Joshua Groppe

CPSC 481/CyberTiger Project Proposal

27 January 2012

Motivations

After having done some amount of mobile software development on a co-op rotation I have found a passion for mobile programming, and specifically iOS development. Mobile technology is a huge part of our society today, and continues to grow at a significant rate. I am very interested in expanding my knowledge and exploring the many facets of this area of computing, and in particular I am eager to learn more about the networking side of mobile applications and how it can be utilized to develop programs of quality.

Background

The CyberTiger team has built a platform-independent network testing framework. It gathers data from both Android mobile devices as well as non-mobile computers. The data is used to create a visualization of the test results that is intended to provide meaningful information about the performance and coverage of Clemson’s wireless infrastructure, as well as the infrastructures of commercial cell data networks. While the existing framework covers a significant portion of smartphone users, the extension of the project to iOS devices will increase the project’s scope of testable mobile devices to over 75% of smartphone users and 90% of tablet users.

Objectives

First, I will familiarize myself with the existing system via the information on the project wiki and the current source code. In order to attain a working understanding, as well as to benefit future developers, I will create documentation summarizing the design of the tools framework and how it can be implemented. I will maintain and add to this documentation as my part of the project develops.

I will create a simple iOS user interface and utilize it to gather the collection of device, network, and location metrics that are needed in order to send a request message to the existing server. This includes information about the device itself (hardware address, model, etc.) the geolocation of the device (longitude, latitude, etc.), and the network being utilized by the device (type, signal strength, etc).
Using the gathered information, I will be able to construct a request that will be sent over TCP to the server asking for a test to be performed. From there I will implement the basic GeoLocation tool, and the TCP bandwidth test.

**Methodology**

Behind the basic user interface an inheritance structure will be implemented at the core of the application to minimize redundant code. A generic test case will be used for the GeoLocation tool, and each of the other tests will extend its basic functionality for their individual specialization.

The initial request and transmission of metrics will be done over TCP, and the both tests that I plan to implement (GeoLocation, TCP bandwidth) will use this same connection.

To serialize the collected data for each of the different tests and communication instances, I will utilize Google’s protocol buffers; a flexible, efficient, automated mechanism which is standard across each instance of the framework.

**Expected Outcomes/Results**

By the end of the semester, the program should be a fully functional and deployable iOS application. It should contain working GeoLocation and TCP bandwidth tests. The documentation that was created at the beginning of the semester should contain the level of information necessary to bring a new developer up to speed on the project and the operating logic of the framework. By extending the existing framework to iOS we will be broadening the scope of the project to a much wider range of student users and increasing the number of testable devices.

**Deliverables**

Milestone 1: Proposal

Milestone 2: Framework design/implementation documentation complete.

Milestone 3: User interface of iOS application complete.

Milestone 4: Metrics and status information sent in request to server and reply received.

Milestone 5: TCP bandwidth test successfully performed.

Milestone 6: Final Submission
Introduction

This documents the development process for the CyberTiger iOS application with respect to the original deliverables. For each milestone it details exactly what was done to reach the goal, the testing performed, and difficulties encountered. Finally, the development is evaluated with respect to future developments.

Milestone 3 Summary

My goal for milestone 3 was to have completed the basic interface of the iOS application. This meant building both the visual and the functional elements of the iPhone/iPad interface. The Xcode SDK makes it relatively simple to design user interfaces. A functional interface consists of a storyboard (the graphical portion) and a view controller which is connected to the storyboard. Interface elements on the storyboard such as buttons are linked to objects and/or methods in the view controller. Figure 1 below shows the storyboard and view controller files in Xcode, as well as the interface builder.

Figure 1: iOS User Interface Builder
Figure 2 shows an example of methods within the view controller that are linked to the buttons on the storyboard.

Figure 2: View Controller Functions

```objective-c
-(IBAction)iPhoneGeoButtonAction:(id)sender
{
    GenericTest *test = [[GenericTest alloc] init];
    [test execute];
    [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:TRUE];
}
-(IBAction)iPhoneUDPButtonAction:(id)sender
{
}
-(IBAction)iPhoneTCPButtonAction:(id)sender
{
    TCPBandwidthTest *test = [[TCPBandwidthTest alloc] init];
    [test execute];
    [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:TRUE];
    [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:FALSE];
}
-(IBAction)iPhoneStreamingButtonAction:(id)sender
{
}
-(IBAction)PadGeoButtonAction:(id)sender
{
    GenericTest *test = [[GenericTest alloc] init];
    [test execute];
}
-(IBAction)PadUDPButtonAction:(id)sender
{
}
-(IBAction)PadTCPButtonAction:(id)sender
{
    TCPBandwidthTest *test = [[TCPBandwidthTest alloc] init];
    [test execute];
    [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:TRUE];
    [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:FALSE];
}
-(IBAction)PadStreamingButtonAction:(id)sender
{
    @end
}
```

This milestone was completed successfully. The two figures below are screenshots of the basic interfaces on both the iPhone and the iPad.
Testing: The testing performed for this milestone was to ensure that the buttons were functional and triggered the appropriate methods when touched.

Remaining Issues: None
Milestone 4 Summary

My goal for milestone 4 was to have successfully performed a geolocation test. Reaching this milestone involved gathering the necessary metrics, integrating the Protocol Buffers library, investigating and deciding upon a communication option, and constructing the test itself.

I created a Singleton object called the Collector that would act as a centralized controller for the gathering of the metrics. Because the information that it needs to collect can be grouped into three categories (device, network, and location) I divided up the collection process into three stages, respective to those groups. I did this so that the collection process could be more modular, which, because of nature of multithreading on iOS, allows me to keep the user updated on the progress of the test at more frequent intervals. The object in iOS which is used as a dialog is called a UIAlertView. If a developer creates an alert view within a method, sets its properties (title, message, buttons, etc.) and then ‘reveals’ it, it cannot change dynamically change those properties within that same method.

The device metrics were easily accessed by means of the UIDevice class, which is a global object available for use at any time. I was able to collect all of the information that I needed by accessing the class’s properties; namely the make, model, system name, and system version.

In order to gather the necessary location information, I needed to make use of the CLLocationManager. This object delivers location updates to other objects which have registered themselves to receive notifications from the location manager and which have implemented certain methods. So I made my LocationInfo class a delegate of the location manager, and then signaled it to begin to send out updates. Once it receives this signal the manager immediately sends out an update of the current location. Because I implemented the delegate method didUpdateToLocation in my custom class, I receive this update. From there I was able to collect the latitude, longitude, altitude, and accuracy directly. However, there is no direct way of determining the geolocation method. Because at this point I have only been able to test the app on an iTouch (which has no gps) I am currently setting it to triangulation by default.

The metrics that remained to be gathered were those relating to the network, and proved to be the biggest challenge. I was able to attain the carrier, the SSID, and the BSSID via built in libraries. These were the CoreTelephony and CaptiveNetwork frameworks. With slightly more effort, I was also able to acquire the IP address and MAC address. There are no built-in methods for directly accessing these, but I was able to manipulate lower-level functions to get the information. This left the signal strength, network type, and link speed. After much research I determined that at several points in the past there had been
undocumented, private Apple frameworks which a developer could access illegitimately to get some of this information. However, in recent years Apple has removed this ability entirely. So far, I have only been able to test the application on an iTouch, the networking capabilities of which is limited to wifi. So, for the present I am manually setting the network type to ‘Wifi’, the link speed to 54 mbps, and the signal strength to null.

The next step was to integrate the Google Protocol Buffers library into my project. It was difficult to find a working version of the compiler that was both up to date and compatible with Objective-C. While the core compiler released by the Google team is controlled, all extensions that support Objective-C are open source projects that have little documentation. After experimenting with several options, I eventually found a solution which required me to install the core compiler first and then install the Objective-C extension on top of it. Using this I was able to compile the protobuf object into Objective-C source. While this extension was the most up to date that I could find, it still produced source code that was two years old. So, with the assistance of Xcode, I made the necessary changes to bring it up into a compatible state. Another issue I faced was with the library itself. Unlike the libraries for C/++ and Java, the one that Google provides for Objective-C is itself a self-contained Xcode project. This confused me at first, but eventually I figured out that I needed to list that project as a dependency in my own. From there I was able to create a protobuf object in my code from the data that I had gathered.

The protocol buffers class in Objective-C provided methods for interacting with Apple’s `ReadStream` family of classes. So the first approach that I took to communicating with the server was an attempt to use Apple’s guidelines on how to do streaming in iOS ‘properly’. However, this method proved to be incompatible with what I was trying to accomplish, and so I settled for using basic socket programming (which is also used the C++ and Java clients). I was able to manipulate the provided methods build and serialize the protocol buffers object to a basic buffer.

From there it was simply a matter of porting the C++ client Geolocation test code (which required very little alteration). I was able to successfully perform the test.

**Testing:** The testing performed for this milestone was as follows:

- Perform the geolocation test in a number of locations all of which had different levels of perceived signal quality, and also in locations where no wifi was available
- Test how the protocol buffer method used to build/serialize the object handled missing information

**Remaining Issues:** Geolocation method, signal strength, network type, and link speed
Milestone 5 Summary

The goal for this milestone was to successfully perform the TCP bandwidth test. Because I chose to use the basic C socket interface for communication, achieving this milestone was, for the most part, simply a matter of porting the C++ client code. I have designed the Objective-C classes that represent the various tests in such a way so that no redundant code exists. That is, the TCP bandwidth test inherits from the Geolocation test; utilizing its methods for initializing and completing the test while overriding its method that contains test-unique functionality.

Testing: As with the geolocation test, I performed the TCP bandwidth test in a number of locations all of which had different levels of perceived signal quality, and also in locations where no wifi was available.

Remaining Issues: None

Conclusion

This semester’s initiative to begin the development of the CyberTiger iOS application has been successful. While there were significant challenges faced at several stages, the majority of them were overcome. We now have a working application which can successfully perform the Geolocation and TCP bandwidth tests over wifi. I believe that the most challenging areas of the development of this application were encountered in milestones 3 and 4, and so future addition of the UDP bandwidth and Streaming tests should not prove to be very difficult.

The only remaining issue from this semester’s work that needs to be addressed is the collection of the geolocation method, network type, link speed, and signal strength. Because I have only had the opportunity to test the application on an iPod Touch, the absence of this data has not had any immediate repercussions. While there does not appear to be any options for gathering this information, more research into the issue should be included in any future work done.

Future Work

As previously stated, the UDP bandwidth and streaming tests need to be added in order to bring this application to the same operating level as its Android counterpart. I do not expect these additions to be overly difficult, and so I suggest that once those tasks have been completed the main focus of any future development should be the robustness and seamless functionality of the app. Further research should also be done on acquiring the missing network metrics. While there is no known way of collecting that information directly, it is possible that techniques could be developed to gather that data without the use of normal methods. Also, the application needs to be tested on an actual iPhone.