Detection and Repair of Faulty Access Points

H.J. Pan and S. Keshav
School of Computer Science, University of Waterloo
Waterloo, ON, Canada, N2L 3G1

Abstract—**In large-scale infrastructure wireless networks several access points (APs) may be unusable at any given moment in time. Unlike completely failed APs, whose failure can be detected by probes to their wired interface, an AP with a faulty wireless interface or whose antenna has been accidentally shielded can only be diagnosed by the actual use of the wireless interface for data communication. We present several algorithms that detect such failed access points by online analysis of AP usage logs. In particular, we demonstrate that we can exploit device mobility to detect faulty APs. We also present efficient heuristics to select a path for a technician to repair failed access points. We evaluate our algorithms using actual log files from an infrastructure network at Dartmouth College. We find that our best algorithm is able to detect nearly 90% of failed access points simply by processing log files. Compared to a naive approach, our algorithm has more than six times fewer false positives. We are also able to construct tours that are up to an order of magnitude more effective that a straightforward greedy approach. Our algorithms require no modifications to either APs or devices. We believe that these properties make our work immediately applicable to real-world scenarios.**

I. Introduction

There has been a recent upsurge in the deployment of IEEE 802.11-based large-scale wireless infrastructure networks. Several encompass entire cities or large geographical areas (up to 700 square miles in a notable example) . Detecting failed APs in such large-scale infrastructure wireless networks, where at any given point in time one or more access points may be unusable, is a difficult problem. For instance, in a network with, say, 4000 APs, with a failure rate of 1%, 40 APs would be faulty at any time. Some of these may be in highly trafficked areas where a failed AP would inconvenience many people. Moreover, a faulty AP may respond to status probes to its wired interface, yet may have a faulty or accidentally blocked radio preventing its use by mobile devices: such failures can only be diagnosed by the use of the wireless interface for data communication. However, it is both inconvenient and expensive to have a device walk or drive by every AP just to monitor the wireless interface. We, therefore, believe that detection of failed APs is an interesting and challenging open problem.

In this paper, we study implicit detection of access point failure by leveraging device mobility to improve detection. We also present heuristics to compute a path that a technician should take in order to repair the faulty APs.

We assume that there is a central location or management server that collects usage logs when a mobile device enters (and potentially when it exits) an access point. Such logs record either 802.11 MAC-level association and dissociation requests or reassociation requests in the Inter-Access Point Protocol (IAPP) [9]. We call each such request a $report$. Existing APs already support such logging, and multi-year traces of log files are publicly available, for instance from Dartmouth College [5]. In the unlikely eventuality that APs do not support this feature, we assume that the mobile sends a short message with its own ID and the MAC address of the AP to a network operation centre (NOC) every time it enters an AP. We hope to detect failed APs, with high probability, by analyzing these logs. Note that because this is a passive approach, *we do not require any modifications to existing clients or APs*.

Our contributions are threefold. First, no reports may be received from a particular AP either because it is faulty or because no mobile happened to go past that AP. Any passive diagnosis algorithm needs to distinguish between these situations. We present a novel technique that exploits device mobility to aid faulty AP diagnosis. We show that this technique far outperforms a naive approach in terms of false positives.

Second, once we know which APs are likely to have failed, we need to send a technician to diagnose and potentially repair these APs. Ideally, the technician should be able to visit all failed APs. But, for wide-area deployments, or those that cover large metropolitan areas, this may be impossible. So, we would like to construct a tour that is shorter than some predetermined maximum but diagnoses as many of the 'popular' APs as possible. Such multi-objective tours are known to be NP-hard [4]. Therefore, we need heuristics to generate efficient tours. We present a best-in-class approach to multi-objective tour construction that efficiently solves the AP touring problem.

Third, we show that probabilistic approaches, that measure and correlate transition rates between APs, appear to be infeasible, because of the non-stationarity of the device mobility process.

Section II presents the system model. Section III describes several algorithms to detect failed access points by online analysis of AP usage logs. In Section IV and V, we present our heuristic for tour generation based on the Ellipse algorithm [3]. We evaluate our algorithms using actual log files from an infrastructure network at Dartmouth College in Section VI. We present related work in Section VII and conclude in Section VIII.

II. System Model

We assume that the system has N access points indexed by i and M mobiles. Time is quantized into fixed length $time\ periods$ of arbitrary duration; typically about an hour. Let $A(t, i)$ be 1 if the network operations center has heard a report from AP i during time period $t, 1 \leq t \leq T$, where T is the most recent time period. For example, if the operations center receives 7 reports from AP 5 during time period 23, $A(23, 5) = 1$. Let $Path(j, k)$ denote the ID of the k^{th} AP encountered by mobile host j on its path (more than one AP may be encountered during a single time period). For instance, if mobile 2 consecutively visits AP 4 and AP 7, then $Path(2, 1)$ = 4, $Path(2, 2)$ = 7. Note that $A(t, i)$ and $Path(i, j)$ can be passively computed

This research was supported by grants from the National Science and Engineering Council of Canada, the Canada Research Chair Program, Nortel Networks, Sun Microsystems Canada, Intel Corporation, and Sprint Corporation.

from the set of reports received by the network operations centre.

III. Failure Detection Algorithms

In this section, we present three algorithms that use the A and $Path$ arrays to compute the set of failed APs. In the sequel, $p(t, i)$ denotes the probability of AP i being up (i.e. not faulty) at the end of time period t.

A. Naive Algorithm

This memoryless straw man algorithm simply marks all APs not heard from in the current time period as being faulty. In other words, $p(T, i) = A(T, i)$. This naive algorithm allows us to benchmark the performance of more sophisticated algorithms.

B. Dynamic Bayesian (DB) Algorithm

We now extend the naive algorithm to take history into account. Intuitively, the longer we haven't heard from an AP, the greater the probability that it is faulty. Our approach is a variation of the well-known Dynamic Bayesian approach [11]. For this approach, we require a prior distribution on the probability that an AP is faulty, denoted by f_0. The probability that an AP is up at each time period is initially assigned a value of 1, that is, $\forall i, p(0, i) = 1$. At time period t, if we havent heard from AP i, we reduce its probability of not being faulty by a factor of $1 - f_0$. At time T, we compute $p(T, i)$ by updating the current value of p i.e. $p(T - 1, i)$ as shown below.

```
if A(t, i) = 0, then p(t, i) = p(t-1, i)*(1-f0)
if A(t, i) = 1, then p(t, i) =1
```

C. Anomalous Paths (AnP) Algorithm

The two algorithms presented above do not use path information. We now describe how to exploit device mobility i.e. path information, for fault detection. In Section VI, we show that this technique substantially improves on the previous two algorithms.

Recall that the usage logs allow us to recover the path of APs traversed by each mobile device. The union of these paths, collected over a sufficiently long period of time, allows us to compute the *neighbor graph*[1] [9], an undirected graph where vertices represent APs and an edge connects links vertex i to vertex j if APj follows APi in some path. Given a neighbor graph collected over say, a few months time, we can use observed path fragments to detect anomalous paths that suggest AP outages.

For example, consider three access points A, B, and C that are located adjacent to each other in a long hallway with no exits (Figure 1).

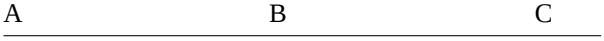

Figure 1: The simple case

A and B are neighbors in the neighbor graph and we expect these two APs to appear after each other in the path of *every* mobile terminal that traverses the hallway. So if a path fragment from a mobile has one of the two APs but not its successor, we can guess that the successor AP is faulty. For instance, if we see a path fragment 'AC', we can guess that B is faulty.

Determining which AP is faulty is easy when mobile paths are constrained and each AP has a unique successor. However, we can still make a good guess even if successors are not unique. Consider the situation in Figure 2.

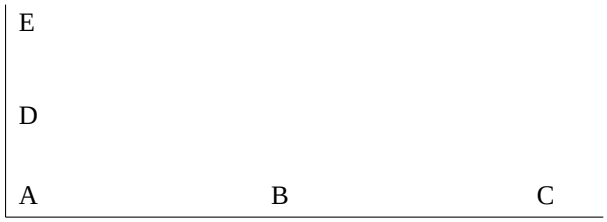

Figure 2: A harder case

Here, both B and D can follow A, so A does not have a unique successor. Nevertheless, if we see a path fragment 'AC', we can guess that B is faulty.

This form of inference requires every successor of every node to have one or more unique successors i.e. the graph should be tree-like. However, graphs may have cycles. Consider Figure 3.

Figure 3: An undetectable case

Here, both 'ABC' and 'AC' are valid paths. If we see 'AC' we cannot say anything about the status of B. Therefore, a fault at B is *undetectable* using our approach.

However, we can still make inferences if we augment topological analysis with presence data. Consider Figure 4.

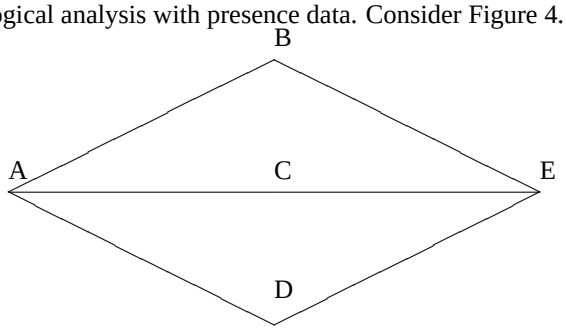

Figure 4: The general case

Suppose we see a path fragment 'AE'. We can infer that one of B,C, or D is faulty. If, in the past time period we have presence reports from C and D, we can eliminate them to determine that B is likely to be faulty.

We now formalize the definition of the anomalous path algorithm. The algorithm proceeds in two phases. In the observation phase, the neighbor graph is constructed. We define the neighbor graph $G(V, E)$ as a graph where each vertex corresponds to an AP, and there is an edge between two vertices A and B, denoted $A - B$, if a mobile associates with these two APs consecutively during the observation period. If the observation period is long enough, the neighbor graph will include all possible

[1] Unlike the work cited here, where neighbors are defined by means of an IAPP reassociation relationship, we call AP A and B neighbors if any mobile has associated *or* reassociated with AP B after having been associated with AP A

neighbor transitions. However, note that, due to failures and repairs during the observation period, there are likely to be spurious edges in a real-life neighbor graph. Unfortunately, these are unavoidable. In practice, it will be necessary to remove these spurious edges explicitly; for the purposes of our analysis, we will assume that this has been done.

During the detection phase, in each time period, for all the M mobiles, we examine path fragments that lie within that time period. If a path fragment has a transition $A \rightarrow B$ such that $A - B$ is not an edge in G, then we form the $candidate\ set\ C$, C = $\{V$: $A - V \in G$ and $V - B \in G\ \}$. Next, we prune C by removing all $V \in C$, s.t. $A[T, V]$ = 1. We declare all the APs left in C as being faulty. As a refinement, to prevent too many false positives, we do not declare any AP to be faulty if C has more than a threshold number of elements. This general algorithm is sufficient to detect anomalies of the form described in Figures 1, 2 and 4, but not 3.

D. Markovian approach

Anomalous-path based fault detection fails when we have a situation similar to that in Figure 3, where an P has a link to a neighbor as well as to that neighbor's neighbor. Nevertheless, in such cases, changes in link traversal rates might still give some indication of faults.

In Figure 3, suppose that, over a long observation period, we find that of 100 mobile users who come to AP A, 80 associate next with B, and 20 associate next with C. If B were to fail, then all 100 would associate next with C, allowing us to detect B's failure.

Indeed, it is tempting to model the neighbor graph as a Markov chain, with the APs corresponding to Markov states and transitions between APs modeled as Markov transition probabilities. Then, a fault could be detected by the discrepancy between the observed transition rate and the underlying Markov chain.

Unfortunately, this approach has a problem both subtle and insidious. Fundamentally, the issue is that the transition from one AP to another is heavily time dependent. For example, in Figure 3, if AP A is in a classroom, and its neighbors B and C are in another classroom and in a residence hall respectively, the rate of transition to B will be higher during class hours than after hours. In general, the transition rates from one AP to another may depend in detail on the time of day, day of week, and week of year.

How can this be modeled? We believe that are only two options. One would be to compute transition probabilities for each set of time period clusters that have 'similar' behavior. So, for instance, we could model 'working' and 'non-working' times and so on. Each would have its own associated Markov chain. The second approach would be to measure transitions over sufficiently long intervals, so that the measured rates average out temporal variations. This way, the transition probabilities would be (nearly) time-independent.

However, both approaches have insuperable problems. With the first approach, we need to cluster neighbor graph transitions measured over some time interval based on their similarity. This is easier said than done. For example, when we clustered the Dartmouth data set by day of week, we found that Mondays during the term break are very different from Mondays during term. Moreover, clustering time periods from Mondays during terms is not enough: if a holiday lies on a Monday, it looks more like a weekend! Using such time-based chains also leads to a problem of circularity: we create clusters based on usage, then try to find the cluster that most closely describes the current observation. Clearly, this approach is infeasible.

TABLE I
MEAN AND STANDARD DEVIATION AS A FUNCTION OF AVERAGING INTERVAL

Averaging interval	*Mean*	*Standard Deviation*
1 hour	7.18	12.43
2 hours	7.18	10.96
3 hours	7.18	10.15
4 hours	7.18	9.42
5 hours	7.18	9.20
6 hours	7.18	8.84
1 day	7.18	6.35

With the second approach, to get time-independent probabilities, i.e. transition probabilities with low coefficients of variation, we need to measure transitions over a time scale long enough to accommodate all short-term time dependencies. For instance, if we were to measure transition probabilities over periods of a day or week, we would not be affected by the end-of-day time dependency. However, there we find that in real data sets, there are variations with respect to time of day, day of month, and month of year. To take all these into account, we need to average over a period of a year or more. But then, to see if the observed transition rate matches the long term rate (in order to detect faults), we need to measure the transition rate over *similar intervals*. This means that, if we are measuring transition probabilities over a year, we will have to wait a year to detect a fault! This is far too long a time in practice, ruling out this approach also.

We illustrate this phenomenon quantitatively using Table I. This shows the mean and standard deviation of observed transition rates between two selected access points on the Dartmouth campus [5] when these statistics are computed over successively longer time periods. In all cases, the means and standard deviations are computed for a 3-month long trace. Note that when the statistics are computed over a short observation period, the system is not stationary over the duration of the time period, so the standard deviation is huge. The standard deviation decreases with increasingly longer observation periods. However, even with time periods of a day, a 95% confidence interval would allow a large variation in the observed rate before it was marked as being incompatible with the long term rate. This defeats our goal of fault detection.

We conjecture that all probabilistic approaches, where probabilities are viewed as limiting distributions of statistics, are unusable in fault detection in heavily time-dependent systems, such as those that arise from user mobility in infrastructure wireless AP networks. This is a strong result that rules out entire classes of approaches. Indeed, the only approaches that can work are those that do not presume time-invariance and essen-

tially observe anomalies over short time windows, such as those described in Section III.A-III.C.

IV. Touring

We now turn our attention to the problem of correcting failed APs. At the end of each time period, our algorithms mark one or more APs as being potentially faulty. At this point a technician has to go to each AP and diagnose it.

In what order should the APs be checked? A naive solution is to visit APs in random order. A better solution would try to minimize an objective function, such as the total length of the tour; a variant of the classic Traveling Salesman Problem (TSP). We model tour construction in terms of a generalized version of the TSP called multi-objective touring [4]. The intuition is that we want to prioritize diagnosing and/or repairing APs that are heavily used because an outage on these APs causes more disutility to mobile devices than an AP that is rarely used. The tour, therefore, ought to minimize both the length of the tour as well as visit 'popular' APs.

We make this more concrete by defining the $weight$ of an AP as the mean number of occurrences of the AP during a time period. Given a set of APs represented by a neighbor graph, where link lengths represent the time taken to walk or drive on that link, we wish to construct a tour that simultaneously maximizes the weight of a tour and minimizes its link cost.

This definition has the following problem: if an AP is faulty, but with low weight, it may never be visited, because it will be pre-empted by other APs with heavier weights. Consequently, we wish to increase the weight of an AP the longer it is faulty. This is done as follows: we define $downtime(i)$ as the number of time periods AP i has not been visited after it has been declared to be faulty. The adjusted weight of AP i is calculated as: $adjustedweight(i) = w_1 * downtime(i) + weight(i)$ where w_1 is a tuning parameter.

V. The Multi-Objective Touring Problem

We now formally define the multi-objective touring problem. We are given a neighbor graph corresponding to all the APs, potentially enhanced with additional waypoints (such as staircases and building entries and exits) that must be traversed in going from one AP to another. A node has a location in 3-dimensional space. Both edges and nodes have weights. The weight of an edge is the cost of traversing the edge in time units. The (adjusted) weight of a node is defined in Section IV. We are also given a set of nodes that must form part of the tour, these correspond to the APs that have been declared faulty.

We wish to find a tour that starts and ends at the origin node and that simultaneously maximizes the sum of the node weights and minimizes the tour length.

We assume that a technician takes a fixed amount of time R to diagnose and/or repair a failed AP – this is approximately constant because an AP that fails diagnostic tests can simply be replaced with a new one. We also assume that we have a fixed upper bound T_{max} on the maximum length of a tour.

It has been shown that this problem, also called $orienteering$, is NP-complete [4]. However, many heuristics have been proposed in the literature. We adapt the ellipse approach proposed in [3] with minor modifications, as sketched below. We chose this heuristic because independent evaluation has shown this to have the best performance of known heuristics [7]. We compare this approach with two greedy touring heuristics: Highest score first (GS) and Nearest-faulty-AP next (GD). In GS, we start at the origin and use the shortest path to go visit faulty APs in order of decreasing score until we run out of time (i.e. the tour time exceeds T_{max}). In GD, we start at the origin, and go to the closest faulty AP, and so on, until we run out of time.

A. The Ellipse Algorithm

The ellipse heuristic for orienteering [3] proceeds in several phases. In the first or initialization phase, an ellipse with foci at the start and end points and major axis of size T_{max} is drawn on the plane. Points outside the ellipse trivially cannot be in the final solution and are eliminated. Then, starting from the origin, several alternative solutions are constructed using a greedy approach. The best of these is the initial solution, and the others are designated as alternates.

In the second phase, the initial solution is improved by swapping two points at a time between it and one of the alternate solutions ('two-point exchange'). When this shows no further improvement, *one point movements* are used in an attempt to further improve the solution. A 'clean-up' procedure then tries to shorten the solution using the well-known 2-opt approach.

In the third phase, we move a small number of points greedily from the solution to one of the alternates ($reinitialization$) and repeat all three phases. The algorithm terminates after a fixed number of repetitions.

It has been found that this heuristic produces paths that are very close to optimal, and with low computational overhead. We now describe three modifications made to this algorithm to suit our problem context.

First, in [3] the graph is assumed to be complete. However, the neighbor graph is not complete. Therefore we first convert the neighbor graph to a complete graph by calculating Dijkstra shortest paths from every vertex to all other vertices, and setting the edge cost between a pair of vertices to their shortest path cost. That is, if the shortest path from AP i to AP j has weight w, in the modified graph the edge weight $W[i, j] = w$. We call this modified complete graph G'. The rest of the algorithm runs on G' instead of G. Interestingly, the triangle inequality, required by [3], also holds in G'. To see this, assume the triangle inequality does not hold in G'. Then there exist 3 APs A, B, and C such that $W(A, B) + W(B, C) < W(A, C)$. But $W(A, C)$ is the weight of the shortest path from A to C, so there cannot be a path with less total weight, implying a contradiction.

As a second modification, instead of choosing points inside an ellipse, we select APs i that satisfy the following condition: $W(origin, i) + W(i, origin) + R \leq T_{max}$.

Our third and final modification is to include the repair/diagnosis time R in the length of a path. So, for instance, the cost of a path from AP A to AP B to AP C would be $W(A, B) + W(B, C) + 3 * R$.

VI. Evaluation

We now turn our attention to a quantitative evaluation of fault detection and touring algorithms. We first compare three competing algorithms for fault detection: Naive (N), Dynamic

Bayesian (DB), and Anomalous Path (AnP). We then compare three competing algorithms for touring: ellipse-based multi-objective algorithm (OP), and two greedy approaches, highest-score-first (GS), and nearest-faulty-AP next (GD).

A. Data sets

In order to evaluate fault detection and touring algorithms, we need several inputs:

- The neighbor graph of APs
- A method to determine the cost to traverse an edge in the neighbor graph.
- The time-stamped log of association and reassociation requests at each AP.
- The set of APs that are actually faulty (so that we can compare this with the APs that we detect to be faulty).
- The repair time R.
- The maximum tour time T_{max}

To derive realistic neighbor graphs, edge weights, and association logs, we use the data set freely available from Dartmouth College [5]. This data set represents log messages due to every 802.11 client association, authentication, reassociation, dissociation, and deauthentication in Dartmouth's wireless network over a period of nearly four years. The data set also includes the (x, y, z) position of each access point, where the z coordinate is the floor where the AP is located.

We chose a 3-month period from Jan 1, 2003 to March 31, 2003 for our analysis. This portion of the trace includes about 540 access points and nearly 14,000 clients, mostly laptops. Each trace record has a timestamp, the client's ID, the AP ID, and the event type. Converting this data into a neighbor graph, however, in non trivial.

Consider a laptop user who accesses AP A, then shuts down the laptop, walks across campus, and accesses AP B. Should A and B be neighbors? If we answer yes, then the edge degree of each AP is very large (we found a mean edge degree of nearly 65 with this definition of neighbor). With a large edge degree, however, a neighbor graph loses topological significance. We therefore restrict the definition of a neighbor to APs with whom the same client has associated within a 30 minute period, and whose Euclidean distance is less than 30m, which is the typical 802.11b range. With this restriction, the mean edge degree drops to a more reasonable value of 12.

To find the edge weights for the Dartmouth neighbor graph, we could, in principle, examine at the inter-association times for mobiles that move from one AP to another. However, this is not accurate because the trace data only has rough estimates of dissociation times[2]. Consequently, if a user were to spend five hours associated with AP A, then associates with AP B, we do not have an accurate estimate to tell how long it takes to get from AP A to AP B. To get around this, we use the (x, y, z) coordinates of the APs to estimate transit times. Recall that the z coordinate is the floor of a building. We assume that all buildings are connected at the ground floor. So, the distance between two APs at locations (x_1, y_1, z_1) and (x_2, y_2, z_2) is given by $(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + \alpha * (|z_1| + |z_2|))$, where α is the mean height of a building floor. In our calculations, we use a value of α of 4.5m.

In the sequel, instead of using edge transit times, we work directly with distances, since the transit time is simply this distance divided by the mean repair person velocity. One distance unit is equivalent to 18cm.

Finally, we assume R to be 7000 distance units (the distance covered by a repair person walking at 5km/hr in 15 minutes) and vary T_{max} to compare the effectiveness of different touring algorithms.

B. Results

To generate faults, we choose one day at random during the trace duration, choose ten APs at random, then and remove every occurrence of the AP in the log files. This roughly simulates what one would see if the AP had in fact been faulty. We then measure the fraction of these artificially induced faults detected by our fault detection algorithms. We repeat this process 50 times, and report the averages across these 50 runs.

We first compare the effectiveness of the three heuristics (Naive, Dynamic Bayesian, and Anomalous Path) in detecting faults in Table II. In addition to the algorithms already described, we show results for two variants of the DB algorithm, corresponding to different values of f_0, and three variants of the AnP algorithm. These three variants differ in the choice of threshold as defined in Section III.C. Increasing the threshold increases the probability of fault detection, but at the cost of additional numbers of false positives.

For each algorithm, Table II shows the probability that it detected one of the failed APs. For the N and AnP algorithms, this is simply the number of failed APs detected divided by 500 (i.e. 50 runs, with 10 APs deleted in each run). For the DB algorithm, we define $Prob(detection) = \frac{1}{50}(\sum_{j=1}^{50}\sum_{i=1}^{10}(1 - p^j(T, i))$ where $p^j(T, i)$ is the probability that the DB algorithm declares AP i to be not faulty at the end of time T in the j^{th} run. As noted earlier, some APs are more important than others, so we show both the probability of detection and the weighted probability, where the weight of an AP is the number of times it occurs in the 3-month trace. We also show number of false positives due to each algorithm. Standard deviations are in parentheses.

We find that both the Naive and the Dynamic Bayesian algorithms can detect all faults even for unpopular APs. The reason for their success is because they are very conservative, and tend to guess that an AP is faulty soon after reports stop arriving. The anomalous path algorithm is only triggered by anomalies, and so is unable to detect failures that do not cause anomalies. However, the Naive and Dynamic Bayesian schemes perform much more poorly in terms of false positives. When there are ten actual faults, the Naive algorithm reports roughly 400 false positives (of a population of 539 APs) and the DB algorithm reports around 250 false positives. This makes them unusable in practice. The anomalous path algorithm reports only between 30 and 45 false positives, on average, depending on the choice of threshold. The false positives with AnP is due to the large node degree in the Dartmouth trace.

Note also that variations in the value of f_0 does not change DB's performance very much. We also see that the best results are for AnP with a threshold of 3. This algorithm detects almost

[2]An AP generates a deauthenticate message for a client that has not sent any message for the past 30 minutes, and the trace assumes that the client disassociated 30 minutes prior to the deauthenticate message

TABLE II
PERFORMANCE OF FAULT DETECTION HEURISTICS

Algorithm	*Weighted Probability*	*Probability*	*False Positives*
Naive	1 (0.0)	1 (0.0)	409 (62)
DB(f_0=0.96)	1 (0.0)	1 (0.0)	248 (74)
DB(f_0=0.999)	1 (0.0)	1 (0.0)	207 (74)
AnP(Thresh. 5)	0.89 (.13)	.84 (.13)	45 (23)
AnP(Thresh. 3)	0.89 (17)	.79 (.19)	35 (17)
AnP(Thresh. 2)	0.84 (.18)	.76 (.17)	31 (12)

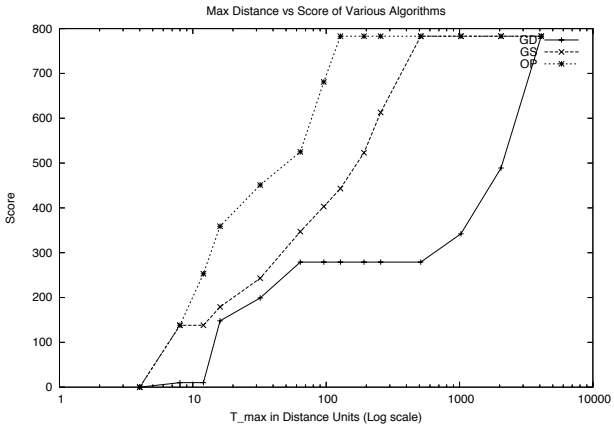

Figure 5: Performance evaluation of touring algorithms

80% of all faults (and has a weighted detection probability of 0.89), yet has about nine times fewer false positives than Naive and about five times fewer false positives than DB.

We now compare the performance of the touring algorithms, as shown in Figure 5. This figure shows, for an example set of failed APs, the score achieved by the three touring algorithms as a function of T_{max}. We see that the Ellipse heuristic is able to achieve the maximum score at a T_{max} value of only around 100, while the GS algorithm achieves it at a value of 400, and the GD at a value of 2000. Thus, the Ellipse heuristic performs about four times better than GS, and an order of magnitude better than GD. This illustrates the potential gains from careful design of touring algorithms. Although we have space to present only a single example, we have seen similar behavior for most of the other scenarios we studied, with the Ellipse heuristic outperfoming GS, and GS outperforming GD in every case.

VII. RELATED WORK

The problem of fault diagnosis in IEEE 802.11 infrastructure networks was first proposed by Adya et al [1] in 2004. Their solution focuses primarily on the use of enhanced clients to both detect and, to some extent, self-diagnose failures and poor wireless performance, with the assistance of a diagnosis server. While the general scope of the problem is the same, our approach is completely passive, and restricted to diagnosing stuck-at AP faults. We are not aware of any other work in the area of implicit detection of 802.11 AP faults.

There is a rich literature on multi-objective touring that is surveyed in [10]. The problem was originally posed by Golden et al [4]. Many approaches have been proposed to solve it, including genetic algorithms and ant-colony approaches. The heuristic we use has been shown to be the best known one [7].

Variations of the orienteering problem (OP) are also well known. For example, in OP with a time window, each control point can only be accessed during a specified time period, [6]. This can model situations where certain APs can be accessed only during some times of the day, perhaps due to security restrictions. If more than one service person is available, an appropriate heuristic is suggested in the work on team OP [2].

VIII. DISCUSSION AND CONCLUSIONS

Detecting failed access points is a hard problem, and, to our knowledge, this is the first attempt to solve it. Our contributions are threefold. First, we present a novel technique that exploits device mobility to aid faulty AP diagnosis. We show that this technique far outperforms a naive approach in terms of false positives. Second, we show that probabilistic approaches, that measure and correlate transition rates between APs, are infeasible, because of the non-stationarity of the mobility process. Finally, we present a best-in-class approach to multi-objective tour construction that efficiently solves the AP touring problem. We present a detailed analysis of our algorithms using a real-world trace. A detailed evaluation on a real-world trace validates our algorithms and demonstrates that they are viable for immediate adoption in infrastructure wireless networks.

IX. ACKNOWLEDGEMENTS

The fault detection problem was suggested to us by Victor Bahl and Lili Qiu at Microsoft Research. We would like to thank Minkyong Kim and Lily Li for their help in processing the Dartmouth data sets.

REFERENCES

[1] A. Adya, P. Bahl, R. Chandra, and L. Qiu, "Architecture and Techniques for Diagnosing Faults in IEEE 802.11 Infrastructure Networks," in $Proc. Mobicom$2004, 2004.
[2] I-M Chao I-M, B. Golden, and E. Wasil, "The Team Orienteering Problem, *European Journal of Operational Research*, Vol. 88, 1996, pp. 464-474.
[3] I-M Chao I-M, B. Golden, and E. Wasil, A Fast and Effective Heuristic for the Orienteering Problem, *European Journal of Operational Research*, Vol. 88, 1996, pp. 475-489.
[4] B. Golden, L. Levy, and R. Vohra, "The Orienteering problem," $Naval Research Logistics$, Vol. 35, 1987, pp. 307-318.
[5] D. Kotz et al Dartmouth Data, http://cmc.cs.dartmouth.edu/data/
[6] M. Kantor and P. Rosenwein, "The Orienteering Problem with Time Windows, *Journal of the Operational Research Society*, Vol. 43, No. 6, 1992, pp. 629-635.
[7] Y. Liang, S. Kulturel-Konak, and A. E. Smith, "Meta Heuristics for the Orienteering Problem," *Proc. 2002 Congress on Evolutionary Computation*, May 2002, Honolulu, Hawaii, pp. 384-389.
[8] A. Medina, A. Lakhina. I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation, " Proc. IEEE MASCOTS 201.
[9] A. Mishra, M. Shin, and W.A. Arbaugh, "Context Caching using Neighbor Graphs for Fast Handoffs in a Wireless Network," in $IEEE Infocom$2004, March 2004.
[10] A. Roberts, "Mathematical Research into the Orienteering Problem," http://www.qoa.asn.au/rroc/omaths.html
[11] S. Russell and P. Norvig, "Artificial Intelligence: A modern approach," Prentice Hall, Upper Saddle River, New Jersey, 1995.
[12] T. Tsiligirides, "Heuristics Methods Applied to Orienteering," *Journal of Operational Research Society*, Vol. 35, No. 9, 1984, pp. 797-809.