLAGS) test_udpbroadcast.o sverresnetwork.o -lpthread -o $@

test_tcp_server: test_tcp_server.o sverresnetwork.o
gcc $(CFLAGS) test_tcp_server.o sverresnetwork.o -lpthread -o $@

test_tcp_client: test_tcp_client.o sverresnetwork.o
gcc $(CFLAGS) test_tcp_client.o sverresnetwork.o -lpthread -o $@

test_sverresnetwork: test_sverresnetwork.o sverresnetwork.o
gcc $(CFLAGS) test_sverresnetwork.o sverresnetwork.o -lpthread -o $@

/*** End of File ***/
```