WiFiProfiler: Cooperative Diagnosis in Wireless LANs

Ranveer Chandra Venkata N. Padmanabhan Ming Zhang
Microsoft Research

ABSTRACT

While 802.11-based wireless hotspots are proliferating, users often have little recourse when the network does not work or performs poorly *for them*. They are left trying to manually debug the problem, which can be a frustrating and disruptive process. The users' troubles are compounded by the absence of network administrators or an IT department to turn to in many 802.11 hotspot settings (e.g., cafes, airports, conferences).

We present *WiFiProfiler*, a system in which wireless hosts cooperate to diagnose and possibly resolve network problems in an automated manner, without requiring any infrastructural support. The key observation is that even if a host's wireless link to an access point is not working, the host is often within the range of other wireless nodes and is in a position to communicate with them (a little) peer-to-peer. We leverage this ability to create a shared information plane, which enables wireless hosts to exchange a range of information about their network settings and the health of their network connectivity. By aggregating and correlating such information across multiple wireless hosts, we infer the likely cause of the problem. Our implementation on Windows XP shows that WiFiProfiler is effective in diagnosing a range of problems and imposes a low overhead on the participating hosts.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Misc.

General Terms

Management, Reliability

Keywords

Wireless networks, 802.11, fault diagnosis, peer-to-peer

1. INTRODUCTION

There has been a remarkable growth in the deployment of 802.11-based wireless networks in offices, homes, airports, cafes, and even across entire cities [26]. Despite this success, it is not uncommon for users to experience connectivity or performance problems. When they do experience problems, users often go through a frustrating and disruptive process of manual debugging, say by tapping on the shoulders of nearby users to compare notes or arbitrarily resetting their wireless NIC or rebooting their computer. The users' troubles are compounded by the absence of network administrators or an IT department to turn to in many 802.11 hotspot settings (e.g., cafes, airports, conferences).

To address this problem, we present *WiFiProfiler*, a system in which wireless hosts cooperate to help diagnose and possibly resolve network problems in an automated manner. The setting we consider is an infrastructure-based wireless LAN (WLAN) with access points (AP) providing connectivity to the wired network. We do not assume or depend on any special capabilities in this infrastructure, which makes our work applicable in a wide range of WLAN settings.

Our key observation is that even if a host is disconnected (i.e., its wireless link to an AP is not working), the host is often within the range of other wireless nodes and is in a position to communicate with them (a little) peer-to-peer. We leverage this ability to communicate to create a shared information plane, on which the cooperating wireless hosts exchange a range of information about their network settings and the health of their network connectivity. Examples of such information include the networks and APs that hosts are or are not able to connect to, whether they have been able to obtain an IP address and if so how recently, and whether they have been experiencing poor performance.

By aggregating and correlating such information across multiple wireless hosts, we infer the likely cause of the problem (e.g., incorrect WEP key or port blocking at a firewall) and in some cases help users resolve the problem themselves (e.g., by suggesting an alternative network to connect to or a different HTTP proxy setting). Even when the problem is not resolved, we believe that telling the user the nature of the problem is valuable, as it can significantly reduce user frustration. For instance, a user who might have otherwise tried fiddling with his/her computer in the hope of fixing the problem can sit back and do other work if told that there is a more widespread network problem that is not specific to their computer.

The architecture of WiFiProfiler includes three components. The *sensing* component comprises software "sensors" running on each end host to passively monitor the health of the host's connectivity and its configuration across layers (WiFi, IP, TCP, application). The *communication* component enables communication among end hosts, whether they are connected or disconnected. It gathers relevant information requested by the *diagnosis* component, which also deduces the likely cause of the problem based on the information gathered.

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

We have prototyped WiFiProfiler on the Windows XP platform. A key goal for our design and implementation has been to depend only on "standard" functionality, both on end hosts (e.g., in terms of wireless NICs and drivers) and the infrastructure (e.g., no special monitoring nodes). While other design choices are clearly possible, we made our choice with a view to easing deployment and facilitating user adoption.

Our evaluation in an experimental testbed demonstrates the effectiveness of WiFiProfiler in diagnosing a range of problems and the low overhead it imposes on the participating hosts. The latter is critical since hosts may be reluctant to expend significant resources to help their peers. WiFiProfiler demands much less of the peers than, for instance, past proposals on using cooperative multi-hop routing to extend the reach of wireless networks.

Our work is inspired by and builds on two bodies of work. First, there have been proposals for pooling together information across end hosts to aid diagnosis, whether in a networking context (e.g., [21, 23]) or otherwise (e.g., [24, 25]). In comparison, WiFiProfiler addresses a number of additional challenges that are unique to the wireless context, e.g., enabling *disconnected* clients to gather information from peers with minimal overhead. Second, there has been some work on management and diagnosis in wireless LANs (e.g., [11, 12]). However, this prior work has been geared to helping network administrators (typically in enterprises) manage their networks (e.g., detect rogue APs), by leveraging a monitoring infrastructure. In contrast, WiFiProfiler focuses on problems that impact end users in hotspot settings, without depending on any special infrastructure. We believe that a key contribution of our work is in recognizing and demonstrating the potential for cooperative diagnosis, which may be easier to realize in practice than alternatives that require infrastructural support.

The rest of this paper is organized as follows. In Section 2, we discuss the problem context, including the nature of faults that wireless users are likely to experience. We present the design and implementation of WiFiProfiler in Section 3 and an evaluation in Section 4. Our design focuses on the basic functionality of cooperative diagnosis. We defer the issues of security and incentives to the discussion in Section 5. We present related work in Section 6 and conclude in Section 7.

2. PROBLEM CONTEXT

We set the context for our work by briefly discussing the architecture of wireless LANs and then considering the nature of failures that might happen, including some that the authors have themselves experienced in practice.

2.1 Wireless LAN Architecture

802.11-based WLANs comprise APs that connect the wireless cloud to the wired network. The APs typically have a radio interface and a wired interface (e.g., Ethernet). Wireless clients typically communicate with an AP via a single wireless hop, although it is possible to go multiple hops with link-layer repeaters. The AP provides bridging or routing functionality to connect the wireless clients to the wired network. In our discussion here, we use the term wide-area network (WAN) to refer to the wired network, including the access link to the wireless subnet.

Although this basic picture is simple, several other network mechanisms and components may be involved in providing connectivity, which we discuss in turn.

2.1.1 Wireless Security

The wireless network could employ a range of mechanisms to restrict access, authenticate hosts before permitting them to establish a link-layer association, and to encrypt the wireless transmissions. *MAC filtering* is a simple mechanism, which allows APs to reject packets arriving on the wireless interface unless their source MAC address is in an allowed list. While MAC filtering can be defeated using MAC address spoofing, this mechanism is still widely used because of its simplicity.

Wired Equivalent Privacy (WEP) is a mechanism for key-based authentication and encryption. The AP and wireless clients share a key, which is typically configured manually. The shared key could be used for authentication and/or encryption. Even if authentication is "open", the use of WEP for encryption would prevent a client without the correct key from communicating through the AP.

An alternative to WEP is *WiFi Protected Access* (WPA or WPA2, which is a form of the IEEE 802.11i standard), where a "master" key is set up either automatically using 802.1X (WPA enterprise mode) or manually by having the user enter a passphrase (WPA personal or pre-shared key (PSK) mode). The PSK mode is designed for home and small office networks that cannot afford the cost and complexity of an 802.1X authentication server.

2.1.2 DHCP

Once the client has successfully authenticated itself, it must typically obtain an IP address using DHCP, since the dynamic nature of wireless LANs makes static IP address assignment difficult. In a single AP environment (e.g., a typical home network), the AP itself could serve as the DHCP server. Alternatively, the DHCP server could reside on the wired network, which is typical when there are multiple APs on the same subnet.

DHCP is a simple broadcast-based protocol, which allows a client to discover one or more DHCP servers, typically on the local subnet, and request and receive an IP address lease. The DHCP server typically also returns other configuration information such as the IP addresses of the gateway and the local DNS servers.

2.1.3 Firewall

The network could include a firewall for added security. The firewall could be configured to filter inbound and/or outbound traffic based on various criteria. Filtering based on TCP/UDP port numbers is commonly employed since the port number is often, even if not necessarily, indicative of the application. However, filtering based on other criteria is also possible.

2.1.4 Application-level Proxies

The network could include application-level proxies, whether for performance reasons (e.g., caching) or for security reasons (e.g., URL blocking). In particular, HTTP caching proxies are often employed for web accesses. Typically, the web browser is configured with proxy information, either manually or automatically using Web Proxy Auto-Discovery (WPAD). Less common are transparent proxies [16], which are not visible to the browser.

2.2 Nature and Causes of Network Problems

Network problems can arise not only due to the wireless medium but also because of the failure, malfunctioning, or misconfiguration of the network mechanisms and components noted above. We briefly consider the symptoms that

a client may observe and the underlying cause(s), as shown in Figure 1.

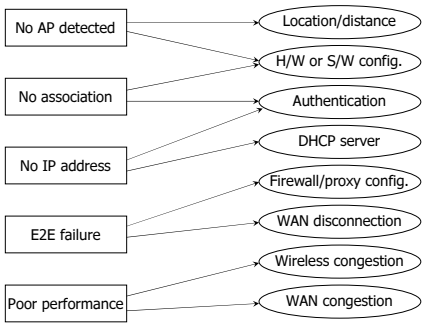

Figure 1: Problem symptoms observed at a client and some of the associated causes.

2.2.1 Inability to detect an AP

The client may not hear beacons from any AP for several reasons. It may be out of their range or suffering from channel noise, both of which may be a function of the client's current location. There may also be software or hardware incompatibilities that prevent the client from detecting the AP; we discuss these in more detail below.

2.2.2 Inability to associate with an AP

Association refers to the establishment of a link-layer connection between the client and the AP. There are several reasons why association may fail.

First, the AP could be malfunctioning (e.g., not allowing any client to associate). Second, the client's wireless link to the AP may be too lossy to permit successful association. Third, security mechanisms such as MAC filtering or WEP/WPA with shared key authentication would prevent association unless the client's MAC address is on the "allowed" list and the client has the correct key.

Finally, software or hardware incompatibilities could prevent the client from communicating with and hence associating with the AP. The client might have a buggy or outdated driver for the wireless NIC. There might be incompatibility between the wireless NIC and the AP hardware. For example, the AP might be using the short preamble optimization by default but the client's NIC might not support it. Although the WiFi Certification process [9] tests for interoperability, incompatibilities arise in practice, whether because non-certified equipment is used or because not all features are thoroughly tested. (For example, the U.S. National Science Foundation alerts visitors to specific incompatibilities arising from their wireless LAN infrastructure [8].) Indeed, Intel has found it necessary to create a separate Wireless Verification Program, over and above WiFi Certification, to ensure the interoperability of its Centrino chipset with APs from various vendors [5]. Note that even when there are interoperability problems, the client may be able to communicate with some of its peers, given the diversity of peer hardware/software configuration and the simplicity of such communication (e.g., no dependencies on WEP, MAC filtering, DHCP, etc.).

2.2.3 Inability to obtain an IP address

Even after it has associated successfully, there are several reasons why the client may fail to obtain a dynamic IP address via DHCP. First, the client may have an incorrect WEP/WPA key or other setting, which prevents communication while not preventing association (for instance, because open authentication is used). Second, the AP's wired interface might be malfunctioning or disconnected, preventing communication with the DHCP server. Finally, the problem might be at the DHCP server end, with the server being down or having exhausted its pool of addresses. The authors have encountered multiple instances of the latter (e.g., at the Sigcomm 2004 and Infocom 2005 conferences). Often wireless networks at conferences are set up hastily, with the default address pool size of 255 addresses (i.e., a /24 subnet, which is common in many DHCP server implementations), which is inadequate when there are several hundred attendees.

2.2.4 End-to-End Communication Failure

Obtaining an IP address is a necessary but not a sufficient condition for successful end-to-end communication over the wide-area network. While end-to-end communication failure could happen for reasons such as a server being down or disconnected, there are a number of reasons why a wireless client, in particular, might experience partial or total end-to-end communication failure.

First, DNS resolution might fail either because the client has an incorrect local DNS (LDNS) server setting or because there is a failure in the DNS infrastructure.

Second, a firewall might selectively block communication. For example, the authors have encountered wireless hotspots that block accesses to remote servers on SMTP port 25 (ostensibly as an anti-spam measure) and/or VPN ports (to encourage "business" users to subscribe to a higher tier of service).

Third, the network might require clients to use application-level proxies (e.g., web proxies). A client may encounter a failure either because it has a missing or incorrect proxy setting, or because the proxy it is configured to use is not functioning.

Finally, total communication failure might occur because of disconnection of the wireless LAN (including the APs) from the wide-area network, say because of equipment malfunction or misconfiguration.

2.2.5 Poor Performance

It is possible that wireless clients experience poor performance even if not hard communication failure. The poor performance might be reflected as low throughput or high latency for end-to-end communication. The problem could either be in the wireless link or in the wide-area network. The wireless link could be lossy because of a weak signal or noise. The wireless medium or the WAN connection could also be congested, sometimes because of a small number of misbehaving users. For example, at the Sigcomm 2005 conference, the organizers had to make a general announcement to ask the user(s) who were engaged in large eDonkey downloads to desist from this activity because of the network slowdown it was causing.

2.3 What a host can do in isolation

A wireless client that is experiencing problems like the ones discussed above can learn something about the nature of the problem based on local observations. For example, the client can tell whether it has detected any wireless networks or APs, been able to associate with a network or obtain an IP address, or experienced partial or total wide-area communication failure. However, local observations only provide a limited view. For instance, a client that is unable to associate with a network would not be able to tell by itself that it has an incorrect security setting (e.g., an incorrect

WEP key). A client that is unable to obtain a dynamic IP address would not be able to tell whether it is a persistent problem with the DHCP infrastructure or an intermittent one. The limited view provided by local observations serves as the motivation in WiFiProfiler to have wireless clients "compare notes" to diagnose problems more effectively and learn how they might resolve them.

3. DESIGN AND IMPLEMENTATION

We now present the design of WiFiProfiler, our system for cooperative network fault diagnosis in wireless LANs. Where appropriate, we also present specifics pertaining to our implementation of WiFiProfiler on the Microsoft Windows XP platform. The components of WiFiProfiler include:

1. **Sensing:** make local observations of network configuration and health at the individual wireless clients.
2. **Communication:** enable peer-to-peer communication among wireless hosts within range.
3. **Diagnosis:** infer the likely cause(s) of the problems experienced by clients and possible steps for resolution.

Before discussing each component in more detail, we note two requirements that underlie our design, both of which are motivated by the goal of facilitating the deployment and adoption of WiFiProfiler. First, although peer cooperation is a key element of WiFiProfiler, we seek to keep the burden on the participating hosts minimal. We restrict the peers' involvement to the "information plane" (i.e., for sharing network information) rather than the data plane (i.e., for packet forwarding). Second, we base our design on the capabilities of off-the-shelf 802.11 hardware and the information that is generally made available by wireless device drivers. While there is no universal standard for 802.11 drivers, given our target platform, viz. Microsoft Windows XP, we use the list of mandatory APIs defined at [6] as the guideline. Not having any specific dependencies on the wireless NIC hardware or driver eases deployment.

3.1 Sensing

The goal of the sensing component is to make passive observations of the network health and the (relatively static) network configuration information at the individual wireless clients. We seek to keep the overhead on the client computer as well as on the human user minimal. So, for instance, although much information about the wireless medium can be gleaned from operating the wireless NIC in RFmon or promiscuous mode [11], doing so for an extended length of time can impose a significant energy overhead. Furthermore, RFmon mode could affect the wireless connectivity of clients. We choose to sacrifice the ability to do such detailed sensing for the benefit of keeping the overhead minimal.

We are interested in observations across the layers of the protocol stack, since these could, individually or in combination, have a bearing on the user's network experience. We categorize the sensing information as pertaining to the wireless, network, transport, or application layers. We discuss these in turn.

3.1.1 Wireless Layer

We obtain information on the wireless configuration of the client, the wireless networks in the vicinity, and the condition of the wireless channel.

The wireless hardware/software configuration comprises relatively static information, which could have a bearing on the (in)ability of the client to connect to a wireless network because of incompatibilities between wireless NICs and APs.

- **NIC model:** the device ID of the host's wireless network interface card (NIC). The device ID is a globally unique, vendor-defined identifier of the NIC model [3]. For example, the device ID of a built-in Centrino 802.11b NIC is `PCI\VEN_8086&DEV_1043&SUBSYS_25818086&REV_04\4&16793A72&0&28F0`.
- **NIC name:** a human readable description of the wireless NIC. For the above Centrino example, it is "Intel PRO/ Wireless LAN 2100 3B Mini PCI". While the NIC model enables easy comparison across machines, the NIC name is convenient when interfacing with the user (e.g., when advising them to switch to a particular kind of NIC).
- **Driver version:** the version of the wireless NIC driver (e.g., "1.2.3.17" for the above Centrino NIC). This is driver specific and we treat it as an opaque string.

Besides the above information, there is other static configuration information (e.g., the power save setting) that might be of some relevance. However, we do not consider this information in our current implementation.

We obtain the list of wireless networks and APs in the vicinity of the client, including the one that the client may have connected to.

- **BSSID list:** the list of BSSIDs (basic service set identifiers) corresponding to the APs from whom beacons have been heard. For infrastructure 802.11 networks, the BSSID is the MAC address of the AP.
- **SSID list:** the SSID (service set identity) for each wireless network, which is a human-readable identifier of the network (e.g., "T-Mobile"). In general, there could be multiple BSSIDs (i.e., APs) corresponding to an SSID and a client can choose to associate with any of them.
- **RSSI list:** the RSSI (received signal strength indication in dBm) corresponding to each BSSID. Drivers typically report the average RSSI over a recent set of frames, which is sufficient for our purposes.[1]

For the wireless network that the client is connected to, we obtain information about the security settings that are visible to the client. Note that certain settings (e.g., MAC filtering) are made at the AP and are not visible to the client.

- **Security protocol:** whether WEP is in use for authentication and/or encryption. In general, we could also obtain information pertaining to other security mechanisms such as WPA, WPA-PSK (WPA in pre-shared key mode), and 802.1X.
- **Key/passphrase:** the WEP key (or WPA-PSK passphrase). Only a one-way hash of this information is shared with peers to avoid exposing the key.

[1]We would ideally like to obtain the RSSI from only the beacon frames, since that would give us a consistent sample. RSSI obtained from larger frames would tend to miss low RSSI values since the corresponding frames may not be successfully decoded. However, drivers vary in how they perform sampling.

Finally, we obtain the following information pertaining to the state of the wireless channel.

- **Beacon loss rate:** the fraction of 802.11 beacon frames from the AP that are not received at a client. We use this as an estimate of the frame loss rate on the wireless link, in the absence of a direct indication of frame loss from the wireless NIC (which typically uses link-layer retransmissions to mask frame loss).

  While it is advantageous to use the periodic beacon frames for this measurement because it imposes no overhead (since beacons are part of normal 802.11 operation), their small size could introduce a bias. Furthermore, although computing the beacons loss rate in the driver should be trivial and some drivers do so [4], not all drivers do. So we approximate it by having the clients themselves broadcast UDP "beacons" and measuring the loss rate of such beacons (inspired by the effectiveness of ETX [17]). Although we do not do so in our current implementation, information that is relevant to WiFiProfiler could be piggybacked on these beacons to increase their usefulness.

- **Interface queue length:** We sample the packet queue length at the wireless interface on a continual basis. A persistently backlogged queue might indicate congestion although not necessarily (e.g., the backlog could be because the link is very lossy, leading to significant backoff at the MAC level). We rely on this indirect indication of wireless congestion, since wireless NICs do not expose information on the busyness of the wireless medium (e.g., how often carrier sense finds the channel to be busy).

3.1.2 Network Layer

We gather information on the IP-level connectivity of wireless clients. The goal is to enable a client to determine, for instance, whether there is a DHCP server on the local subnet and how recently it has granted address leases to other clients. Some or all of the information below could be null depending on the state of connectivity of a client, for instance, if it has not been able to obtain an IP address.

- **IP address/subnet/mask:** the IP address, subnet, and netmask corresponding to the wireless interface.
- **IP mode:** whether the client's IP address is assigned statically or obtained dynamically using DHCP.
- **DHCP information:** the IP address of the DHCP server from whom an address lease was obtained, and the lease start/end times.
- **LDNS information:** the IP address(es) of the local DNS server(s).

3.1.3 Transport Layer

We gather information at the transport layer to learn about the health of end-to-end network connectivity over the wide-area network. Even if the wireless subnet itself is operating normally, end-to-end communication could be impeded by firewalls, or congestion on or disconnection of the WAN link. While we would like to tell whether the WAN is responsible for a problem, we do not attempt to diagnose problems within the WAN itself (e.g., routing problems).

We obtain the following information pertaining to TCP:

- **Failed connection attempts:** the total number of connection attempts and the number that failed.
- **Packet retransmissions:** the count of outgoing TCP segments that are retransmitted.
- **Server port numbers with successful TCP connections:** whether the client has been able to successfully establish connections on a set of well-known server port numbers (e.g., HTTP, SMTP, etc.). If the client is never able to establish connections on certain ports, it might be because a firewall is blocking access to those ports.

We obtain the above information in a lightweight manner by periodically polling (once per second in our implementation) the protocol state reported by the network stack. This information includes the overall connection attempt and failure counts as well as the state of each active connection. If a connection to a remote port reaches the ESTABLISHED state or beyond (e.g., the TIME_WAIT state), we infer that a connection has successfully been established to that port. On the other hand, if a connection is ever in SYN_SENT state but is never seen to be in the ESTABLISHED state or beyond, we infer that the connection attempt must have failed. Although we only sample the protocol state once per second, this is sufficient in practice for sampling connections. When there is port blocking, the connection initiator typically remains in the SYN_SENT state for several seconds, waiting for a SYN-ACK response, before giving up. In the case of successful connections, the initiator typically remains in the ESTABLISHED state and/or TIME_WAIT state for several seconds.

This lightweight procedure does not yield similar information for UDP because there is no UDP connection state. On Windows XP, we could infer the success or failure of UDP communication to various ports based on a trace of TDI events (which records Transport Driver Interface events [7] such as the sending and receiving of datagrams) or a packet-level trace obtained using the libpcap or netmon filter driver. The latter would also enable us to glean more detailed information for TCP connections (e.g., the packet loss rate). However, we do not obtain more detailed TCP information or information pertaining to UDP in our current implementation.

3.1.4 Application Layer

At the application layer, we are interested in configuration information that is of relevance to network communication. In our current implementation, we focus only on web access, given its dominance among networked applications:

- **Web proxy setting:** whether an HTTP proxy is being used and if so its host name and port number.

3.1.5 Summarizing Sensing Information

Each client obtains a snapshot of the above sensing information periodically, by default once per second. Since the volume of sensing information can grow to be large, we need to summarize it to reduce the overhead of sharing it with peers when needed. The summarization procedure varies depending on the type of the sensed information. For configuration information such as the NIC type, security settings, or LDNS server IP address, the summary only contains values from the most recent snapshot. For dynamic performance-related information such as TCP connection failures and interface queue length, we compute an aggregate metric over a certain period (60 seconds for wireless-related information and 300 seconds for TCP-related information). The aggregation is based on computing an average (as for the TCP

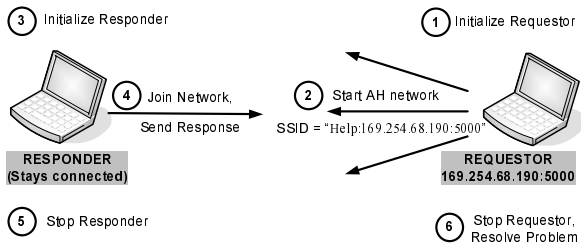

Figure 2: Steps in the communication process

connection failure rate) or on applying a threshold (as for interface queue length, where we calculate the fraction of samples corresponding to a non-empty queue). Finally, for information such as the BSSID list and SSID list, we compute the union of all distinct values seen in the recent past (30 seconds by default, to accommodate channel scanning delays).

3.2 Communication

The communication component of WiFiProfiler enables a wireless client that is experiencing problems (the "requester") to request and obtain network health information from its peers (the "responders").

The problem of data sharing in WiFiProfiler is non-trivial. The main reason is that the responder's wireless card can be associated to only one wireless network at a time. It can only communicate with hosts that are connected to the same wireless network or are reachable through the AP's wired backhaul. It cannot communicate with a disconnected wireless client, even if it is within radio range. So, in order to respond to the requester in another wireless network or on a disconnected wireless network, the responder would have to disconnect from its current network, associate to the requester's network, send out its response, and then reconnect to its original network. During this time, the responder would not have connectivity to its original wireless network. This would discourage connected machines from using WiFiProfiler to help their peers.

The alternative is for all connected clients to send their information to a centralized location, such as an AP. However, a disconnected client would not be able to retrieve the information. Besides, this would require infrastructure support that we want to avoid in WiFiProfiler.

The communication component of WiFiProfiler addresses this problem by providing a framework for nodes to quickly exchange information, even in the case when they are not on the same network, or when they are disconnected. Furthermore, it imposes very little overhead on the responder's primary connection. The responder does not see a significant drop in throughput. The low overhead also suggests a minimal impact on battery lifetime, although we do not present energy measurements in this paper.

The design of the communication layer is driven by two observations. First, a disconnected node can initiate an ad hoc network for exchanging information with the responders. Second, a responder can connect to the requester's ad hoc network without disconnecting from its primary network. This is possible using two wireless NICs, or using VirtualWiFi [15] over a single wireless NIC, as discussed below.

Each client using WiFiProfiler has two network adapters: a *Primary Adapter* and a *Helper Adapter*. The client uses the Primary Adapter for its normal communication. It only activates the helper adapter when network health information needs to be exchanged with peers. This architecture can be realized in two ways. If a client has only one wireless NIC, we use VirtualWiFi to abstract the physical card as two virtual wireless cards. We use one virtual card as the primary adapter, and the other as the helper adapter. If a client has two wireless NICs, we dedicate one of them as a helper adapter.

In the rest of this subsection, we first describe the steps of the communication protocol in detail. We then describe its implementation for two cases: when a client uses VirtualWiFi and when it has two wireless NICs.

3.2.1 Steps in the Communication Protocol

The communication protocol works as follows. If the requester and responder are connected to the same network, or are reachable through the AP's wired backhaul, they exchange information using it. However, if the requester is disconnected, then our communication protocol uses a different scheme. The steps of this protocol are illustrated in Figure 2, and we describe them in detail below.

1. **Initialize Requester:** When a client experiences wireless network problems, it activates the helper network adapter.

2. **Start AH Network:** The client then starts an ad hoc network over the helper network adapter. The SSID of the ad hoc network is set to: "Help:<IPaddr>:<Port>" (up to 26 bytes long), as shown in Figure 2. $IPaddr$ is the IP address of the requester, and $Port$ is the port on which it is listening for responses. Our use of the SSID for signaling is inspired by [11]. Note that we could reduce the SSID length to 11 bytes by sending only the last two octets of $IPaddr$ (the first two are implicitly set to 169.254, the autoconfig prefix) and fixing $Port$ to a default value.

3. **Initialize Responder:** Peers in the neighborhood of the requester periodically scan for new wireless networks as part of normal operation (e.g., Wireless Zero Configuration in Windows XP). Each peer parses the SSID field of new ad hoc networks that it detects, and checks to see if it matches the pattern described above. If the SSID corresponds to a new requester, the peer activates its helper adapter.

4. **Join Network, Send Response:** The responder joins the ad hoc network set up by the requester, and sets up a socket connection with the IP address and port specified in the SSID. The responder then sends the requested information on that socket.

5. **Stop Responder:** After sending its response to the requester, the responder closes the socket connection and stops the helper adapter.

6. **Stop Requester:** The requester stops when it has received a sufficient number of responses to diagnose the problem or if it times out. It shuts down the socket, and stops the helper adapter, thereby tearing down the ad hoc network.

This scheme creates a new plane for exchanging network health information, even across wireless clients that are connected to different networks or are disconnected.

3.2.2 Communication Protocol using VirtualWiFi

When a machine has only one wireless NIC, WiFiProfiler uses VirtualWiFi to simultaneously connect the responder to its primary and helper wireless networks. VirtualWiFi [15] is a virtualization architecture for wireless network cards that allows a user to simultaneously connect to multiple wireless networks using a single wireless card. It comprises a kernel driver and a user-level service (i.e., daemon). The driver abstracts a single card into multiple virtual wireless cards, where each virtual card corresponds to a different network that the user wants to connect to. Each virtual card maintains the association state for the corresponding wireless network, such as the SSID and the network mode. The VirtualWiFi driver also has mechanisms for loading the state of a virtual card onto the physical wireless card. The user-level service implements a network-hopping scheme that switches the physical card across multiple wireless networks, activating the corresponding virtual adapter each time. The time slice devoted to each virtual card is configurable. The details of the virtualization architecture and its implementation, including how it uses IEEE 802.11 Power Save Mode (PSM) to avoid packet loss, appear in [15].

However, VirtualWiFi takes around 15 seconds [14] to add a virtual adapter for a network. This overhead is unacceptable for a responder. In our implementation, we modified VirtualWiFi to reuse a virtual adapter across many wireless networks. So the wireless state of the virtual adapter (e.g., SSID and network mode) is changed dynamically from the user level. WiFiProfiler uses this scheme to initialize two virtual adapters during installation: a Primary Virtual Adapter and a Helper Virtual Adapter. The state of the primary adapter is set to the user's primary wireless network. The helper adapter's state is initially set to a default value. The VirtualWiFi service is stopped so that the user is on the primary network.

The communication protocol performs the following steps when used with VirtualWiFi. Upon disconnection, the requester activates its helper virtual adapter and configures it with the appropriate parameters (e.g., the "help" SSID noted above), to start the ad hoc network. When a responder detects an ad hoc network with the "help" SSID, it activates its helper adapter and configures it with the SSID of the requester's ad hoc network. The VirtualWiFi service starts switching the physical card across the primary and helper networks. The responder sets the helper adapter's SSID and mode to that of the requester's ad hoc network. After sending its response, the responder unbinds its helper adapter from the physical NIC, by stopping the VirtualWiFi service. Note that the software stack corresponding to the helper adapter is still preserved. Finally, the requester stops the ad hoc network by activating its primary adapter.

All the above steps require `ioctl` calls to be made to the VirtualWiFi driver. These operations are quite lightweight and complete within a few milliseconds.

3.2.3 Communication Protocol using Two NICs

When a machine has two NICs, WiFiProfiler uses one of them as the helper adapter, thereby avoiding the need to disturb the connection on the primary adapter. Since the helper adapter is useful only when the client is responding to a request, we disable it during normal operation to save battery energy.

WiFiProfiler assigns a static IP address to the helper adapter, even though it is not connected on any network. It does so for two reasons. First, it allows the requester to set its IP address in the beacon in Step 2 of Figure 2. Second, it allows the responder to quickly set up its network stack in Step 4. Note that both these issues do not arise for VirtualWiFi where the software stack corresponding to the helper virtual adapter is always maintained, even when the adapter is not bound to the physical NIC [15].

The communication protocol performs the following steps when using two NICs. Upon disconnection, the requester enables its helper adapter. It uses `ioctl` calls to set the mode and SSID of the ad hoc network. The primary adapter on all clients keeps scanning the wireless channels for the requester's beacons. Upon detecting a requester, the responder activates the helper adapter. The helper adapter then scans all the channels to locate the requester's network. After discovering the network, it issues an `ioctl` to join the ad hoc network. After sending its response, the responder disables its helper adapter.

Disabling and enabling a network adapter incurs a significant overhead in Windows XP. These operations are much more heavyweight than the `ioctl` calls used by VirtualWiFi, as discussed in Section 4.2.

3.2.4 Other Issues

A key consideration for the communication component is to keep the overhead on the responders low. We accomplish this through several means.

First, we summarize the sensing information at a node in 1200 bytes, which fits in a single packet. As a result, we are able to keep the request-response protocol simple — there is only a single request type and a single response type, viz., one that asks for and returns the full sensing information. The requester's beacons on the helper ad hoc network serve as an implicit request for peers to return their full sensing information (i.e., information across all layers). The compact message size obviates the need for separate request/response message types for different kinds of problems (e.g., wireless association problem, TCP connectivity problem, etc.).

Second, we use UDP for communicating the response back to the requester, which minimizes overhead on the responder, especially in the case where the requester is disconnected. Given the compact size of the response, the responder needs to send just a single packet and then can leave the ad hoc network. Since the (disconnected) requester remains on the ad hoc network continuously, there is a high likelihood of it receiving the UDP packet sent by the responder. While TCP would provide full reliability, it entails a longer wait for the responder. Multiple round-trips would be involved for connection setup and the data-ack exchange. Each round-trip could take long if the responder uses VirtualWiFi to switch between the ad hoc network and its primary network in the interim.

Third, we limit the rate at which nodes respond to requests for help. This provides protection from a malicious requester who sends requests constantly to deplete the resources of the responders and degrade their performance. We also have responders wait for a random length of time before switching to the ad hoc network and responding. Doing so is advantageous in dense network settings, where there is a large number of potential responders. Once the requester has received the requisite number of responses, it can terminate its request by leaving the ad hoc network, thereby stopping its beacons requesting help. So other potential responders do not have to incur the overhead of switching to the ad hoc network and responding. In our implementation, we rate limit responses to one per minute and also pick a waiting time uniformly distributed between zero and one minute before responding. Thus the requester is assured of

receiving responses within a minute, which is still quick from the viewpoint of helping a user who is experiencing network problems.

There are further optimizations that we plan to consider in future work. A responder could monitor local network activity and switch to the ad hoc network and send its response only when there is a relative lull in activity. This would reduce the impact on the responding user's network activities.

Responders could also cache responses they may have received in response to a recent request and relay these to the new requester. This, coupled with the random wait mentioned above, would limit the number of responders who incur the overhead of responding, while still providing the requester with sufficient information. This idea could be extended to have the requester first contact other disconnected nodes (say using a special SSID to signal to such nodes) to retrieve information that the latter may have themselves recently obtain from their peers (both connected and disconnected nodes). This would avoid imposing on the connected nodes repeatedly.

Finally, in our current design, we only seek responses from one-hop neighbors. That is, there is no relaying of requests or responses. This works well in dense wireless LANs, where there is a large number of potential responders within range that could provide information that is relevant to the network problem that the requester is encountering. However, if there is an inadequate response, the requester could send a fresh request and have it relayed over multiple hops. We defer this enhancement to future work.

3.3 Diagnosis

The diagnosis component uses the information aggregated from the peer nodes to infer the likely cause of the problem. In some cases, this component also suggests ways of resolving the problem. The user initiates diagnosis when he/she is experiencing network problems. His/her client host determines the problem symptoms (e.g., association failure) automatically based on local information and then proceeds to diagnose the problem using information from peers. We use a simple GUI to interface with the user.

We consider the problem symptoms and associated causes described in Section 2.2. Except for our discussion in Section 3.3.1 below, we assume that WiFiProfiler-enabled clients are present in the vicinity of the requester and respond to its request for help. Nevertheless, the number of peers from whom the requester is able to obtain information may be small, often in the single digits. This limited number of samples may not be amenable to statistical analysis and inference techniques. So we consider a rule-based procedure, which allows us to draw conclusions based on observations made at a small number of peers. We start with checks pertaining to the problems that have the most severe impact and then proceed to the less severe problems. The basic observation underlying our diagnosis procedure is that the extent of a problem across clients (e.g., whether it affects only a subset of clients or all clients) is often indicative of the nature of the problem.

3.3.1 Inability to detect an AP

The client host may be unable to detect beacons from any AP for several reasons: (a) there are no APs in its vicinity, (b) there are APs in its vicinity, but their beacons are not detected at the client's current location, (c) incompatibility between the client's wireless NIC or driver and the AP hardware is preventing the client from successfully receiving beacons, or (d) the client's wireless NIC is not functioning.

Based on the aggregated information from peers, we diagnose the problem using the following steps in order:

1. If the client does not hear from any peers, it is likely that either there are no WiFiProfiler-enabled clients in its vicinity or the client's wireless NIC is not functioning. If the client is able to sniff the medium, and this reveals the presence of other wireless traffic yet no response is received, it is likely that none of the clients in the vicinity supports WiFiProfiler; otherwise, NIC malfunction is the likely culprit. We assume that the user has determined through out-of-band means (e.g., a visual check of their surroundings) that they are in a WiFi-enabled location, thereby discounting the possibility that the absence of wireless traffic is due to their isolated location.

2. If a peer with the same wireless NIC type and driver version reports seeing beacons, it is likely that the client's current location is preventing it from hearing beacons.

3. If all the peers, if any, that have the same wireless NIC type as the client and are receiving beacons have different version(s) of the driver, it likely that either the client's wireless driver or its location is the cause of the problem.

4. If the only peers that are receiving beacons have different wireless NIC types, it is likely that either the client's NIC type, its wireless driver version, or its location is the cause of the problem.

The resolution of the above problems would involve user action to change NICs, install a new driver, or change location.

3.3.2 Inability to associate with an AP

The client host may be unable to associate with an AP because: (a) the AP employs a security mechanism such as MAC filtering or WEP with shared key authentication (or other mechanisms such as WPA), (b) the wireless link is too weak at the client's current location to permit the two-way exchange needed for association, (c) incompatibility between the client's wireless NIC or driver and the AP hardware is preventing the client from associating successfully.

Based on the aggregated information from peers, we diagnose the problem using the following steps in order:

1. If the client's authentication configuration does not match that of its peers who have successfully associated, this inconsistency is the likely cause of the problem. The client might be configured to use open authentication when shared key authentication is required, or vice versa. Or the client might have an incorrect WEP key. Note that this can be determined without exposing the key, by exchanging and comparing one-way hashes of these quantities. Nevertheless, doing so raises the possibility of dictionary attacks, a point we discuss in Section 5.1.

2. If the client is experiencing a significantly higher beacon loss rate than its peers who have successfully associated, it is likely that the weak link resulting from the client's location is the cause of the association failure. However, as noted in Section 3.1.1, not all drivers provide beacon loss rate information and our alternative procedure of using broadcast packets only works

for clients that have associated. So instead we check whether the RSSI at the client is significantly lower than that at the peers that have successfully associated.

3. If a peer with the same wireless NIC type and driver version has associated successfully, the client's problem is likely due to MAC filtering at the AP, which is not directly detectable at the client. Even if no such peer is found but there is a pair of peers with (mutually) matching NIC type and driver versions such that one of them is able to associate successfully while the other is not, that would indicate MAC filtering is being used (assuming that the authentication configuration or beacon loss rate checks above did not indicate a problem for this pair of peers). Note that even if these checks fail, MAC filtering remains a potential cause, as noted next.

4. If the above checks have not narrowed down the cause, the possible causes of the problem are the client's NIC type, its wireless driver version, MAC filtering, or AP malfunction.

The resolution of the above problems would involve user action to set the correct authentication key/passphrase, change location, change NICs, install a new driver, or have the operator register the NIC's MAC address with the MAC filter.

3.3.3 Inability to obtain an IP address

Once it has associated, the client may fail to obtain an IP address because: (a) it has an incorrect WEP encryption setting, which is preventing communication with the AP, (b) there is a hardware malfunction or disconnection at or close to the AP that is preventing communication with the DHCP server, or (c) the DHCP server is down or out of addresses, and so is not responding to fresh DHCP requests.

Based on the aggregated information from peers, we diagnose the problem using the following steps in order:

1. If the client's WEP encryption configuration does not match that of its peers who have associated successfully, this inconsistency is the likely cause of the problem. The checks performed are akin to those for authentication configuration in Section 3.3.2 above.

2. If one or more other peers is in a similar state (i.e., has associated successfully and has the correct encryption configuration but has been unable to obtain an IP address) and none of the peers has obtained a DHCP lease since the failed attempts by the client and the other affected peers, that would indicate a problem in reaching the DHCP server, either because of general connectivity problems or because of DHCP server problems.

3. If at least one peer reports having performed wide-area communication successfully in the recent past, that makes general connectivity problems less likely to be the root cause. So failure or address exhaustion at the DHCP server would be the likely cause.

The resolution of the above problems would involve user action to set the correct encryption configuration information, or operator action to resolve the DHCP server problem or hardware malfunction/disconnection problem.

3.3.4 End-to-End Communication Failure

End-to-end communication can fail either because of DNS resolution failure or because of end-to-end connectivity problems. The client can determine locally which of these two kinds of failures it is experiencing.

A DNS resolution failure could happen because (a) the client has an incorrect local DNS (LDNS) server setting, (b) the LDNS server is down or unreachable, or (c) there is a general problem with DNS that is not specific to the local wireless network.

Based on the aggregated information from peers, we diagnose the problem using the following steps in order:

1. If a peer with a different LDNS server setting reports a high success rate for DNS resolution attempts and no peer with the same LDNS server setting reports a high success rate, the problem is likely due to an incorrect LDNS server setting at the client.

2. If all peers report a high failure rate for DNS resolution, with no response from the LDNS server, it is likely that the server is down or unreachable.

3. Otherwise it is likely that there is a general DNS problem (say because of misconfiguration or WAN connectivity issues) that is causing DNS resolution failure. Given the limited number of peers and hence limited degree of sharing, we are not in a position to narrow the problem down, say to identify the particular domains, if any, for which peers are consistently experiencing DNS resolution failures.

Resolution would involve the user changing the client's LDNS setting, if that is the cause of the problem. Otherwise operator intervention would be needed to look into the state and configuration of the LDNS server.

If there is end-to-end communication failure despite successful DNS resolution, the cause could be (a) an incorrect application proxy setting or an application proxy that is down or disconnected, (b) a firewall that is blocking access, or (c) a connectivity problem between the wireless LAN and the wide-area network.

Based on the aggregated information from peers, we diagnose the problem using the following steps in order:

1. We diagnose problems pertaining to the application proxy setting (only the web proxy setting, in our current implementation) in a manner similar to the procedure used for diagnosing problems with the LDNS setting.

2. If the client and its peers are consistently experiencing failures when communicating on specific ports (only TCP ports, in our current implementation) but have been able to successfully communicate on other ports, it is likely that a firewall is blocking communication on the affected ports. We focus on port-based blocking since it is commonly employed to block specific applications. Also, there is a greater likelihood of a shared experience among the wireless peers if there is blocking of accesses to certain well-known ports (e.g., if access to the SMTP port were blocked, peers accessing SMTP servers would be affected regardless of which server they access) than if the firewall were using a different policy for blocking (e.g., blocking accesses to certain remote addresses).

3. If at least one peer is able to communicate on the problematic port, it is likely either the specific remote host that the client is trying to communicate with is unreachable or that a firewall policy other than port-based blocking is in effect.

4. If no peer reports successful end-to-end communication, it is likely that there is a connectivity problem between the wireless LAN and the wide-area network. While it is possible that this problem is AP-specific (i.e., affecting clients associated with one AP but not those associated with other APs on the same wireless LAN), it is unlikely to be so since AP-specific problems would likely have manifested themselves as DHCP or DNS failures.

Resolution would involve the user changing the proxy setting, if that is the cause of the problem. Otherwise operator intervention would be needed to address the firewall configuration or WAN disconnection issues.

3.3.5 Poor performance

The client may experience poor network performance because (a) its wireless link is weak, (b) the wireless medium is congested, or (c) there is a WAN problem (e.g., congestion or routing instability).

Based on the aggregated information from peers, we diagnose the problem using the following steps in order:

1. If the beacon loss rate at the client is significant, a weak wireless link is the likely cause of poor performance. However, as noted in Section 3.1.1, the "beacons" in our current implementation are just broadcast packets transmitted by the individual wireless nodes. As such, an individual node does not know how many beacons have been transmitted in all or how many were lost on the hop from the source node to the AP, making it difficult for it to compute the beacon *loss rate*. Instead, we have each peer report the *number* of beacon packets it has received in the recent past. If the number of beacons received by the client is significantly lower than the highest count reported by a peer, we infer that the wireless link to the client is weak and is the likely cause of poor performance.

2. If one or more peers reports persistent queuing at its wireless interface but none of them reports a weak wireless link, the wireless medium is likely congested. The check that the wireless link is not weak is necessary because otherwise MAC backoff might account for the queuing even when the wireless medium itself is not congested (Section 3.1.1).

3. Otherwise, either there is a WAN problem or there is wireless congestion but none of the peers is transmitting fast enough to experience any local queuing (i.e., they are all only downloading, causing persistent queuing at the AP's wireless interface, unbeknownst to the clients).

If the wireless link or medium is the cause of the problem, resolving the problem would involve the user changing his/her location or switching to a less congested AP or network, which may be operating on a different, less congested channel. If there isn't a different AP or network to switch to or if the problem is not wireless related, operator intervention would be needed to resolve the problem (e.g., to shut off a bandwidth hog, like the eDonkey user at Sigcomm 2005 noted in Section 2.2.5).

3.3.6 Discussion

We now discuss a couple of issues pertaining to the diagnosis procedure. First, there is inherent uncertainty in the diagnosis is some cases. For example, in the absence of direct information from the infrastructure, it is impossible to definitely establish that a particular MAC address is being filtered. We can only make an informed conjecture that this is happening because it is common practice and other factors (e.g., NIC type, driver version, etc.) have been ruled out based on information from peers. Nevertheless, we believe that it is useful to provide users with a short list of potential causes, even if a single definitive cause cannot be determined.

Second, there is the possibility of conflicting information being provided by peers. Assuming that peers report information accurately (we defer discussion of deliberate deception to Section 5.1), any disagreement in the information provided by peers would be resolved as part of the diagnosis procedure discussed above. For example, in the case of association problems, if two peers with identical NIC type and driver version report association success and failure, respectively, the requester can rule out incompatibility of the NIC type or driver version as a possible cause.

4. EXPERIMENTAL EVALUATION

We now present an experimental evaluation of the various components of WiFiProfiler. Our testbed consists of a subnet with a D-Link DWL-7100AP access point providing connectivity between a wired server host and wireless clients. We keep this subnet isolated from our corporate network, to avoid creating a "rogue" AP. The wired host runs a web server, in addition to providing firewall functionality using the Internet Connection Firewall (ICF) feature in Windows. The D-Link AP itself provides DHCP service. We used 7 laptop-class wireless clients: 5 Compaq Evo N800c laptops (Pentium-4 2GHz, 512 MB) and 2 Toshiba Protege 3500 Tablet PCs (Pentium-III 1.33 GHz, 1 GB). The machines were equipped with built-in 802.11 NICs. In addition, we used Cisco Aironet 340 and Orinoco 802.11ag ComboCard Gold PCMCIA NICs for some of our experiments. We set the AP to 802.11b mode for all of our experiments.

4.1 Evaluation of sensing

The key questions with regard to sensing are its accuracy and the overhead it imposes on the wireless host. We discuss each of these in turn.

Much of the information that is sensed by WiFiProfiler pertains to static or dynamic configuration (e.g., the authentication settings, whether the client has a DHCP address, etc.). The discrete nature of this information means that its accuracy is guaranteed (modulo software bugs). So we focus our discussion here on sensing information pertaining to the quality of the wireless link, where there is more uncertainty.

4.1.1 Sensing the quality of the wireless link

WiFiProfiler needs to sense the quality of the wireless link between a client host and the AP it is associated with in order to present the appropriate diagnosis (and resolution advice) to a user who is experiencing poor end-to-end performance. If the link is weak and lossy, we would like to advise the user to change his/her location to improve their connectivity to the AP. On the other hand, if the wireless medium is congested, the user should switch to a different AP or network on a non-overlapping channel, if available. If congestion on the wired network is culprit, switching wireless connectivity would not help, unless doing so also changes

NIC	Metric	Location					
		$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$
Compaq	RSSI (dBm)	-34	-62	-77	-80	-82	-93
	BLR (%)	0.3	1.2	3.3	7.0	19.5	27.7
Toshiba	RSSI (dBm)	-27	-60	-77	-83	-84	-95
	BLR (%)	0.4	0.6	2.0	5.0	11.2	37.5
Cisco	RSSI (dBm)	-37	-67	-82	-87	-89	NA
	BLR (%)	1.3	2.2	5.5	61.9	90.0	NA
Orinoco	RSSI (dBm)	-41	-74	-86	-88	-89	NA
	BLR (%)	1.2	2.5	43.4	IA	IA	IA

Table 1: The received signal strength (RSSI) and beacon loss rate (BLR) measured at different locations and with different NICs. The measurement data is unavailable when the client has no association (NA) to the AP. The BLR metric cannot be meaningfully computed even when the client has intermittent association (IA).

the WAN connectivity.

Accurately sensing the quality of the wireless link is challenging because wireless drivers typically do not expose information on the link loss rate and the link busyness. In fact, carrier sensing and link-layer retransmissions in 802.11 are performed in the wireless NIC and are thus masked from the host software. Also, the wide range in the nature of end-to-end communication that clients may be engaged in (web browsing versus bulk downloads, remote versus local servers, etc.) makes it difficult to meaningfully compare end-to-end communication performance information obtained through passive monitoring. So we investigate how effectively we can sense link quality using information that is available, viz., the received signal strength (RSSI) and the beacon loss rate (approximated using broadcast UDP packets with 4-byte payloads as the "beacons", per Section 3.1.1).

In the first experiment, we examine the relationship between RSSI and the beacon loss rate (BLR). We place a client at 6 different locations in our office building (labeled $\mathcal{A}$ through $\mathcal{F}$), at increasing distances from the AP. Since the NIC hardware/driver could have an impact on the measurements, we repeat our measurements with 4 different wireless NICs: the built-in Compaq and Toshiba NICs as well as the external Cisco and Orinoco NICs.

Table 1 shows the RSSI and BLR measured at the 6 locations. While the trend is similar for the 4 NICs, there are noticeable differences in the actual RSSI and BLR values. Indeed, some NICs are even unable to associate with the AP at certain locations. These differences are not unexpected given the disparity in the RF circuitry, antenna design, manufacturing tolerances, and sampling performed by the driver. However, in all cases the BLR exceeds 5% when the RSSI drops below -80 dBm. So -80 dBm could be used as a threshold for deciding when the wireless link is significantly lossy. This would be useful, for instance, if the BLR were not available. Note, however, that a stronger RSSI does not necessarily mean that the link is not lossy, because of noise or interference.

In Table 2, we report the TCP throughput (with the Toshiba NIC) in the AP-to-client direction measured using a 16 MB `ttcp` transfer. We find that the throughput drops sharply when the BLR exceeds 5%. In fact, at higher BLRs, the 16 MB transfer does not even complete. This again is consistent with the thresholds for a "lossy" link noted above. Note that BLR is an optimistic measure because of the relatively small size of the beacon packets. Large data packets are likely to suffer a higher loss rate under similar circumstances.

Thus we conclude that the BLR and/or RSSI can serve as an effective, even if not perfect, indicator of link lossiness. This permits us to correctly distinguish between performance degradation due to link lossiness from that due to congestion.

NIC	Metric	Location					
		$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$
Toshiba	RSSI (dBm)	-27	-60	-77	-83	-84	-95
	BLR (%)	0.4	0.6	2.0	5.0	11.2	37.5
	Thruput (KB/s)	565	572	451	147	IC	IC

Table 2: The TCP downlink throughput at various locations with the Toshiba WLAN NIC. The RSSI and BLR information is repeated from Table 1 for easy reference. "IC" indicates that the TCP remained incomplete because of extremely poor performance.

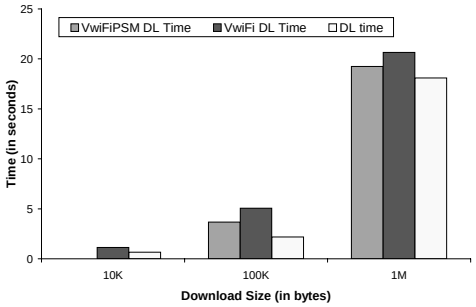

Figure 3: Impact of the communication protocol on web page download time at the connected client. The VwiFiPSM bar is missing in the 10 KB case because the download finished too quickly for us to be able to reliably interpose VirtualWiFi switching during the download.

4.1.2 Overhead of sensing

We consider the overhead that sensing imposes on the client host. If sensing is to be an ongoing process in WiFiProfiler (which offers the benefit of reducing diagnosis latency compared to on-demand sensing), it is important that the overhead be low, in terms of both CPU usage and network performance impact.

In our current implementation of WiFiProfiler, the sensing component gathers a snapshot every second. In addition, it scans other wireless channels every 10 seconds to discover other APs as well as peers seeking help. Despite these activities, we find that the overhead is negligible. The sensing component uses under 1% of the CPU even on the slower (1.33 GHz) Toshiba laptops. Also, there is no measurable impact on network performance (as quantified by the `ttcp` throughput for a 16 MB transfer). Nevertheless, other components of WiFiProfiler — the communication component in particular — could impose a greater overhead, so we turn to it next.

4.2 Communication Layer Performance

In this subsection, we evaluate the performance of the communication component of WiFiProfiler. We first quantify the overhead of providing help on a client's network performance. We then microbenchmark various steps of the communication protocol at the responder. Finally, we look at the total time taken for the request-response exchange to complete.

We evaluate the communication protocol for all combina-

	Initialize Responder	Send Packet	Stop Responder
VWiFi$_I$	0.039 (0.042)	0.064 (0.030)	0.004 (0.004)
VWiFi$_{II}$	0.079 (0.046)	1.434 (0.058)	0.007 (0.008)
Two NICs	5.614 (0.520)	5.271 (1.186)	5.861 (2.222)

Table 3: Time (in seconds) at the responder to complete each step of the communication protocol. The standard deviation is presented in braces.

tions of the requester and the responder using VirtualWiFi or two NICs. When using VirtualWiFi, we set the time spent on the helper network to 500 ms and that on the primary network to 800 ms, in each cycle. In our experiment, we disconnect the requester from its primary wireless network, and have it initiate the request process. A responder node placed in the vicinity of the requester then goes into helping mode, sends a 1200-byte response to the requester (the typical size of the sensing state in our current implementation), and then exits the helping mode.

4.2.1 Impact of Providing Help on the Responder

We first evaluate the overhead of providing help on the responder's network performance, by measuring its impact (compared to when it is not in helping mode) on the time to complete web downloads of three different sizes: 10 KB, 100 KB, and 1 MB. We set up a web server on the local wired subnet and use the WiNE network emulator [10] to introduce an additional round trip latency of 75 ms between the responder and the web server. Upon hearing beacons from the requester, the responder first issues a `wget` request before starting to help. This corresponds to the worst case scenario where the `wget` download coincides with the period when the responder is engaged in WiFiProfiler helping activities.

We present download time results for three different cases in Figure 3. In the first case (marked as "DL time" in the figure), the responder uses separate primary and helper NICs. So the time for the `wget` download via its primary NIC is unaffected by the WiFiProfiler helping activity. This serves as a baseline for comparison. In the second case (marked "VWiFiPSM DL time"), the responder uses VirtualWiFi to emulate both the primary and helper adapters. Furthermore, the AP implements IEEE 802.11 Power Save Mode (PSM), which VirtualWiFi leverages to have packets buffered at the AP when it is on the helper ad hoc network (and hence unable to receive packets on its primary network). This ensures that the responder does not lose any packets due to VirtualWiFi switching. However, it does incur an additional delay, since it will not receive the buffered packets until it is back on the primary network, which takes about 800 ms.[2] The impact of this delay is greatest if it occurs during TCP's slow start phase, since it delays TCP's window growth. The third case (marked "VWiFi DL time") is when the AP does not implement PSM, so packets from the web server would be lost when the responder is on the helper ad hoc network. Nevertheless, the impact on the download time is acceptable in all cases: about 500 ms for the small downloads, and 2-3 seconds for the large downloads. Note again that our experiment considers a pessimistic case where the responder is in the middle of a download when it starts helping. The overhead would be lower (or even zero) if the helping happened during a less busy (or idle) period, per the optimization noted in Section 3.2.4.

[2]The responder is on the helper ad hoc network for 500 ms, followed by around 300 ms to switch back to the primary network. Although the switching might complete faster, VirtualWiFi requires nodes using PSM to provide an estimate of the switching time as a multiple of the beacon period (100 ms).

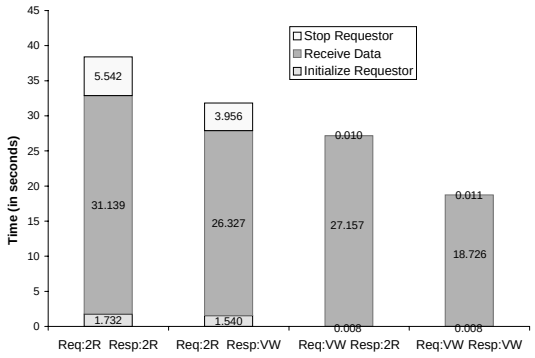

Figure 4: Time for the communication protocol to complete when the requester is waiting for one responder. 2R: Two Radios (NICs), VW: VirtualWiFi

4.2.2 Time Taken at the Responder

We now study the time taken by the responder to complete each step of the communication protocol, shown in Figure 2. Table 3 presents these numbers for two cases: when the responder is using VirtualWiFi and when it is using two NICs. In more than 45 runs when the responder used VirtualWiFi, around 50% of them finished within 150 ms, while the rest took around 1.5 seconds to complete. We denote these scenarios by VWiFi$_I$ and VWiFi$_{II}$ respectively. The reason for the difference, as evident from the table, is the time to send the response packet to the requester. VirtualWiFi sends the packets as soon as the card finishes switching to a network. In our experiments, we found this time to be highly variable. If switching takes longer than 500 ms, VirtualWiFi would have switched back to the primary network, and so the response would be delayed by one cycle until the responder reconnects to the helper network.

When using two NICs, all steps take significantly longer at the responder. Enabling and disabling the helper NIC (i.e., turning on and off the physical device) takes a few seconds, as compared to starting and stopping a service for VirtualWiFi, which only takes tens of milliseconds. Furthermore, the time to send a packet is also significant for the two NIC scheme. The helper NIC has to associate to the network, and initialize the network stack. It might even have to wait for the requester to initialize its network stack. Only then can it send the response packet. Note that we see this delay despite manually assigning an IP address to the helper adapter. The "Send Packet" step would take much longer if the adapter were to wait for an autoconfig IP address. VirtualWiFi does not incur this latency because it always maintains the network stack, including an IP address, for the virtual helper adapter.

4.2.3 End-to-End Latency of the Comm. Protocol

We now look at the the latency of each step in the communication protocol at the requester. The results for all possible combinations of the requester and the responder using either VirtualWiFi or two NICs is presented in Figure 4.

The communication protocol takes the longest time to complete when both the requester and the responder use two NICs. Initializing and stopping the requester requires enabling and disabling the helper adapter. Both these operations take a few seconds to complete. The time to receive a

response includes the time taken for the responder to detect the requester's ad hoc network (~18 seconds), for it to enable its helper adapter (~5 seconds), for its helper adapter to scan for the requester's ad hoc network, for the responder to join the ad hoc network, and for the requester and the responder to initialize their network stacks and to exchange data. The total time taken for these operations is approximately 32 seconds.

If the responder uses VirtualWiFi, the total time taken is less than the time taken with two NICs. This is so because the responder saves on the following steps: time to enable the helper adapter, scanning again on the helper adapter and initializing the network stack at the responder. However, this saves only about 5 seconds on average since the requester still waits for the responder to join its network before initializing its network stack.

The results are better when the requester uses VirtualWiFi. Here the responder goes through the same steps described for the two NIC case. However, the time to receive the data is smaller because the requester initializes its network stack before the responder joins the network. The overall time taken at the requester is also small because it does not spend time enabling and disabling the physical adapter.

Finally, we get the best results when both the requester and the responder use VirtualWiFi. In this case, the biggest overhead is the time to receive data. This delay is incurred at the responder, which has to scan across channels, looking for the requester's beacons.

4.3 Diagnosis Performance

In this section, we evaluate the performance and effectiveness of WiFiProfiler in diagnosing problems in wireless networks. Using our testbed, we inject different types of faults and see if WiFiProfiler running on the wireless clients makes the correct diagnosis under each scenario. Here are the faults we inject:

- **No beacon:** Place a disconnected client far from the AP with some other clients in the middle, i.e., within range of the AP as well as the disconnected client.
- **MAC filtering:** Filter a client's MAC address at the AP.
- **Incorrect WEP key for authentication/encryption:** Enable authentication/encryption and configure a client with the wrong WEP key.
- **DHCP problem:** First let 3 clients get IP addresses from the DHCP server, then turn off the DHCP server before the other 3 clients come online.
- **Port blocking:** Have different clients attempt communication on various well-known ports (TCP ports 21 (FTP), 22 (SSH), 23 (Telnet), 25 (SMTP), and 80 (HTTP)), but configure the firewall to block accesses to port 80 (HTTP). Our goal is to mimic a practical setting, where a group of clients is collectively communicating across a range of well-known ports (e.g., some users may be browsing the web while others may be sending email).[3]
- **Wireless congestion:** Create congestion by having 4 clients upload data and 2 clients download data simultaneously via the same AP.

[3]In the event that none of the clients attempts any communication on certain ports, passive sensing would not yield any information on the connectivity on those ports.

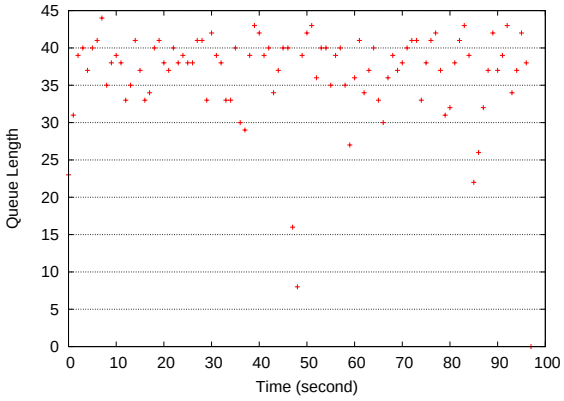

Figure 5: Queue length (in packets).

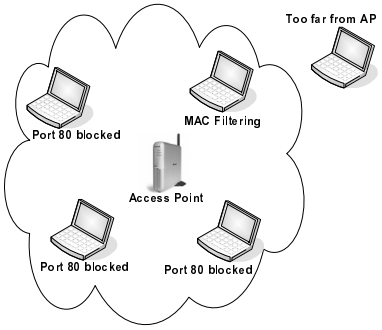

Figure 6: Setup for testing simultaneous diagnosis of faults.

We first inject one type of problem in each experiment. We repeat each experiment three times. Since we are interested in measuring the actual response time, we disable the random one-minute waiting time (noted in Section 3.2.4) for these experiments. In all of the experiments, WiFiProfiler is able to correctly identify the cause of the problem and the total time taken for diagnosis is under 40 seconds. Thus WiFiProfiler is much more responsive (and much less disruptive) than manual diagnosis, which could take several minutes, if not longer.

We now consider the wireless congestion scenario in more detail. When the wireless link is congested, we expect to observe queueing at the wireless interfaces of clients who are uploading data. There would *not* be such queuing if the congestion were elsewhere in the network. Figure 5 shows the output queue length at one of the clients that is uploading data. As expected, we find that the queue length at such a client is always non-zero, which indicates wireless congestion. We want to emphasize that WiFiProfiler can detect congestion as long as the congestion leads to queueing on at least one client (and that client does not have a weak wireless link, as noted in Section 3.3.5). But when all the clients are only downloading data, WiFiProfiler will not detect congestion since it has no knowledge of queueing at the AP.

Finally, we study the effectiveness of WiFiProfiler in diagnosing multiple simultaneous problems. We set up 5 clients and an AP as shown in Figure 6. The first client is placed far from the AP, such that it receives no beacons. The second

client is placed between the first client and the AP, such that it can both receive the beacons from the AP and communicate with the first client. But its MAC address is filtered by the AP. The remaining 3 clients are placed close to the AP and attempt to send HTTP requests. However, we configure the firewall to block access to port 80.

As expected, the first client diagnoses the problem as "no beacon" and is prompted to change its location because it knows of at least one peer (the second client) who is receiving beacons. The second client diagnoses the problem as "MAC filtering" because it knows of peers that are able to associate with the AP that it is unable to associate with. The remaining three clients deduce that port 80 is blocked since their HTTP requests have never succeeded while communication on other ports has. All of these diagnoses are made within 40 seconds. Thus even in an environment where each client is affected by a different problem, WiFiProfiler is able to quickly and correctly identify the problem(s) affecting the individual clients.

5. DISCUSSION

We now discuss some issues and concerns pertaining to WiFiProfiler and directions for future work.

5.1 Security Issues

There are several security issues pertaining to WiFiProfiler: denial-of-service (DoS) attacks launched by clients pretending to be in trouble, clients that mislead their peers by reporting bogus information, and the potential for leaking sensitive information.

We mitigate the DoS threat in our design by limiting the frequency with which a client is willing to help its peers (and incur the attendant overhead), as noted in Section 3.2.4.

Addressing the threat of bogus information is more challenging. We could alleviate this problem by basing diagnosis on information obtained from a larger number of peers. However, this procedure is susceptible to a Sybil attack [18], in which an adversary pretends to be multiple distinct nodes, using MAC address spoofing and varying its transmit power to defeat attempts to identify or locate it. In future work, we plan to investigate countermeasures based on the observation that it is hard for an adversary equipped with an off-the-shelf NIC to listen on multiple MAC addresses simultaneously.

Finally, WiFiProfiler runs the risk of leaking sensitive information. For example, revealing the one-way hash of a WEP key opens up the possibility of a dictionary attack on the key. Likewise, revealing the NIC driver version risks inviting attacks that take advantage of vulnerabilities that are specific to that driver version. In ongoing work, we are looking to determine the sensitivity of various pieces of information shared by WiFiProfiler based on whether it can be obtained via passive sniffing, and separately using zero-knowledge protocols (e.g., [19]) to share the bare minimum information needed for diagnosis.

5.2 Incentives for Participation

Since WiFiProfiler imposes a low overhead on clients, it can be left running all the time with little cost to the individual but potentially significant benefit to the community of wireless users as a whole (although we are yet to conduct a user study to quantify the benefit). By running WiFiProfiler, a client not only provides help to others but can also receive help from others when it is experiencing problems. While there is scope for cheating (e.g., a selfish node can seek help when needed but not respond when help is sought), we believe that this risk can be minimized by including WiFiProfiler as a standard system component that users are unlikely to tamper with (in much the same way that the majority of TCP stacks are conformant, despite the incentives for cheating).

5.3 Opportunities for Broader Participation

There are opportunities for broadening participation in WiFiProfiler, making the system more effective. First, although we have chosen Microsoft Windows XP as the platform for our implementation, there is little about WiFiProfiler that is OS platform-specific (other than information on the wireless driver version). Hence, information sensed by WiFiProfiler can be meaningfully shared across clients on different OSes, thereby increasing the opportunities for sharing.

Second, WiFiProfiler could be used for long-term "profiling" of wireless hotspots in cafes, airports, etc. Network health information, say indexed by the BSSID, could be shared widely, enabling a client to warn the user when he/she is trying to connect to a network that is known to be problematic.

Finally, the wireless network can be used to share information about non-wireless-related problems. For example, hosts in an apartment that has suffered a disconnection of its access link could use a wireless network to learn about the state of connectivity of hosts in nearby apartments that may be served by the same DSL or cable modem provider.

6. RELATED WORK

Of most direct relevance to our work is previous work on fault diagnosis in wireless LANs. Commercial wireless LAN vendors provide network diagnosis capability using specialized hardware sensors (sometimes incorporated within the AP itself) to monitor the wireless medium [1, 2]. [11] uses wireless clients and enhanced APs to do the sensing and feed information into a back-end server for analysis. [12] uses wired desktops machines equipped with wireless NICs for sensing.

All of these previous solutions focus on the network operator's view. Their architecture incorporates specialized sensors and/or equipment on the wired network for monitoring and diagnosis. Their focus is on problems of interest to network operators, e.g., detecting unauthorized ("rogue") APs, detecting RF holes in a wireless LAN deployment, and locating disconnected clients. While this approach is suitable for enterprises and university campuses with dedicated IT departments, it is less appropriate for hotspots settings where there may be no IT department to monitor the network. In contrast, our focus in WiFiProfiler is on the end user viewpoint, with the goal of enabling users to help themselves. As such, we only depend on cooperation among wireless clients, without any special support from the wireless or wired infrastructure. The problems of interest are those directly affecting end users (e.g., incorrect WEP key setting) rather than those that pertain to general network health (e.g., the presence of rogue APs).

There are also some specific differences between [11] and our work. The clients in [11] were assumed to be equipped with special "Native WiFi" NICs and drivers that provide raw access to 802.11 frames and also enable snooping on wireless traffic in promiscuous mode. The former is used to support the "client conduit" mechanism, in which a disconnected client beacons like an AP to obtain connectivity via a connected client. In contrast, our approach in WiFiProfiler is to avoid dependence on special hardware or driver

support; indeed, even promiscuous mode is not widely or correctly supported, so we do not depend on it. The VirtualWiFi mechanism allows peer-to-peer communication in a device- and driver-independent way.

There is much prior work on network diagnosis in wired networks, typically in wide-area networks (WANs). Active probing using tools such as ping and traceroute has been used for narrowing down the location of failures such as routing loops or lossy links in wide-area paths [20,22]. Other work has considered simultaneous active probing from multiple vantage points to leverage the diversity that this provides [27]. While such probing is useful for diagnosing wide-area network problems, it is not as useful in a wireless LAN setting, where, for instance, a disconnected client would not be able to probe beyond the local host. The location of the problem, viz. the last hop, may be evident but not the underlying cause.

End-host-based techniques have been developed to discover the local layer-2 network topology [13]. If this topology information were extended to include physical location, it could be used by WiFiProfiler, for instance, to direct users to a different physical location with better wireless connectivity.

The idea of aggregating passive observations made at end hosts to diagnose problems and improve user experience has been applied before, both in networking and non-networking contexts. In the networking context, SPAND [23] proposed pooling together information on download performance to help clients make an informed replica selection decision. NetProfiler [21] proposed pooling together network failure observations across clients to help diagnose WAN problems (e.g., determine whether a problem is ISP-specific). WiFiProfiler shares this basic idea of leveraging passive observations made at end hosts. However, the wireless setting is unique in the opportunity it provides even disconnected clients to share information with their peers. The nature of problems considered by WiFiProfiler and the level of detail of its diagnoses (e.g., incorrect WEP key) distinguish it from prior work such as [21] that focused on inferring just the location of the failure in the WAN.

Aggregation and correlation of configuration information across end hosts has also been used to diagnose non-networking problems [24, 25]. The basic approach is to apply statistical analysis on system configuration data (e.g., registry settings in Microsoft Windows) and changes thereof to infer the settings that are likely responsible for the observed problems. The focus is on black box analysis of a large volume of configuration information to identify errors, independent of their semantics. In contrast, WiFiProfiler leverages specific knowledge of the wireless networking domain to make semantically meaningful diagnoses based on a limited volume of relevant information received from peers.

7. CONCLUSION

We have presented WiFiProfiler, a system for enabling wireless clients to diagnose network problems by leveraging the cooperation of their peers within range. Based on our implementation on Windows XP, we have shown that WiFiProfiler can effectively diagnose a range of faults while imposing a low overhead on the participating clients. We believe that WiFiProfiler provides a generic framework that could be extended to diagnosing faults other than the specific ones discussed in this paper.

Acknowledgments

We thank our shepherd, Ramón Cáceres, the anonymous MobiSys 2006 reviewers, and Christian Huitema and Jawad Khaki at Microsoft for their insightful feedback.

8. REFERENCES

[1] AirTight Networks. *http://www.airtightnetworks.com/.*

[2] Aruba Networks. *http://www.arubanetworks.com/.*

[3] Device Identification Strings. *http://msdn.microsoft.com/library/en-us/DevInst_d/hh/DevInst_d/idstrings_a974863f-a410-4259-8474-b57a3e20d326.xml.asp.*

[4] Intel pro/wireless 2915abg network connection. *http://support.intel.com/support/wireless/wlan/pro2915abg/.*

[5] Intel Wireless Verification Program. *http://www.intel.com/standards/execqa/qa0104cha.htm.*

[6] Microsoft Windows 802.11 Wireless LAN Objects. *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/NetXP_r/hh/NetXP_r/217wirelessoid_bca9862e-feea-406f-b11d-ea01859bfbd3.xml.asp.*

[7] Microsoft Windows Transport Driver Interface (TDI). *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/NetXP_d/hh/NetXp_d/nettdidriv_27c4b7c0-7e21-441d-bb08-335d48204a06.xml.asp.*

[8] NSF Computer Security Policy – Prerequisites for Wireless Access. *https://www.fastlane.nsf.gov/documents/security/computer_policy.jsp#wifi*, updated November 2005.

[9] Wi-Fi Alliance. *http://www.wi-fi.org/.*

[10] Wireless/Wired Network Emulator (WiNE). *http://research.microsoft.com/users/qianz/testbed.htm.*

[11] A. Adya, P. Bahl, R. Chandra, and L. Qiu. Architecture and Techniques for Diagnosing Faults in IEEE 802.11 Infrastructure Networks. In *MOBICOM*, 2004.

[12] P. Bahl, J.Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. DAIR: A Framework for Troubleshooting Enterprise Wireless Networks Using Desktop Infrastructure. In *Hotnets-IV*, 2005.

[13] R. Black, A. Donnelly, and C. Fournet. Ethernet Topology Discovery without Network Assistance. In *ICNP*, 2004.

[14] R. Chandra. A Virtualization Architecture for Wireless Network Cards, Sep. 2005. Ph.D. Thesis, Cornell University.

[15] R. Chandra, V. Bahl, and P. Bahl. MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card. In *INFOCOM*, 2004.

[16] I. Cooper, I. Melve, and G. Tomlinson. Internet Web Replication and Caching Taxnomy. RFC 3040, IETF, Jan. 2001.

[17] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. High-Throughput Path Metric for Multi-Hop Wireless Routing. In *MOBICOM*, 2003.

[18] J. Douceur. The Sybil Attack. In *IPTPS*, March 2002.

[19] R. Fagin, M. Naor, and P. Winkler. Comparing Information Without Leaking It. *CACM*, May 1996.

[20] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet Path Diagnosis. In *SOSP*, October 2003.

[21] V. N. Padmanabhan, S. Ramabhadran, and J. Padhye. NetProfiler: Profiling Wide-Area Networks Using Peer Cooperation. In *IPTPS*, February 2005.

[22] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM ToN*, 5(5):601–615, October 1997.

[23] S. Seshan, M. Stemm, and R. H. Katz. SPAND: Shared Passive Network Performance Discovery. In *USITS*, 1997.

[24] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y. Wang. Automatic Misconfiguration Troubleshooting with PeerPressure. In *OSDI*, 2004.

[25] Y. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang. STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support. In *USENIX LISA*, 2003.

[26] Wireless Philadelphia Executive Committee. Wireless Philadelphia. *http://www.phila.gov/wireless/.*

[27] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. In *OSDI*, 2004.