Cell Tower Tracking

6.883 – 5508 Problem Set 4

1 Background

There are 27 satellites (24 plus 3 spares) all orbiting in their own patterns, but making two complete rotations each day, so that everyone can always see at least 4 of them from anywhere on earth (that seems like a fun puzzle). So, perhaps, 24 hours later we will see the same set (maybe).

2 Cell Tower Tracking

This assignment is to see if we can collectively map out where there is cell coverage and how good it is. Many operators publish their own maps as to the quality of cell coverage. No one knows if these are accurate. There are commercial services that will sell the location of cell towers.

We have gps and telephones and servers. We can build up our own maps and make them public. If done well, others around the world can contribute as well.

In S60 python, there is a location package that returns cell tower information and there is a call that returns the signal strength in the sysinfo package:

```
a,b,c,d = location.gsm_location()
ss = sysinfo.signal()
```

2.1 The Main Idea

The idea is to take readings, say every 5 min when you are roaming around, and keep a record. Then, periodically upload it to our server. Soon we will then have a reliable database.

```
<raw_tower_info>
  <id> larry </id>
  <gsm_location> a, b, c, d </gsm_location>
  <time> time of reading -- in gps universal time format </time>
  <signal_strength> number in range 0 to 7 </signal_strength>
  <gps_location> lat, lon, altitude </gps_location>
</raw_tower_info>
```

These records are then uploaded to the database, where they can be combined with the cell tower info. Uploading does not have to be done immediately, but at least once a day. (you can move information from phone to pc and then to server if your phone does not have data conneectivity).

The server accepts two commands. The first is "store" and the parameter is a record as above. The second is "load" and the parameter is a set of cell towers. The set is just a string in python style of "[(a, b, c, d) , (a1,b1,c1,d1), ..]" The server will return a set of gps locations and signal strengths that satsify all the cell towers.

2.2 Server Details

The server script for this part of the assignment is at http://ozone.csail.mit.edu/upload_cell_tower.php (source code at http://ozone.csail.mit.edu/upload_cell_tower.phps). As described above, there are two commands the server accepts - "store" and "load". These are read as form parameter names and associated values by the script, as if an html form post was made. Note that you can specify both fields in one request, in which case a store and load are performed, and the response will be whatever the output of the load request was. When sending request, you'll need to use http post with multipart form data (see the code snippets above for how to do this).

For the "store" request, the server expects the parameter's value to be an XML string of the form:

```
<?xml version="1.0"?>
<records>
   <raw_tower_info>
     <id> unique id </id>
     <gsm_location> a,b,c,d </gsm_location> <!-- Be wary of whitespace -->
     <time> time of reading -- in gps universal time format </time>
     <signal_strength> number in range 0 to 7 </signal_strength>
     <gps_location> lat,lon,altitude </gps_location>  <!-- e.g., 3432.343N,0384.333W,23.43M -->
   </raw_tower_info>
   <raw_tower_info>
     ...
   </raw_tower_info>
   ...
</records>
```

For the "load" request, the server expects the parameter's value to be the python-style string above ("[(a,b,c,d),...]"). The output of the request is a gzipped XML string of the same form as the store request. That is,

```
<?xml version="1.0">
<records>
  <raw_tower_info>
    <id>...<id>
    <gsm_location>...<gsm_location>
    <time>...</time>
    <signal_strength>...</signal_strength>
    <gps_location>...</gps_location>
  </raw_tower_info>
  <raw_tower_info>
    ...
  </raw_tower_info>
  ...
</records>
```

3 Extensions

The following are things that are "extra credit" – stuff that I would like to see but not part of the basic assignment.

- Feel free to develope your own server that might provide additional services, such as a map of the current location.
- algorithm that tries to approximate the data with hexagonal shapes that outline the actual cells.

4 What to hand in

The servers will have your data, but hand in

- your code
- you id specifier
- a write-up as to how this application might be power efficient. In otherwords,
- what can be done to minimize the power requirements. This is a big deal if one wants it to be runing all the time.

More details

We will all be taking several readings, hopefully during at least 3 of the four time periods: Monday night at 11:00 pm, Tuesdday morning at 11:00 am, Tuesday night at 11:00 pm, and Wednesday morning at 11:00 am. (All times are Boston based)

There has been a change in the directions (hopefully, you will see this in time). Original directions:

"Each period will last 10 min, and you should collect a reading each second. All 600 readings should be uploaded to our server."

It is better if you record all the information that the gps provides and forget about parsing it on the handset. Collect every byte that is sent by the gps receiver and dump it into a file. The file can be parsed locally and the normal record sent to the server.

Then at the end of the period, we will upload the whole file (or up to 4 files).

Of course not everyone can do all readings since you are sharing devices. Just work it out who is doing which period. [edit] Server Details

The best way to do the uploads is to save your readings to a file, and then upload the file at a convenient time. Our server script at ozone.csail.mit.edu/upload_global_gps.php (source code at http://ozone.csail.mit.edu/upload_gl expects a pipe-delimited file to be uploaded, with one reading per line. Each line should look like the following:

```
<ID>|<location>|<time>|<latitude>|<latitude_direction>|<longitude>|<longitude_direction>|
<num_satellites>|<hdop>|<altitude>|<height above WGS84>
```

As you can see, almost every element of a gps reading is to be recorded. The ID should be something unique that you can identify (e.g., username). The location should either be "Cambridge" or "Singapore". Note that for latitude and longitude, specify the value and the direction separately. The last two entries, altitude, and height above WGS84 ellipsoid, are expected to be in meters. Here's an example entry (values taken from lecture slide on gps):

```
nsong|Cambridge|170834|4124.8963|N|08151.6838|W|05|1.5|280.2|-34.0
...
```

The table in the database looks like this:

```
+---------------------+------------------+------+-----+---------+-------+
| Field               | Type             | Null | Key | Default | Extra |
+---------------------+------------------+------+-----+---------+-------+
| id                  | varchar(25)      | NO   |     |         |       |
| location            | varchar(25)      | YES  |     | NULL    |       |
| time                | int(10) unsigned | YES  |     | NULL    |       |
| latitude            | double unsigned  | YES  |     | NULL    |       |
```

```
| latitude_direction  | varchar(5)       | YES  |     | NULL    |       |
| longitude           | double unsigned  | YES  |     | NULL    |       |
| longitude_direction | varchar(5)       | YES  |     | NULL    |       |
| satellites          | int(10) unsigned | YES  |     | NULL    |       |
| hdop                | double           | YES  |     | NULL    |       |
| altitude            | double           | YES  |     | NULL    |       |
| height_WGS84        | double           | YES  |     | NULL    |       |
+---------------------+------------------+------+-----+---------+-------+
```

There are two ways to upload to the server: the cheap way, and the python way. The script itself contains a simple form which you can use to manually upload your files to the server. To do it within python, you may find the following helpful (the name for the file is "gpsreadings"):

```
import httplib
import mimetypes

def post_multipart(host, selector, fields, files):
    """
    Post fields and files to an http host as multipart/form-data.
    fields is a sequence of (name, value) elements for regular form fields.
    files is a sequence of (name, filename, value) elements for data to be uploaded as files
    Return the server's response page.
    """
    content_type, body = encode_multipart_formdata(fields, files)
    h = httplib.HTTPConnection(host)
    h.putrequest('POST', selector)
    h.putheader('content-type', content_type)
    h.putheader('content-length', str(len(body)))
    h.endheaders()
    h.send(body)
    response = h.getresponse()
    return response.read()

def encode_multipart_formdata(fields, files):
    """
    fields is a sequence of (name, value) elements for regular form fields.
    files is a sequence of (name, filename, value) elements for data to be uploaded as files
    Return (content_type, body) ready for httplib.HTTP instance
    """
    BOUNDARY = '----------ThIs_Is_tHe_bouNdaRY_$'
    CRLF = '\r\n'
    L = []
    for (key, value) in fields:
        L.append('--' + BOUNDARY)
        L.append('Content-Disposition: form-data; name="%s"' % key)
        L.append('')
        L.append(value)
    for (key, filename, value) in files:
        L.append('--' + BOUNDARY)
        L.append('Content-Disposition: form-data; name="%s"; filename="%s"' % (key, filename))
        L.append('Content-Type: %s' % get_content_type(filename))
        L.append('')
```

```
        L.append(value)
    L.append('--' + BOUNDARY + '--')
    L.append('')
    body = CRLF.join(L)
    content_type = 'multipart/form-data; boundary=%s' % BOUNDARY
    return content_type, body

def get_content_type(filename):
    return mimetypes.guess_type(filename)[0] or 'application/octet-stream'
```

4.1 Comments:

This problem set has a bunch of parts. Each is small, but putting them together can be complicated. Please spend some time planning overall flow of the application. It can be extended in many ways, and I encourage you to think about extensions.

The pieces, are fairly independent. So they can be worked on as separate modules. Try to generate and display content, without the triggers first.