Model-based Fault Diagnosis for IEEE 802.11 Wireless LANs

Bo Yan and Guanling Chen
Department of Computer Science, University of Massachusetts Lowell
1 University Avenue, Lowell, MA 01854
{byan, glchen}@cs.uml.edu

***Abstract*— The increasingly deployed IEEE 802.11 wireless LANs (WLANs) challenge traditional network management systems because of the shared open medium and the varying channel conditions. There needs to be an automated tool that can help diagnosing both malicious security faults and benign performance faults. It is often difficult, however, to identify the root causes since the manifesting anomalies from network measurements are highly interrelated. In this paper we present a novel approach, called *MOdel-based self-DIagnosis* (MODI), for fault detection and localization.**

Our solution consists of Structural and Behavioral Model (SBM) that is constructed using both *structural* causality from wireless protocol specifications and *behavioral* statistics from network measurements. We use logic-based backward reasoning to automate fault detection and localization based on SBM, by comparing observed network measurements with expected network behaviors and by tracing back causality structures. The reasoning algorithm and the model description are decoupled so a SBM model can be easily updated for varying WLAN configurations and changing network conditions. Compared to previous work, the contribution of this paper is the architecture and the algorithm of the diagnosis core, rather than the WLAN measurement techniques.

We built and deployed MODI-embedded wireless APs that can detect both security attacks and troubleshoot performance problems. These MODI-enabled APs can also cooperate to diagnose cross-AP problems, such as those caused by device mobility. The evaluation results demonstrate that the proposed model-based diagnosis is fast and effective with little overhead.

I. Introduction

The ubiquitously available IEEE 802.11 Wireless LANs (WLANs) boost work productivity and increase user mobility. The open shared wireless medium and the varying channel condition, however, often challenge mobile users with malicious security attacks or benign performance problems, which are difficult for network administrators to manually troubleshoot the root causes because the manifesting anomalies from network measurements are highly interrelated. Thus an automated fault diagnosis system for 802.11 WLANs is necessary.

There are existing solutions aiming to manage complex 802.11 WLANs [2], [3], [24]. These tools allow detailed WLAN network measurements and provide low-level statistics, such as packet throughput, MAC layer retransmissions, and device signal strength. While they are helpful to *monitor* WLAN operation conditions and detect anomalies, they usually cannot determine the internal dependencies across network behaviors, thus providing little help on high-level fault diagnosis and remedy for WLANs.

Other WLAN management approaches integrate various network measurements statistics and employ ad hoc rules for specific fault diagnosis, such as detecting MAC-layer misbehaviors [27] and unauthorized rogue access points (APs) [5], [33], locating physical-layer errors [30], isolating delay-causing components [11]. These rules are often unstructured and may become unmanageable when considering a comprehensive set of faults and as network configurations change. While distributed dependency can be automatically learned to some degree to assist network diagnosis [4], this technique is limited to application level and WLAN problems are often localized.

In this paper we present the MODI (*MOdel-based self-DIagnosing*) system that addresses the fault diagnosis challenges in WLANs. The focus of our approach is how to structure the diagnosis rules to reason a diverse set of WLAN faults, instead of proposing new WLAN measurement methods. We require that the reasoning process be human understandable so it can be used to take remedy actions. The system must also be tunable since new faults may be introduced and WLAN configurations may be changed.

MODI consists of three components. Firstly, MODI uses an explicit fault model that represents protocol and functional components (Section III-A). The model's *causality structure* comes from the "knowledge" of 802.11 protocol specifications and WLAN configurations, and the model's *expected behaviors* comes from statistical observations and state inferences from WLAN measurements. Secondly, MODI employs the logic-based backward reasoning that can recursively traverse the fault model for automated diagnosis (Section III-B). Finally, MODI uses a rule-based engine to decouple the model representation and reasoning algorithm, thus the model can be easily updated to include additional faults or to reflect configuration changes (Section IV).

For demonstrative purposes, we built and deployed an office-based testbed with MODI-embedded self-diagnosing APs. MODI enables these APs to detect both malicious security attacks and benign performance problems. The evaluation results suggest that MODI is effective in fault diagnosis and imposes little overhead. We also demonstrate that MODI can be deployed outside of APs to diagnose, for example, cross-AP problems such as those caused by device mobility.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6833
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6833

The remainder of this paper is organized as follows. Section II discusses related work. In Section III, we present MODI's fault model and backward reasoning algorithm. Section IV describes the system design and implementation and Section V describes the system evaluation results. Finally, we discuss the limitations and scalability issues of MODI in Section VI and conclude in Section VII.

II. Related Work

Existing work on WLAN management has focused on building new architectures integrating wireless measurements and/or other wired monitors, such as using all server, AP, and client modules [1], using client-only cooperative components [10], using desktop as wireless monitors [5], or using dedicated wireless sniffers [28]. New measurement techniques have also been proposed to address channel coverage problem, such as using different channel sampling strategies [15], using nearby wireless sniffer cooperations to increase frame captures [16], and using centrally scheduled channel allocation to increase AP coverage [34], [32]. Since a single wireless sniffer may not be able to capture all the frames in the range, either due to resource constraints or frame collisions, traffic merging from distributed sniffers and frame recovery from captured traces have also been proposed [12], [22], [28].

To diagnose individual faults, researchers have studied what *measurement* data is necessary, such as to detect MAC-layer misbehaviors [27], to detect physical-layer problems [30], to detect unauthorized rogue APs [5], [34], and to isolate delay-causing WLAN components [11]. Most of these fault-diagnosis approaches employ ad hoc rules based on statistical measurements. While they may work well for individual fault, ad hoc rules are unstructured and may become unmanageable when considering a comprehensive set of interrelated faults or as network configurations change. More structured approaches may involve statistical modeling, matching measurements against previously learned statistical model, such as to detect wireless spoofing [29]. Other approaches include learning application-level dependency-graph models from passive network measurements [6], or learning decision-tree models from detailed simulation [26].

Applying learning-based models in WLANs, however, may require frequent model re-learning given dynamic wireless environment. Thus it is more desirable to have stable model structures for WLAN diagnosis. Davis describes a diagnostic reasoning technique, called constraint suspension, to establish an explicit causality model [13], which provides a powerful troubleshooting tool based on dependency structures and expected behaviors. Kleer and Williams further develop a general diagnostic theory using the perspective of diagnosis as identifying consistent modes of behaviors [14]. While these modeling techniques have been mostly used for troubleshooting hardware problems, we believe that they are also suitable for WLAN diagnosis. Although the model's "expected behaviors" may change as WLAN operations change, the model's causality structure remains the stable as long as the WLAN configurations do not change.

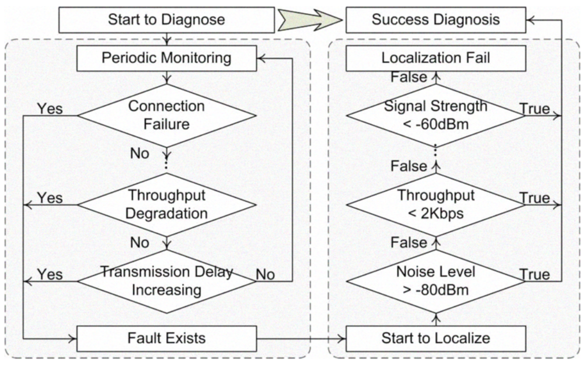

Fig. 1. Ad hoc rule based diagnosis.

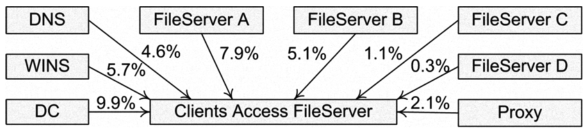

Fig. 2. Probability-based dependencies.

III. Model-based Fault Diagnosis

When fault diagnosis rules are organized in an ad hoc fashion, such as Snort-Wireless does [31], the rules need to be checked sequentially. On the other hand, the rules are usually built either based on packet headers or statistical measurements from WLAN traffic, such as the number of MAC-layer retransmissions and the packet throughput, as shown in Figure 1. Thus matching rules against such detailed measurements can be quite wasteful if irrelevant rules are also needed to be checked.

It is possible to learn which rules are more likely to be triggered given current measurements based on historical information, thus rule-checking can be ordered in such a way that diagnosis process may stop quicker when a high-probability rule triggers. For example, Sherlock system learns the inference graph automatically based on application communication patterns [6]. When a fault occurs, such as the failed client access shown in Figure 2, the highest probable cause Domain Controller (DC) will be checked first. While this approach imposes semi-structure to rule space, the diagnosis process is stochastic at best, can only handle one fault at a time, and the model structure has to be frequently re-learned to cope with the dynamic traffic.

In this paper, we present a new approach MODI (MOdel-based self-DIagnosis), which borrows the modeling technique from traditional hardware troubleshooting domain. The core reasoning method relies on explicitly-constructed causality structural and expected behavioral model [13], and uses backward reasoning for automated diagnosis over this model, achieving a balance between diagnosis speed and accuracy.

Next we describe the *Structural and Behavior Model* (SBM) that represents the functional components in WLANs and the *Effect-Mismatch Algorithm* (EMA) for the fault localization process using SBM. Note that SBM and EMA are decoupled so the model can be updated independent of the reasoning

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6833
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6833

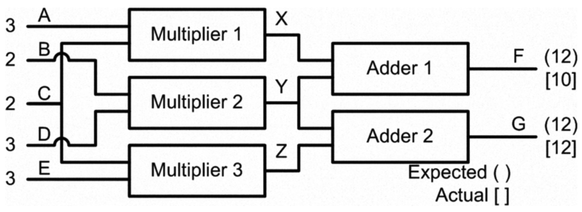

Fig. 3. Structural and behavioral model for a circuit board.

algorithm.

A. Structural and Behavior Model

The idea of SBM is simple and we use an electronic circuit as an example, shown in Figure 3. An SBM describes behavior of a functional model using logical level interrelations of the components in the structure [13]. Clearly the behavior of the adder 1 can be expressed by the logical causality of its inputs X, Y, and its output F. Namely, the expected value of F is equal to the sum of the values of X and Y. If the measured value of F is different than the expected value, either the adder itself or the inputs have errors. Similarly, the output X of multiplier 1 also has an expected value to be the multiplication of the input values of A and C. Thus an SBM consists of functional modules connected with inputs and outputs, and each output has an expected value that can be used to check against measured value for determining potential malfunctions.

SBM can also be applied in a similar fashion to troubleshoot WLAN problems. Take diagnosing wireless connectivity problems as an example. The connectivity between client and AP relies on several other functional components: good physical-layer connection, good MAC-layer connection, successful authentication, and successful association. Whether the physical-layer connection is good (output) in turn depends on signal strength and noise level (inputs). Whether the MAC-layer connection is satisfying (output) depends on channel congestion level [19] and whether there are MAC misbehaviors (inputs) [27]. We can use some MAC-layer measurements to estimate current MAC-layer connection quality, such as the number of frame retransmissions and the frame transmission delay (time between placed in transmission queue and transmitted over the air) [11]. If the value of these metrics exceed a threshold determined by empirical experience, we consider the (output of) the MAC-layer connection is not as expected (malfunction) and start checking its inputs (channel congestion and MAC misbehaviors).

The above example is by no means to be comprehensive. The root causes of wireless connectivity issues may also reside at higher layers, such as that the AP is overloaded, the client fails to obtain a DHCP address, or the DNS request fails to resolve. Fortunately SBM is fairly general and easy to extend, as long as the new functional module has a measurable output that can be checked against an "expected" value to determine whether the output is abnormal or not. In Figure 4 we give an extended SBM model for wireless connectivity diagnosis. The functional modules are represented by non-shaded boxes, the root causes of faults are shaded boxes, and the measurable metrics are labeled on the causality links.

B. Effect-Mismatch Algorithm

Given a SBM and the symptoms of malfunctions, fault diagnosis process determines which components in the SBM is the root cause of the malfunction. Instead of postulating a possible fault and matching its consequences against measurements, we use discrepancy detection [13] that looks for mismatches between the expected values and the operational measurements. If a mismatch has been found for a functional component's output value, we recursively check its upstream inputs for any mismatches.

Again we use Figure 3 as an example, and we assume that the inputs A, B, C, D, E have the values 3, 2, 2, 3, 3 as shown. If all the multipliers and adders work correctly, the expected values of F and G should both be 12. If the value of F turns out to be 10, there is a mismatch between the expected value (12) and the measurement (10). Instead of concluding adder 1 is broken, we have to trace back to its inputs along the causality links.

The causality structure of adder 1 indicates that it depends on the output values X and Y from multiplier 1 and 2. The expected values of X and Y should both be 6. If X, Y, and Z are not measurable, it may be difficult to determine which multiplier is faulty. On the other hand, since the value of G is as expected (12), we may conclude that it is most likely multiplier 1 has malfunction because adder 2 and its inputs (Y and Z) work correctly.

In Figure 4 we rearrange the model so all the potential faults are listed as sources without inputs. The effects of the faults will propagate through the causality structure and manifest as behavioral mismatches of some output values. The middle unshaded boxes act as virtual functions that aggregate their inputs: if their output value has discrepancy, one of their inputs must have some malfunction. This generic definition allows decoupling of the diagnosis algorithm, so the model itself can be updated (either the causality structure or the expected values) separately.

Formally, we call the diagnosis procedure *Effect-Mismatch Algorithm* (EMA), which differentiates SBM modules as three types, *surface*, *cause*, and *middle* modules. The *surface* modules correspond to functional components, whose failures are observable to end-users or monitoring tools, such as connectivity, performance, or malicious attacks. Once the failure is detected, it triggers fault diagnosis process, shown as follows.

```
function em_trigger (module)
  if (is_triggered(module)) then
    to_check(module) = true
  endif
```

The *cause* modules are directly related to the root causes of malfunctions that determine the failures of the surface modules. If the EMA inference has reached one of such module, the fault is localized and returned.

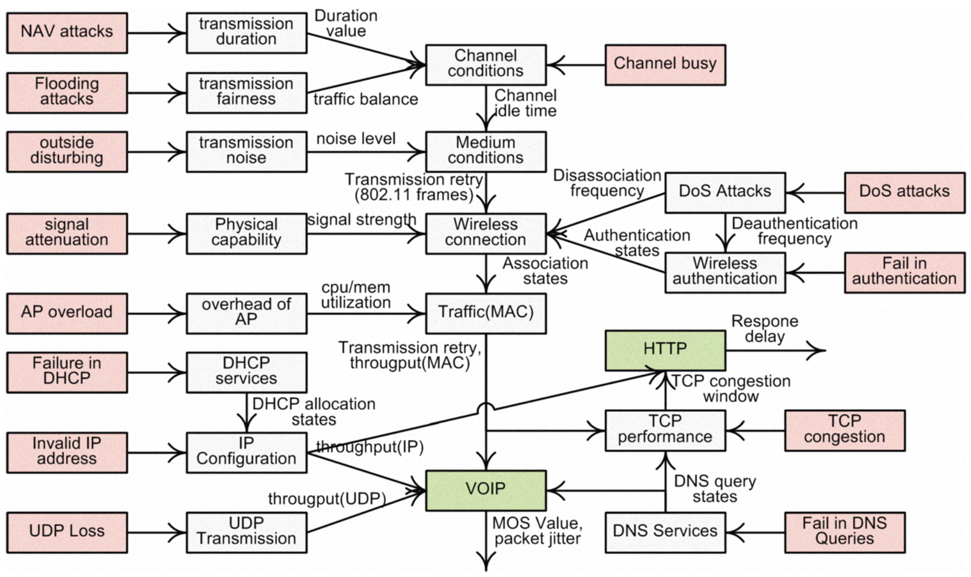

Fig. 4. Structure and Behavior Model for wireless connectivity problems.

```
function em_localize (module)
    if (no_inputs(module)
        and to_check(module)) then
        fault_localize(module)
    endif
```

The *middle* modules connect the surface modules and the cause modules, which contain different behaviors cross multiple layer protocols and services that indicate the relationships between malfunctions and their causes. EMA simply checks individual inputs and continues a depth-first search if an input mismatch is found.

```
function em_diagnose (module_out)
    for module_in in inputs(module_out)
        if (is_linked(module_in, module_out)
            and to_check(module_out)
            and is_misbehavior(module_in)) then
            to_check(module_in) = true
        endif
    next
```

Note that we list three separate functions here only for the illustration purpose. The EMA is not implemented using a procedural programming language. Rather, it is implemented as a set of rules like logic programming (Section IV-C). The execution order is not predefined and is solely dependent on the model structure. For example, if an anomaly is detected and the $is\_triggered(module)$ is set to true, the rule in $em\_diagnose$ will run to check that module's inputs. Setting to_check to be true in the conditional statement may recursively trigger $em\_diganose$ towards upstream input modules until no_inputs satisfies in $em\_localize$.

The order of checking a module's inputs is not specified by EMA. It can be optimized, however, that inputs are assigned with priorities so EMA can check them in a priority-descending order, so the more likely faulty input will be checked earlier than others. The actual priorities can be assigned manually according to WLAN administrator's experience or they can be learned and adjusted automatically based on historical diagnosis results. Currently EMA triggers diagnosis from the surface modules, and it can be further optimized by jumping directly to certain middle module based on its historical faulty probability and backtrack if this heuristic guessing is wrong.

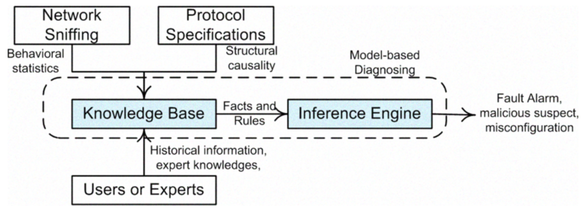

Fig. 5. Architecture of the MODI system.

IV. System Design and Implementation

Figure 5 shows the basic architecture of the proposed MODI system. A knowledge base holds the SBM model representation and the inference engine implements the EMA algorithm. The network monitor is a general term, which can be PCAP-based network sniffers or other relevant measurement points, such as log analyzers. The measurements are used to compare against with SBM's expected behaviors for EMA-based inference. As the WLAN configurations or the understandings of the WLAN operations change, administrators can update the text representation of the SBM model in the knowledge base without touching the inference engine.

A. Self-Diagnosing APs

For evaluation purposes, we enhanced existing APs with MODI to build self-diagnosing APs (SAPs). Many existing

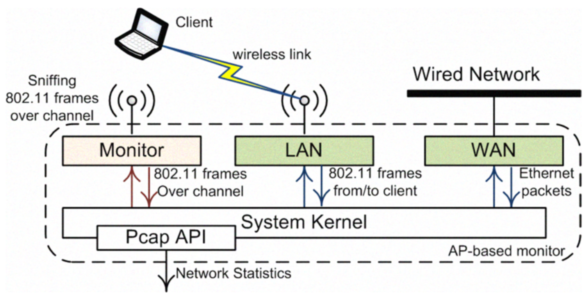

Fig. 6. The structure of self-diagnosing AP.

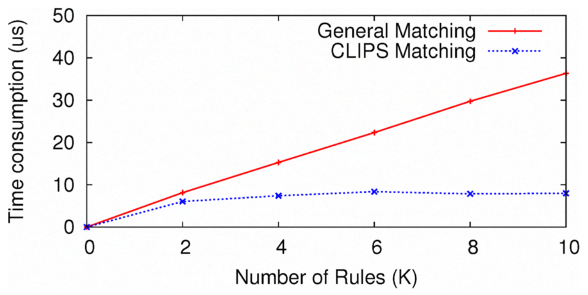

Fig. 7. Time consumption for rule matching.

approaches use dedicated wireless sniffers to obtain detailed physical and MAC-layer measurements [5], [23]. We decided, however, to run MODI inside the AP to avoid a separate wireless sniffing infrastructure and MODI can better correlate wireless and wired measurements inside the bridging AP.

It is possible to time-share a single wireless card so it can perform two tasks simultaneously: serving clients and sniffing the air, such as using MultiNet [9]. This approach, however, disrupts normal client communication and provides poor measurement fidelity. Thus we built SAPs with two wireless cards, one dedicated for serving clients and the other for wireless sniffing.

We used the MadWifi [1] driver that supports both AP and sniffing modes, in which it allows wireless cards with Atheros chipsets to capture IEEE 802.11 frames. We chose wireless cards with Atheros AR5212 chipset for the SAPs. Figure 6 shows the structure of a SAP. There are three network cards in a SAP. The Ethernet card works as the WAN interface. One of the wireless cards is running in AP mode as LAN interface, and the other is running in sniffing mode for packet sniffing over the wireless channel. We can easily observe network traffic from three interfaces to calculate traffic statistics and infer protocol states for MODI.

Running MODI inside SAP raises the concern of its overhead, and we evaluate MODI's impact on embedded devices like SAPs in Section V.

B. Information Collection

The MODI inside a SAP runs two threads, one sniffing the wireless interface and the other sniffing the wired one, so it is possible to correlate the wired and wireless events. For example, if the TCP connection is observed on the wired side to have prolonged congestion window, MODI should be able to tell whether this is caused by poor wireless transmissions. Note that, however, the communication between a client and the SAP may be encrypted if WEP, WPA, or 802.11i is used. This poses a challenge to associate wireless clients, identified by MAC addresses on the wireless interface, with higher-layer communication end points, identified by the IP addresses on the wired side. It may be too resource consuming or even infeasible to get the key for MODI to decrypt wireless packets on the fly. Instead, depending on whether SAP runs in switching or routing mode, MODI can either snoop the ARP packets or to query Linux kernel tables to find the MAC-IP address mappings.

[1]http://madwifi-project.org

The wireless sniffer captures the MAC-layer frames on the same channel operated by the SAP, including the low-level physical-layer information in the radiotap header provided by the MadWifi driver. It can estimate client's signal strength, channel congestion level [19], frame retransmissions, frame transmission delay [11], detect MAC-layer misbehaviors [27], and infer current protocol state based on the IEEE 802.11 specification, such as whether the client is being authenticated, is associated, is transmitting data, or is in the power-saving mode.

C. Diagnosis Engine

MODI is built upon a logic-based rule system, CLIPS (C Language Integrated Production System) [18], which is a software tool that is often used to develop expert systems. CLIPS contains *facts* and *rules*, which have text representations and follow first-order logic formalism, thus supporting both forward and backward reasoning. The rules are evaluated when new facts are inserted or existing facts are updated. To avoid the naive evaluation of every rule against all facts, CLIPS uses an efficient pattern-matching algorithm RETE to pre-compile the conditions of the rules [17]. Figure 7 shows the comparison of the time consumption of matching one rule with sequential evaluation and with CLIPS' RETE evaluation. Clearly CLIPS scales well in terms of the ruleset size.

MODI's network sniffers extract statistical traffic summaries and state inferences, and insert them into CLIPS' knowledge base as facts regarding current WLAN operations. The SBM model is also specified as facts describing the structural connections and expected output values. The EMA reasoning algorithm is implemented as rules in CLIPS' inference engine, which will be triggered if there is a mismatch between the measurements and the expected values. The diagnosis results are saved into an HTML file that can be viewed by WLAN administrators remotely.

D. Fault Injection

There are a variety of faults, either malicious or benign, that may disrupt WLAN operations [7], [1], [27], [5], [28]. Here we describe several representative faults that we have implemented and injected into the WLAN to test MODI.

The 802.11 header of MAC frames has a "duration" field containing a value of the time the sender wants to reserve the channel for its transmission. All other stations hearing this frame will set their network allocation vector (NAV) and wait until the reserved time passes. Thus a greedy transmitter can manipulate the protocol parameters when sending RTS or DATA frames to increase the included NAV value in order to prevent the stations in range from contending during this time [27]. We call this "NAV Attack."

The 802.11 MAC frames can easily be spoofed, meaning the source MAC address in the frames can be set to arbitrary value [29]. Thus an attacker can spoof a Disassociation frame from an AP to disconnect an associated client, resulting a type of Denial-of-Service attack [7]. We call this "Disassociation Attack." To be effective, an attacker needs to send multiple spoofed Disassociation frames to keep the client from re-association.

If a wireless channel is congested [19], the perceived application performance at a client may degrade significantly, because a frame may need to wait a long time before it can grab the channel for transmission [11]. To evaluate the MODI's ability to detect channel congestion, we used a dedicated laptop that can inject wireless frames at a controlled rate. This fault and previous two attacks can all be implemented using a frame injection tool, such as File2Air.[2]

At the physical layer, the signal strength plays an important role on wireless transmission performance [30]. The signal strength may vary due to distance, environment changes, or device mobility. This fault is relatively easy to detect since MODI captures frame-level signal strength from wireless sniffers and there is a direct relationship between average signal strength and MAC-layer throughput [8].

Besides wireless issues, the perceived performance problems may be caused at the wired components that is part of the WLAN infrastructure. For example, a slow DNS server may force a HTTP session to wait a long time. We implemented a "DNS Query Flooding" attack to achieve this effect.

V. Experimental Results

In this section we present the evaluation results of the proposed MODI system. First we describe the testbed deployed in our department building. Then we evaluate the system overhead when running MODI in the self-diagnosing APs. We also discuss the effectiveness using MODI to diagnose several common WLAN faults. Finally we deployed MODI outside of APs to diagnose cross-AP problems, such as those caused by device mobility. In this paper we focus on the IEEE 802.11 MAC layer faults and have not implemented detection methods for TCP problems, which are addressed by Adya and others [1].

A. Test Environment

We deployed a MODI testbed in our department building, a six-story concrete structure. Each floor is 260 feet long and 85 feet wide, with hallway walls and floors made of concrete. We built three MODI-enabled self-diagnosing APs (SAPs) on the third floor of our building. These APs are RouterBOARD 532A devices, installed with OpenWrt Kamikaze 7.09 with Linux kernel 2.6.21-5. Each SAP has MIPS 400MHz CPU, 64MB RAM, 2GB Compact Flash disk. We chose Wistron Neweb CM9 with Atheros AR5212 chipset as our wireless radio cards, and installed MadWifi 0.9.4 (r2568-20070710 svn snapshot) for serving clients in the AP mode and for frame captures in the sniffing mode. Furthermore, We used omni-directional antennas that have 3dbi gains on the 2.4GHz frequency channel. Besides these three SAPs, we also used a set of T42 laptops installed with Debian Linux or Windows XP as wireless clients, which generate network traffic through SAPs or inject faults for testing purposes. Each wireless client can access the backbone network through SAPs with which the clients are associated.

We implemented several WLAN faults as described in Section IV-D. We used File2Air to inject attacking frames into our testbed. File2Air is a command-line utility, which allows to specify and inject arbitrary 802.11 frames into a wireless channel using many wireless drivers. We customized the header fields to create specific 802.11 frames with large duration values for NAV attacks or the victim's MAC address for the Disassociation attacks.

B. System Overhead

Wireless access points are typically embedded devices with limited resources. For example, our SAPs have 400MHz CPU and 64MB RAM. Therefore, the fault diagnosis system running on wireless APs should have low resource consumption. In addition to serving clients as a normal AP function, MODI on the SAPs also sniffs the wireless and Ethernet interfaces, decodes captured packets, calculates traffic summaries and infers protocol states, and interacts with CLIPS-based diagnosis engine.

To determine the impact on incoming and outgoing traffic when running MODI directly on APs, we considered a scenario in which a wireless client is associated with a SAP and generates UDP traffic to a desktop PC connected to the wired backbone network. The wireless client used iperf [3] for traffic generation and throughput reporting. In this scenario, both the AP and the client's wireless card are likely to be continuously busy and the packet loss of the UDP traffic is not caused by the weak transmission signal.

We first measured the capacity of the SAPs without running MODI. After allowing iperf to generate different traffic loads from 2Mbps to 54Mbps, we found that the maximum bandwidth of these APs was about 17Mbps while the packet jitter was up to 3ms and the average packet loss was up to 1fps (frame per second). On the other hand, we repeated the same experiments while running the MODI on the SAPs. As shown in Figure 8–10, the impact of running MODI on SAPs was minimal, and the bandwidth, packet jitter, and average packet loss were close to the normal APs without running MODI.

[2]http://www.willhackforsushi.com/File2air.html

[3]http://iperf.sourceforge.net/

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6833
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6833

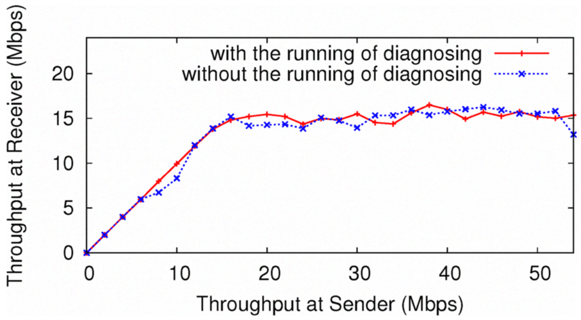

Fig. 8. Throughput stress test for self-diagnosing AP

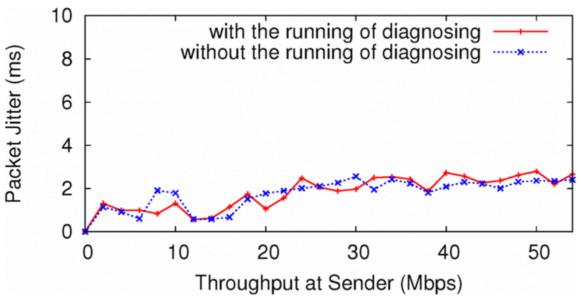

Fig. 9. Jitter stress test for self-diagnosing AP

Next we set the UDP traffic generated by the wireless client to be 20Mbps, which allowed the SAPs to achieve their maximum throughput. We then monitored the CPU and memory utilization on the SAPs for ten hours and recorded their average values every 30 minutes. The results suggested that the CPU utilization mostly stayed below 50% during the whole experiment period, shown in Figure 11. After running MODI, CPU utilization increased from 50% to 75%, and stayed below 80%. In Figure 12, memory consumption always stayed below 30% even if the needed memory increased when MODI was running. It shows that MODI consumed less than 3% additional memory during the entire experiment period.

To increase the load targeting against MODI, we set up another wireless station that injected arbitrary frames into the air. These frames were not destined to SAP's client-serving wireless interface, whose radio firmware directly dropped any frames not for itself. On the other hand, the MODI's sniffing interface picked up these frames and did all normal processing.

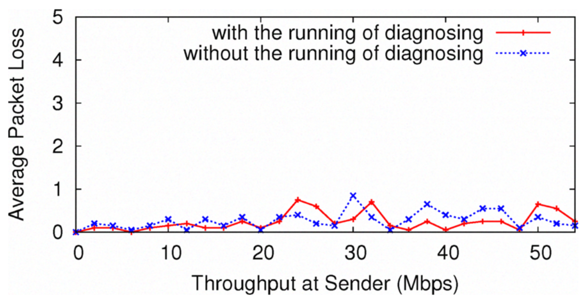

Fig. 10. Packet Loss test for self-diagnosing AP

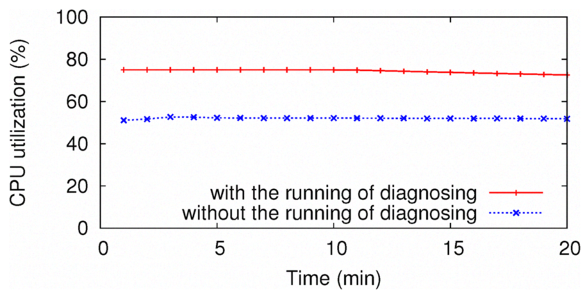

Fig. 11. CPU utilization of self-diagnosing AP

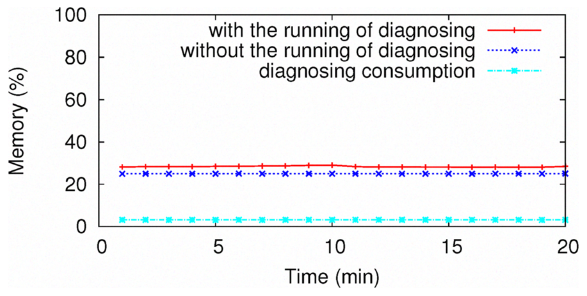

Fig. 12. Memory utilization of self-diagnosing AP

As shown in Figure 13 and 14, CPU utilization by MODI increased significantly due to increased frame processing load while the memory utilization remained low. While CPU utilization may be high, MODI itself did not reduce AP's throughput. Rather, it may drop sniffed frames as discussed below. Thus we conclude that MODI consumed reasonable resources on SAPs and imposed little performance overhead.

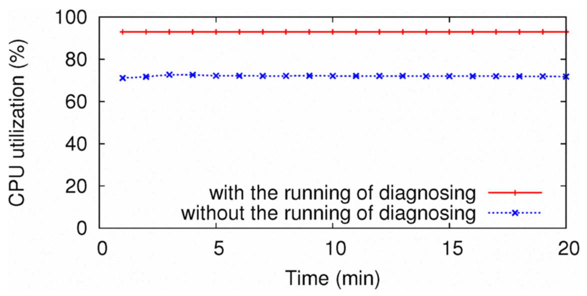

Fig. 13. CPU utilization of self-diagnosing AP with frame injection

C. Fault Diagnosis

Diagnosing WLAN faults is particularly challenging since the basis for detecting these faults is most likely the anomalies found in various network measurements. There are often no precise "signatures" that can be defined for individual faults, unlike the well-specified rules used by wired intrusion detection system such as Snort.[4] Differentiating anomalies caused by faults or by natural traffic variations from workload and environment sometimes can only rely on thresholds, which are learned from historical observations. The focus of this

[4]http://www.snort.org/

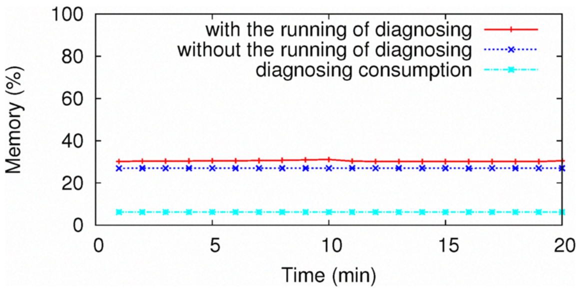

Fig. 14. Memory utilization of self-diagnosing AP with frame injection

TABLE I
RESULTS OF INDIVIDUAL FAULT DIAGNOSING

Faults	Occurred	Detected	Accuracy
DNS Query Flooding	200	200	100%
NAV Attack	200	192	96%
Disassociation Attack	200	198	99%
Channel Congestion	200	200	100%
Weak Transmission Signal	200	190	95%

paper is the diagnosis architecture, rather than the techniques finding most appropriate thresholds for individual problems, which we discuss elsewhere [29], [34].

To demonstrate the diagnosis effectiveness of the proposed MODI system, we set up a SAP through which two wireless clients were continuously downloading a 3.4GB file using HTTP. We used additional wireless stations to inject various faults as described in Section IV-D. We repeat each fault injection separately for 200 times, and each fault injection last for about 1 second. The results of the MODI diagnosis reported on the SAP are shown in Table I. MODI achieved reasonable diagnosis accuracy, catching most of the injected faults.

MODI did miss some injected faults, such as 8 NAV attacks, 2 disassociation attacks, and 10 weak transmission signal cases. While the thresholds we chose for these faults may not be the best ones, we found that another contributing factor of mis-diagnosis was frame loss by the wireless sniffers. SAP is an embedded device and MODI's CPU utilization may become high as it processes more captured frames. When the processing capability lags behind the rate of incoming frames, the PCAP buffer in the Linux kernel starts to drop captured frames. Loss of wireless frame by wireless sniffers will thus reduce the MODI's diagnosis effectiveness.

We can determine the frame loss ratio by comparing the number of frames captured by the radio interface, available through a system call, and the number of frames captured by MODI sniffers. The difference is the number of frames dropped by PCAP library. Table II shows that the frame loss ratio may reach 11% when running MODI on the SAP. Note that the frames for the wireless interface serving clients in AP mode did not suffer such losses, since the frame processing by the driver is quite efficient. Rather, frames on the sniffing interface were lost because MODI needed CPU cycles to analyze captured frames and its speed may become slower than the rate of inbound frames.

TABLE II
THE IMPACT OF DIAGNOSIS ON ON SNIFFING LOSS

diagnosing state	wireless card	Our system	Loss Ratio
stop	4642639	4500352	3.07%
running	4395746	3907172	11.11%

TABLE III
RESULTS OF MULTIPLE FAULTS DIAGNOSING

Faults	Occurred	Detected	Accuracy
DNS Query Flooding	80	80	100%
NAV Attack	40	37	92.5%
Disassociation Attack	40	36	90%
Channel Busy	40	40	100%
Weak Transmission Signal	80	76	95%
Total	280	264	96.1%

Frame loss is inevitable for wireless sniffers, thus having an impact on diagnosis accuracy. Previous work has described how to merge frames captured from nearby cooperating sniffers [28], to improve frame capture efficiency by directly modify the MadWifi driver without passing through PCAP library [12], or to recover lost frames using 802.11 MAC protocol state machine [22]. We plan to look into integration of some of these techniques as future work.

We also challenged MODI by injecting all five faults simultaneously to see whether MODI was capable of diagnosing multiple faults. The results are shown in Table III, suggesting MODI is also quite effective in catching simultaneous faults. In particular, the diagnosis accuracy for DNS Query Flooding, Channel Congestion, and Weak Transmission Signal remained similar to single-fault diagnosis, while the accuracy for diagnosing NAV and Disassociation attacks decreased due to their sensitivity to the frame loss.

Weak transmission signal is inherently difficult to diagnose, because signal strength fluctuates all the time even if the transmitter is stationary [29]. Interestingly, the fluctuation of signal strength may cause a station to swing back and forth between APs, sometimes called Ping-Pong effect [20]. Thus it may be difficult for a single AP to determine whether the signal strength changes are caused by channel variations or by device mobility. To address such problems, it is important for APs to cooperate for cross-AP diagnosis as we discuss in next section.

D. Cross-AP Diagnosis

Here we demonstrate MODI can be deployed outside of APs, which serve as measurement collection points while the diagnosis engine is centralized [5], [23] to address the cross-AP faults, such as those caused by device mobility. Today's enterprise WLANs employ thin-AP switched architecture, where APs are light-weight devices that forward frames to centralized WLAN switch for all the processing. This is sometimes called split-MAC design, so time-sensitive tasks (such as ACK) are performed at the APs, while all management and data frames are handled at the switch. It is thus feasible to run MODI

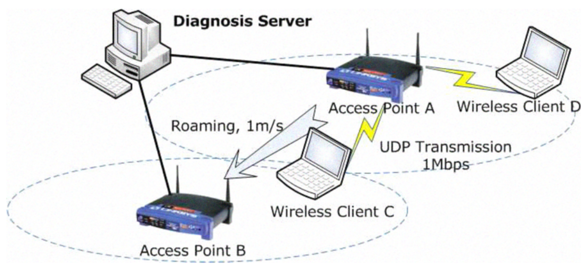

Fig. 15. An example of Wireless Mobility

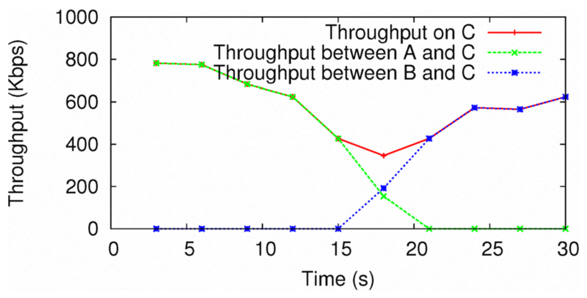

Fig. 16. Throughput during Mobility

sniffers on the APs, which send traffic summaries to the switch who runs MODI diagnosis engine, since MODI architecture is flexible so it can run either on a single device or run in a distributed fashion.

To demonstrate switch-based MODI, unlike previously discussed MODI-embedded SAP, we set up a mobility experiment. In our department building, 802.11g provides only about 100 feet communication range, as the signal drops quickly as it goes through concrete walls [21]. We deployed two APs A and B on opposite sides of the third floor. These two APs had the same ESSID and operated on the same channel, so that a wireless client C can roam from one to the other. We deployed MODI on a centralized server S to test mobility diagnosis. The experimental setup is illustrated in Figure 15.

The client C, associated with A, moved towards B with a speed of 3 feet per second, till it was re-associate with B. The entire process took 30 seconds. To evaluate the performance change during the roaming of C, we had it generate 1Mbps UDP traffic to another client D (stationary). APs A and B still ran the wireless sniffers, but not the CLIPS knowledge base and inference engine, and sent traffic summaries and state inference every 3 seconds to S who ran MODI diagnosis subsystem.

After analyzing the traffic information obtained from A, we found that the throughput between A and C decreased to 0 at the 21st second, as shown in Figure 16, and the average transmission rate between A and C decreased to 0 at the 21st second. Intuitively, the degradation of the throughput between A and C was caused by weak signal strength. By analyzing the traffic information obtained from B, however, we found the throughput between B and C increased at the 18th second. That is, C disconnected with A and then connected to B. Furthermore, according to C's disassociation state with A and the re-association state with B, inferred by analyzing management frame exchanges captured by the wireless sniffers on A and B, MODI concluded that the wireless mobility caused the throughput degradation between C and D.

VI. Discussions

While MODI has not focused on WLAN measurements, its performance will certainly be impacted by different measurement techniques. For example, the detection accuracy can usually be improved if the measurement component can recover some missed frames by merging from nearby sniffers or by recovering frames from existing wireless or wired observations. Similarly, cross-layer measurements such as the wireless/wired correlation used in the MODI-enabled SAPs can also reduce inference complexity.

The EMA algorithm uses threshold values to detect mismatching anomalies and then trace back the SBM causality structures. While threshold-based detection is quite general and worked well in our cases, there may be other non-threshold based techniques can achieve better performance. For example, Bayesian or Neural networks encode their learning models in graphical structures. To integrate such techniques, we may need to modify SBM to have a binary module output (true or false) matched against a separate anomaly-analysis component (outside of SBM/EMA). It is, however, not difficult to make such modifications since SBM has a text representation and is decoupled from EMA.

Due to the limitation of our testbed, we have not conducted large-scale evaluations. For a single MODI-enabled SAP, we know the CPU can be a bottleneck and MODI may lose frames if the channel or the AP becomes overloaded. This calls for MODI deployment outside of APs, particularly for the enterprise environment. We do not expect the network will be the bottleneck as the periodic traffic summaries sent from APs are small and the intervals can be adjusted to tradeoff detection delays. For large campuses, we can deploy multiple MODI systems, each covering a building or domain [28].

VII. Conclusion and Future Work

To address the WLAN fault management challenge, we propose MOdel-based self-DIagnosis (MODI) that leverages both Structural and Behavioral Model (SBM) and logic-based backward reasoning Effect-Mismatch Algorithm (EMA). The novelty of this approach results in decoupled architecture where SBM can be quickly updated independent of EMA, which is often necessary for changing WLAN environment and configurations. MODI is lightweight as demonstrated to run on embedded device to build self-diagnosing APs, imposing little overhead on the AP's normal client-serving functions. While wireless sniffers may lose frames, MODI achieved reasonable accuracy results when diagnosing both individual and simultaneous faults. We also demonstrated that MODI can be deployed with enterprise-scale switch-based WLAN architecture, to address cross-AP diagnosis problem, such as those caused by device mobility.

As future work, we plan to investigate techniques that can reduce frame loss by the wireless sniffers to improve the diagnosis robustness. Based the on diagnosis results, we also plan to integrate MODI with automatic remedy actions, such as re-schedule APs' channels or re-allocate users for load balancing [25], as a foundation for self-healing WLANs.

Acknowledgments

This work is supported in part by the Science and Technology Directorate of the U.S. Department of Homeland Security under Award NBCH2050002. Points of view in this document are those of the authors and do not necessarily represent the official position of the U.S. Department of Homeland Security. We thank MAP/DIST project team at Dartmouth College and Aruba Networks for their collaboration on this research.

References

[1] A. Adya, P. Bahl, R. Chandra, and L. Qiu. Architecture and techniques for diagnosing faults in ieee 802.11 infrastructure networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, pages 30–44, Philadelphia, PA, Sept. 2004.

[2] AirMagnet. Airmagnet distributed system. http://www.airmagnet.com/.

[3] AirWave. Airwave management platform. http://www.airwave.com/.

[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 *Proceedings of the ACM SIGCOMM*, pages 13–24, Kyoto, Japan, Aug. 2007.

[5] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. Enhancing the security of corporate Wi-Fi networks using DAIR. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Uppsala, Sweden, June 2006.

[6] P. V. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 13–24, Kyoto, Japan, August 2007.

[7] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the 12th USENIX Security Symposium*, Washington, DC, Aug. 2003.

[8] J. Camp, J. Robinson, C. Steger, and E. Knightly. Measurement driven deployment of a two-tier urban mesh access network. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 96–109, Uppsala, Sweden, June 2006.

[9] R. Chandra and P. Bahl. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *Proceedings of the 23rd Conference of the IEEE Communications Society (INFOCOM)*, pages 882–893, Hong Kong, China, Mar. 2004.

[10] R. Chandra, V. N. Padmanabhan, and M. Zhang. WiFiProfiler: Cooperative diagnosis in wireless LANs. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 205–219, Uppsala, Sweden, June 2006.

[11] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benkö, J. Chiang, A. C. Snoeren, S. Savage, and G. M. Voelker. Automating cross-layer diagnosis of enterprise wireless networks. In *Proceedings of the ACM SIGCOMM*, pages 25–36, Kyoto, Japan, Aug. 2007.

[12] Y.-C. Cheng, J. Bellardo, P. Benkö, A. C. Snoeren, G. M. Voelker, and S. Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *Proceedings of the ACM SIGCOMM*, pages 39–50, Pisa, Italy, Sept. 2006.

[13] R. Davis. Diagnostic reasoning based on structure and behavior. In D. G. Bobrow, editor, *Qualitative Reasoning about Physical Systems*, pages 347–410. MIT Press, Cambridge, MA, 1985.

[14] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proceedings International Joint Conferences on Artificial Intelligence*, pages 1324–1330, Detroit, MI, Aug. 1989.

[15] U. Deshpande, T. Henderson, and D. Kotz. Channel sampling strategies for monitoring wireless networks. In *Proceedings of the Second International Workshop on Wireless Network Measurement (WiNMee)*, April 2006.

[16] U. Deshpande, C. McDonald, and D. Kotz. Coordinated sampling to improve the efficiency of wireless network monitoring. In *Proceedings of the Fifteenth IEEE International Conference on Networks (ICON)*, pages 353–358, November 2007.

[17] C. L. Forgy. *On the efficient implementation of production systems*. PhD thesis, Carnegie Mellon University, 1979.

[18] J. C. Giarratano and G. D. Riley. *Expert Systems: Principles and Programming, Fourth Edition: Principles and Programming*. Course Technology, Oct. 2004.

[19] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer. Understanding Congestion in IEEE 802.11b Wireless Networks. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 279–292, Berkeley, CA, Oct. 2005.

[20] M. Kim and D. Kotz. Periodic properties of user mobility and access-point popularity. *Journal of Personal and Ubiquitous Computing*, 11(6):465–479, August 2007. Special Issue of papers from LoCA 2005.

[21] N. Li, B. Yan, and G. Chen. A measurement study on wireless camera networks. In *Proceedings of the Second International Conference on Distributed Smart Camera (ICDSC)*, Stanford, CA, Sept. 2008.

[22] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing the MAC-level behavior of wireless networks in the wild. In *Proceedings of the ACM SIGCOMM*, pages 75–86, Pisa, Italy, Sept. 2006.

[23] MAP: Security through measurement for wireless LANs. Dartmouth College, July 2006. http://www.cs.dartmouth.edu/ map/.

[24] M. Milner. NetStumbler WLAN detection software, 2004. http://www.stumbler.net/.

[25] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill. Designing high performance enterprise Wi-Fi networks. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 73–88, San Francisco, CA.

[26] L. Qiu, P. Bahl, A. Rao, and L. Zhou. Troubleshooting wireless mesh networks. *ACM SIGCOMM Computer Communication Review*, 36(5):17–28, Oct. 2006.

[27] M. Raya, J.-P. Hubaux, and I. Aad. DOMINO: A system to detect greedy behavior in IEEE 802.11 hotspots. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 84–97, Boston, MA, June 2004.

[28] Y. Sheng, G. Chen, K. Tan, U. Deshpande, B. Vance, H. Yin, C. McDonald, T. Henderson, D. Kotz, A. Campbell, and J. Wright. MAP: A scalable monitoring system for dependable 802.11 wireless networks. *IEEE Wireless Communications, Special Issue on Dependability Issues with Ubiquitous Wireless Access*, 15(5):10–18, Oct. 2008.

[29] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell. Detecting 802.11 MAC layer spoofing using received signal strength. In *Proceedings of the 27th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1768–1776, April 2008.

[30] A. Sheth, C. Doerr, D. Grunwald, R. Han, and D. Sicker. MOJO: A distributed physical layer anomaly detection system for 802.11 WLANs. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 191–204, Uppsala, Sweden, June 2006.

[31] Snort-Wireless: A scalable 802.11 intrusion detection system. http://snort-wireless.org/.

[32] Y. Song, X. Chen, Y.-A. Kim, B. Wang, and G. Chen. Sniffer channel selection for monitoring wireless LANs. In *Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications (WASA)*, Boston, MA, Aug. 2009.

[33] W. Wei, K. Suh, B. Wang, Y. Gu, J. Kurose, and D. Towsley. Passive online rogue access point detection using sequential hypothesis testing with TCP ACK-pairs. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, San Diego, CA, Oct. 2007.

[34] B. Yan, G. Chen, J. Wang, and H. Yin. Robust detection of unauthorized wireless access points. *Mobile Networks and Applications*. In Press.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6833
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6833