On-Demand Discovery of Software Service Dependencies in MANETs

Petr Novotny, Bong Jun Ko, and Alexander L. Wolf, *Fellow, IEEE*

***Abstract*—The dependencies among the components of service-oriented software applications hosted in a mobile ad hoc network (MANET) are difficult to determine due to the inherent loose coupling of the services and the transient communication topologies of the network. Yet understanding these dependencies is critical to making good management decisions, since dependence data underlie important analyses such as fault localization and impact analysis. Current methods for discovering dependencies, developed primarily for fixed networks, assume that dependencies change only slowly and require relatively long monitoring periods as well as substantial memory and communication resources, all of which are impractical in the MANET environment. We describe a new dynamic dependence discovery method designed specifically for this environment, yielding dynamic snapshots of dependence relationships discovered through observations of service interactions. We evaluate the performance of our method in terms of the accuracy of the discovered dependencies, and draw insights on the selection of critical parameters under various operational conditions. Although operated under more stringent conditions, our method is shown to provide results comparable to or better than existing methods.**

***Index Terms*—Network and service management, wireless and mobile networks, enabling technologies for management, software services.**

I. Introduction

UNDERSTANDING the dependencies among the components of a distributed system is critical to making good operational and maintenance decisions. For example, fault localization and change impact analysis are tasks enabled by accurate and timely data on component dependencies. The importance of dependence information increases with the complexity of the system, both in terms of the number of interacting components required to carry out a given computation and the nature of the environment in which the system operates.

In service-based systems, such as those based on the Service-Oriented Architecture (SOA) or Web Services frameworks, computations are structured as a set of services that respond to requests, where a request typically originates at a user-facing client. Fig. 1 illustrates such a system, with client applications and services deployed across the nodes of a network. 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 to fulfil the requests made upon them. Obtaining dependence information in such a system is made difficult by the inherent loose coupling of services, as many dependencies are unknown at design time, and only established at run time through a dynamic, run-time service binding mechanism (so-called "service discovery"). This run-time dynamisim is felt even more so when the service-based system is deployed on the mobile nodes of an ad hoc network, where the mobility itself can force reconfigurations of service bindings. The consequence is that the dependencies among run-time instances of services are not something that can be reliably specified before execution, but instead must be discovered during or after execution.

Existing dependence discovery methods focus on statically structured systems operated in fixed networks [2]–[6], [9], [11], [14], [20], [23]. A critical assumption made by these methods is that the dependence data, although changing, is relatively stable over time. The significance of the stability assumption is that the methods can make use of statistical techniques based on data collected over long execution periods. Furthermore, by operating in the context of a fixed-network environment, the methods can assume no practical limits on the storage, computational, and communication resources needed to support those statistical techniques.

The context for our work is instead service-based systems deployed on mobile ad hoc networks (MANETs) [1], [10], [16], [21]. Mobility and ad hoc networking bring increased dynamicity to service dependencies, beyond those caused by the basic run-time service-binding regime of SOA or Web Services. Moreover, the MANET environment is typically characterized by severe limitations on the resources available for dependence discovery. Existing methods based on the stability assumption cannot adequately cope with such high levels of dynamicity nor stringent resource constraints.

We have formulated a relatively simple dependence discovery method that is nonetheless more suitable for services operating in the challenging MANET environment. Our intuition is that dependence discovery must be focused on capturing snapshots of dependence data relevant to each service request of concern, rather than on the tradition of determining statistical averages for long-term, system-wide dependencies as a whole. Furthermore, the method must be lightweight in its resource usage, which to our thinking means that dependence data should be collected locally, aggregated locally, and drawn to some central location only when and if needed. We have designed the method to allow engineers to trade accuracy against cost (i.e., data collection overhead), yielding probabilistically accurate dependence graphs that nonetheless support useful

Manuscript received September 1, 2014; revised February 1, 2015; accepted February 25, 2015. Date of publication March 5, 2015; date of current version June 12, 2015. The associate editor coordinating the review of this paper and approving it for publication was X. Fu.

P. Novotny and A. L. Wolf are with the Department of Computing, Imperial College London, London SW7 2AZ, U.K.

B. J. Ko is with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA.

Digital Object Identifier 10.1109/TNSM.2015.2410693

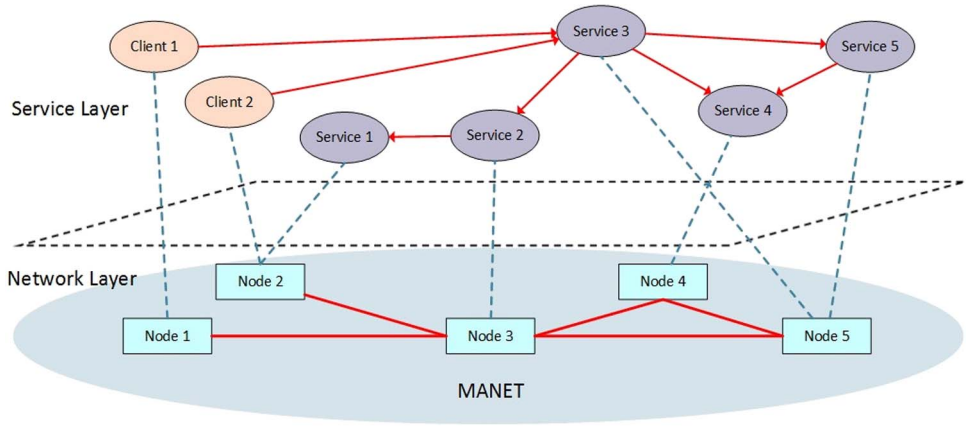


Fig. 1. Hosting service-based systems on MANETs.

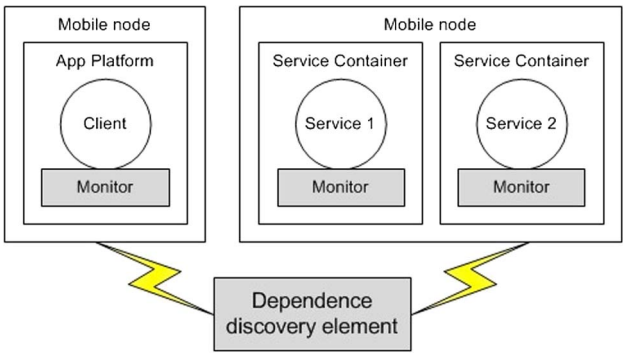


Fig. 2. Monitoring architecture.

analyses. In particular, the method has been successfully used in design of a probabilistic fault localization analysis [17] (briefly sketched in Section IV) and a cross-layer performance anomaly diagnosis [22].

Our approach is based on the use of *monitors* deployed onto the mobile nodes, as illustrated in Fig. 2. The monitors collect dependence data by observing the message traffic between services and extracting relevant information. The data collected by the monitors provide only a local view of the dependence information. When a more global picture of the dependence relationships among the services is required (e.g., to carry out some particular analysis), the monitors are contacted by a central discovery element charged with integrating the data. Importantly, only the monitors relevant to a particular analysis question typically need to be contacted, and therefore communication can be reduced. Moreover, the monitors can aggregate the data they collect, and can impose limits on the amount of data they store.

Three prerequisites must be met to use our method. First, to obtain complete dependence information, the monitors should be deployed on the nodes that are either the source or the target of service messages; intermediate nodes in the network used only to store and forward network-level messages are not involved in data collection. Second, the monitors need access to synchronized clocks to allow consistent time-stamping of the collected dependence data. Clock synchronization in MANETs is a well-researched topic, with techniques available to achieve precision of tens or even single microseconds [15], [24]. The smallest time scale we use to time stamp aggregated data is on the order of a few milliseconds, well within this precision. Third, the monitors must be able to observe service messages and obtain certain information from those messages, including such things as client and service identifiers. On the other hand, there is no need for the monitors to have access to the payload of messages. This kind of general information is typically available and visible, since it is used by the underlying service infrastructure to manage service interactions.

In this paper we introduce our dependence discovery method and evaluate its sensitivity to a distinguishing aspect of the MANET environment, namely *time-dependent behavior*. Clearly, the method is subject to inaccuracies due to such effects as the delay between data collection and data analysis, storage constraints that might require monitors to "forget" some data, and failures in nodes and links. The essence of the present work is to understand how these effects impact the accuracy of dependence discovery, subject to tuning and environmental parameters. Because no benchmarks yet exist for MANET-hosted service-based systems, we develop synthetic data to explore the space of independent variables. The dependent variables in these experiments are the *true positives* and *false positives* in the discovered dependence relationships.

Our evaluation is carried out through a series of simulation-based experiments under a realistic usage scenario: advanced software services supporting the collaborative activities of a group of 50 mobile users, such as a military unit, a team of fire fighters, or an emergency rescue squad, operating in a confined area without access to an infrastructure network. Each group member is equipped with a mobile computer device providing mission applications that access a range of services, such as those for constructing and rendering street maps, surface topography, group-member locations, tasking orders, sensors, and the like, hosted both locally and on other nodes in the network. The services vary in their degree of interdependence. Our results for this scenario indicate that for a system of 30 services, with dependence chains ranging from two to eight services deep, our method provides nearly perfect accuracy (true positives), with only a minor proportion of falsely included dependencies (false positives) as compared to existing methods requiring significantly more data and resources. Moreover, our method appears to be less sensitive to network mobility and workload factors, and hence robust to variations in operational conditions.

We next review prior and related work. Section III presents our dependence discovery method and its underlying dependence model, and Section IV a sample analysis, fault localization, that is based on the dependence data provided by the method. Section V describes the experimental approach we used to evaluate the method. We then present our experimental evaluation in Section VI. We conclude with a summary, and a brief look at on-going and future work in Section VII. The present paper extends a preliminary version appearing elsewhere [18].

II. Background and Related Work

Existing dependence discovery methods can be generally classified as to whether they operate at the network level or at the (application) service level. Network-level discovery [2]–[5], [11], [12], [14], [19] focuses on coarse-grained dependencies between network hosts. Network-level dependencies are usually described in terms of IP addresses and port numbers. They can be augmented with additional information, such as port mappings [9] or a classification of client applications [20]. On the other hand, service-level discovery [6]–[8], [23] focuses on the detection of fine-grained relationships between services. Services are hosted in application containers, such as J2EE or .NET, and typically associated with application identifiers, such as URIs.

Our dependence discovery method, although informed by the network level, operates at the service level in the sense that we wish to discover service dependencies that can be used for fault localization in service-based software systems. In particular, we focus on discovering service-level dependencies in MANETs, where dependencies may change at a rapid rate. It is important to point out that we do not assume the availability of prior information, such as a port mapping, application categories, or even a specification of the services, nor do we require changes in existing software components to support discovery, as assumed for other methods [5], [8].

Many of the service-level discovery methods apply statistical techniques to traffic traces collected by network hosts and monitors. The statistical techniques correlate network packets or service-level messages and identify co-occurrences of messages across different services. While the particular statistical techniques may differ (e.g., correlation based on a time window [2]–[4], delay distribution [9], time difference of messages [6], or timing and frequency of packet flows [11]), all of the methods share the same limitation: they require a long period of time to collect statistically stable data and, therefore, are inappropriate in highly dynamic environments such as MANETs.

Alternatives to statistical approaches exist. For example, Macroscope [20] samples and analyzes a subset of packets and network connections to identify relationships between network flow data and applications. Lu *et al.* [14] collect system log data and correlate the events in the logs using data-mining techniques. Magpie [5] instead correlates events based on input from (human) network operators. These methods, however, require the transfer of large amounts of trace data from the collection points to a central analysis element, which is prohibitive in resource-constrained MANETs. In contrast, our method transfers dependence information selectively and on demand, contacting only the monitors that are potentially relevant to the events of interest (e.g., the possible receivers of a failed service request). In addition, our monitors aggregate the dependence data before they are transferred to the analysis node (e.g., they may transfer only the number of messages exchanged between two services, rather than sending the content of those messages).

Pinpoint [8] uses a technique for correlating messages that resembles the one found in our method. The technique uses identifiers to uncover the flows of end-to-end processing through application servers hosted in a fixed network. Pinpoint requires servers to insert the identifiers into the headers of messages sent over extensible protocols such HTTP or SOAP. The identifier is passed across processing threads within components and inserted into messages. Monitors deployed in the network record requests into traces from which the paths are reconstructed, post hoc, and further aggregated into dependencies. Similar to Pinpoint, we make use of what service frameworks call *conversation identifiers* appearing in messages. However, beyond the common use of some sort of identifier scheme, the Pinpoint method for inferring dependencies differs substantially from ours and, indeed, is not viable in the constrained environment of a MANET: Pinpoint assumes a single, global respository to hold large amounts of data to statistically detect any occurences of, and changes to, dependencies, whereas our method stores and draws only a small portion of the data to a central analysis point, and then only on demand as needed.

III. Dependence Discovery and Representation

In this section we present our dependence discovery method. We begin by describing two types of dependencies in which we are interested. We then describe how we discover dependencies, represent dependencies, and construct the representation.

A. Service Dependencies

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 on targets, thus defining a directionality to the dependence.

We are concerned with two types of dependencies over a given set of services: *inter-dependencies* and *intra-dependencies*. An inter-dependence is the basic dependence relation that exists between the requester of a service and the receiver of that request. Fig. 3 illustrates a set of inter-dependencies in a hypothetical system, where the arrows indicate the directionality of the dependencies, from sources to targets. For instance, service S3 is directly dependent upon services S9, S11 and S20, and indirectly dependent upon services S10, S12, S15, S21, S22, S23, and S25. Each inter-dependence

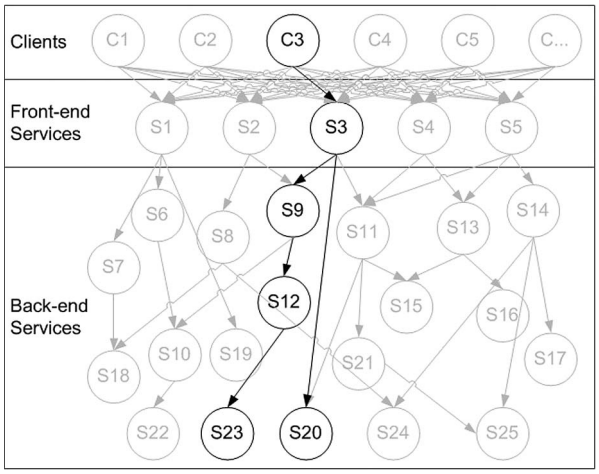


Fig. 3. Inter-dependencies among a set of services. The conversation originating at client C3 is highlighted.

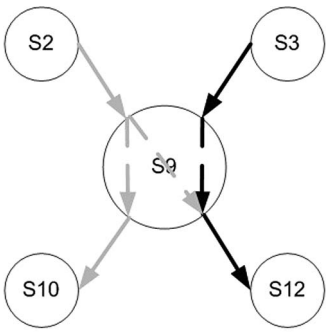


Fig. 4. Intra-dependencies of service S9 in Fig. 3.

in which a particular service is engaged can be classified as either *incoming* or *outgoing*. Service S9 has incoming inter-dependencies with S2 and S3, and outgoing inter-dependencies with S10 and S12. The figure highlights the subset of services and dependencies involved in single conversation originating at client C3. In this conversation, service S3 uses services S9 and S20 to complete the request, while service S11 is presumably used by S3 in processing other conversations.

An intra-dependence is a more complex relation between services that relates an incoming inter-dependence to an outgoing inter-dependence. In this sense, intra-dependencies reflect more detailed insight into the nature of the dependencies between services than do the basic inter-dependencies. This is illustrated in Fig. 4, which shows the dependencies among S2, S3, S9, S10, and S12 resulting from the four given inter-dependencies (solid arrows) and the three given intra-dependencies (dashed arrows). It is instructive to compare the information gained from Fig. 4 to that available in Fig. 3. We can see that S2 is (indirectly) dependent upon S10 and S12, while S3 is only (indirectly) dependent upon S12. This is not evident from Fig. 3.

B. Discovering Dependencies

As mentioned above, dependencies arise from the flow of messages among services. To discover dependencies, we must therefore track these flows. Because our aim is to be minimally intrusive, we restrict ourselves to observing the message traffic (i.e., messages that contain service requests and responses) as it occurs. Our method makes use of *monitors* deployed within the network to observe messages and record information about the flows. A convenient place to deploy a monitor is within a service's container. The monitor is then easily aware of the associated service's identity, as well as being provided a context in which to execute.

The main advantage of an approach based on monitors is that it allows the discovery of dependencies instantaneously and precisely, with minimal delays between dependence occurrence, detection, and the availability of the dependence information. Moreover, we can do so without having to modify the services themselves. Monitors can also minimize data storage and communication requirements, since they can aggregate the data. Thus, our approach can be thought of as a process for collecting *evidence* of dependencies, which is in sharp contrast to methods that require storage and transfer of large amounts of data for later statistical analysis.

Inter-Dependence Discovery: The source and target service pairs that induce inter-dependencies can be identified from the flow of messages exchanged between the services. In fact, an individual monitor needs to witness only *outgoing* inter-dependencies, since the inter-dependencies of a system can be reconstructed by integrating over the sets of outgoing inter-dependencies recorded at individual monitors. This assumes, of course, that services are uniquely identified and, universally, this is the case. For instance, the Web Services framework uses URIs for this purpose.

In practice, a monitor is aware of the identity of the source service with which it shares a container, so it can record the outgoing inter-dependence simply by extracting the identifier of the target service from any outgoing request messages originating at the service. The target service identifier is an essential field present in all request messages, such as plain HTTP or SOAP requests. Outgoing inter-dependencies are therefore easily discoverable in all existing service invocation protocols such as SOAP, REST or even from plain HTTP requests, without requiring any modifications to existing systems.

Intra-Dependence Discovery: Conceptually, an intra-dependence is a correlation at a given service between an incoming and an outgoing inter-dependence. We already saw how outgoing inter-dependencies are recorded. Incoming inter-dependencies are a bit more involved, as they require request messages to contain the unique identifier of the requesting service. This is satisfied by most service standards (e.g., the WS-Addressing standard provides the fields `wsa:To` and `wsa:From`). Alternatively, when a field containing the identifier of the requesting service is not present in the request message, we must make the stronger assumption that we can modify the message in flight to insert the additional header field into a request message at the requesting service. In practice, this should rarely be necessary.

Notice, however, that simply having the incoming and outgoing inter-dependencies *between services* is not sufficient to correlate the specific incoming and outgoing inter-dependencies *within a service* that give rise to intra-dependencies. For this purpose we rely on the presence of *conversation identifiers*

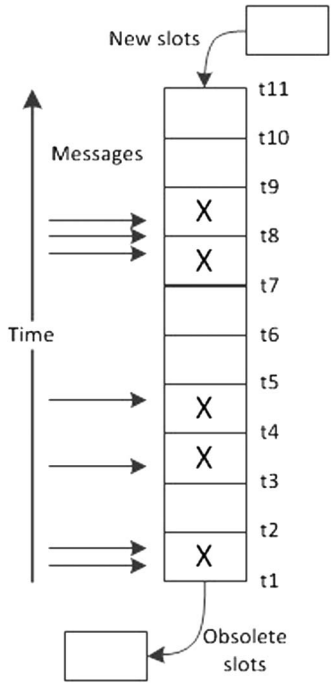


Fig. 5. Bit flag settings in the time slots maintained by a monitor.

within messages. In service-based systems, a conversation is the set of all messages exchanged during the processing of a single client request. Typically, the conversation is identified across the messages using tags inserted into message headers. Of course, since the notion of "conversation" is an application-specific semantic concept, conversation identifiers must be implemented explicitly by service programmers. Fortunately, this is a relatively routine task, as there are several existing standards for doing so, including WS-Addressing,[1] WS-SecureConversation,[2] and WS-Coordination,[3] that are increasingly used in current-day applications. Discovering an intra-dependence then reduces to having a monitor relate incoming and outgoing messages using the conversation identifiers appearing in both.

C. Storage of Dependence Data

Extracting the source and target fields of messages is a simple read-only operation causing a minimal computational overhead. Of greater concern is the amount of required data storage. Under our method, a monitor will only record the history of the dependencies it has seen for some bounded period of time. In particular, the dependence data are stored with a sliding expiration window such that only a limited history is maintained, using a fixed-size data structure to represent sliding *time slots*, as illustrated in Fig. 5.

Entries for each time slot maintain Boolean data about whether or not a given dependence occurred within that time slot. Each dependence is associated with a set of those time slots, such that when the monitor detects the occurrence of a dependence, it signifies this by setting a 1-bit flag in the corresponding data structure. It also records identifying information about the source and target of the dependence. Of course, the size of the time slot affects the precision of the data maintained. For example, a time slot of 0.1 seconds will provide up to 10 times more data per second as compared to a time slot of 1 second, but will require 10 times more space to store those data. Hence, a time slot of 0.1 seconds provides a finer granularity of information than a time slot of 1 second; there are 10 times more data entries created within the same period of time.

Beyond the size of the time slot, the size of the whole history can be controlled through the pruning of expired time slots. Notice that multiple messages reflecting the same dependence occurring within the time slot (e.g., `t1`) result in only a single aggregate record of the dependence.

The length that each time slot represents can be configured according to a desired level of resolution. When combined with the time period and the size of each dependence entry, the data storage needs for each monitor can be calculated. We estimate the required bytes of storage space S per monitor as:

$$S = \frac{T_h}{T_s} \times \frac{(N_{inter} + N_{intra})}{8} + (B_{inter} N_{inter} + B_{intra} N_{intra})$$

where T_h is the length in seconds of the time period, T_s is the length in seconds of a time slot, N_{inter} and N_{intra} are the average number of inter-dependencies (both incoming and outgoing) and intra-dependencies recorded by a monitor during the time period, and B_{inter} and B_{intra} are the maximum sizes of the identifiers of inter- and intra-dependencies. We use this equation to evaluate the data storage needs of our experimental scenarios in Section VI.

The aggregation of observed service interactions can cause the monitors to "forget" some of the details necessary to reconstruct fully accurate dependence information. This is the case when internal behaviors of a service that are not visible to the monitor can generate seemingly identical interactions in aggregate. As an example, consider the incoming inter-dependence of S2 on S9, and the outgoing inter-dependencies on S10 and S12, shown in Fig. 4. The incoming inter-dependence could in fact be the result of aggregating two separate conversations between S2 and S9, where the first resulted in a message to S10 and the second resulted in a message to S12. The aggregate dependence information held by the monitor will not, depending on the size of the time slot, record the true dependencies regarding these individual conversations.

D. Dependence Graph

The integration of the data recorded by individual monitors results in a *dependence graph*. A dependence graph (DG) is a directed acyclic graph constructed from a set of nodes representing services and a set of edges representing direct inter-dependencies.[4] The direction of an edge represents the direction

[1]http://www.w3.org/2002/ws/addr/
[2]http://www.ibm.com/developerworks/library/specification/ws-secon/
[3]http://www.ibm.com/developerworks/library/specification/ws-tx/

[4]The acyclic structure of the DG is a limitation of the current method. In practice, however, cyclic dependencies of requests *occurring within a single conversation* are rarely to be expected and likely result from poor design.

of the inter-dependence, from source to target. Each node can be annotated with intra-dependence information, conceptually adding directed edges between the incoming and outgoing inter-dependencies of the service.

The DG maintains information concerning a specific *time window*, reflecting only the dependence information collected by (or perhaps available from) monitors during that period. The time window is a property of the interaction between the application behavior, the network behavior, and the information accessible to monitors. The size of the time window has many effects on the results of analysis. For example, a small time window serves to reduce the size of the DG, but some critical service interactions might be missed. A large time window provides a more complete record of dependencies, but might include stale or irrelevant interactions (e.g., those belonging to conversations other than the target conversation for the analysis).

Conceptually, a DG could be used to represent the full set of dependencies of an entire application system. In practice, many analysis techniques only require a subgraph of the full dependence graph related to a specific node or subset of nodes. For example, a failure impact analysis might examine only the nodes that can reach (i.e., are dependent upon) a given node, and a fault localization analysis might examine only the nodes that are reachable from a given node.

E. Dependence Graph Construction

Our method uses a set of distributed monitors that provide information to a dependence discovery element, as illustrated in Fig. 2. The intent of this architecture is to minimize resource utilization, while still providing timely data. The monitors perform continuous dependence discovery and maintain aggregate dependence data. The dependence discovery element itself can be hosted on any node of the network. In fact, since the dependence discovery element is a small software component, it can be hosted on multiple nodes of the network. For example, every node that hosts some client application is a good candidate for also hosting a dependence discovery element.

The discovery element will construct a DG on demand, querying the relevant monitors to harvest their local dependence data for the time window of interest. The harvesting algorithm is designed to incrementally construct the DG—typically a subgraph of the full application dependence graph—by visiting only the monitors considered relevant based on the data seen to that point. In this way, the amount of data transmitted over the network can be significantly reduced compared to existing methods. Of course, the most common case of DG construction is for a particular client, revealing the services upon which that client depends directly or indirectly.

Conceptually, the data are harvested by a straightforward walk rooted at the monitor associated with the client. The first round of the harvesting algorithm adds root dependencies to an empty DG. The root dependencies might be limited to a single root dependence, which is useful when discovering the DG for a particular conversation, such as for a fault localization analysis. Otherwise, the root dependencies include all of the root dependencies occurring within the time window. This is useful when discovering the set of all services involved in requests during a particular time period, such as for a composition analysis. In the following, recursive rounds, the algorithm issues harvesting requests based on the outgoing dependencies of newly discovered dependent services so that their dependencies can be added to the DG.

```
1  public DepGraph constructDG (String rootNodeURI,
2              long fromTimestamp, long toTimestamp)
3  {
4    DepGraph dg = new DepGraph(rootNodeURI);
5    int nodeIndex = 0;
6    Node node;
7    String[] nodeDeps;
8    while ((node = dg.getNodes()[nodeIndex]) != null) {
9      nodeDeps = getDepsForNode(node,
10                       fromTimestamp, toTimestamp);
11     for (String nodeDep : nodeDeps) {
12       dg.addDep(node.getId(), nodeDep);
13     }
14     nodeIndex++;
15   }
16   return dg;
17 }
```

Listing 1. Java implementation of a harvesting algorithm.

Listing 1 demonstrates how the harvesting algorithm could be implemented as a simple, sequential, breadth-first process in Java. constructDG has as input parameters the identifier of a root node (i.e., client) and timestamps for the beginning and end of the relevant time window. It uses a helper function, getDepsForNode, to query a monitor for its outgoing dependencies. constructDG initially creates an object to represent the DG, which has a single root node to represent the client.

For each node in the DG, getDepsForNode queries the corresponding monitor to retrieve any recorded dependencies. For every dependence added, a new node representing the target service is created, if does not already exist. Obviously, more sophisticated implementations are possible, such as one that harvests the data from multiple monitors in parallel.

The accuracy of DG construction mostly depends on the time window and workload. With a short time window and a small workload (i.e., infrequent interactions among services), we may not observe some critical dependencies. With a long time window and a large workload, we may include obsolete and irrelevant interactions (e.g., dependencies belonging to conversations that are not of interest). In Section VI, we investigate the impact of these and other factors on the accuracy of the DG obtained by our method. We then suggest ways to select parameter values to optimize the discovery process.

IV. An Example Application

Dependence information forms the basis for a variety of important system-level analyses. As an example, we have experience in using the dependence information obtained from the discovery method presented here to develop a new *fault localization* aid for the operators of service-based systems deployed on MANETs [17]. Describing this experience in detail is beyond the scope of the present paper. However, for purposes of illustration we briefly sketch its basic design.

Fault localization (sometimes also referred to as *fault identification*) in general refers to a technique for identifying the

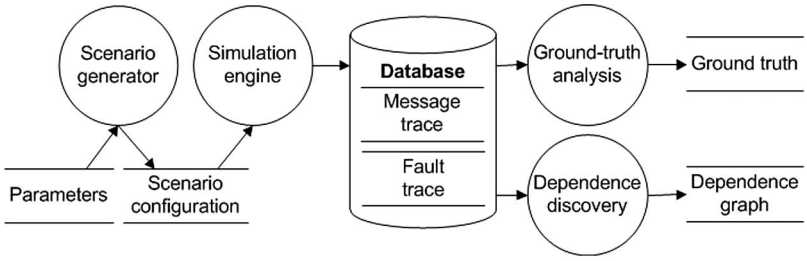


Fig. 6. Experimental framework.

likely root causes of failures observed in systems formed from components. Fault localization in systems deployed on MANETs is a particularly challenging task because these systems are subject to a wider variety and higher incidence of faults than those deployed in fixed networks, the resources available to track faults are severely limited, and many of the sources of faults 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 even manifest themselves as failures at the service level, since some of those faults may occur outside of the time during which the relevant services are communicating, or involve communication outside of the relevant conversation. Moreover, service-based systems are typically designed to tolerate certain kinds of faults through mechanisms for dynamic service (re)binding. Therefore, fault localization must be especially adept at sifting through noisy data.

Our technique uses service-level dependence relationships as the basis of the analysis. We envision being given a failure report in terms of a failed client conversation. Then, as described in the previous section, the data needed to construct the dependence graph are collected by using the client as the root of a walk. The technique also makes use of *fault symptoms* recorded in logs, as well as a *fault propagation pattern* built from a classification of the failures that can be experienced by a client. The pattern indicates how the symptoms can be propagated through the system from root causes to clients. The specific pattern associated with a given failed conversation is then mapped onto the discovered dependence graph corresponding to the conversation, resulting in a *fault propagation model* (FPM) that represents how specific faults propagate through the individual services involved in the conversation. The FPM is combined with the occurrence times of the relevant symptoms. This allows the technique to reconstruct the possible fault propagation paths, both in time and in space, and thus to form a set of plausible hypotheses about the causes of the failure. The hypotheses are represented as a *causality graph*.

After the causality graph is constructed, a ranking algorithm is applied to the faults in the hypothesis set based on their likelihood of being the root cause of the failure. We have devised two alternative approaches to ranking. The first is a simple timing-based approach that ranks hypotheses based on the time difference between possible root causes and the failure. The second is a probabilistic method that uses a Bayesian network to infer independent probabilities of individual root-cause hypotheses. We obtain the Bayesian network by combining the FPM with fault propagation rules that represent conditional probability distribution parameters. The algorithm infers the posterior probability of a hypothesis being a root cause given the evidence of the failure.

Our results show that the timing-based ranking is very effective in localizing the failures caused by explicit service exceptions, while the Bayesian-based ranking is more effective in the case of implicit timeout failures. The accuracy of the discovered dependence graph plays a major role in determining the quality of the outcome from either ranking. However, this effect is particularly significant in the presence of a high ratio of false positives in the discovered dependencies when attempting to locate the root of exception faults [17].

V. Experimental Setup

The evaluation of our dependence discovery method is based on a simulation framework for service-based systems operated in MANETs. As shown in Fig. 6, the framework 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 is prototyped as a query over this database. In addition, we derive the "ground truth" used in Section VI to evaluate our method.

Without loss of generality, we make the assumption that faults occur only at the network level, and that the services themselves are error free. From an observational point of view, any non-Byzantine service-level fault can be simulated by a corresponding network-level fault (e.g., whether the absence of a response message is due to a service fault or a network fault is indistinguishable to a service waiting for that response). Therefore, this assumption does not materially impact our results, but does allows us to concentrate on simulating the effect of the network on service interactions.

A. Simulation Engine

The simulation engine is built on top of the discrete-event network simulator NS-3,[5] which provides a comprehensive network simulation using detailed implementations of low-level network protocols. However, the highest abstraction provided by NS-3 is that of sockets and network packets. We therefore extend NS-3 with higher-level abstractions for simulating service entities and their interactions. We do this by first encapsulating the socket layer into a *messaging layer* that provides

[5]http://www.nsnam.org/

the abstraction of service-level messages. The messaging layer itself is then encapsulated into a *service layer* that provides abstractions for services, clients, and their service-oriented interactions. Finally, we provide a means to configure and run the extended simulation.

B. Simulation Models

Service-level scenarios are created using several basic models and associated parameters.

Entity Models: These models provide the building blocks of service-based systems: clients and services. Clients behave autonomously, making requests on specific services at configurable times (see the workload model, below). Services provide methods that can be requested by clients or other services. Each method contains a specification of its behavior, consisting of delays to simulate computations, and a set of steps for making further requests on other services.

Deployment Model: The deployment model specifies where in the network the clients and services are hosted.

Interconnection Model: The interconnection model defines the (dynamic) binding of clients to the methods of services, and of each method to the methods of other services.

Message Model: Three types of messages are exchanged between entities: requests, responses, and exceptions. Requests are used to invoke methods in other services, while responses are sent by services back to the requester upon completion of the requested processing. Exceptions are used to propagate fault symptoms caused by failures. As described in Section III, the flow of messages related to an original client request is called a *conversation*. In the simulation, all messages contain a conversation identifier, possibly available to the dependence discovery element (according to what we assume about the discovery process), but always available to establish the ground truth for dependence relationships.

Workload Model: The workload is determined by the rates at which clients send requests to services. Currently, our simulator implements workloads in which clients, at uniformly random times, request a method chosen uniformly at random from its set of currently bound methods.

Fault Propagation Model: Two mechanisms are used to propagate a fault from its root cause back to the client that initiated the failed conversation: exceptions and timeouts. Exceptions are explicit messages sent by services to their requesters to notify them of a failure to fulfil a request. Typically, they engender a cascade of such messages. Timeouts are implicit indications of faults detected by the requesters upon the lack of a response from services. Timeouts engender an implicit propagation cascade.

The simulation scenarios are constructed by a configuration generator. NS-3 defines the available network parameters, while the models above provide the service parameters.

VI. Evaluation

In our dependence discovery method, DGs are constructed on demand by a discovery element. The graphs are rooted at a given client, beginning at a given time instant, and for some time window. Therefore, the data provided to the discovery element will include both relevant and irrelevant information, since a monitor will provide data about *all* interactions traversing its associated service during the time window and, thus, involve not just the interactions of the given conversation of interest, but also those of others. Under such circumstances it would be difficult for any dependence discovery method to provide a perfect result. Moreover, our method by design loses information (e.g., monitors retain only aggregate data, not individual messages).

Thus, the evaluation questions of interest center on the accuracy of the resulting DG. In particular, we examine the following factors that should influence accuracy in this setting.

Time Window Size: The size of the time window is the main parameter of the method. With a short time window we may not observe some critical dependencies and with a long time window we may include obsolete and irrelevant dependencies. We measure the impact of time window size as the ratios of false positive and true positive dependencies (see Section VI-A).

Degree of Service Connectivity: The degree of service connectivity 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 data.

Dynamics of Service Interconnection: Service dependencies change (i.e., adapt) in response to the dynamics of the network. A higher rate of change in service dependencies should lead to a greater number of irrelevant dependencies included in the discovered DG.

Client Workload Rate: We examine the sensitivity of the method to a range of rates at which clients issue service requests. We would expect that as the rate increases, the higher the overlap among conversations, and the more noise in the dependence data.

Mobility Speed: We also examine the sensitivity of the method to a range of node mobility speeds. We would expect that as nodes move faster, the method will have increasing difficulty maintaining consistent results.

Notice that the degree of service connectivity, the dynamics of service interconnections, and the client workload rate are application properties, the mobility speed is an operating environment property, and the time window size is a tuning parameter for the method.

In addition to these basic factors, we evaluate three other aspects of the method.

Inter vs. Intra Dependence: We would like to understand whether supporting intra-dependence discovery provides improved accuracy when compared to using only inter-dependence discovery, since the former makes greater assumptions about the service implementation than the latter.

Data Storage and Transfer: Given the limited resources available in mobile devices, we evaluate the data storage and data transfer requirements of our method.

Comparison With Existing Methods: Performing a comparative analysis is difficult, since we do not have access to implementations of other methods. Nevertheless, we make an effort to simulate their designs so as to produce a basic, coarse-grained comparison in terms of true and false positives.

TABLE I
NETWORK-LAYER PARAMETERS

Number of nodes	50
Spatial bounds	75m x 75m
Mobility speed	10 m/s
Mobility model	RandomDirection2dMobility
Propagation delay model	ConstantSpeedPropagationDelay
Propagation loss model	LogDistancePropagationLoss/α3
WiFi standard	80211b
WiFi rate	11Mbps
Routing protocol	OLSR and Ipv4StaticRouting
Protocol stack	UDP/IPv4

TABLE II
SERVICE-LAYER PARAMETERS

Number of clients	50 (one per node)
Number of services	30
Invokable methods per service	2
Workload (client request rate)	10 – 80 sec
Service dependence switch rate	5 – 15 min
Message size	500 – 1500 bytes
Response timeout	60 sec
Inter-service connectivity probability (Low)	0.0125
Inter-service connectivity probability (Medium)	0.025
Inter-service connectivity probability (High)	0.05

TABLE III
DEPENDENCIES INDUCED BY CONNECTIVITY

	Low	Medium	High
Client-to-service dependencies	250	250	250
Service-to-service dependencies	30	59	130
Dependencies in graph (avg.)	2.03	3.21	7.9
Dependencies in graph (std. dev.)	1.2	2.28	3.63

A. Evaluation Metrics and Parameters

Our experiments focus on a particular hypothetical conversation C. A good result for our method would be that it can discover as many dependencies of C as possible, while not including the dependencies of other conversations. We use two metrics to characterize the quality of our results, namely the ratio of true positives (TP) and the ratio of false positives (FP), defined as follows:

$$TP\ ratio = \frac{|D(C) \bigcap GT(C)|}{|GT(C)|}$$

$$FP\ ratio = \frac{|D(C) - GT(C)|}{|D(C)|}$$

where $D(C)$ is the set of discovered dependencies, and $GT(C)$ is the set of ground-truth dependencies. The true positives are in the intersection of these two sets, and false positives are in the set difference. The TP ratio represents the fraction of discovered dependencies as compared to the actual dependencies in the conversation, and is therefore equivalent to the *recall* metric of information retrieval (taking "ground truth" as "relevance"). The FP ratio represents the fraction of discovered dependencies not belonging to C, and is therefore the complement of *precision*. We assume $D(C)$ and $GT(C)$ are non-empty.

A high TP ratio indicates that most of the dependencies in C have been discovered. A high FP ratio indicates that the discovery result mistakenly includes a large number of dependencies of conversations other than C. These irrelevant dependencies need to be minimized to improve the accuracy of the DG.

Table I summarizes the basic network-layer parameters used in our simulations. These are standard settings used widely in the networking community and embodied in the NS-3 simulator. Specifically, we use the log distance model with path-loss exponent 3 for wireless signal propagation, reproducing a network operated in an urban area [13]. We set the spatial mobility bounds to a 75 meter square, which is a limitation imposed by the chosen WiFi standard, as larger regions induce long-term network partitioning. Another important parameter is the mobility speed of the mobile hosts. For most of the experiments we set the mobility speed of all nodes to 10 m/s. This is a challenging scenario that allows us to simulate environments in which message exchanges between the nodes are materially affected by the temporary disruptions in communication at the network layer. In Section VI-E we report results in which we vary this parameter from 0 m/s (i.e., a "stationary" wireless network) to 15 m/s.

Table II summarizes the basic service-level parameters used in our simulations. The message sizes and timeouts are derived from standard values found in SOA and Web Services implementations. Note that the number of methods in each service is not significant from a simulation point of view, as long as we have at least two methods available so that we can examine the impact of intra-dependence information on dependence discovery.

For the interconnection model, we use a 2-tiered topology. The first tier consists of the connections between the clients and a set of "front end" services, while the second tier consists of the connections between the services themselves. In our simulations, we use 50 clients, five front-end services, and 25 "back end" services. When starting a conversation, each client invokes a method selected uniformly at random from all methods provided by the five front-end services. We experimented with other topologies, including a single-tier topology in which there are no designated front-end services, and with different numbers of clients and services, but found that the results were consistent. We therefore only report results based on the 2-tier topology.

Of particular importance is the degree of connectivity among the services. We configure the simulations to capture three different connectivity scenarios, denoted as "Low", "Medium", and "High". The connectivity degree is induced by the probability that a method in one service invokes a method in another service; the higher the probability, the denser the interconnection topology.

The effect of different connectivities on the resulting ground-truth dependencies for each scenario is shown in Table III. Because the 50 clients will always invoke the five front-end services over the course of an experimental run, there will be 250 client-to-service dependencies in all scenarios. However, the total number of unique service-to-service dependencies increases with the connectivity probability, from 30 dependencies for low connectivity to 130 dependencies for high connectivity. Also increasing with the connectivity probability is the resulting number of dependencies per conversation (i.e., the number of arcs appearing in the ground-truth DG rooted at a given client). The average and standard deviations for the

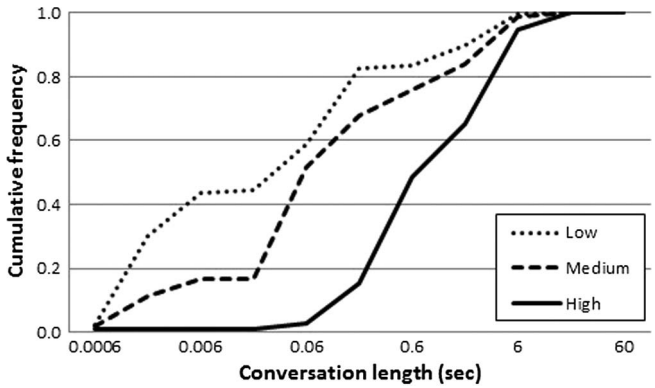


Fig. 7. Cumulative distribution of conversation lengths (based on request messages) for three connectivity scenarios.

conversations occurring in the three scenarios are given in the table. Notice that a high standard deviation in this case indicates a good mix of different kinds of conversations.

An artifact of having more services involved in a conversation due to a higher degree of connectivity is that it increases the time it takes to complete the conversation. This is borne out by Fig. 7, which gives the cumulative distribution of the conversation lengths in our simulations for the three connectivity scenarios. For example, 80% of the conversations in the low connectivity scenario are completed in under 0.2 seconds, whereas the top 20% of the conversations in the high connectivity scenario take longer than 4 seconds.

Service dependencies may change not only due to normal computational progress, but also to optimize the use of otherwise equivalent services. In this way the service system adapts to the network dynamics, such as a large fluctuation in communication quality and availability. To see the impact of this form of dynamic service rebinding, we introduce into our experiments a *service-switching* behavior. Every service is given two designated alternative services, and every dependent service then switches among these three services after a randomly selected time period. In Section VI-C we report results in which we vary the switching-time parameter in increments from 10 seconds on average (representing an extremely dynamic behavior) to infinite (representing a non-switching behavior).

We collect our results from 30 minutes of simulated execution time after excluding 30 seconds of warm up. Each combination of parameters in our experiments results in thousands of conversations occurring during the simulated 30-minute execution. For instance, the low, medium, and high connectivities combined with 10-minute switching-time periods and 10 m/s mobility speeds result in 8440, 8139, and 7165 conversations, respectively. The results given below are averages over the data collected from these conversations, where each conversation is then a statistical sample subject to the random variables.

B. Impact of Time Window Size and Service Connectivity

We first look at the impact of the time window size and connectivity degree on the accuracy of the results. We hypothesize that as the time window size grows, so too should the TP ratio,

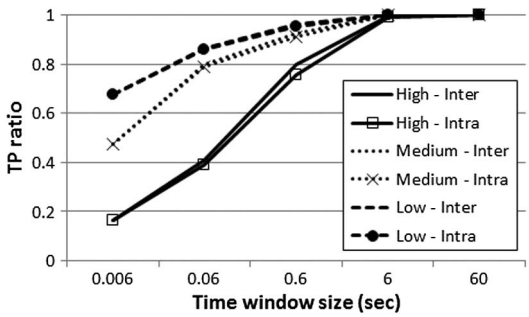


Fig. 8. Accuracy, given as TP ratio, for different time window sizes.

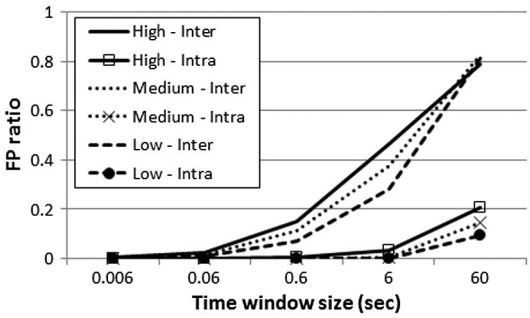


Fig. 9. Accuracy, given as FP ratio, for different time window sizes.

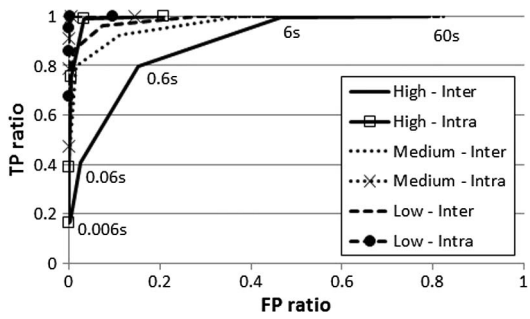


Fig. 10. Trade off between TP and FP ratios for different time window sizes.

since more dependencies will be captured. However, increasing the time window size should also increase the FP ratio, since there is a greater chance that messages belonging to other conversations are included in the DG. For a given time window size, we expect the TP ratio to be negatively correlated with the connectivity degree, since a higher connectivity increases the conversation length, which in turn increases the chances that some dependencies are missed. Similarly, we would expect the FP ratio to be higher in densely connected service configurations, since dependencies in other conversations are more likely to overlap those of the subject conversation.

We calculate the TP and FP ratios for both inter- and intra-dependence discovery separately. Figs. 8, 9, and 10 depict the results, where each data point is the ratio averaged over all conversations. Here we consider only fixed service bindings (i.e., non-switching behavior). The variances of the TP and FP ratios are small, and therefore omitted from the figures. For example, the largest 95-percentile confidence intervals for TP and FP ratios in the medium connectivity scenario are 0.006 and 0.0053, respectively.

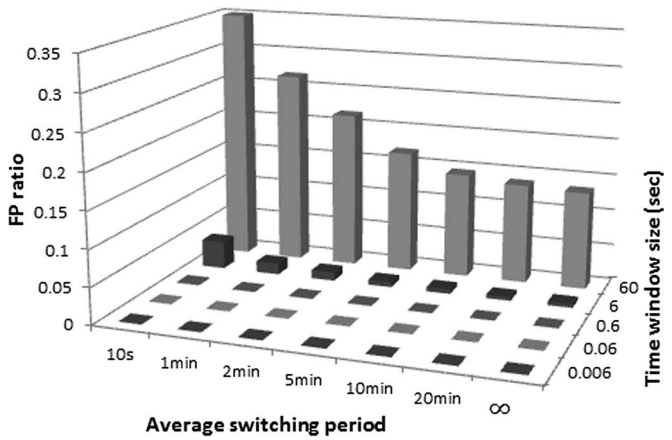


Fig. 11. FP ratio for intra-dependence discovery under medium connectivity for various switching-time periods and time window sizes.

As shown in Fig. 8, increasing the time window size increases the TP ratio, both for inter- and intra-dependencies. A larger window will include more messages and thus discover more dependencies. However, increasing the time window size also increases the FP ratio, as shown in Fig. 9. A larger window will include more messages from other conversations. The same figures also confirm our hypotheses about the impact of the connectivity degree: the TP ratio decreases and the FP ratio increases as the service topology becomes denser. Notice, too, that intra-dependence discovery has a significantly lower FP ratio than inter-dependence discovery. This is due to the fact that it can precisely correlate incoming and outgoing inter-dependencies, something to which inter-dependence discovery is blind (recall Fig. 4). To directly display the trade off between the TP and FP ratios under various time window sizes, we plot them against each other in Fig. 10, where each point represents the given time window size.

C. Impact of Dynamic Service Rebinding

We now investigate the sensitivity of the method to dynamic service rebinding (i.e. the dynamics of service interconnections). For a given time window size, we would expect a high switching rate to cause many dependencies belonging to irrelevant conversations to be included in the discovered DG, thereby increasing the FP ratio. In contrast, we would expect the TP ratio to be insensitive to the switching rate because once a message is seen, any additional messages, relevant or irrelevant, should not increase that ratio.

We ran experiments using seven different average switching-time periods, from 10 seconds up to a case in which no switching occurs; a lower value results in a faster switching rate. Fig. 11 shows the intra-dependence FP ratio in the medium connectivity scenario. (As hypothesized, the TP ratio is essentially unaffected by the switching rate, so we do not show that result here.) The effect on FP ratio is most noticeable when the switching-time period is on the same order as the time window size, which indicates possibly many service switches happen within the time window, rendering the resulting dependencies irrelevant from the discovery element's point of view.

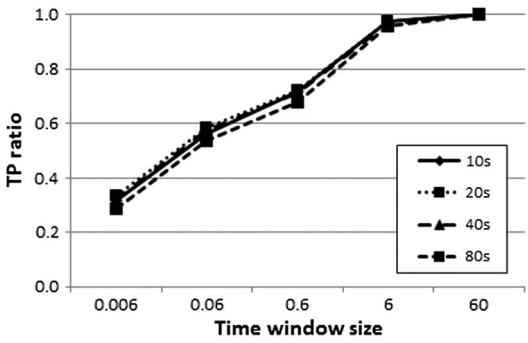


Fig. 12. TP ratios for inter-dependence discovery under four workloads.

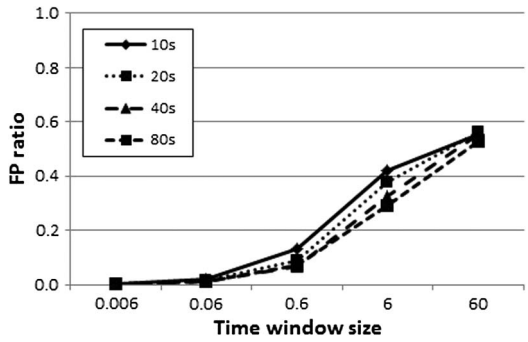


Fig. 13. FP ratios for inter-dependence discovery under four workloads.

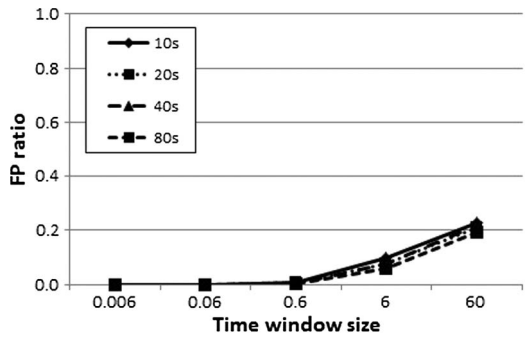


Fig. 14. FP ratios for intra-dependence discovery under four workloads.

D. Impact of Client Workload

We define the workload to be the rate at which clients issue service requests. We hypothesize that the workload has a positive correlation with the FP ratio, since a high workload will generate more messages that are part of irrelevant conversations. In contrast, we expect the TP ratio to be insensitive to the workload. This is because the TP ratio is related to the fact of messages being exchanged between services, rather than to the volume of those messages. In other words, once a message is seen to have been exchanged between two services, any additional messages, relevant or irrelevant, should not increase the TP ratio.

Using the high dependence scenario, we experiment with four different workload rates: 10 s, 20 s, 40 s, and 80 s. Each rate represents the average of a uniformly random waiting time from the completion of a given service request by a client to the issuance of the next request by that client. A lower value therefore indicates a higher workload.

Figs. 12, 13, and 14 present our results. Overall, the effect of the workload appears to be minor, with only slight differences in TP and FP ratios evident as the length of the time window increases. This is what we expected to see for the TP ratio, but the very weak correlation for the FP ratio is a somewhat surprising result. We hypothesize that this is an artifact of the particular service configuration used in the experiments.

E. Impact of Mobility Speed

The next set of experiments investigate the impact of node mobility speed on the inter-dependence discovery TP and FP ratios. We work with four different mobility speeds: 0 m/s (amounting to a fixed network), 5 m/s, 10 m/s, and 15 m/s.

We would expect that as the speed of the mobile nodes increases, the quality of the links between them should deteriorate and, consequently, the failure rate of message exchanges should increase. Higher message failure rates should in turn increase conversation lengths, as more messages must be resent in an attempt to complete the conversations. Therefore, increasing the mobility speed should have the effect of decreasing the TP ratio, since messages require longer time periods to be exchanged and detected. This, however, should not impact the FP ratio, since the proportions of relevant and irrelevant messages remain unaffected.

The results are reported in Figs. 15, 16, and 17. As expected, the TP ratio generally has a positive correlation with mobility speed, with the fixed network (0 m/s) exhibiting the highest ratio. Furthermore, we observe that the lengths of the conversations increase with mobility speed (and network failure rate), which is visible in the TP ratios between the 0.006 s to 0.6 s time windows. However, for longer conversations, the time windows of 6 s and 60 s are long enough to capture all cases. The FP ratios are virtually unaffected. This coincides with our understanding that the ratio of relevant and irrelevant messages remains the same.

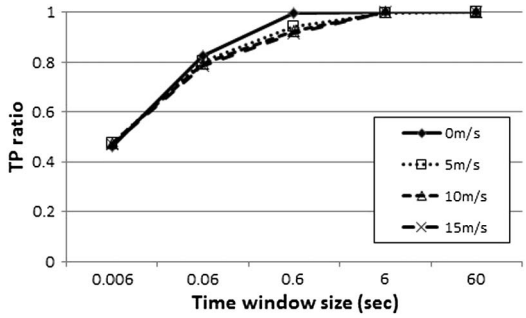


Fig. 15. TP ratios for inter-dependence discovery under four mobility speeds.

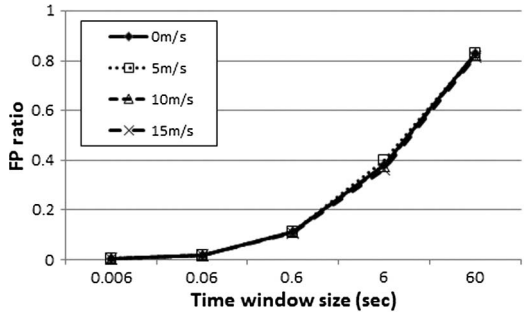


Fig. 16. FP ratios for inter-dependence discovery under four mobility speeds.

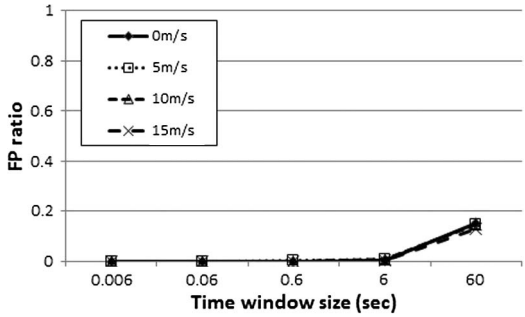


Fig. 17. FP ratios for intra-dependence discovery under four mobility speeds.

F. Comparison With Existing Methods

We now compare the accuracy of our dependence discovery method to that of existing methods. We make this comparison by implementing two alternative methods to represent the two major classes of existing approaches: those that perform discovery at the service level [6], [8] and those that perform discovery at the network level [2]–[5], [11], [14]. These implementations are, like our own method, prototyped as queries over the trace database (see Section V). The service-level alternative discovers a global system dependence graph by observing all the service messages exchanged over the whole execution period and, from this, builds DGs for the individual client conversations. The network-level alternative works similarly, but only observes the flow of messages by inspecting the information contained in the headers of packets exchanged over the relevant IP ports. It then builds DGs using external information provided to it about the deployment of clients and services on hosts.

We compare our method against the two alternatives using the medium connectivity scenario, a 10 minute service-switching time period, a 10 m/s mobility speed, and workload rate of 10 s. The average number of ground-truth dependencies is 3.21 (see Table III). We configure our method to use a 60 s time window size, which is large enough to capture all such dependencies (see Fig. 7). The comparison then reduces to one based on the absolute number of *false dependencies* appearing in the discovered DGs.

As we discuss in Section II, the existing service- and network-level methods are designed for use in fixed networks and for relatively stable service configurations. Therefore, since both these alternative methods build DGs from long-term observations, we do not expect them to adequately filter out stale dependencies caused by the dynamics of the scenarios, resulting in higher false positives than with our discovery method. Moreover, the network-level method should include even more false positives than the service-level method because it builds DGs from coarser-grained information.

The results are reported in Fig. 18, where a vertical line is used to separate the results for our method on the left from the results for the alternative methods on the right. We give the FP ratio, as well as a count of the false dependencies appearing

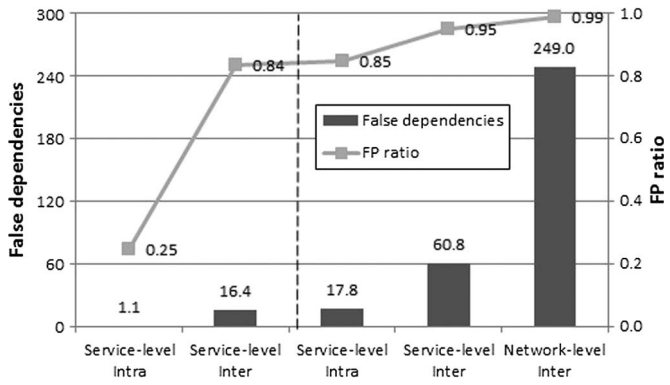


Fig. 18. Comparison of methods in medium connectivity scenario. Results for existing methods are to right of dashed line.

in the DG. Both are computed as the average over the total number of conversations (8139) occurring in the 30-minute execution period. As expected, our method provides DGs having a significantly lower FP ratio than the alternative methods. Furthermore, although the FP ratio for our inter-dependence discovery is similar to that of the alternative methods, the actual number of false dependencies is significantly lower. We note that experiments using various other workloads produced similar comparison results.

G. Estimates of Data Storage and Transfer Needs

We conclude our evaluation of the dependence discovery method by investigating the data storage requirements placed on the monitors, and the data transfer requirements placed on the network. We do this by positing a representation for the data, analyzing the space requirements of that representation, and computing the total for a particular experimental scenario.

Consider the medium connectivity scenario, where the average number of inter- and intra-dependencies per service is 12 and 53, respectively. Assume a maximum size for identifiers of 256 single-byte characters, which corresponds to the Web Services framework's use of the maximum URL length. With a time slot length of 0.01 seconds (an especially precise configuration) maintained for a time period of 10 minutes, we calculate the required data storage (using the formula given in Section III-B) to be approximately 506 kilobytes per monitor.

The estimates above are in some sense worst case. In practice the identifiers will be smaller (certainly smaller than the maximum allowed URL length). Moreover, the storage used for intra-dependence identifiers can be significantly reduced, simply by maintaining them as references to the corresponding incoming and outgoing inter-dependencies.

While the monitors locally store data describing all dependencies detected during the time period, the data transferred over the network to the discovery element contains only the data for the desired time window. This time window is typically much shorter than the stored time period.

Consider again the medium connectivity scenario. The typical data bundle transferred by an individual monitor contains data for approximately one outgoing inter-dependence and one intra-dependence. This amounts to a maximum of 512 bytes of data per service. The average number of services involved in a client request is 3.21, so the discovery element will issue on average 4.21 requests (including the initial request to the client's monitor) for dependence data to construct the DG. This results in a total of approximately 1.6 kilobytes of data transferred over the network, which is substantially less than the total amount of dependence data stored by the monitors.

H. Discussion of Results

The experiments presented above establish how the time window size, degree of service connectivity, dynamics of service interconnection, client workload rate, and mobility speed affect the accuracy of our dependence discovery method. In addition to these results, we have also experimented with several other parameters, such as the rate of service faults. However, they do not seem to have a significant impact on the accuracy of the method, so are not reported here.

In general, the results for intra- and inter-dependence discovery indicate consistently equivalent accuracy as measured in terms of TP ratio. On the other hand, the FP ratio is significantly lower for intra-dependence discovery compared to that for inter-dependence discovery. Both the TP and FP ratios are significantly affected by the selection of the time window size: as the time window size increases, so do the TP and FP ratios. The TP ratio, however, tends to reach a maximum value at a certain time window size, whereas the FP ratio degrades (i.e., increases) unabated beyond that value. This implies that the time window can be selected such that it will yield a discovery result striking a good balance between true positives and false positives.

We can also see that the FP ratio is significantly affected by the service-switching time period. This factor depends on the service discovery and selection technique used in the system. In general, however, as TP ratio is relatively insensitive to the service interconnection dynamics, one can select the time window size after taking into account the service switching rate, which in a MANET is typically related to the dynamics of the network.

In contrast to time window size, the accuracy of the dependence results appears to be less sensitive to network mobility and workload factors, assuming the service interconnection dynamics are at a given level. This implies that our method is relatively robust to variations in operational conditions.

The comparison with existing methods validates our hypothesis that MANET-hosted service-based systems require a fundamentally different approach to dependence discovery. Particularly striking is the difference in FP ratio between our method's intra-dependence discovery and that of the other methods. While our method provides approximately one false dependence on average per DG, the existing approaches yield DGs significantly larger and constructed mostly from false dependencies. This is especially evident in the network-level approach, which yields DGs containing almost the entire set of services, rendering dependence discovery virtually useless.

Finally, our estimates for the method's storage requirements fall well within the capabilities of today's mobile devices. Moreover, the method's aggregation of data results in an extremely low requirement for data transport.

I. Threats to Experimental Validity

The results reported are a selection from the experiments we have conducted. This selection focuses on particular aspects of the method, as detailed above. The results withheld support the results reported.

The threats to the validity of the results derive from the prerequisites listed in Section I and the experimental parameter values detailed above. We have chosen these values based on the joint experiences of the NS-3 community of users, and on the recommendations of commercial SOA and Web Services providers. Where specific community experience was lacking in the choice of values (e.g., the absence of a benchmark), we attempted to broadly sample in the space of values. We made use of uniformly random distributions in several instances (e.g., the request behavior of clients), which are a simplification that is a methodologically accepted practice at this stage of investigation. This yields statistically sound results, as each data point in our results is an average over all arising conversations, each of which is a statistical sample subject to the random variables.

We make no claim that the results generalize beyond the reported experiments, but overall they give us confidence that the method is a viable approach to service dependence discovery in the challenging MANET environment.

VII. Conclusion

We have presented an on-demand method to discover the dependencies among services operated in the highly dynamic and resource-constrained environment of MANETs. Unlike existing approaches, the method does not require stable dependence relationships, nor does it require that large amounts of evidence data be collected over long periods. Through an extensive set of simulation-based experiments, we have evaluated the accuracy of the method in terms of operational factors characteristic of both service-based systems and MANETs. The method exhibits good behavior when subjected to the stress of a changing underlying network topology. Furthermore, its data storage and data transfer requirements scale well with the number and connectivity of the services involved.

Dependence information is not particularly useful in and of itself, but instead serves as an indispensable building block for important analysis capabilities. Based upon the dependence discovery method presented in this paper, we are currently developing such analyses, including those for probabilistic fault localization, as briefly sketched in Section IV, and cross-layer performance anomaly diagnosis [22].

Acknowledgment

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] F. Al Shahwan and K. Moessner, "Providing SOAP web services and RESTful web services from mobile hosts," in *Proc. 5th Int. Conf. Internet Web Appl. Serv.*, May 2010, pp. 174–179.

[2] P. Bahl *et al.*, "Discovering dependencies for network management," in *Proc. 5th Workshop Hot Topics Netw.*, Nov. 2006, pp. 97–102.

[3] P. Bahl *et al.*, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *Proc. Conf. Appl., Technol., Architect., Protocols Comput. Commun.*, Aug. 2007, pp. 13–24.

[4] P. Barham *et al.*, "Constellation: Atomated discovery of service and host dependencies in networked systems," Microsoft Res., Mountain View, CA, USA, Tech. Rep. MSR-TR-2008-67, 2008.

[5] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for request extraction and workload modelling," in *Proc. 6th USENIX Symp. Oper. Syst. Des. Implementation*, 2004, pp. 259–272.

[6] S. Basu, F. Casati, and F. Daniel, "Toward web service dependency discovery for SOA management," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2008, pp. 422–429.

[7] C. Chen, A. Zaidman, and H.-G. Gross, "A framework-based runtime monitoring approach for service-oriented software systems," in *Proc. Int. Workshop Qual. Assurance Serv.-Based Appl.*, 2011, pp. 17–20.

[8] M. Y. Chen *et al.*, "Path-based faliure and evolution management," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2004, pp. 1–14.

[9] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "Automating network application dependency discovery: Experiences, limitations, new solutions," in *Proc. 8th USENIX Symp. Oper. Syst. Des. Implementation*, 2008, pp. 117–130.

[10] P. Choudhury, A. Sarkar, and N. Debnath, "Deployment of service oriented architecture in MANET: A research roadmap," in *Proc. 9th IEEE Int. Conf. Ind. Informat.*, Jul. 2011, pp. 666–670.

[11] D. Dechouniotis, X. Dimitropoulos, A. Kind, and S. Denazis, "Dependency detection using a fuzzy engine," in *Proc. 18th IFIP/IEEE Int. Workshop Distrib. Syst.—Oper. Manage.*, 2007, pp. 110–121.

[12] M. Ding, V. Singh, Y. Zhang, and G. Jiang, "Application dependency discovery using matrix factorization," in *Proc. IEEE 20th Int. Workshop Qual. Serv.*, Jun. 2012, pp. 1–4.

[13] I. K. Eltahir, "The impact of different radio propagation models for Mobile Ad Hoc Networks (MANET) in urban area environment," in *Proc. IEEE 2nd Int. Conf. Wireless Broadband Ultra Wideband Commun.*, Aug. 2007, pp. 1–9.

[14] J.-G. Lou, Q. Fu, Y. Wang, and J. Li, "Mining dependency in distributed systems through unstructured logs analysis," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 91–96, Jan. 2010.

[15] S. Mo, J. Hsu, J. Gu, M. Luo, and R. Ghanadan, "Network synchronization for distributed MANET," in *Proc. IEEE Mil. Commun. Conf.*, Nov. 2008, pp. 1–7.

[16] H. Neema, A. Kashyap, R. Kereskenyi, Y. Xue, and G. Karsai, "SOA-MANET: A tool for evaluating service-oriented architectures on mobile ad-hoc networks," in *Proc. IEEE/ACM 14th Int. Symp. Distrib. Simul. Real Time Appl.*, Oct. 2010, pp. 179–188.

[17] P. Novotny, A. L. Wolf, and B.-J. Ko, "Fault localization in MANET-hosted service-based systems," in *Proc. 31st Int. Symp. Rel. Distrib. Syst.*, Oct. 2012, pp. 243–248.

[18] P. Novotny, A. L. Wolf, and B. J. Ko, "Discovering service dependencies in mobile ad hoc networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2013, pp. 527–533.

[19] B. Peddycord, III, P. Ning, and S. Jajodia, "On the accurate identification of network service dependencies in distributed systems," in *Proc. 26th Int. Conf. Large Installation Syst. Admin.—Strategies, Tools, Tech.*, 2012, pp. 181–194.

[20] L. Popa, B.-G. Chun, I. Stoica, J. Chandrashekar, and N. Taft, "Macroscope: End-point approach to networked application dependency discovery," in *Proc. 5th Int. Conf. Emerging Netw. Experiments Technol.*, 2009, pp. 229–240.

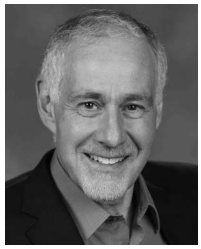
[21] F. Sailhan and V. Issarny, "Scalable service discovery for MANET," in *Proc. 3rd IEEE Int. Conf. Pervasive Comput. Commun.*, Mar. 2005, pp. 235–244.

[22] S. Tati *et al.*, "Diagnosing degradation of services in hybrid wireless tactical networks," presented at the SPIE Defense, Security, Sensing, Baltimore, MD, USA, 2013, Paper 874 210.

[23] S. Wang and M. A. M. Capretz, "A dependency impact analysis model for web services evolution," in *Proc. IEEE Int. Conf. Web Serv.*, 2009, pp. 359–365.
[24] D. Zhou and T.-H. Lai, "An accurate and scalable clock synchronization protocol for IEEE 802.11-based multihop ad hoc networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 12, pp. 1797–1808, Dec. 2007.

Petr Novotny received the M.Sc. degree in software systems engineering from University College London and the Ph.D. degree in computing from Imperial College London, and was a graduate intern at IBM Research. He is a post-doctoral Research Associate in the Department of Computing at Imperial College London. His interests are in software engineering and networking.

Bong Jun Ko received the B.S. and M.S. degrees from Seoul National University in South Korea and the Ph.D. degree in electrical engineering from Columbia University in 2006. He is a research staff member in the Cloud-Based Networking Department at IBM T.J. Watson Research Center. He received the best paper awards at the ICNP 2003 and IM 2013 conferences. His research interests include data analytics, cloud computing, mobile networking, and distributed systems.



Alexander L. Wolf (M'85–SM'09–F'11) received the B.A. degree in geology and computer science from the City University of New York and the M.S. and Ph.D. degrees from the Department of Computer Science at the University of Massachusetts at Amherst. He is a Professor in the Department of Computing at Imperial College London. He was previously a professor at the University of Lugano, Switzerland, the C.V. Schelke Chair at the University of Colorado at Boulder, and a member of the technical staff at AT&T Bell Laboratories in Murray Hill, New Jersey. His research interests span the areas of distributed systems, networking, and software engineering. He served on the editorial boards of the ACM Transactions on Software Engineering and Methodology and the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. He currently serves as President of the ACM. Wolf is a Fellow of the ACM and BCS.