Improved Access Point Selection

Anthony J. Nicholson[†,*]
tonynich@eecs.umich.edu

Yatin Chawathe[*]
yatin.research@chawathe.com

Mike Y. Chen[*]
mike.y.chen@intel.com

Brian D. Noble[†]
bnoble@eecs.umich.edu

David Wetherall[*,‡]
djw@cs.washington.edu

[†]University of Michigan [*]Intel Research Seattle [‡]University of Washington

ABSTRACT

This paper presents Virgil, an automatic access point discovery and selection system. Unlike existing systems that select access points based entirely on received signal strength, Virgil scans for all available APs at a location, quickly associates to each, and runs a battery of tests to estimate the quality of each AP's connection to the Internet. Virgil also probes for blocked or redirected ports, to guide AP selection in favor of preserving application services that are currently in use. Results of our evaluation across five neighborhoods in three cities show Virgil finds a usable connection from 22% to 100% more often than selecting based on signal strength alone. By caching AP test results, Virgil both improves performance and success rate. Our overhead is acceptable and is shown to be faster than manually selecting an AP with Windows XP.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management, Public networks*
; C.2.4 [**Computer-Communication Networks**]: Local and Wide-Area Networks
; D.4.4 [**Operating Systems**]: Communications Management—*Network communication*

General Terms

Experimentation, Management, Measurement

Keywords

802.11, access point selection, opportunistic connectivity, public networks, wireless networking

1. INTRODUCTION

Mobile users have come to expect nearly constant connectivity, provided in part by the ever-increasing density of wireless access points. 802.11 wireless LAN access points (APs) are increasingly widespread in urban areas, with users commonly finding multiple APs on each scan.

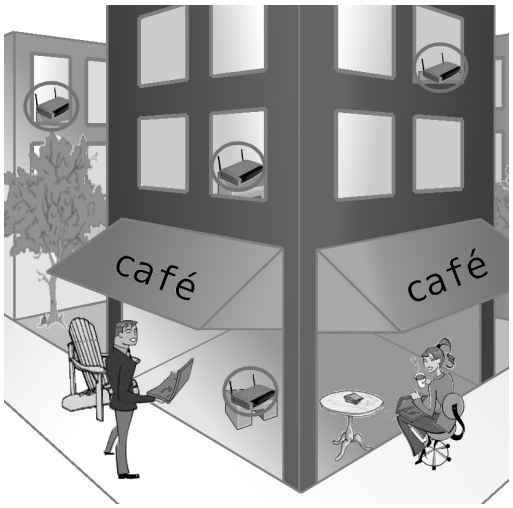


Figure 1: A sea of bandwidth. Users increasingly must choose the "best" access point from many (circled in the illustration).

Access point selection is still a critical problem, however. Consider the scenario illustrated in Figure 1. Customers at a sidewalk cafe encounter four access points—one from the coffee house, two from residents on the floors above (who may not even allow strangers to use their access points), and another next door, part of the city's free WiFi deployment. Which AP will provide the best quality of service?

Unfortunately, these APs are under decentralized control, and are managed by a varied set of residents and businesses. Consequently, many APs reject or restrict foreign users in a variety of ways. Since there is no common administrative control, there is also no centralized database to guide users' selection policies in favor of the APs providing the best service. While many searchable databases of "wardriving" maps exist [16, 30], these maps become outdated quickly and provide no information about access points apart from the basic information broadcast in the beacon signal.

Worse, AP selection is driven by the physical layer. The selection policy currently used by all common operating systems for automatically selecting an access point simply

MobiSys'06, June 19–22, 2006, Uppsala, Sweden.

scans for APs and then chooses the unencrypted one with the highest signal strength. However, this policy, which we call *strongest signal strength*, or *SSS*, ignores other factors that matter to the end user. For example, the AP with the strongest signal might belong to a pay service, to which the user does not subscribe. APs that appear open may use MAC address filtering to block traffic from foreign users. The bandwidth and latency of an AP's Internet connection depends on the type of ISP to which its owner subscribes—a cable modem, DSL, or dial-up. The signal strength of the access point is orthogonal to such considerations.

Currently this problem can be solved by hand-tuning connection preference tables, to enumerate the APs and networks to which one's device should connect. This is only feasible for the most common locations a user visits. At other locations, users might have to try several available networks before finding a usable connection. This is an onerous task at best that should be automatic.

Furthermore, in the future, users' computing devices will increasingly be always-on, pervasive devices with wireless radios. Such devices will continually need to find the best wireless connection at new locations without any user input as their owners move through their daily routine. For such a utopian wireless future to be possible, we need to reduce the friction mobile devices currently encounter when trying to easily find the best available wireless connection.

To determine the scope of the problem, we conducted a small field study, examining the efficacy of strongest-signal selection. The results showed that the SSS algorithm often chose an unusable AP when a usable AP existed, and in fact performed no better than choosing an AP at random. Usable in this case means an AP which both grants an IP address to unknown clients via DHCP, and allows Internet traffic through at least one port. This suggested that signal strength is an insufficient predictor of AP quality.

Ideally, we would want our wireless clients to quickly examine all available connection points, and *automatically* select the one appropriate for our current needs that provides the best quality of service. In this paper, we present Virgil[1], an improved access point selection system. Virgil quickly associates to each new AP found in a scan set and runs a battery of simple tests, designed to probe the AP's suitability for use. We use a small set of *reference servers* spread throughout the Internet to let Virgil estimate expected bandwidth and round-trip-time to remote servers. Users also want seamless mobility from an application-level perspective, but different access points may allow or deny traffic on different network ports. Virgil therefore connects to reference servers on a wide range of TCP and UDP ports to check for port traffic blocked or redirected by each prospective AP. Based on the test results, Virgil chooses the best access point available, rather than guessing based on metrics like signal strength.

Our evaluation results from five neighborhoods in three different cities show Virgil found a usable access point from 22% to 100% more often than selecting based on signal strength. We also showed that maintaining a database of AP test results boosts these success rates even higher for neighborhoods a user visits often. Finally, analysis showed that Virgil's overhead, while not negligible, is still reasonable enough as to be useful to users. Compared with selecting access points manually, Virgil is faster and fully automatic, removing an unnecessary burden from users. Furthermore, overheads in revisited neighborhoods are indistinguishable from that required by SSS policies.

In the course of our field study and subsequent evaluation of our prototype implementation of Virgil, we encountered and tested nearly 4000 access points. Our trace logs and AP databases are freely available to the community through the CRAWDAD[2] archive.

1.1 Legal and Security Issues

Previous "war-driving" studies passively scanned the air for beacon signals that access points willingly broadcast. What we are proposing—actively connecting to each open access point and transmitting a small amount of data to estimate that AP's connectivity to the Internet—arguably raises the question of whether it is legal to connect to any open (but possibly private) wireless network. While most jurisdictions worldwide prohibit unauthorized access to computer systems, it is not clear how these laws apply to using someone's wireless connection [18].

Such concerns are not trivial. However, it is also true that many individuals (and enterprises) are completely willing to allow strangers to connect to the Internet via their wireless networks. Many coffee shops offer free wireless connectivity. Most major cities have one or more "grassroots" wireless collectives, such as the Bay Area Wireless Users Group [2], Seattle Wireless [28], and NYCWireless [23]. Many local governments are deploying free APs in public spaces as well. For example, in our field study, we could often detect APs belonging to the city's infrastructure and to a grassroots organization in the same location. In the presence of such truly open networks, we argue that our technique is still useful. If it were possible to modify the 802.11 broadcast beacons to include an "open" flag, we could leverage it to restrict our search to only open networks.

A second problem with scanning and using relatively unknown wireless networks is caused by the rise of "evil twin" or "pharming" attacks on public access points [3]. In such attacks, a criminal uses his laptop to masquerade as a wireless AP. When other users connect to his "AP", he interposes on all their data traffic before forwarding it on to a valid AP (or simply dropping it altogether). Thus, even if users negotiate encryption keys—by using HTTPS, for example—the attacker can interfere with key establishment and steal all subsequent credit card numbers, bank data, or passwords protected by such session keys.

We argue that if users cannot trust their network access points, end-to-end encryption is the only reliable way to protect sensitive data. Recent work of ours [21, 22] focused on solving this problem of establishing end-to-end trust when neither party trusts any of the intervening network hops completely—not even their network access points. This and other related work [7, 20, 25] are complementary to the main focus of this paper—improving the wireless access point discovery process.

1.2 Contributions

We make the following contributions in this work. First, we show that the AP selection algorithm most frequently in current use (strongest-signal-strength) often performs no better than selecting access points at random. Second, we

[1]In *The Divine Comedy*, Virgil was Dante's guide through the underworld.

[2]http://crawdad.cs.dartmouth.edu/

```
scan for all available APs
log AP beacon information for all APs
for each unencrypted AP do
   try to get IP address by DHCP
   if DHCP successful then
      (1) estimate round-trip-time to reference server
      (2) test open ports
      (3) estimate bandwidth to reference server
```

Figure 2: Field study script.

are the first to illustrate the benefit of quickly associating to each available AP and testing the suitability of each for use. Third, we collected detailed data on the properties of over 4000 real-world access points, including not just beacon frame information but also application-level test results.

2. FIELD STUDY

Many popular operating systems (such as Windows XP, Mac OS, and Linux) use essentially the same policy to guide wireless access point selection when more than one AP is available. If the system finds one of the APs in a list of "preferred networks" explicitly saved by the user, it chooses that AP. Otherwise, it simply scans for all available APs and chooses the unencrypted AP with the strongest signal strength. We will call this algorithm *strongest-signal-strength* or SSS.

The problem of AP selection first drew our interest because we believed selecting APs based on signal strength is often the wrong thing to do. Specifically, we designed a field study to answer the following questions:

1. Do users commonly see multiple access points each time they scan for a new AP?
2. Does strongest-signal-strength selection often choose an unusable access point when a different, usable AP was available?
3. Do usable access points vary significantly with regard to the quality of Internet connection they provide?

By "usable", we mean an unencrypted access point that both grants a DHCP address to anonymous clients and allows Internet traffic through at least one port. For example, a public hotspot that blocks all TCP traffic except port 80 (HTTP) would still be considered "usable".

2.1 Methodology

For our field study setting, we chose Chicago, the third-largest city in the United States (population: 2.8 million [5]). Since all cities have different neighborhoods of varying density, we studied three representative neighborhoods:

- The Loop (Downtown): Chicago's central business district. Workday population density: 235,000/km^2 [5].
- Wicker Park (Residential): A middle-class, high-density urban neighborhood. Residential population density: 7400/km^2 [5].
- Evanston (Suburban): An upper-middle-class suburb and college town, north of the city limits. Residential population density: 3700/km^2 [5].

	Downtown	Residential	Suburban
APs found	797	464	256
APs per scan	2.4	2.0	1.8
APs granted IP address	78 (9.8%)	81 (17.5%)	43 (16.8%)
APs using encryption	445 (55.8%)	287 (61.9%)	148 (57.8%)

(a) All Encountered Access Points

	Downtown	Residential	Suburban
APs granted IP address	78	81	43
Usable APs	42 (53.9%)	81 (100%)	42 (97.7%)
APs redirect port 80	38 (48.7%)	1 (1.2%)	1 (2.3%)
APs with open port 80	37 (47.4%)	75 (92.6%)	39 (90.7%)

(b) Open Access Points

Table 1: Field Study: AP Statistics. We walked an approximately 1.3 km^2 area in each of three Chicago neighborhoods. "Usable APs" were APs that granted an IP address to our handheld device and allowed port traffic through at least one TCP port.

In all three neighborhoods, we walked a 1/2 mi^2 (1.3 km^2) grid of city streets with a PDA containing an 802.11 wireless card. We chose to "warwalk" rather than "wardrive" so our script would have time to associate with found APs and run tests, rather than just log 802.11 beacon information. The PDA ran Familiar Linux, a distribution targeted for handheld devices [13].

Note that these results are not intended to represent any realistic mobility model. We quite literally walked up and down streets in these neighborhoods in a grid fashion. The results in aggregate, however, are useful for drawing conclusions about the quantity, quality and frequency of wireless connectivity available in the target neighborhoods.

We used a Compaq iPAQ handheld with an 802.11b wireless LAN card to collect data on the density and properties of different access points in an urban environment. Figure 2 summarizes the field study script in pseudocode.

The reference server (RS) was a dedicated machine at the University of Michigan, directly connected to the Internet. To estimate round-trip-time, the script simply pinged the RS twice, and used the second result. The RS also ran a simple daemon which listened on 37 common TCP ports, including ssh (22), SMTP (25), HTTP (80), Windows DCOM (135) and Samba (445). To test for open ports, the field study script sent a random integer nonce to the RS on each TCP port number and the RS returned (nonce + 1). We performed this nonce exchange rather than simply testing for establishment of TCP connections in order to verify that the access point was not redirecting traffic on that port to its own server.[3] If the script received anything other than

[3]Many commercial access points redirect all traffic to a special sign-on page (a splash screen).

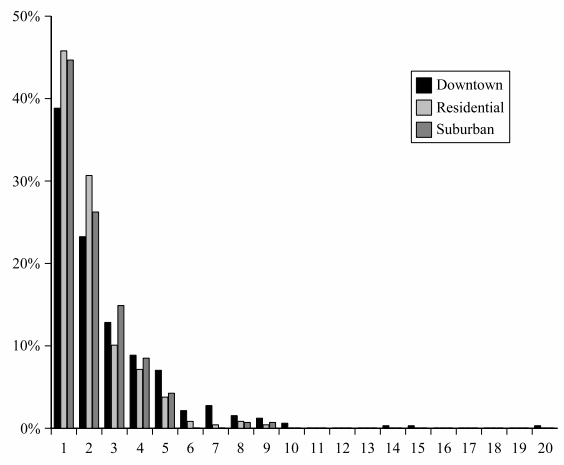


Figure 3: Field Study: Histogram, APs encountered per scan. Percentage of scans for each neighborhood that found a given number of APs.

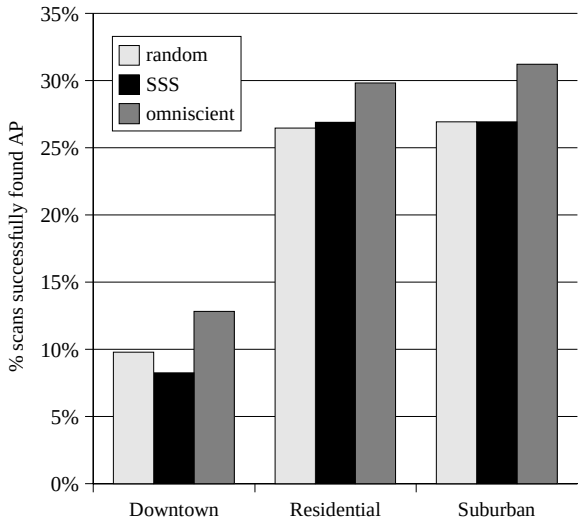


Figure 4: Field Study: Simulated percentage of scans that found a usable AP. "Random" algorithm chooses an unencrypted AP at random. "SSS" chooses the unencrypted AP with the strongest signal strength. "Omniscient" simulates an algorithm which uses the results of AP probes to choose the AP with the best bandwidth.

the expected nonce + 1 value, the port was marked "redirected". If the connect to the reference server failed, the port was marked "closed". To estimate bandwidth to the RS, the script connected to a special reserved port on the RS. The RS then transmitted random data at full speed over the TCP connection. The script received data for one second and then broke the connection. To avoid the effects of TCP slow start, we discarded the first 500 ms of data and calculated the bandwidth estimate from the remainder.

2.2 Access Point Statistics

We encountered a total of 1517 unique access points in all three neighborhoods. However, as Table 1(a) shows, few of these granted a DHCP address to our iPAQ handheld. It is interesting to observe that well over half of all APs had WEP or WPA encryption enabled. This suggests that a majority of users proactive enough about their security to manually enable a feature that typically defaults to off on most consumer-grade APs.

Figure 3 shows the histogram of APs found per scan in each neighborhood. Note that, for a substantial percentage of scans, multiple APs were available. This is encouraging since, when foraging for bandwidth, only one access point out of many need be usable at a given location for a user to be satisfied.

Table 1(b) gives statistics for the subset of access points that granted a DHCP address to our client. Around half of the open APs in the central business district block all TCP port traffic, and redirect port 80. This corresponds to the expected use of "splash-screen" logins for commercial hotspots. As we will see, such APs are a major source of error for the strongest-signal-strength AP selection policy, since what appears to be an open AP with a strong signal is in fact useless unless the user has an account with the service provider.

2.3 Missed Connectivity Opportunities

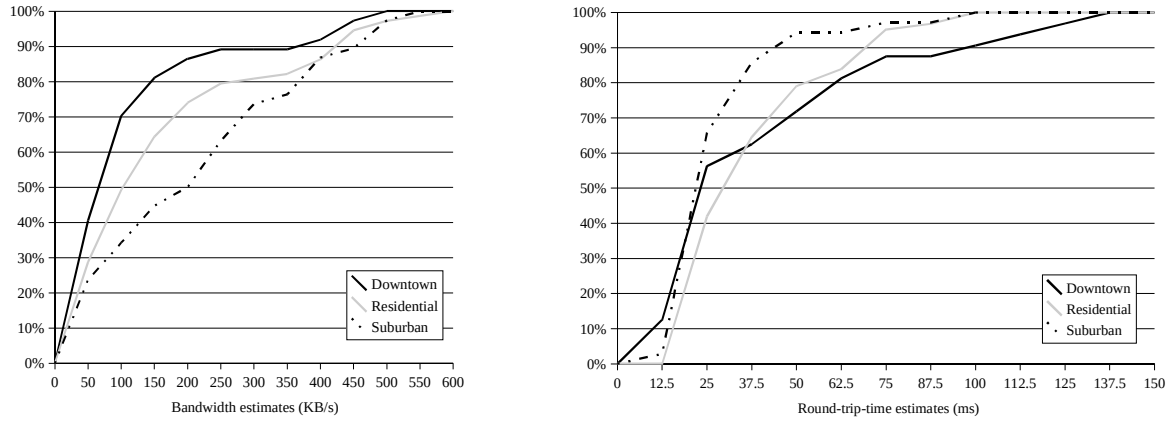
For each of the three neighborhoods studied, we took a 60-minute trace segment and applied a sliding window to generate several 30-minute "walks". Each walk represents a sequence of scans with a different set of seen APs.

For each walk, we first simulated a "random" algorithm, which simply chooses a random unencrypted AP. Next, we simulated the strongest-signal-strength selection algorithm. This generated a sequence consisting of the strongest signal strength APs from each scan in the walk. Lastly, we applied an omniscient algorithm, which used the results of our tests to choose the "best" AP from each scan set. For this experiment, the best AP was the one that granted a DHCP address and had at least one port open. If more than one AP qualified, we picked the one that had the best bandwidth estimate to the reference server.

For each generated AP sequence, we used as our metric the percentage of scans that would have found a usable AP. The difference between the performance of the omniscient algorithm and the SSS algorithm represents the time during which a client using SSS would have been disconnected even though there was a usable AP within range. We averaged the results of all the different 30 minute walks for each algorithm (random, SSS and omniscient), and graphed the results above.

As Figure 4 shows, in comparison to SSS, the omniscient algorithm found a usable AP 56%, 11%, and 16% more often in the downtown, residential and suburban neighborhoods, respectively. This represents significant missed connectivity opportunities for users. As we noted above, this is partly due to hotspots with "splash screen" logins. But since such hotspots were almost entirely confined to downtown, the connectivity gap in the residential neighborhoods cannot be accounted for solely by SSS choosing commercial hotspot APs. This suggests SSS is often passing on usable APs because of their signal strength, when the APs with stronger signals are in fact unusable.

Most strikingly, our simulations show that simply choosing an AP at random outperforms SSS for downtown, and is roughly the same in the other two neighborhoods. We inter-



(a) Bandwidth

(b) Round-trip-time

Figure 5: Field Study: RTT and bandwidth. Cumulative distribution functions. Note the variance in RTT and bandwidth per AP within each neighborhood.

		Downtown			Residential			Suburban		
port #	**service**	**open**	**redirected**	**closed**	**open**	**redirected**	**closed**	**open**	**redirected**	**closed**
135	DCOM	36	8	34	30	1	50	0	3	40
445	SMB	29	8	41	27	1	53	0	4	39
25	SMTP	28	9	41	31	0	18	36	0	7
21	FTP	29	16	33	63	1	17	37	0	6
22	SSH	37	8	33	69	2	10	37	0	6
23	telnet	39	10	29	73	1	7	37	0	6
79	finger	40	9	29	70	2	9	40	0	3
80	HTTP	37	38	3	75	1	5	39	1	3

Table 2: Field Study: Ports of interest. DCOM (Microsoft's RPC) and Samba (Windows file sharing) were the most-targeted services.

preted this result as yet another validation of our belief that an AP's signal strength is a poor predictor of its suitability for use.

Furthermore, across all three neighborhoods, only 10.8% of access points were usable, but 22.6% of scan sets had a usable AP. This further reinforced our belief that choosing the best access point out of all the ones that can be seen at any given point is critical.

2.4 All APs Are Not Created Equal

Lastly, the field study sought to examine how access points vary in the quality of service they provide. If different APs all provide basically the same quality connection, then when multiple usable APs were present at one spot an AP selection algorithm might as well just choose one at random.

However, our results showed access points are heterogeneous. Figure 5 shows the cumulative distribution functions for both the round-trip-time and bandwidth estimates, for all APs encountered during the field study. None of the CDFs converge rapidly to 100%, indicating a large variance in the results. This further bolstered our belief that, when multiple usable APs are present at one location, an AP selection algorithm should use the results of tests like those conducted for this field study to guide its choice.

We also found that access points vary widely with regard to what TCP port traffic they allow, block, or redirect. Table 2 illustrates the results of port scans for the eight most-blocked services. While most APs allowed the majority of port traffic, these were some notable exceptions. Ports 135 (DCOM) and 445 (Samba) are used by Microsoft Windows for remote procedure calls and file sharing. These are common points of entry for hackers, and are often blocked at the ISP for that reason. Port 25 handles the Simple Mail Transfer Protocol (SMTP). ISPs often block this port to prevent spammers from using the AP as a broadcast point. Finally, note that in downtown, HTTP traffic is often redirected (for splash-screen logins), but such hotspots are rarely seen in the residential neighborhoods.

These port results suggest it is useful for AP selection algorithms to know what port traffic a given AP allows. For example, suppose a user has configured her e-mail program to send e-mail by connecting to her own ISP's SMTP server over port 25. When she moves to a new location and opens her laptop, several usable networks are available. All things being equal, she would prefer that her computer uses an access point that allows her to connect directly to her ISP's SMTP server without blocking or redirecting port 25 traffic. Otherwise, she must close Thunderbird and switch to a webmail interface which may or may not be available from her mail server.

3. VIRGIL

The results of the field study motivated our belief that selecting access points based on signal strength results in a significant waste of potential network connectivity. Given that SSS performed no better than random selection in our field study simulations, we concluded that signal strength is an insufficient criterion to consistently predict if a given AP will be usable.

Armed with these lessons, we designed a new AP selection system, which we named Virgil. Virgil scans for available APs, then quickly and cheaply probes each for suitability of use.

Virgil's algorithm for selecting a new access point is as follows:

1. Scan for all available APs
2. Test each unencrypted AP in the scan set
   - Get AP properties (SSID, MAC address, signal strength, etc.)
   - Try to get DHCP address from AP
   - If successful, probe the AP
   - Store test results in a local database
3. Select the "best" AP, based on test results

In addition to open access points, the user may have authorization to use certain non-public APs. For example, she may encrypt her home wireless AP, and/or buy service from a hotspot provider such as T-Mobile. Virgil therefore allows users to manually enumerate "closed" APs that should be considered for use when seen. The user must obviously enter either the WEP/WPA encryption key for encrypted access points, or her username/password credentials for APs with "splash-screen" logins.

Since Virgil stores tests results in a local database, it improves performance by not rescanning often-seen APs. Our design consists of a user-level AP selection daemon running with root-level privileges. This process scans for new APs in the background, building this history database. Virgil chooses a new access point when (1) the device first boots, or returns from hibernation or suspend, and (2) the current AP is no longer usable. The selection daemon uses a simple heartbeat to a reference server to determine when the current AP is no longer usable.

3.1 Probing an AP

The goal of our AP selection daemon is to always choose the "best quality" access point out of all the APs available at a given physical location. "Quality" is highly subjective, but we consider the following to be important criteria:

- Bandwidth from the Internet to our client via this AP
- Port traffic that this AP blocks or redirects
- Round-trip-time from our client to remote servers

Since AP selection must be quick to be beneficial to users, Virgil performs all of these tests in parallel by spawning a thread to handle each port test and the RTT and bandwidth tests.

To aid in AP testing, we use a set of *reference servers* that is diverse in terms of both geography and network topology. Reference servers simply listen for TCP and UDP connections on a wide range of port numbers (e.g., 1–65535) and respond to port and bandwidth probe requests. At the start of each scan set, Virgil randomly chooses a reference server to use for that round of testing. This mitigates false negatives in the case where an AP is fine but the Internet route to a certain reference server is broken.

As in our field study, Virgil tests the status of a hand-compiled list of 45 common port numbers. Additionally, however, this base set is augmented at runtime by other TCP and UDP ports that are currently in use by the client, or have recently been used. For example, suppose the user is currently connected to her office through a virtual private network (VPN), on a port number not in our base set. When migrating to a new access point, she would obviously prefer an AP which permits traffic on that port number, so her VPN services remain available.

For UDP ports, Virgil simply sends the nonce (since there is no concept of "connect" for UDP). If it receives (nonce+1), the port is "open". If it receives something different, the port is "redirected". If it receives nothing before a timeout expires, the port is "closed".

Round-trip-time and downstream bandwidth from the reference server were calculated in the same fashion as for the field study above. We focus on estimating downstream bandwidth rather than upstream bandwidth because applications such as web traffic, streaming media, email, and newsgroup reading are overwhelmingly unidirectional. A recent study of wireless traffic on a campus WLAN [14], however, showed that upstream traffic comprised a significant portion of not just peer-to-peer but also web traffic. It is unclear if such workloads comprise as large a fraction of network traffic for the general population (as opposed to college students). If so, it would be useful to revise our design to estimate bandwidth in both directions.

3.2 Leveraging History

Each time Virgil scans an AP, it saves the AP's information in a local database. Each database record includes information shown in Figure 6.

Virgil keeps this database to improve performance, by not repeatedly rescanning access points that the user frequently encounters. Therefore, when Virgil examines the APs seen in a scan set, it only tests APs that do not already have a database record.

Naïvely, this would forever "blacklist" any APs which performed poorly the first time they were seen. The quality of service provided by an access point can change over time (as network conditions change, customers switch to different ISPs, or the access point load fluctuates).

We therefore force periodic re-scans of access points. Each time Virgil sees an access point that is already in the database, it doesn't re-scan it but updates its timestamp field, and increments the number of times it has been seen since it was last scanned. The user configures two thresholds—the maximum time that should pass between forced rescans, and the maximum number of times seen. Once either threshold has been exceeded, the next time the AP is seen in a scan set, Virgil forces a rescan and resets the "times seen" counter to zero.

To ensure freshness, Virgil periodically re-scans the AP currently being used by the user. Since her device is already associated with that AP, there is no interruption of service

Basic AP information

- ESSID, MAC address, and channel number
- Encryption (on or off)
- Signal strength, noise level, transmit power
- DHCP success (did AP grant IP address?)
- Timestamp last scanned
- Number of times seen since last scanned

For APs which granted an IP address via DHCP

- Round-trip-time estimate (ms)
- Bandwidth estimate (bytes/second)
- Port status for each scanned port (open, closed, or redirected)

Figure 6: AP Database Entries.

apart from the minimal network load imposed by the AP tests. By default, Virgil freshens the current AP's database record every 30 minutes, but this is a configurable value.

3.3 Choosing the "best" AP

Once Virgil is finished with a scan set, each AP in the set has a record in the database. Based on these "test results", Virgil chooses an AP, associates with it, and retreats into the background until needed to choose an AP again.

As the obvious first step, Virgil creates a candidate set consisting of only those APs which were open. By "open", we mean that both (1) the AP granted a DHCP address to the client, and (2) at least one port was found to be open. To this set we add APs the user has manually configured, such as pay hotspots to which she subscribes and/or encrypted APs for which she holds a key. This eliminates other pay hotspots (which block and/or redirect all traffic until users subscribe) and APs that selectively block traffic based on (for example) MAC addresses.

This may often leave more than one candidate. The user specifies how they would like such ties to be broken. If the user is primarily browsing the web or transferring large amounts of data, an obvious choice is to use bandwidth as a tie-breaker. On the other hand, if the user is dealing with latency-sensitive applications (for example, ssh), they may choose RTT. Finally, if a certain critical application needs an open port, the user may prefer APs which allow such traffic. Automatically negotiating the optimal tradeoff between these considerations is a focus of future work.

3.4 User Feedback

Most of the AP selection mechanism described thus far happens automatically without any intervention or attention from the user. Once Virgil settles on a new AP, it notifies the user by raising an alert (similar to the alert balloons used by Windows XP) in the corner of the screen.

Some users may want more information about the ramifications of this choice. Such users click on the notice, loading a status screen. This status screen summarizes the test results of the AP that was just chosen. Most importantly, this summary indicates which applications currently in use may stop working as a result of using this new AP. Recall our earlier example of a user who uses SMTP mail. If she moved to an access point that blocks port 25, this summary screen would inform her that her email program will not be able to send email at this location. Virgil infers that Thunderbird is an email program from the well-known port number of the connections it has established in the past. The user is *not* merely told that port 25 is blocked, since that information is meaningless to the vast majority of users.

If the user decides that using her mail reader is critically important, she indicates this, causing Virgil to review the most recent scan set and see if another open AP is available that doesn't block the port in question. If so, it associates to this new access point and informs the user. If no other AP is available, the user is informed so she can decide to walk to another location, for instance.

4. PROTOTYPE

We implemented our prototype on Linux, and have cross-compiled it for both x86 laptops and ARM-based handhelds (specifically, the Compaq iPAQ). The prototype implements all aspects of the design outlined above, with three exceptions. First, we do not constantly scan for new access points in the background. Ideally, Virgil would leverage systems such as MultiNet [8] or SyncScan [24] to continuously scan for new access points, without having to disassociate from its current AP, but we have not implemented this. AP scanning happens when the current AP becomes unusable. Second, we have not implemented the user feedback component described in Section 3.4. Third, rather than using multiple reference servers, we use the same, single reference server as for our field study.

When multiple usable APs were available, our prototype used bandwidth to the reference server as the tiebreaker.

4.1 Active Scanning

When scanning for new access points, the primary challenge is the tension between delay and false negatives. If we cut off testing too early, Virgil may not find all usable access points. On the other hand, if the delay Virgil imposes is burdensome to users, they will abandon our system.

As a result, the prototype has some built-in timeouts. Specifically, DHCP address acquisition times out and fails after 5 seconds. Similarly, port scans fail and return "closed" if the TCP connect takes longer than 5 seconds, or if Virgil has not received a response to the nonce in 5 seconds. We experimentally chose the value of 5 seconds by successively lowering the timeout value until we started to notice false negatives. That is, DHCP attempts and port scans were failing simply because there was not enough time for the process to complete. This also caps the average time to scan an unusable AP at around 5 seconds.

The prototype makes extensive use of the Linux wireless extensions toolkit. It uses the output of `iwlist scan` to generate a scan set at the start of AP discovery, and uses `iwconfig` to record each AP's MAC address, channel number, et cetera, in the local database.

All of the tests on a given AP (port probes, RTT, and bandwidth estimates) occur in parallel for maximum efficiency. We use *pthreads* to spawn a thread for each of the

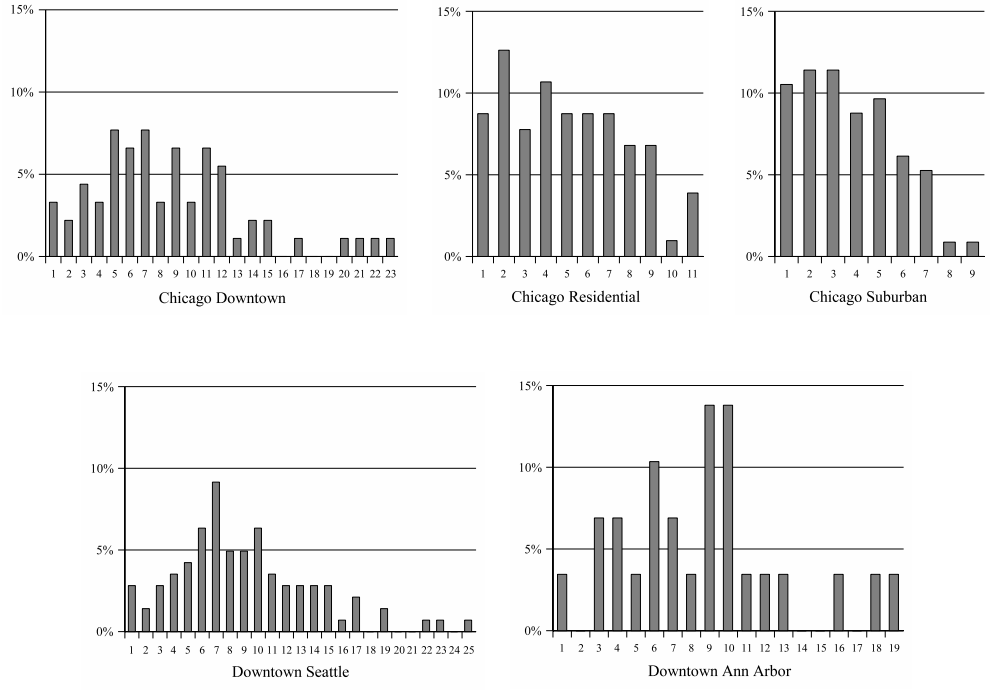


Figure 7: Evaluation: Histogram, APs per scan. Percentage of scans for each neighborhood that found a given number of APs.

	Downtown Chicago	Residential Chicago	Suburban Chicago	Seattle	Ann Arbor
APs seen	559	438	273	870	225
Scan sets	91	103	114	142	29
APs per scan set	6.1	4.3	2.4	6.1	7.8
APs granted IP address	23 (4.1%)	61 (13.9%)	41 (15.0%)	54 (6.2%)	25 (11.1%)
APs using encryption	292 (52.2%)	261 (59.6%)	128 (46.9%)	475 (54.6%)	151 (67.1)

Table 3: Evaluation: AP statistics.

tests, and the main thread performs a pthread_join on each thread to wait until all tests have finished before proceeding.

4.2 Tracking open connections

Our prototype uses the Linux utility netstat to track open TCP and UDP connections. A thread wakes every 60 seconds, runs netstat, and updates an in-memory database. Since the report generated by netstat buffers all used ports for the last 60 seconds, this lets Virgil capture even the briefest port activity.

Each record in this database corresponds to one port number and type (UDP or TCP). For TCP connections, it notes if the connection was inbound (listening), or outbound. Finally, a timestamp notes the last time that the port was seen to be in use. By sorting this database in order by timestamp, Virgil easily determines which ports have been in most recent use, and are therefore most important to ensure remain open when migrating to a new AP.

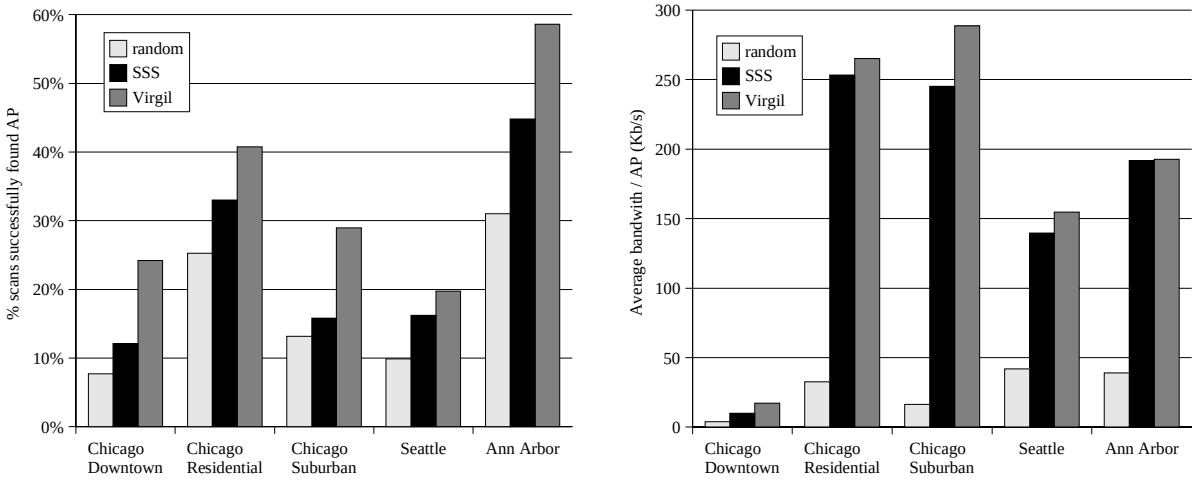
5. EVALUATION

In evaluating our prototype implementation, we sought answers to the following questions:

- How much more often does Virgil let users be connected, than simply selecting on signal strength?
- How much better are the connections that Virgil finds?
- How beneficial is tracking AP history?
- Is Virgil's overhead reasonable such that it is useful to users?

We used a similar methodology to that of our earlier field study. Along with the three neighborhoods in Chicago previously studied, we also tested Virgil in Seattle, Washington (city population: 573,000, metropolitan area: 3.8 million) and Ann Arbor, Michigan (population: 114,000) [5]. These two cities gave us data points for medium- and small-sized cities, respectively.

Figure 7 charts the histogram of APs encountered per scan, for each of the five neighborhoods. One notices that we found more APs per scan on average, when evaluating Virgil than we did during our field study. This may partly result from the fact that, while we re-walked the same three Chicago neighborhoods, we did not retrace our steps exactly. We also used different hardware (a different iPAQ) for the evaluation runs than for the field study, due to equipment



(a) Percentage of Scans Successful

(b) Average Bandwidth of Open APs (KB/s)

Figure 8: Evaluation: Improvement of Virgil over SSS. Virgil finds usable APs more frequently than selecting based on signal strength—from 22% to 100% more often. The quality of the APs Virgil finds is also better (based on bandwidth to the network).

failure. As Table 3 shows, we encountered 2365 different APs. As stated above, our evaluation log data and the logs from our field study are freely available via the CRAWDAD archive.

5.1 Connection Time and Quality

We returned to Chicago with the completed Virgil prototype on the same iPAQ handheld. Although we re-walked the same neighborhoods in the same street-by-street grid fashion as for our field study, we by no means made an effort to re-trace our exact steps. In Seattle and Ann Arbor, we walked a similarly-sized portion ($\sim$1.3 km^2) of each city's downtown area.

As we walked each neighborhood, Virgil ran in the background, handling AP selection for the Linux operating system on the iPAQ. We configured Virgil to log information on all APs seen on each scan, the test results of all APs which were probed, and the final choice of AP for each scan set.

This log data let us reconstruct, after the fact, the sequence of access points which the strongest-signal-strength algorithm would have chosen. Based on the test results of probed APs, we calculated two metrics. First, we calculated the percentage of scans which would have found a usable AP, given the selection algorithm (random, SSS or Virgil). Second, we calculated the estimated average bandwidth, in KB/s, of the APs that each algorithm selected. The results are shown in Figure 8.

For all five neighborhoods, Virgil found a usable AP significantly more often than did SSS or random selection. The improvement over SSS ranged from 22% (in Seattle) to 100% (in downtown Chicago). While Virgil's connectivity percentages (ranging from 19.7% to 58.6%) are still insufficient for applications requiring seamless connectivity, it represents a significant improvement over the current state of the art.

Figure 8(b) illustrates the average bandwidth estimates of the open APs chosen by each algorithm. Virgil still outperforms SSS, but by a much smaller margin than for connectivity. The reason for this is that SSS may only find a handful of APs in a neighborhood, but the ones it finds may happen to have high bandwidth to the reference server. On the other hand, Virgil finds more APs and therefore must average across a wide range of bandwidth connections.

5.2 History

Next, we sought to quantify the benefit of storing AP test results in our local database. A rough estimate of the space overhead imposed by this database can be derived from our test results above. Roughly two hours of constant scanning and walking in each neighborhood generated databases on the order of 20-30 KB in size. These are unoptimized, text-file databases. Clearly, though, if our results showed storing this history leads to little performance benefit, then they could be discarded.

We walked a 1.5 km loop from downtown Ann Arbor to campus and back five times, logging Virgil's output as before. The walk was meant to simulate the daily mobility of a hypothetical student who lives downtown, attends class on central campus, and walks roughly the same route between the two each day.

Figure 9 shows the percentage of scans, for each algorithm, that a usable AP was selected on each "lap" around Ann Arbor. As expected, Virgil outperforms SSS on the first lap, finding a usable AP twice as often. On the subsequent laps, however, Virgil maintains its steady success rate while the random and SSS algorithms fluctuate wildly. This is partly a consequence of unreliable AP scanning algorithms. Running a utility such as Linux's iwlist scan several times in succession, from the same location, can return varied sets of access points. This happens because APs broadcast their beacon signals at unpredictable times, and do not always respond to beacon requests in a timely fashion [24].

On subsequent laps, all three algorithms (Virgil, SSS and random) may find new access points. In the case of Virgil, if

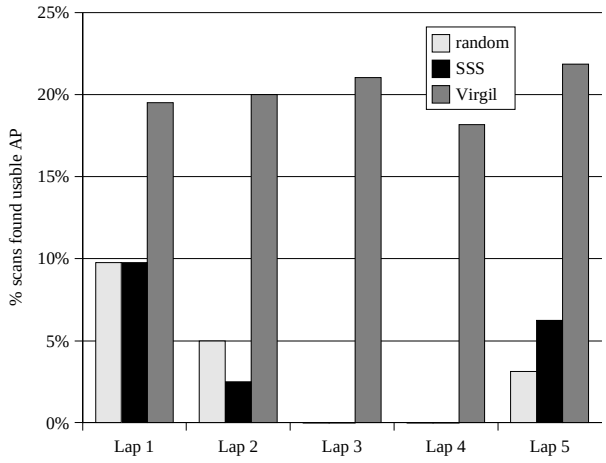


Figure 9: History: Percentage of successful scans. Five walks along the same 1.5 km length path from downtown, to campus, and back. The AP history database helps Virgil consistently find better access points more often.

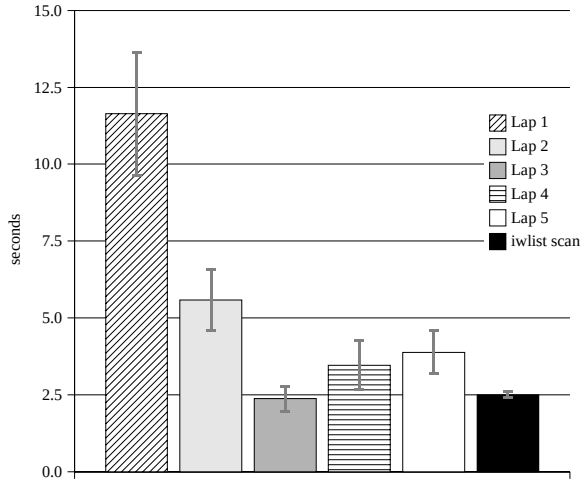


Figure 10: History: Mean time to complete one AP selection cycle. Time to scan for available APs, and test all new APs. The rightmost bar ("iwlist scan") is the mean time to just scan for available APs.

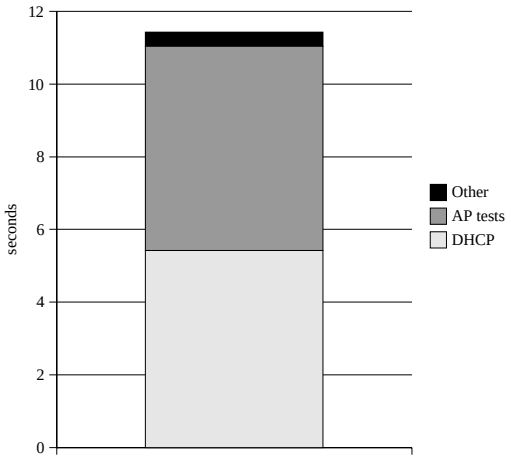


Figure 11: Overhead: Time to scan one AP, by phase. The time to run AP tests and to associate with the AP are comparable.

it sees an AP that it happens to "remember" from a previous lap, Virgil will continue to use it unless the new AP proves to be of even higher quality. On the other hand, SSS does not have the benefit of such history information and has to make a spot decision based on instantaneous measurements. Thus, SSS may pick "correctly" one lap and incorrectly the next, but once Virgil finds a good AP it will stick with it.

One of the biggest advantages of maintaining history information for Virgil is the ability to reduce the average time to complete a scan cycle. We measured the difference between laps in the average time to complete an entire scan cycle. This includes the time to discover all available access points, test each new, unencrypted AP, and finally acquire an IP address from the chosen AP. Figure 10 shows the average time to complete one scan cycle, for each lap. Additionally, the rightmost bar shows the mean time to only perform the scan for new APs. After the first two laps, mean time per scan cycle is nearly indistinguishable from the mean time to simply scan for APs. This is due to the effects of history. Once Virgil has "mapped-out" all the APs on a given path, it need not re-probe them. It simply scans for all available APs and chooses the best one from the list, based on its past history. This confirms our belief that history will mitigate the overhead Virgil incurs in the AP probing process. Users who run Virgil every day will soon map out the routes they most commonly traverse, and per-scan overhead would be no more than current schemes such as SSS.

5.3 Overhead

5.3.1 Client Overhead

We collected a diverse set of data on the time overhead inherent to our prototype implementation.

Across all five neighborhoods, for all 204 APs that granted an IP address, we calculated the time spent in each phase of the AP probing process. As Figure 11 shows, probing an AP took just over 11 seconds on average. This time was split fairly evenly between first acquiring an IP address via DHCP, and then running the port, RTT, and bandwidth tests. Since our timeout was 5 seconds for both of those operations, it is unsurprising that few AP tests or DHCP attempts exceeded that value.

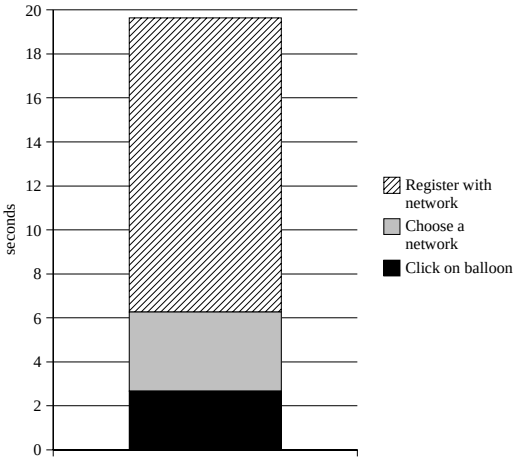
As discussed above, we chose to "cast a wider net" by setting these timeouts to the relatively high value of 5 seconds. Thus, we discovered more APs than we would have with a lower timeout, at the cost of increased time per scan. We argue that this overhead, while clearly not negligible, is acceptable. As we showed above, Virgil will quickly map the neighborhoods users spend most of their time in, erasing such overhead for subsequent visits.

Virgil would ideally be integrated with a system such as MultiNet [8], which allows one device to simultaneously associate with multiple access points. This hides most of this per-AP scan overhead, since Virgil could associate to the first candidate AP it finds, and keep scanning other APs in the background while the user is connected.

(a) Windows XP's manual AP discovery notice.

	Click on balloon	Choose an AP	DHCP acquire	Total
mean	2.7	3.6	13.4	19.7
median	2.5	3.8	13.1	19.4
σ	1.1	0.8	1.8	1.4

(b) Statistics for manual AP selection in Windows XP.



(c) Breakdown of manual AP selection in Windows XP.

Figure 12: Overhead: Cost of manual AP selection. Many operating systems require users manually intervene to choose an AP.

Most importantly, we argue that using Virgil to automatically select an access point, even with the overhead shown above, is still faster than the current practice of forcing users to choose manually. To reinforce this point, we measured the time required for a user to select an AP using Windows XP's integrated selection tool.

When Windows XP first boots or wakes from hibernation, it scans for all available APs. If it finds an AP which the user has previously selected as a "preferred" AP, it automatically associates to it. Otherwise, it raises the alert balloon shown in Figure 12(a). The user must see the alert, move the mouse to the corner of the screen and click on it. This raises a screen which lists all available APs. The only information users are given is SSID, encrypted status and signal strength (0-5 bars). Based on this information, the user chooses an AP from the list. XP then attempts to associate with the AP and receive an IP address via DHCP.

A user performed this task 10 times, and we recorded the time required for three operations: (1) time between the balloon's appearance and when the user clicked on it, (2) time between the AP selection window's appearance and the user clicking "Connect" to choose an AP, and, (3) time Windows XP required to associate with the AP, acquire an IP address, update whatever internal state it keeps, and update the AP selection window to indicate success.

This is clearly not an exhaustive study of user behavior. However, it does provide us with some evidence of the time required to associate a Windows laptop with an AP using existing techniques. Figures 12(b) and 12(c) show it takes a user roughly 20 seconds to manually select an AP in Windows XP. We argue this is a hard bottom limit, since the user knew exactly which SSID he was looking for *a priori*, and wasted no time deciding on the selection screen, as a real user would in an unfamiliar environment.

It is curious that the time to associate with the AP dominates, since in our tests the AP was nearby and signal strength remained excellent throughout. Regardless, the time spent in active user work is significant. Furthermore, after choosing an AP, the user would need to try to load a web page, or otherwise check that the connection is usable before proceeding with her work. All of this incurs significant overhead and is burdensome to users.

5.3.2 Reference Server Overhead

Finally, we sought to quantify the load our suite of AP tests would impose on the reference servers.

First, we wrote a script that ran a full set of AP tests (bandwidth, round-trip-time, and 35 TCP port status tests). This script ran the test set 25 times in a row. This attempted to simulate a worst-case scenario, where a Virgil client found a large number of new, open APs at one location, and tested them all in rapid sequence.

Since multiple clients may be connecting to one reference server at the same time, we ran 20 trials, each time running the script described above on one additional machine.

The reference server was 2.40 GHz Intel CPU with a 512 KB L1 cache and 768 MB of system RAM. It was connected to a 10-Mbps wired Ethernet LAN. All 20 machines used to launch test clients have a 3.40 GHz Intel CPU with a 2048 KB L1 cache and 2 GB of system RAM.

To launch multiple tests as simultaneously as possible, we first pre-positioned our test script on each. To start k instances of the test script, then, from a separate machine we forked k copies of a Python script that used `ssh` to remotely launch the test script on a given machine.

Because not all copies were in fact started at the same time, we measured the peak CPU utilization during each run. This is presumably the point at which all k copies of the script are finally hammering the reference server.

Figure 13 shows that CPU utilization rises more or less linearly with the number of clients actively using the reference server. At the maximum load, with 20 different machines each running the AP test suite 25 times in quick succession, the CPU of the reference server was only 10.5% utilized. During none of the tests was memory a consideration.

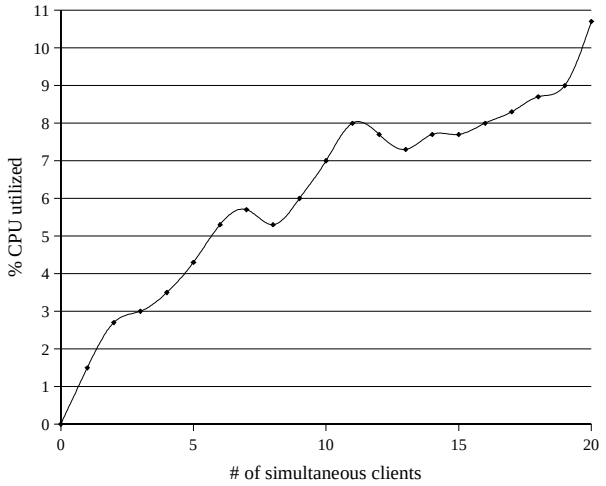


Figure 13: Reference server load testing results.

These results suggest that a reference server with modest hardware resources can easily support dozens of client connections per second. It is unclear how many APs a typical user would need to test per day. Understanding this demand for reference server resources is crucial to ongoing work that seeks to determine how many reference servers would be needed to deploy Virgil on a large scale.

6. RELATED WORK

Several previous "wardriving" studies collected 802.11 AP beacon information [1, 9], which contributed to the many Internet databases of wardriving maps [15, 16, 30]. Users must manually scour these maps to find access points, however, while our system is fully automatic. None of these systems associate with APs to run performance tests as we do. As our results showed, the information gleaned from these tests allows Virgil to outperform selection schemes driven solely by the signal-strength data in such datasets. Furthermore, these static maps become unreliable and outdated over time [6], while Virgil continuously rediscovers and probes the user's environment.

The increasing popularity of real-time, stateful applications, in particular, voice-over-IP, has sparked a great deal of research into making 802.11 handoffs as seamless as mobile phone handoffs. MultiNet [8] virtualizes a device's wireless interface, fooling applications into believing the device is connected simultaneously to different APs on different channels. Integrating MultiNet with Virgil would let us continuously search for and test new access points, without having to disassociate with our current access point.

SyncScan [24] modifies access points as well as clients, forcing APs to synchronize their beacon frame broadcast schedules. Since clients know in advance what channels will be broadcasting at which times, they can quickly collect all beacons and return to their original channel before any user-perceivable service disruption is noticeable. Shin [29] similarly optimizes the scan process, since AP discovery has been shown to dominate the 802.11 handoff process. Neither technique associates with the scanned APs. Therefore, such techniques only speed up the selection process, but neither makes the choice any more accurate than existing strongest-signal-strength algorithms.

Lee and Miller [19] propose adding information to the access point beacon signal, to help guide clients' AP selection. While their focus was on facilitating roaming between commercial wireless access networks, the concept could be generalized. One could envision access points broadcasting their current load, current estimated latency to reference servers in the Internet, etc. We argue it is preferable for clients to discover this information for themselves. As we showed, testing one AP takes at most a matter of seconds, a reasonable overhead. Furthermore, when clients are roaming in public, they have no reason to trust the stranger who administers an access point. In fact, it is likely in the AP administrator's self-interest to falsely advertise his AP as low-quality, to prevent anonymous traffic from overloading it.

Judd and Steenkiste [17] recognized that basing AP selection policy solely on signal strength results in uneven loading of multiple access points. They suggested AP load as a beneficial metric, since their work was more focuses on balancing load between access points than directly focusing on client performance as the primary goal.

It has become accepted that the push toward ubiquitous computing makes automatic service discovery in new environments more important than ever [27]. Existing work, however, has focused on application-level services [10, 12], but is silent on how the client chooses an appropriate network connection in the first place. Our work seeks to fill this gap.

Several systems seek to allow clients of one wireless service provider to access foreign wireless hotspots when roaming [4, 11, 20, 26]. Our work is complementary, since users must find and associate to an access point before negotiating such roaming agreements. The service discovery Virgil provides is similarly critical for grassroots wireless collective initiatives [2, 23, 28].

7. CONCLUSION

802.11 access point density has exploded in urban areas, to the point where users commonly have multiple APs to choose from on each scan. Since these access points are managed by a variety of individuals, businesses, and governments, a small percentage are open and usable. The quality of Internet connection APs provide often varies widely due to choice of service provider, AP load, and wireless network conditions.

Current selection algorithms focus on AP signal strength as an important metric. We conducted an extensive field study of three neighborhoods in Chicago, which showed that choosing an AP based on signal strength misses significant opportunities for Internet connectivity.

We presented the design and implementation of Virgil, an automatic AP discovery and selection system. Virgil quickly associates to each AP found during a scan, and runs a battery of tests designed to discover the AP's suitability for use by estimating the bandwidth and round-trip-time to a set of reference servers. Virgil also probes for blocked or redirected ports, to guide selection in favor of preserving application services currently in use.

We evaluated Virgil in five different neighborhoods across three different cities. Our results show Virgil finds a usable connection from 22% to 100% more often than simply selecting based on signal strength alone. By caching AP test results, Virgil improves both performance and accuracy for

neighborhoods the user commonly travels. We showed our overhead to be acceptable and less burdensome than current selection techniques which require user intervention.

8. REFERENCES

[1] Aditya Akella, Glenn Judd, Srinivasan Seshan, and Peter Steenkiste. Self-Management in Chaotic Wireless Deployments. In *Proceedings of the 11th International Conference on Mobile Computing and Networking (MobiCom)*, pages 185–199, August 2005.

[2] Bay Area Wireless Users Group. http://bawug.org/.

[3] BBC News. 'Evil Twin' Fear for Wireless Net, 20 January 2005.

[4] Mauro Brunato and Danilo Severina. WilmaGate: A New Open Access Gateway for Hotspot Management. In *Proceedings of the 3rd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)*, September 2005.

[5] United States Census Bureau. 2000 Census of Population and Housing, Summary Population and Housing Characteristics, Washington, DC, 2002.

[6] Simon Byers and Dave Kormann. 802.11b access point mapping. *Communications of the ACM*, 46(5):41–46, May 2003.

[7] Srdjan Capkun, Jean-Pierre Hubaux, and Levente Buttyan. Mobility Helps Security in Ad-hoc Networks. In *Proceedings of the Fourth ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc '03)*, pages 46–56, Annapolis, Maryland, USA, 2003.

[8] R. Chandra, V. Bahl, and P. Bahl. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *Proceedings of INFOCOM*, 2004.

[9] Y. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm. Accuracy Characterization for Metropolitan-scale Wi-Fi Localization. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications and Services (MobiSys)*, June 2005.

[10] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Proceedings of ACM/IEEE MobiCom '99*, pages 24–35, Seattle, Washington, August 1999.

[11] Elias C. Efstathiou and George C. Polyzos. A Peer-to-Peer Approach to Wireless LAN Roaming. In *Proceedings of the First ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, pages 10–18, San Diego, CA, 2003.

[12] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall, and Stephen Pink. Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments. *Wireless Networks*, 10(6):631–641, November 2004.

[13] Familiar Linux. http://www.handhelds.org/.

[14] Tristan Henderson, David Kotz, and Ilya Abyzov. The changing usage of a mature campus-wide wireless network. In *Proceedings of the Tenth Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 187–201. ACM Press, September 2004.

[15] Intel Research Seattle. Place Lab: A Privacy-Observant Location System. http://placelab.org/.

[16] Wi-Fi Hotspot Locator. http://jiwire.com/.

[17] Glenn Judd and Peter Steenkiste. Fixing 802.11 Access Point Selection. *ACM SIGCOMM Computer Communications Review*, 32(3):31, July 2002.

[18] Orin S. Kerr. Cybercrime's Scope: Interpreting "Access" and "Authorization" in Computer Misuse Statutes. *New York University Law Review*, 78(5):1596–1668, November 2003.

[19] Yui-Wah Lee and Scott Miller. Network Selection and Discovery of Service Information in Public WLAN Hotspots. In *Proceedings of the 2nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, pages 81–92, Philadelphia, PA, 2004.

[20] Yasuhiko Matsunaga, Ana Sanz Merino, Takashi Suzuki, and Randy Katz. Secure Authentication System for Public WLAN Roaming. In *Proceedings of the First ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, pages 113–121, San Diego, CA, 2003.

[21] Anthony J. Nicholson, Junghee Han, David Watson, and Brian D. Noble. Exploiting Mobility for Key Establishment. In *Proceedings of the 7th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, Blaine, Washington, USA, April 2006.

[22] Anthony J. Nicholson, Ian E. Smith, Jeff Hughes, and Brian D. Noble. LoKey: Leveraging the SMS Network in End-to-end, Decentralized Trust Establishment. In *Proceedings of the Fourth International Conference on Pervasive Computing*, Dublin, Ireland, May 2006.

[23] NYCWireless. http://nycwireless.net/.

[24] I. Ramani and S. Savage. SyncScan: Practical Fast Handoff for 802.11 Infrastructure Networks. In *Proceedings of INFOCOM*, March 2005.

[25] Naouel B. Salem, Jean-Pierre Hubaux, and Markus Jakobsson. Reputation-based Wi-Fi Deployment Protocols and Security Analysis. In *Proceedings of the Second ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, October 2004.

[26] Naouel Ben Salem, Jean-Pierre Hubaux, and Markus Jakobsson. Reputation-based Wi-Fi Deployment Protocols and Security Analysis. In *Proceedings of the Second ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, pages 29–40, Philadelphia, PA, 2004.

[27] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.

[28] SeattleWireless. http://seattlewireless.net/.

[29] Minho Shin, Arunesh Mishra, and William A. Arbaugh. Improving the Latency of 802.11 Hand-offs using Neighbor Graphs. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 70–83, June 2004.

[30] WIGLE: Wireless Geographic Logging Engine. http://wigle.net/.