

EE450 Socket Programming Project, Fall 2009

Due Date: Sunday November 22nd, 2009 11:59 AM (Noon)

(The deadline is the same for all on-campus and DEN off-campus students)

Hard deadline (Strictly enforced)

The objective of this assignment is to familiarize you with UNIX socket programming. This assignment is worth **10%** of your overall grade in this course.

It is an individual assignment and no collaborations are allowed. Any cheating will result in an automatic F in the course (not just in the assignment).

If you have any doubts/questions please feel free to contact the TAs and cc professor Zahid as usual.

The Problem:

In this project you will be simulating a race among 4 cellphone units to find their way with the help of 2 base stations and reach a target before running out of battery or going out of range. All communications take place over TCP and UDP sockets in client-server architecture. The project has 3 major phases: Bootstrapping, Route discovery and Contacting Target.

Code and Input files:

You must write your programs either in C or C++ on UNIX. In fact, you will be writing (at least) 3 different pieces of code:

1- Cellphone units

You must create 4 concurrent cellphone units.

- i. Either by using `fork()` or a similar Unix system call. In this case, you probably have only one piece of code for which you need to use one of these names: **cellphone.c** or **cellphone.cc** or **cellphone.cpp** (all small letters). Also you must call the corresponding header file (if any) **cellphone.h** (all small letters). You must follow this naming convention.
- ii. Or by running 4 instances of cellphone code. However in this case, you probably have 4 pieces of code for which you need to use one of these sets of names: (**cellphone1.c, cellphone2.c, ..., cellphone4.c**) or (**cellphone1.cc, ...,cellphone4.cc**) or (**cellphone1.cpp, cellphone2.cpp, ..., cellphone4.cpp**) (all small letters). Also you must call the corresponding header file (if any) **cellphone.h** (all small letters) or **cellphone1.h, cellphone2.h,... cellphone4.h** (all small letters).

2- Base stations

You must create 2 concurrent base stations.

- i. Either by using `fork()` or a similar Unix system call. In this case, you probably have only one piece of code for which you need to use one of these names: **bs.c** or **bs.cc** or **bs.cpp** (all small letters). Also you must call the corresponding header file (if any) **bs.h** (all small letters). You must follow this naming convention.
- ii. Or by running 2 instances of base station code. However in this case, you probably have 2 pieces of code for which you need to use one of these sets of names: (**bs1.c, bs2.c**) or (**bs1.cc, bs2.cc**) or (**bs1.cpp, bs2.cpp**) (all small letters). Also you must call the corresponding header file (if any) **bs.h** (all small letters) or **bs1.h, bs2.h** (all small letters).

3- Target

You must create 1 target.

Your code must use one of these names: **target.c** or **target.cc** or **target.cpp** (all small letters). Also you must call the corresponding header file (if any) **target.h** (all small letters).

You must follow above naming convention.

4- Input files: **scenario.txt, strategy.txt**

- i. scenario.txt contains the x-y GPS coordinates (x is an integer between 0 and 400 meters and y is an integer between 0 and 400 meters) for cellphone units, base stations and the target. A sample file is provided as follows:

```
cellphone1 100 320
cellphone2 150 150
cellphone3 10 300
cellphone4 300 20
basestation1 100 200
basestation2 300 200
target 10 10
```

- ii. strategy.txt contains the velocity (in terms of meters/sec), movement direction (W for west, E for east, S for South and N for North) and battery life (a positive integer less than or equal to 100) for cellphone units. The format of the sample input files are as follows:

```
cellphone1 50 W 80
cellphone2 60 N 100
cellphone3 100 S 50
cellphone4 90 E 60
```

In this file each line starts with the identifier string. The 2nd field is the velocity. The 3rd field is the direction, followed by the battery life. Please note that there are always 4 rows and 4 columns in this file.

As can be seen in both files, the delimiter used between the columns is one space. You may use the above samples to test your project. However when your project is graded, TAs will use different input files with the exact same format.

This project is divided into 3 phases. It is not possible to proceed to one phase without completing the previous phase. In each phase you will have multiple concurrent processes that will communicate either over TCP or UDP sockets.

Phase1: Bootstrapping

In this phase, each entity opens one or both input files and reads the necessary info from the input files then define address structures and creates UDP and/or TCP sockets.

Phase2: Route Discovery

In this phase cellphone units start moving according to the provided strategies in strategy.txt and try to correct course with the help of Base Stations while communicating with Base Stations over UDP sockets. Only cellphone units that get to a certain proximity of the Target will receive the TCP port number of the Target from Base Station. The rest of cellphone units will be terminated. This is the end of phase 2.

Phase3: Contacting Target

At this point Target is up and listening for the incoming connection requests from the cellphone units that moved on to phase3. The target accepts the incoming connection requests in the FCFS manner and sends responses back to cellphones through its TCP socket.

A more detailed explanation of the problem:

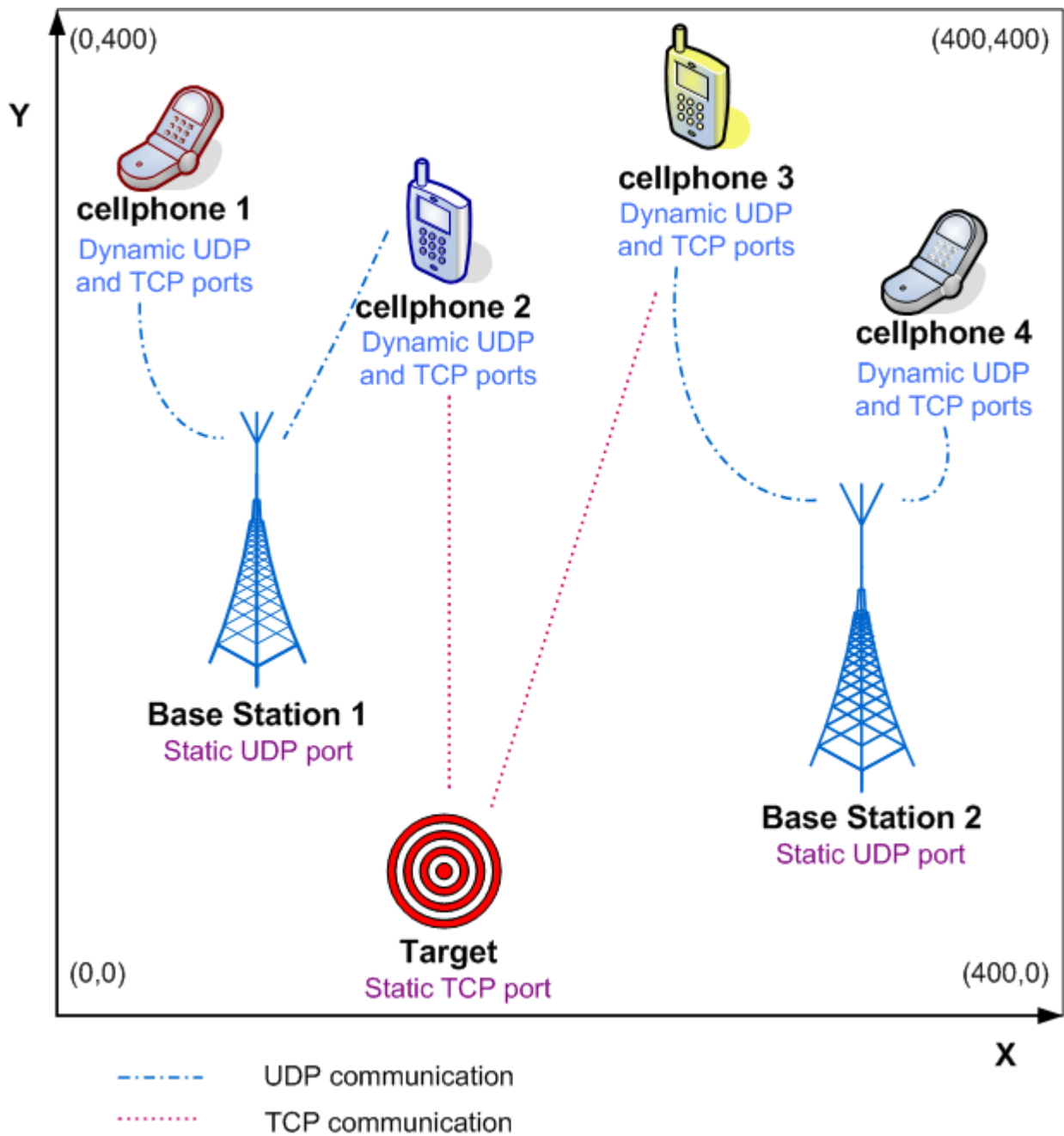


Figure 1

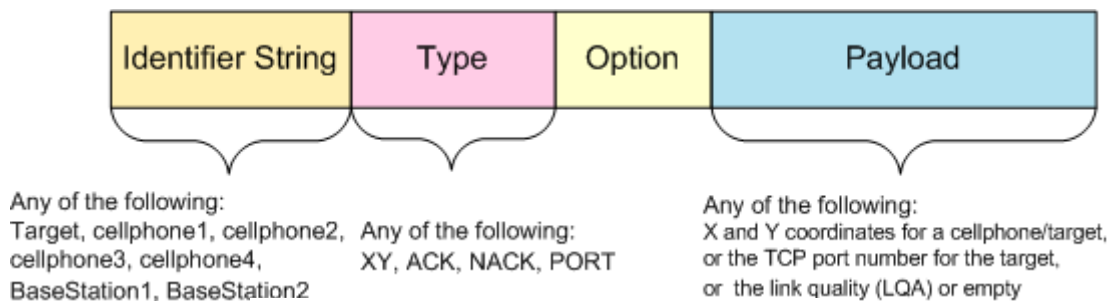


Figure 2. General format of a message

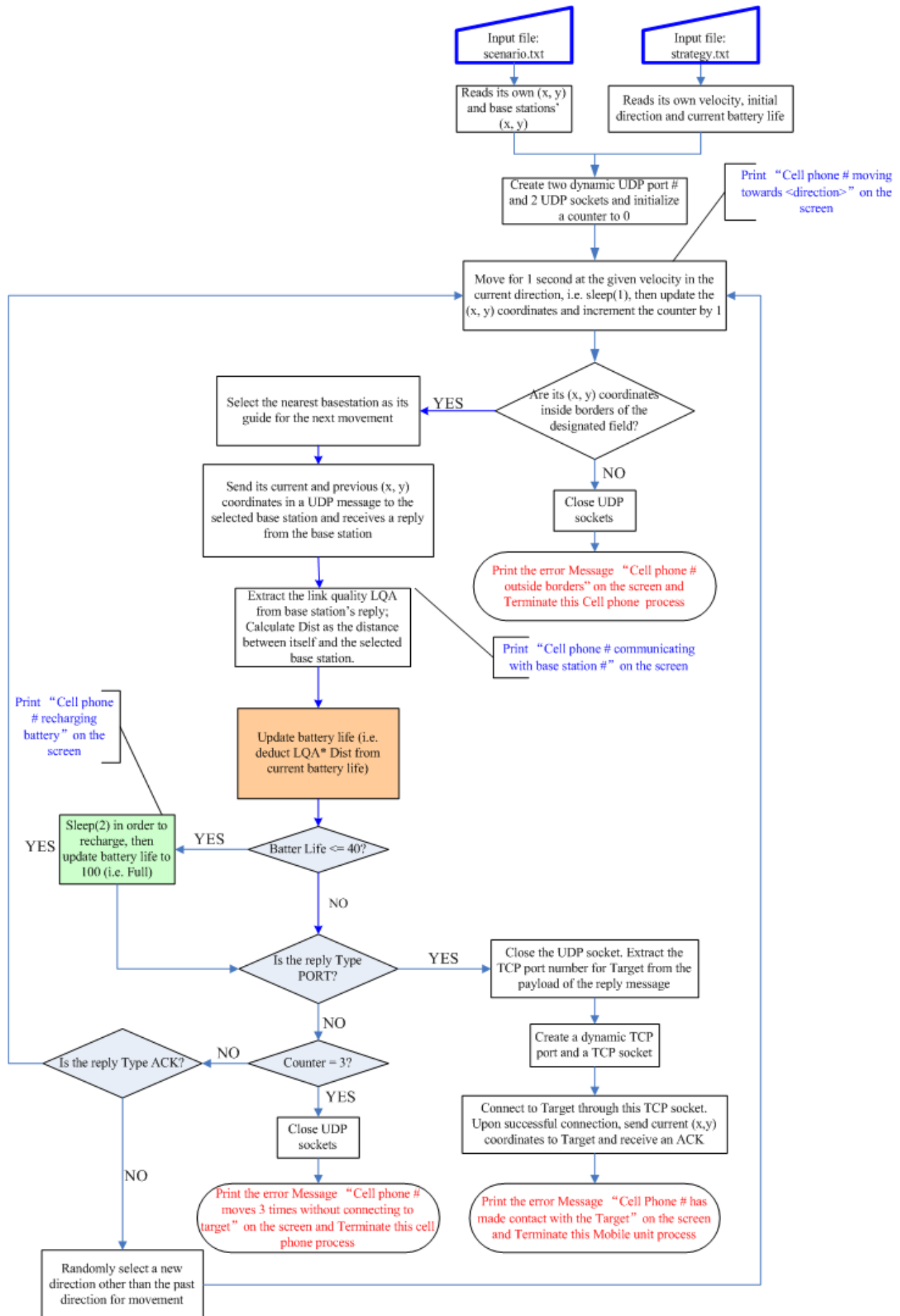


Figure 3. Flowchart of a Cell Phone unit

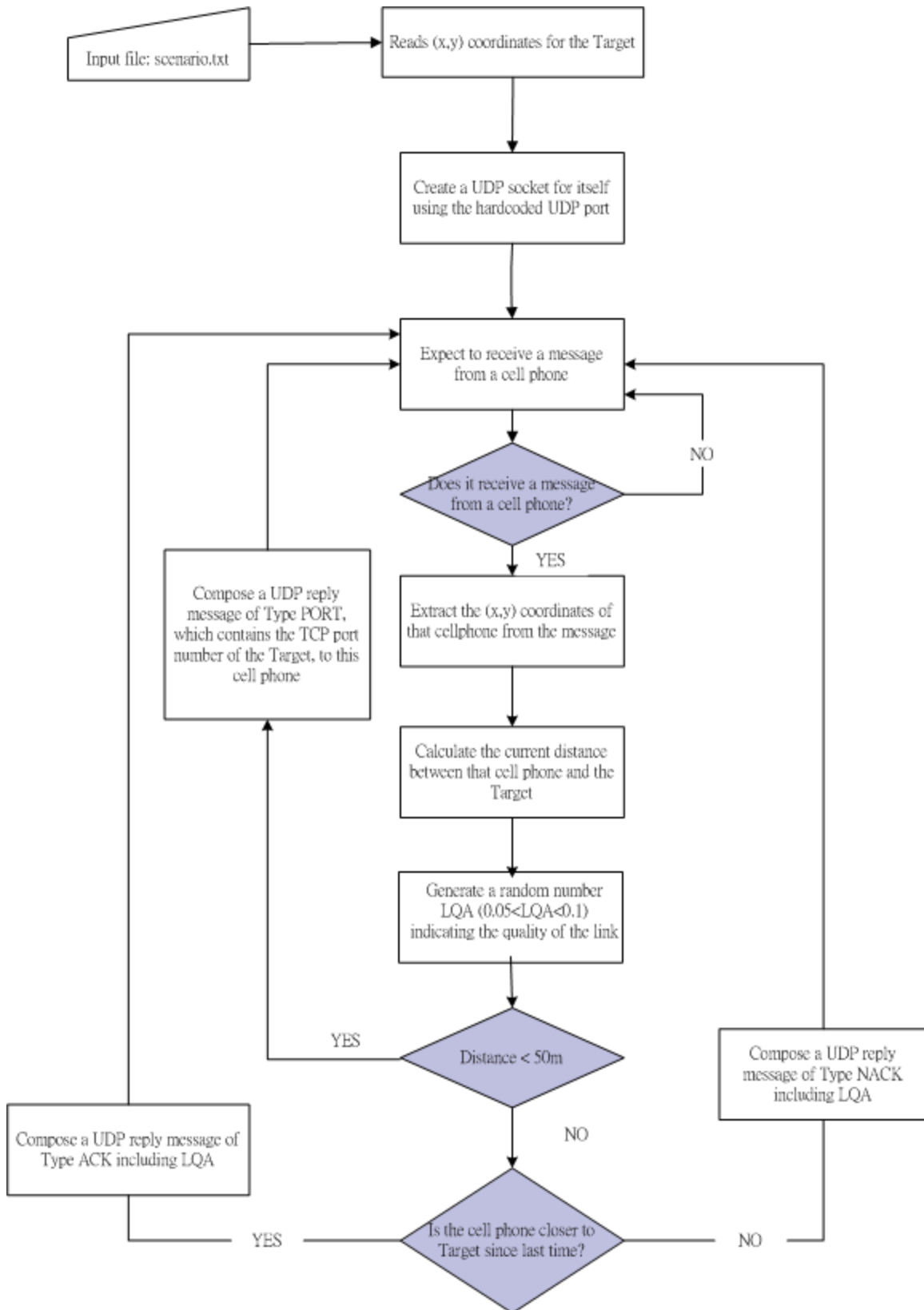


Figure 4. Flowchart of a Base Station

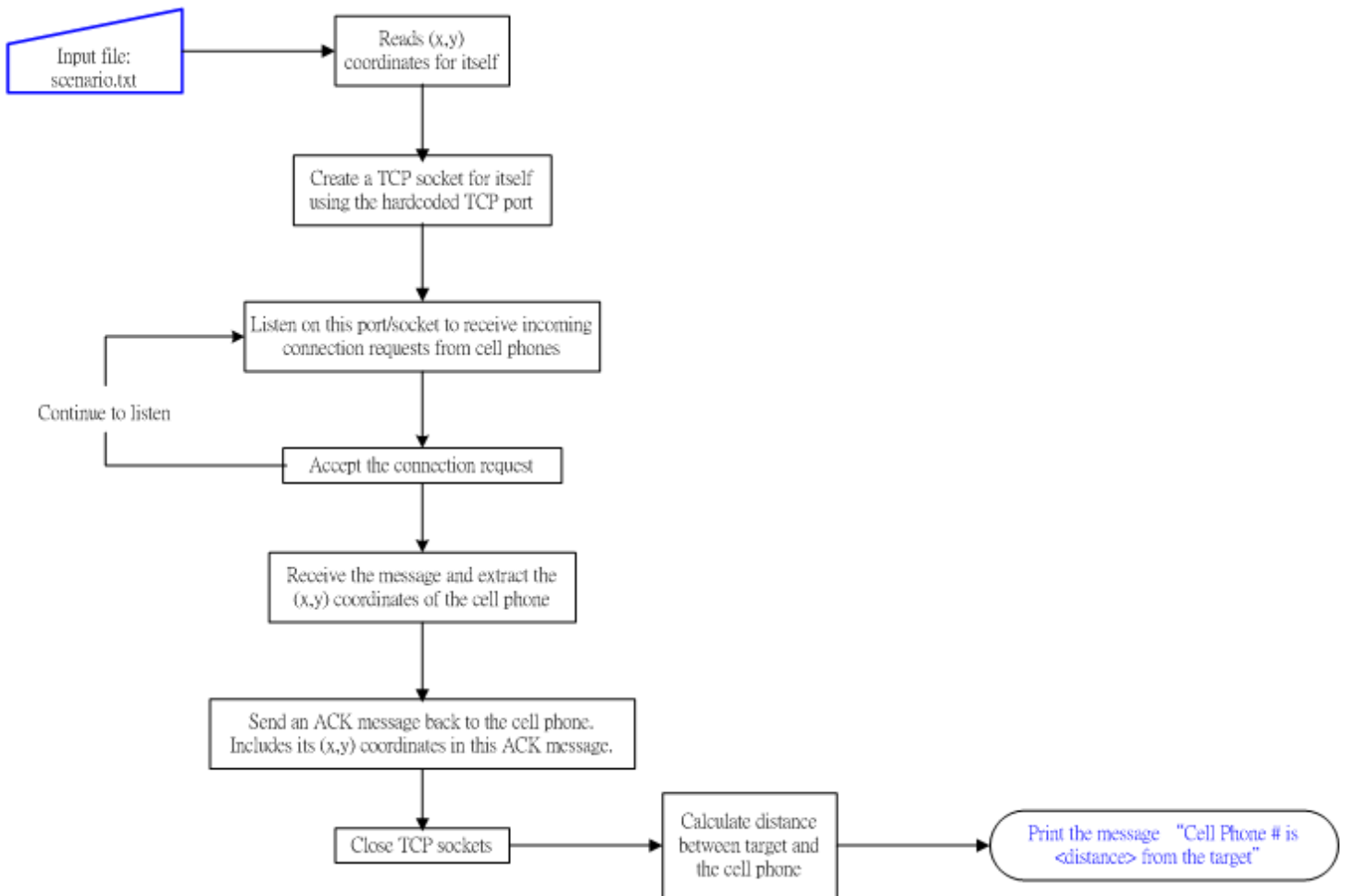


Figure 5. Flowchart of the Target

Table 1: A Summary of Static and Dynamic assignment of TCP and UDP Ports and IP addresses

| Process | Dynamic ports | Static Ports | IP address obtained from |
|---------------|------------------------------|--|-----------------------------|
| Target | None | 1 TCP Port, 21100 plus last 3 digits of your ID (e.g. 21100+751=21851) | local host or nunki.usc.edu |
| Base Station1 | None | 1 UDP Port, 4100 plus last 3 digits of your ID (e.g. 4100+751=4851) | local host or nunki.usc.edu |
| Base Station2 | None | 1 UDP Port, 4200 plus last 3 digits of your ID (e.g. 4200+751=4951) | local host or nunki.usc.edu |
| cellphone1-4 | 1/2 UDP port 0/1 TCP port | None | local host or nunki.usc.edu |

Packet formats:

Figure 2 shows the general format of a message. Each message starts with its sender's identifier string. Message types are: XY, ACK, NACK, PORT. You may use your own way to implement it. But they must contain the following information as shown in Table 2.

Table 2: A summary for different messages

| Type | From | To | Information |
|------------|-------------|-------------|---|
| XY | cellphone | basestation | Current (x,y) and previous (x,y) coordinates of the cellphone |
| ACK / NACK | basestation | cellphone | LQA (link quality) |
| ACK | target | cellphone | (x,y) coordinates of the target |
| XY | cellphone | target | Current (x,y) coordinates of the cellphone |
| PORT | basestation | cellphone | static TCP port number of the target |

On-screen Messages:

In order to clearly understand the flow of the project, your codes must print out the following messages on the screen as listed in the following Tables.

Table 3: Cell phone on-screen messages

| Event | On-screen Message |
|---|---|
| Upon Startup | <cellphone #>: UDP Port #s..... and IP address; initial (x,y) coordinates and battery life <cellphone #>: moving towards <direction> at <velocity> End of phase 1. |
| Choose a closer base station to contact. Send XY to the base station; | <cellphone #>: communicating with base station #. |
| Receiving ACK from a base station | <cellphone #>: closer to the target. LQA is ... Current battery life is |
| Receiving NACK from a base station | <cellphone #>: further to the target. LQA is ... Current battery life is |
| If the battery needs to be recharged | <cellphone #>: recharging battery life |
| If counter > 3 | <cellphone #>: moved 3 times without connecting to the target |
| If out of range | <cellphone #>: outside borders |
| Receiving PORT from a base station | <cellphone #>: has made contact with the target. receive TARGET's port # End of phase 2. |
| Upon creating TCP socket | <cellphone #>: TCP Port |
| Sending XY to the target | <cellphone #>: connecting to the target |
| Receiving ACK from the target | <cellphone #>: the distance to the target is |
| Close the corresponding TCP socket | <cellphone #>: End of Phase 3. |

Table 4: Base Station on-screen messages

| Event | On-screen Message |
|-------------------------------|--|
| Upon Startup | Base Station has UDP Port and IP address |
| Receiving XY from a cellphone | <cellphone #> is now at<x,y>. Its previous distance to the target was.... |
| Sending ACK to a cellphone | <cellphone #> is closer to the Target. Its current distance to the target is .. |
| Sending NACK to a cellphone | <cellphone #> is farther. Its current distance to the target is ... |
| Sending PORT to a cellphone | Sending TCP port # to <cellphone #> |

Table 5: Target on-screen messages

| Event | On-screen Message |
|-------------------------------|--|
| Upon Startup | <Target> has TCP Port and IP address |
| Receiving XY from a cellphone | <Target>: <cellphone #> is at (x,y), <distance> from the target. <you can be creative here> |

Assumptions:

1. The Processes are started in the order that you believe is necessary or appropriate. Please mention it in your README file.
2. You are allowed and required to insert delays into your code e.g. by using sleep().
3. TCP port number of the target is hardcoded in each base station and in the target.
4. UDP port numbers for base stations are hardcoded in each cell phone units and the corresponding base station.
5. Each message starts with the identifier string for the sender of that packet.
6. Messages of type PORT carry the TCP port number of target as the payload.
7. If you need to have more code files than the ones that are mentioned here, please use meaningful names and all small letters and **mention them all in your README file.**
8. You are allowed to use blocks of code from Beej's socket programming tutorial (Beej's guide to network programming) in your project.
9. When you run your code, if you get the message "port already in use" or "address already in use", please first check to see if you have a zombie process (from past logins or previous runs of code that are still not terminated and hold the port busy). If you do not have such zombie processes or if you still get this message after terminating all zombie processes, try changing the static UDP or TCP port number corresponding to this error message (all port numbers below 1024 are reserved and must not be used). If you have to change the port number, **please do mention it in your README file**

Requirements:

1. Each entity must display its TCP and/or UDP port number and IP address on screen upon startup or later during the runtime of project.
2. Refer to Table1 to see which ports are statically defined and which ones are dynamically assigned. Use *getsockname()* function to retrieve the locally-bound port number wherever ports are assigned dynamically as shown below:

```

//Retrieve the locally-bound name of the specified socket and store it in the sockaddr structure
getsock_check=getsockname(TCP_Connect_Sock,(struct sockaddr *)&my_addr, (socklen_t *)&addrlen) ;
//Error checking
if (getsock_check== -1) {
    perror("getsockname");
    exit(1);
}

```

3. Use *gethostbyname()* to obtain the IP address of *nunki.usc.edu* or the local host. However the host name must be hardcoded as *nunki.usc.edu* or *localhost* in all pieces of code.
4. You can either terminate all processes after completion of phase3 or assume that the user will terminate them at the end by pressing *ctrl-C*.
5. All the naming conventions and the on-screen messages must conform to the previously mentioned rules.
6. All messages sent through sockets must start with an identifier string as instructed in the project description.
7. You are not allowed to pass any parameter or value or string or character as a command-line argument. No user interaction must be required (except for when the user runs the code obviously). Every thing is either hardcoded or dynamically generated as described before.
8. All the on-screen messages must conform exactly to the project description. You should not add anymore on-screen messages. If you need to do so for the debugging purposes, you must comment out all of the extra messages before you submit your project.
9. Using *fork()* or similar system calls are not mandatory if you do not feel comfortable using them to create concurrent processes.
10. Please do remember to close the socket and tear down the connection once you are done using that socket.

Programming platform and environment:

1. All your codes must run on *nunki* (*nunki.usc.edu*) and only *nunki*. It is a SunOS machine at USC. You should all have access to *nunki*, if you are a USC student.
2. You are not allowed to run and test your code on any other USC Sun machines. This is a policy strictly enforced by ITS and we must abide by that.
3. No MS-Windows programs will be accepted.
4. You can easily connect to *nunki* if you are using an on-campus network (all the user room computers have *xwin* already installed and even some *ssh* connections already configured).
5. If you are using your own computer at home or at the office, you must download, install and run *xwin* on your machine to be able to connect to *nunki.usc.edu* and here's how:
 - a. Open *software.usc.edu* in your web browser.
 - b. Log in using your username and password (the one you use to check your USC email).
 - c. Select your operating system (e.g. click on windows XP) and download the latest *xwin*.
 - d. Install it on your computer.
 - e. Then check the following webpage: <http://www.usc.edu/its/connect/index.html> for more information as to how to connect to USC machines.
6. Please also check this website for all the info regarding "getting started" or "getting connected to USC machines in various ways" if you are new to USC: <http://www.usc.edu/its/>

Programming languages and compilers:

You must use only C/C++ on UNIX as well as UNIX Socket programming commands and functions. Here are the pointers for Beej's Guide to C Programming and Network Programming (socket programming):

<http://www.beej.us/guide/bgnet/>

(If you are new to socket programming please do study this tutorial carefully as soon as possible and before starting the project)

<http://www.beej.us/guide/bgc/>

Once you run xwin and open an ssh connection to nunki.usc.edu, you can use a unix text editor like emacs to type your code and then use compilers such as g++ (for C++) and gcc (for C) that are already installed on nunki to compile your code. You must use the following commands and switches to compile yourfile.c or yourfile.cpp. It will make an executable by the name of "yourfileoutput".

```
gcc -o yourfileoutput yourfile.c -lsocket -lnsl -lresolv
g++ -o yourfileoutput yourfile.cpp -lsocket -lnsl -lresolv
```

Do NOT forget the mandatory naming conventions mentioned before!

Also inside your code you need to include these header files in addition to any other header file you think you may need:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/wait.h>
```

Submission Rules:

1. Along with your code files, include a **README file** which is plain text file (NOT a word document or a pdf file). In the **README** file please write
 - a. Your **Full Name** as given in the class list
 - b. Your **Student ID**
 - c. What you have done in the assignment
 - d. What your code files are and what each one of them does. (Please do not repeat the project description, just name your code files and briefly mention what they do).
 - e. What the TA should do to run your programs. (Any specific order of events should be mentioned.)
 - f. The format of all the messages exchanged (other than what is mentioned in the project description).
 - g. Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
 - h. Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.)

Submissions WITHOUT README files WILL NOT BE GRADED.

2. Compress all your files including the README file into a single “tar ball” and call it: **ee450_yourUSCusername_session#.tar.gz** (all small letters). For example, my file name would be **ee450_chen2_session1.tar.gz**. Please make sure that your name matches the one in the class list. Here are the instructions:
 - a. On nunki.usc.edu, go to the directory which has all your project files. Remove all executable and other unnecessary files. Only include the required source code files and the README file. Now run the following commands:
 - b. **you@nunki>> tar cvf ee450_yourUSCusername_session#.tar *** - Now, you will find a file named “ee450_yourUSCusername_session#.tar” in the same directory.
 - c. **you@nunki>> gzip ee450_yourUSCusername_session#.tar** – Now, you will find a file named “ee450_yourUSCusername_session#.tar.gz” in the same directory.
 - d. Transfer this file from your directory on nunki.usc.edu to your local machine. You need to use an FTP program such as CoreFtp to do so. (The FTP programs are available at software.usc.edu and you can download and install them on your windows machine.)
3. Upload “ee450_yourUSCusername_session#.tar.gz” to the Digital Dropbox (available under Tools) on the DEN website. After the file is uploaded to the dropbox, you must click on the “**send**” button to actually submit it. If you do not click on “**send**”, the file will not be submitted.
4. Right after submitting the project, send a one-line email to your designated TA (NOT all TAs) informing her that you have submitted the project to the Digital Dropbox. Please do NOT forget to email the TA or your project submission will be considered late and will automatically receive a zero.
5. You will receive a confirmation email from the TA to inform you whether your project is received successfully, so please do check your emails well before the deadline to make sure your attempt at submission is successful.
6. You must allow at least 12 hours before the deadline to submit your project and receive the confirmation email from the TA.
7. By the announced deadline all Students must have already successfully submitted their projects and received a confirmation email from the TA.
8. Please take into account all kinds of possible technical issues and do expect a huge traffic on the DEN website very close to the deadline which may render your submission or even access to DEN unsuccessful.
9. Please do not wait till the last 5 minutes to upload and submit your project because you will not have enough time to email the TA and receive a confirmation email before the deadline.
10. Sometimes the first attempt at submission does not work and the TA will respond to your email and asks you to resubmit, so you must allow enough time (12 hours at least) before the deadline to resolve all such issues.
11. **You have plenty of time to work on this project and submit it in time hence there is absolutely zero tolerance for late submissions! Do NOT assume that there will be a late submission penalty or a grace period. If you submit your project late (no matter for what reason or excuse or even technical issues), you simply receive a zero for the project.**

Grading Criteria:

Your project grade will depend on the following:

1. Correct functionality, i.e. how well your programs fulfill the requirements of the assignment, specially the communications through UDP and TCP sockets.
2. Inline comments in your code. This is important as this will help in understanding what you have done.
3. Whether your programs work as you say they would in the README file.
4. Whether your programs print out the appropriate messages and results.
5. If your submitted codes, do not even compile, you will receive 10 out of 100 for the project.
6. If your submitted codes, compile but when executed, produce runtime errors without performing any tasks of the project, you will receive 10 out of 100.
7. If your codes compile but when executed only perform phase1 correctly, you will receive 20 out of 100.
8. If your codes compile but when executed perform only phase1 and phase 2 correctly, you will receive 70 out of 100.
9. If your code compiles and performs all tasks in all 3 phases correctly and error-free, and your README file conforms to the requirements mentioned before, you will receive 100 out of 100.
10. If you forget to include any of the code files or the README file in the project tar-ball that you submitted, you will lose 5 points for each missing file (plus you need to send the file to the TA in order for your project to be graded.)
11. You are encouraged to edit your code and readme file on a unix/linux machine. If you work on Windows and ftp the code to nunki, remember to use proper data format. You will lose 5 points if your code and/or readme files are not transferred to nunki properly.
12. If your code does not correctly assign and print out the TCP or UDP port numbers dynamically (in any phase), you will lose 20 points.
13. You will lose 5 points for each error or a task that is not done correctly.
14. The minimum grade for an on-time submitted project is 10 out of 100.
15. There are no points for the effort or the time you spend working on the project or reading the tutorial. If you spend about 2 months on this project and it doesn't even compile, you will receive only 10 out of 100.
16. Using fork() or similar system calls are not mandatory however if you do use fork() or similar system files in your codes to create concurrent processes (or threads) and they function correctly you will receive 10 bonus points.
17. If you submit a makefile or a script file along with your project that helps us compile your codes more easily, you will receive 5 bonus points.
18. The maximum points that you can receive for the project with the bonus points is 100. In other words the bonus points will only improve your grade if your grade is less than 100.
19. Your code will not be altered in any ways for grading purposes and however it will be tested with different input files. Your designated TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not.

Cautionary Words:

1. Start on this project early!!!

2. In view of what is a recurring complaint near the end of a project, we want to make it clear that the target platform on which the project is supposed to run is *nunki.usc.edu*. It is strongly recommended that students develop their code on nunki. In case students wish to develop their programs on their personal machines, possibly running other operating systems, they are expected to deal with technical and incompatibility issues (on their own) to ensure that the final project compiles and runs on nunki.

3. You may create zombie processes while testing your codes, please make sure you kill them every time you want to run your code. To see a list of all zombie processes even from your past logins to nunki, try this command:

```
ps -aux | grep <your_username>
```

4. Identify the zombie processes and their process number and kill them by typing at the command-line:

```
kill -9 processnumber
```

5. There is a cap on the number of concurrent processes that you are allowed to run on nunki. If you forget to terminate the zombie processes, they accumulate and exceed the cap and you will receive a warning email from ITS. Please make sure you terminate all such processes before you exit nunki.

6. Please do remember to terminate all zombie or background processes, otherwise they hold the assigned port numbers and sockets busy and we will not be able to run your code in our account on nunki when we grade your project.

Academic Integrity:

All students are expected to write all their code on their own.

Copying code from friends is called **plagiarism** not **collaboration** and will result in an F for the entire course. Any libraries or pieces of code that you use and you did not write must be listed in your README file. All programs will be compared with automated tools to detect similarities; examples of code copying will get an F for the course. **IF YOU HAVE ANY QUESTIONS ABOUT WHAT IS OR ISN'T ALLOWED ABOUT PLAGIARISM, TALK TO THE TA.** "I didn't know" is not an excuse.