Kismet Mobile Client

Robert Bauer

I. Abstract

The goal of this project is to create a Kismet client which could be run from a mobile device, such as an iPod, connect to a Kismet server, and monitor and review the wireless access points processed by the server. Kismet is a Unix/Linux 802.11x protocol analyzer, which has a server and client components. The client component is available to monitor and review the data the server has processed, but is a console based application and difficult to read on a mobile device.

II. Problem

Kismet is a Unix/Linux 802.11x protocol analyzer, which has a server and client components. The client component is available to monitor and review the data the server has processed. The Kismet client is a console based application, which is difficult to read on a mobile device or remote display. The goal of this project is to enable a mobile device, such as an iPhone, to connect to a Kismet server as a client and leverage features available to the native device, such as mapping and a touch interface.

There exists a need to be able to remotely monitor a Kismet server from a mobile device. For instance, Kismet may be used to monitor wireless access points in a given area and trigger an alarm if an unauthorized access point is attached to the network, and to detect this remotely.

For this project, a Kismet server has been configured to be mobile. It resides in a vehicle and can be driven around - logging wireless access points and their GPS coordinates where the signal was detected (much like how Google Maps logs wireless access points to help triangulate a user's position without a GPS signal). The Kismet server is running "headless," meaning it does not have a monitor or LCD screen attached nor an input device. In most cases, it runs fine without interruption or problems. At times, a hardware fault, either on the Kismet server hardware or Kismet drone, may interrupt monitoring. This interruption goes unnoticed until it is time to review the Kismet results. It would be ideal to be able to determine if Kismet is running as well as what it is currently monitoring. Therefore, the goal of this project is to create a Kismet client which could be run from a mobile device, connect to a Kismet server, and monitor and review the wireless access points processed by the server.

Also a problem is mobile devices have limited resources and processing power. A large amount of data must be communicated via a small screen. The device needs to be able to establish a network connection to the Kismet server and establish a conversation with it using the

Kismet protocol. A Wi-Fi connection to the same network the Kismet server is on will be sufficient. An added bonus would be to leverage Google Maps to display access points geographically. Another concern is local storage. It would be ideal to retain the collected access points to local storage. This would allow reviewing the access points at a later time, helpful when not driving.

III. Design

The Kismet setup consists of the following equipment and their role:

- An Asus Eee Box PC running Ubuntu. It runs the Kismet server. Kismet drones and the mobile client communicate with this machine. The Kismet server is configured to get GPS data from the GPS enabled Linksys router.
- One Linksys WRT54G router running as a Kismet drone
- Two Linksys WRTSL54GS routers
    - One router is a Kismet drone with a GPS mounted and running GPSd. The Kismet server polls this device to get the GPS coordinates for logging.
    - The second router is configured as a Wi-Fi bridge. It links the devices on the local network (eeePC, and other routers) to a cellular phone which provides the internet connectivity.

The routers and the PC all have startup scripts that launch the necessary services to start Kismet monitoring.

For this project, the mobile device being used is an Apple iPod. It has Wi-Fi built in, but does not have access to the cellular network. This means the iPod must get internet access from its connected Wi-Fi network. The connected network gets its internet connection from the bridged router and a mobile phone. The iPod connects via Wi-Fi to the bridged router and can access the Kismet server and the internet, including Google Maps.

The application itself should be able to store a collection of Kismet server connections. The user should be able to edit and delete existing connections. The user should be able to tap a connection to connect to a Kismet server. Upon connecting, the application will show the access points as they are received from the Kismet server, with the most recent listed in descending order. This screen is the WAPListView (WAP being Wireless Access Point). From the WAPListView, the user can do one of two actions. The user can click on the action button at the top right and see a list of access points plotted on Google Maps or the user can click on an access point to view the access point details.

The goal of the project was to develop a prototype and if time allowed, to develop a version for the iPad and Android.

IV. IMPLEMENTATION

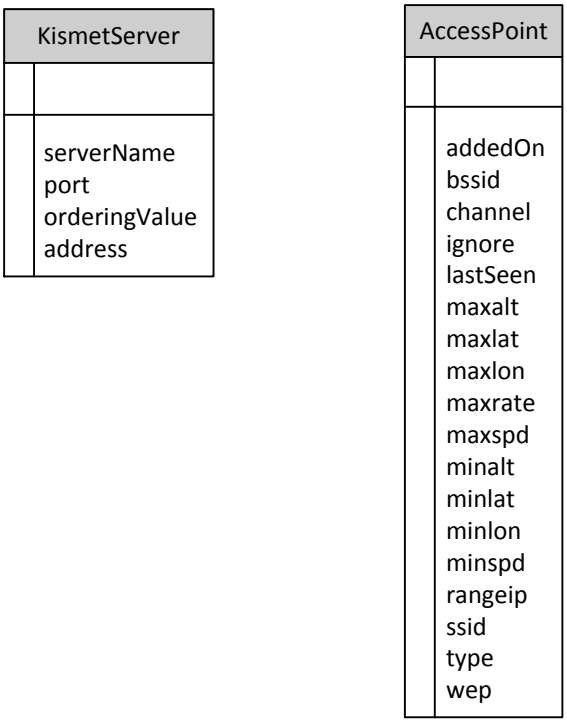
Development for the iPod was done using Apple's XCode development tool. A prototype application was built using Objective-C. GitHub was used as the source code repository. GitHub is also being used to track the project status as well as the open and closed issues [1].

One of the first issues identified was the need to be able to perform integration testing using the mobile app against a Kismet server. The issue is the Kismet server equipment resides in the trunk of a vehicle and is powered off during development. It was also realized if the mobile app is to be submitted in the Apple App Store, they will need to test the application and they will not have a properly configured Kismet server available to test against. To resolve both issues, a Kismet server simulator was built using PHP and is located in GitHub [2]. For security, it does not act on any commands given to it. When executed, it listens for connections. When there is a connection, the script forks and establishes the connection. It then starts mimicking the Kismet server by sending a notification header and then starts sending mock access point data back to the client. This mock server allows testing to take place without the actual Kismet network.

The Kismet mobile application was developed and followed the model-view-controller design pattern recommended by Apple. The following frameworks were used:

- CoreData – used to persist access point data
- MapKit – used for interacting and displaying Google Maps
- CoreLocation – used to get device location
- SystemConfiguration – used to persist Kismet server connections
- AsyncSocket – a third party open source asynchronous socket communications class
- REVClusterMap – a third party open source Google Map's annotation clustering API. This is used to reduce the number of annotations on the map as the map is zoomed out. Through experimenting, the iPod could handle about 600 annotations on a map before suffering performance issues or crashing.

CoreData is an Apple provided data store. It manages the Kismet server connections, which allows the user to enter multiple Kismet servers and save them when the user returns to the application. When the user is connected to a Kismet server, all access points returned to the client are stored using CoreData. The CoreData model is:



The CoreData model contains an one-to-many object relationship between the KismetServer object and AccessPoint object.

The client application establishes a socket connection to the Kismet server and issues commands to setup receiving access point and GPS data. As the data is received, it is parsed and stored in the AccessPoint table. A listener has been setup to listen for changes to this table and update the display. This process continues until the connection is lost or the user selects a different activity within the app or device.

V. Conclusion and Future Work

There were performance issues with receiving the data, storing it, and updating the display. After a number of access points, the application's performance would degrade to the point it was no longer updating in real time. The application has been tweaked from 200 access points to 1500 access points, but this is still not performing as desired. Apple mentions this approach of recording a record at a time is not ideal for performance and other alternatives are recommended. Some approaches were tried such as disabling updates and changing how many records are buffered before being written with degradation in performance. The current method has been kept for demo purposes, but likely needs to be rewritten to remove the object relationship and replace it with a primary/foreign key relationship, although there is no guarantee this will resolve the performance issue. Another possible solution would be to use an observer pattern for both the display and the CoreData object model. Both would get updated as data comes in, with the object model being updated possibly in a different thread, leaving the GUI thread available and responsive.

Overall, the Kismet Client application is functional and achieves the objectives of this project as a mobile Kismet client. The user interface could use some additional work to make it less plain. An iPad specific version would be useful. It could make use of the split view (only available on the iPad) to display access points as they are received and show and update Google Maps in real time.

This project allowed me to practice and apply my Objective-C skills. I learned how to use CoreData objects. I also learned how to implement and communicate over sockets in iOS. I encountered performance issues with the UITableView object and was able to resolve those, but was not able to resolve issues with inserting and updating data in CoreData, although strategies have been discussed for possible solutions. The project source code is in publicly accessible GitHub, which gave me an opportunity to learn distributed revision control Git and the web-based Git repository hosting service GitHub.

VI. References

[1] https://github.com/rsbauer/KismetClient
[2] https://github.com/rsbauer/KismetServerSimulator
[3] Project page: http://www.rsbauer.com/apps/kismetclient