CITS320 Computer Networks: Project Documentation

1 Instructions

To construct the geoserver code, change the working directory to the geoserver directory. Inside the directory, use an open terminal and call the command

```
make geo
```

This will compile the relevant object files and binary. The binary file for the geoserver is called geoserver, which takes two arguments: the port number which the geoserver will listen on, and the file with which all known WAPs have been calculated, and which will be used to store the data. This file will be overwritten during the geoserver's runtime. It should contain its data in csv format. The format for a line of the csv data is: SSID,latitude,longitude,numberOfPreviousSamples

An example invocation is

```
./geoserver 1830 test.csv
```

Here the geoserver will listen on port 1830 and read/write sample data from/to test.csv. To compile with all debug statements, uncomment `#define VERBOSE` in `trilaterate.h` and recompile.

2 geoserver Algorithm

The overall purpose of the geoserver is to compute locations and provide images to the client iPod application. As such, its primary algorithm is the calculation of the location of the client iPod given sets of wi-fi beacon records, accomplished via trilateration. Other functionality leveraged for improved efficiency includes caching.

2.1 Location Determination via Distance from Access Points

As per the project specification, the app must provide approximate geographic and Wireless Access Point (WAP) signal information to the geoserver so that the geoserver may calculate the client app's position. Essentially, the WAP signal information must be converted into an approximation of the distance from the listening iPod app to the sender, before any further calculations can be performed. Two primary approaches to utilising WAP signal information for location determination were considered: smoothed round trip time (SRTT) and received signal strength indicator (RSSI).

A phased-base approach using **SRTT** could work because the constant speed of electromagnetic radiation would allow an immediate conversion from trip time to distance. However, this approach required that the WAP and listener have synchronised clocks in order to calculate SRTTs, and due to the extremely small times involved, the synchronizations must be very precise. This in turn requires significant amounts of communication between the listener and _each_ WAP within range, either concurrently or in sequence. In both cases, this could be cumbersome for the app in terms of congestion and flow management, and for the WAPs as well if the system were scaled up to multiple iPod apps.

RSSI can provide a distance calculation because the strength of a signal of electromagnetic radiation decreases as a function of the distance travelled by the inverse square law. And yet this too has some inherent problems:

- Changes in signal strength can also be due to changes in wind, air humidity and most notably, physical obstacles
- RSSI can be a vendor specific measurement and will most likely change between different devices. Converting between RSSI and distance was difficult, as no "vendor model"/wireless card specification was available publicly. Empirically measuring this data was unsuccessful due to the random nature of the measurements and the difficulty in estimating the resulting relation between RSSI and distance.
- Beacon signals in UWA implement a form of beacon strength regulation, causing the RSSI to fluctuate as the demand upon a WAP increases or decreases.

In considering the above two approaches, RSSI was found to be more advantageous to the circumstances. By using RSSI to obtain a distance to multiple WAPs, the location of the listener itself can be determined through trilateration.

2.2 Trilateration

Trilateration is the algorithm used in GPS to determine one's position based on what indicators (signal sources) can be heard. Suppose that a GPS device finds that an indicator can be "heard" with a known RSSI (or SRTT, but RSSI is used in this example). As mentioned above, signal strength decreases over distance by a known factor, the signal strength can be converted to a measurement of distance, *d* metres. Then the position the device is listening from must be *d* metres away from this indicator, meaning the device could be at any position along the circumference of a circle of radius *d* (shown in red, with the black central dot being the source of the signal) in from the indicator.

If a second indicator can be heard with a strength of signal which corresponds to a distance of *d2* metres, then a second circle of radius *d2* centred at the location of the second indicator of possible locations is known (shown in yellow), and thus the device must be listening at either one of the two points (1 & 2) at which the circumferences of these two circles intersect. With a third indicator circle (shown in blue), the listening device's location can be found as the intersection of the three circles' circumferences, point 2, as the third circle does not reach point 1. This works best if the geometry is guaranteed to occur on a single 2D plane. See Trilateration on Wikipedia for more details on the equations used.

An analogous situation exists for the iPod app: the iPod app client is a "listening point" and a WAP is an "indicator", whilst the RSSI is the strength of the signal to be converted to a distance. Yet the client's position can only be determined relative to the positions of the WAPs, so the WAP positions must also be determined. This can be done by using the same process in reverse, with an _approximate_ location input from the user: multiple samples are taken from We can use trilateration to find the position of each WAP based upon multiple sample measurements, and use trilateration again to find the position of a listener based upon what WAPs it can hear.

2.3 Trilateration Algorithm Summary

1. For a WAP identified by MAC address m, there are *n* samples associated with it, where *n* >= 20
2. Perform (*n* choose 3) trilateration calculations
3. Find the centroid *c* of the set of points produced by *s* successful trilateration calculations
4. If a previous centroid *p* had been calculated using *k* points, then the final calculated position is weighted according to the formula (*kp* + *sc*) / (*k* + *s*)
5. Discard all *n* of the collected samples so that they will not biased (i.e. be able to be reused in) future calculations

When determining one's location based on the WAPs that can be heard, steps one to three are performed whilst steps four and five are irrelevant. This is due to the fact that no previous calculation of your current location can be made, and thus there are no previous points to utilise or previous samples to use. If trilateration fails or no sample points can be collected from the algorithm, the algorithm simply keeps its previously associated samples and continues processing data. This is because the samples have not been effectively used and so discarding them will be wasteful.

2.4 Trilateration: Reflection

We encountered some errors when using our own data for trilateration, and found that the values we were getting did not seem to make sense; every point was considered co-linear as a result. In addition, it was extremely difficult to send to the photoserver due to the unreliability of the university network. In order to construct some sample data, we used some of the information gathered by Michael Froend and Alice McCullagh to form our own sample data through an offline trilateration calculation.

We found that our trilateration algorithm worked but that due to differences in iPod specifications and wireless specifications, a poor model and an assumption that RSSI needed to be converted to distance and thus the compounding of error in our sample data, our program did not work as intended. Some possible solutions that we were informed of included discarding distance in favour of ratios, relaxing some of the constraints on when and when not to trilaterate and performing more offline calculations due to the problems with using the network to always connect to our geoserver.

2.5 Caching

Whenever the geoserver receives a request for an image of a particular view (location+direction), it checks an internal cache of views. If the requested view has been cached and the cached image is not too old, the geoserver simply sends the requesting client the cached image. Images which are not cached or too old are fetched from the photoserver, immediately cached, and then sent to the client. A curious note: the photoserver appears to be approximately 2-3 seconds in the future, according to its timestamps.

When full, the cache utilises a naïve cyclical replacement strategy: the cache item overwritten is the 'oldest', in terms of the order of writing. A more efficient and scalable solution would be to overwrite item which was accessed the longest time ago, however the gains from this solution would only apply if the geoserver received a significantly larger amount of traffic.

In future, location-based caching (checks for significant movement before requesting a new image) could be done on the iPod to reduce unnecessary transmissions, though has the cost of a greater workload on the iPod. Should the iPod's battery life or CPU use be more valuable than the saturation of the transmission medium and use of the server CPU, this could instead be done on the geoserver.

3 geoserver – iPod Communication

Please see GeneralAlgorithmFlow.png in this directory for General Algorithm Flowchart.

3.1 Protocol

We used TCP protocols with IPv4 addressing and streaming sockets. This meant that we were constantly able to check for errors and were able to use bidirectional communication in the client-server relationship. While streaming-sockets meant that we had to create and destroy socket connections often, it also allowed data to be sent in an ordered fashion, making it a worthwhile tradeoff.

The first 4 bytes sent from the iPod app notify the receiver of the `packetType:` whether sending WAP sample data or requesting a photo. The app sends a single sample each time, with each sample consisting of latitude, longitude, number of APs heard, and each AP's information itself. When the app sends a photo request, the geoserver responds with 0 if no photo is available, otherwise it sends back the size of the image followed by the binary data of that size. The geoserver knows how many bytes to read whenever it receives communication from the iPod app because the size of each iPod app communication is constant and defined in the geoserver's header file.

Since most of the data sent across were multi-byte types, the sockets were wrapped in file streams using the fdopen() function. This allowed the program to work with buffered I/O, so that it could easily send data one byte at a time and not have to worry about the byte ordering. Even though TCP guarantees that it will send all data in the correct order, it does not guarantee that every recv will be of equal size. Thus, one cannot know in advance how many times one will need to call the function. Using file streams allowed the use of inbuilt buffers, so that cumbersome loops were not needed*. It also gave us the choice, although we never needed it, to put a byte back into a stream after we had read it, using ungetc(). We were not very familiar with using file streams and so developing protocols with them introduced us to new bugs and different problems.

*Such loops were utilised in the photoserver communications component of the geoserver to strong effect, though the ease of this was not realised until late in the project, when it was too late to refactor the file stream code.

3.2 Blocking

While non-blocking would be innately preferable in dealing with multiple connections, the project specification calls for a single geoserver/iOS application connection pairing. For future extension and scalability, a change to non-blocking would be recommended.