Discovering Service Dependencies in Mobile Ad Hoc Networks

Petr Novotny and Alexander L. Wolf
Imperial College London
London, United Kingdom
{p.novotny09,a.wolf}@imperial.ac.uk

Bong Jun Ko
IBM T.J. Watson Research Center
Hawthorne, New York, USA
bongjun_ko@us.ibm.com

***Abstract*—The combination of service-oriented applications, with their run-time service binding, and mobile ad hoc networks, with their transient communication topologies, brings a new level of complex dynamism to the structure and behavior of software systems. This complexity challenges our ability to understand the dependence relationships among system components when performing analyses such as fault localization and impact analysis. Current methods of dynamic dependence discovery, developed for use in fixed networks, assume that dependencies change slowly. Moreover, they require relatively long monitoring periods as well as substantial memory and communication resources, which are impractical in the mobile ad hoc network environment. We describe a new method, designed specifically for this environment, that allows the engineer to trade accuracy against cost, yielding dynamic snapshots of dependence relationships. We evaluate our method in terms of the accuracy of the discovered dependencies.**

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 Web Services Architecture, computations are structured as a set of services that respond to requests, where a request typically originates at a user-facing client. The computation fulfilling each request results in a cascade of further requests across some subset of the services. 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 service binding mechanism (so-called "service discovery"). The consequence is that the dependencies among run-time instances of services cannot be 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 [1], [2], [3], [4], [5], [7], [8], [10], [13], [14]. 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). Mobility and ad hoc networking bring increased dynamicity to service dependencies, beyond those caused by the basic service-binding regime. Moreover, the MANET environment is characterized by 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 method for use in the MANET environment. Our intuition is that dependence discovery should capture snapshots of dependence data relevant to each service request of concern, rather than determine 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.

Our approach is based on the use of *monitors* deployed onto the mobile hosts. 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 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.

We introduce our dependence discovery method and evaluate its sensitivity to a distinguishing aspect of the MANET environment, namely *time-dependent behavior*. The evaluation is carried out through a series of simulation-based experiments under various scenarios that represent a range of service connectivities and critical parameter settings.

978-3-901882-50-0 ©2013 IFIP

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 [1], [2], [3], [4], [8], [10] focuses on coarse-grained dependencies between network hosts, which are usually described in terms of IP addresses and port numbers. They can be augmented with additional information, such as port mappings [7] or a classification of client applications [13]. On the other hand, service-level discovery [5], [6] focuses on the detection of fine-grained relationships between services. Services are hosted in application containers, such as JavaEE and .NET, and typically associated with application identifiers, such as URLs.

Our 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 and impact analysis in service-based software systems. 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 found in other methods [4], [6].

Many of the service-level discovery methods apply statistical techniques to traffic traces. These 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 [1], [2], [3], delay distribution [7], time difference of messages [5], or timing and frequency of packet flows [8]), 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 do exist. For example, in Macroscope [13] a subset of packets and network connections is sampled and analyzed to identify relationships between network flow data and applications. Lu et al. [10] collect system log data and correlate the events in the logs using data-mining techniques. Magpie [4] instead correlates the 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, the monitors actively summarize and aggregate dependence information before that information is transferred (e.g., they may transfer only the number of messages exchanged between two services, rather than sending the content of those messages).

III. Dependence Discovery and Representation

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

A. Design Assumptions

In order to employ our method, three basic prerequisites must be met. First, to obtain complete dependence information, the monitors should be deployed on the mobile hosts that are either the source or the target of service messages; intermediate hosts 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]. The shortest period we use for time-stamping data is 6 milliseconds, well within this precision. Third, the monitors must be able to observe service messages and obtain information from those messages, such 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.

B. 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 S_i and S_j, 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. Figure 1 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 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.

Figure 1 highlights the services involved in a particular *conversation* originating at client C3. In service-based systems, a conversation is the set of messages exchanged during the processing of a client request. Service S3 uses services S9 and S20 to satisfy C3's request, while the other dependent services in the figure, such as S11, are used in other conversations.

An intra-dependence is a more complex relation between services that relates an incoming inter-dependence to an out-

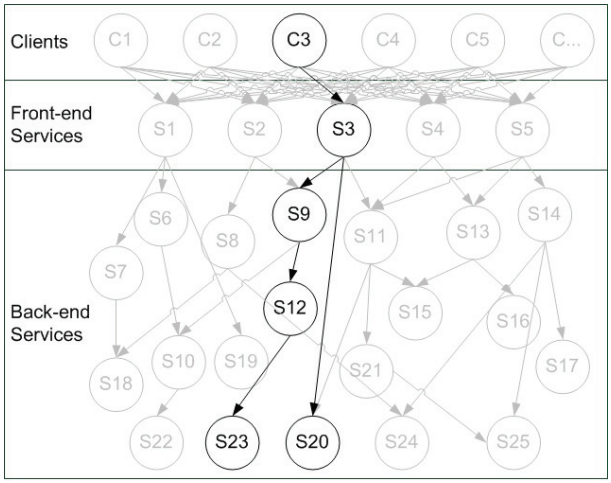


Fig. 1: Inter-dependencies of a hypothetical system. The services include front-end client services and back-end processing services. A single conversation originating at client C3 is highlighted.

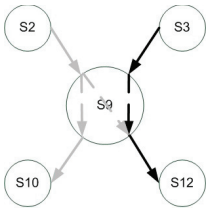


Fig. 2: Intra-dependencies (dashed lines) of service S9.

going inter-dependence. Intra-dependencies thus reflect greater insight into the nature of service dependencies than do the basic inter-dependencies. This is illustrated in Figure 2, 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 Figure 2 to that available in Figure 1. 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 Figure 1.

C. Discovering and Storing Dependencies

Dependencies arise from the flow of messages among services. To discover dependencies, we must therefore track these flows. Because we aim 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* 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 using monitors is that they allow dependencies to be discovered 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 actively aggregate and summarize the information. 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. Pairs of source and target services that induce inter-dependencies can be identified from the flow of messages exchanged between the services. In a service-based system, services are uniquely identified by application-specific identifiers, such as the URIs of the Web Services framework. Although the specific type of information provided within messages differs with the service platform and standard used, request messages always contain identifiers for the requested services.

Since a monitor is aware of the identity of the service with which it shares a container, it can record outgoing inter-dependencies simply by extracting the identifiers of target services 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. The inter-dependencies are therefore easily and immediately discoverable in all existing service invocation protocols such as SOAP and REST, and even from plain HTTP requests.

Intra-Dependence Discovery. Intra-dependence discovery requires knowledge of both outgoing as well as incoming inter-dependencies. However, discovering incoming inter-dependencies is a bit involved, as it requires 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`). With source and target identifiers present in messages, monitors can detect all incoming and outgoing inter-dependencies.

To expose the correspondence between the incoming and outgoing inter-dependencies of a service, we rely on the presence of *conversation identifiers* within messages. Of course, conversation identifiers must be implemented at the application level by service developers. Fortunately, this is a relatively routine task, as there are several existing standards for doing so, including WS-Addressing, WS-SecureConversation, and WS-Coordination. Discovering an intra-dependence then reduces to having a monitor relate incoming and outgoing messages using the conversation identifiers appearing in both.

Storing Dependencies at Hosts. Monitors store the history of the dependencies they have seen for some bounded period of time, divided into time slots. The dependence data are stored with a sliding expiration window such that only a limited history is captured, using a data structure representing time slots. 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 time slot. 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 slot size of 0.1 seconds will provide up to 10 times more data compared to a slot size of 1 second, but will require 10 times more space to store those data. Beyond the slot size, the size of the whole history can be controlled through the pruning of expired time slots.

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 required at each monitor is determined. 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 an monitor during the time period, and B_{inter} and B_{intra} are the maximum sizes of the identifiers of inter- and intra-dependencies.

The aggregation of observed service interactions into dependencies can cause the monitors to "forget" some of the details necessary to reconstruct 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 Figure 2. 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 aggregated dependence information held by the monitor will not, in our current approach, record the true dependencies regarding these individual conversations.

D. Dependence Graph

A dependence graph (DG) is a directed acyclic graph constructed from nodes representing services and edges representing direct inter-dependencies. The direction of an edge represents the direction 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 and network behaviors, 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 conversation of interest).

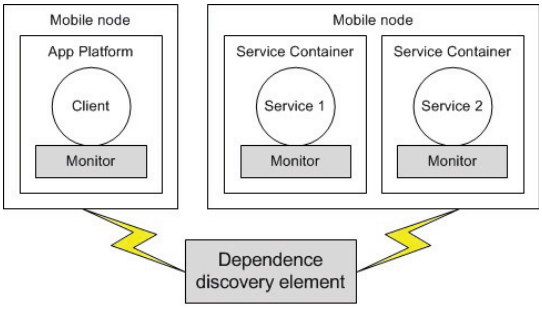


Fig. 3: Architecture of dependence discovery.

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

The distributed monitors used in our method provide information to a centralized dependence discovery element, as illustrated in Figure 3. The intent of this architecture is to minimize resource utilization, while still providing timely data. The monitors perform continuous dependence discovery and maintain aggregated dependence data.

The discovery element is a logical entity that can be deployed on any host of the network, or even in multiple instances on some or all the hosts. At the same time, the discovery element can be accessed locally or remotely by a network manager. Most importantly, the discovery element can be used as a component of an automated network analysis task, such as fault localization. Consider, for example, a monitor that detects a symptom of a system failure in the form of an exception or a response timeout. In order to identify the likely root cause of the observed failure, the fault localization component would contact (one of) the discovery element(s) in order to obtain the DG associated with the client that initiated the failed conversation. This DG would serve as an input to the fault analysis carried out by the network manager.

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 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, then, the data are harvested by a breadth-first walk of the monitors, where the walk is rooted at the monitor associated with the client of interest. For certain analyses, we may additionally

limit the dependence discovery to a specific conversation, something particularly useful in fault localization.

IV. Evaluation

In our method, dependence graphs 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. The data provided to the discovery element include both relevant and irrelevant information, since any given monitor will provide data about *all* interactions traversing its associated service during the time window. These interactions involve not just those of the given conversation of interest, but also those of others.

Under such circumstances it would be difficult for a dependence discovery method to provide a perfect result. Moreover, the method by design loses information (e.g., monitors retain only aggregate data, not individual messages) and is sensitive to the dynamics of the operational environment. Thus, the evaluation questions of interest center mainly on the accuracy of the resulting dependence graph. In particular, we examine the impact of time window size on the accuracy of the dependence graph. We also compare our method to other methods. Additional results are reported elsewhere [11].

A. Methodology

The evaluation is carried out on a simulation framework for service-based systems hosted on MANETs. The framework consists of a simulation engine built on top of the discrete-event network simulator NS-3 extended with higher-level abstractions for simulating service entities and their interactions [11]. 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:

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

where $D(C)$ are the discovered dependencies, $GT(C)$ are the ground-truth dependencies, true positives are in their intersection, and false positives are in the set difference.

The network-level behaviors in the simulation are based on the log distance model with path-loss exponent 3 for wireless signal propagation, reproducing a network operated in urban areas [9]. The network consists of 50 mobile hosts deployed in a 75x75 meter area, and uses the Optimized Link State Routing Protocol (OLSR) for routing. In the experiments, we set the node mobility speed to 10 m/s. Other mobility speeds result in similar outcomes, as we report elsewhere [11].

The service-level parameters used in our simulations are derived from standard values found in 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 (see Figure 1). Each service exposes two methods to be used by other services and clients.

We use "low", "medium", and "high" connectivity scenarios. The average number of services involved in a conversation is 2.03, 3.21, and 7.9 in each connectivity configuration, respectively. The impact of having more services involved in a conversation (due to a higher connectivity) is that the time to complete the conversation increases.

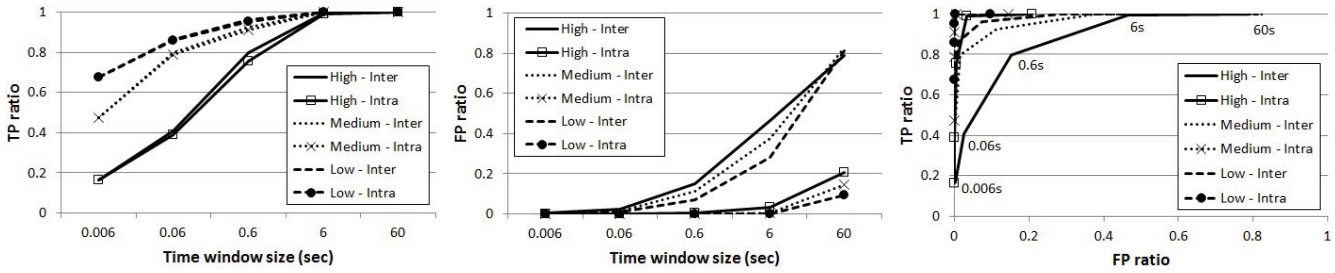
We collect our results from 30 minutes of simulated execution time after excluding 30 seconds of warm up. Each combination of parameters in our experiments results in thousands of conversations occurring during the simulated 30-minute execution. For instance, the low, medium, and high connectivities result in 7165, 8139, and 8440 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 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, 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 dependence graph. 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 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 conversation of interest.

We calculate the TP and FP ratios for both inter- and intra-dependence discovery separately. Figure 4 depicts the results, where each data point is the ratio averaged over all conversations. The variances of the 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.

As shown in Figure 4a, increasing the time window size increases the TP ratio, both for inter- and intra-dependencies. However, increasing the time window size also increases the FP ratio, as shown in Figure 4b. 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 Figure 2). To directly display the trade-off between the TP and FP ratios under various time window sizes, we plot them against each other in Figure 4c, where each point represents the given time window size.



(a) (b) (c)

Fig. 4: Accuracy of inter- and intra-dependence discovery methods for different time window sizes given as (a) TP ratios, (b) FP ratios, and (c) a trade off between TP and FP ratios.

C. 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 network level and those at the service level. 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 dependence graphs 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 addresses and ports. It then builds dependence graphs using external information provided to it about the deployment of clients and services on hosts.

We use the medium connectivity scenario in our comparison, where the average number of ground-truth dependencies is 3.21. We use a 60s time window size, which is large enough to capture all such dependencies (see Figure 4). The comparison then reduces to one based on the false dependencies appearing in the discovered dependence graphs.

The results are reported in Figure 5, 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 in the dependence graph. Both are computed as the average over the total number of conversations (8139) occurring in the 30-minute execution period. 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 of these alternative methods build dependence graphs from long-term observations, they do not adequately filter out stale dependencies caused by the dynamics of the scenarios, resulting in higher FP ratios than our discovery method. Moreover, the network-level method includes even more false positives than the service-level methods because it builds dependence graphs from coarser-grained information. 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.

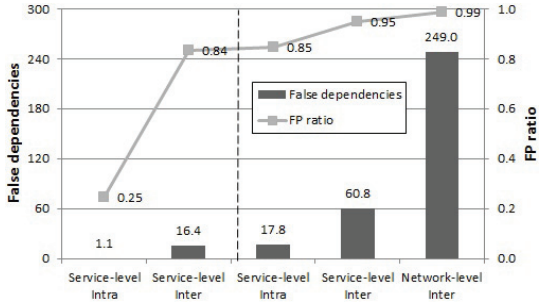


Fig. 5: Comparison of methods in medium connectivity scenario. Results for existing methods are to right of dashed line.

V. Conclusion

We have presented a run-time 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 large amounts of evidence data collected over long periods. Through a 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. Although not reported here, its data storage and data transfer requirements scale well with the number and connectivity of the services involved [11].

Dependence information is not particularly useful in and of itself, but instead serves as a building block for important analysis capabilities. We are currently developing such analyses, including those for probabilistic fault localization [12] and cross-layer performance anomaly diagnosis.

Acknowledgement. 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] P. Bahl, P. Barham, R. Black, R. Ch, M. Goldszmidt, R. Isaacs, S. K, L. Li, J. Maccormick, D. A. Maltz, R. Mortier, M. Wawrzoniak, and M. Zhang. Discovering dependencies for network management. In *Proceedings of the Fifth Workshop on Hot Topics in Networks*, Nov. 2006.

[2] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 13–24. ACM, August 2007.

[3] P. Barham, R. Black, M. Goldszmidt, R. Isaacs, J. MacCormick, R. Mortier, and A. Simma. Constellation: Atomated discovery of service and host dependencies in networked systems. Technical Report MSR-TR-2008-67, Microsoft Research, 2008.

[4] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 2004.

[5] S. Basu, F. Casati, and F. Daniel. Toward web service dependency discovery for SOA management. In *Proceedings of the IEEE International Conference on Services Computing*, pages 422–429. IEEE Computer Society, 2008.

[6] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based faliure and evolution management. In *Proceedings of the Symposium on Networked Systems Design and Implementation*. USENIX Association, 2004.

[7] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, pages 117–130. USENIX Association, 2008.

[8] D. Dechouniotis, X. Dimitropoulos, A. Kind, and S. Denazis. Dependency detection using a fuzzy engine. In *Proceedings of the 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, number 4785 in Lecture Notes in Computer Science, pages 110–121. Springer-Verlag, 2007.

[9] I. K. Eltahir. The impact of different radio propagation models for mobile ad hoc networks (MANET) in urban area environment. In *Proceedings of the 2nd International Conference on Wireless Broadband and Ultra Wideband Communications*. IEEE, Aug. 2007.

[10] J.-G. Lou, Q. Fu, Y. Wang, and J. Li. Mining dependency in distributed systems through unstructured logs analysis. *SIGOPS Operating Systems Review*, 44:91–96, March 2010.

[11] 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.

[12] P. Novotny, A. L. Wolf, and B.-J. Ko. Fault localization in MANET-hosted service-based systems. In *Proceedings of the 31st International Symposium on Reliable Distributed Systems*, 2012. To appear.

[13] L. Popa, B.-G. Chun, I. Stoica, J. Chandrashekar, and N. Taft. Macroscope: End-point approach to networked application dependency discovery. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, pages 229–240. ACM, 2009.

[14] S. Wang and M. A. M. Capretz. A dependency impact analysis model for web services evolution. In *Proceedings of the IEEE International Conference on Web Services*, pages 359–365. IEEE Computer Society, 2009.

[15] 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.