OpenWiFi: A Consumer WiFi Sharing System

by

Hongyi Hu

S.B. C.S., M.I.T., 2006

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2007

Author .....
Department of Electrical Engineering and Computer Science
May 24, 2007

Certified by .....
Hari Balakrishnan
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

OpenWiFi: A Consumer WiFi Sharing System

by

Hongyi Hu

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2007, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

This thesis describes the design, implementation and evaluation of **OpenWiFi**, a consumer WiFi-sharing system. OpenWiFi aims to allow owners of WiFi access points (APs) to function as private internet service providers. The OpenWiFi model combines WiFi access facilities owned by participating individuals or businesses under a virtual network operator who offers internet access from these facilities to clients who subscribe and pay for the service.

The major problems in designing and implementing OpenWiFi include providing secure separation between private users of home networks from paying clients, developing a billing model that guarantees correct and fair payments when both clients and AP owners are untrusted and generalizing our mechanisms so that OpenWiFi may operate on hetergenous underlying hardware and software. We solved these problems with simple, clean procedures implemented at wireless APs and at a central OpenWiFi server that require no client modification. We conducted a number of static and mobile experiments, the result of which show that OpenWiFi is usable by roaming users who do not move continuously as well as high mobility users who connect from automobiles traveling at vehicular speeds.

Thesis Supervisor: Hari Balakrishnan
Title: Professor

Acknowledgments

I want to express my gratitude towards my advisor Hari Balakrishnan, who has been a wonderful mentor and supportive of my academic and professional endeavors. He is extremely quick and knowledgeable, and he was always able to offer some insights or advice whenever I had questions. Hari first conceived the idea of OpenWiFi and inspired me to pursue it as a viable research topic.

I want to thank Sam Madden for his input during the early stages of designing OpenWiFi. He sparked a great deal of discussion when he, Hari and I first talked about OpenWiFi and how it should be have. His advice greatly helped me analyze many different approaches and choose the right trade offs in the design and implementation.

I want to thank Jakob Eriksson for all of his help setting up high mobility experiments on the automobile Soekris boxes. He was kind enough to take time out of his busy schedule as a graduate student to walk me through his experimental setups and grant me use on two of his test cars.

I want to thank Kevin Chen and Alex Vandiver for their technical help with various parts of my implementation. They are both very knowledgeable people with detailed understandings of network systems.

I want to thank my wonderful, loving life-partner Gwenievere Capron for all of her unconditional support.

Finally, I want to thank my parents for all their support and sacrifices over the years. They have always done everything they could to ensure that I would have better life than they did.

This work was supported in part by the National Science Foundation under grant CNS-0205445 and by the Thomas M. Siebel Scholarship.

Contents

1 Introduction **13**

1.1 Motivation 13

1.2 OpenWiFi Overview 15

1.3 Contributions 16

2 Related Work **19**

2.1 RADIUS and 802.1X 19

2.2 FON 20

2.3 Wibiki 22

2.4 Share My WiFi 23

3 Design Considerations **25**

3.1 Trust and Incentives 25

3.1.1 Flat Rate 26

3.1.2 Bandwidth 27

3.1.3 Time 30

3.2 Security 32

3.2.1 Authentication 33

3.2.2 Encryption 33

3.3 Clients 35

3.4 Technological Constraints 35

4 Design **37**

4.1 Certificates . . . 37
4.2 OpenWiFi Servers . . . 37
4.2.1 Web Server . . . 39
4.2.2 Database . . . 39
4.2.3 Heartbeats . . . 41
4.2.4 Timeouts . . . 42
4.3 Access Points . . . 42
4.3.1 Firewall . . . 43
4.4 Clients . . . 44
4.5 Authentication Protocol . . . 44
4.6 Billing . . . 45

5 Implementation **49**
5.1 Server . . . 49
5.1.1 Database . . . 49
5.1.2 Web Server . . . 50
5.1.3 Scripts . . . 50
5.2 Access Point . . . 50
5.3 Client . . . 51

6 Evaluation **53**
6.1 Security . . . 53
6.2 Rate Limiting . . . 54
6.3 Authentication . . . 54
6.3.1 Pre-Authentication . . . 55
6.3.2 Cookies . . . 55
6.4 Experiments . . . 56
6.4.1 Low Mobility Experiments . . . 56
6.4.2 High Mobility Experiments . . . 57

7 Conclusion **69**

List of Figures

1-1 OpenWiFi System Diagram 16

4-1 OpenWiFi Overview 38

4-2 OpenWiFi Server Diagram 39

4-3 Database Diagram 40

4-4 Access Point Diagram 43

4-5 Authentication Protocol 47

4-6 Authentication Protocol After Timeout 48

6-1 Empirical CDF of OpenWiFi authentication time in static experiments, comparing certificates with cookies. Cookies tend to perform better than certificates. The lower portion of the graph where certificates seem to perform faster should be an artifact. 58

6-2 Empirical CDF WiFi connection duration in vehicular experiments. Average duration was 47 s. Most connections lasted 10 s or less. . . . 59

6-3 Empirical CDF heartbeat latency in vehicular experiments, comparing certificates with cookies. Both tests ran under identical network conditions. Cookies clearly perform faster than certificates. 60

6-4 Empirical CDF of heartbeat interarrival delay comparing certificates and cookies. 63

6-5 Empirical CDF of WiFi HTTP download speed. Each trial downloaded the index page of a website randomly chosen out of a list of 73 popular websites. 64

6-6 Empirical CDF of WiFi HTTP upload speed. Each trial uploaded a 30 KB file to our webserver. 65
6-7 Empirical CDF of downloaded HTTP file sizes over WiFi. About half of all files were under 5KB. 66
6-8 Empirical CDF of WiFi HTTP transfer speed comparing upstream and downstream transfer rates. Downstream speed tends to be higher than upstream speed, which fits expectations. 67

List of Tables

4.1 **client_accounts** Table Schema 41
4.2 **ap_accounts** Table Schema 41
4.3 **current_users** Table Schema 41

Chapter 1

Introduction

This thesis describes the design, implementation and evaluation of a consumer WiFi sharing system named OpenWiFi. Private residential WiFi is extremely popular and widely available today, but use is generally restricted to a small group of family and friends. Our system unites home WiFi Access Point (AP) owners into miniature ISPs by paying them to open up their home networks to roaming clients. These clients in turn pay the AP owners for access through our system.

We first discuss our motivation for this system, related work, and many critical design problems. Then we describe the design and an implementation of our system. Finally, we describe an experimental analysis of our end product.

1.1 Motivation

Most people can obtain Internet access at their homes or offices easily. There are a number of competing ISPs out there that can serve a residence or a business. However, once a person steps outside, he has few available options for internet access. Many cellular providers offer data services such as GPRS that are expensive, high latency and low bandwidth. WiFi hot-spots are often few and far apart. As FON states on its website, "At home you're WiFi king, but away you become WiFi beggar." [11]

Near ubiquitous WiFi would be a boon for sharing and accessing information, potentially enhancing the way people communicate, collaborate, work and play. For ex-

ample, drivers could outfit automobiles with WiFi-enabled traffic monitoring and performance sensors to identify maintenance issues, track emissions and other efficiency data, and analyze and select driving routes based on traffic and road conditions[3]. People could sit down anywhere on the road within range of a participating access point and interactively collaborate with their friends, colleagues and business partners anywhere in the world, instead of passing emails and text messages back and forth.

Unfortunately, the technology for providing cheap, widespread WiFi access does not exist today. Many cities have started initiatives to install thousands of APs to provide municipal WiFi access to their citizens. However, studies on the subject of city-wide wireless access have concluded that the costs are simply too high; the average maintenance cost for city-wide WiFi is on average $150,000 per square mile in five years[4]. They estimate that the municipal governments could charge users $25 a month for service and still not meet costs [4].

Until low cost WiFi technology becomes feasible, the next logical step is to take advantage of the existing WiFi infrastructure, most of which is personally or commercially owned. However, in order to build a WiFi sharing system using third-party owned APs, many challenges and issues need to be addressed.

- *Legality:* Most ISPs disallow sharing of an Internet connection with the general public.

- *AP support:* There is a wide variety of WiFi hardware on the consumer market that potentially require support to have the widest possible user base.

- *Incentives for participation:* We need to give AP owners incentives to open up their networks and participate in OpenWiFi. One possibility is to bill clients for using the service and pay AP operators for providing service according to a heuristic with some notion of fairness. If we use financial compensation as an incentive, should we bill and pay at flat rates or do we measure usage by time or bandwidth consumed? Another possibility is to offer membership in a social circle, where members of the circle provide Internet service to each other.

- *Cheating:* We trust neither clients nor service providers. How do we prevent clients or AP operators from cheating the system for profit? Is it possible for different parties to collaborate in order to profit?

- *Data security and authentication:* We need to somehow segregate client and AP provider to protect their data from each other even though they share the same WiFi networks. How do we keep client transmitted data private? How do we prevent clients from accessing sensitive data owned by AP operators that provide their service? How do we authenticate clients for access control and billing purposes?

- *Rate control and performance:* How do we allocate bandwidth fairly to clients using an AP? How do we minimize overhead of our system features so that WiFi connectivity and performance is not significantly affected?

- *Vehicular mobility:* In the real world, clients may be moving constantly at pedestrian speeds, bicycle speeds or vehicular speeds. Supporting connectivity when clients are stationary is simple, but we want to support high mobility clients. Therefore, we need to minimize association time and WiFi hand off overhead.

1.2 OpenWiFi Overview

In OpenWiFi, there are three classes of entities: *clients*, who subscribe to the service, *access point owners*, who provide internet service to the clients, and the *virtual network operator*, who manages the service, bills clients and disburses payment to the AP owners. Figure 1-1 shows a general overview of OpenWiFi.

Our design uses a distributed set of servers that manage access control and billing, much like in RADIUS. AP owners install software on their APs that allow access to clients authenticated by these OpenWiFi servers. Clients obtain WiFi service by connecting to a participating AP and sending authenticated heartbeats to an

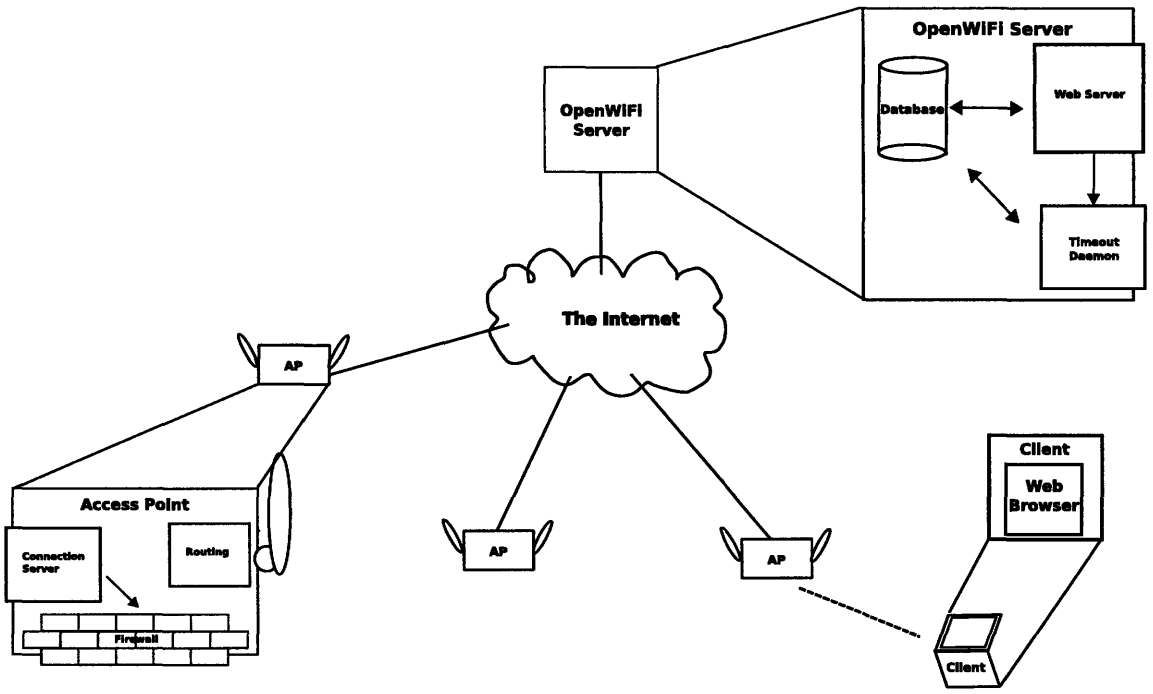

Figure 1-1: OpenWiFi System Diagram

OpenWiFi server. The server responds to the heartbeats by signalling to the AP to accept the client's WiFi traffic.

A key point in OpenWiFi is our trust model. Since the virtual network operator is doing business with potentially malicious parties, we cannot trust **anyone**, not clients nor AP owners.

1.3 Contributions

We solved the WiFi sharing problem by designing a simple, practical framework that accumulates authentication, access control and other operations in a separate application layer independent of underlying hardware or link technology.

We made the following contributions:

1. *Fair Payment Model:* We designed a time-based billing protocol to ensure fairness even when both clients and APs are untrusted. We show that flat rate billing allows certain cheating behavior to occur. We introduce a dual signature bandwidth billing protocol that guarantees fairness in theory, but we show that

it is infeasible in practice. Finally, we show a time-based billing protocol that gives approximate fairness as well as convenience.

2. *Heterogeneous Compatibility:* We designed OpenWiFi to be compatible with many different WiFi products on the market. We do not require specialized hardware, and we do not require any client modification.

3. *Software Implementation:* We developed an implementation of OpenWiFi that consists of a central server and several APs. Our implementation supports both static clients such as laptops, and highly mobile clients such as WiFi-enabled vehicles. Our implementation will be released under the GPL ver. 2 and will be available for download.

4. *High Mobility* We designed OpenWiFi to be usable by both static clients and high mobility clients moving at vehicular speeds.

5. *WiFi Performance Analysis:* We collected data on WiFi performance out in the field, including download speeds, upload speeds, and OpenWiFi latency. We tested and compared two different methods of authentication: cookies and certificates. We show that our implementation performs well in low and high mobility scenarios.

Chapter 2

Related Work

This chapter discusses related work and other WiFi sharing solutions and initiatives such as RADIUS, FON and Wibiki.

2.1 RADIUS and 802.1X

The RADIUS protocol provides a solution to authenticate wireless users based on a variety of different credentials ranging from passwords to Transport Layer Security (TLS) certificates [15]. In RADIUS, paired with the IEEE 802.1X standard [9], a centralized RADIUS server stores access and accounting information, and also authenticates users. When a client first associates with an AP, the AP opens a port for the client and forwards his credential information to the centralized RADIUS server using the Extensible Authentication Protocol (EAP)[1] using TLS certificates[2]. All deployed APs maintain a shared secret with the server so that this communication can be encrypted. The RADIUS server responds to "accept" or "deny" the client.

RADIUS is widely used not just for wireless authentication, but also for general network access control and accounting. RADIUS support is built into Chillispot[8], an open source WiFi hot-spot management software package. A successor to RADIUS, known as DIAMETER, is under development [6].

Unfortunately, RADIUS has a number of drawbacks that make it inadequate for OpenWiFi. First, it requires a homogenous AP and client infrastructure, which is im-

practical. Consumer WiFi hardware and operating systems do not necessarily share the same RADIUS or 802.1X authentication features. Second, it assumes a two party service provider to client relationship where the providers have control of the central server and the APs. In OpenWiFi, we have a three party system consisting of clients, homeowners or businesses who operate APs, and a trusted third party administrating OpenWiFi. OpenWiFi requires mechanisms that prevent AP operators cheating clients, a feature that is not present in the RADIUS specification. Finally, RADIUS without modification requires relatively high authentication overhead, which is not feasible for clients traveling at vehicular speeds.

2.2 FON

FON[11] is a European company that provides a way for private individuals to share WiFi. FON's model is built around class-based WiFi sharing. There are three categories of users in the FON model:

1. **Linuses**, named after Linux creator Linus Torvalds, are people who share their WiFi openly, without charge. Because Linuses share their WiFi freely, they may also use other APs in the FON network for free.

2. **Bills**, named after Microsoft founder Bill Gates, are people wish to profit from FON. They make money from Alien users (see below) who use their APs.

3. **Aliens** are FON users who do not share WiFi. They pay a flat daily rate to use the FON network.

FON encourages its users, or Foneros, to share their WiFi. They intend for most of their user base to be Linuses who share openly. The FON website introduces Aliens as users who do not yet share WiFi, implying that they have not yet converted to the FON philosophy.

When we first started working on OpenWiFi in early 2006, FON consisted of a firmware release based on OpenWRT for certain models of WiFi routers. They

seemed to incorporate Chillispot for authentication, but they used no WiFi encryption or bandwidth throttling features. We noted their approach to tackling some of the issues that we also faced, and we proceeded to design and implement OpenWiFi in our own way.

As of this writing, nearly a year and a half later, FON has released their own customized "social" router, dubbed "La Fonera." La Fonera allows for two SSIDs, one encrypted for private use and one unencrypted for other FON users. It is still possible to download their legacy software for use on non-customized commercial routers, but they seem to encourage purchase of La Fonera. FON seems to have made significant strides in providing a quality WiFi sharing system.

However, FON fails to meet or address several of the constraints we discussed previously.

- *Hardware support:* We built our system around hardware and software low level heterogeneity. Consumer WiFi products change all the time, and new wireless standards are in development that are not necessarily compatible with previous specifications. By pushing user adoption of the customized La Fonera, FON is effectively dividing the market by creating a specialized router for a unique purpose. FON also has legacy firmware that will function on compatible WiFi routers, but they do not share the dual SSID feature of La Fonera.

- *Billing:* A flat rate billing model implies that certain cheating problems have to be addressed, as we will describe in the next chapter.

- *Data security* FON does not have any mechanisms to keep user transmitted data safe except on the private channel.

- *Performance* Authentication in FON requires a username and password check via a web interface at every AP. Thus the handoff cost between APs is very high. This works fine for scenarios where a client sits down for relatively long periods of time, but not for clients who wish to to roam, even between two access points. Imagine what happens when a client attempts to make a VoIP call and moves between two APs.

- *High mobility applications* FON's authentication protocol is completely useless for high mobility applications because it requires a separate login per access point.

FON's description of other salient features of its network is vague, so it is unclear whether there are further issues without buying a La Fonera and analyzing its behavior. For example, it is not certain whether an AP operator could steal username and password information from clients logging in through that AP. The previous incarnation of the FON firmware certainly allowed for that scenario.

In summary, FON attempts to solve a similar WiFi sharing problem to our own by implementing a P2P system. It meets some of the goals we set but falls short of others such as fair billing and high mobility support.

2.3 Wibiki

Wibiki[16] [12], short for "Wireless Ubiquity," is a subsidiary company of Speedus that provides WiFi connection management software for both client devices and APs. Wibiki also attempts to distinguish itself with a strong emphasis on WiFi security and ease-of-use.

Wibiki claims that most private home APs are unsecured, possibly because most built-in configuration utilities are difficult to use, and many users are not WiFi experts. Wibiki software on APs automatically enables security features such as WEP encryption to assist users with low technical proficiency in setting up their routers. Wibiki does not install custom firmware on APs; it merely configures features that are already available on the APs.

Wibiki software on client machines is about usability. At home, the Wibiki application will connect automatically to the user's AP and handle any security configuration that is required. While roaming, Wibiki automatically displays a list of both open and secured APs to the user, and it assists the user in connecting to an AP. If a Wibiki-enabled router is available, the application will connect to that router automatically.

Wibiki's business model is based on opt-in advertisement sales. The idea is to avoid "traditional" ads such as pop-up ads or splash pages by customizing ads to an individual user's context and interests.

2.4 Share My WiFi

Share My WiFi [17] is another project that does P2P WiFi sharing by paring up users through a centralized web service. Users who own WiFi routers or APs post listings on sharemywifi.com with approximate (for privacy) geographic location. Users who seek WiFi can search on sharemywifi.com for available APs near them. The service strictly matches users and does nothing more. It is up to the WiFi provider and the WiFi seeker to determine the conditions of their WiFi sharing agreement. They are free to exchange money or other services.

Chapter 3

Design Considerations

In this section, we discuss the trade-offs of various approaches to designing OpenWiFi. We explore different billing models, authentication techniques and data encryption options. Finally, we highlight some technical constraints to consider for OpenWiFi.

3.1 Trust and Incentives

In OpenWiFi, we need to give AP owners incentives to share their WiFi networks. Incentives could be in the form of money or access to a social WiFi network such as in FON or Wibiki. We emphasis however, that OpenWiFi is not strictly limited to these kinds of incentives.

We need to solve the problem of accounting no matter what type of incentive system we decide to use. If we choose to use money, then clearly we need a way to bill clients and reimburse AP owners, perhaps by time or bandwidth. If we choose to do social networking, we may decide to allow a user access to the network based on how much time or bandwidth his AP has contributed to the network. Overall, while we discuss monetary incentives for the rest of this section, our approach can be directly applied to other types of incentives.

We have several choices for accounting:

- Charge clients a flat rate and pay AP owners back at a flat rate.

- Charge clients by bandwidth and pay AP owners based on total bandwidth served.

- Charge clients by time and pay AP owners based on the total time clients spent using their wireless access points.

3.1.1 Flat Rate

The flat rate model is common in the home consumer market for Internet service. Many users pay a flat monthly rate for unlimited use from their local ISP. The typical expected behavior is that the ISP-installed link is shared among a small group of family members or housemates with occasional use by friends and associates. It is generally a violation of ISP contract for a customer to share his link with the public. However, in the OpenWiFi model, we want homeowners to share their links with registered clients.

A flat rate billing model is problematic because it permits troublesome lopsided behavior. For example, consider a case where homeowner Ben opens his AP to provide service to OpenWiFi clients. Ben's neighbor Alyssa decides to sign up for OpenWiFi service, and she connects to Ben's AP on a permanent basis. Now Alyssa can provide her entire household with Internet service with a single OpenWiFi client account. In addition, Alyssa can resell her bandwidth and potentially make a profit.

Alyssa's lopsided behavior can exist because billing by flat rate is not sensitive to client usage. Since there is no penalty for heavy use, clients have an incentive to consume as much bandwidth as they can to maximize value for dollars spent. In addition, paying AP owners a flat rate can become complicated because there are several issues to consider.

First, there is the boundary case: do we pay AP owners who have served no clients? The answer should clearly be no, unless we wish to pay AP owners for no work.

Next, if we pay for only a nonzero number of clients served, should we pay a marginal rate for each additional client or a total flat rate? If we pay a total flat

rate, then AP owners have no incentive to provide good service since they get paid the same amount no matter how many clients they serve (as long as it is nonzero). However, even if we pay a marginal rate per additional client, bandwidth consumed by clients will widely vary. We could have a scenario where all of AP owner A's bandwidth is used by a few clients and only half of AP owner B's bandwidth is used by many clients. In this case, we would pay B much more than we would pay A, which could be considered unfair.

Finally, we would have to set some kind of cap on payments to ensure that AP owners can't collude to form some kind of pyramid scheme in order to game our system.

3.1.2 Bandwidth

It seems that flat rate billing is inadequate for our purposes. However, if we choose to charge by bandwidth, then we face the challenge of accurately measuring consumed bandwidth for each client. This requires cooperation from the clients and the AP owners, but unfortunately, both parties have incentives to lie. The client has incentive to cheat to minimize his bill, and the AP owner has incentive to cheat to maximize his profits. It seems that we require a bandwidth reporting mechanism that is certifiable by client and AP owner. We designed two possible reporting systems: the **Funnel scheme** and the **Dual Signature scheme**.

Funnel Scheme

The basic idea in the Funnel scheme is to funnel all traffic through OpenWiFi controlled *nodes*, which could be servers or routers, we and count packets for each client at the nodes.

In theory, we can perfectly determine bandwidth use per client at each AP, but this scheme is crippled by its lack of scalability. Funneling requires an expensive infrastructure that can support high traffic load. Furthermore, funneling defeats the point of persuading private AP owners to share their WiFi. If we can deploy an

expensive and efficient infrastructure to carry our clients' data, we should simply deploy our own APs in the first place, and we have already determined the costs of a municipal AP network are prohibitively high [4]. The Funnel scheme is not practical for OpenWiFi's needs.

Dual Signature Scheme

The basic idea in the Dual Signature scheme is to require both client and AP operator to sign packets with a predetermined frequency. We count the double-signed packets to measure bandwidth.

For a formal definition, let Alyssa be the client and Ben be the AP owner. The OpenWiFi service issues both Alyssa and Ben their own cryptographic certificates. For every n packets that Alyssa transmits, she sends one heartbeat packet signed by her certificate. Ben's AP receives this signed heartbeat packet and signs it with Ben's certificates. Then Ben's AP forwards this packet to an OpenWiFi server to be recorded. If Alyssa misses sending a heartbeat packet, then Ben's AP disconnects her. The OpenWiFi service pays Ben proportional to the number of heartbeat packets signed by his certificate and some other client certificate. If Ben's AP misses forwarding a heartbeat packet, then Ben will not be paid. Thus, both client and AP owner have an incentive to cooperate in this scheme.

In theory, this dual signature heartbeat scheme should fairly allocate cost and payment according to a bandwidth model, assuming that the variance of client packet sizes is small. However, in practice, there are several issues.

First, an implementation of dual signed heartbeat packets requires low level control at the both the client and AP level. Low level changes to APs are reasonable, since AP owners would be willing to modify their hardware given enough financial motivation. However, low-level changes at the client level seems far less feasible. Since clients are already paying for the service, they would probably be much less willing to accept such modifications to their laptops or other wireless devices, especially at the kernel or driver layer. One of our major objectives was to avoid low-level client modifications, so a packet level scheme such as the one we described is not acceptable. In addition,

it is unclear how packet level processing will impact client WiFi performance.

Second, AP owners have complete physical control over their hardware, which means they have total control over any communication channel they grant to their clients. An AP owner can easily cheat by firewalling all client traffic except for signed heartbeat packets. Of course, clients could report unscrupulous APs to the OpenWiFi service, but in practice it would take a great deal of effort to investigate and confirm such reports while it is nearly effortless for AP owners to cheat. We prefer a more self-policing approach, which requires either creating a separate communications pipe for clients not controlled by APs or obscuring heartbeats to defeat an AP's heartbeat firewall.

Unfortunately, a separate communication channel for reporting heartbeats seems infeasible. A roaming client's only method of reporting to the OpenWiFi service is through registered APs when he is out in the field. Having a channel independent of APs would usually be untenable since the client could simply use the separate channel for Internet service in the first place. In addition, a separate channel, like a cell phone provider's data network, could be extremely expensive. The alternative is for the client to report his bandwidth usage after the fact. However, *a posteriori* reporting fails because in this scenario the client has already received service, so he has no incentive to give true bandwidth reports. Bandwidth reporting during connections works because clients are forced to report or they will be disconnected. Clearly, an independent bandwidth reporting channel is inadequate because the AP owner may lie.

Unfortunately, obscuring heartbeats seems to be similarly ineffective at preventing cheating. The idea is to hide heartbeats among normal client traffic to force AP owners into a guessing game. If heartbeats are indistinguishable from other packets, then APs cannot filter heartbeats efficiently. The best they can do is guess which packets to drop or forward, but there is reasonable danger of dropping heartbeats, which results in a smaller payment. This seems good in theory, but in practice it is difficult to implement. We require client signatures on heartbeat packets so that they cannot be forged by APs. Therefore, for heartbeat packets to look like regular

packets, we must require that clients sign all of their packets. Otherwise, APs could simply drop non-signed packets. In addition, heartbeat packets cannot be obviously addressed to an OpenWiFi server or otherwise APs could drop all packets except ones with a destination IP address that matches a known OpenWiFi server. Therefore, we must require that clients redirect heartbeat packets through proxies in order to deter destination IP filtering. Choosing proxies is not a trivial task; proxies should ideally be chosen at random from a large list to hide any potential traffic patterns that APs could use to identify heartbeat packets. However, even random proxy selection is not enough; AP owners could collude with clients or even sign up for a client account in order to access a proxy list. Ultimately, unless clients are willing to send all of their network traffic through these proxies, obscuring heartbeats is not possible. Even if clients were tolerant of this scheme, redirecting heartbeats via random proxies would also require low level changes at the client level. In addition, the use of proxies would cause network performance hits.

At the present moment, we are not certain that the fair bandwidth billing problem is solvable under the practical constraints of the OpenWiFi model.

3.1.3 Time

The only option left is to charge clients by time. Consider the following model where again, Alyssa is the client, and Ben is the AP owner. The OpenWiFi service issues both Alyssa and Ben their own cryptographic certificates. For billing, we require that Ben's AP only provides service to Alyssa when she sends send periodic heartbeats to the OpenWiFi service. Note that heartbeats are no longer sent for every n packets but rather for every n minutes. If the heartbeats from Alyssa stop, then Ben's AP disconnects her after some timeout period. The result is that we can measure Alyssa's usage by recording her heartbeats, and we can pay Ben based on the heartbeats that originated from his AP.

In theory, this heartbeat scheme limits the ability for clients and AP owners to cheat provided that the heartbeats are authenticated and protected against replay attacks. Heartbeats cannot be underreported because clients must provide them

to obtain service, and heartbeats cannot be over-reported because clients have no incentive to do so, and AP owners should not be able to falsify or replay them. Our time based heartbeat scheme does have its limitations, but we can address those issues.

One problem is that AP owners could still firewall their APs so that all client traffic is dropped except for heartbeats. They would still receive payment while effectively denying service to any clients. In our previous discussion of bandwidth billing, we exhausted practical options for addressing the heartbeat firewall problem. However, we believe cheating would be mitigated by the open market nature of our model, where AP owners are essentially competing service sellers, and clients are consumers. One could imagine, for example, a situation where local coffee shops register with the OpenWiFi service to provide WiFi Internet to their patrons. The coffee shops are already competitors, and superior WiFi service is simply an edge one establishment would have over the others. A cheating AP owner may make money from his clients when they first associate with his AP, but they will quickly discover that their packets are being dropped. As a result, clients will flock to APs that provide superior service, and cheating AP owners will rapidly lose business. Of course, sometimes clients may only have one AP choice for connectivity, but in those situations, we expect that clients will not pay providers who are actively scamming them. In general, an AP owner should have incentive to provide a minimum level of service to clients.

A second problem is that heartbeats require that APs always have network connectivity with the OpenWiFi server. An isolated AP would be unusable by clients since heartbeats would not reach the virtual network operator. Also, there is no incentive for an AP to allow access when a network partition exists since the clients would not be billed. However, network connectivity with the OpenWiFi server is necessary in our model for time reporting. Fortunately, there is a readily available solution; we can distribute the OpenWiFi service among multiple servers in different geographic locations. A distributed service could provide backup routes in case of network partitions and minimize the network latency between AP and server. We could contract a third-party content delivery company such as Akamai to distribute

our system.

One advantage of time billing over bandwidth billing is that we can easily implement time billing at an end-to-end level. We do not have to modify client hardware or software because we no longer have to do packet level processing.

Ultimately, we decided to bill clients by time because it is more practical than bandwidth billing and addresses the issues with flat-rate billing. We will maintain the invariant that only one AP will be opened for a client at any given time. Overall, our model will underestimate the amount of time a client spends browsing because we will only bill and pay for received heartbeats. A client may have spent more time using the service than indicated by his heartbeats, but we expect that discrepancy to be small, and we will attempt to minimize the discrepancy in our design.

3.2 Security

Since the OpenWiFi model involves the exchange of money, security is an important concern. We consider the following threat model:

- **Malicious client:** A malicious client generally attempts to avoid paying for the OpenWiFi service or obtain more service than what he paid for. A malicious client may try to underreport time spent on the service or steal other clients' authentication information to use the service for free.

- **Malicious AP owner** A malicious AP owner attempts to maximize his profits by over-reporting heartbeats, severely rate-limiting clients. A malicious AP provider could also eavesdrop on client WiFi traffic and could potentially steal sensitive financial information.

- **Malicious cooperation of client and AP owner** Malicious clients and malicious AP providers could conspire to cheat the OpenWiFi model. This is possible, for example, if the rate that we charge clients is less than the rate that we pay AP owners. We must ensure that no such loopholes exist in our model.

- **Wireless eavesdropper** A wireless eavesdropper could steal sensitive information from clients if they send their data in the clear using the service.

We should require some type of authentication mechanism to ensure that we bill/pay the correct parties at the end of every business cycle and to prevent cheating by malicious clients or AP operators. We should also require some kind of encryption to keep client transmissions private in order to prevent leaking of sensitive client information. Finally, we must design our billing model to prevent conspiracies of malicious clients and AP providers from making a profit from OpenWiFi.

3.2.1 Authentication

We determined two requirements for heartbeats in our model. First, heartbeats must be authenticated by clients. Since we bill and pay by heartbeats we receive, each one is a transaction that requires authentication. Otherwise, a malicious client, for example, could masquerade as someone else and use our service on that other person's dollar. Second, each heartbeat must be unique in some verifiable way; otherwise, a malicious AP operator could replay many heartbeats and reap an unfairly large payment.

We can satisfy our requirements easily with a lightweight public key infrastructure system. Clients can sign heartbeats with their private keys, which in turn should be verifiable by the certificate authority that issued client certificates. Since heartbeats require timestamps already, we have a built-in nonce that will deter replay attacks.

3.2.2 Encryption

Encryption turns out to be a much more difficult issue to resolve than authentication. There are three main options we can provide.

1. **Client-AP encryption:** Client-AP encryption is already supported in the existing hardware and firmware available on the consumer market. In general, it should not require additional changes to client machines. However, there are multiple 802.11 encryption standards supported by WiFi APs and cards, and

so not all products are compatible with each other. We discuss further consequences of technological heterogeneity in the next section. Furthermore, while client-AP encryption could prevent malicious clients or wireless eavesdroppers from reading the packet contents of other clients, it does not prevent malicious AP-owners from reading those packets. Client-AP encryption is not sufficiently secure for our model.

2. **End-to-end encryption:** End-to-end encryption is a more secure solution than Client-AP encryption alone. We could provide a VPN service for clients that tunnels all of their traffic to a server operated by our service since presumably our clients trust our service more than they trust AP operators. Unfortunately, the VPN solution is not practical. VPN tunnels incur additional overhead for connecting to an AP because clients must associate with an AP, authenticate to an OpenWiFi server and then set up a VPN tunnel session with the server before he can send any data. VPN also scales poorly since we need an infrastructure to both encrypt and funnel client data. Finally, VPN requires the installation of specific software on client machines. For all of these reasons, it is infeasible for our service to offer end-to-end data privacy services.

3. **No encryption:** It seems that the best option for us is to offer no encryption. Clients must realize that any data they while using OpenWiFi is vulnerable to eavesdropping, and it is their responsibility to use encryption (i.e. SSL or TLS) whenever they conduct sensitive online transactions such as credit card purchases or bill payments. More conscientious clients can set up their own VPN tunnels, but the choice should be made by clients, not by us. This option allows our model to be flexible enough for both high and low mobility client use.

3.3 Clients

In the OpenWiFi model, we identify two categories of client behavior: *low mobility* and *high mobility*. Low mobility clients tend to roam infrequently at pedestrian speeds, which means that they connect to few APs in the OpenWiFi system for relatively long periods of time. Low mobility clients are people who walk to coffee shops, bookstores, libraries and other potential OpenWiFi hot-spots. High mobility clients tend to roam frequently at vehicular speeds, so they connect to many OpenWiFi APs for short periods of time. These are people carrying their laptops in their cars or in taxi cabs or on commuter trains, for example.

In real life, most clients will exhibit both low and high mobility behavior during different periods of the day or week. Therefore, OpenWiFi should support both ends of the behavior spectrum, which means that we must minimize overhead of any authentication or encryption scheme that we use. Otherwise, high mobility clients would spend most of their time authenticating and very little time actually transmitting data.

3.4 Technological Constraints

Today, a great number of home consumers have cable modems, DSL, or other similar high-bandwidth lines (relative to the previous generation of dialup networking). However, the wireless routers and access points on the consumer market vary greatly in supported features and capabilities; some cheaper products support only 802.11b with WEP support while other, more expensive products support draft versions of the new 802.11n standards along with many other bells and whistles.

We live in a WiFi world filled with an electic assortment of hardware with support for different stages of the 802.11 standards. This leaves three main choices for the system we wish to design.

1. **Open System** We could open all access points and allow anyone to connect to them. However, it would be impossible to track client usage for billing.

2. **Homogenous System** We could force all participants of OpenWiFi to own hardware that complies with the new 802.11i security standards. In a perfect world, this would be the ideal solution. The OpenWiFi service could issue SSL certificates to clients and run a RADIUS server for authentication and billing. All OpenWiFi access points would authenticate client users and encrypt data with WPA2.

   For this kind of system, both AP owners and clients would have to buy 802.11i compliant hardware. Currently, only the newest, most expensive routers and WiFi cards support 802.11i features. Many are not interchangeable with each other, and some operating system platforms either do not support the new 802.11i features or require special upgrades. It is not practical given current consumer usage of WiFi technology to expect such a homogenous system to become popular or profitable. Perhaps this solution will be more viable in a few years when nearly all routers and WiFi cards on the market will support the 802.11i standards. However, we expect WiFi standards to always be in state of flux; as standards are released and implemented, people are already designing and debating the next version. An implementation of our system that depends on aspects of the new 802.11i standards may not be compatible in the next release.

3. **Mixed System** We could design OpenWiFi to operate independently of the underlying wireless technology and standards that drive the hardware. This end-to-end approach would be viable given today's technological constraints because we could appeal to a wide range of customers, and it could prove ideal for a future of perpetually advancing WiFi technology and standards. We ultimately adopted this approach for OpenWiFi.

Chapter 4

Design

The OpenWiFi system is composed of a distributed set of servers, a network of access points and a pool of clients. The servers sit on the Internet and control client authentication, access and billing. The access points are owned by individual private consumers and are potentially distributed all over the world. They provide WiFi service to clients who register their wireless-enabled devices with the server to use the OpenWiFi service.

4.1 Certificates

For authentication, OpenWiFi uses public key cryptography. The OpenWiFi service maintains its own Certificate Authority (CA), and generates and distributes certificates to clients and AP owners. The CA and its supplementary management system can be hosted on an OpenWiFi server. Each client or AP operator receives only one certificate, but that certificate may be installed on multiple machines or APs.

4.2 OpenWiFi Servers

The servers have three main functions. First, they act as a web portal for non-registered users to join the system and for registered clients to access and change their billing and personal information. Second, they remotely authenticate clients who are

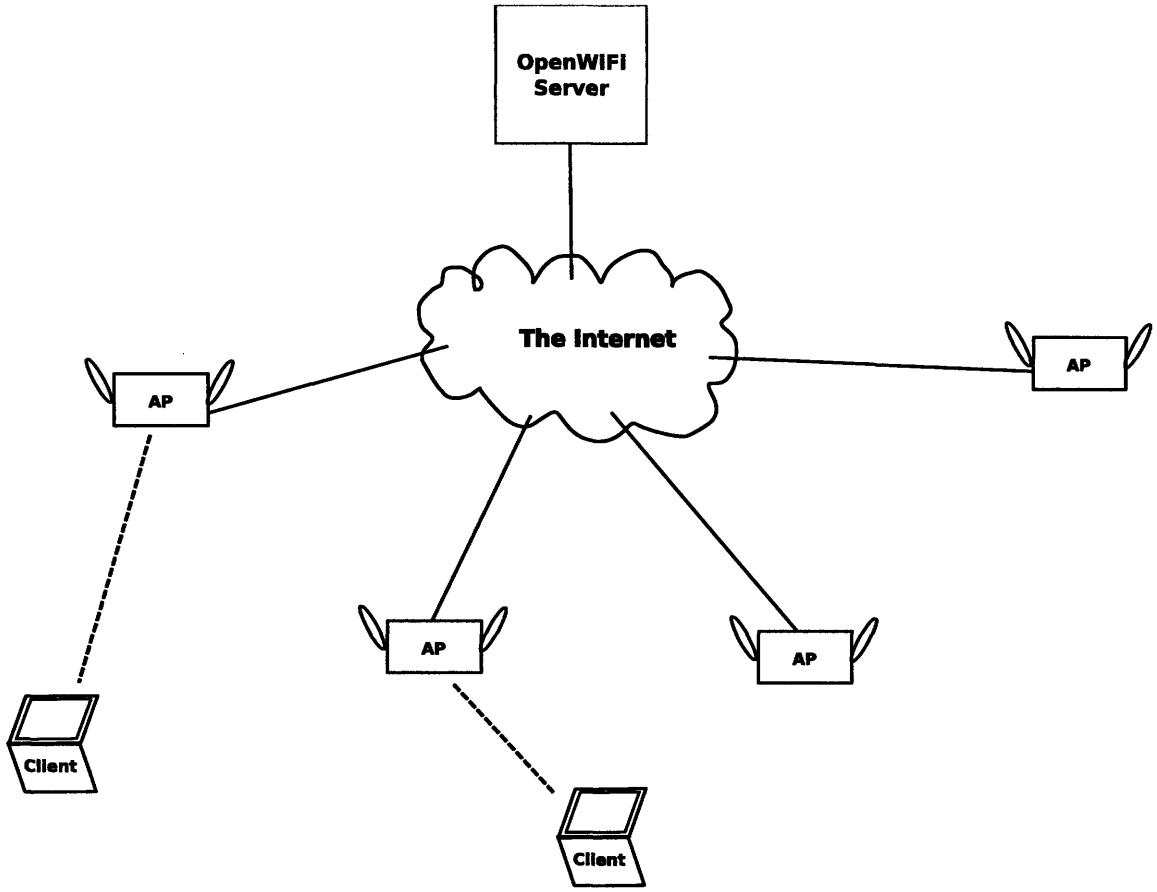

Figure 4-1: OpenWiFi Overview

currently using the OpenWiFi system for internet service. Third, they tracks clients' usage in order to bill them at a fair rate and disburse payment to the AP owners.

To accomplish these tasks, the OpenWiFi servers run a web server front end with a database back end and an event-driven daemon. The web server provides a registration process for new users and an SSL certificate based-authentication and heartbeat mechanism over HTTPS. The database backend stores client and AP data, and it maintains a table of current active clients using the OpenWiFi service. Finally, the server will run a daemon that processes incoming authentication requests and sends **connect** and **disconnect** requests over HTTPS to AP connection servers in order to control client access at the APs.

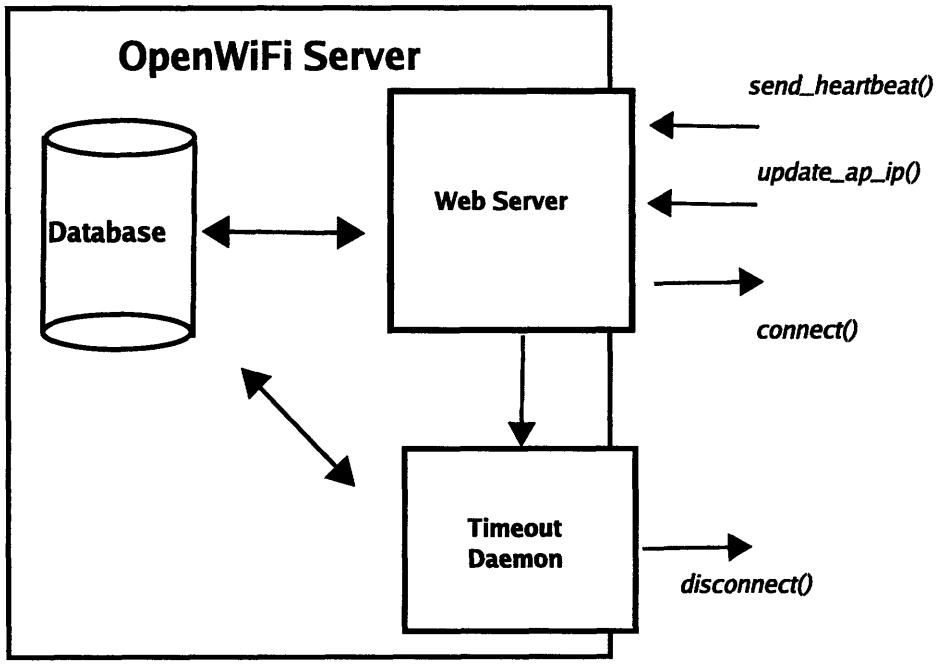

Figure 4-2: OpenWiFi Server Diagram

4.2.1 Web Server

The web server allows both clients and AP operators to register or update their account information and download certificates. The web server uses a two factor authentication scheme requiring certificates and a password to log in users.

4.2.2 Database

The database contains three tables: **client_accounts**, **ap_accounts**, and **current_users**. The **client_accounts** table and the **ap_accounts** tables store identification and payment information for clients and AP owners respectively. When a client or an AP operator registers with OpenWiFi, his information is permanently stored in a row in the appropriate table.

In the schema for the **client_accounts** table (Table 4.1), the *username* and *passwd_hash* fields contain a client's username and a hash of his password. The *mac_address* field contains the MAC address of his WiFi hardware, used for packet filtering at the AP level. Finally, the *billing_info* contains the billing address and

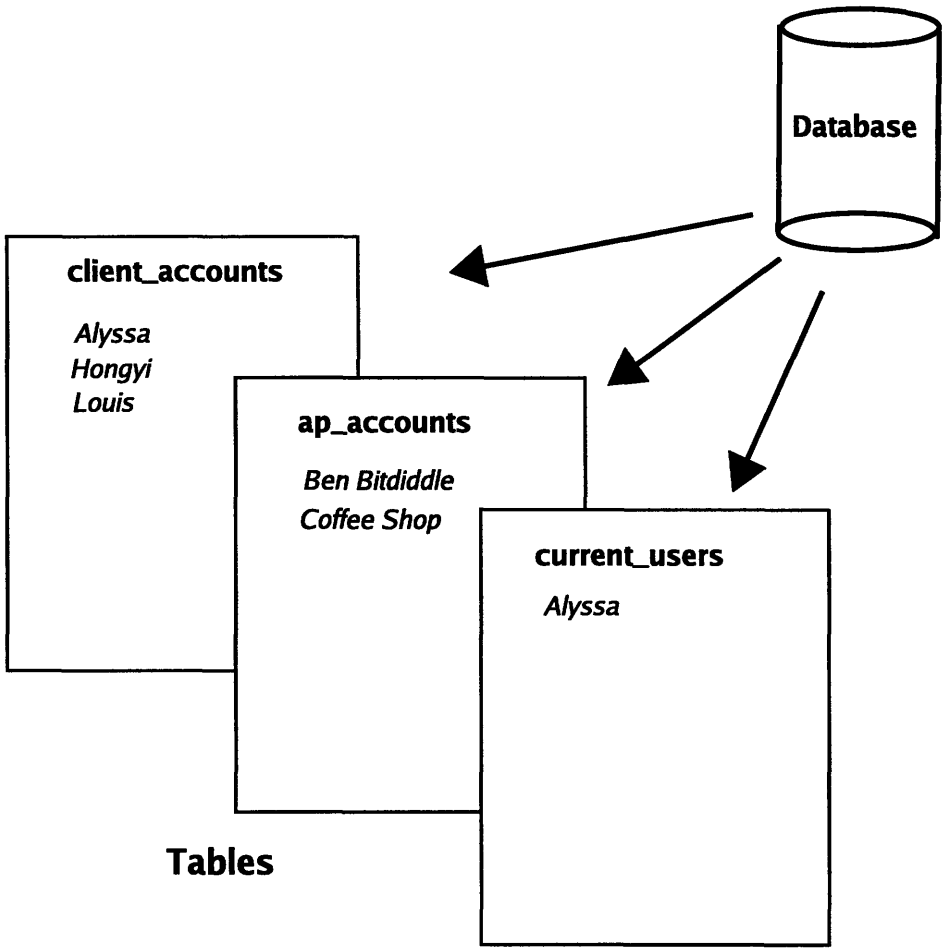

Figure 4-3: Database Diagram

potentially other billing information for the client.

The schema for **ap_accounts** (Table 4.2) is almost identical to **client_accounts**. The main difference between the **ap_accounts** and the **client_accounts** table is that we track the AP's IP address instead of MAC address. We expect that most AP operators will own only one AP, but there will be multiple rows in the **client_accounts** table for those who own more APs.

The **current_users** table (Table 4.3) is a dynamic table that stores heartbeat information for clients who are actively using the service at the moment. The *ip_address* field refers to the IP address of the AP that the client is currently connected to, and the *first_heartbeat* and *last_heartbeat* fields refer to the timestamps of the first and latest heartbeats recorded for that client.

username	passwd_hash	mac_address	billing_info
alyssa	12vPsqi5NBQu6	05:ac:5e:39:10:f0	32 Vassar, MIT
louis	56KerTlWd826Y	fc:40:bb:e3:90:02	Pierce Hall, Harvard
rob	40bExwLabD0E1	ab:12:15:19:2a:7f	800 Commonwealth Ave, BU

Table 4.1: **client_accounts** Table Schema

username	passwd_hash	ip_address	billing_info
ben	56KerTlWd826Y	128.30.52.64	77 Mass Ave, MIT

Table 4.2: **ap_accounts** Table Schema

An additional note regarding scale: if one server could handle the load from the entire pool of clients, then it could house the entire database. However, as the size of the client population grows, more servers should be distributed to address the increased load as well as increase reliability in the face of network faults and decrease latency for clients. In a distributed scenario, the **client_accounts** and **ap_accounts** tables should be stored in a replicated network accessible database. Every server stores local **current_users** tables, querying the main database as needed to check credentials or log billing.

4.2.3 Heartbeats

For non-account related operations, the API for the server has two functions, **send_heartbeat** and **update_ap_ip**.

The **send_heartbeat** method takes a *timestamp*, *IP address* and *client certificate* as arguments. Clients call **send_heartbeat** periodically to send heartbeats to an OpenWiFi server, authenticating with their certificates and identifying the AP by the IP address. The timestamp is included for accounting.

The **update_ap_ip** method takes a *username*, *IP address* and *Access Point cer-*

username	ip_address	first_heartbeat	last_heartbeat
alyssa	128.30.52.64	2007-04-16 11:27:53	2007-04-16 11:52:41

Table 4.3: **current_users** Table Schema

tificate as arguments. An AP will automatically call **update_ap_ip** using its owner's username, its current IP address and its certificate whenever its IP address changes.

4.2.4 Timeouts

Each OpenWiFi server runs a **timeout daemon** that periodically checks every row in the *current_users* table for clients who have timed out. The time out is based on a local threshold value (*Server_Timeout*) that is experimentally set, based on network and client conditions. For each client that has timed out, the **timeout daemon** calls **disconnect(client_mac, OW_cert)**, which forces the AP that the client is currently using to disconnect him.

4.3 Access Points

Access Points in the OpenWiFi system generally sit in private residences or small businesses, and provide WiFi service to registered clients who come into range. APs can be configured with any number of hardware and wireless features available on the consumer market. An OpenWiFi AP is required to run three components: (1) firewall, (2) connection server, and (3) quality of service (QoS) package. The AP owner can otherwise configure his AP however he wishes.

The AP's firewall by default drops all traffic from the AP's wireless interface except for packets whose source or destination is the OpenWiFi server. The AP's connection server processes **connect** and **disconnect** requests from the OpenWiFi server by allowing or disallowing a client's hardware MAC address through the firewall. The QoS package applies stochastic fair queuing to the AP's traffic so that no one client can dominate the AP's outgoing connection.

When a client first associates with an AP, he can immediately send and receive data to and from the internet. The AP expects him to send a heartbeat to an OpenWiFi server and receive a response. If the AP does not receive a **connect** request after some timeout period (*AP_Timeout*), then the client's traffic is immediately firewalled. *AP_Timeout* should be set experimentally depending on network and client

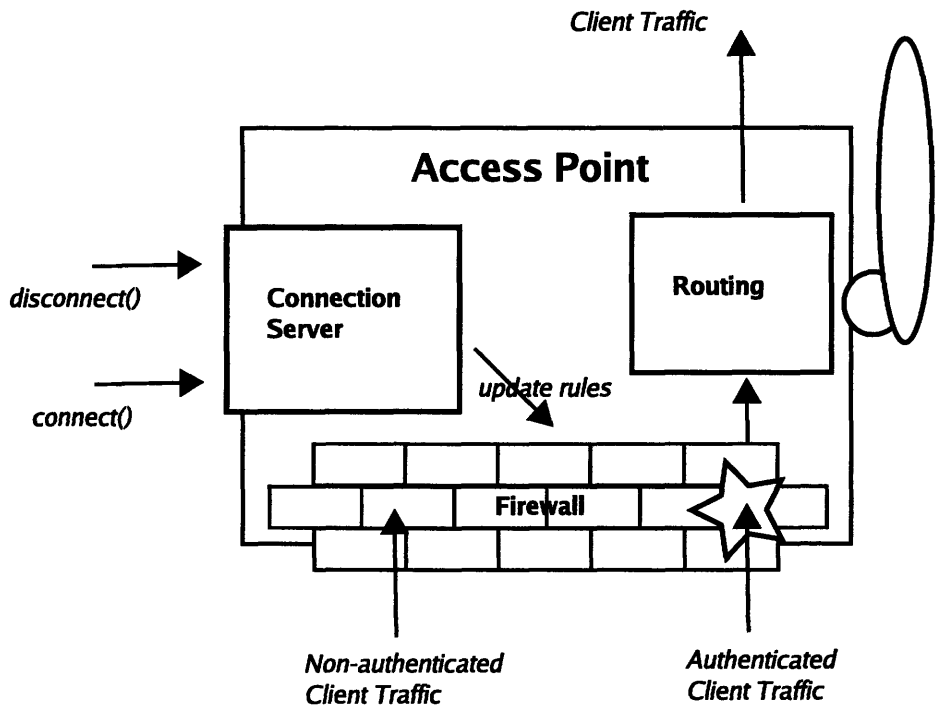

Figure 4-4: Access Point Diagram

conditions. This feature allows high mobility clients to send data even during tiny transmission windows by allowing a grace period for clients who have not yet authenticated themselves. We find in our experiments that a grace period, or *AP_Timeout*, of 10 s is adequate as over 90% of all authentications completed within 10 s.

4.3.1 Firewall

The AP firewall by default has three basic chains, the **INPUT** chain, the **OUTPUT** chain and the **FORWARD** chain. All client packets to the internet traverse the **FORWARD** chain, so OpenWiFi-related packet filtering occurs in that chain.

We add two new chains, the **OW AUTH** chain and the **OW WAIT** chain to the firewall. The **FORWARD** chain's rules are

1. JUMP to **OW WAIT**
2. JUMP to **OW AUTH**
3. REJECT

When a client first connects to an AP, the AP adds a rule accepting all traffic from his MAC address to the **OW WAIT**. After the client exceeds *AP_Timeout* without authenticating himself, this rule is removed.

The client is authenticated by an OpenWiFi server when the AP receives a **connect** request. The **connect** method takes *client MAC address* and *OW certificate* as arguments. When it is called, the AP's firewall adds a rule to the **OW AUTH** chain that accepts all packets from the specified MAC address.

After a client stops sending heartbeats and times out at the OpenWiFi server, the AP should receive a **disconnect** request. The **disconnect** method also takes *client MAC address* and *OW certificate* as arguments. When it is called, the AP's firewall removes the rule to the **OW AUTH** chain that accepts all packets from the specified MAC address.

4.4 Clients

Clients use the OpenWiFi service with any WiFi-capable device that has a standard web browser with JavaScript support. Since clients do not require special hardware or software, new people can try out and join the service easily. A client registers the hardware MAC address of his wireless device with the OpenWiFi server. To send heartbeats, a client visits a particular web page on an OpenWiFi server through a participating OpenWiFi AP. The web page calls **send_heartbeat** on the server. We will explain this mechanism in more detail in Chapter 5.

4.5 Authentication Protocol

To receive Internet service, a client associates his device with an OpenWiFi AP in wireless range and sends a certificate authenticated heartbeat to an OpenWiFi server. The OpenWiFi server sends an **connect** request to the client's AP, and the AP adds a temporary rule to its firewall to allow traffic from the client.

When the OpenWiFi server stops receiving heartbeats from a client after *Server_Timeout*

seconds, it sends a **disconnect** request to the client's AP. If the server hears a new client heartbeat coming from a different AP than the one before, the server sends a **disconnect** request to the old AP and a **connect** request to the new AP. Thus, we maintain the invariant that only one AP provides service for a particular client at any given time.

Heartbeats are sent over TCP and not UDP, so we do not have to worry about how packet loss or delay affects heartbeats. TCP gives us reliable transmission of heartbeats whereas UDP does not.

4.6 Billing

A client's usage is measured by sessions at each OpenWiFi AP he uses. The session length at a particular AP is the interval between the timestamps of the first and the last heartbeat the client sends to the OpenWiFi server from that AP. The client is billed at some rate x per unit time. This rate x depends on market conditions, competition, and other factors, and it may even vary depending on location (e.g., city, state or country). AP operators are paid back at a rate $f \cdot x$ per unit time ($0 < f \leq 1$) based on the client sessions at their AP.

We achieve a good approximation of fair billing by time. We can guarantee that a client's session length at a particular AP is an underestimate of the actual time the client is connected to that AP. Our approximation factor is *Server_Timeout* since a client who ceases heartbeats will be disconnected from an AP after he exceeds *Server_Timeout*. From our experiments, we found that a good value of *Server_Timeout* should be on the order of seconds, so we should generally have a very close approximation for session length.

We also prevent profitable colluding of malicious clients and AP operators. Since we pay AP operators at a rate $f \cdot x \leq x$, there is no payment cycle involving clients and AP providers that results in non-zero profit. Without loss of generality, suppose an AP operator colludes with m clients. The rate at which money is earned by APs is $m \cdot f \cdot x$, and the rate at which money is paid by clients is $m \cdot x$. Therefore, the

net rate of profit is $m \cdot f \cdot x - m \cdot x = m \cdot x(f-1)$. Since $f - 1 \leq 0$, the net rate of profit $m \cdot x(f-1) \leq 0$ for any $m \geq 0$ and $x \geq 0$.

It is important to note that while we describe monetary incentives here, we can use any general type of incentive in our design. Our design for OpenWiFi is modular, so an incentive system based on social networks can be interchanged with a monetary incentive system without additional functionality.

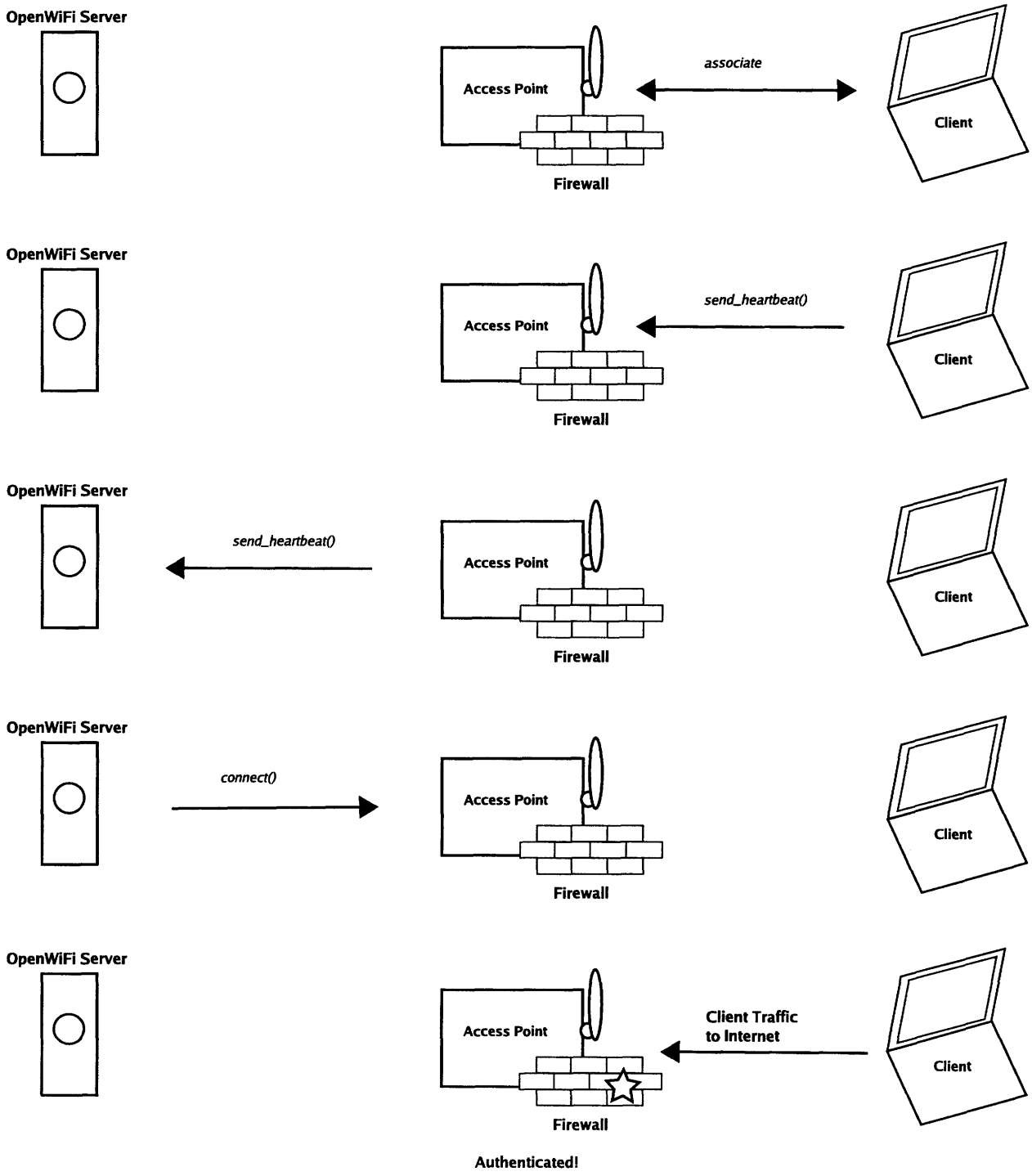

Figure 4-5: Authentication Protocol

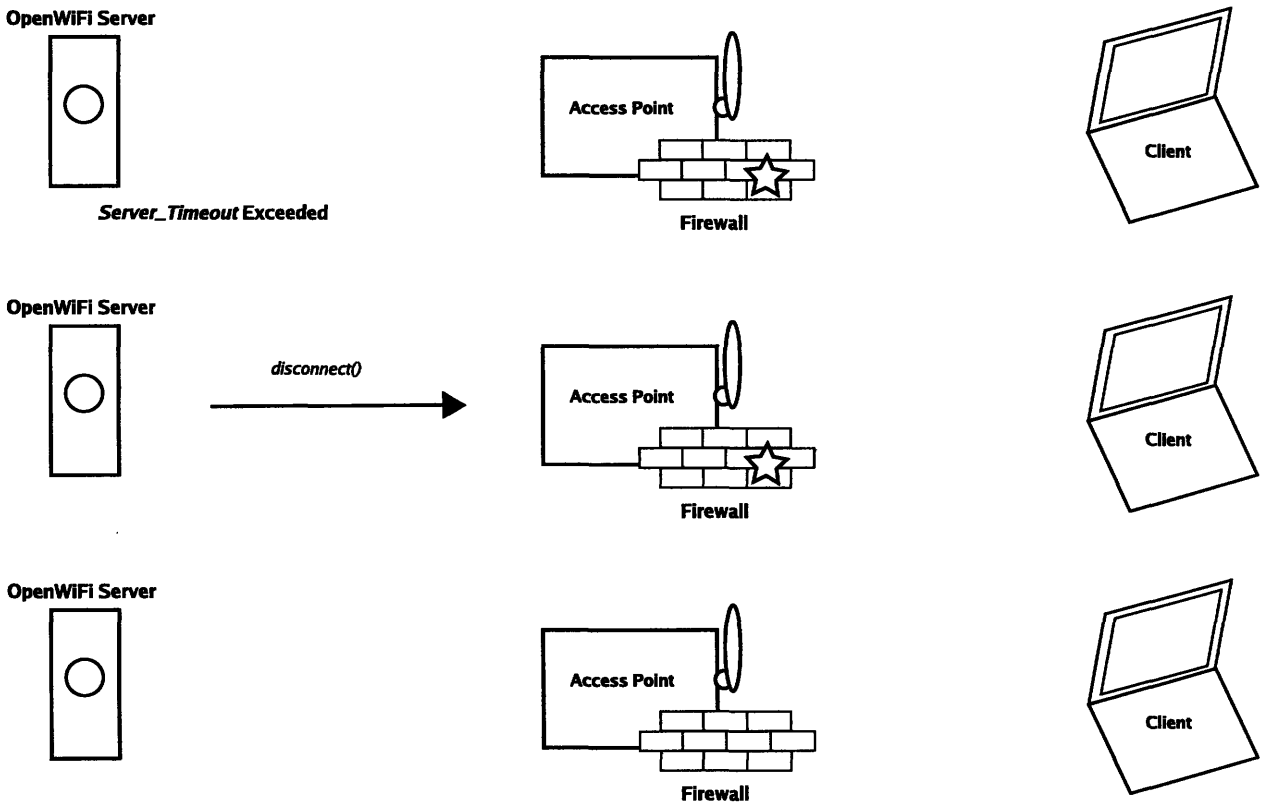

Figure 4-6: Authentication Protocol After Timeout

Chapter 5

Implementation

We implemented a proof-of-concept version of OpenWiFi using a centralized server. Our OpenWiFi server runs Apache with a MySQL database and uses CGI authentication and billing scripts to handle heartbeats and other server operations. In our experiments, our APs were off-the-shelf Linksys wireless routers with modified firmware running several customized scripts, and our clients were a laptop running a stock Linux system and a pair of embedded lin ux machines installed in automobiles.

5.1 Server

Our server ran Apache 2 web server frontend and a MySQL database backend. We wrote several Python CGI scripts to handle certificate generation and management and process heartbeats.

5.1.1 Database

We created an **openwifi** database that contains three tables: the **ap_accounts** table, the **user_accounts** table and the **current_users** table. These tables use the schemas described in our design. We used MySQL 5.0 with autocommit disabled to increase performance.

5.1.2 Web Server

We generated our own OpenWiFi CA PKI using OpenSSL's builtin package. Apache was configured to serve a set of nonauthenticated pages and CGI scripts for registering new users and a set of authenticated scripts for processing and timing out heartbeats. We provided two virtual hosts, one over port 80 (unencrypted) and one over port 443 (encrypted).

5.1.3 Scripts

We implemented heartbeats as HTTP GET requests for a CGI script over SSL. When a client requests the URL for the heartbeat script, his browser and the OpenWiFi web server mutually authenticate, and the web server executes the script. For ease of use, this script returns a web page with some small JavaScript code that periodically reloads the URL. The heartbeat script takes the client's username and the currently associated AP, and it queries the **current_users** table for the client's current status. If the client was last associated with another AP, the script sends a **disconnect** request to the previous AP and removes the client's old entry from the **current_users** table. Finally, the heartbeat script sends a **connect** request to the currently associated AP and inserts a new row for the client, recording the client's username, the AP's name and the timestamp of the HTTP GET request.

A timeout script written in Python runs continuously in the background as a daemon. It periodically checks the timestamp of the last heartbeat from each client in the **current_users** table and compares with the current time. If there has been no heartbeat from a client for a period exceeding *Server_Timeout*, then the script sends a **disconnect** request to the client's last associated AP.

5.2 Access Point

Our access points were Linksys WRTSL54GS wireless routers running OpenWRT RC6. We chose OpenWRT because it offered a relatively stable and flexible embedded

Linux platform that afforded complete control over the routers' features.

We used iptables to handle our firewall functions. OpenWRT already offered a minimum set of iptables rules to handle access control, forwarding and NAT, so we simply built an additional framework on top for OpenWiFi. By default, all traffic over the wireless interface was dropped except for packets with the OpenWiFi server as the source or destination address. We added two chains called **OW_WAIT** and **OW_AUTH** to the end of the default **FORWARD** chain as described in our design.

We ran a tiny HTTP server based on mini-httpd as our connection server. Mini-httpd is a very lightweight web server that supports SSL connections and CGI scripting. As was the case with heartbeats, we implemented **connect** and **disconnect** requests as HTTP POST requests for Haserl CGI scripts over SSL. Each script takes a MAC address from the OpenWiFi server through a CGI form and either adds or removes a rule allowing traffic to and from that MAC address to the **OW_AUTH** chain.

To implement an initial grace period timeout, we attached a small script to dnsmasq, the DHCP server on OpenWRT. Every time the DHCP server assigns a new address to a client, our script adds a rule to **OW_WAIT** accepting all traffic from that client's MAC address, and our script sets a timer to expire after a period of *AP_Timeout* seconds. Once the timer expires, our script removes the accepting rule for the client from **OW_WAIT**. If the client authenticated himself before the timer expired, then he will have a rule in **OW_AUTH** that will accept traffic from his MAC address, so there will be no break in connectivity.

5.3 Client

Clients simply require a JavaScript enabled web browser that supports SSL certificate authentication to use our OpenWiFi implementation. It is also possible to use a Linux commandline utility such as *curl* for advanced scripting, but that is entirely optional.

Chapter 6

Evaluation

This chapter evaluates the OpenWiFi protocols and our implementation. It also analyzes the results of our experiments using our implementation.

6.1 Security

OpenWiFi provides no mechanism to keep client transmitted data private. All wireless transmissions are sent in the clear, so they are at the mercy of eavesdroppers. This could be a significant problem if OpenWiFi served many customers since it would be a gold mine for any malicious agent who wished to steal credit card and other sensitive financial information. In our model, clients can only trust the OpenWiFi server; even the AP owners could be potentially malicious. We could offer VPN service, but that solution does not scale. In addition, VPNs would incur a great deal of overhead; a client traveling in a car would generally not have much time to connect to an AP, authenticate, and establish a VPN session before he moves out of range. At best, we would offer VPNs as an optional, distributed feature. We believe that data privacy is best addressed above the OpenWiFi layer.

6.2 Rate Limiting

OpenWiFi has no technical mechanism for enforcing reasonable rate-limiting at the AP level yet. Since AP owners have full control of their hardware, they also fully control the internet service that they provide to clients. For example, a malicious AP owner could configure his APs to only forward heartbeats. He would still be paid while denying service to clients. In addition, a malicious (or stingy) AP owner could impose an extremely low bandwidth ceiling at his APs for any OpenWiFi client.

There is no way to prevent the first case without introducing some kind of proxy; if a client used a random proxy to send heartbeats, then the malicious strategy of only allowing traffic to the OpenWiFi server fails. The AP owner would be forced to guess which packets from the client are heartbeats and which are regular traffic. There are potentially ways to defeat random proxies, but there is a much heavier burden on the AP owner to do so.

In the other case, clients will generally notice very quickly if they have been heavily rate-limited by their APs. Because our model charges by time instead of bandwidth, it is possible and in fact likely that clients will encounter varying bandwidth ceilings as they roam between OpenWiFi APs. While we cannot prevent AP owners from rate-limiting their hardware as they wish, clients are free to not use APs with stringent bandwidth allocations. AP owners have an incentive to offer bandwidth at competitive rates or otherwise they lose business. In addition, clients could report to the OpenWiFi servers any APs that do not meet some minimum bandwidth ceiling level (e.g. "abuse reports"). A sufficient number of abuse reports, which are authenticated, for a particular AP may result in the AP being blacklisted.

6.3 Authentication

Certificate authentication before allowing client access provides "strong" security (the same level as online bank transactions or purchases) at the expense of high latency. To alleviate this issue, we considered two possibilities: pre-authentication and cookies.

6.3.1 Pre-Authentication

We could pre-authenticate clients at APs in proximity to dramatically reduce hand-off latency. Kassab et al. showed how to use predictive pre-authentication to reduce hand-off latency in 802.11i [14]. Similarly, we could use geographic information about AP locations to predict where a client is and where he will travel to pre-authenticate. Doing so would certainly cut authentication delay significantly, but unfortunately, it pushes a great deal of computation on the OpenWiFi service to support this pre-authentication feature. In addition, pre-authentication would violate our invariant; only one AP open at any given time for a client. Pre-authentication could potentially lead to cheating since a client could give his friends his OpenWiFi authentication credentials so that they could all use different, albeit geographically close, APs. Because of these reasons, we decided against pre-authentication.

6.3.2 Cookies

Alternatively, we can require a one-time upfront authentication with certificates and use encrypted cookies every time afterwards. Every day, when a client first sends a certificate authenticated heartbeat to an OpenWiFi server, the server attempts to set a cookie on the client's browser. Afterwards, every time the client sends a heartbeat, the heartbeat is encrypted with his certificate but authenticated with his cookie. Cookies expire after one day to limit brute force attacks.

This cookie scheme allows us to trade off some security for lower overhead on heartbeats. There are some security risks. For example, it is possible for a third party to steal a client's cookie data via cross-site scripting[7] or other techniques.

However, other features of our design mitigate these risks. First, cookies expire after one day on the server, which puts a burden on malicious parties to steal cookies daily. Second, our design maintains the invariant that only one AP is open for a client at any given time. If a malicious party and the actual client attempted to use OpenWiFi at the same time, we could easily detect this scenario server-side. Finally, we can make cookie use optional for authentication to push the trade off decision

to the clients. High mobility clients may choose lower overhead cookies with more security risks while others may opt for more secure certificate authentication all the time.

We tested the performance of both certificate authentication and cookie authentication in our wireless experiments.

6.4 Experiments

We conducted two sets of experiments to test the performance of our system and to evaluate the relative performance of certificate authentication versus cookie authentication. First, we ran static experiments under controlled conditions to analyze system behavior for zero or low-mobility clients. Then, we ran field experiments using WiFi enabled cars to analyze system behavior for high-mobility clients.

6.4.1 Low Mobility Experiments

For our static client experiment, we installed a Linksys WRTSL54GS router on a 100 Mbps network that was only five hops away from our OpenWiFi server. Our client device was a Pentium-M 1.6 GHz laptop with 512 MB RAM and an Orinoco 802.11b wireless card as a client device. The laptop ran Ubuntu 6.10 (Edgy Eft) with the *orinoco_cs* wireless driver. Our server was an Intel Pentium 4 machine running at 1.8 GHz with 1 GB of RAM, and it sat on a 1 Gbps wired network.

In this experiment, we intended to measure system latency given a stable WiFi connection. Our laptop was set down in a static location running a test script that used curl to send heartbeats to the OpenWiFi server. For each trial, the script would do the following:

1. Associate with test Linksys AP.
2. Request IP address.
3. Send heartbeat.

4. Ping to local gateway.

We measured the time elapsed after sending a heartbeat to the first acknowledgement received from the local gateway, and we took this value to be the authentication latency for our system. Our average authentication time for certificates was 1.31 s with a standard deviation of 0.15 s. Our average authentication time for cookies was 1.16 s with a standard deviation of 0.04 s. Compared to previous results, our implementation has a slightly higher overhead than full EAP/TLS with RADIUS authentication which had an average latency of 1.1 seconds [14]. The relatively high latency in our implementation for certificates is not surprisingly given the performance impact of SSL. However, our average latency for both cookies and certificates is indeed comparable to that of experimental results of EAP/TLS with RADIUS. In addition, cookies seem to perform better than certificates approximately 80% of the time under static conditions. We believe that the other 20% of our data which shows that certificates perform better than cookies can be explained by transient network conditions. We performed the certificate test at a different time than the cookie test where network congestion was an uncontrolled factor. A network traffic spike during our cookie test could have added a significant amount of delay to our data.

6.4.2 High Mobility Experiments

For our high mobility experimenst, we used two cars outfitted with Soekris net4801 devices [10]. The Soekris boxes have a CPU speed of 233-266 MHz, 256 MB of RAM and use a Ubiquity Networks SR2 WiFi card with the madwifi-ng driver. Our goals for the experiments were three-fold:

1. Estimate overall heartbeat latency in the field for vehicular mobility.

2. Analyze heartbeat interarrival times to determine a good value for *Server_Timeout*.

3. Analyze OpenWiFi download and upload performance.

Overall, we ran several thousand tests over the span of several weeks. Figure 6-2 shows the distribution of WiFi connection duration that we achieved. The average

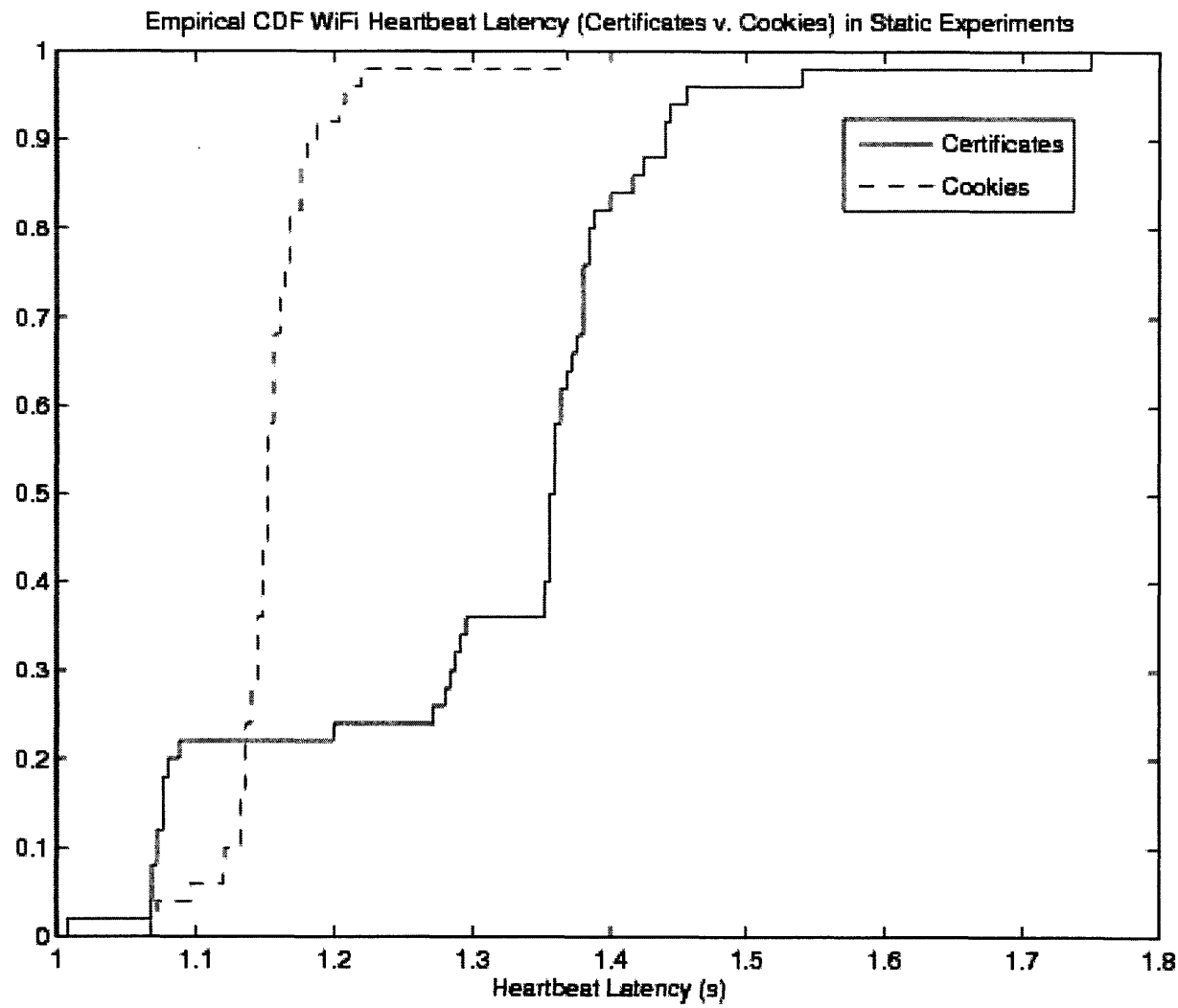

Figure 6-1: Empirical CDF of OpenWiFi authentication time in static experiments, comparing certificates with cookies. Cookies tend to perform better than certificates. The lower portion of the graph where certificates seem to perform faster should be an artifact.

connection lasted 47 s with a very large standard deviation of 96 s. We can explain this large variance through the behavior of our cars. Our test vehicles spent a significant amount of time idling near publically available WiFi, which resulted in high measured connection times. They spent the rest of their experiment time driving through local traffic, which typically results in low measured connection times. Approximately 20% of all connections lasted less than 1 s, and around 50% lasted 10 s or less. About 90%

of all connections lasted three minutes or less.

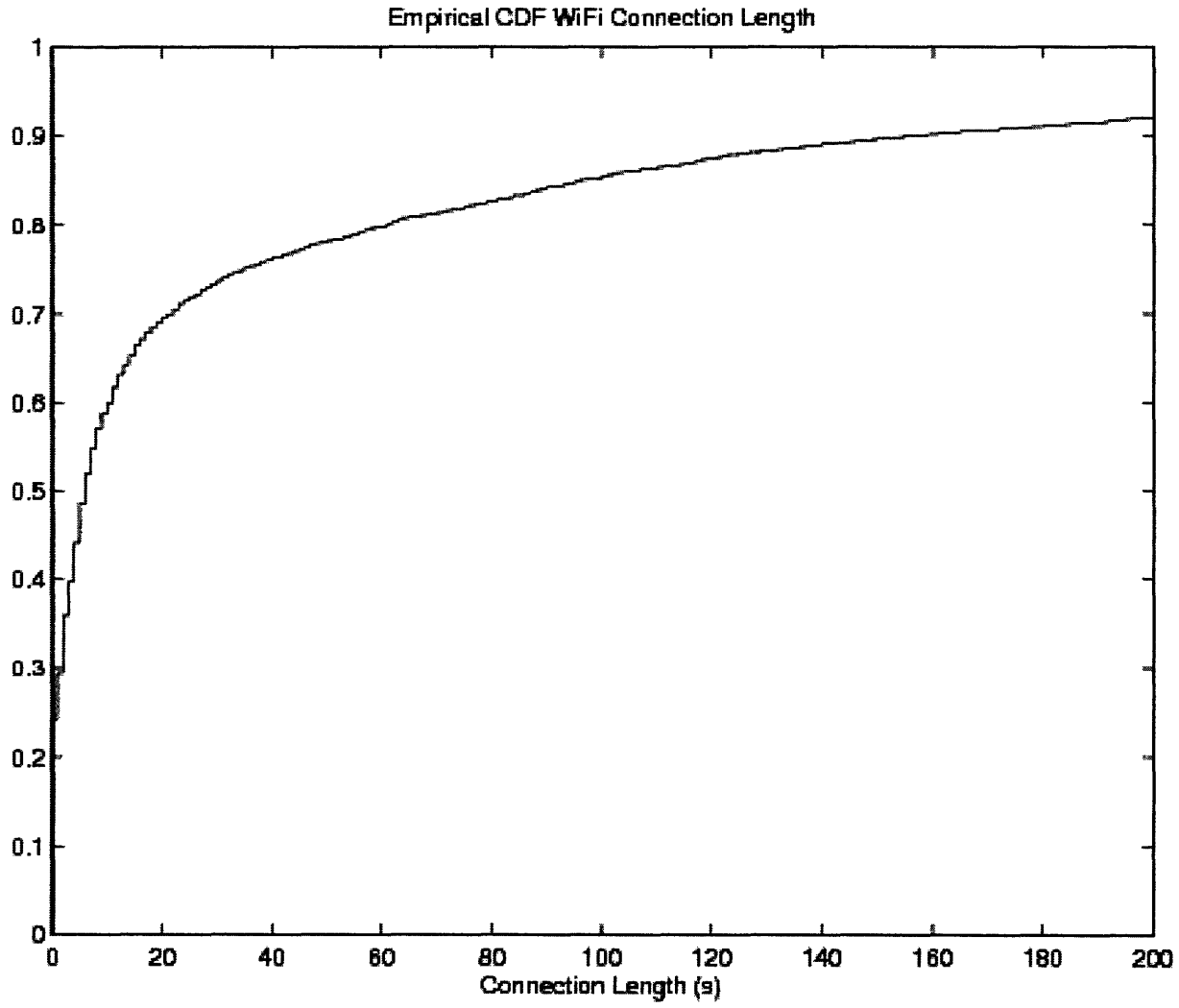

Figure 6-2: Empirical CDF WiFi connection duration in vehicular experiments. Average duration was 47 s. Most connections lasted 10 s or less.

Heartbeat Latency

Since a widespread deployment of OpenWiFi APs in the greater Boston metropolitan area was infeasible, we modified the Soekris devices to act both as clients and as APs in order to mimic our system as closely as possible. To measure heartbeat latency, we used a similar heartbeat sending script from the low mobility experiment as our

client script. We also modified the Soekris devices to act as APs so that we could receive responses from the OpenWiFi server. We were able to estimate the overall time for a client device traveling at vehicular speeds to authenticate to a relatively close OpenWiFi server and reach a usable state, i.e. able to send packets to the internet.

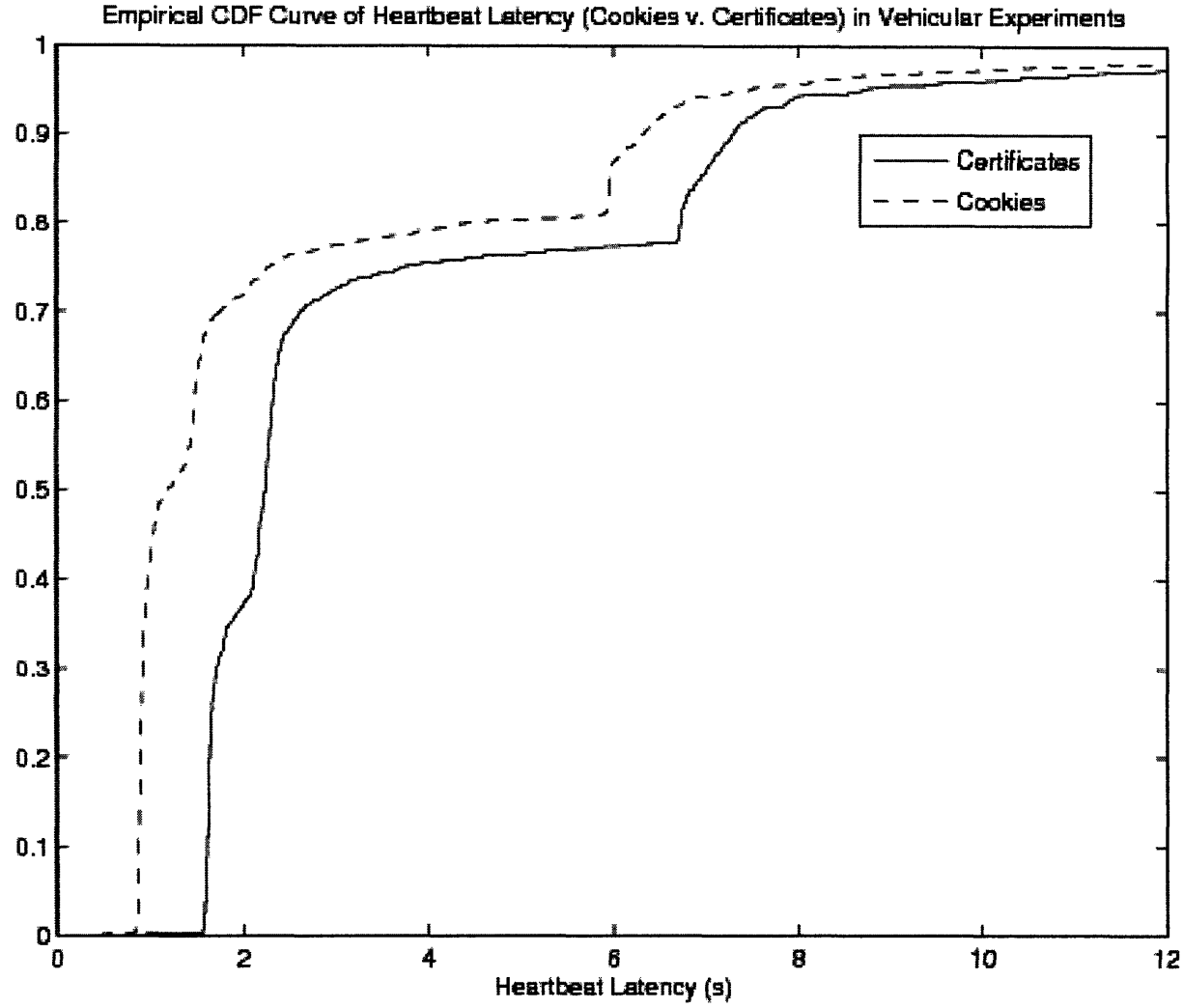

Figure 6-3: Empirical CDF heartbeat latency in vehicular experiments, comparing certificates with cookies. Both tests ran under identical network conditions. Cookies clearly perform faster than certificates.

The mean latency for certificate-based heartbeats was 4 s with a standard deviation of 4 s, and the mean latency for cookie-based heartbeats was 3 s with a standard

deviation of 3 s. It is clear that cookies have a definite performance advantage over certificate authentication. From Figure 6-3, at least 95% of all heartbeats were processed within 10 s, suggesting that an *AP_Timeout* value of 10 s should be adequate.

Heartbeat Interarrival Times

On the OpenWiFi server, we accumulated a log of all the heartbeats sent from our cars over the course of several weeks. We built an associative array that mapped IP addresses to lists of heartbeats recorded from those addresses. We generated an empirical cumulative distribution curve of heartbeat interarrival times to find a good empirical estimate for the *Server_Timeout* value.

In Figure 6-4, we plotted an empirical CDF of heartbeat interarrival times for both certificate-based and cookie-based heartbeats. we eliminated any values above 60 seconds as they indicated the reuse of IP addresses over multiple associations with the same AP. The mean interarrival time for certificate-based heartbeats was 4.6 s with a standard deviation of 4 s, and the mean interarrival time for cookie-based heartbeats was 4 s with a standard deviation of 4 seconds. Once again, cookies have a noticeable advantage over certificates for authentication performance. From the graphs, approximately 95% of all heartbeats have an interarrival time around or below 10 s, so it is also a good empirical value for *Server_Timeout*.

OpenWiFi Download and Upload Performance

On the automobile WiFi boxes, we also ran test scripts that tested download and upload performance over HTTP whenever there was a connection to an AP. One script selected an IP address out of a list of 73 popular websites and downloaded the index page. The other script uploaded a 30KB file to a webserver.

Out of approximately 6700 trials, the mean download speed was 14 kB/s with a standard deviation of 30 kB/s. From Figure 6-5, approximately 80% of all downloads had throughput under 20 kB/s. There is a sudden flattening in the curve for trials with throughput starting around 10 kB/s. We suspect that this is related to the distribution of downloaded file sizes.

Figure 6-7 shows the distribution of the index file sizes from all 73 websites. The average file size was 38.7 KB with a standard deviation of 47.3 KB. From the graph, approximately 50% of all index files were under 5 KB. Since we had a wide range of file sizes, this could explain the behavior we saw in Figure 6-5. Small file sizes would result in smaller throughput as we would not have reached a large TCP window by the time we finished downloading the file. Larger file sizes, on the other hand, would result in larger throughput as we would have had more time to scale up our TCP window than with smaller files.

Out of about 2700 trials, the mean upload speed was 28 kB/s with a standard deviation of 21 kB/s. It may seem odd that our mean upload throughput was higher than our mean download throughput, but the discrepancy can be explained by the file size difference in the two tests. For our upload test, we used a 30 KB file while for our download test, most of our files were smaller, with about half under 5 KB.

To do a valid comparison of our upload and download throughput, we ran another vehicular test to download a 29 KB file consistently from the same source. Both upload and download CDF curves are plotted in Figure 6-8. From the graph, the download curve is to the right of the upload curve. The download curve is not a perfect translation of the upload curve, but it still fits expectations since ISP service for home consumers tend to offer lower upstream bitrate than downstream bitrate.

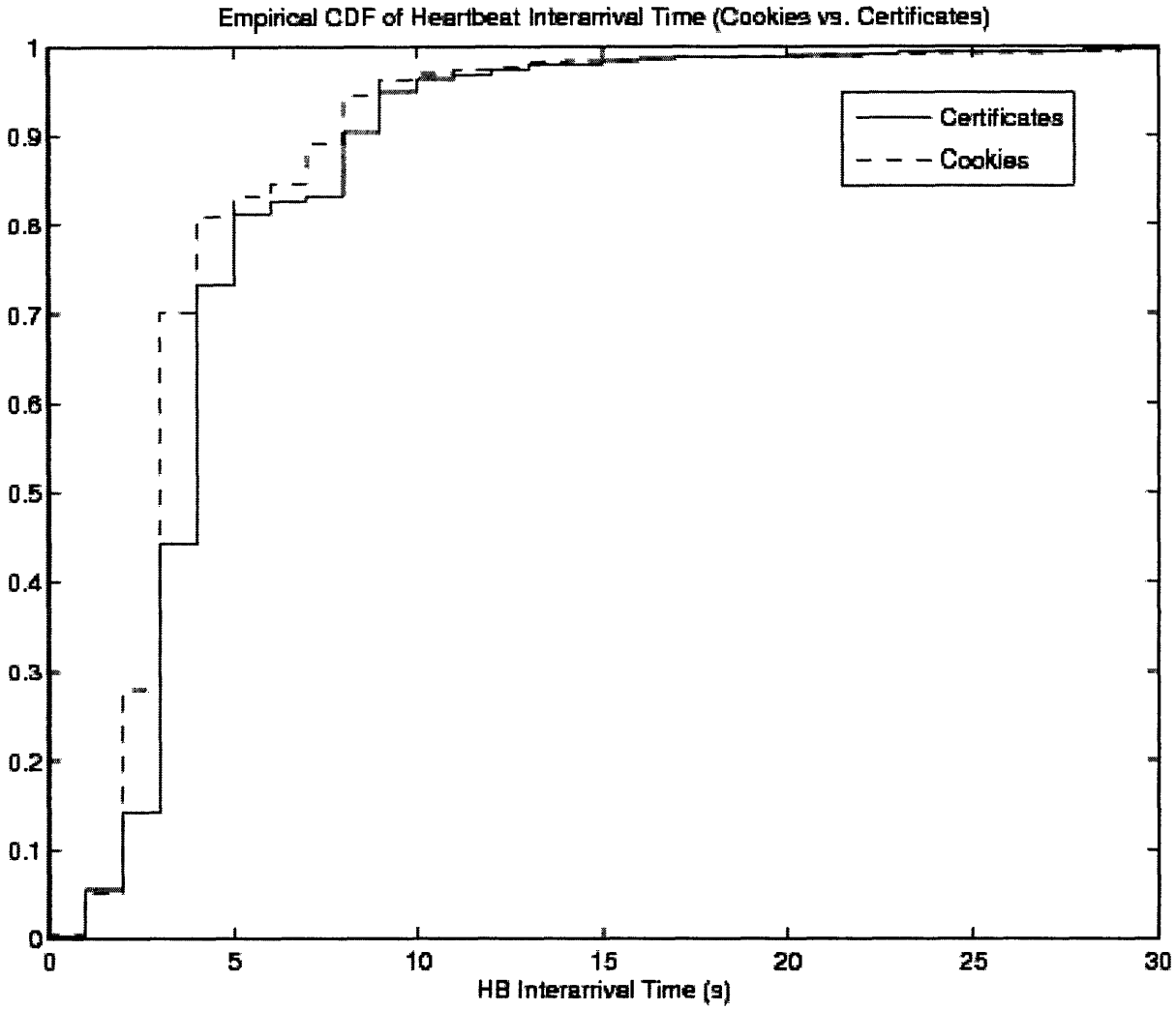

Figure 6-4: Empirical CDF of heartbeat interarrival delay comparing certificates and cookies.

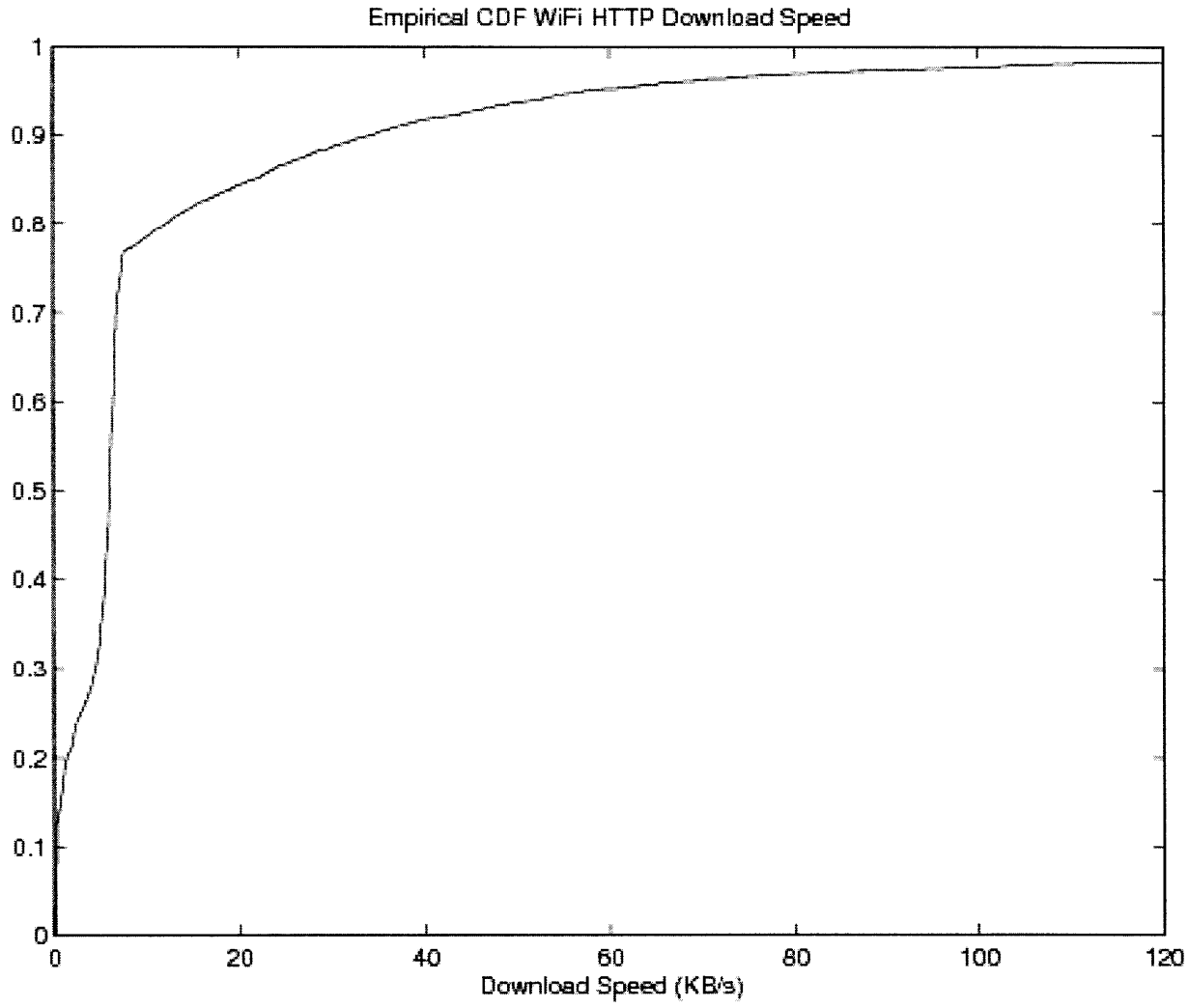

Figure 6-5: Empirical CDF of WiFi HTTP download speed. Each trial downloaded the index page of a website randomly chosen out of a list of 73 popular websites.

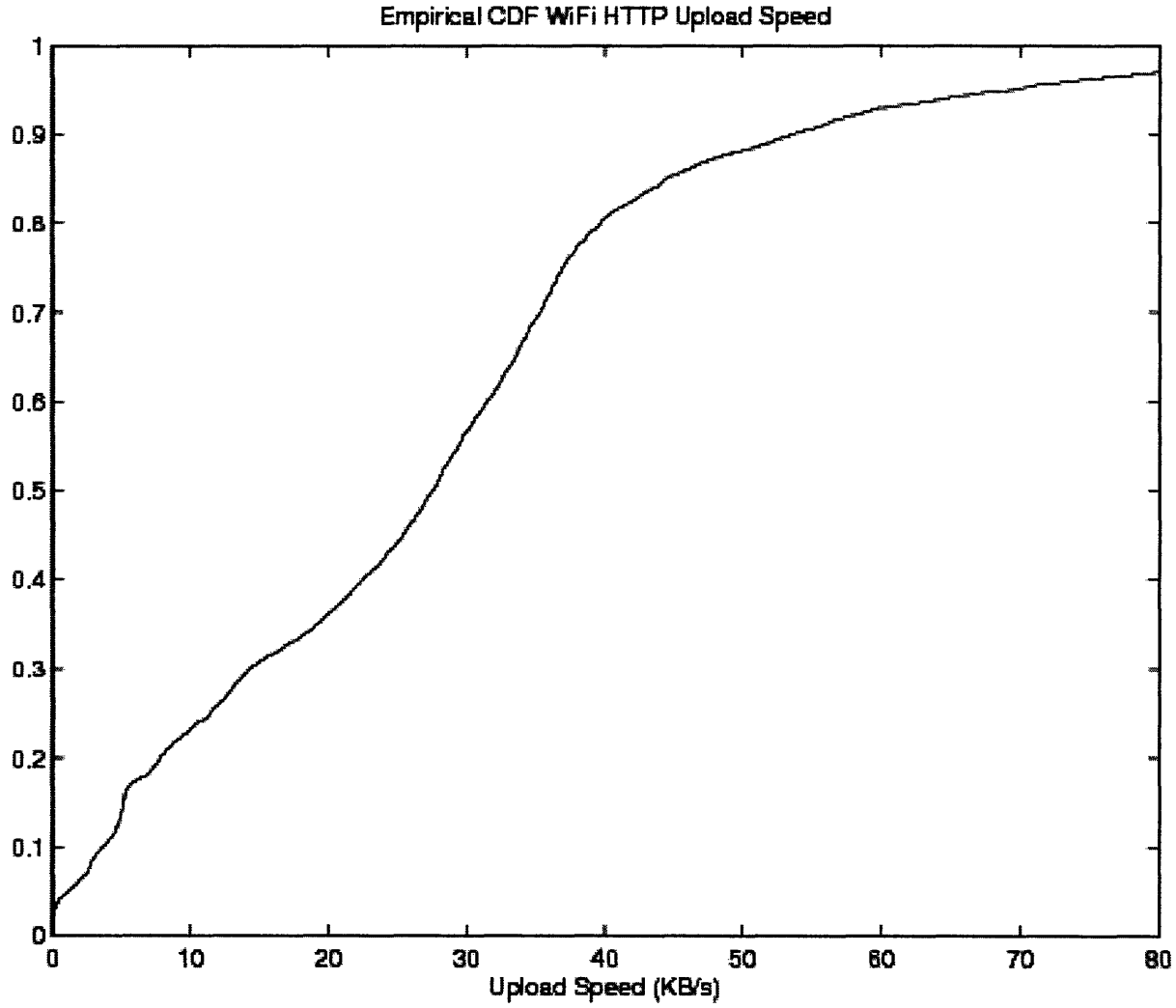

Figure 6-6: Empirical CDF of WiFi HTTP upload speed. Each trial uploaded a 30 KB file to our webserver.

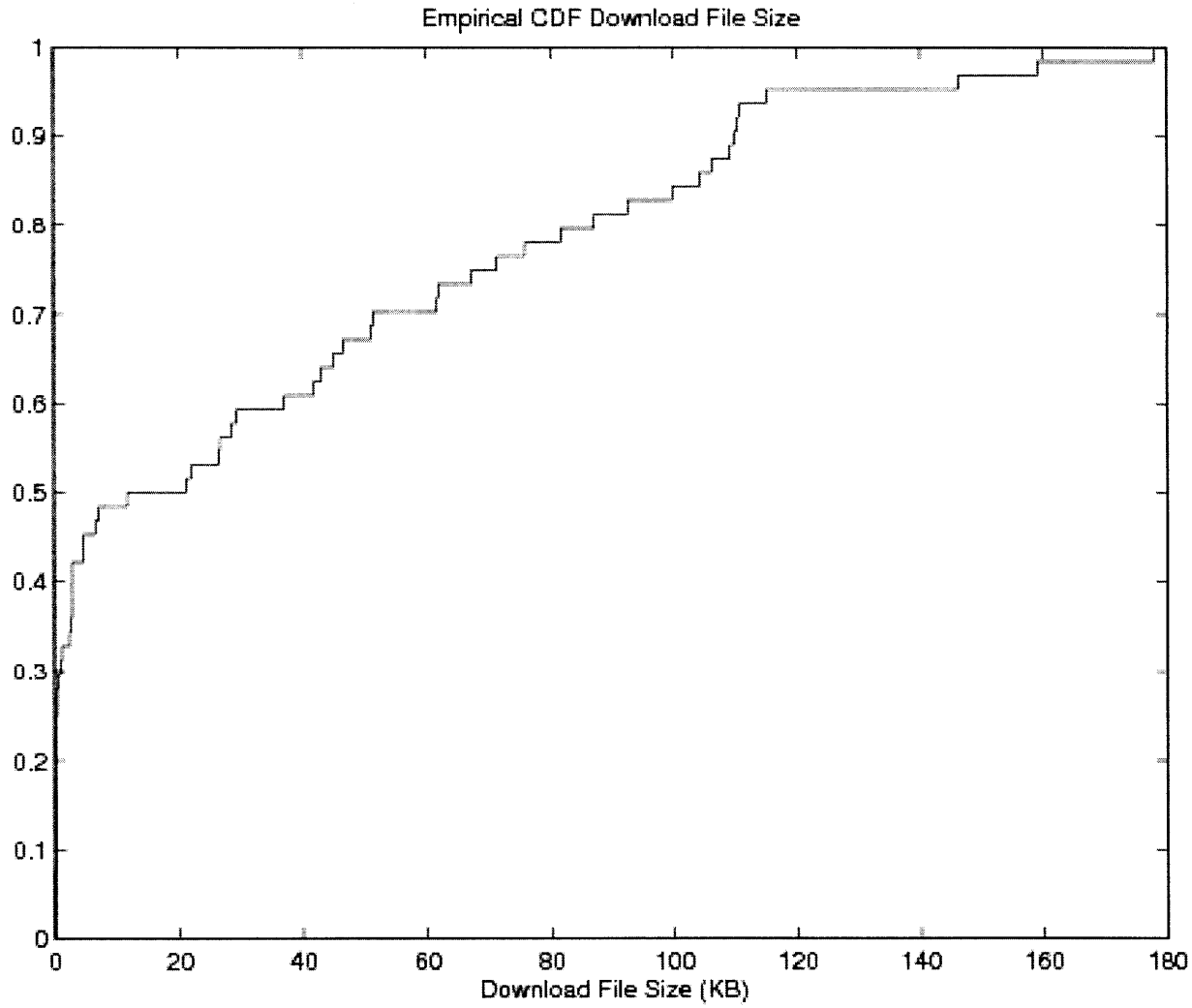

Figure 6-7: Empirical CDF of downloaded HTTP file sizes over WiFi. About half of all files were under 5KB.

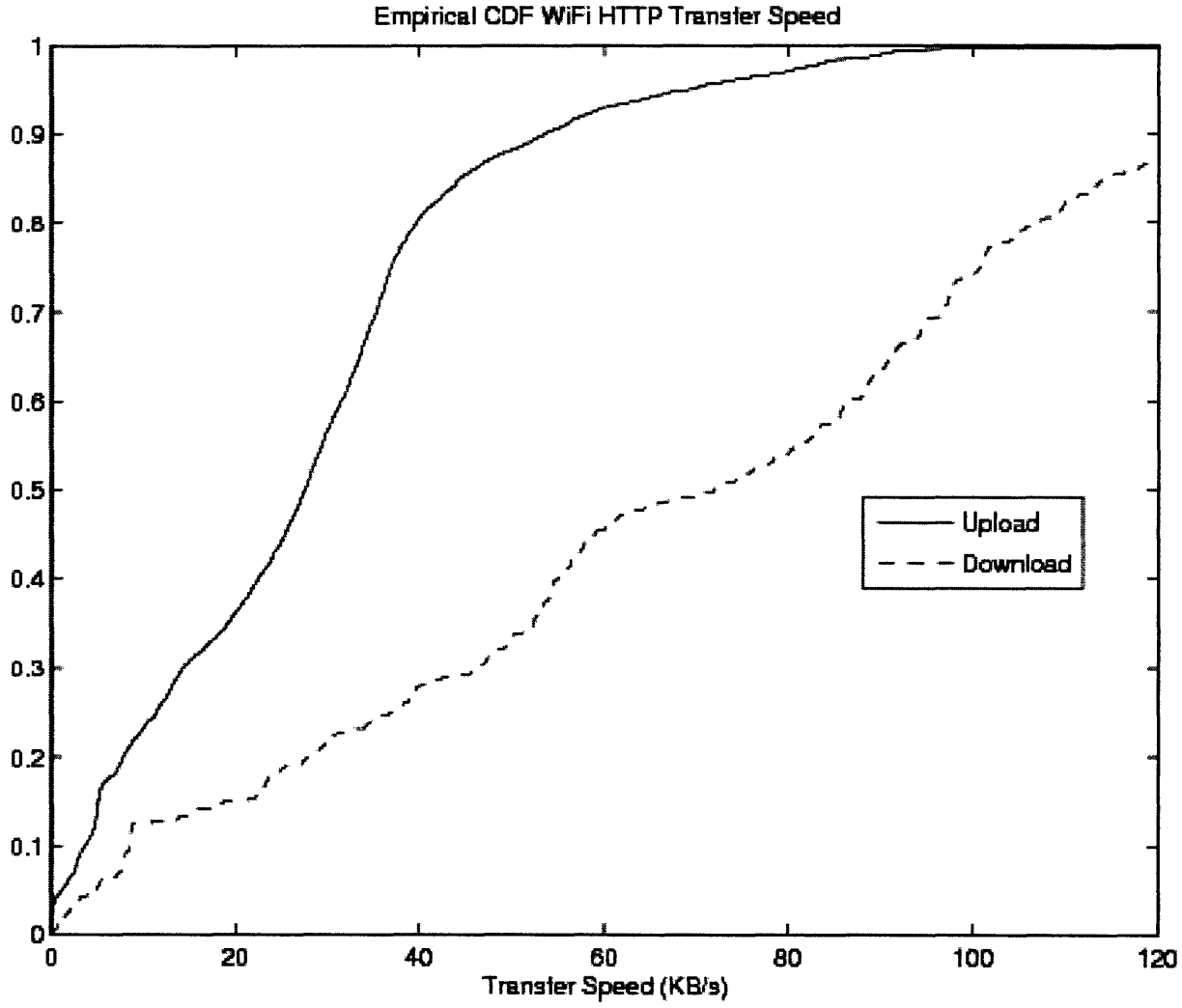

Figure 6-8: Empirical CDF of WiFi HTTP transfer speed comparing upstream and downstream transfer rates. Downstream speed tends to be higher than upstream speed, which fits expectations.

Chapter 7

Conclusion

We designed and implemented a consumer WiFi sharing system that addressed challenges such as fair secure billing, heterogeneous compatibility and support for high mobility applications. We analyzed several different approaches for billing, authentication, and encryption to create a simple but effective design for our system, OpenWiFi.

For billing, we pointed out problems with cheating in a flat rate model and feasibility in a bandwidth-based model, and we recommended a time-based model as the winner. For authentication, we investigated a certificate-based approach and an HTTP cookie-based approach before deciding on a hybrid protocol utilizing both certificates and cookies for a good trade-off in security and performance. Finally, for encryption, we considered client-AP methods such as WPA2 with RADIUS/802.1X as well as end-to-end methods such as virtual private networks (VPNs). All encryption schemes we considered did not meet our design constraints, so that question still remains open.

We ultimately pursued an end-to-end heartbeat-based authentication protocol that ensured fair time-based billing and prevented many types of cheating by clients or AP operators. We also introduced a grace period for high mobility clients to avoid the high association overhead of switching APs.

We tested the performance of OpenWiFi in a number of static and high mobility experiments. We measured authentication latency, heartbeat frequency and

upload/download performance. We also compared the relative performance of cookie-based authentication versus certificate authentication. Our results show that our implementation's authentication latency is comparable to 802.11i with RADIUS when APs and the OpenWiFi server are several hops away under static conditions. Under dynamic, high mobility conditions, most authentications resolved within 4 s, and approximately 95% resolved within 10 s. In addition, we were able to obtain relatively reasonable download and upload performance in such a dynamic WiFi setting. Ultimately, we were successful in making OpenWiFi suitable for both static and high mobility applications.

Bibliography

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 2284, June 2004.

[2] B. Aboba and D. Simon. PPP EAP TLS Authentication Protocol. RFC 2716, October 1999.

[3] Hari Balakrishnan, Samuel Madden, Vladimir Bychkovsky, Kevin Chen, Waseem Daher, Michel Goraczko, Hongyi Hu, Bret Hull, Allen Miu, and Eugene Shih. CarTel: A Mobile Sensor Computing System. MIT CSAIL Research Abstract.

[4] L. Barber. City WiFi Costs Way Too High. *Mobile Magazine*, 2005.

[5] Vladimir Bychkovsky, Bret Hull, Allen K. Miu, Hari Balakrishnan, and Samuel Madden. A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks. In *12th ACM MOBICOM Conf.*, Los Angeles, CA, September 2006.

[6] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588, September 2003.

[7] CERT Coordination Center, DoD-CERT, JTF-CND, FedCIRC, and NIPC. CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests, February 2000.

[8] Chillispot. http://www.chillispot.org.

[9] P. Congdon, B. Aboba, A. Smith, G. Zorn, and J. Roese. IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines. RFC 3580, September 2003.

[10] Soekris Engineering. http://www.soekris.com/net4801.htm.

[11] FON. http://www.fon.com.

[12] Naomi Graychase. Wibiki Strives for Wi-Fi Ubiquity. *Wi-Fi Planet*, February 2006.

[13] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen K. Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: A Distributed Mobile Sensor Computing System. In *4th ACM SenSys*, Boulder, CO, November 2006.

[14] Mohamed Kassab, Abdelfettah Belghith, Jean-Marie Bonnin, and Sahbi Sassi. Fast pre-authentication based on proactive key distribution for 802.11 infrastructure networks. In *WMuNeP '05: Proceedings of the 1st ACM workshop on Wireless multimedia networking and performance modeling*, pages 46–53, New York, NY, USA, 2005. ACM Press.

[15] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Dial In User Service. RFC 2685, June 2000.

[16] Wibiki. http://www.wibiki.com.

[17] Share My WiFi. http://www.sharemywifi.com/.