Massive WiFi Data Gathering

Salvador Aguinaga, Jianxu Chen
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
Email: saguinag@nd.edu, jchen16@nd.edu

***Abstract*—**

Access to open or unlocked WiFi access nodes and their characteristics, such as signal strength, network bandwidth, etc. is essential when choosing to connect to a network on-the-go. Building a system that provides accurate WiFi and cellular (3G, 4G) coverage requires cooperation from users and a system that is reliable and scalable. Users help collect network characteristics any where there is network access. The aim of this article is to propose a system solution capable of handling a large number mobile-computing users and a significant number of network connections to upload and download coverage data. In addition, we propose incentives to encourage cooperative and collaborative mobile computing, as the project relies on users and their mobile devices for data collection. Furthermore, we propose a data structure to store, retrieve, and update network coverage minimizes storage requirements and maximizes access to these data. Lastly, we provide cloud computing options designed to balance cost, functionality, and failure prevention. A system that offers stable, reliable data, and rich useful features goes a long way to encourage cooperation from the mobile community at large.

***Index Terms*—WiFi, heat-map, cloud-service, mobile-computing, load-balancing, balanced binary search tree**

I. Introduction

Massive WiFi data gathering (MWFDG) is a cooperative mobile computing service that offers comprehensive coverage of WiFi access points (AP) around your neighborhood and around the world. Users download a mobile app on their smartphone that will collect geolocation access point and signal strength pool it or uploaded immediately to a central data storage service. In cooperation with thousands of or even more users, the collected data can be utilized to offer users coverage maps (heat-maps) that show access points availability, signal strength, and approximate bandwidth.

The MWFDG service (available via the mobile app) also offers a travel-mode feature that lists WiFi access points located within a radius. Intended to be used when traveling, this feature lets users know what the WiFi coverage is near certain locations of interest (near the hotel a user is staying at or near the meeting or dining venue the user will be at).

In this article, we discuss two problems required for successful design implementation: first, how to incentivize users to collaborate by installing the app and use it from time to time and secondly, how to design and specify a backend infrastructure that scales up. We propose examples to incentivize users to cooperate and collaborate on this project that are both, feasible and sustainable. The backend infrastructure, we also propose, is detailed in Sections III and IV. Here we offer two potential solutions that highlight an optimal cost-to-functionality relationship.

II. Related Work

OpenSignal is a company that maintains a web service with coverage information for cell phone towers, cell phone signal strength, and now WiFi access points [1]. Google Play offers apps for Android devices offering similar functionality as OpenSignalMaps. Features included are: range of coverage for a given AP node, saving the information to local storage, ability to export the collected data to KLM (Google Earth). WiFi Map Maker and Sensorly are two such examples [2], [3]. On the same thread as the mobile *consumer* products, companies like Meraki offer enterprise product-solutions for WiFi deployment indoors and out. They offer a tool for mapping WiFi coverage that includes a heat-map web app [4]. One of the features Meraki offers is a planning tool for deploying AP nodes.

The shortcomings with the three works above are as follows: OpenSignal's Android app is cell phone tower centric, so the UI lacks intuitiveness for WiFi operation; the *Overview* tab signal strength is not accurate; the listed AP nodes (in map mode) show up clustered together requiring to zoom in; the user interface (UI) does not provide provide a simple list with the nearest free AP nodes. The WiFi Map Maker does not offer a heat map. Meraki's solution is not available as a consumer product nor as a mobile app.

III. Implementation

The massive collection of WiFi node information has three significant design requirements: i) a mobile, ii) desktop web app, and iii) a responsive and scalable backend web server with caching proxy and load balancing features.

A. Data Characteristics

Table II provides a brief summary of the data objects to be exchanged between the mobile, desktop and backend server. These data will be collected from each user from every location where users launch the app.

TABLE I
DATA EXCHANGE REQUIREMENTS

Typical Data Upload/Download			
User ID	Device Info (ID, vendor)	Geocode (latitude, longitude)	AP Attributes (MAC, SSID, signal strength, bandwidth, lock, unlock, type:a,b,g,n, ...)

B. Mobile and Desktop App

Both, the mobile and desktop web-app offer similar user-experience features designed to incentivize users to collaborate and be part of this project.

- → Map (which defaults to current geolocation, with a heat-map overlay)
- → Short list of the *best* AP nodes closest to the user's current location; selecting any of the additional AP nodes lists the node's characteristics (signal strength, bandwidth rates, distance from current location in meters or feet, and the option to highlight directions to the new location)
- → *[Optional]* battery savings or optimization estimate if the user chooses the *best* AP node nearest to user's current location

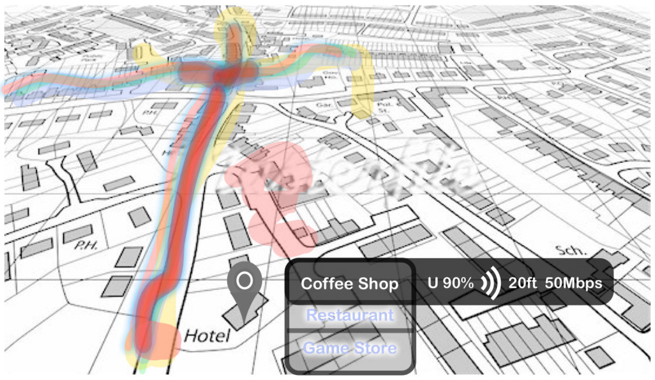

Fig. 1. Heat-map showing list of nearest AP nodes and top-selection details

C. Architecture

In this problem, we proposed to combine the advantage of web server and cloud service in order to keep our system in good performance while severing over a million clients. Therefore, besides a cloud space, we need to purchase several machines as web servers. This is necessary in making the system highly available (avoid crushing when a single server down) and scalable.

Amazon S3 is designed [5] for a high degree of static content distribution. It can easily sever millions of clients by adapting a highly scalable distributed implementation. However, the biggest drawback of Amazon S3 is that S3 does not support server-side processing, like PHP or JSP. And in our program, we have to deal with dynamic contents. Therefore, we adopt a new architecture designed in [6] to implement a scalable infrastructure.

The architecture is described in Figure2. The basic idea is (1) using a highly scalable cloud storage service (we use Amazon S3 here, but could also use nginx alternative) to deliver static content, (2) using web-server for dynamic content processing, and (3) using a load balancing algorithm[7]on client system (see Figure 3 for detail) to help the client browser choosing the best back-end web-server. Moreover, the the client-side logic and list of web-servers, as well as the server's status are hosted in cloud so that no single point of scalability bottleneck will trouble the system.

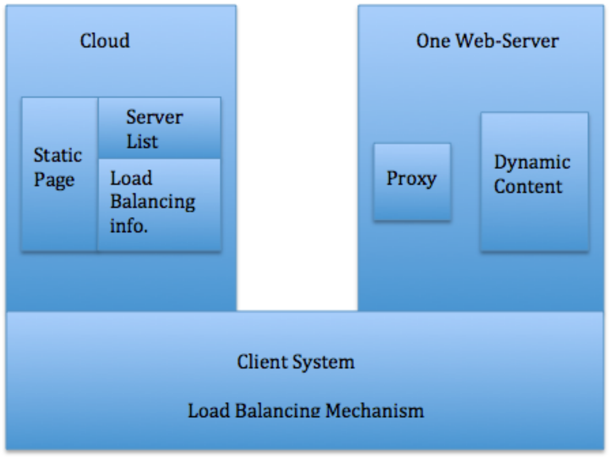

Fig. 2. Client-side Load Balancing Architecture[6]

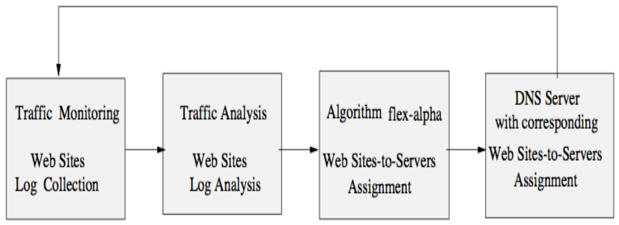

Fig. 3. Load Balancing Strategy[7]

D. Scalability

There are two significant features that ensure the scalability in this infrastructure. On one hand, by employing the load balancing strategy [7], the network traffic can be avoided. A web-server with larger available bandwidth is automatically selected. On the other hand, all these search work and decision are made on client-side. So we can benefit from load-balancing without bring much overhead to server-side.

Cloud computing is one of the best solutions to obtain scalability in number of users and in network traffic. Here we present two options each with different features, but with some customization capability to achieve cost targets. You can see the details in Table II.

TABLE II
CLOUD-COMPUTING: HARDWARE

Amazon Webservices		
Service Type	**Instances (size)**	**Cost/Hr Est.**
Amazon EC2 On-Demand	Small (linux) with free tier discounts	$0.06
Windows Azure		
Cloud Services	Small (1.6GHz,1.75GB)	$0.12

IV. WIRELESS SIGNAL MAP

A. Description of Data Structure

Inspired by [8], we propose a novel data structure to maintain and generate the wireless signal map. Based on some public map, like google map, we will create a tree structure for each town. The only criteria of defining a town is that the region's latitude and longitude don't differ over 0.5 degree and the region has intensive buildings. So a town in this paper might be a city, a county, or just part of a city. By definition, the entire wireless signal map would be represented by a forest.

Physically, every town will be divided into subregions every 0.0003 degree in latitude and every 0.0002 degree in longitude. It is a rectangular of about 75 feet by 85 feet. From [9], we know that the typical coverage range for Wi-Fi connectivity indoors is usually 50 to 150 feet. So, if an AP is reported in one subregion, we can simply assume that the same AP is available in this subregion.

As we know, one degree difference in longitude or latitude does not correspond to a certain physical distance in map, because it is calculated on a sphere. However, if we temporarily constrain out map in Unite States, then we can have a good approximation. (If the system is used in Australia or Russia, the size needs to be modified.) Geographically, the latitude of US ranges from 11 degree to 49 degree; and the longitude ranges from -124 to -66, based on Google Map. In this specific area, the physical size of one subregion is no bigger than 75×95. Therefore, the definition of subregion in our model is reasonable.

For example, based on Google Map, the town of South Bend is from (41.6300, -86.3400) to (41.7660, -86.0700) approximately. Thus, there are

$$\frac{41.7660 - 41.6300}{0.0002} \times \frac{-86.0700 - (-86.3400)}{0.0003}$$
$$= 680 \times 900$$
$$= 612000$$

subregions in this town.

In each tree, i.e. a specific town, each node holds the information of each subregions. The tree is created in as a modified balanced binary search tree (mbBST). In a binary search tree, each node has two children/subtrees, one has smaller key and the other one has larger key than the parent. But here, the key of each node is a two-tuples, namely the longitude and latitude. Therefore, each node in this tree has four children/subtrees, which have smaller longitude larger latitude, smaller longitude smaller latitude, larger longitude larger latitude, and larger longitude smaller latitude, respectively. You can see an example of this structure in Figure4

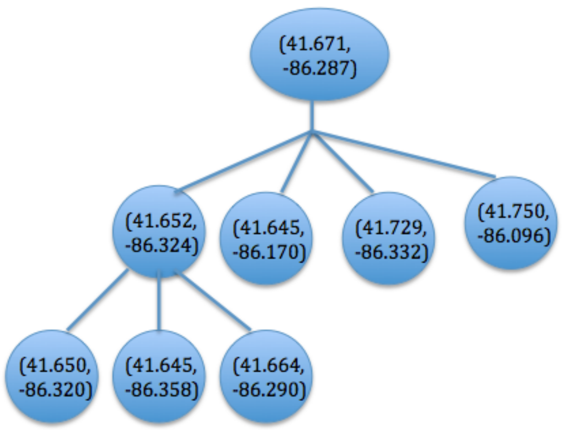

Fig. 4. mbMST Structure

We assume that the number of subregions where free WiFi is available is much less than the total number of subregions. If a 2D array is used to store the information, it must be a sparse matrix. That is why we use an mbBST rather than a 2D array. A matrix is a great waste of storage space in this case. Also, with an mbBST, all the data can be updated in O(log(N)) time (N is the number of subregions where signal has been detected)[10]. Thus, we believe our model is quite efficient practically.

B. Insert/Update Information

When new data is collected, we want to update the information in the corresponding node. To find that node, we can use a search operation similar to that for binary search tree [10], based on the key (longitude, latitude). If a node is found, just update the information; if not, a new node will be created. Note that for each node, value in the key will first be rounded, because the size of subregion is (0.0003, 0.0002). For example, (41.67785,-86.21127) will be changed into (41.6786, -86.2116).

C. Generate WiFi Signal Heat-Map

In the wireless signal map, only the information of the town, where the user requires or the user is physically in, is displayed. Therefore, to generate the wireless signal map, we can traverse the correspongind tree and pop up the information in each node sequentially into the map. In this way, the map will be generated in O(log(N)) time (N is the number of subregions where signal has been detected) [10].

D. Information in Node

In each node, three keys are in use. The first one is the location, i.e. a two-tuples (latitude, longitude). The second on is a pointer, directing to a link-list. In each node of the link-list, there are four values, AP ID, AP type, AP bandwidth, and AP strength. The last one is a number, representing the goodness of AP in that region. The goodness is computed by adding up the ”performance” of all APs in the link-list, while the performance for each AP is defined as the product of bandwidth and strength.

E. Size of Storage Space

The total space for storage can be approximated by the following equation.

$$\begin{aligned} TotalSpace =&(average\ size\ of\ info.\ associated\ at\ a\ node)\\ &\times(average\ number\ of\ nodes\ in\ a\ tree)\\ &\times(number\ of\ trees)\end{aligned}$$

Assume that there are no more than 10 APs at each subregion on average. The average space for all information associated at each node is

$$\begin{aligned} Total =&\ (8\ Bytes\ per\ entry)\times\\ &(4\ entries\ per\ node \times 10\ nodes\ per\ list\\ &+1\ entry\ for\ goodness + 2\ entries\ for\ location\\ &+1\ entry\ for\ pointer)\\ =&\ 352 Bytes\end{aligned}$$

Moreover, if we assume there are 30K subregions where AP is available on average, then the storage space for each town is about 10MB. Totally, suppose there are 10000 ”towns” across U.S. (For a big city like Chicago or New York City, we need to divide it into several smaller ”towns” to make the size controllable.), then 10GB storage space is needed in average case.

V. System Evaluation

A client/server system can be simulated by SPECjbb2005 [11]. SPECjbb2005 is a Java application emulating a 3-tier system with emphasis on the middle tier. In SPEC2005, data is in various structures, like Hash or Tree. And it is stored as a warehouse, which is approximately 25MB. Each warehouse feed a single thread. Thus, more clients are simulated, then more warehouse will be generated, and therefore more threads will be activated.

The metric of SPECjbb2005 consists of two components. One is the total throughput measurement. The other metric is the average throughput per JVM instance.

As this paper is focusing on the infrastructure design. No simulation would be conducted. However, we believe, as described in section III-D, our infrastructure is scalable, even up to 100K clients.

VI. Conclusion

Massive WiFi data gathering systems require cooperation from the mobile user community and a web service that offers a rich experience (as we want users to continue collaborating on this project). *Rich experience* refers to accurate, reliable, and fast access to network coverage information via a mobile device, like a smartphone, tablet, or laptop. Delivering this type of user-experience requires a system capable of scaling and one offering fault-tolerance (e.g. a single point of failure). We have proposed a cloud-computing solution that offers scalability and a set of software modules that play well with cloud services to provide fault tolerance to a degree. Cost is a significant consideration in designing and deploying this kind of system, thus in this article we propose an architecture optimized for load-balancing (with techniques borrowed from [6]) as an option to those offered by web-services providers like Amazon’s EC2. We have also detailed a data structure based on a binary search tree that minimizes volume of data and speeds up retrieval, insertion, and updates to the data, see section IV-A for the details. The architecture described leaves room for swapping out software modules that can help balance the overall startup costs or costs of ownership as the user base increases. The implementation of the mobile-app has to offer better experience and features, than those currently being offered by others (see the related works section). A combination of stable, reliable, scalable and feature-rich network coverage map system will draw users to cooperatively build a better service and one that users will want to use after the novelty has ran out.

References

[1] OpenSignal. (2012, Nov) Opensignalmaps - cell phone tower and signal heat maps. [Online]. Available: http://www.opensignal.com

[2] Sensorly. (2012, Nov) Find the right carrier for you. [Online]. Available: http://www.sensorly.com

[3] K. Dave. (2012, Nov) Find the right carrier for you. [Online]. Available: http://www.davekb.com/apps

[4] Meraki. (2012, Nov) About meraki. [Online]. Available: http://www.meraki.com/company/about

[5] H. Liu and S. Wee, “Web server farm in the cloud: Performance evaluation and dynamic architecture,” in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 369–380.

[6] S. Wee and H. Liu, “Client-side load balancer using cloud,” in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC ’10. New York, NY, USA: ACM, 2010, pp. 399–405. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774173

[7] L. Cherkasova, “Flex: load balancing and management strategy for scalable web hosting service,” in *Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on*, 2000, pp. 8 –13.

[8] Q. Liao, A. Blaich, D. VanBruggen, and A. Striegel, “Managing networks through context: Graph visualization and exploration,” *Computer Networks*, vol. 54, no. 16, pp. 2809 – 2824, 2010.

[9] UNC. (2012, Aug) Wireless and wi-fi best practices. [Online]. Available: http://help.unc.edu/help/wireless-and-wi-fi-best-practices-and-faqs/

[10] U. Manber, *Introduction to Algorithms: A Creative Approach [Paperback]*. Addison-Wesley, 1989.

[11] S. J. 2005, “http://www.spec.org/jbb2005/.”