

Change Is Hard: Adapting Dependency Graph Models For Unified Diagnosis in Wired/Wireless Networks

Lenin Ravindranath[†], Paramvir Bahl[‡], Ranveer Chandra[‡],
David A. Maltz[‡], Jitendra Padhye[‡], Parveen Patel[‡]

[†]MIT, [‡]Microsoft Research

ABSTRACT

Organizations world-wide are adopting wireless networks at an impressive rate, and a new industry has sprung up to provide tools to manage these networks. Unfortunately, these tools do not integrate cleanly with traditional wired network management tools, leading to unsolved problems and frustration among the IT staff. We explore the problem of unifying wireless and wired network management and show that simple merging of tools and strategies, and/or their trivial extension from one domain to another does not work. Building on previous research on network service dependency extraction, fault diagnosis, and wireless network management, we introduce *MnM*, an end-to-end network management system that unifies wired and wireless network management. MnM treats the physical location of end devices as a core component of its management strategy. It also dynamically adapts to the frequent topology changes brought about by end-node mobility. We have a prototype deployment in a large organization that shows that MnM's root-cause analysis engine out-performs systems that do not take user mobility into account when localizing faults or attributing blame.

Categories and Subject Descriptors C.4 [Performance of systems]

Wireless, reliability, availability, serviceability

General Terms

Management, performance, reliability, wireless

Keywords

Wireless, corporate networks, performance

1. INTRODUCTION

Data from IT departments of large corporations and dominant PC manufacturers show that employees prefer to use just one device (e.g., a laptop computer) for all their computing needs [17]. Consequently, many large IT departments are moving towards a future that includes a significantly reduced role for the traditional wired desktop computer [11]. They envision a future where enterprises deploy wireless networks in all corporate campus buildings, and swarms of nomadic users access corporate resources through wireless Access Point (APs). They expect users to frequently change

their point of attachment to the corporate network. In this new world, the corporate IT departments need tools to manage and diagnose both wired and wireless parts of their network.

Current enterprise network management and diagnosis systems use separate tools to diagnose wired and wireless networks. In an environment where a large number of users are nomadic, debugging application performance problems using separate tools is both difficult and frustrating [13].

For example, consider Figure 1 that shows the time required to fetch a URL, measured simultaneously from a wired desktop host and a wireless laptop as the laptop was moved between rooms every 5 minutes. Unsurprisingly, both the wired and wireless host see significant variation in the response time. Interestingly, however, the variation is sometimes seen by the wireless host only, potentially indicating problems in the wireless connectivity, and sometimes the variation is seen only in the wired host, potentially indicating congestion in the wired network. Sometimes the variation is seen in both, potentially indicating congestion in a server involved in providing the requested URL.

A natural question to ask is: why not diagnose performance problems by using the existing wireless and wired network diagnosis systems separately?

The answer is that a diagnosis system that looks at only the wired network or the wireless network is likely to misinterpret some of the spikes in the response time and blame the wrong network component. In this paper, we show that quality of diagnosis is better when both wired and wireless aspects of the enterprise networks are analyzed jointly.

Three main features distinguish our approach from the recent research on enterprise network diagnosis systems:

Changing network topology: Many recently proposed network fault diagnosis systems such as Sherlock [4] and SMARTS [22] implicitly assume that the fundamental structure of the network is either static or changes slowly. This assumption allows these systems to build Inference Graphs [4] and codebooks [22] to pinpoint the cause of performance problems seen by the users. However, these approaches cannot be used without substantial modifications in an environment where clients frequently change their point of attachment to the corporate network.

Joint Consideration of Wired and Wireless Networks: To diagnose end-to-end performance of networked applications across wired and wireless networks requires re-thinking some core aspects of fault diagnosis. For example, geographic location must become a first class object in the analysis for determining if a problem is in the backhaul network, the wireless link, or the servers in the data center.

Absence of Fixed Observers: Since many problems in wireless networks are location specific, existing wireless network monitor-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WREN'09, August 21, 2009, Barcelona, Spain.

Copyright 2009 ACM 978-1-60558-443-0/09/08 ...\$5.00.

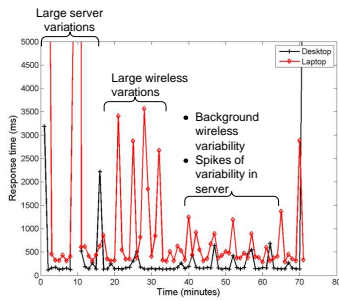


Figure 1: Time to fetch a URL as measured simultaneously from a wired desktop host and a wireless laptop. The laptop was moved between rooms every 5 minutes.

ing systems rely on fixed desktops [8] or specialized monitoring hardware [3, 10]. However, in a network consisting primarily of nomadic users, systems like DAIR [8] are impractical, while systems like Jigsaw [10] and Wit [18] are expensive to deploy.

We have developed an end-to-end network diagnosis system, called MnM, that successfully diagnoses performance of networked services and applications running on nomadic hosts. MnM builds on recent research on network service dependency extraction [4], fault diagnosis, and wireless network monitoring. It treats the physical location of end devices as a core component of its diagnosis strategy. It also dynamically adapts to the frequent topology changes brought about by end-node movement. Our system is implemented entirely in user-level software, and it does not require any specialized monitoring hardware. We have deployed the MnM system on a segment of our organization’s network. Over a period of two weeks, we monitored 27 users and 10 servers. We detected and correctly diagnosed a variety of performance issues, including poor Wi-Fi coverage, congestion in wired networks, and misconfigured DNS entries. As we shall show later in the paper, at least 140 performance problems would have been mis-diagnosed had we not taken an integrated, holistic view of wired and wireless networks.

MnM extends the state-of-art in enterprise network management by making two important contributions:

1. We identify issues that an enterprise network management system must consider when the end-hosts are nomadic. We show that recently developed systems are not able to cope with these issues. We quantify mistaken diagnoses that occur in systems that do not compensate for user nomadicity, and we argue that location must be treated as a core component in future enterprise network management systems.
2. We present an enterprise network management system that unifies wired and wireless network management, and handles nomadic users. It is easy to deploy, as it requires no special fixed infrastructure for wireless monitoring and automatically initializes its location system. We evaluate its accuracy through both controlled experiments and a 2-week field study.

2. RELATED WORK

There is a significant amount of prior work in enterprise network management. However, it has either focused on managing wired networks or wireless networks, not both simultaneously. The closest thing to unified management tools are systems that let network managers view the wired and wireless networks simultaneously [13].

Wireless Network Management: Adya et. al. [1] built one of the first enterprise wireless network management systems. Their

system is similar to ours in that they focus on performance problems faced by Wi-Fi enabled mobile clients. They detect problems by analyzing link data collected by monitoring agents residing on clients and wireless APs. Unlike our system, their techniques miss out on problems that a mobile client may have because of a performance issue in the wired part of the network.

The DAIR system [8] also detects performance problems faced by users of Wi-Fi networks. DAIR uses corporate desktop computers to monitor the airwaves and, like MnM, location-awareness is a core component of its management strategy. Fundamentally, DAIR relies on the existence of fixed desktop devices to monitor performance of wireless link. In contrast, MnM assumes a world where every client is mobile. In such an environment, monitoring must be done by mobile clients themselves. This presents several unique challenges, such as bootstrapping, which systems like DAIR cannot handle. Furthermore, DAIR requires the monitoring devices to sniff packets in promiscuous mode, which may not always be possible on battery constrained mobile clients.

Jigsaw [10] and WIT [18] are Wi-Fi monitoring systems that combine the data from multiple monitors to generate a comprehensive view of network events. Jigsaw uses dedicated, custom-built, multi-radio monitoring nodes and provides a detailed view of low-level network effects such as interference. WIT is able to analyze and detect MAC-level mis-behavior. While useful in investigating why individual locations have poor performance, these tools are not designed for diagnosing end-to-end networked services in a corporate environment.

Commercial systems [2, 3] are available for managing wireless networks, but they do not detect performance issues due to problems in the wired part of the network. Furthermore, systems like DAIR, Jigsaw, WIT, Airtight, etc. do not have visibility into application-level performance problems, whereas, as we will show, MnM does.

Wired Network Management: The Sherlock system [4] manages networked services in enterprise networks by extracting inference graphs and then using these to diagnose performance problems. Software agents running on desktop machines determine the set of services the host depends on and a centralized inference engine captures the dependencies between the components of the IT infrastructure by merging the views of each client. Sherlock then diagnoses faults by running an inference algorithm on the inference graphs. Sherlock makes a fundamental assumption that dependencies are static or, at most, change slowly. This is not true for applications running on devices used by nomadic users. As we show in Section 3, systems like Sherlock perform poorly when dependencies are dynamic and fast changing. Furthermore, such systems cannot be trivially extended to handle nomadic clients.

Other network management systems like Shrink [14] and SCORE [15] have made seminal contributions in diagnosing faults in wide-area networks, but unfortunately, they cannot be used easily for managing nomadic users. Similarly, sophisticated commercial products such as SMARTS [22], OpenView [19], and Tivoli [23] provide powerful tools for managing enterprise wired networks, but fall short when extended to manage mobile clients and Wi-Fi users.

Finally, we note that a longer version of this paper is available as a technical report [5].

3. FORMULATING THE PROBLEM

Figure 2 illustrates an enterprise network of the future. Users located on the corporate campus access the enterprise data center servers via APs deployed in campus buildings, and these users move around frequently. Some users may work remotely, and connect to the corporate network via VPN. In this paper we focus pri-

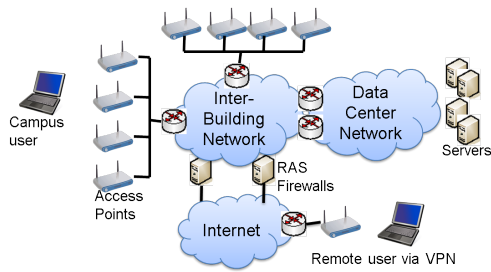


Figure 2: Example of the typical enterprise network of the future. Most users access corporate resources from laptop computers connected to wireless networks or from remote locations via VPNs over the Internet.

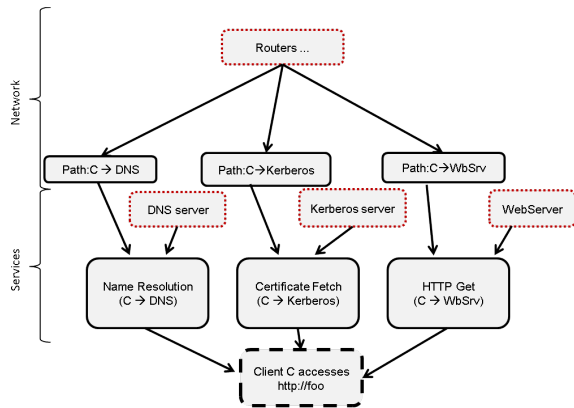


Figure 3: Example Inference Graph. The response time measured for fetching `http://foo` (dashed outline) is affected by the root causes (shown with dotted outlines).

marily on nomadic users who change location but conduct most of their work when stationary. Some other papers refer to these as mobile users, and we use the terms interchangeably. We believe MnM is applicable to users in constant motion, but it is out of the scope of this paper.

3.1 Fault Diagnosis using Inference Graphs

Prior work in fields as diverse as network management [15, 4, 24] and medical diagnosis has shown the advantages of using an *Inference Graph* to diagnose faults in the presence of noisy observations. However, we have found that nomadic users violate some of the important assumptions on which these systems are based, and, consequently, these systems perform poorly when used to diagnose the problems experienced by nomadic devices.

MnM attempts to leverage the expressiveness of inference graphs while fixing the problems that prevent them from use with nomadic systems. We begin with a brief overview of inference graphs. For more details, see [4, 15]. Then, in Section 3.2, we describe the problems caused by nomadic users. Section 4 describes our techniques for applying Inference Graphs to nomadic hosts.

The Inference Graph:

We use the model proposed in Sherlock [4]. An Inference Graph consists of directed edges and three types of nodes: *root causes*, *meta-nodes*, and *observations*. The graph encodes how root causes, which represent components or services that can be faulty, affect the observation nodes, which represent aspects of the system that can be measured. Meta-nodes are the glue that ties together the root

causes involved in particular services or network paths.

Figure 3 illustrates an example Inference Graph for a single client *C* using a web server. In this figure, the response time the client *C* observes when fetching a web page will be affected by the health of the DNS service, the Kerberos service, and the web server itself, since to successfully fetch the web page, *C* must first use DNS to convert the name of the website to an IP address, then fetch certificates to access the website, and finally retrieve the content from the website. The health of these services, in turn, is affected by the health of the servers that implement the service and the ability of the client *C* to successfully reach the servers over the network. The health of each network path is affected by the routers on the path.

Nodes in the Inference Graph are conceptually in one of two states: *up* or *down*. Root causes that are operating normally and observations indicating normal performance are *up*. Nodes causing or indicating poor performance are *down*, even if they have not failed completely but are merely slow in returning answers.

While our example Inference Graph has only a single client and a single observation of a single application, a system-wide Inference Graph is built by combining the graphs for each client application and service. These graphs share the same root cause nodes, but have different observation and service nodes for the combination of each client and application.

The Inference Algorithm: Given the inference graph and the state of the observation nodes, an inference algorithm can infer which root causes are most likely to have failed. This is especially useful in the cases where root causes can not be directly observed [4, 15].

Many inference algorithms have been developed, but the goal of each is the same: given a set of observations of system performance, good and bad, determine a set of root causes whose failure would best explain that pattern of observations. To cope with the uncertainty in the real world, MnM uses probabilistic inference. Specifically, every root cause has a *prior* probability — that is, the fraction of time the root cause is typically *down*. The inference algorithm takes these priors into account when computing which root causes are most likely to be down. The algorithm used in this paper is the same as that used by Sherlock [4].

3.2 Impact of Nomadic Users

One could ask the question, would a trivial combination of wireless monitoring methods [8, 10, 18] and wired monitoring methods [4] be able to diagnose the problems experience by nomadic users? We answer this question by making the following four observations:

3.2.1 Dynamic Inference Graph

A defining characteristic of nomadic users is that they move, changing their location and their point- and method-of-attachment to the network up to several times during a day. As a result, Inference Graphs for nomadic users change frequently and significantly. For example, when a nomadic user connects to the enterprise network via a wireless network, the AP changes as she moves from one location to another. Worse yet, the servers in other parts of the Inference Graph change as well, as the DNS and Kerberos servers that a host uses may change whenever the subnet changes and a new IP address is issued from the DHCP server. Figure 4 illustrates how the Inference Graph for a particular application changed compared to the inference graph of Figure 3 as client *C*'s point of attachment changed from a wired network to a wireless network at a different location.

Such dynamism inside the network is a problem for current inference systems. Prior work has proposed techniques for learning

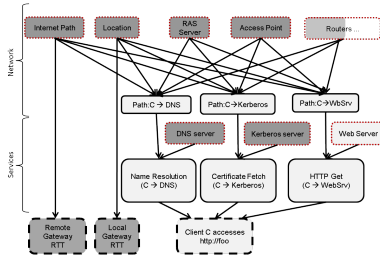


Figure 4: Example Inference Graph when a nomadic user connects to the corporate network using a 802.11 wireless network. To ease comparison with Figure 3, nodes affected by mobility are shown with dark backgrounds.

the Inference Graph via monitoring the packets that hosts send and receive [20, 4]. However, these learning algorithms assume that the Inference Graph remains unchanged long enough to be learned. For example, Sherlock reports that it takes several hours for the learned Inference Graph to stabilize. Other researchers have shown that users change location frequently [7, 16], so for most cases the Sherlock algorithm would not be able to learn the Inference Graph before it changed.

MnM’s approach is to separate the Inference Graph into the portions which are relatively static and can be learned (e.g., dependencies among servers in the wired data center) and the portions that change frequently. We use the *Domain Experts* described in Section 4.1.4 to compute these portions as needed.

3.2.2 Importance of Location

Researchers have previously shown that the physical location of a mobile device has a direct impact on the performance of the applications it is running [8, 12]. For example, two users running the same application, connected to the network via the same AP, may experience different performance — one might see short response times from a web server while the other sees long response times, all due to variations in the RF environment around their physical location. If location is not incorporated into the Inference Graph, then the inference algorithm will blame the wrong root cause as it tries to explain the performance problems seen by the host experiencing longer delays. Consequently, MnM treats physical location as a core component of its end-to-end network diagnosis system.

3.2.3 Dynamics of Monitoring and its Limitations

State of the art Wi-Fi network management and diagnosis systems such as Jigsaw [10], WIT [18], and DAIR [8] rely on the existence of fixed infrastructure, either in the form of specialized hardware or always-available desktop computers, to monitor the RF environment. Specialized hardware is expensive to deploy and maintain. Furthermore, the general trend in large IT departments is to replace desktop computers with laptops. Without the support of ‘static’ infrastructure, determining the physical location of a client becomes difficult. Further, the laptops of ordinary users cannot be used to take detailed measurements of their wireless environment because that would require running their Wi-Fi interface cards in promiscuous mode. Promiscuous mode prevents the cards from entering their power save states and thus places an unacceptable strain on the laptops’ batteries and increases the barrier to deployment.

Consequently, end-to-end network diagnosis systems must use light-weight self-configuring location determination techniques that do not depend on support from existing infrastructure.

3.2.4 Difficulties Identifying Root Causes

One might argue that running existing wireless and wired diagnostic tools separately can diagnose application-level performance problems for nomadic users. However, low level wireless performance metrics such as signal strength and packet loss rates have a complex relationship to the performance of higher layers [9]. One cannot simply assign thresholds to translate link-layer measurements into application-level throughputs. For example, using the data collected from our 2-week study presented in Section 6.2, we see that there is no significant correlation between the AP signal strength seen by a client and the end-to-end performance it achieves. Further, there are some dependencies in the wired network that are specific to wireless machines, e.g. APs, the wireless gateway and the wireless authentication servers. It is hard to measure their impact on application performance without unifying wired and wireless performance diagnosis.

4. ARCHITECTURE

A system that jointly manages wired and wireless networks needs three unique capabilities: an ability to determine the locations of mobile clients without relying on fixed monitoring resources, an ability to frequently update the inference graph and an ability to determine the performance of different components of the network. In addition to end-to-end observations, MnM also measures the performance of some individual network components, such as the capacity of the wireless link, and includes these into its inference algorithm when diagnosing application-level performance problems. In this section we describe the architecture of MnM and show how these capabilities are incorporated within it.

Figure 5 illustrates MnM’s architecture. MnM consists of two main components: the *MnM Agent* that runs on each mobile device in the network, and the *MnM Inference Engine* that accepts data from these agents. The Inference Engine analyzes data from agents to determine the root cause of performance problems, and raises alerts to the network operator. In addition, we have *domain experts*, whose functionality is split between the agent and the inference engine. The role of domain experts is to modify the inference graph in some special cases. Below we provide more details on each of these components.

Comment about Privacy: This paper focuses on enterprise networks. In such networks the IT department has the authority to require every user to run monitoring software. Therefore, the issues of user consent and privacy are out of scope.

4.1 The MnM Agent

The MnM agent is a light-weight application that runs on users’ laptops. It includes *Monitors* that gather information about the system, user activity and network connectivity. This data is processed by *Domain Experts* that encapsulate the special logic required to deal with different problem domains. The *Domain Experts* generate data for the inference graph and performance observations. The agent sends all this data to the MnM Inference Engine over a transport called the *Trickle Integrator* that is designed to cope with intermittent and variable connectivity. The MnM Agent does not require any driver modifications in the clients and hence is easy to deploy.

4.1.1 (Agent) Controller

The Controller is the agent’s lightweight workflow engine. It provides a publisher-subscriber service to moderate the interactions between *Monitors*, *Domain Experts*, and the *Trickle Integrator*. All messages between the components in MnM take the form of tuples: a list of fields and their values. The experts and monitors register

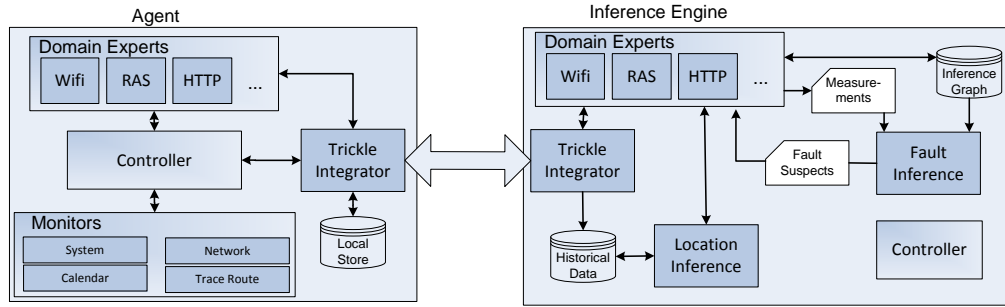


Figure 5: Architecture

triggers with the Controller. Whenever the Controller processes a message matching a trigger, it invokes the associated callback with the message as an argument. The Controller itself generates messages to mark important events, such as agent startup and expiration of a periodic timer. Monitors and Domain Experts are the “plugable” components. They can be developed independently of one another — only the format of fields and values must be agreed on to ensure proper intra-agent communication.

The agent generates a START message on startup. Then it generates a PERIODIC TIMER message every polling interval, which triggers the monitors to generate messages encapsulating their measurements. In addition to the messages generated by the agent, the monitors also register for system-wide events such as network address change and wireless hand-off event.

4.1.2 Trickle Integrator

We designed MnM to handle situations when mobile hosts are unable to reach the Inference Engine. Specifically, MnM includes a module inspired by Coda [21] for dealing with measured data during weakly connected and disconnected operation. Every tuple of data created by a Domain Expert or a Monitor is passed to the Controller, and from there it is placed in a local store. Data from the local store is then pushed to the Inference Engine whenever the client has connectivity. The Trickle Integrator also rate-limits the messages sent by the client to the server, and, if a backlog develops, new messages are given priority over old ones.

4.1.3 Monitors

As mentioned in Section 4.1.1 monitors can be developed independently and dynamically added to the MnM Agent on an as-needed basis. In our current implementation, the MnM Agent contains four monitors.

System Monitor: This monitor reports various system properties from the current polling interval. It reports information such as, whether the system’s battery is being charged and whether the system is connected to a wired network (e.g. Ethernet). It also reports whether a user is currently active on the system (the system is considered idle if there is no user input for n minutes, where n is currently set to 2).

Calendar Monitor: The Calendar Monitor tracks the time and location of accepted meetings from the users enterprise calendar (e.g., Exchange or Lotus server). This information is used to bootstrap the location engine, as we shall describe in Section 4.2.1.

Network Monitor: The Network Monitor reports information about network connectivity. The monitor is triggered by the network change related events from the system, such as network address change. It reports information about active network interfaces including: IP and MAC addresses, gateways, DNS and default gate-

way servers, and ping times to the first hop router. If the Network Monitor detects that the user is connected to the network via a wireless interface, it periodically collects additional information such as the AP the interface is associated with, other APs it can detect and the signal strengths of their beacons. The monitor also generates messages that are specific to the wireless interface. For example, if the wireless client is handed off from one AP to another, it generates a HANDOFF message.

Trace Route Monitor: This monitor uses *traceroute* to discover the network path between the client and the other machines to which it is sending packets.

The total amount of data pushed to the Inference Engine for each observation is less than 1K bytes and hence pushing data to server takes very negligible amount of the users network bandwidth. This issue is examined in detail in Section 5.

4.1.4 Domain Experts

In Sherlock, the authors assume that the Inference Graph is stable, and hence it is learnable via black-box techniques. However, mobility causes changes to the Inference Graph, and even though the changes may be regular and sometimes predictable, they are generally too rapid for black-box techniques to learn the graph. To handle this, we define the concept of a *Domain Expert* - a module that is responsible for making the appropriate changes to the Inference Graph when triggered by a host changing its connection point or other dependencies.

A typical Domain Expert has code both on the host, as part of the Agent, and on the Inference Engine. Domain Experts respond to triggers such as change in IP address, or AP handoff event. Upon such changes, the Domain Expert on the client notifies the Domain Expert on the Inference Engine of the triggering event. The Domain Expert on the Inference Engine then updates the Inference Graph appropriately. For example, when an AP Handoff event occurs, the WiFi Domain Expert on the agent notifies its counterpart on the Inference Engine. The Inference Engine then updates the Inference Graph to account for the change in topology.

WiFi Expert: The WiFi Expert is responsible for managing the details of how wireless connectivity affects the performance of applications running on a mobile node. It does this by adding new root cause and observation nodes to the Inference Graph in a particular pattern, which we call a *graph gadget*. Based on reports from the monitors on the client, the expert fills in the correct AP and location information. Figure 6 illustrates the new Inference Graph generated with the help of a WiFi Expert.

Most importantly, for every client whose location can be determined, the WiFi Expert adds a new root cause node that represents the location. There is one location root cause node for each location known to MnM — all the clients predicted to be in that location

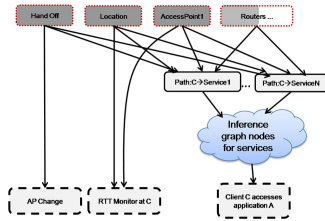


Figure 6: “Gadget” added to the Inference Graph of mobile hosts by the WiFi Expert. New elements shown in grey or with darker lines.

share that node. Associated with location is the a priori probability that the location causes performance problems. MnM determines locations and computes priors as described in the subsections that follow. The expert connects the location root cause to an observation node whose value is tied to measurements of the RTT of pings between the client and the current AP. The RTT provides a degree of direct estimation of current wireless channel quality, while location priors provide historical information about the wireless channel quality at this location.

HTTP Expert: The HTTP Expert monitors the response time of web servers when URLs are fetched, and reports these to the Inference Engine. The Inference Graph uses these as observations about the application’s health. For testing purposes, our HTTP Expert also includes a URL polling robot that can be ordered to fetch particular URLs during experiments.

Network Expert: The Network Expert computes the network topology-related dynamic part of the inference graph whenever a network change event occurs on the client. It is responsible for filling in two types of information. First, it computes network path to network services by using topology discovery techniques, such as *traceroute*. Second, it detects changes in location-dependent network services, such as the DNS and Kerberos servers. The Network Expert counterpart on the Inference Engine updates this information in the inference graph.

Service Expert: The Service Expert is a special expert that runs only on the Inference Engine, and has no client counterpart. The Service Expert is responsible for building a static, service-level dependency graph for all networked applications. A service is identified by the service name and the server that is providing that service. For example, a website is identified by its URL and the web server hosting it. The Service Expert gets the data needed to construct the dependency graph from a variety of sources. For example, systems like [10, 4] use temporal correlation in packet traces to infer dependencies. Some information, such as topology of the data center, can be extracted from network configuration files. The static dependency graph is combined with dynamic information from other domain experts, such as the Network Expert and the WiFi Expert, to build an inference graph.

Comment: We note that the Domain Expert architecture is a general technique that will be useful for handling other types of domains where the Inference Graph changes faster than it can be learned. An example of this is peer-to-peer systems where the servers being invoked change depending on the query being made.

4.2 The MnM Inference Engine

The MnM Inference Engine is responsible for monitoring the health of the mobile device and the applications running on it. The engine stores and analyzes the data sent to it from each of the MnM Agents. Using this information and the service-level dependency

graph, it generates and updates an Inference Graph that reflects where the mobile clients are located and how they are connected to the network. It uses the Inference Graph to generate a list of probable causes whenever it identifies performance problems, and subsequently raises alerts.

4.2.1 Location Inference

The physical location of a wireless client may have a strong impact on its network performance [8]. Thus, management tools designed for wireless networks must include an integrated location estimation system.

A number of techniques [6, 8, 25] have been proposed for estimating the location of clients in a Wi-Fi network. These techniques offer a wide range of tradeoff between accuracy, measurement overhead, required infrastructure support and the need for detailed profiling of the physical environment. For the purpose of network management, it is generally sufficient to determine the client location at the granularity of one office. However, unlike the scenario described previously [8], we can not rely the presence of densely deployed, fixed desktop to serve as monitors. Hence, we have built a location system using the technique described in [25].

Location Profiles: Our system stores a *profile* for each location of interest. To allow for easy interpretation, we define location in terms of office numbers, rather than (x,y,z) coordinates. The profile for each office consists of a list of APs (i.e. their BSSIDs) that are visible from that location along with the distribution of observed signal strength of each AP. We assume a Gaussian distribution and characterize it with its mean and variance. These profiles are generated automatically, as we will explain later in this section.

Determining Client Location: As part of its observations (e.g., measuring URL response times), the Wi-Fi Monitor running on each client sends the inference engine the list of APs seen by the client, along with their signal strengths. Using the stored profiles, and the Bayesian inference technique described in [25], the location inference module determines the most likely location of the client, and persists it with the observation data in a history database. The median error for computed location is about 5 meters (one or two offices). We will present a detailed evaluation of the accuracy of our location system in Section 6.1.

Automatic Generation of Profiles: To reduce the effort required to roll out MnM, we automatically generate location profiles by using the information provided by the Calendar Monitor running on each client.

Most corporate environments provide a calendar service that employees use to schedule meetings with each other. For each meeting, the calendar records the identities of invited attendees and the location of the meeting (e.g., a conference room or another employee’s office). MnM generates profiles for rooms that appear as meeting locations using the Wi-Fi observations reported by the employees’ laptops during the meeting time. To reduce the amount of erroneous information included in the location profile, MnM verifies both that there is activity on the user’s laptop during the meeting (i.e., the user has the laptop with them at the meeting) and that Wi-Fi observations are roughly consistent with those of other attendees (i.e., the user has actually gone to the meeting, rather than remaining in their office).

To generate a profile for a user’s office, MnM looks for Wi-Fi observations made during times when the user has no meeting scheduled. Many people plug their laptops into wired Ethernet and/or wall power when they are in their offices, and MnM looks for these clues when selecting observations to use in constructing the office profile.

We also note that in an environment where APs are deployed

densely, it may be sufficient to characterize the location of the client simply by the AP that the client is associated with. This method requires no profiling, but is subject to inaccuracies, since clients sometimes associate with APs that are far away. We evaluate the usage of APs as a stand-in for location in Section 6.2.

4.2.2 Fault Inference

The fault inference module of MnM is responsible for taking the data produced by the agents in the system and determining which root causes are responsible for any problems. The resulting list of *fault suspects* is given to the network managers for reporting and resolution.

The module consists of two components: the computation of location priors, which is invoked once a day, and the inference module, which is invoked every 3 minutes or whenever there is a significant change in the observations being reported by clients.

Once invoked, the inference module updates the Inference Graph, computes the state of the observation nodes, and then runs the inference algorithm to determine a list of fault suspects.

Computing Priors for Locations: Instead of detailed current measurements, MnM relies on analysis of past experience to compute a *prior probability of failure* for each location known to the system. These priors are then used by the inference algorithm when determining the root causes responsible for bad observations. Priors can be cheaply computed from information already available in the historical database present on the Inference Engine, and, as shown in our evaluation, they largely eliminate the need for detailed current measurements when diagnosing faults.

Once a day, the Inference Engine computes priors for each location l by retrieving from its history database all response time observations from locations within 6.7 meters of l — 6.7 meters is the median error of our location inference system, so observations labeled as being from those locations could have come from l . MnM then computes the fraction of those response times that are *down* and uses this fraction as the prior probability that l is faulty.

This simplistic approach implicitly assumes that all *down* observations are due solely to the location alone — discounting the effect of the servers and other components that might affect the observations. However, since our approach averages over the response times of many servers contacted from location l over long periods of time, any systematic bias is most likely due to the location. More complicated Bayesian estimation techniques could be used, but our evaluation shows they are unnecessary in our environment.

Computing the Inference Graph: The Inference Engine controller orchestrates the construction of the Inference Graph by the various Domain Experts through a publish-subscribe system. The basic inference graph is generated by the service expert. Each Domain Expert subscribes to be notified whenever nodes or edges with specified properties are added or deleted from the graph. Upon receiving such notification, the Domain Expert makes its own alterations to graph. This process repeats until no further changes are made to the graph, at which point the graph is ready to use for inference.

The process of altering the Inference Graph is triggered whenever a monitor or expert on a client detects a change. For example, when the HTTP Expert on client C observes the client accessing a web page `http://foo.com` with response time rt , the HTTP Expert on the Inference Engine will create a new observation node for C accessing `foo.com` if it does not already exist in the Inference Graph. The addition of this observation node causes the Service Dependency Expert to add nodes and edges reflecting the servers involved in accessing `foo.com` (e.g., DNS, Kerberos, and `foo.com` itself). The addition of these nodes causes the Network Expert to fill in additional root causes and edges for the network paths from

C to those servers, the DNS servers currently being used by C , etc. **Computing Observations:** Before invoking the inference algorithm, the inference module scans all observation nodes in the Inference Graph and invokes the Domain Expert that created the node. The Domain Expert is expected to determine whether the observation node is *up* or *down*, and typically does so by retrieving recent measurements for that node and determining if they are normal or abnormal. For example, the observation node for a HTTP response time returns *down* if the response time is greater than a threshold based on the normal distribution of response times for that web-server, and *up* otherwise.

Diagnosing Faults: Given an Inference Graph, prior probabilities for locations, and the *up* and *down* status of the observations, MnM uses the Ferret inference algorithm described in [4] to compute the root causes that are most likely responsible for the *down* observations. These root causes are returned as the fault suspect list.

5. IMPLEMENTATION

We have implemented the MnM system shown in Figure 5. The Agent Controller is implemented as a daemon (service) process. The Domain Experts and Monitors are implemented as loadable modules that are loaded and invoked by the Controller. The Inference Engine is implemented as a centralized service. The Inference Engine uses a database to store historical data but keeps the Inference Graph and the current observations in memory for fast access. The Inference Engine can run inferences on live incoming data or on the historical data. Our Inference Engine integrates with the enterprise network management system deployed in our organization and generates alerts through its console whenever it diagnoses a performance problem.

Scalability is a frequent concern with centralized systems. We evaluated two aspects of scalability of our design — the CPU and network overhead on the client machines and the performance of the Inference Engine as the number of nodes increases.

The CPU overhead of running the MnM agent on client machines is negligible. Each client machine, on average, generates less than 1000 bytes per minute (0.13 Kbps), which is also negligible.

The traffic from all clients aggregates at the central Inference Engine. Even with 10,000 active clients, the Inference Engine receives less than 1.5 Mbps of traffic. The CPU overhead of our Inference Engine is also small. The authors of Sherlock [4] show that the overhead of inference scales linearly as the number of nodes increases. We observed similar behavior with our system. On a machine with 3GB of RAM and four 3.2 GHz CPUs, our inference algorithm processes an Inference Graph containing more than 100,000 nodes in less than 5 seconds.

6. EVALUATION

We evaluated MnM in a large enterprise network, performing two types of experiments. We first conducted controlled experiments with intentionally injected faults to evaluate the accuracy of our system when diagnosing the faults that might occur in an enterprise network with all nomadic users. Once we had confidence the system was performing correctly, we then ran the system for two weeks on the machines of 27 volunteers, creating a dataset that we use to analyze the sensitivity of the system and the types of problems found in the network. All the experiments presented in this section were conducted on a live production enterprise network with thousands of computers, so the background traffic is entirely realistic.

We installed MnM on 42 computers: 27 user laptops, 5 test laptops, and 10 servers. These computers were used normally by their

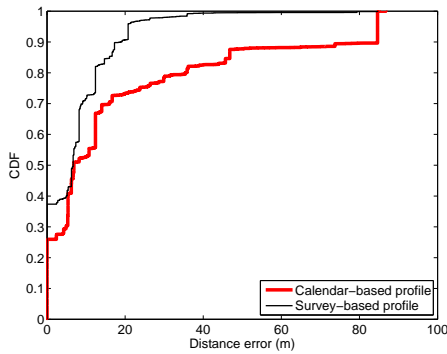


Figure 7: CDF of error in predicted location, measured in meters, over 22,000 observations among 96 locations over a period of two weeks.

owners in their daily activities. The users represent a variety of corporate users, including programmers, managers and researchers. Because we are not part of the corporate IT department and had to recruit volunteers, we did not monitor the actual web sites that users visited out of privacy concerns. Instead, we added an agent to their machines that fetched content from a set of five internal production web sites every three minutes.

6.1 Location Inference Evaluation

As described in Section 4.2.1, the location estimation module infers a location for every record submitted to the Inference Engine, as long as the submitted record contains a wireless fingerprint. Most offices on our floor are approximately 9 square meters (3x3) in size. The conference rooms are much larger. The size of the floor is 101 meters by 86 meters and it has approximately 200 offices.

During the two week study, the location estimation module inferred locations for over 77,000 records. Of these 77,000 records, 22,000 were manually labeled by the volunteers with their true location (i.e. the office or the conference room the machine was actually in at that time).

Figure 7 shows the CDF of the distance error between the geometric center of each record’s true location and its inferred location, using two different sets of profiles. When using profiles generated automatically by our calendar heuristics, as described in Section 4.2.1, the inferred location matches the true location exactly 37% of the time. The median difference is 6.7 meters, which translates to an error of about two offices. We believe that this accuracy is sufficient for our purposes.

The calendar-based profiles will contain some errors as machines are not always located where the calendar heuristics guess they will be. To estimate the loss in accuracy caused by these mistakes, we conducted a survey of our building by manually placing a laptop in roughly every other office for a fixed period of time and gathering the signal strengths of beacons broadcast by the various APs. We computed profiles from these observations, and then computed the distance error of the records when locations were inferred using these survey-based profiles.

The error is less when using survey-based profiles as all observations used to generate the profile are labeled with the correct location. The difference between the two curves measures the loss of accuracy due to mistakes made guessing the machine’s location from the users’ calendar. Interestingly, the median error with our automatically generated calendar-based profiles is roughly the same as the median error with survey-based profiles. This suggests that calendar-based profiling works well for a large number of lo-

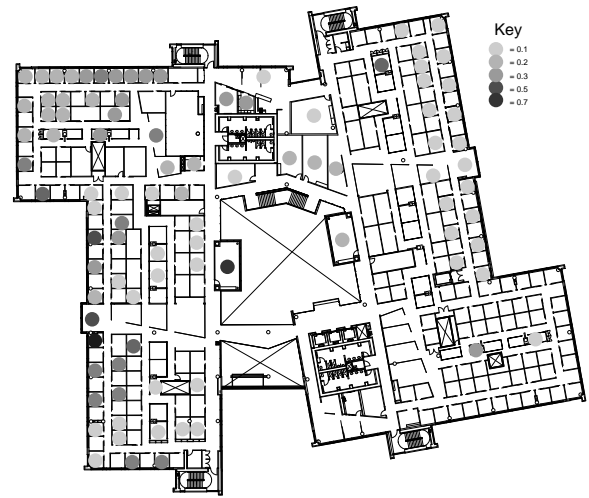


Figure 8: Location priors in our building.

cations and records, although more observations labeled with calendar data would be needed to match the accuracy of survey-based profiles across all locations.

6.2 Field Study

In this section we describe the results of our 2-week study of real users using MnM.

Location Priors: Figure 8 shows the prior probability that fetching a URL will take unacceptably long from an office, where the darker the circle the greater the probability of that location being a problem. There is clear variation in the priors over the building, indicating that location does have a strong effect on the ability of nomadic users to access the company’s servers. The middle-left of the building is particularly bad, the middle-top offices are slightly better, and the conference rooms in the middle and the offices to the right are, for the most part, the best. Priors vary from 0.01 in the best areas to almost 0.7 in the worst.

Fault Diagnosis: The Inference Engine was run every 10 minutes during the 2-week study: a total of 1530 times. It diagnosed a fault during 434 of these runs. Unsurprisingly, most faults were concentrated during the working hours when more laptops are present and network and server usage is highest. We have confidence in the accuracy of the faults diagnosed by the system based on its performance in the controlled experiments.

Figure 9 shows the number of faults of each type that were diagnosed during the study. The bar for “With location priors” represents the results of MnM as we intend it to be used, with location priors taken into account by the inference algorithm. As there can be more than one fault diagnosed during a single run of the inference algorithm, the number of faults discovered totals to more than 434. The most common source of problems was the laptops themselves (“machines”), followed by a server in the data center. Of the 310 faults attributed to a server in the data center, 114 were to a server well-known to have problems with intermittent overloads. MnM also correctly identified DNS misconfiguration on one of the servers. The server’s primary DNS was configured to 127.0.0.1 while it was not running a DNS server. This was causing delay in DNS lookup, which ultimately impacted total URL fetch times.

Importance of location: Location was to blame for 144 problems – 10% of the total – indicating that it is a significant source of errors. During 31 10-minute intervals, all problems seen by users

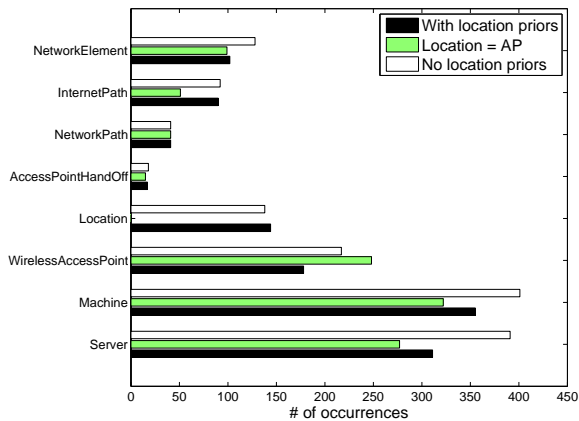


Figure 9: Number of faults diagnosed during 2-week study, broken out by type of fault and location information used.

were due solely to the users’ location. Based on this data, we expect that MnM would be at least 10% more accurate in its fault diagnoses than a system that does not consider location.

To predict the performance of a system that does not include location but does model wireless components like access points, we configured MnM to use the AP with which each laptop was associated as the “location” of that laptop. As expected, the number of problems attributed to the access points increases. Interestingly, the number of problems attributed to the servers goes down — without the ability to blame specific locations, the system blames too many problems on wireless issues.

Importance of location priors: To evaluate the effect of location priors on fault diagnosis we ran MnM with locations, but assigning all locations the same prior (labeled “no location priors” in the figure). The system correctly diagnoses location faults as often as MnM does when using accurate priors, but it also blames the machines and servers more than it should. Many locations have only a single machine reporting observations, as they are private offices, and without the historical perspective provided by the prior the system does not have enough independent observations to confidently distinguish between a problem with the location, the user’s laptop, or the remote server.

6.3 Controlled Experiments

To evaluate the accuracy of our system in diagnosing problems that arise in client mobility scenarios, we conducted controlled experiments where we deliberately impaired parts of the network to create faults. These experiments were conducted on our production corporate network, so there was normal corporate background traffic and some naturally occurring failures during the experiments. However, the results here give a lower bound on the accuracy of MnM.

Methodology: For the following experiments, all 42 machines polled four enterprise websites once every 60 seconds. The MnM Agents ran the application experts and monitors described in Section 4.1.4.

Each experiment ran for at least 60 minutes, with the specified fault injected at the beginning of the experiment. The Inference Engine ran once every minute, producing at least 60 set of fault suspects for each experiment. For these experiments, we required that the Inference Engine return the root cause representing the injected fault with rank one or two before counting it as a successful

diagnosis. This is because network managers are unwilling to look beyond the top few root causes. Table 1 presents a summary of the results.

Problems Due to Bad Location: To measure the accuracy of our Inference Engine in identifying bad locations, we created the following experimental setup. We place two laptops in a location with poor performance characteristics due to its long distance from an AP, and force the laptops to associate with that AP. Three other laptops, placed closer to the AP, were also associated with the AP. The experiment tests whether MnM can correctly determine that multiple performance faults observed for clients associated with the same AP do not necessarily imply that the AP is at fault. Instead, MnM must determine the impact of a client’s location on its performance. The first row of Table 1 presents a summary of the results. We made two observations during this experiment:

First, when the location module accurately infers the locations of the two laptops seeing poor performance, the Inference Engine correctly identified the location as the highest ranked root cause.

Second, when the location module does not report the two poorly-performing laptops being at the same location, the Inference Engine reports the location as the second-highest ranked root cause. The wireless access point was reported as the highest ranked root cause, as it was a shared dependency between the two laptops in the Inference Graph, whereas each laptop was (incorrectly) connected to a different location root cause.

Problems Due To Bad Access Point: To determine the accuracy of MnM in identifying a poorly performing AP (e.g. one suffering from interference near it), we created the following experimental setup. We connect four laptops from different locations to a specific AP. We reduced the capacity of the AP by introducing a 500 ms delay on all packets traversing through it. The experiment tests whether MnM can correctly determine that multiple performance faults observed for clients associated with the same AP do, in some cases, imply that the AP is at fault. As shown in the second row of Table 1, MnM correctly identified the AP as the root cause for all of our observations.

Problems Due to Handoff: Wireless laptops sometimes experience bad performance because their device driver is too aggressive at changing APs in an attempt to achieve better performance.

We setup the following experiment to evaluate MnM’s ability to correctly detect problems due to AP handoffs. We forced one laptop to switch between two APs every 30 seconds, causing the performance of the client to suffer. Other clients associated with the two APs from different locations, and they continued to perform normally. As shown in the third row of Table 1, MnM identified the handoff as the correct root cause for 86% of the observations.

For the remaining 14% of the observations, the AP was identified as the topmost root cause and the handoff was ranked second. This is actually the correct result, as further investigation showed one of the two APs began experiencing outside interference during the experiment, and hence all clients associated with that AP saw poor performance. This experiment highlights how the Inference Engine is able to quickly identify the right root cause even under rapidly changing conditions.

Simultaneous Diagnosis: To measure how well MnM deals with multiple simultaneous failures, we performed two experiments where we injected multiple faults at the same time.

For the first experiment, we deliberately delayed the packets entering and leaving the server by 500 ms, and we simultaneously placed two clients at a location with known poor performance. The expected outcome for this experiment is for the server to be the highest-ranked root cause and the location to be the second highest. MnM correctly ranked these two root causes for all the obser-

Target Root Cause	% the target Root Cause is first	Other Root Causes in top two	Reasons for other root causes
Location	55	Machine, Server, AP	Location error
AP	100	First-hop router	Real congestion at the server
AP Handoff	86	Location, Machine, AP	Few positive observations through the first-hop router
Server	100	Last-hop router	Location error, AP failures
Simultaneous Faults	100	AP	Few positive observations for the last-hop router
		First-hop router	Few positive observations for the first-hop router

Table 1: Root cause analysis

uations.

In the second experiment, we placed two clients in a bad location, and we again delayed packets traversing the AP so that performance of all clients associated with it suffered (not just the two at the bad location). The inference algorithm performed as expected and correctly ranked the AP as the highest-ranked root cause and the bad location as the second-highest-ranked root cause for all observations.

7. CONCLUSION

This paper highlights the issues that an enterprise network management and diagnosis system must handle when all its users are nomadic. These issues include rapidly changing dependencies, root cause analysis in unified wired and wireless networks and the impact of physical location on application performance. We present MnM, an end-host based, integrated network monitoring and fault diagnosis system, and we show that taking an integrated approach to wired and wireless monitoring improves the accuracy of fault diagnosis.

8. REFERENCES

- [1] A. Adya, P. Bahl, R. Chandra, and L. Qiu. Architecture and Techniques for Diagnosing Faults in IEEE 802.11 Infrastructure Networks. In *MOBICOM*, 2004.
- [2] AirDefense: Wireless LAN Security. <http://airdefense.net>.
- [3] AirTight Networks. <http://airtightnetworks.net>.
- [4] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *SIGCOMM*, 2007.
- [5] P. Bahl, R. Chandra, D. maltz, P. Patel, J. Padhye, and L. Ravindranath. Towards Unified management of Networked Services in Wired and Wireless Enterprise Networks. Technical report, 2008. MSR-TR-2008-18.
- [6] P. Bahl and V. N. Padmanabhan. RADAR: An in-building rf-based user location and tracking system. In *INFOCOM*, 2000.
- [7] M. Balazinska and P. Castro. Characterizing mobility and network usage in a corporate wireless local-area network. In *MOBISYS*, 2003.
- [8] R. Chandra, J. Padhye, A. Wolman, and B. Zill. A Location-based Management System for Enterprise Wireless LANs. In *NSDI*, 2007.
- [9] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benko, J. Chiang, A. Snoeren, G. Voelker, and S. Savage. Automated cross-layer diagnosis of enterprise wireless networks. In *SIGCOMM*, 2007.
- [10] Y.-C. Cheng, J. Bellardo, P. Benko, A. Snoeren, G. Voelker, and S. Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *SIGCOMM*, 2006.
- [11] Private conversation with Dell lab members.
- [12] F. Giroire, J. Chandrashekar, G. Iannaccone, K. Papagiannaki, E. M. Schooler, and N. Taft. The cubicle vs. the coffee shop: Behavioral modes in enterprise end-users. In *Proc. of PAM*, 2008.
- [13] S. Gittlen. "want to manage your wired/wireless lans together? too bad". *Computer World*, March 2007.
- [14] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A Tool for Failure Diagnosis in IP Networks. In *Proc. MineNet Workshop at SIGCOMM*, 2005.
- [15] R. R. Kompella, J. Yates, A. Greenberg, and A. Snoeren. IP Fault Localization Via Risk Modeling. In *Proc. of NSDI*, May 2005.
- [16] D. Kotz and K. Essien. Analysis of a campus-wide wireless network. In *MOBICOM*, 2002.
- [17] M. Lopez. Forrester Research: The State of North American Enterprise Mobility in 2006. December 2006.
- [18] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing MAC-level behavior of wireless networks in the wild. In *SIGCOMM*, 2006.
- [19] HP Openview. <http://www.openview.hp.com/>.
- [20] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. WAP5: Black-box Performance Debugging for Wide-area Systems. In *WWW*, May 2006.
- [21] M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, Feb. 1996.
- [22] EMC Smarts Family. http://www.emc.com/products/software/smarts/smarts_family/.
- [23] IBM Tivoli. <http://www.ibm.com/software/tivoli/>.
- [24] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High Speed and Robust Event Correlation. In *IEEE Communications Magazine*, 1996.
- [25] M. A. Youssef, A. Agrawala, and A. U. Shankar. WLAN location determination via clustering and probability distributions. In *IEEE Percom*, 2003.