Techniques for Locating Service Faults in Mobile Ad Hoc Networks

Petr Novotny,[†] Bong Jun Ko,[‡] and Alexander L. Wolf[†]

[†]Imperial College London
London, UK

[‡]IBM T.J. Watson Research Center
Hawthorne, New York, USA

Imperial College London
Department of Computing
Technical Report DTR-2012-3 April 2012

ABSTRACT

Abstract

Fault localization in general refers to a technique for identifying the likely root causes of failures observed in systems formed from components. Fault localization in systems deployed on mobile ad hoc networks (MANETs) is a particularly challenging task because those systems are subject to a wider variety and higher incidence of faults than those deployed in fixed networks, the resources available to track fault symptoms are severely limited, and many of the sources of faults in MANETs are by their nature transient. We present a method for localizing the faults occurring in service-based systems hosted on MANETs. The method is based on the use of dependence data that are discovered dynamically through decentralized observations of service interactions. We employ both Bayesian and timing-based reasoning techniques to analyze the data in the context of a specific fault propagation model, deriving a ranked list of candidate fault locations. We present the results of an extensive set of experiments exploring a wide range of operational conditions to evaluate the accuracy of our method.

1 Introduction

Mobile ad hoc networks (MANETs) are used in difficult or remote situations, such as emergency response or forest-fire fighting, where a reliable, fixed-network communication infrastructure may be absent. However, the applications deployed upon MANETs are increasingly expected to exhibit sophisticated features, mimicking the availability of rich applications in "normal" network environments.

It is no surprise, then, that MANETs are now hosting applications that are architected as flexible and dynamic collections of *software services*. In service-based systems, such as those based on the Service-Oriented Architecture (SOA) or Web Services frameworks, computations are structured as sets of services that respond to requests, where a request typically originates at a user-facing client. The computation required to fulfil each request results in a cascade of further requests across some subset of the services; services make requests on other services in order to fulfil the requests made upon them. The set of messages exchanged during the processing of a single client request is referred to as a *conversation*.

We are concerned with a crucial operational aspect of service-based systems deployed on MANETs: *fault localization*. This is a particularly challenging management task because the systems are subject to a wider variety and higher incidence of faults than those deployed in fixed networks, the resources available to track fault symptoms are severely limited, and many of the causes of failures in MANETs are by their nature transient (e.g., a mobile host that temporarily moves out of radio range). Further complicating the situation is that faults at the network level may not manifest themselves as failures at the service level, since service-based systems are designed to mask certain kinds of faults through mechanisms for dynamic service discovery and (re)binding. Therefore, a fault localization method suitable for MANET-hosted service-based systems must be adept at sifting through noisy data.

In this paper we describe a method that makes use of two kinds of basic information: (1) a *service dependence graph* rooted at the client that initiated the conversation and reported the failure, and (2) various network- and service-level *fault symptoms* recorded in host-local logs. The dependence graph is discovered at run time by a decentralized monitoring system deployed in the network [10]. The monitoring system also collects the symptoms from the local logs. The graph and symptoms are used to carry out a centralized fault propagation analysis based on a *fault propagation pattern*, which indicates how faults can be propagated through the system from root causes to clients. The result of the analysis is a *fault propagation model* that relates possible root causes to the failure reported by the client. Finally, the faults are *ranked* for their likelihood of causing the failure.

This paper makes the following specific contributions:

- a method to synergistically analyze fault symptom data gathered at both the service and network levels;
- a fault propagation model based on propagation patterns devised specifically for the operational environment;
- timing-based and probabilistic root-cause ranking algorithms; and
- an extensive, simulation-based, experimental analysis.

Existing fault localization methods designed for the MANET environment [2, 3, 4, 5, 6, 8, 9] focus on network-level identification of individual faulty hosts and/or the links between them. The methods are limited to low-level observations and measurements of packet flows and host failures, and therefore are blind to the end-to-end context of the service-level conversations affected or unaffected by those faults. Our method instead makes use of both service- and network-level information. This enables a decoding of the fault propagation through the hosted services via the network. The client is therefore a good entry point for fault localization, tracing failures back to their sources using information that is unavailable from observations at the network level alone.

Clearly, our method is subject to inaccuracies due to effects such as the delay between data collection and data analysis, and inaccuracies in the discovered dependencies representing the system structure. The essence of the present work is to understand how these and other factors impact the accuracy of our method, subject to various tuning and environmental parameters. We focus on the formulation of a basic method that assumes the complete availability of dependence and symptom information for all non-failing hosts in the network. However, in the longer term, but beyond the scope of this paper, we are also studying how incomplete dependence and symptom information can be accommodated in fault localization.

We carry out the evaluation through a series of simulation-based experiments under various scenarios that represent a range of mobile-network dynamics, service dynamics, and critical parameter settings. Because no benchmarks yet exist for MANET-hosted service-based systems, we develop synthetic data to explore the space of independent variables. The primary dependent variable in these experiments is the *mean position of correct fault* in the ranked list of candidates.

We next review prior and related work in Section 2. In Section 3 we present the high-level architecture and assumptions we make about fault localization. In Section 4, we describe in detail our fault localization method, including its underlying fault propagation model and ranking algorithms. We then present our experimental evaluation of the method in Section 5, and conclude with a summary of our observations and a look at on-going and future work in Section 6.

2 Related Work

Although fault localization has been studied extensively for wired network environments (see Steinder and Sethi [13] for a survey), attempts to solve the problem for wireless networks are only more recent. Some of these newer approaches address the issue in the context of stationary, infrastructure-based wireless networks [1, 11, 12], but are not able to handle the dynamics of mobile ad hoc networks.

The few existing approaches to fault localization in MANETs can be generally classified into two broad categories.

In the first category fall those seeking to identify a set of fault-free hosts, for which methods based on *output comparison* have been proposed. In the basic method, pre-defined tasks are assigned to network hosts and their outputs are compared to ones defined *a priori*; a hosts is considered fault free if the results match. This approach was introduced into MANETs by Chessa and Santi [3]. In their approach, hosts continuously diagnose the status of their immediate neighbors and learn the status of all other hosts in the network via dissemination protocols. Variations and improvements, using different task assignment strategies and/or state dissemination techniques, have also been developed [4, 5]. Because these approaches focus on the identification of fault-free hosts through continuous monitoring and dissemination, they are indirect (and inefficient) methods for fault localization.

In the second category, more relevant to the subject of this paper, are methods seeking to identify individual faulty hosts and links in MANETs. Natu and Sethi [8, 9] use a temporal event-correlation approach with dynamic discovery of the network topology. Their method periodically identifies the end-to-end paths and the states of hosts based on correlations of measurements, identifying faulty hosts and links through active probing of paths. Fecko and Steinder [6] use a combinatorial approach and variance analysis to correlate the occurrences of multiple failures in a network. There approach is based on probabilistic dependence graphs obtained from expert knowledge of the intended system structure and the history of system faults. Cavalcante and Grajzer [2] propose a form of probabilistic fault propagation model based on Bayesian probability for a specific architectural model of dependable networks [15]. All these methods are limited by the fact that they focus on identifying the faults at low-level network components (hosts and links), not taking into account the applications actually making use of the MANET. Because network-level faults do not necessarily manifest themselves as failures in the application layer and (obviously) vice versa, there is a conceptual disconnect that severely reduces the effectiveness and usefulness of these methods. Our approach, by contrast, leverages information available at both levels.

3 System Architecture and Assumptions

The goal of our fault localization method is to find the root cause of a failure in a conversation initiated by a client of a service-based system. The method uses a set of distributed monitors, as illustrated in Figure 1. The monitors are deployed in service components (e.g., in a service container) and are responsible for extracting the symptoms of service failures from system logs. The monitors provide the symptoms, on demand, to a centralized fault localization element (e.g., located at an operator's station) that runs the fault localization method.

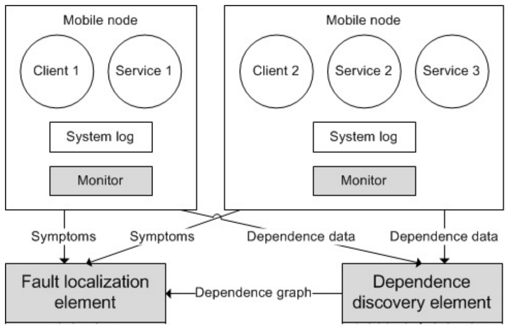


Figure 1: Architecture of fault localization.

An essential input to our method is a *dependence graph* (DG) that captures the run-time dependencies among services. In service-based systems, a *dependence* is a relation between services defined by the message flow induced by a client request. (As an edge case, a dependence is also the relation between a client and a service. Without loss of generality, we mainly focus here on relations among services.) When a dependence relation exists between two services S1 and S2, one service is considered the *source* and the other the *target*. In general, sources issue requests (i.e., method calls) on targets, thus defining a directionality to the dependence. Targets are expected to provide replies (i.e., response messages) back to sources. A DG can be used to represent the full set of dependence relations in the system, or can be restricted to a subset of those relations. Figure 2 shows a simple example of a DG rooted at a particular client.

In earlier work [10], we developed a method to obtain probabilistically accurate DGs in a MANET-hosted, service-based system. The method is designed to take account of the peculiarities of that environment, specifically the dynamic nature of service-level binding and host-level mobility, combined with the general lack of resources for tracking dependencies. The method works by building (i.e., "discovering") the DG for a client on demand, which is an approach that is particularly well suited to fault localization because it allows us to obtain the DG for the client that has reported a failed conversation.

Similar to symptoms, dependence information is gathered locally by distributed monitors deployed in service components that, in this case, observe the service-level message traffic (see Figure 1). The dependence

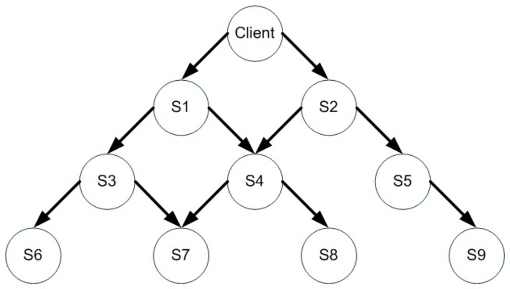


Figure 2: Dependence graph rooted at a client.

information is provided—on demand, with respect to a particular client, and covering a specific time period—to a centralized dependence discovery element that then produces a corresponding DG. Of course, due to the difficult operational environment, the resulting DG can be expected to contain some false dependencies, as well as miss some true dependencies, although the discovery method is designed to minimize the occurrence of such inaccuracies. We must therefore take account of this in the design of the fault localization method.

Besides the availability of dependence information, we make the following assumptions. First, each service records all failure symptoms in a local log, and the local monitor has access to the log. Second, log entries include information such as a time stamp, an identifier for the relevant host or service, and the type of symptom. Third, time stamps are globally consistent due to synchronized clocks. Clock synchronization in MANETs is a well-researched topic, with techniques available to achieve precision of tens or even single microseconds [16]. The smallest time period we use to distinguish time stamps is on the order of several milliseconds, well within this precision. Finally, there is a global timeout parameter shared by all clients and services. The timeout parameter defines the maximum waiting period for responses to requests. This is easily achieved in systems that use a common application platform, such as .NET or J2EE, which have a predefined default value. In systems that use a mix of platforms, explicit enforcement would be required.

4 Fault Localization

The previous section introduces the general architecture we envision for recording and gathering the information needed for fault localization. Assuming this information is available to the fault localization element shown in Figure 1, our method for analyzing the information consists of the following high-level steps:

1. A *fault propagation pattern* (Section 4.1) is mapped onto the dependence graph associated with a failed conversation, resulting in a *fault propagation model* (Section 4.2) that represents how specific faults might propagate through the individual services involved in the conversation.

2. The fault propagation model is combined with the occurrence times of relevant symptoms. This assignment allows the method to reconstruct the possible propagation paths for the fault, both in time and in space, and thus to form a set of plausible *candidates* for the root cause of the failure.

3. A *ranking* (Section 4.3) is applied to the candidate set based on their likelihood of being the root cause of the failure. We have devised two alternative ranking algorithms based on established fault localization techniques.

Before describing our technique in detail, we first introduce the terminology used throughout the description.

Fault: An exceptional condition occurring in a software or hardware component. A fault may or may not cause a failure in another component.

Failure: An external and visible manifestation of a fault as a deviation from the expected behavior of a component.

Fault propagation: A failure in one component may cause failures in dependent components that are otherwise fault free. Thus, a fault may propagate via a cascade of failures.

Symptom: An observable manifestation of a fault or failure. Symptoms are typically recorded in a system log and include ancillary descriptive information. We consider here two specific kinds of symptoms: ***exceptions***, which are explicit notifications of failures caused by a service or network fault, and ***timeouts***, which are implicit notifications of failures caused when an expected response message has not arrived within a specified period of time.

Root cause: The apparent origin of a fault propagation. We use the more specific terms *root-cause fault* and *root-cause failure* interchangeably, where the latter is the first visible (and, hence, observable) manifestation of the former.

4.1 Fault propagation pattern (FPP)

We describe how faults can propagate as failures from one component in a system to another by defining a *fault propagation pattern* (FPP). More precisely, an FPP is a recursive description of how different kinds of failures can cause further failures, where the first failure in the propagation chain is the failure manifesting the root-cause fault. In service-based systems, the propagation flows upstream from target services back to source services in the service dependence structure.

To enable a fine-grained analysis using propagation patterns, we classify the observable failure symptoms—exceptions and timeouts—into failure *modes*.

A propagated *exception* (Figure 3a) can be generated as a result of one of the following modes: (1) a send failure (SENDF), which represents a service that is unable to send a message to another service due to a network-level fault; (2) a software failure (SF), which represents any internal software or data fault that throws an exception; and (3) an exception (EX), which is itself generated by a service in response to a failure in a downstream service. Notice that the first two are (manifestations of) actual root causes, while the third is the recursive behavior that leads to propagation.

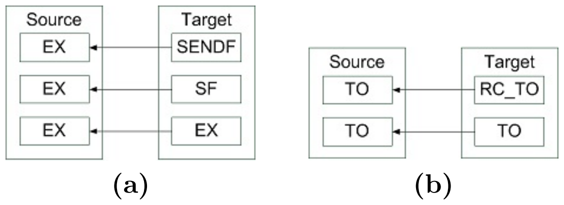


Figure 3: Fault propagation pattern refined into modes for exception failures (a) and timeout failures (b) in service-based systems. Faults propagate upstream from targets back to sources in the service dependence structure.

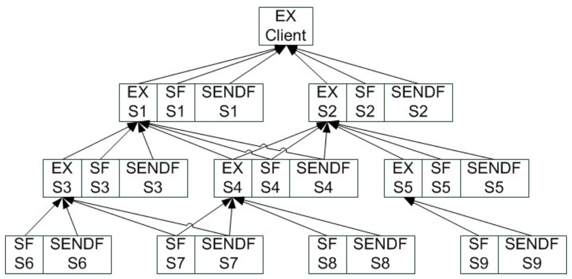
A propagated *timeout* can be caused only by some other timeout. This is because the timeout event is an implicit symptom of a fault whose real cause is not directly observable and, hence, not itself recordable as a symptom. Examples include a network host that moves out of communication range or the physical failure of a host. We identify two timeout modes (Figure 3b) that can cause further upstream timeout failures: (1) a root-cause timeout mode (RC_TO), which results implicitly from a network-level fault, and (2) a transitive timeout mode (TO), which results from a timeout occurring in some downstream service.

A subtle and counter-intuitive point: A timeout failure is consistently witnessed by a client *before* the root-cause RC_TO failure occurs. This is because the client starts its response timer when it initiates the conversation, and all downstream timers in the conversation are also set to this same default value, as mentioned in Section 3. Thus, the client's timer would actually be the *first* to expire in the conversation.

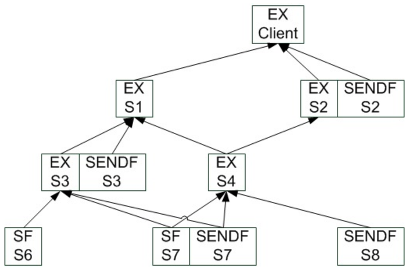
4.2 Fault propagation model (FPM)

Our technique makes use of a graph, called a *fault propagation model* (FPM), representing the causality relations between failure events in a system. An FPM is the result of mapping the fault propagation pattern (FPP) described above onto the service dependence graph (DG) of a given client. It is a rooted reverse-directed acyclic graph whose nodes and edges correspond to the client, services, and dependencies of the DG. More precisely, the graph contains a single node at its root to represent the failure mode witnessed by the client, and a set of nodes at its leaves to represent the failure modes of the candidate root causes. The internal nodes represent transitive failure modes. The edges signify possible failure propagation paths, which follow the reverse dependence edges in the DG. Figure 4a shows an example constructed by mapping the FPP of Figure 3 onto the DG of Figure 2.

Faults tend to be temporary in MANETs and can affect the clients only for a limited period of time. In order to capture this temporal aspect of the problem, an FPM typically maintains information concerning a specific *time window*, reflecting only the failure symptoms collected by monitors during that period. The



(a)



(b)

Figure 4: Fault propagation model constructed from exception modes of Figure 3 mapped onto dependence graph of Figure 2. The full FPM (a) is reduced (b) by considering actual symptoms recorded in a given time window.

selection of the time window size is critical in the construction of the FPM. For example, a small time window serves to reduce the size of the FPM, but some critical service interactions and symptoms might be missed. A large time window provides a more complete record of dependencies and symptoms, but might include stale or irrelevant interactions and symptoms (i.e, those belonging to conversations other than the conversation of concern to the analysis). In principle, one needs to consider how long it would take all potential root-cause faults to propagate to a client. Ideally, the time period should be long enough to cover the longest such propagation, but no longer, so as not to include considerable noise in the data. We examine this issue carefully in the context of our experimental evaluation (Section 5.1).

Operationally, upon receiving a failure report from a client, the fault localization element builds the FPM with respect to the reported failure. To do so, the dependence discovery element is queried for the DG associated with the failed conversation. The DG is then combined with the FPP and transformed into an FPM for the specific kind of failure, according to the following steps:

1. Every leaf service of the DG is transformed into the root-cause failure modes of the FPP; every intermediate node of the DG is transformed into root-cause and transitive failure modes; and the client node is transformed into a single transitive mode representing the client failure.

2. The edges representing cause-and-effect relations between modes are assigned based on the dependencies in the DG such that every mode of a target service is connected to the respective mode of a source service.

3. The fault localization element contacts the monitors of the services included in the DG/FPM in order to harvest locally aggregated symptom data for the time window of interest. The aggregation applied depends on the kind of failure mode associated with the recorded symptom: For exceptions it is the latest in the time window, while for timeouts it is the earliest. (The justification for these aggregations is given in Section 4.3.) The data are then assigned to the relevant modes of the FPM.

4. The FPM is *reduced* to a set of candidate root-cause failure modes based on the actual symptoms

recorded within the given time window. To do so, any modes (whether transitive or root cause) without associated symptoms are removed from the FPM, as they (probabilistically) could not have either caused or propagated to the client. The FPM is then further reduced to the set of candidate root-cause modes, which is connected to the client failure mode either directly or by propagation paths through the remaining transitive modes.

The resulting FPM contains the set of candidate root-cause failure modes that are reachable to the client directly or through transitive modes. The FPM also contains the set of transitive modes that are on propagation paths between at least one root-cause mode and the client. For example, the FPM of Figure 4a can be reduced to the FPM of Figure 4b if we assume the following symptoms are recorded in the given time window: services S6 and S7 experienced software failures (SF); services S2, S3, S7, and S8 experienced message send failures (SENDF); and services S1, S2, S3, and S4 received exception messages (EX). The client, which presumably triggered the analysis, also received an exception message (EX).

4.3 Ranking candidate faults

Given an FPM that consists of possible root-cause failure modes, the next step is to rank the candidate root-cause faults in the order of their likelihood of being the actual root cause of the client failure. We consider two approaches to ranking the candidates: one based on *occurrence time* and the other based on *Bayesian probabilities*. We compare the two approaches in our experimental evaluation (Section 5.2).

Timing-based ranking

The timing-based ranking approach follows from the following observation: Although conversations might last a long time, failures will propagate relatively quickly from root causes to clients. For instance, intermediate services typically generate exception responses immediately upon receiving an exception response from a downstream service in the conversation, ultimately causing the client to see the exception within a relatively short time after the root cause fault has occurred. Similarly, timeout events in the services involved in a conversation tend to occur within a relatively short period. This is due to the fact that services set their timeout timers when they send service requests, and the forward propagation of those requests through the system is typically much quicker than the time it takes for the conversation as a whole to complete.

Based on this observation, our timing-based algorithm ranks the candidate root-cause failures appearing in the reduced FPM in increasing order (from shortest down to longest) in terms of the difference between the time stamp of the client failure symptom and the time of occurrence of the candidate, which is itself determined by the time stamp of:

- the *latest* symptom of the failure, in the case of an exception, and
- the *earliest* symptom of the failure, in the case of a timeout.

This aggregation ensures that the ranking favors the candidate that occurs closest in time to the failure witnessed by the client.

Bayesian-based ranking

Our second algorithm assigns the ranks to the candidates in the order of *probability* that the candidate is the root cause. The probabilities are inferred on the Bayesian network (BNet) constructed from the FPM. We use a multi-level BNet model, where each node of the graph is a binary valued random variable that describes the state of a single FPM mode.

The BNet is an isomorphic transformation of the reduced FPM. However, the direction of the BNet edges is reversed from FPM edges because in the BNet we measure the probability of each candidate node to be a root cause, given the evidence that the client witnessed a failure. Thus, we measure the probability of propagation in reverse order.

For the failure propagation probabilities in the BNet, we use the noisy-OR gate distribution model of the conditional probability distribution (CPD), due to its computational and space efficiency [7]. The CPDs are calculated based on two assumptions concerning fault propagation. First, given that the reduced FPM only provides the relevant subset of candidate root causes, the probability of the antecedent mode to propagate to a dependent mode is equal for all transitions. Second, the probability of a candidate to be the root cause decreases with the increasing number of transitions through which the fault propagates to the client.

Therefore, we calculate the CPD of a node in the BNet such that the conditional probability of a parent node in CPD given the occurrence of the mode in a child node (i.e., the probability of propagation from parent to child node) is equal to the proportional fraction that the parent represents in the set of all parents of the particular child node in the FPM. For example, a child node with two parents has a CPD of 50% propagation probability of/from each parent. Furthermore, in order to reflect the decreasing effect of each transition on the probability of the mode to be the root cause, the conditional probability of each parent is modified with the constant 0.99. This constant is sufficient to distinguish between the candidate modes at different levels of the FPM, yet small enough not to interfere with the parent probabilities.

For the inference of the posterior probability of the root cause modes, we use the junction-tree algorithm [14], which is an exact inference algorithm suitable for small-to-medium sized, multi-level directed acyclic graph models such as those we expect in our analysis context.

5 Evaluation

In this section we present an evaluation of our fault localization method. We begin by describing the experimental methodology and evaluation criteria.

5.1 Methodology and evaluation criteria

The evaluation of our fault localization method is based on a simulation framework for service-based systems hosted on MANETs [10]. The framework consists of a simulation engine built on top of the discrete-event network simulator NS-3,[1] extended with higher abstractions for simulating service entities and their interactions. As shown in Figure 5, the engine first generates message and fault traces according to various simulation parameters and scenario configurations. These traces are stored in a database to hold the results of the simulation runs. Dependence discovery, the causality analysis that contributes to an FPM, and the timing-based ranking are prototyped as queries over this database. For Bayesian-network ranking we use MATLAB[2] with the Bayes Net Toolbox.[3]

The evaluation questions of interest center mainly on the accuracy of the resulting candidate set, specifically the ranking of root-cause candidates. A good result, of course, would be that our method consistently ranks the correct root-cause candidate at or near the top. We examine the following four factors that might influence accuracy:

- **Ranking algorithm:** We compare the two ranking algorithms, timing and BNet, in terms of the position at which they place the correct root-cause failure.
- **Dependence graph accuracy:** The dependence graph (DG) that results from the dependence discovery mechanism may contain some services that are irrelevant to the conversation of interest, causing the FPM induced from the DG to include failure symptoms irrelevant to the failure observed by the client. We examine the affects of DG accuracy—more precisely, the ratio of false positives—used in the construction of an FPM.
- **Service connectivity:** We examine the impact of service connectivity, which represents the complexity of a service-based system. The higher the degree of interconnection between the services, the larger the number of services and overlap among conversations, and the more noise in the dependence and symptom data.

[1] `http://www.nsnam.org/`
[2] `http://www.mathworks.com/products/matlab/`
[3] `http://code.google.com/p/bnt/`

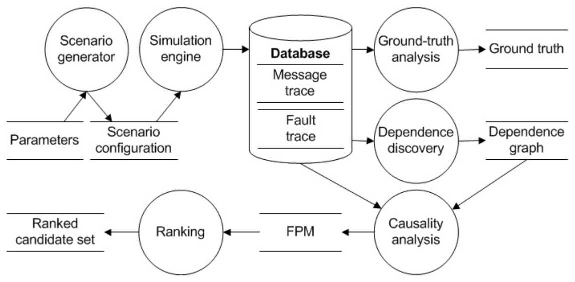


Figure 5: Experimental framework.

	Low	High
Services in dependence graph (avg.)	2.03	7.9
Services in dependence graph (std. dev.)	1.2	3.63
Number of conversations in scenario	8541	6810
Ratio of failed conversations	5.01%	20.88%
Proportion of SF root causes	16.6%	16.0%
Proportion of SENDF root causes	75.7%	76.8%
Proportion of TO root causes	7.7%	7.2%

Table 1: Dependencies and faults induced by connectivity.

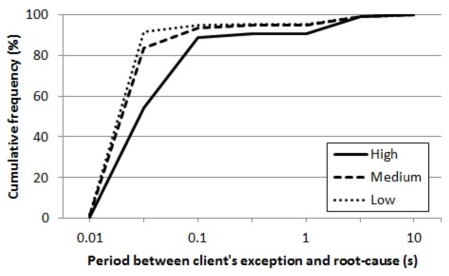
- **Service fault rate:** We examine the sensitivity of the ranking algorithms to a range of service fault rates. We would expect that as the rate increases, the algorithms might have difficulty maintaining consistent results.

For the network-level behaviors in the simulation, such as wireless signal propagation models, we use standard settings used widely in the networking community and embodied in the NS-3 simulator. We configure a network of 50 hosts, and simulate the random mobility of the hosts at a fairly high speed (10 m/s). The significance of high mobility is that the hosts are frequently disconnected from each other and, as a result, the message exchanges between the services fail frequently, causing message send failures (SENDF) and timeout failures (RC_TO) in the context of our fault models.

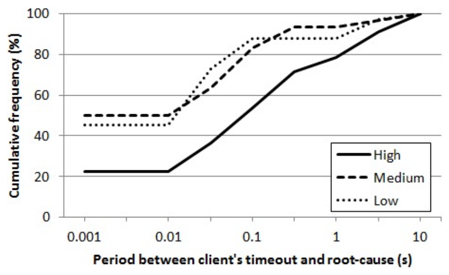
The service-level parameters used in our simulations are derived from standard values found in SOA and Web Services implementations. We generate a system of 50 clients and 30 services, with the services arranged into five "front end" (client facing) and 25 "back end" services. In addition to the network-level faults, we also cause two kinds of service-level faults: In the first kind, the service fails to generate proper responses to the service requests and instead generates exceptions, while in the second, the service itself fails due to, for example, a host failure, and does not generate any response. In our context, a failure of the first kind corresponds to the root cause of an EX failure, while the second results in the root cause of a TO failure. All but the last of our experiments use a service fault rate of 0.5%, which means 5 out of every 1000 service requests fail. We configure the simulations such 90% constitute exception root-cause faults (SENDF or SF) and 10% timeout root-cause faults (RC_TO).

We collect our results from 30 minutes of simulated execution time after excluding a 30 second warm-up period. Each combination of parameters results in thousands of conversations occurring during the simulation, among which the failed conversations constitute the statistical samples of our analysis subject to the randomness in the service request times, host mobility, wireless signal fading, and the like.

Table 1 summarizes the failure symptoms observed in our simulations, where the columns "Low" and "High" represent sparse and dense service connectivity scenarios, respectively. For instance, the low connectivity configuration (combined with 0.5% service fault rate and 10 m/s mobility speed) results in 8541 conversations. The average number of services in the dependence graph of each conversation is larger in highly connected service configurations, as is the ratio of the failed conversations: about 21% of all conver-



(a)



(b)

Figure 6: Cumulative distribution of time between occurrence of root-cause faults and when clients witness exception messages (a) or timeouts (b).

sations fail in the high connectivity scenario, compared to about 5% in the low connectivity scenario. Of all root-cause failures, SENDFs constitute about 75%. These are caused by faulty network links, rendering services unable to successfully send request messages to other services. In comparison, timeout failures occupy a small portion of only about 7%.

Selection of time window size

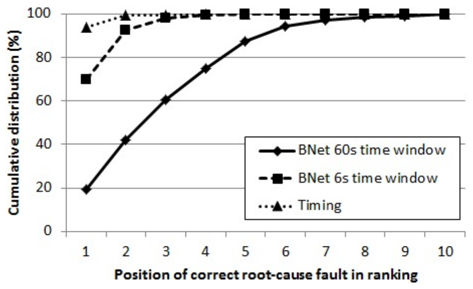
As mentioned in Section 4.2, the time window is an important tuning parameter for our method, as it affects which failure symptoms are included in the FPM. When selecting the size of the time window, one needs to consider how long it would take for root-cause faults to propagate to clients. A naïve approach would be to set the value equal to the longest duration that a conversation can last, which in turn is limited by the service response timeout parameter, since any failure in the system should occur within this period. Following standard .NET and similar implementations, the timeout parameter is set to 60 seconds in our experiments. In practice, however, the failure propagation time can be much shorter than the duration of the entire conversation, and hence the time window size also needs to be selected accordingly.

Figure 6 gives the cumulative distribution of the propagation time for exception and timeout root-cause faults to reach clients in each of the two service connectivity scenarios. We can see from the distributions, regardless of the kind of failure and service connectivity, that the time difference between the failure seen by the client and the root-cause failure is fairly small: in almost all cases, the failures are propagated within a few seconds. This result suggests that the time window size indeed needs to be set much smaller than the maximum duration for the conversation so as not to include too many irrelevant symptoms in the analysis. In the results below, we generally use a 6-second time window for this purpose. Note, however, this value is chosen based on empirical results, and hence should be considered as one that is close to ideal choice. To reflect the cases when such an empirical basis is not available, results with the 60-second time window are also shown whenever applicable.

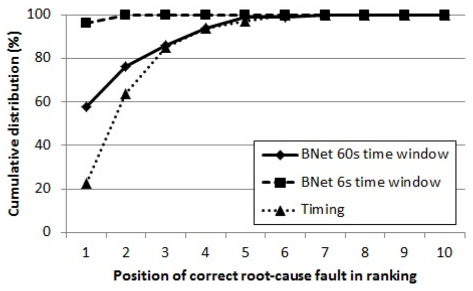
5.2 Comparison of ranking algorithms

We first compare the accuracy of the two ranking algorithms. To isolate their effect from those of other parameters, we show the results for high service connectivity with 0.5% service fault rates, using 100% accurate dependence graphs (referred to as "ground truth" in Figure 5), which contain exactly the services involved in each conversation.

Accuracy is measured in terms of the ranking position of the correct root-cause fault for each failed conversation in the experiment. The results are given in Figure 7, which shows the cumulative distribution of those positions. For purposes of comparison, we include the results for the BNet-based algorithm under both 6- and 60-second time windows; The results by the timing-based algorithm are virtually the same for both time window sizes. Notice that the average sizes of the candidate sets under the 6-second time window are 1.5 for exceptions and 2.6 for timeouts, and 5 and 3.5, respectively, under the 60-second window.



(a)



(b)

Figure 7: Accuracy of ranking algorithms for exception (a) and timeout (b) faults in high connectivity scenario. The FPM is constructed from the ground-truth dependence graph.

We can see that the timing-based method is very effective for exception failures, resulting in more than 90% of correct root causes placed in the first position of the ranking, under both 6-second and 60-second time windows. Compare this with an ordering of candidates by random guess, which would do so only for 66% and 20% under 6- and 60-second time windows, respectively. This good result is due to the fact that exception failures are propagated through a quick succession of explicit messages, which makes a ranking based on the time proximity particularly suitable for such failures.

On the other hand, the BNet-based algorithm is more effective in ranking the root causes of timeout failures. Under a 6-second time window size, the BNet algorithm places the correct root causes in the first position for more than 95% of the failures, and for nearly 60% under 60-second time window. Compare this again with a random ordering, which would achieve the same only for 38% and 28% of the cases under the two time window sizes, respectively. The accuracy of the timing-based algorithm is not nearly so good for the timeout failures.

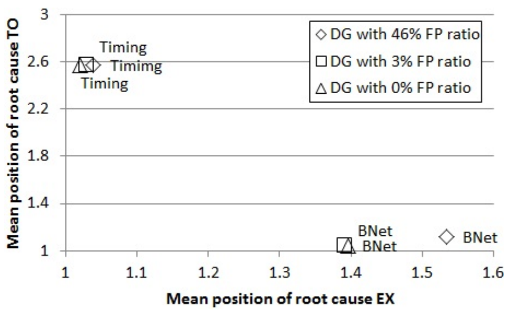


Figure 8: Impact of dependence graph false positive ratio on mean ranking position of correct root-cause fault. Ratio of 0% represents the ground truth.

5.3 Impact of dependence graph accuracy

We next investigate the sensitivity of our method to the accuracy of the dependence graph (DG) rooted at a client. Recall that the DG is created using a run-time discovery facility that produces probabilistically correct results [10]. Its accuracy can be measured in terms of any extraneous dependencies included in the graph. An extra dependence is a false positive (FP) drawn from some other, irrelevant conversation. The ratio of false positives to true positives (the FP ratio) is influenced by various factors, such as mobility speed, service workload, and most significantly the particular technique used for dependence discovery.

Here we make use of two dependence discovery techniques. The details of those techniques are given elsewhere [10], suffice to say that they are represented here, respectively, by the FP ratios 3% and 46% in our experiments. We also include in our experiments the ground-truth DG that contains no false positives. Again, we show results for the high connectivity scenario and 6-second time window.

Figure 8 shows the impact of the FP ratio on the accuracy of our method under 6-second time window. The average ranking positions of correct root causes for exception and timeout faults are plotted against each other and shown for the two ranking algorithms. Consistent with the results in Section 5.2, the algorithms behave differently for the two kinds of faults. Notice that the timing-based algorithm is essentially insensitive to the accuracy of the DG, which reinforces our conclusion that its effectiveness at localizing exception faults is dominated by the time proximity of the fault occurrence. The BNet algorithm also exhibits only some small sensitivity to the DG's accuracy.

5.4 Impact of service connectivity

We now turn to the impact of service connectivity (i.e., system complexity) on the accuracy of our method. It is important to investigate this factor because it influences many other contributing factors, such as the size and false positive ratios of dependence graphs, the duration of conversations (and, consequently, the amount of overlap between separate conversations), and the overall failure rate exhibited by the system. We hypothesize that the accuracy of the fault localization method will suffer under highly connected services, since the combined effect of the increases in the above factors should have a negative impact on the ranking position.

We present our results in Figure 9, where we compare the low and high connectivity scenarios using the same sort of plot as in Section 5.3. These results confirm our hypothesis: In the low-connectivity scenario, our method achieves significantly better accuracy for both ranking algorithms than in the high-connectivity scenario. The better result for low connectivity is due to the relatively small number of services involved in each conversation (on average, 2.03 versus 7.9), the relatively low degree of overlap between conversations, the relatively low dependence graph FP ratio (0.2% versus 3%). and a lower inherent failure rate (5.01% versus 20.88%). The reverse is true for the high-connectivity scenario.

Notice that in our other experiments we use the settings for a high service connectivity. This means that those results are based on a substantially more challenging scenario.

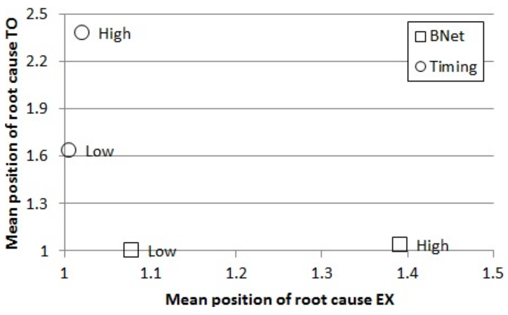


Figure 9: Impact of service connectivity on mean ranking position of correct root-cause fault.

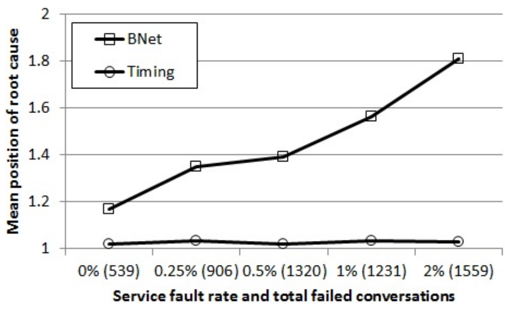


Figure 10: Impact of service fault rate on mean position of correct root-cause exception faults. Both the fault rate and the total number of failed conversations for each rate are shown.

5.5 Impact of service fault rate

In our final experiment, we investigate the accuracy of our method under a range of service fault rates. Recall that we configure the simulations such that when a client experiences a failure, 90% of the time the root cause is an exception.

The results are shown in Figure 10 for both ranking algorithms, under the high connectivity scenario, and a 6-second time window. The service fault rate is varied in the x-axis from 0% to 2%. For each rate, we also give the total number of failed conversations. Note that even with a 0% service fault rate, the scenario reflects failed conversations caused by network faults. Only the results for exception faults are shown, since the effect on the much lower 10% proportion of timeout faults is minimal.

The results demonstrate that the timing-based ranking algorithm is fairly robust in a wide range of service fault rates, maintaining consistently high accuracy even under high fault rates. The accuracy of BNet-based ranking, however, deteriorates as the service fault rate increases. This is because the BNet algorithm is sensitive to the size of the FPM, which increases not only due to inclusion of irrelevant dependencies (recall Figure 8), but also due to the increase in the fault rate, causing some irrelevant failures from overlapping conversations to be included in the FPM.

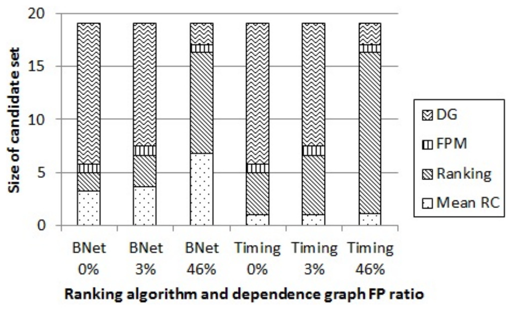
5.6 Summary of results and discussion

The experimental results above demonstrate the effectiveness of our fault localization method in ranking the correct root causes of failed service conversations. In particular, the timing-based ranking is shown to be very effective in localizing the failures caused by service exceptions, while the Bayesian-based ranking is more effective in the case of timeout failures. This suggests these two algorithms can be separately employed according to the kind the failure reported by a client.

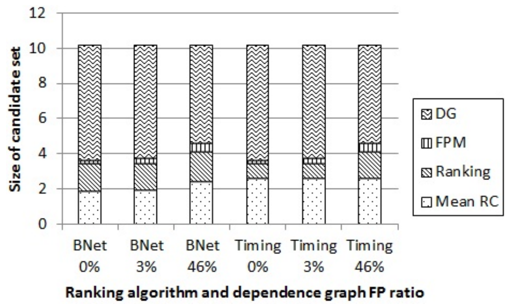
The results also show the importance of the proper selection of the time window size, where a value much smaller than the longest duration of the conversations is shown to provide good outcomes, reflecting a relatively quick propagation of the failures in the system, compared to the time it takes to complete the conversations.

An important consideration in the MANET environment is the requirement for data storage and transfer. Our method performs well in this regard. The method does not store any of its own data on hosts beyond dependence data, which elsewhere we showed to be highly compact [10]. Instead, the method makes use of standard log information typically already stored at a host. Communication costs are minimized by collecting the necessary data on demand in aggregated form; we have calculated that on average less than 1KB of payload is required for a given client failure analysis.

We conclude our evaluation with a look at how each aspect of our fault localization method contributes to the overall results. To do this, we break down the method into the contributions of its three main elements, each of which is applied in turn to obtain a result: (1) the dependence graph, (2) the fault propagation model, and (3) the ranking algorithm. To see the contribution of each element, we start from the total number of candidate root-cause faults observed within the selected time window. We then determine how many are eliminated at each step of the method. We use the high-connectivity scenario with 0.5% service fault rate and 60-second time window size. The larger window size allows us to see the reduction in each step more clearly; results for the 6-second window size show similar trends.



(a)



(b)

Figure 11: Contribution of fault localization method steps to exception (a) and timeout (b) fault analysis.

The results are shown in Figure 11. The height of each bar represents the size of the candidate set. As noted above, there are 19 and 10 candidate root causes on average for the exception and timeout failures, respectively. (The total number of *symptoms* observed by the monitors is much higher—102 and 13 failure symptoms for exception and timeout failures, respectively—but as the monitors report only *aggregated* data, the number of candidates is much smaller.) Each section of the bar represents the number of candidates eliminated by an element of the method: the DG area represents the reduction to those services appearing in the dependence graph rooted at the client that witnessed the failure; the FPM area represents the reduction to those reachable to the client in the FPM; the Ranking area represents the reduction to those including the correct root cause; and the Mean RC area represents the final mean position of the correct root cause.

Overall, we can see that each element indeed makes a contribution. The specific contribution differs with the ranking algorithm and accuracy of the dependence graph. Indeed, the ranking algorithm and dependence graph play the major roles in localizing the root cause of client failures. The effect of DG accuracy is particularly obvious when the 46% FP ratio is compared against those of 0% and 3% for exception faults, suggesting that it is important to have an effective dependence discovery in diagnosing such failures. The timing-based ranking algorithm contributes a significant portion in narrowing down exception faults, even when the DG and FPM is not effective. The BNet-based ranking algorithm is more effective for timeout faults and less so for exception faults. Again, this supports our suggestion to apply the ranking algorithm most appropriate to the reported failure.

6 Conclusion

We have presented a new fault localization method suitable for service-based systems hosted on mobile ad hoc networks. The method analyzes the failures experienced and reported by clients of the service-based system in order to locate the root-cause fault, making use of symptoms observed at both the service and network levels. To cope with the temporal aspects of the problem, the method uses multiple elements to filter out irrelevant symptoms. These include a dynamic dependence graph rooted at the client, a fault propagation model, and algorithms to rank candidate root causes. An extensive set of simulation-based experiments demonstrates that our method achieves fault localization results with high accuracy in a range of situations. It does so while incurring low data storage and transfer costs, making it ideally suited to the resource-constrained MANET environment.

We are working to address other open issues in the context of localizing service-based system faults in MANETs, such as methods for adaptive tuning of the parameters used in dynamic dependence discovery and fault symptom collection, and fault localization under incomplete and partial information.

Acknowledgments

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] A. Adya, P. Bahl, R. Chandra, and L. Qiu. Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 30–44, New York, NY, USA, 2004. ACM.

[2] A. Cavalcante and M. Grajzer. Fault propagation model for ad hoc networks. In *Proceedings of the IEEE International Conference on Communications*, pages 1–5, June 2011.

[3] S. Chessa and P. Santi. Comparison-based system-level fault diagnosis in ad hoc networks. In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, pages 257–266, 2001.

[4] M. Elhadef, A. Boukerche, and H. Elkadiki. Diagnosing mobile ad-hoc networks: Two distributed comparison-based self-diagnosis protocols. In *Proceedings of the 4th ACM International Workshop on Mobility Management and Wireless Access*, pages 18–27, New York, NY, USA, 2006. ACM.

[5] M. Elhadef, A. Boukerche, and H. Elkadiki. A distributed fault identification protocol for wireless and mobile ad hoc networks. *Journal of Parallel and Distributed Computing*, 68(3):321–335, Mar. 2008.

[6] M. Fecko and M. Steinder. Combinatorial designs in multiple faults localization for battlefield networks. In *Proceedings of the IEEE Military Communications Conference*, volume 2, pages 938–942, 2001.

[7] D. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, pages 163–172, 1989.

[8] M. Natu and A. Sethi. Adaptive fault localization in mobile ad hoc battlefield networks. In *Proceedings of the IEEE Military Communications Conference*, pages 814–820, Oct. 2005.

[9] M. Natu and A. S. Sethi. Using temporal correlation for fault localization in dynamically changing networks. *International Journal of Network Management*, 18(4):301–314, Aug. 2008.

[10] P. Novotny, B. J. Ko, and A. L. Wolf. Discovering service dependencies in mobile ad hoc networks. Technical Report DTR-2012-2, Department of Computing, Imperial College London, Feb. 2012 (under submission).

[11] L. Paradis and Q. Han. A survey of fault management in wireless sensor networks. *Journal of Network and Systems Management*, 15(2):171–190, June 2007.

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

[13] M. Steinder and A. S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004.

[14] T. A. Stephenson. An introduction to Bayesian network theory and usage. Technical Report IDIAP-RR-03-2000, Dalle Molle Institute for Perceptual Artificial Intelligence, Martigny, Switzerland, Feb. 2000.

[15] N. Tcholtchev, M. Grajzer, and B. Vidalenc. Towards a unified architecture for resilience, survivability and autonomic fault-management for self-managing networks. In *Proceedings of the International Conference on Service-Oriented Computing Workshops*, number 6275 in Lecture Notes in Computer Science, pages 335–344. Springer, 2010.

[16] D. Zhou and T.-H. Lai. An accurate and scalable clock synchronization protocol for IEEE 802.11-based multihop ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(12):1797–1808, Dec. 2007.