Fault Detection, Isolation, and Diagnosis in Multihop Wireless Networks

Lili Qiu, Paramvir Bahl, Ananth Rao, and Lidong Zhou

December 2003

Technical Report
MSR-TR-2004-11

Network management in multihop wireless networks is key to efficient and reliable network operation. In this paper, we focus on a part of the general network management problem, namely fault detection, isolation, and diagnosis. We propose a system that employs online trace-driven simulation as a diagnostic tool for detecting faults and performing root cause analysis. We apply our system to diagnose faults such as packet dropping, link congestion, MAC misbehavior, and external noise, and show that it yields reasonably accurate results. In addition, we show that our system can be used to evaluate alternative network and node configurations to improve performance of on-going long-lived flows. Moreover, our technique is general enough to be applied in other wireless and wireline network management system.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

1. INTRODUCTION

Network management is one of the key ingredients in a successful deployment of multihop wireless networks. Unfortunately, it has received relatively little attention in the research community. Complete and correct implementation of network management requires continuously monitoring the functioning of the network, collecting information about the nodes and links in the network, removing inconsistencies and noise from the reported information, analyzing this information, and taking appropriate actions to improve network reliability and performance.

In this paper, we focus on fault detection, isolation, and diagnosis, collectively referred to as a *fault management*. This component is responsible for maintaining the "health" of the network and for ensuring its smooth and continued operation [27].

Diagnosing faults in a network, may it be wireline or wireless, is a difficult problem because of the interactions between the different network entities and the interactions between faults. Fault management in a multihop wireless network is further complicated by the following factors:

- Such networks are overly prone to link errors with signal propagation affected by fluctuating environmental conditions. This makes the network topology dynamic and unpredictable. Node mobility further aggravates the problems.
- The capacity of such networks is generally limited. Scarcity of resources such as bandwidth and battery power puts a tight constraint on the amount of management traffic overhead the network can tolerate.
- Wireless communication is known to be vulnerable to link attacks [19]. Consequently, such networks are highly susceptible to attacks from malicious parties. The attackers can inject false information to disrupt or interfere with the network management effort. [1]

To address the challenges, we propose a novel fault diagnosis framework that uses on-line trace-driven simulation to detect faults and analyze root causes. Our framework is based on reproducing inside a simulator the events that took place in the network. This framework has the following advantages. First, it is flexible. Since a simulator is often highly customizable and applies to a large class of networks under different environment with appropriate parameter settings, fault diagnosis built on top of it inherits its flexibility. Second, the use of a simulator enables us to capture the complicated interactions within the network, and between the network and the environment, as well as among the different faults. Therefore it allows us to systematically diagnose a wide range and combination of faults. Third, the framework is extensible in that the ability of detecting new faults can be built into the framework by modeling the faults in the simulator independent of the other faults in the system—the interaction between the faults is captured implicitly by the simulator. Fourth, reproducing the network inside a simulator facilitates *what-if analysis*, which offers predictions for network behavior if certain changes were to be applied to the network. The simulator can provide accurate quantitative feedback to the administrator on the performance impact of possible corrective measures.

For the framework to be effective, we need to address two key issues: (i) how to accurately reproduce what happened in the network inside a simulator; (ii) how to build fault diagnosis on top of a simulator to perform root cause analysis.

To address the first issue, we drive an existing network simulator (e.g., Qualnet [7]) with traces obtained from the network being diagnosed. Using real traces removes the dependency on generic theoretical models that may not capture the nuances of the hardware, software and environment of the particular network in question, thus improving the accuracy of the system. We quantify the accuracy of trace-driven simulation through a set of real measurements, and show that for the purpose of fault diagnosis, it is possible to use simulations to reproduce what happened in the real network, after the fact. Further, we address the issue of collecting good quality trace data (in face of measurement errors, nodes supplying false information, and software/hardware errors) by developing a technique to effectively rule out erroneous data from the trace.

To address the second issue, we develop a novel fault diagnosis scheme. It uses the performance data emitted by the trace-driven simulation as the baseline for expected performance from the real network; any significant deviation indicates a potential fault. Further, the scheme takes advantage of the ability to selectively inject a set of suspected faults into a simulator, and perform root-cause analysis by reducing fault diagnosis to a problem of searching a set of faults, such that, when injected, the simulation produces the expected network performance that best matches the observed performance. Finally, we design an efficient search algorithm to determine root causes. We apply the technique to detect and diagnose packet dropping, link congestion, external noise sources, and MAC misbehavior. These faults may have relatively long lasting impact on performance, and are more difficult to detect than on/off failures (e.g., a node turns itself off due to power or battery outage), and therefore is the focus of our research. We demonstrate that our approach is able to diagnose over 10 simultaneous faults of multiple types in a simulated 49-node network with more than 80% coverage and very few false positives. We also implement our approach in a small multihop IEEE 802.11a test-bed and show that it is able to detect random packet dropping and link congestion. (We cannot evaluate diagnosis of the other types of faults in the testbed due to the difficulty in injecting those faults in a controllable manner.)

Summing up, the primary contribution of our paper is as follows: *we propose and evaluate the use of on-line trace-driven simulation as an analytical tool to diagnose faults and test alternative potentially performance enhancing configurations in a complex multihop wireless network.* We believe this is a fundamentally new way of diagnosing and managing faults in an operational network. Along this line, we make the following contributions:

- We propose a fault diagnosis framework built on top of on-line trace-driven simulation for fault detection, isolation, and diagnosis.
- We identify the traces that need to collect to drive simulation, and show that it is possible to use an existing simulator to replay reality through the use of these traces.
- To supply simulation with good quality data, we develop a technique to effectively rule out erroneous data from the trace.
- We propose an efficient search algorithm to diagnose multiple faults in the network, and demonstrates its effectiveness.
- We show how the simulator can be used to carry out what-if analysis and quantify the performance benefit of possible actions on the current network.

The rest of this paper is organized as follows. In Section 2, we discuss the motivation for our research and give a high-level description of our approach. In Section 3, we describe the rationale for using on-line simulations for fault diagnosis. In Section 4, we show the feasibility of using a simulator as a real-time diagnostic tool through a set of simple scenarios. These scenarios are instructive in identifying the data that need to be collected to drive simulations. In Section 5, we propose novel techniques that make use of trace-driven simulation to diagnose faults in multihop wireless networks. We present a prototype implementation of our network monitoring and management module in Section 6. We evaluate the overhead and effectiveness of our diagnosis approach in Section 7. We discuss the limitations of our approach in Section 8. We survey related work in Section 9, and conclude in Section 10.

[1]We do not consider malicious attacks in this paper; security mechanisms for network management that mitigate risks of malicious attacks are outside the scope of this paper.

2. SYSTEM OVERVIEW

Our research is motivated by observing the grassroots interest in community networking [16, 28]. We are interested in providing enabling technology for neighbors to collaboratively form a self-managed community wireless mesh network. With such a network, neighbors can, for example, share a fast Internet gateway in a cost-effective way [22, 6].

In a neighborhood mesh network, most routers reside within a home and are plugged in electric outlets, thereby incurring minimum mobility. The relative stability of such a network makes fault management more important because faults might have lasting influence over the network performance. But the lack of router mobility does not take away the dynamism in network topology because wireless links can be up and down due to environmental changes.

The growth of a community mesh network is organic as users buy and install equipment to join [5], but there is often a lack of a centralized entity responsible for network administration. Therefore, self-manageability and self-healing capabilities, as envisioned in [27], are key to the survival of such networks. It is this vision that inspires us to carry out the research on network fault management for multihop wireless networks.

Our management architecture consists of two types of software modules. An *agent* runs on every router node in the multihop wireless network, gathers information from various protocol layers and from the router's wireless network card, and reports this information to a management server, called *manager*. It is the manager that performs the analysis and takes appropriate actions. Management of the network could be centralized by placing the manager on a single node, or distributed by running the manager on a set of nodes as in [44].

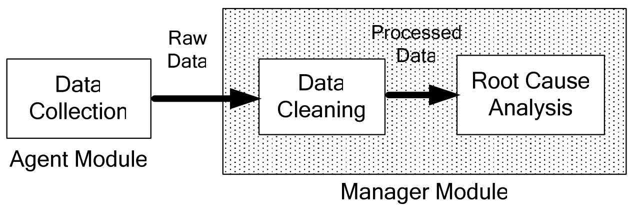


Figure 1: Fault diagnosis process

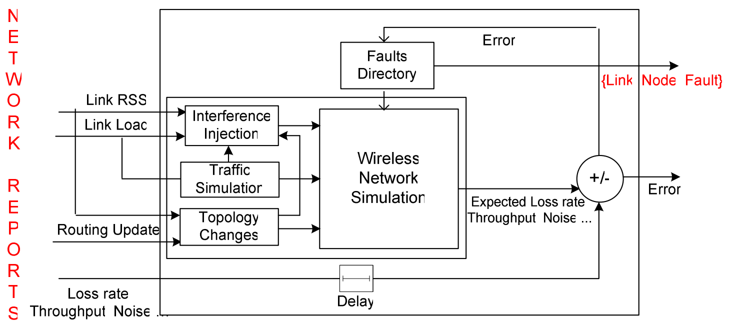


Figure 2: Root cause analysis module

The fault diagnosis process is illustrated in Figure 1. The process starts by agents continuously collecting and transmitting their (local) view of the network's behavior to the manager(s). Examples of the information sent include traffic statistics, received packet signal strength on various links, and re-transmission counts on each link.

It is likely that the data the manager receives from the various agents results in an inconsistent view of the network. Such inconsistencies could be the result of topological and environmental changes, measurement errors, or misbehaving nodes. The *Data Cleaning* module of the manager is used to resolve such inconsistencies before any analysis takes place.

F_1	F_2	F_3	F_4	F_5
2.50 Mbps	0.23 Mbps	2.09 Mbps	0.17 Mbps	2.55 Mbps

Table 1: Throughput of 5 competing flows in Figure 3

Once the inconsistencies have been resolved, the data is fed into the root-cause analysis module for further investigation. We propose the use of on-line trace-driven simulation in this module, shown in Figure 2, to determine the root cause. The analysis module uses the cleaned data to drive on-line simulations and establish the expected performance under the given network configuration and traffic patterns. It detects faults by comparing the expected performance with the observed performance. When discrepancies are observed, the module determines the root cause for the discrepancies by efficiently searching for the set of faults that results in the best match between the simulated and observed network performance.

After the cause for the anomaly has been identified, the analysis module can further simulate several alternative actions under the current traffic pattern and network topology, and suggest appropriate actions to alleviate the faults and enhance overall performance. For example, administrators/users can be notified if the software or hardware are suspected as faulty; the topology can be changed via transmission-power adjustment if poor local connectivity is detected; and rate limiting at the routers can be invoked to alleviate congestion.

3. RATIONALE FOR SIMULATION-BASED DIAGNOSIS

Using a simulator for on-line diagnosis offers benefits over traditional heuristic or theoretical diagnostic techniques for the following reasons.

First, a high quality simulator can provide much better insights into the behavior of the network than any heuristic or theoretical technique can. An operational wireless network is a complex system with many intricate pieces including traffic flows, networking protocols, signal processing algorithms, hardware, RF propagation and, most importantly, the interaction between all of these. Further, the network behavior is governed by the interaction between nodes within range of one another and by noise sources in the vicinity. We know of no heuristic or theoretical technique that captures the behavior of such networks and explains the interaction between all of its different components.

As an example, consider a 7 * 3 grid topology network shown in Figure 3. Assume there are 5 long-lived flows F_1, F_2, F_3, F_4 and F_5 in the network, each with the same amount of traffic to communicate. All adjacent nodes can hear one another and the interference range is twice the communication range. The traffic between nodes A & O interferes with the traffic between nodes C & Q, and similarly traffic between nodes G & U interferes with the traffic between nodes E & S. However, traffic between G & U and between A & O does not interfere with traffic between D & R. Table 1 shows the throughput of the flows when each flow sends CBR traffic at a rate of 11 Mbps. As we can see, the flow F_3 receives much higher throughput than the flows F_2 and F_4.

A simple heuristic may lead the manager to conclude that flow F_3 is unduly getting a larger share of the bandwidth, whereas an on-line trace-driven simulation will conclude that this is normal behavior. This is because the simulation will take into account the link quality and realize that flows F_1 and F_5 are interfering with flows F_2 and F_4, therefore allowing F_3 to open channel more often.

So given the current traffic flows and link qualities, a good simulator is able to advise the manager on what to expect from the network. In other words, it can comment on what constitutes normal behavior. In the example above, the fact that F_3 is getting a greater share of the bandwidth

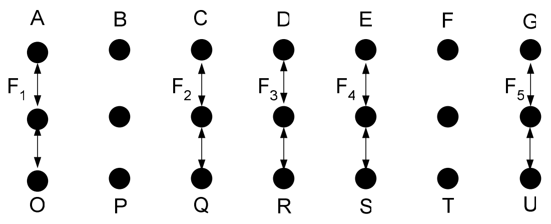


Figure 3: The flow F_3 gets a much higher share of the bandwidth than the flows F_2 and F_4, even though all the flows have the same application-level sending rate. A simple heuristic may conclude that nodes D and R are misbehaving, whereas simulation can correctly determine the observed throughput is expected.

will not be flagged as a fault by the simulator. When the observed behavior is different from what the simulator thinks is normal behavior, the manager can invoke the fault search algorithms to determine the reasons for the deviation.

In addition, while it might be possible to apply the traditional signature-based or rule-based fault diagnosis approach to a particular type of network and under a specific environment and configuration, simple signatures or rules are insufficient to capture the intrinsic complexity for fault diagnosis in general settings. In contrast, a simulator is often highly customizable and applies to a large class of networks under different environment with appropriate parameter settings. Fault diagnosis built on top of such a simulator therefore inherits its generality.

Yet another key advantage of simulation-based approach is the ability to perform what-if analysis. That is, by modifying the settings or performing certain actions in the simulator, a simulator can provide performance data in an imaginary situation. Based on this data, a manager can instruct the agents to take appropriate actions to optimize the network performance. Such what-if analysis is valuable because it is often hard to foresee the consequences of a corrective action due to the interplay of multiple factors in a network. For example, increasing the power of a transmitter might improve the quality of a link, but at the same time create more interference.

Finally, recent advances in simulators for multihop wireless networks, as evidenced in products such as Qualnet, have made the use of a simulator for real-time on-line analysis a reality. This is especially true for the relatively small-scale multihop wireless networks, up to a few hundred nodes, that we intend to manage.

4. BUILDING CONFIDENCE IN SIMULATOR ACCURACY

From the previous sections, it is clear that our system relies on the use of on-line simulations. Consequently, its performance depends on the accuracy of the simulator in reproducing observed network behavior. We now take a close look at how well it performs in matching the behavior of a real network. That is, can we build our research on fault diagnosis and what-if analysis using on-line simulations as the core tool? The answer to this questions cuts to the heart of our work and is necessary for building confidence that we are on the right track.

Factors such as variability of hardware performance, RF environmental conditions, and presence of obstacles make it difficult for simulators to model wireless networks accurately [34]. To illustrate this problem, we conduct a simple experiment as follows.

We study the variation of received signal strength (RSS) with respect to distance for a variety of IEEE 802.11a cards (Cisco AIR-CB20A, Proxim Orinoco 8480-WD, and Netgear WAG511), and plot the results in Figure 4. The experiments are conducted inside a building with walls separating offices every 10 feet; as the distance increases, the number of walls (obstacles) between the two laptops also increases. The signal strength measurements are obtained using the wireless research API (WRAPI) [12]. For comparison, we plot the RSS computed using the two-ray propagation model obtained from Qualnet [7] in Figure 4.

The theoretical model fails to take into account the presence of walls and hence does not estimate the RSS accurately. Accurate modeling and prediction of wireless conditions is a hard problem to solve in its full generality but we show that errors that affect our work can be significantly reduced by replacing theoretical models with data obtained from the nodes within the network.

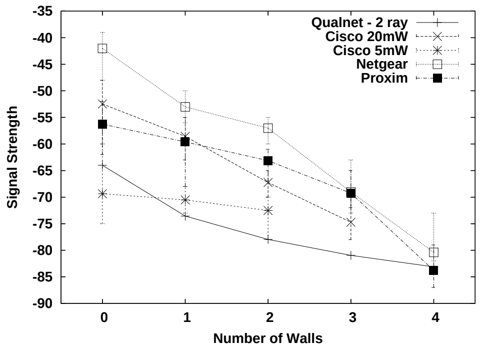


Figure 4: Comparing the simulator's two-ray RF wave propagation model for received signal strength with measurements taken from IEEE 802.11a WLAN cards from different hardware vendors.

There are two main reasons why it is difficult to accurately simulate a real network. First, it is hard to accurately model the physical layer and the RF propagation. Second, the traffic demands on the routers are hard to predict since they depend on human interaction with the network. Fortunately, for the purpose of fault diagnosis, it is not necessary to have predictive models, and it is sufficient to simulate what happened in the network *after the fact*. To do so, we require agents to periodically report information about the link conditions and traffic patterns to the manager. This information is processed and then fed into the simulator. We claim that this approach overcomes the known limitations of RF and traffic modeling in simulators within the context of on-line simulation-based fault diagnosis.

4.1 Baseline Comparison

Next we compare the performance of a real network to that of a simulator for a few simple baseline cases. We design a set of experiments to quantify the accuracy of simulating the overhead of the protocol stack as well as the effect of RF radio interference. The experiments are for the following scenarios:

1. A single one-hop UDP flow (1-hop flow)
2. Two UDP flows within communication range (2 flows - CR)
3. Two UDP flows within interference range (2 flows - IR)
4. One UDP flow with 2 hops where the source and destination are within communication range (2-hop flow -CR)
5. One UDP flow with 2 hops where the source and destination are within interference range but not within communication range. (2-hop flow -IR)

All the throughput measurements are done using Netgear WAG511 cards and Figure 5 summarizes the results. Interestingly, in all cases the throughput from simulations are close to the real measurements. Case (1) shows that Qualnet simulator models the overheads of the protocol stack, such as parity bits, MAC-layer back-off, IEEE 802.11 inter-frame spacing and ACK, and headers accurately. The other scenarios show that the simulator accurately takes into account contention from flows within the interference and communication ranges.

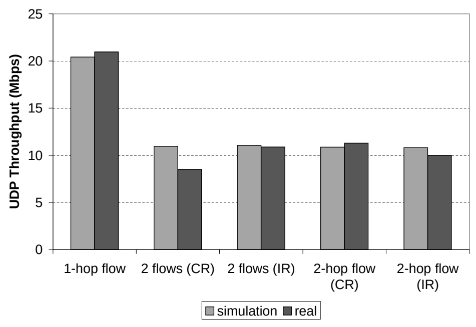


Figure 5: Estimated throughput from the simulator matches measured throughput in a real network when the RF condition of the links is good.

In the scenarios above, data are sent on high-quality wireless links, and almost never get lost due to low signal strength. In our next experiment, we study how received signal strength (RSS) quality affects throughput. We vary the number of walls between the sender and receiver, and plot the UDP throughput for varying packet sizes in Figure 6.

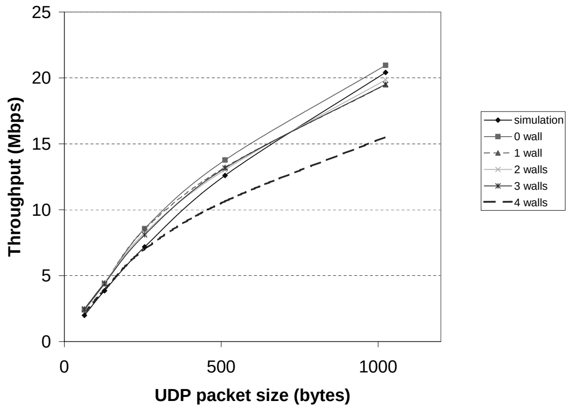


Figure 6: Estimated throughput matches with measured throughput when the RF condition of the links is good, and deviates when the RF condition of the links is poor (1-hop connection).

When the signal quality is good (e.g., when there are fewer than 4 walls in between), the throughput measured matches closely with the estimate from the simulator.

When the signal strength is poor, e.g., when 4 or more walls separate the two laptops, the throughput estimated by the simulator deviates from real measurements. The deviation occurs because the simulator does not take into account the following two factors:

- Accurate packet loss as a function of packet-size, RSS and ambient noise. This function depends on the signal processing hardware and the RF antenna within the wireless card.
- Accurate auto-rate control. On observing a large number of packet retransmissions at the MAC layer, many WLAN cards adjust their sending rate to something that is more appropriate for the conditions. The exact details of the algorithm used to determine a good sending rate differs from cards to cards.

Of the two factors mentioned above, the latter can be taken care of as follows. If auto-rate is in use, we can again employ the idea of trace-driven simulation as follows: we collect the rate at which the wireless card is operating, and provide the reported rate directly into the simulator (instead of having the simulator adapt in its own way). When auto-rate is not used (e.g., researchers [21] have shown that auto-rate is often undesirable when considering aggregate performance and therefore should be turned off), the data rate is known.

# walls	loss rate	measured throughput	simulated throughput
4	11.0%	15.52 Mbps	15.94 Mbps
5	7.01%	12.56 Mbps	14.01 Mbps
6	3.42%	12.97 Mbps	11.55 Mbps

Table 2: Estimated and measured throughput match, when we compensate the loss rates due to poor RF in the real measurements by seeding the corresponding link in a simulator with an equivalent loss rate.

The first issue is much harder to address because it may not be possible to accurately simulate the physical layer. One possible way to address this issue is through offline analysis. We can calibrate the wireless cards under different scenarios and create a database to associate environmental factors with expected performance. For example, we can carry out real measurements under different signal strength and noise level to create a mapping from signal strength and noise to loss rate. Using such a table in simulations allows us to distinguish between losses caused by collisions from losses caused by low signal strength or by external source of interference like a cordless phone or a microwave oven.

Based on this idea, in our experiment, we collect another set of traces in which we slowly send out packets so that most losses are caused by poor signal (instead of congestion). We also place packet sniffers near both the sender and receiver, and derive the loss rate from the packet-level trace. [2] Then we take into the account the loss rate due to poor signal by seeding the wireless link in a simulator with a Bernoulli loss rate that matches the loss rate in real traces.

We find that after taking into account the impact of poor signal, the throughput from simulation matches closely with real measurements as shown in Table 2. Note that the loss rate and measured throughput do not monotonically decrease with the signal strength due to the effect of auto-rate. (The driver of the wireless cards we use does not allow us to disable auto-rate.) Note that even though the match is not perfect, we do not expect this to be a problem in practice for the following reason. Many routing protocols try to avoid the use of poor quality links by employing some appropriate routing metric (loss rate, signal strength, ETX [29], smoothed roundtrip time [17]). In fact, we resort to using poor quality links for routing traffic only when certain parts of the mesh network have poor connectivity to the rest of the network. In a well-engineered network, not many nodes depend on such bad links for routing.

4.2 Stability of Channel Conditions

So far, we have shown that with the help of trace collection a simulator is able to accurately mimic reality. However, one question remains: how rapidly do conditions change and how often must we collect a trace? When the channel conditions are fluctuating very rapidly, collecting an accurate trace and shipping the trace to the manager may be difficult and costly. Figure 7 shows the temporal fluctuation in RSS under the same measurement setup as described above. The duration of each trace is around 10 minutes. As expected, the RSS fluctuates over time. Fortunately, from our diagnosis perspective, the magnitude of the fluctuation is not significant, and the relative quality of the signals across different numbers of walls remains stable. This suggests that when the environ-

[2] Note that packet-level traces from sniffers are necessary to derive link loss rate because due to MAC layer retransmissions, the loss rate seen at the application layer can be quite different from the true link loss rate. We are interested in true link loss rate, and sniffers enable us to see all packets including retransmissions, from which we can compute the true link loss rate.

ment is generally static, the nodes may report only the average and standard deviation of the RSS to the manager every few minutes (e.g., 1 - 2 minutes).

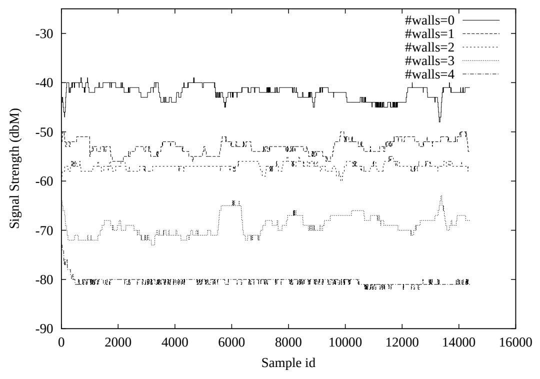


Figure 7: In good environmental conditions, received signal strength remains stable over time.

4.3 Remarks

In this section, we have shown that even though simulating a wireless network accurately is a hard problem, for the purpose of fault diagnosis, we can use trace-based simulations to reproduce what happened in the real network, after the fact.

To substantiate this claim, we look at a number of simple scenarios and show that the throughput obtained from the simulator matches reality after taking into account information from real traces. We require very little data collected at the nodes, at a fairly low time-granularity. While these results are positive and encouraging, we have only looked at some simple examples in this section, mainly for the building confidence in trace-driven simulation before we proceed further. Also, we have only looked at the physical and MAC layers so far. We will address simulation of traffic and routing protocols, and evaluate their performance in Section 5 and Section 7, respectively.

5. FAULT ISOLATION AND DIAGNOSIS

We now present our simulation-based diagnosis approach. Our high-level idea is to re-create the environment that resembles the real network inside a simulator. To find the root cause, we *search over a fault space* to determine which fault or set of faults can re-produce performance similar to what has been observed in the real network.

In Section 5.1 we extend the trace-driven simulation ideas presented in Section 4 to reproduce network topology, routing behavior, and traffic pattern observed in the real network.

Using trace-driven simulation as a building block, we then develop a diagnosis algorithm to find root-causes for faults. The algorithm first establishes the expected performance under a given set of faults. Then based on the difference between the expected and observed performance, it efficiently searches over the fault space to re-produce the observed symptoms. This algorithm can not only diagnose multiple faults of the same type, but also perform well in the presence of multiple types of faults.

Finally we address the issue of how to diagnose faults when the trace data used to drive simulation contains errors. This is a practical problem since data in the real world is never perfect for a variety of reasons, such as measurement errors, nodes supplying false information, and software/hardware errors. To this end, we develop a technique to effectively rule out erroneous data from the trace so that we can use good quality trace data to drive simulation-based fault diagnosis.

5.1 Trace-Driven Simulation

Taking advantage of real trace data enables us to accurately capture the current environment and examine the effects of a given set of faults in the current network.

5.1.1 Trace Data Collection

We collect the following sets of data as input to a simulator:

Network topology: Each node reports its neighbor and routing tables. To be efficient, only changes in neighbors or routes are reported. Routing data is used to drive route simulation described in Section 5.1.2.

Traffic statistics: Each node maintains counters for the volume of traffic sent to and received from its immediate neighbors. This data drives traffic simulation described in Section 5.1.2.

Physical medium: Each node reports the signal strength of the wireless links to its neighbors. According to the traces collected from our test-bed, we observe the signal strength is relatively stable over tens of seconds. Slight variations in signal strength with time can be captured accurately through the time average, standard deviation, and other statistical aggregates.

Network performance: To detect anomaly, we compare the observed network performance with the expected performance from simulation. Network performance includes both link performance and end-to-end performance, both of which can be measured through a variety of metrics, such as packet loss rate, delay, and throughput. In our work, we focus on link level performance.

Data collection consists of two steps: collecting raw performance data on a local node and distributing the data to collection points for analysis. For local data collection, we can use a variety of tools, such as WRAPI [12], Native 802.11 [15], SNMP [25], and packet sniffers (e.g., Airopeek [11], tcpdump [46]).

Distributing the data to a manager introduces overhead. In Section 7.1, we quantify this overhead, and show it is low and has little impact on the data traffic in the network. Moreover, it is possible to further reduce the overhead using compression, delta encoding, multicast, and adaptive changes of the time scale and spatial scope of distribution. For example, in the normal situation, a minimum set of information is collected and exchanged. Once the need arises for more thorough monitoring (e.g., when the information being collected indicates anomaly), then the manager requests more information and increases the frequency of data collection for the subset of the nodes that require intensive monitoring.

5.1.2 Simulation Methodology

We classify the characteristics of the network that need to be matched in the simulator into the following four categories: (i) traffic load, (ii) routing, (iii) wireless signal, and (iv) faults. Below we describe how to simulate each of these components.

Traffic Load Simulation: A key step to replicating the real network inside a simulator is to re-create the same traffic pattern. One possible approach is to simulate end-to-end application demands. However, this is not scalable since there can be potentially $N * N$ demands for an N-node network. Moreover, given the heterogeneity of application demands and the use of different transport layers, such as TCP, UDP, and RTP, it is challenging to describe, obtain, and simulate end-to-end demands in a simulator.

For scalability and avoiding the need for obtaining end-to-end demands, we use link-based traffic simulation. Our high-level idea is to adjust application-level sending rate at each link to match the observed link-level traffic counts. Doing this abstracts away higher layers such as the transport and the application layer, and allows us to concentrate only on packet size and traffic rate. However, matching the sending rate on a per-link basis in the simulator is non-trivial because we can only adjust the application-level sending rate, and have to obey the medium access control (MAC) protocol. This implies that we cannot directly control sending rate on a link. For example, when we set the application sending rate of a link to be 1 Mbps, the actual sending rate (on the air) can be lower due to back-off at the MAC layer, or higher due to MAC level retransmission. The issue is further complicated by interference, which introduces inter-dependency between sending rates on different links.

To address this issue, we use the following *iterative search* to determine the sending rate at each link. We try two search strategies: (i) multiplicative increase and multiplicative decrease, and (ii) additive increase and additive decrease. As shown in Figure 8, each link individually tries to reduce the difference between the current sending rate in the simulator and the actual sending rate in the real network. We introduce a parameter α, where $\alpha \leq 1$, to dampen oscillation (In our evaluation, we use $\alpha = 0.5$). The process iterates until either the rate becomes close enough to the target rate (denoted as $targetMacSent$) or the maximum number of iterations is reached. We use multiplicative increase and multiplicative decrease in our evaluation, and plan to compare it with additive increase and additive decrease in the future.

```
while ($not\ converged$ and $i < maxIterations$)
    $i = i + 1$;
    if ($option == multiplicative$)
      foreach link($j$)
        $prevRatio = targetMacSent(j)/simMacSent(j)$;
        $currRatio = (1 - \alpha) + \alpha * prevRatio$;
        $simAppSent(j) = prevAppSent(j) * currRatio$;
    else // additive
      foreach link($j$)
        $diff = targetMacSent(j) - prevMacSent(j)$;
        $currAppSent(j) = prevAppSent(j) + \alpha * diff$;
    run simulation using $simAppSent$ as input
    determine $simMacSent$ for all links from simulation results
    $converged = isConverge(simMacSent, targetMacSent)$
```

Figure 8: Searching for the application-level sending rate using either multiplicative increase multiplicative decrease or additive increase additive decrease.

Route Simulation: Routing plays an important role in multi-hop wireless networks. One possible method to handle routing is to run the same routing protocol as in the real network inside the simulator. But simulating a routing protocol behavior precisely as in a real network is a daunting task for the following reason: the operation of a routing protocol depends on some low level details such as the delay experienced by certain control packets used to estimate routing metrics, or the timing of packets in the propagation of a flood, etc. It is unreasonable to expect a trace-driven simulation to match the network at this level of detail.

To circumvent the problem of accurately simulating the routing protocol, we use the actual routes taken by packets as input to the simulator. When routes do not fluctuate frequently, we only need to keep track of the routing changes (instead of collecting routes on a packet-by-packet basis) at the manager. For this purpose, we have implemented a trace-driven route simulation module in Qualnet. This module takes the routing updates and their timestamps as input, and then ensures that the packets in the simulator follow the same route as in the real network.

Wireless Signal: Signal strength has a very important impact on wireless network performance. As discussed in Section 4, due to variations across different wireless cards and environments, it is hard to come up with a general propagation model to capture all the factors. To address this issue, we drive simulation using the real measurement of signal strength, which can be easily obtained using newer generation wireless cards (e.g., Native 802.11 [15]).

Fault Injection: To examine the impact of faults on the network, we implement the ability to inject different types of faults into the simulator, namely (i) random packet dropping, (ii) external noise sources, and (iii) MAC misbehavior [38].

- Random packet dropping: a misbehaving node randomly drops traffic from one of its neighbors. This can occur due to hardware/software errors, buffer overflow, and/or malicious drops. The ability to detect such random dropping is useful, since it allows us to differentiate losses caused by end hosts from losses caused by the network.
- External noise sources: we support the ability to inject external noise sources in the network.
- MAC misbehavior: a faulty node does not follow the MAC etiquette and obtains an unfair share of the channel bandwidth. For example, in IEEE 802.11 [38], a faulty node can choose a smaller contention window (CW) to send traffic more aggressively [35].

In addition, we also generate link congestion by putting a high load on the network. Unlike the other types of faults, link congestion is implicitly captured by the traffic statistics gathered from each node. Therefore trace-driven simulation can directly assess the impact of link congestion. For the other three types of faults, we apply the algorithm described in Section 5.2 to diagnose them.

5.2 Fault Diagnosis Algorithm

We now describe an algorithm to systematically diagnose root causes for failures and performance problems.

General approach: Applying simulations to fault diagnosis enables us to reduce the original diagnosis problem to the problem of searching for a set of faults such that their injection results in an expected performance that matches well with observed performance. More formally, given a network settings, NS, our goal is to find $FaultSet$ such that $SimPerf(NS, FaultSet) \approx ObservedPerf$, where the performance is a function value, which can be quantified using different metrics. It is clear that the search space is high-dimensional due to many combinations of faults. To make the search efficient, we take advantage of the fact that different types of faults often change one or few metrics. For example, random dropping only affects link loss rate, but not the other metrics. Therefore we can use the metrics in which the observed and expected performance have significant difference to guide our search. Below we introduce our algorithm.

Initial diagnosis: We start by considering a simple case where all faults are of the same type, and the faults do not have strong interactions. We will later extend the algorithm to handle more general cases, where we have multiple types of faults, or faults that interact with each other.

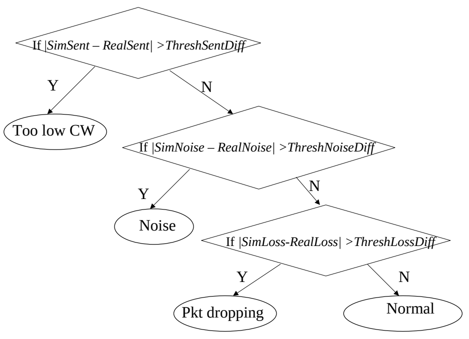
For ease of description, we use the following three types of faults as examples: random packet dropping, random noise, MAC misbehavior, but the same methodology can be extended to handle other types of faults once the symptoms of the fault are identified.

As shown in Figure 9, we use trace-driven simulation, fed with current network settings, to establish the expected performance. We consider large deviations from the expected performance as anomalies, and look for the faults that can explain the discrepancy. In particular, based on the difference between the expected performance and observed performance, we first determine the type of faults using a decision tree as shown in Figure 10. Due to many factors, simulated performance is unlikely to be identical with the observed performance even in the absence of faults. Therefore we conclude that there are anomalies only when the difference exceeds a threshold. The fault classification scheme takes advantage of the fact that different faults exhibit different behaviors. While their behaviors are not completely non-overlapping (e.g., both noise sources and random dropping increase loss rates; and lowering CW increases the traffic and hence increases noise caused by interference), we can categorize the faults by checking the differentiating component first. For example, external noise sources increase noise experienced by its neighboring nodes, but do not increase the sending rates of any node, and therefore can be differentiated from MAC misbehavior and random dropping.

After the fault type is determined, we then locate the faults by finding the set of nodes and links that have large difference between the observed and expected performance. The fault type determines what metric is used to quantify the performance difference. For instance, for random packet dropping, we identify dropping links by finding links with large difference between the expected and observed loss rates.

```
1) Let $NS$ denote the network settings
       (i.e., signal strength, traffic statistics, routing table)
   Let $RealPerf$ denote the real network performance
2) $FaultSet = \{\}$
3) Predict $SimPerf$ by running simulation with input $(NS, FaultSet)$
4) if $|Diff(SimPerf, RealPerf)| > threshold$
      determine the fault type $ft$ using the decision tree shown in Fig. 10
      for each link or node $i$
          if $(|Diff_{ft}(SimPerf(i), RealPerf(i))| > threshold)$
             add $fault(ft, i)$
```

Figure 9: Initial diagnosis: one pass diagnosis algorithm



If |SimSent – RealSent| >ThreshSentDiff
Y
N
Too low CW
If |SimNoise – RealNoise| >ThreshNoiseDiff
Y
N
Noise
If |SimLoss-RealLoss| >ThreshLossDiff
Y
N
Pkt dropping
Normal

Figure 10: An algorithm to determine the type of faults

The algorithm: In general, we may have multiple types of faults interacting with each other. Even when all the faults are of the same type, they may still interact, and their interaction may make the above one pass diagnosis insufficient. To address these challenges, we develop an interactive diagnosis algorithm, as shown in Figure 11, to find root causes.

The algorithm consists of two stages: (i) initial diagnosis stage, and (ii) iterative refinements. During the initial diagnosis stage, we apply the one-pass diagnosis algorithm described above to come up with the initial set of faults; then during the second stage, we iteratively refine the fault set by (i) adjusting the magnitude of the faults that have been already inserted into the fault set, and (ii) adding a new fault to the set if necessary. We iterate the process until the change in fault set is negligible (i.e., the fault types and locations do not change, and the magnitudes of the faults change very little).

We use an iterative approach to search for the magnitudes of the faults. At a high level, the approach is similar to the link-based simulation, described in Section 5.1.2, where we use the difference between the target and current values as a feedback to progressively move towards the target. In more details, during each iteration, we first estimate the expected network performance under the existing fault set. Then we compute the difference between simulated network performance (under the existing fault set) with real performance. Next we translate the difference in performance into change in faults' magnitudes using a function, $g()$. This function essentially maps the impact of a fault into its magnitude. For example, under the random dropping fault, $g()$ function is the identity function, since the difference in a link's loss rate can be directly mapped to a change in dropping rate on a link (fault's magnitude); under the external noise fault, $g()$ is a propagation function of a noise signal. After updating the faults with new magnitudes, we remove the faults whose magnitudes are too small.

In addition to searching for the correct magnitudes of the faults, we also iteratively refine the membership of the fault set by finding new faults that can explain large difference between expected and observed performance. To control false positives, during each iteration we only add the fault that can explain the largest mismatch.

```
1) Let $NS$ denote the network settings
       (i.e., signal strength, traffic statistics, and routing tables)
   Let $RealPerf$ denote the real network performance
2) $FaultSet = \{\}$
3) Predict $SimPerf$ by running simulation with input $(NS, FaultSet)$
4) if $|Diff(SimPerf, RealPerf)| > threshold$
      go to 5)
   else
      go to 7)
5) Initial diagnosis:
   initialize $FaultSet$ by applying the algorithm in Fig. 9
6) while (not converged)
      a) adjusting fault magnitude
         for each fault type $ft$ in $FaultSet$ (in the order of decision tree in Fig. 10)
            for each fault $i$ in $(FaultSet, ft)$
               $magnitude(i) -= g(Diff_{ft}(SimPerf(i), RealPerf(i), t))$
               if $(|magnitude(i)| < threshold)$
                  delete the fault $(ft, i)$
      b) adding new candidate faults if necessary
         foreach fault type $ft$ (in the order of decision tree in Fig. 10)
            i) find a fault $i$ s.t. it is not in $FaultSet$
               and has the largest $|Diff_{ft}(SimPerf(i), RealPerf(i))|$
            ii) if $(|Diff_{ft}(SimPerf(i), RealPerf(i))| > threshold)$
                  add $(ft, i)$ to $FaultSet$ with
                  $magnitude(i) = g(Diff_{ft}(SimPerf(i), RealPerf(i)))$
      c) simulate
7) Report $FaultSet$
```

Figure 11: A complete diagnosis algorithm: diagnose faults of possibly multiple types

5.3 Dealing with Imperfect Data

In the previous sections, we describe how to diagnose faults by using trace data to drive online simulation. In practice, the raw trace data collected may contain errors for various reasons as mentioned earlier. Therefore we need to clean the raw data before feeding it to a simulator for fault diagnosis.

To facilitate the data cleaning process, we introduce *neighbor monitoring*, in which each node reports performance and traffic statistics not only for its incoming/outgoing links, but also for other links within its communication range. Such information is available when a node is in promiscuous mode.

Due to neighborhood monitoring, multiple reports from different sources are likely to be submitted for each link. The redundant reports can be used to detect inconsistency. Assuming that the number of misbehaving nodes is small, our scheme identifies the misbehaving nodes as the minimum set of nodes that can explain the discrepancy in the reports. Based on the insight, we develop the following scheme.

In our scheme, a sender i reports the number of packets sent and the number of MAC-level acknowledgements received for a directed link l as $(sent_i(l), ack_i(l))$; a receiver j reports the number of packets received on the link as $recv_j(l)$; in addition, a sender or receiver's immediate neighbor k also reports the number of packets and MAC-level acknowledgement it observes sent or received on the link as $(sent_k(l)$, $recv_k(l), ack_k(l))$. An inconsistency in the reports is defined as one of the following cases.

1. The number of packets received on a link, as reported by its destination, is noticeably larger than the number of packets sent on the same link, as reported by its source. That is, for the link l from node i to node j, and given a threshold t:

$$recv_j(l) - sent_i(l) > t$$

2. The number of MAC-level acknowledgments on a link, as reported by its source, does not match the number of packets received on that link, as reported by its destination. That is, for the link l from node i to node j, and given a threshold t:

$$\mid ack_i(l) - recv_j(l) \mid > t$$

3. The number of packets received on a link, as reported by a neighbor of its destination, is noticeably larger than the number of packets sent on the same link, as reported by its source. That is, for the link l from node i to node j, j's neighbor k, and given a threshold t:

$$recv_k(l) - sent_i(l) > t$$

4. The number of packets sent on a link, as reported by a neighbor of its source, is noticeably larger than the number of packets sent on the same link, as reported by its source. That is, for the link l from node i to node j, i's neighbor k, and given a threshold t:

$$sent_k(l) - sent_i(l) > t$$

Since nodes do not send their reports strictly synchronously, we need to use a threshold $t > 0$ to mask the resulting discrepancies. Note that in the absence of inconsistent reports, the above constraints cannot be violated as a result of lossy links.

We then construct an *inconsistency graph* as follows. For each pair of nodes whose reports are identified as inconsistent, we add them to the inconsistency graph, if they are not already in the graph, with an edge connecting between the two. Based on the assumption that most nodes send reliable reports, our goal is to find the smallest set of nodes that can explain all the inconsistency observed. This can be achieved by finding the smallest set of vertices that covers the graph, where the identified vertices represent the misbehaving nodes.

This is essentially the minimum vertex cover problem [30], which is known to be NP-hard. We apply a greedy algorithm, which iteratively picks and removes the node with the highest degree and its incident edges from the current inconsistency graph until no edges are left.

History of traffic reports can be used to further improve the accuracy of inconsistency detection. For example, we can continuously update the inconsistency graph with new reports without deleting previous information, and then apply the same greedy algorithm to identify misbehaving nodes.

6. SYSTEM IMPLEMENTATION

We have implemented a prototype of network monitoring and management module on Windows XP platform. We present the components of the prototype implementation, the design principles, and its features in this section.

Our current prototype consists of two separate components: *agents* and *managers*. An agent runs on every wireless node, and reports local information periodically or on-demand. A manager collects relevant information from agents and analyzes the information.

Simplicity and extensibility are two main design principles we follow. The information gathered and propagated for monitoring and management is cast into performance counters supported on Windows. Performance counters are essentially (name, value) pairs grouped by categories.

This framework is easily extensible. Adding to the information being monitored simply involves creating a new category of performance counters and writing a module that updates the performance counter values as the information changes. Performance data related to TCP, UDP, IP, and WRAPI have all been incorporated into the framework with little work.

Values in these performance counters are not always read-only. Writable counters offer a way for an authorized manager to change the values and influence the behavior of a node in order to fix problems or initiate experiments remotely.

Each manager is also equipped with a graphical user interface (GUI) to interact with network administrators. The GUI allows an administrator to visualize the network as well as to issue management requests.

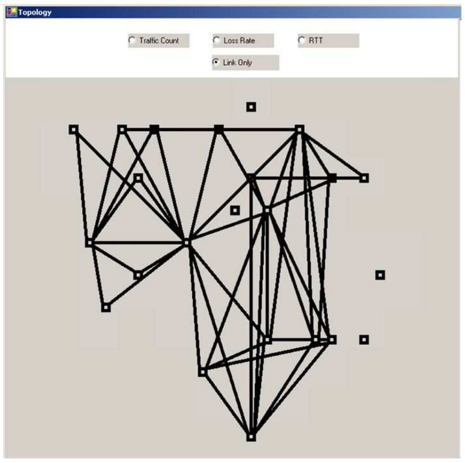


Figure 12: A manager GUI displays the topology for a multihop wireless network testbed.

Figure 12 shows a snapshot from the manager window with agents deployed over a testbed of 23 nodes. The manager can display the topology based on the relative coordinates of the nodes either directly obtained (as in the case shown in Figure 12) or inferred. The GUI also allows the administrator to zoom in on a particular part of the network for more detailed information and to click on a link to look at all the performance data about that link in a table format.

The manager is also connected to the back-end simulator. The information collected will be processed and then converted into a script to drive the simulation and produce fault diagnosis results.

In this paper, we report results from a multihop wireless testbed built using IEEE 802.11a cards, although our design should work equally well on other types of wireless networks. We choose IEEE 802.11a to avoid cross interference with the 2.4 GHz IEEE 802.11b infrastructure network deployed in our organization.

The capability of the network monitoring and management depends heavily on the information available for collection. We have seen welcoming trends in both wireless NICs and the standardization efforts to expose performance data and control at the physical and MAC layers. For example, in order to reduce the cost and commoditize wireless cards, IEEE 802.11 WLAN NIC vendors [1, 8] are minimizing the functionality of the code residing inside the micro-controller of their NICs. These next generation wireless NICs, popularly known as Native 802.11 NICs [15], provide a rich set of networking information to the operating system. In addition to the Native 802.11 NICs, there is a new initiative within the IEEE standardization body, called the IEEE P802.11k—Radio Resource Measurement Study Group [4], whose charter is to evaluate the inclusion of "hooks" in the MAC MIB to provide PHY and MAC measurements for the use of external (above the MAC) network management. Unfortunately, there are no IEEE 802.11a cards available in market that support Native 802.11 functionalities. To overcome this difficulty, we resorted to using a commercially available network sniffer called Airopeek [11]. But once cards with Native 802.11 functionality are available, we can very quickly integrate the newly available performance counters into our extensible framework for data collection.

7. EVALUATION

In this section, we present our evaluation results. We first quantify the network overhead introduced by data collection and show its impact on the overall performance. Then we evaluate the effectiveness of our diagnosis techniques and inconsistency detection scheme. We use simulations for most of our evaluation, since it enables us to inject different types of faults in a controlled and repeatable manner, and quantify the accuracy of our approach. When fault diagnosis is evaluated in a simulator, we diagnose traces collected from simulation runs that have injected faults. In simulation-based evaluation, we mainly focus on validating the following two important components in our approach: trace-driven simulation (in particular, link-based traffic simulation and route simulation) and fault diagnosis algorithm. Finally we report our experience of applying the approach to a small-scale testbed. Even though the results we could obtain from the test-bed were limited by our inability to inject some types of faults like external noise and MAC misbehavior in a controlled fashion and the lack of transparency in the current generation of drivers for wireless NICs, they demonstrate the feasibility of on-line simulations in a real system. Unless otherwise specified, all the simulation results in this section are based on IEEE 802.11b.

7.1 Data Collection Overhead

For data collection, every node not only collects information locally, but also deliver the data to the manager. We evaluate the overhead involved in having all nodes in a network report the information to a manager.

As described in Section 5.1.1, we collect the following information: routing changes, traffic counters (i.e., the number of packets sent and received), and signal strength. Since the primary goal of this section is to demonstrate the feasibility of distributing all the data to a manager at a modest cost, we only make *conservative assumptions* about the sizes of the packets used. Further optimization is possible as described in Section 5.1.1.

In our evaluation, we place nodes randomly in a square, and the manager is chosen at random amongst the nodes. We keep the average number of neighbors around 6 as we increase the network size. On average, a node takes 1 to 5 hops to reach the manager. Based on the configuration, a conservative estimate of routing update size is 20 bytes, assuming each node address is 4 bytes. The size of link report, which includes traffic counters and signal strength, depends on whether redundant information is sent for consistency check. Therefore we consider two scenarios: when each link is reported by one node (i.e., without data cleaning) and when each link is reported by all the observers (including the sender and receiver) to allow us to check for consistency (i.e., with data cleaning). Since every node has around 6 immediate neighbors, a conservative estimate of a link report is 72 bytes when no redundant link data is sent, and is 312 bytes when redundant link data is sent.

In Figure 13, we plot the overhead of data gathering as a function of the network size. As it shows, even with data cleaning using 60 second report interval, the overhead remains low, less than 800 bits/s/node. Moreover, the overhead does not increase much as the network size increases.

Figure 14 shows the performance of FTP flows in the network with and without the data collection traffic. Ten simultaneous FTP flows are started from random sources to the manager. The graph shows the average throughput of these flows on the y-axis. As we can see, the data collection traffic (with and without sending redundant link data) has little effect on the application traffic in the network.

Summary: In this section, we evaluate the overhead of collecting traces, which will be used as inputs to diagnose faults in a network. We show that the data collection overhead is low and has little effect on application traffic in the network. Therefore it is feasible to use trace-driven simulation for fault diagnosis.

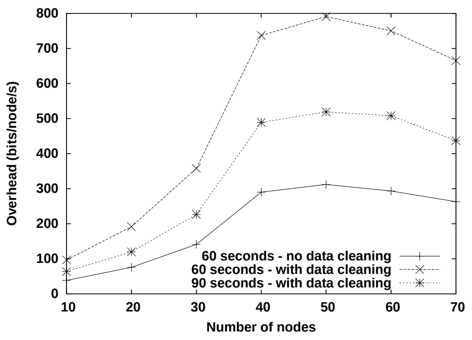


Figure 13: Management traffic overhead

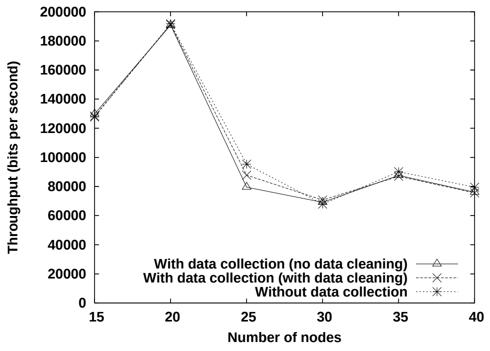


Figure 14: Effect of overhead on throughput

7.2 Evaluation of Fault Diagnosis through Simulations

In this section, we evaluate our fault diagnosis approach through simulations in Qualnet.

7.2.1 *Diagnosing one or more faults of possibly different types*

Our general methodology of using simulation to evaluate fault diagnosis is as follows. We artificially inject a set of faults into a network, and obtain a set of faulty traces, which includes network topology, link load and routing updates. We then feed these faulty traces into the fault diagnosis module to infer root causes, and quantify the diagnosis accuracy by comparing the inferred fault set with the fault set originally injected.

We use both grid topologies and random topologies for our evaluation. In a grid topology, only nodes horizontally or vertically adjacent can directly communicate with each other, whereas in random topologies, nodes are randomly placed in a region. To challenge our diagnosis scheme, we put a high load on the network by randomly picking 25 pairs of nodes to send one-way constant bit rate (CBR) traffic at a rate of 1 Mbps. Under this load, the links in the network have significant congestion loss, which makes diagnosis even harder. For example, differentiating loss caused by random packet dropping is more difficult when there is significant congestion loss. Correct identification of dropping links also implies reasonable assessment of congestion loss. In addition, we randomly select a varying number of nodes to exhibit one or more faults of the following types: random packet dropping, external noise, and MAC misbehavior. For a given number of faults and its composition, we conduct three random runs, which have different traffic patterns and fault locations. We evaluate how accurate our fault diagnosis algorithm, described in Section 5.2, can locate the faults. The diagnosis process takes on the time scale of tens of minutes. (The exact time depends on the size of topologies, the number of faults, and duration of the faulty traces.) Such diagnosis time scale is acceptable for diagnosing long-term performance problems. Moreover, the efficiency can be significantly improved through code optimization.

We use coverage and false positive to quantify the accuracy of fault detection, where coverage represents the fraction of faulty locations that are correctly identified, and false positive is the number of (non-faulty) locations incorrectly identified as faulty divided by the total number of true faults. We consider a fault is correctly identified when both its type and its location are correct. For random dropping and external noise sources, we also compare the inferred faults' magnitudes with their true magnitudes.

Detecting random packet dropping: We start by evaluating how accurately we can detect random packet dropping. In our evaluation, we select a varying number of nodes to intentionally drop packets at a random dropping rate between 0 - 100%. We vary the number of such misbehaving nodes from 1 to 6.

We apply the diagnosis algorithm, which first uses trace-driven simulation to estimate the expected performance (i.e., noise level, throughput, and loss rates) in the current network. Since we observe a significant difference in loss rates, but not in the other two metrics, we suspect that there is random packet dropping on these links. We locate the random dropping links by identifying links whose loss rates are significantly higher than their expected loss rates. We use 15% as a threshold so that links whose difference between expected and observed loss rates exceed 15% are considered as packet dropping links. We then inject the faults into the simulator, and find that this significantly reduces the difference between the simulated and observed performance.

Figure 15 shows the accuracy of detecting random dropping links in a 5×5 grid topology. Note that some of the faulty links do not carry enough traffic to meaningfully compute loss rates. In our evaluation, we use 250 packets as a threshold so that only for the links that send over 250 packets, loss rates are computed. We consider a faulty link sending less than a threshold number of packets as a no effect fault since it drops only a small number of packets.[3] As Figure 15 shows, under our diagnosis scheme, in most cases less than 20% effective faulty links are left un-detected. The false positive (not shown) is 0 except for one case, which has one false positive. Moreover the accuracy does not degrade as the number of faults increases. When we compare the difference between the inferred and true dropping rates, we find the inference error, computed as $\sum_i |infer_i - true_i| / \sum_i true_i$, is within 30%. This error rate is related to the threshold used to determine if the iteration has converged. In our simulations, we consider an iteration converges when changes in loss rates are all within 15%. We can further reduce the inference error by using a smaller threshold at a cost of longer running time. Also, in many cases it suffices to know where packet dropping occurs without knowing precise dropping rates.

Detecting external noise sources: Next we evaluate the accuracy of detecting external noise sources. We randomly select a varying number of nodes to generate ambient noise at 1.1e-8 mW. We again use the trace-driven simulation to estimate the expected performance under the current traffic and network topology when there is no noise source. Note that simulation is necessary to determine the expected noise level, because the noise experienced by a node consists of both ambient noise and noise due to interfering traffic; accurate simulation of network traffic is needed to determine the amount of noise contributed by interfering traffic. The diagnosis algorithm detects a significant difference (e.g., over 5e-9mW) in noise level at some nodes, and conjectures these nodes

[3]These faulty links may have impact on route selection. That is, due to its high dropping rate, it is not selected to route much traffic. In this paper, we focus on diagnosing faults based on data traffic. As part of our future work, we plan to investigate how to diagnose faults based on selected routes.

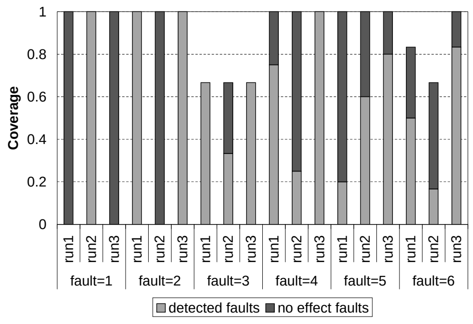


Figure 15: Accuracy of detecting random dropping links in a 5×5 grid topology

generate extra noise. It then injects noise at these nodes with magnitude derived from the difference between expected and observed noise level to the simulator. After noise injection, it sees a close match between the observed and expected performance, and hence concludes that the network has the above faults. In most cases, the algorithm converges within two iterations. That is, after injecting the fault set identified in the first iteration, the expected performance is close to observed performance.

Figure 16 shows the accuracy of detecting noise generating sources in a 5×5 grid topology. As we can see, in all cases noise sources are correctly identified with only 1 to 2 false positives at most. We also compare the inferred magnitudes of noises with their true magnitudes, and find the inference error, computed as $\sum_i |infer_i - true_i| / \sum_i true_i$, is within 4%.

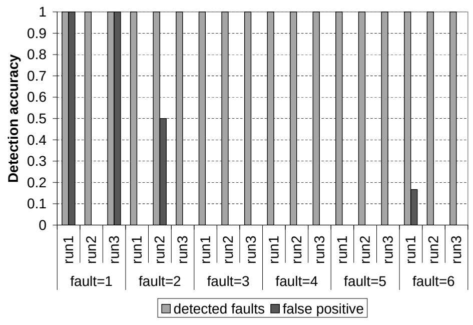


Figure 16: Accuracy of detecting external noise sources in a 5×5 grid topology

Detecting MAC misbehavior: Now we evaluate the accuracy of detecting MAC misbehavior. In our evaluation, we consider one implementation of MAC misbehavior. But since our diagnosis scheme is to detect unusually aggressive senders, it is general enough for detecting other types of implementations of MAC misbehavior that exhibit similar symptoms. In our implementation, a faulty node alters its minimum and maximum MAC contention window in 802.11 (CWMin and CWMax) to be only half of the normal values. The faulty node continues to obey the CW updating rules (i.e., when transmission is successful, CW = CWMin, and when a node has to retransmit, CW = min((CW+1)*2-1, CWMax)). However since its CWMin and CWMax are both half of the normal, its CW is usually around half of the other nodes'. As a result, it transmits more aggressively than the other nodes. As one would expect, the advantage of using a lower CW is significant when network load is high. Hence we evaluate our detection scheme under a high load.

In our diagnosis, we use the trace-driven simulation to estimate the expected performance under the current traffic and network topology, and detect a significant discrepancy in throughput (e.g., the ratio between observed and expected throughput exceeds 1.25) on certain links. Therefore we suspect the corresponding senders have altered their CW.

After injecting the suspected faults, we see a close match between the simulated and observed performance. In almost all cases, the algorithm converges within two iterations. Figure 17 shows the diagnosis accuracy in a 5×5 topology. We observe the coverage is mostly around 80% or higher, and the number of false positives is within 2.

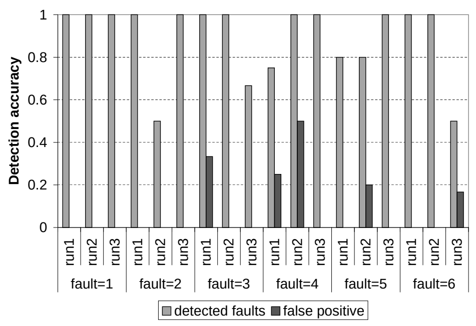


Figure 17: Accuracy of detecting MAC misbehavior in a 5×5 grid topology

Detecting mixtures of random dropping and MAC misbehavior: Next we examine how accurately the diagnosis algorithm can handle multiple types of faults. First we consider mixtures of random packet dropping faults and MAC misbehavior. To challenge the diagnosis scheme, we choose pairs of nodes adjacent to each other with one node randomly dropping one of its neighbors' traffic and the other node using an unusually small CW. We vary the number of node pairs selected to misbehave from 1 to 6 (i.e., the total number of faults is varied from 2 to 12 in the network). Figure 18 summarizes the accuracy of fault diagnosis in a 5×5 grid topology. As it shows, in most cases around 20% or less faults are left undetected. Moreover, the false positive (not shown) is close to 0 in all cases. Comparing the inferred link dropping rates with their actual rates, we observe the inference error is between 20% to 40%.

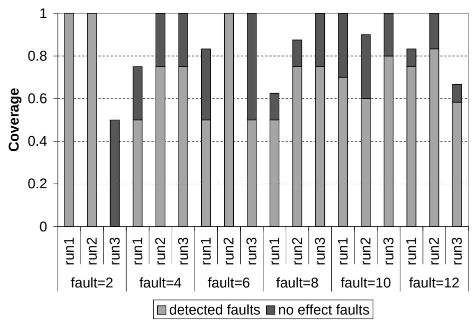


Figure 18: Accuracy of detecting combinations of MAC misbehavior and random packet dropping faults in a 5×5 grid topology

Detecting mixtures of all three fault types: Finally we evaluate the diagnosis algorithm under mixtures of all three fault types as follows. As in the previous evaluation, we choose pairs of nodes adjacent to each other with one node randomly dropping one of its neighbors' traffic and the other node using an unusually small CW. In addition, we randomly select two nodes to generate external noise. Figure 19 summarizes the accuracy of fault diagnosis in a 5×5 topology. As it shows, the coverage is mostly above 80%. The false positive (not shown) is close to 0. The accuracy remains high even when the number of faults in the network exceeds 10. As before, the inference errors in links' dropping rate and noise level are within 30% and 4%, respectively.

To test sensitivity of our results on the network size, we also evaluate the accuracy of the diagnosis algorithm on a 7×7 grid topology. We

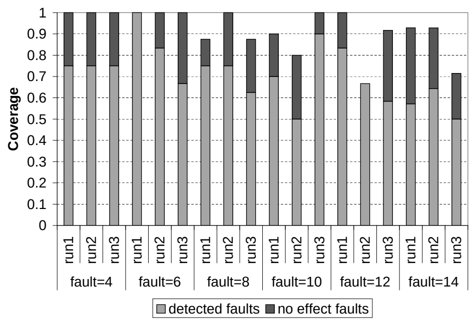


Figure 19: Accuracy of detecting combinations of MAC misbehavior, random packet dropping, and external noises in a 5×5 grid topology

# Faults	4	6	8	10	12	14
Coverage	1	0.67	0.75	1	0.92	0.64
False positive	0	0	0	0.1	0	0.07

Table 3: Accuracy of detecting combinations of MAC misbehavior, random packet dropping, and external noises in a 7×7 grid topology

again randomly choose 25 pairs of nodes to send CBR traffic at 1 Mbps rate. Table 3 summarizes results of one random run. As it shows, we can identify most faults in the network with few false positives.

In addition, we evaluate the algorithm using a 25-node random topology, where the nodes are randomly placed in a 1000m × 1000m region, and 25 pairs of nodes are randomly chosen to send CBR traffic at a 1 Mbps rate. As shown in Table 4, the diagnosis algorithm continues to perform well, yielding reasonably high coverage and close to 0 false positive.

7.2.2 *Using simulation-based diagnosis for what-if analysis*

Finally we demonstrate the utility of simulation-based diagnosis technique for doing what-if analysis. We set up a 7×7 grid topology with 7 vertical flows and 1 horizontal flow as shown in Figure 20. (Note that we use AODV as the routing protocol. Since occasionally the shortest routes timeout due to packet losses, the routes in use sometimes slightly deviate from the vertical/horizontal routes.) The transmission power is 15 dBm; the communication and interference ranges are 250m and 450m, respectively; the width and height of every grid cell are both 200m. Therefore two nodes adjacent horizontally or vertically can communicate directly, but not two nodes adjacent diagonally.

We observe the flows experience much lower throughput than their initial demands. To systematically detect the flow whose presence causes serious degradation in network throughput, we consider a new type of fault—damaging flows. This fault is different from the previous faults in that it is healthy by itself but interfere with the other competing flows.

The way we detect the damaging flows is by looking at the change in the aggregate network throughput if we remove the flow from simula-

# Faults	4	6	8	10	12	14
Coverage	1	1	0.75	0.7	0.92	0.86
False positive	0	0	0	0	0.25	0.29

Table 4: Accuracy of detecting combinations of MAC misbehavior, random packet dropping, and external noises in a 25-node random topology

tion. Since we are interested in knowing the effects of network changes, we consider adaptive flows (i.e., sending rate to the air varies). As shown in Figure 21, removing flow 8, which crosses all the other 7 flows, results in the largest increase in throughput. In other words, its presence caused the most damage. This is consistent with our expectation. We have successfully applied the technique to detect less easily identifiable damaging flows.

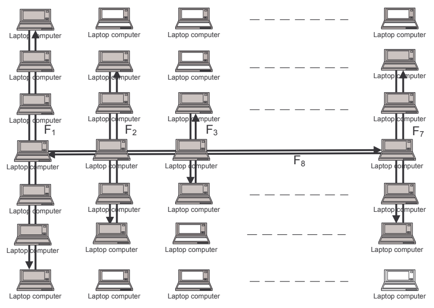


Figure 20: 7×7 grid topology used for detecting damaging flows, where each flow sends 1.222 Mbps CBR traffic in a 802.11b wireless network.

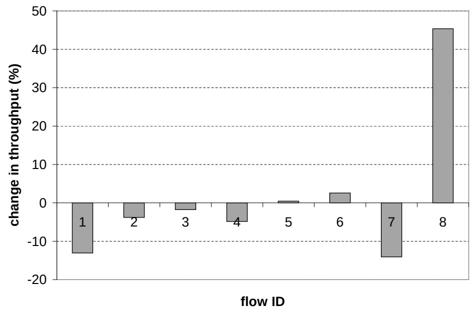


Figure 21: Use what-if analysis to detect damaging flows

Such detection is useful for identifying candidate long-lived flows for corrective action, where long-lived flows can be determined by applying the schemes such as [32, 14] (e.g., based on the number of packets arrived within some time interval). The possible corrective actions may include rate-limiting, re-routing, and topology control. Simulation enables us to further evaluate the benefit of these actions accurately. For example, Table 5 shows the expected throughput for some of the corrective actions. As we can see, increasing the power to 25 dBm yields the highest throughput among the four actions under consideration, since it reduces the number of hops needed to reach destinations. Based on these results, one may consider to increase power to alleviate the performance problem.

Action	Total throughput (Mbps)
No action	1.064
Reduce 8th flow's rate by half	1.148
Route 8th flow via the grid boundary	1.217
Increase power from 15 dbm to 20 dBm	0.990
Increase power from 15 dbm to 25 dBm	1.661

Table 5: Use what-if analysis to quantify the performance impact of corrective actions to the network in Figure 20.

Summary: To summarize, we have evaluated the fault diagnosis approach using a variety of scenarios, and shown it yields fairly accurate results. Moreover the approach is flexible for what-if analysis to assess the performance implications of alternative network configurations.

7.3 Data cleaning effectiveness and overhead

As mentioned earlier, to deal with data imperfectness, we need to process the raw data by applying the inconsistency detection scheme described in Section 5.3 before feeding them to the diagnosis module. In this section, we evaluate the effectiveness of this scheme using different network topologies, traffic patterns, and degrees of inconsistency.

- Network topologies: We use both random and grid topologies for evaluation. In the former, we randomly place nodes in a region while in the latter we place nodes in an $L \times L$ grid, where only the nodes horizontally or vertically adjacent can directly communicate with each other. We vary the size of the region to evaluate how node density affects the accuracy of inconsistency detection, while fixing the total number of nodes at 49 in all cases.

- Traffic patterns: We generate CBR traffic in the network. We consider two types of traffic patterns:

    1. Client-server traffic: in this case, we place one server at the center of the network to serve as an Internet gateway, and the other 48 nodes all establish connections from themselves to the gateway. We assume that the performance reports generated by the server are correct, and if a client's report deviates from the server's, it is the client that supplied incorrect information.

    2. Peer-to-peer traffic: we randomly select pairs of nodes from the network to transfer CBR traffic. We keep the number of connections the same as in client-server traffic.

- Inconsistent reports: We randomly select a varying fraction of nodes to report incorrect information. In addition, for every such node, we vary the fraction of its adjacent links that are reported incorrectly. We use d to denote the fraction, where $d = 1$ means that the selected node reports all the adjacent links incorrectly, while $d < 1$ means that the selected node reports a fraction of its adjacent links incorrectly.

We again use coverage and false positive to quantify the accuracy, where coverage denotes the fraction of misbehaving nodes that are correctly identified, whereas false positive is the ratio between the number of nodes that are incorrectly identified as misbehaving and the number of true misbehaving nodes.

Effects of node density: Figure 22 shows the effect of node density on the fraction of misbehaving nodes detected and false positives in random topologies. When the area is a 1400m $\times$ 1400m, a node has 7 to 8 neighbors within communication range on average, whereas in a 2450m $\times$ 2450m region, a node only has 2 to 3 neighbors on average.

We make the following observations. First, the detection accuracy is high: except for the lowest node density, in most cases the coverage is above 80% and false positive (not shown) is below 15%. Second, as one would expect, the detection accuracy tends to be higher in a denser topology than in a sparser topology. This is because in a denser topology, there are more observers for each link, and majority voting works better. Note that the accuracy does not strictly decrease with the network size due to random selection of misbehaving nodes.

Effects of traffic types: Also, in Figure 22, we see that with peer-to-peer traffic, the detection accuracy is lower than with client-server traffic. This is because for client-server traffic, we trust the server to report correct information; we can detect misbehaving clients whenever their reports deviate from that of the server. In comparison, in peer-to-peer traffic, all nodes are treated equally and we only rely on majority voting.

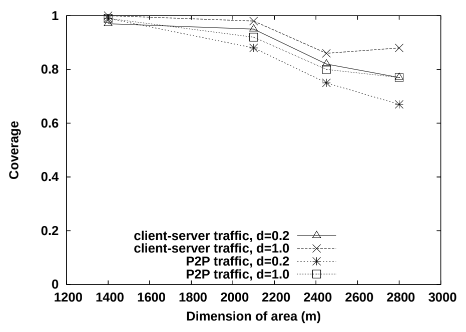


Figure 22: Detection accuracy in random topologies with varying d, node density, and traffic patterns.

Effects of number of misbehaving nodes: Figure 23 plots the detection accuracy versus the number of misbehaving nodes that report incorrect information in the network. The density is held constant by holding the region fixed at a 2450m $\times$ 2450m square. Here, we plot the accuracy for both the grid and the random topologies. As it show, the accuracy is high even when a large fraction (40%) of the nodes in the system are misbehaving. In all cases, the coverage is higher than 80%, and the false positives (not shown) are lower than 12%.

Effects of topology type: Next we examine the effects of network topologies on detection accuracy. We compare the detection accuracy in the grid topology against the random topology, both spanning 2450m $\times$ 2450m. In Figure 23, we can see that the grid topology almost always has a higher detection accuracy than the random topology. A closer look of the topology reveals that while the average node degree in the grid and random topologies are comparable, both around 2 to 3, the variation in node degree is significantly higher in the random topology. There are significantly more nodes with only one neighbor in the random topology. In this case, it is hard to detect which node supplies wrong information. In comparison, nodes in grid topologies have a similar number of neighbors (only corner nodes have fewer neighbors), and no nodes have fewer than 2 neighbors, which makes it easier for majority voting. This observation suggests that the minimum node degree is more important to detection accuracy than the average node degree.

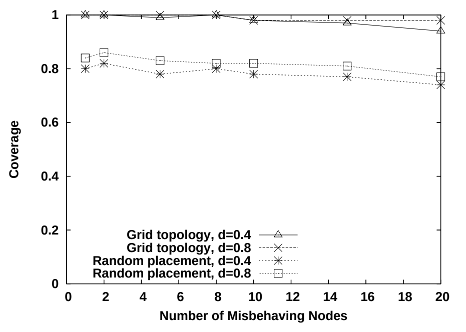


Figure 23: Detection accuracy when the nodes are placed in a 2450m $\times$ 2450m region with varying topology types and the number of misbehaving nodes.

Incorporating history: So far we have studied the case when every node submits one traffic report at the end of simulation. Now we evaluate the case in which nodes periodically send report. In this case, we can take advantage of history information as described in Section 5.3. As shown in Figure 24, we observe a higher coverage when using history information. Similarly, the false positive (not shown) is lower when history information is incorporated.

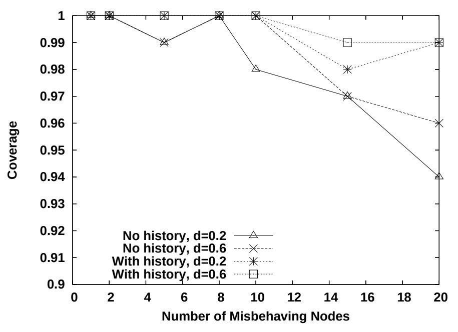


Figure 24: Comparing detection accuracy between with and without using history in a 2450m $\times$ 2450m grid topology using peer-to-peer traffic.

Summary: In summary, we show that the inconsistency detection scheme is able to detect most misbehaving nodes with very few false positives under a wide variety of scenarios we have considered. These results suggest that after data cleaning, we can obtain good quality trace data to drive simulation-based diagnosis.

7.4 Evaluation of Fault Diagnosis in a Testbed

In this section, we evaluate our approach using experiments in a testbed. Our testbed consists of 4 laptops, each equipped with a Netgear WAG511 card operating in 802.11a mode. The laptops are located in the same office with good received signal strength. Each of them runs a routing protocol, similar to DSR [31], to determine the shortest hop-count paths to the other nodes. However, due to packet losses caused by high traffic load and artificial packet dropping, the nodes sometimes switch between 1-hop routes and 2-hop routes. The routing updates and traffic statistics on all links are periodically collected using the monitor tool, described in Section 6. We randomly pick a node to drop packet from one of its neighbors, and see if we can detect it. To resolve inconsistency in traffic report if any, we also run Airopeek [11] on another laptop. (Ideally we would like to have every node monitor traffic in a promiscuous mode, but since we currently do not have such cards, we use Airopeek to resolve inconsistencies.)

First, we run experiments under low traffic load, where each node sends CBR traffic at a rate varying from 1 Mbps to 4 Mbps to another node. We find the collected reports are consistent with what has been observed from Airopeek. Then we feed the traces to the simulator (also running in the 802.11a mode), and apply the diagnosis algorithm in Figure 11. Since in the testbed one node is instructed to drop one of its neighbor's traffic at a rate varying from 20% to 50%, the diagnosis algorithm detects that there is a significant discrepancy between the expected and observed loss rates on one link, and correctly locate the dropping link.

Then we repeat the experiments when we overload the network by having each node sending CBR traffic at a rate of 8 Mbps. In this case, we observe that the traffic reports often deviate from the numbers seen in Airopeek. The deviation is caused by the fact that the NDIS driver for the NIC sometimes indicates sending success without actually attempting to send the packet to the air [37]. This implies that it is not always possible to keep an accurate count of the packets sent locally. However, the new generation of wireless cards, such as Native 802.11 [15], will expose more detailed information about the packets, and enable more accurate accounting of traffic statistics. The inaccurate traffic reports observed in the current experiments also highlight the importance of cleaning the data before using them for diagnosis. In our experiment, we clean the data using Airopeek's reports, which capture almost all the packets in the air, and feed the cleaned data to the simulator to estimate the expected performance. Applying the same diagnosis scheme, we derive the expected congestion loss, based on which we correctly identify the dropping link.

7.5 Summary

To summarize, in this section we evaluate our trace-driven diagnosis approach in both simulation and testbed under a variety of scenarios. Our results show that the approach is effective in detecting and diagnosing faults without imposing much overhead on the network. Moreover we also show that the trace-driven simulation approach can be used for what-if analysis to evaluate the performance of alternate network configurations.

8. LIMITATIONS OF OUR SYSTEM

This paper represents our first attempt in applying simulation-based fault isolation and diagnosis to multihop wireless networks. Although our experience has indicated that this approach is promising, there are a few limitations.

First, simulation-based diagnosis is limited by the accuracy of the simulator and the quality of trace data. Even if we drive the simulator with traces obtained from the real network, we cannot expect a perfect match. Moreover, the time to detect faults is governed by the frequency of data collection, the speed of simulation, and the efficiency of the fault diagnosis algorithm. Our evaluation suggests that its current speed is acceptable for detecting long-term faults, but not for detecting transient faults. In addition, the choice of traces used to drive simulation has important implications on what faults we can diagnose. For example, using trace-driven route simulation implies that we cannot diagnose routing misbehavior. Finally, the current fault diagnosis algorithm has room for improvement. In particular, so far we focus on diagnosing faults resulting in different behavior. In practice, it is possible to have different sets of faults that lead to similar faulty behavior. We plan to explore if it is possible to use a probabilistic approach to diagnose such ambiguous faults.

9. RELATED WORK

To the best of our knowledge, there has been no previous published work on incorporating a simulator as a diagnostic tool for fault detection, isolation, and diagnosis in multihop wireless networks. However, researchers have worked on problems that are related to network management in multihop wireless networks. We broadly classify the work into three areas: (1) protocols for network management; (2) mechanisms for detecting and correcting routing and MAC misbehavior, and (3) general fault management.

In the area of network management protocols, one of the earliest published works that we are aware of is by Chen, Jain and Singh [26]. The authors present Ad Hoc Network Management Protocol (ANMP), which uses hierarchical clustering to reduce the number of message exchanges between the manager and agents. Nodes in the network participate in the cluster construction and management, and the cluster-head polls management information from cluster members in a centralized manner. ANMP takes the vagaries of wireless ad hoc networking into account while preserving compatibility with SNMP [25, 45].

In contrast to ANMP's centralized approach, Shen et al. [44] describe a distributed network management architecture with SNMP agents residing on every node. Some nodes, with greater capabilities, are equipped with a *probe processing* and *nomadic management* module. These modules facilitate management by delegation and the “nomadic managers” collaborate with one another to manage the network.

Our work differs from these two pieces of work in that we do not focus on the protocol for distributing management information, but instead focus on algorithms for identifying and diagnosing faults and testing alternative configurations. Consequently, our work is complimentary to both [26] and [44].

A seminal piece of work in the area of detecting routing misbehavior is by Marti, Giullu, Lai, and Baker [36]. The authors address network unreliability problems stemming from selfish intent of individual nodes, and propose a *watchdog* and *pathrater* agent framework for mitigating routing misbehavior and improving reliability. The basic idea is to have a watchdog node observe its neighbor and determine whether it is forwarding traffic as expected. The pathrater assigns ratings for paths based on observed node behavior. Based on the observations and rating, nodes establish routes that avoid malicious nodes, which in turn results in overall increase in throughput. Other work following this thread of research include [20, 23, 24].

Our work differs from watchdog-like mechanism in the following ways. First, the focus of the above research has been on detecting and punishing malicious nodes, whereas our focus is on detecting and diagnosing general performance problems. Second, we use reports from multiple neighbors and take historical evidence into account to derive more accurate link loss rates. Finally, more importantly, we use online simulation-based diagnosis to determine the root cause for high link loss rates. Different from a simple watch-dog mechanism, which considers end points as misbehaving when its adjacent links incur high loss rate, our diagnostic methodology takes into account current network configuration and traffic patterns to determine if the observed high loss rates are expected, and determines the root causes for the loss (e.g., whether it is due to RF interference, or congestion, or misbehaving nodes).

In the area of wireless network fault management, there exist a number of commercial products in the market. Examples include AirWave [2], AirDefense [18], Computer Associate's UniCenter [3], Symbol's Wireless Network Management System (WNMS) [10], IBM's Wireless Security Auditor (WSA) [13], and Wibhu's SpectraMon [9]. Our work differs from these in that these products target infrastructure or base station based wireless networks. Multihop wireless networks are significantly different.

10. CONCLUSION & FUTURE WORK

Diagnosing faults in a multihop wireless network is challenging due to unpredictable physical medium, distributed nature of the network, and complicated interactions between various protocols, environmental factors, and potentially multiple faults.

To address these challenges, we propose the use of online trace-driven simulation to diagnose faults and test alternative network configurations. We evaluate our approach using several different scenarios, and show that the approach is effective in detecting, isolating, and diagnosing multiple faults. We also show that our approach can be used to evaluate alternative node and network configurations for efficient network operation.

There are a number of avenues for future work. First, we are in the process of validating our diagnosis approach in a larger-scale testbed. Second, we plan to examine how well our diagnosis approach works in presence of node mobility.

Third, we have focused on faults resulting from misbehaving (but benign) nodes, links, or adverse environmental factors. Malicious attacks are hard to detect and can often be disguised as benign faults. Security mechanisms (e.g., cryptographic schemes for authentication and integrity) and measures (such as secure traceroute [39]) are necessary weapons to guard against malicious attacks. We would like to study how to integrate these mechanisms into our diagnosis system.

Fourth, our diagnosis currently assumes a fairly complete knowledge of the network in terms of RF condition, traffic statistics, and link performance. Obtaining such complete information may sometimes be hard. As part of our future work, we plan to investigate techniques to perform detailed diagnosis for a subset of the network. Such capability will enhance the scalability of our approach.

Finally, we are in the process of investigating more deeply the utility of on-line trace-driven simulation for what-if analysis. This has the potential of becoming a useful tool that can be employed for assisting network configuration and planning.

11. REFERENCES

[1] Advanced Micro Devices (AMD). http://www.amd.com/.

[2] Airwave, a wireless network management solution. http://www.airwave.com/.

[3] The future of wireless enterprise management. http://www3.ca.com/.

[4] IEEE 802.11k radio resource measurement. http://standards.ieee.org/802news/.

[5] MIT Roofnet. http://www.pdos.lcs.mit.edu/roofnet/.

[6] Promise of intelligent networks. http://news.bbc.co.uk/2/hi/technology/2787953.stm.

[7] The Qualnet simulator from Scalable Networks Inc. http://www.scalable-networks.com/.

[8] Realtek. http://www.realtek.com.tw/.

[9] SpectraMon, Wibhu Technologies Inc. http://www.wibhu.com/.

[10] SpectrumSoft: Wireless network management system, Symbol Technolgies Inc. http://www.symbol.com/.

[11] Wildpackets Airopeek. http://www.wildpackets.com/products/airopeek.

[12] Wireless research API. http://ramp.ucsd.edu/pawn/wrapi/.

[13] Wireless security auditor (WSA). http://www.research.ibm.com/gsal/wsa/.

[14] ATM Forum MPOA Sub-Working Group. In *Multi-Protocol over ATM Version 1.0 (AF-MPOA-0087.000)*, Jul. 1997.

[15] Native 802.11 framework for IEEE 802.11 networks. Windows Platform Design Notes, March 2003. http://www.microsoft.com/whdc/hwdev/tech/network/802x/Native80211.mspx.

[16] NSF workshop on residential boradband revisited: Research challenges in residential networks, boradband access and applications. http://cairo.cs.uiuc.edu/nsfbroadband/, October 2003.

[17] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou. A multi-radio unification protocol for IEEE 802.11 wireless networks. In *Microsoft Technical Report, MSR-TR-2003-41*, June 2003.

[18] AirDefense: Wireless LAN security and operational support. http://www.airdefense.net/.

[19] W. A. Arbaugh, N. Shankar, C. J. Wan, and K. Zhang. Your 802.11 wireless network has no clothes. In *IEEE Wireless Communications*, volume 9, pages 44–51, December 2002.

[20] B. Awerbuch, D. Holmer, and H. Rubens. Provably secure competitive routing against proactive Byzantine adversaries via reinforcement learning. In *JHU Tech Report Version 1*, May 2003.

[21] G. Berger-Sabbatel, F. Rousseau, M. Heusse, and A. Duda. Performance anomaly of 802.11b. In *Proc. of IEEE INFOCOM '2003*, June 2003.

[22] P. Bhagwat, B. Raman, and D. Sanghi. Turning 802.11 inside-out. In *Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[23] S. Buchegger and J. Y. Le Boudec. Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks. In *Proceedings of the Tenth Euromicro Workshop on Parallel, Distributed and network-based Processing*, pages 403 – 410. IEEE Computer Scoiety, January 2002.

[24] S. Buchegger and J.-Y. Le Boudec. The effect of rumor spreading in reputation systems for mobile ad-hoc networks. In *Proceedings of Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, Sophia-Antipolis, France, March 2003.

[25] J. Case, M. Fedor, M. Schoffstall, and J. Darvin. A simple network management protocol (SNMP). In *Internet Engineering Task Force, RFC 1098*, May 1990.

[26] W. Chen, N. Jain, and S. Singh. ANMP: Ad hoc network management protocol. In *IEEE Journal on Selected Areas in Communications*, volume 17, August 1999.

[27] D. D. Clark, C. Patridge, J. C. Ramming, and J. T. Wroclawski. A knoweldge plane for the internet. In *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, August 2003.

[28] Wireless networking reference—community wireless/rooftop systems. http://www.practicallynetworked.com/.

[29] D. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proc. of ACM MOBICOM '2003*, Sept. 2003.

[30] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[31] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*, 2001.

[32] Y. Katsube, K. Nagami, S. Matsuzawa, and H. Esaki. Internetworking based on cell switch router - architecture and protocol overview. In *Proc. of the IEEE*, Dec. 1997.

[33] A. V. Konstantinou, D. Florissi, and Y. Yemini. Towards self-configuring networks. In *DARPA Active Networks Conference and Exposition (DANCE '02)*, San Francisco, CA, May 2002.

[34] D. Kotz, C. Newport, and C. Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College, Computer Science, Hanover, NH, July 2003.

[35] P. Kyasanur and N. Vaidya. Detection and handling of MAC layer misbehavior in wireless networks. In *Dependable Computing and Communications Symposium (DCC) at the International Conference on Dependable Systems and Networks (DSN)*, pages 173–182, San Francisco, California, June 2003.

[36] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of MOBICOM 2000*, Boston, MA, August 2000.

[37] Network devices and protocols: Windows DDK. NDIS library functions.

[38] L. M. S. C. of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Standard 802.11*, 1999.

[39] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. In *ACM SIGCOMM Workshop on Hot Topic in Networks (HotNets-I)*, Oct. 2002.

[40] N. Pissinou, B. Bharghavati, and K. Makki. Mobile agents to automate fault management in wireless and mobile networks. In *Proceedings of the IEEE International Parallel and Distributed Processing Systems Workshop*, pages 1296–1300, 2000.

[41] R. Prakash and M. Singhal. Low cost checkpointing and failure recovery in mobile computing systems. In *IEEE Trans. on Parallel and Distributed Systems*, volume 7, pages 1035–1048, 1996.

[42] M. Sabin, R. D. Russell, and E. C. Freuder. Generating diagnositc tools for network fault management. In *Integrated Network Management*, pages 700–711, 1997.

[43] G. K. Saha. Transient fault-tolerant mobile agent in mobile computing. In *IEEE Transactions on Computers*, USA, 2003. IEEE Computer Society Press.

[44] C.-C. Shen, C. Jaikaeo, C. Srisathapornphat, and Z. Huang. The guerrilla management architecture for ad hoc networks. In *Proc. of IEEE MILCOM*, Anaheim, California, October 2002.

[45] W. Stalling. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards (1st Edition)*. Addison Wesley Professional, Readin, MA, 1993.

[46] Tcpdump. http://www.tcpdump.org/.