Responsiveness of Service Discovery in Wireless Mesh Networks

Andreas Dittrich, Daniel Solis Herrera, Pablo Coto and Miroslaw Malek
Advanced Learning and Research Institute (ALaRI)
Università della Svizzera italiana
Lugano, Switzerland
Email: {andreas.dittrich,solisd,cotobj,malekm}@usi.ch

***Abstract*—Service Discovery (SD) is an integral part of service networks. Before a service can be used, it needs to be discovered successfully. Thus, a comprehensive service dependability analysis needs to include the dependability of the SD process. As a time-critical operation, an important property of SD is responsiveness: the probability of successful discovery within a deadline, even in the presence of faults. This is especially true for dynamic networks with complex fault behavior such as wireless networks. This work presents results of a comprehensive responsiveness evaluation of decentralized SD in wireless mesh networks, specifically active SD using the *Zeroconf* protocol. For this reason, the *ExCovery* framework has been employed, which provides a unified description, execution, measurement and storage of distributed system experiments. *ExCovery* suports the Distributed Embedded System (DES) wireless testbed at Freie Universität Berlin. We present and discuss the results of the experiments and show how SD responsiveness is affected by the position and number of requesters and providers as well as the load in the network. The results clearly demonstrate that in all but the most favorable conditions, the configurations of current SD protocols struggle to achieve a high responsiveness. We further discuss results that reflect the long-term behavior of the wireless testbed and how its varying reliability may impact SD responsiveness.**

***Index Terms*—Responsiveness; Service Discovery; Wireless Mesh Networks; Experiments; Zeroconf**

I. Introduction

In *Service-Oriented Architecture* (SOA), the principle of discoverability demands that service descriptions include structured data to facilitate publishing and discovering of individual service providers [1]. *Service Discovery* (SD) deals with the communication of this data. A set of abstract service classes S can be provided on a set of concrete service providers P which then use a *Service Discovery Protocol* (SDP) to make the service known to requesting clients C. This process may be supported by a set of service registries R. SD supports two different architectures: Two-party, where all SD actors $A \subseteq P \cup C$ and three-party, where $A \subseteq P \cup C \cup R$ and $A \cap R \neq \emptyset$. In an adaptive or hybrid architecture SD can switch between the two. A service client discovers providers by a combination of passively listening to announcements and actively sending requests, with retries in specific intervals. Depending on role and architecture, different communication types are used: unicast, multicast or broadcast.

SD is a real-time operation and one of its key properties is responsiveness – the probability to finish successfully within a deadline, even in the presence of faults [2]. More precisely, responsiveness constitutes the probability that a SDP enumerates a defined ratio of available provider instances $x = p/P$ within a deadline t_D as required by the discovering client. SD is an integral part of service usage and comprehensive service dependability evaluation should include its responsiveness. This is because common dependability metrics, such as availability and performability, are only independent of SD responsiveness if a successful discovery is assumed at the time of requesting a service. For example, the performability of a service until a deadline decreases with decreasing SD responsiveness, hence, a longer time needed to discover that service with a certain probability: Less time to perform in general means a lower probability to perform as required [3]. On the other hand, reducing the time to discover the service increases the risk of not finding it, in which case it wouldn't be able to perform at all. Unfortunately, until now few works examine SD responsiveness and in service dependability evaluation, it has generally been neglected.

For modern decentralized networks, such as wireless mesh networks with possibly mobile nodes, SD becomes ever more important as the number and position of providers may change dynamically. Additionally, wireless mesh networks exhibit complex fault behavior and fault dependencies [4]. As can be seen in the model-based evaluation in [5], SD responsiveness is difficult to predict in such networks and varies dramatically. Optimizing the responsiveness of SD in these environments should thus be a prime target of service network research.

The goal of this work is an experimental evaluation of responsiveness in wireless mesh networks, to demonstrate if and how SD responsiveness changes depending on the position and number of requesters and providers and depending on the load on the network. We employ the *ExCovery* framework [6] for experiments on dependability in distributed systems and run several series of experiments to examine the responsiveness of SD in wireless mesh networks. The presented research complements the existing work by providing a comprehensive experimental evaluation with a realistic fault model. The experiments serve the following purposes:

1) They demonstrate how to use *ExCovery* for experiments in the wireless DES testbed and how the comprehensive range of measurements stored during runs facilitates diverse types of analysis.

2) The analysis shows the long-term behavior of the DES testbed and the effect of internal and external faults. These faults are being recorded by *ExCovery* during experiment execution and have to be taken into account when interpreting the results. Internal faults contain node crashes or clock drifts, external faults comprise all types of wireless interference or also forced interruptions during execution of experiments.
3) Third, the experiments allow a deep insight into SD responsiveness in wireless mesh networks. For several realistic discovery scenarios, the responsiveness is shown depending on the deadline t_D of the discovery operation, the distance of actor nodes, the load in the network and the required number p of providers P to be discovered. This analysis is done both for the individual SDP packets as well as the complete discovery operation, which includes retries by the requester in case response packets do not arrive in time. The former allows to infer conclusions for other application protocols which use similar packets and can provide input for analytical models such as in [5]. As such, the gained results could be used to verify the validity of employing these models when optimizing the responsiveness of SD configurations in wireless mesh networks.

The remainder of this paper is structured as follows. After an overview of related work in Section II, Section III provides a brief presentation of the concepts used in *ExCovery* and how it was used in the context of this work. The DES testbed is introduced in Section IV. The experiment setup and configurations can be found in Section V. Sections VI and VII present and discuss the results of analysis. Section VIII concludes the work.

II. Related Work

A comprehensive description of SOA and its principles can be found in [1]. Regarding discoverability, [7] provides more details on service discovery operations and architectures. An overview and comparison of current SDPs is presented in [7], [8]. In [9]–[11], SDPs for pervasive and ubiquitous computing systems are surveyed. Such systems are the target environment of the experimental analysis in the paper at hand. Dabrowski et al. evaluate different dependability properties of existing discovery protocols in [12]–[14]. These include *update effectiveness*, the probability to timely restore a consistent state after failure, which resembles a specific case of responsiveness. The foundations for the responsiveness metric as used in this paper have been laid out in [2]. We refer to [15] for a comprehensive definition of various other service dependability metrics, among them performability [3].

Not considered in [12]–[14] was active SD responsiveness during regular operation. Also, the evaluated protocols do not include the widespread *Zeroconf* protocol [16]. Active discovery using *Zeroconf* is evaluated in experiments in [17]. However, the fault model in [17] includes just packet loss but no packet delay and the work investigates only a fully connected, single hop network. Dittrich et al. [5] provide a hierarchy of stochastic models to analytically evaluate the SD responsiveness of common protocols based on the quality of network links. They apply these models to wireless mesh networks, whose general problems and challenges are presented in [4]. The results presented in the work at hand could serve to validate the stochastic models in [5] by correlating model and experiment results under similar conditions. Furthermore, parts of the results may be used as input data for stochastic models to evaluate SD responsiveness in diverse scenarios instead of running time-consuming experiment series for each one of them.

The *ExCovery* framework [6] has been developed recently to support dependability research of distributed processes. Its core is a formal experiment description that facilitates automated checking, execution and additional features, such as visualization of experiments. *ExCovery* is expected to foster experiment repeatability, comparability and transparency as it offers a unified experiment description, measurement mechanism and storage of results and is described in detail in [6], [18]. In this work, we will only highlight the main features of *ExCovery* and focus on the description of the conducted experiments and the analysis of their results.

III. *ExCovery* – The Experimentation Environment

As mentioned previously, all experiments were conducted using the experiment environment *ExCovery*. While we refer to [6] for a complete description of the framework, this section will highlight how the major features were used for the experiments on SD presented in Sections V,VI and VII.

The core of experiments carried out with *ExCovery* consists of a formal description in the extensible markup language (XML). An XML schema is provided with the framework code, which is made available to interested researchers on request. Four main parts are defined in the description of experiments:

General This part defines several parameters describing general information about the experiment to facilitate categorization, for example the name of the experiment and the time it was started.

Platform The platform definition contains all concrete nodes that take part in the experiment. Nodes are classified into two types: Actor nodes take part in the process under experimentation, in this case the discovery process. Environment nodes take part in the specified environment processes, such as load generation and fault injection. All nodes help with regular routing. Figure 1 shows an exemplary platform definition with four actor nodes to carry out the SD operation and five environment nodes.

Factors The factor definition contains a list of factors and their levels whose effect on the results is to be studied. Concrete actor nodes are mapped to abstract roles of the discovery process, such as requesters and responders. The number of load generators, which are pairs of environment nodes, and the data rate per pair is defined. Also, the number of repetitions of each factor combination is stated as a special factor. An exemplary factor definition is listed

```
<platform_specs>
 <description>
  Nodes to be used during the experiment. Four
  ↪ actors (one requester, three responders) and
  ↪ five environment nodes for traffic generation.
 </description>
 <spec_node_mapping>
  <spec_actor_map abstract_id="R0" id="t9-154"
  ↪ ip="172.18.17.8" />
  <spec_actor_map abstract_id="P0" id="t9-006"
  ↪ ip="172.18.17.104" />
  <spec_actor_map abstract_id="P1" id="t9-147"
  ↪ ip="172.18.17.179" />
  <spec_actor_map abstract_id="P2" id="t9-020"
  ↪ ip="172.18.17.88" />
  <spec_env_map id="t9-117" ip="172.18.17.52" />
  <spec_env_map id="t9-k61" ip="172.18.17.92" />
  <spec_env_map id="t9-169" ip="172.18.17.50" />
  <spec_env_map id="t9-018" ip="172.18.17.14" />
  <spec_env_map id="t9-022a" ip="172.18.17.80" />
 </spec_node_mapping>
</platform_specs>
```

Fig. 1. Exemplary platform definition for an experiment description. The *abstract_id* allows to identify abstract nodes for the mapping to actor roles. IP addresses help in filtering the raw capture files for packet based analyses.

```
<factorlist>
 <factor usage="blocking" id="fact_nodes"
 ↪ type="actor_node_map">
  <description>Mapping of abstract nodes to actor
  ↪ roles of the discovery process
  </description>
  <levels>
   <level>
    <actor id="requester">
     <instance id="0">R0</instance>
    </actor>
    <actor id="responder">
     <instance id="0">P0</instance>
     <instance id="1">P1</instance>
     <instance id="2">P2</instance>
    </actor>
   </level>
  </levels>
 </factor>
 <factor usage="random" id="fact_pairs" type="int">
  <description>
   Number of node pairs for load generation,
   ↪ randomly distributed in the network
  </description>
  <levels>
   <level>10</level>
  </levels>
 </factor>
 <factor usage="constant" id="fact_bw" type="int">
  <description>Datarate per node pair</description>
  <levels>
   <level>100</level>
   <level>500</level>
  </levels>
 </factor>
 <replicationfactor usage="replication"
 ↪ id="fact_replication_id" type="int">1000
 </replicationfactor>
</factorlist>
```

Fig. 2. Exemplary factor definition for an experiment description. The factors are explained within the definition code. The replication factor denotes the number of experiment runs for each factor level combination.

in Figure 2. It maps the actor nodes from the platform definition to requester and responder roles, defines 10 node pairs for load generation which will first have a data rate of 100, then 500 kbit/s each. Each factor combination will be run 1000 times as stated by the replication factor. Since there is only one node mapping and one factor level for the load generation, there will be 1000 runs for each data rate.

Processes This part contains descriptions of *experiment processes*, executed only by specific actor roles and *environment processes*, which are executed by possibly all nodes. *ExCovery* describes processes as series of interdependent actions and events. Due to space constraints, no full description of the processes for the discovery actors and environment nodes is shown here. A shortened version can be found in [6], the full descriptions are available on request together with the experiment results.

ExCovery executes an experiment on a dedicated master node. The description is read and the described processes are run with all factor and factor level combinations for a given number of repetitions. Measurements during these runs are stored in temporary directories on both the nodes and the master and comprise events defined in the process descriptions, such as *Discovery Request Sent* or *Discovery Response Received*, and network packets belonging to these processes.

ExCovery is designed to provide a consistent state for each repetition and to minimize effects of any process not described in the experiment description, including its own processes. As such, each repetition consists of a preparation phase, a measurement phase which is the core of the experiment and a cleanup phase. During the measurement phase, only the processes defined in the experiment description are executed and their effects recorded by specified nodes. Only after completing all runs these measurements are imported from their temporary directories, conditioned to contain a globally valid time stamp and written to a database. This database includes the experiment description, logged events and packets of the described processes and diverse information about the state of the testbed itself. The latter can be used to further decrease the effect of external processes (see Sections VI-A and VI-B). The database represents one full experiment and can be shared to allow transparent reusability and repeatability.

For the work at hand, a set of specific analysis functions was added to the framework code to support the presented results (see Sections VI and VII). Additionally, diverse enhancements were developed to improve monitoring during execution of experiments, given the long durations of the individual experiments (see Section V). Finally, the framework code for the execution of experiments was redesigned to reduce the duration of experiments, especially when they are partially completed and resumed. Due to space constraints and since they are not the focus of this work, the implementation of these changes will not be discussed in detail. All of them have been merged to the main framework code respository and are available on request.

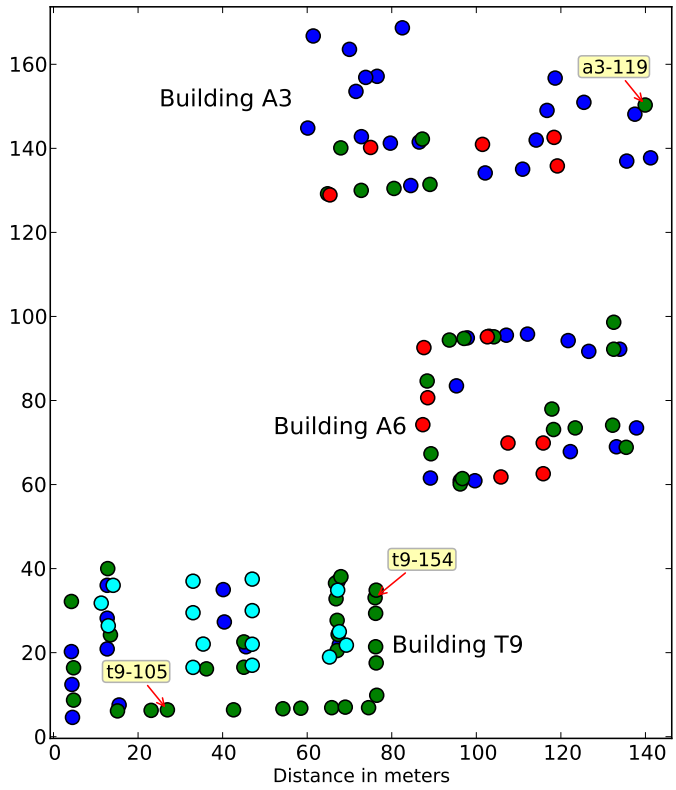


Fig. 3. Overview of the wireless DES testbed. Three buildings can be distinguished, nodes colors define their building floors.

IV. The Wireless Testbed

The wireless DES testbed consists of roughly 130 uniform nodes spread indoors and outdoors over three adjacent campus buildings at Freie Universität Berlin (FUB). An overview of the testbed topology is depicted in Figure 3 which shows the geographical location of the nodes in buildings *a3*, *a6* and *t9*. Nodes have been color-coded to distinguish the different building floors. Special nodes that are the focus of the following analysis are labeled with their identification string. Wireless network links have been left out for reasons of visibility. While the mesh network forms reasonably dense clouds within the buildings, the connections between buildings are not optimal. Building *a6* and *t9* are connected by several links but *a3* and *a6* are often only connected by a single bridge, depending on the overall wireless signal quality. More in-depth information about the properties of the DES testbed can be found in [19].

ExCovery was chosen to support the DES testbed as the first testbed for various reasons. First of all, as opposed to a simulation environment, it allows to evaluate processes in a network with realistic fault behavior. Second, the DES testbed allows to generate manifold topologies due to its wide distribution of nodes on campus and the ability to manipulate their wireless signal range. Finally, the nodes run a relatively modern Linux distribution which simplifies the development and deployment of new software. These reasons were also valid for the decision to use the DES testbed in this work. However, *ExCovery* is designed to support a range of different environments. The requirements that a testbed needs to fulfill to be supported by *ExCovery* are listed in [6].

As service discovery protocol, the wide-spread *Zeroconf* protocol suite based on multicast DNS [20] and DNS-based service descriptions [21] was used. A full description of the protocol is found in [16]. *Zeroconf* implements a two-party architecture and all messages, requests and responses, are sent via multicast. Multicast provides considerable challenges in wireless mesh networks that need to rely on costly flooding mechanisms to deliver these messages. Nevertheless, *Zeroconf* was developed for mobile and dynamic networks and both its focus and prominence make it a prime target for examination. Additionally, the results presented in Sections VI and VII provide insight also in the behavior of other SD protocols, which implement the same operations.

V. Experiment Setup

The experiments have been run in three series from May 2013 to May 2014. Each series took several weeks to complete, due to the overhead involved when carrying out experiments. Every single discovery operation needs to be initialized properly, measured and cleaned up (see Section III). During initialization, the SD daemons are started on all providers. Then, all SD packets are dropped on all nodes for a specific period to prevent service announcements and delayed responses from previous runs on the network. After dropping, the load generation is started. To simulate additional load on the network, environment nodes were chosen randomly to exchange UDP [22] packets at a given data rate. UDP was chosen due to its ”send and forget” strategy, to be able to control the data rate at the given level without corrective measures such as flow and congestion control.

During the measurement phase, the SD process is started and SD packets captured as well as defined SD events recorded, such as ”search started” or ”provider found”. Either when a required number of providers has been found or a time out has been reached, the measurement phase ends and cleanup starts, where load generation is stopped and SD daemons are shut down. Due to the long time-outs of current SD protocols, one experiment run usually takes between 45 and 90 seconds to complete. The total number of runs in all series was 32004 of which 26670 provided valid results for analysis. Among the causes of rendering a run invalid were complete network outages or missing connectivity on too many actor nodes of the discovery operation. While requesters had to be connected at any time to produce valid results, we decided that up to 10 percent of providers where allowed to be temporarily disconnected.

The first two series of experiments cover the discovery of a single provider by a single client. This is a common scenario in service networks: A client needs to use a specific service in the network, such as the printing on a specific printer or the backing up to a network-attached storage (NAS). In both series, the requesting node was *t9-105*. In one series the provider was *t9-154* to cover scenarios where both nodes

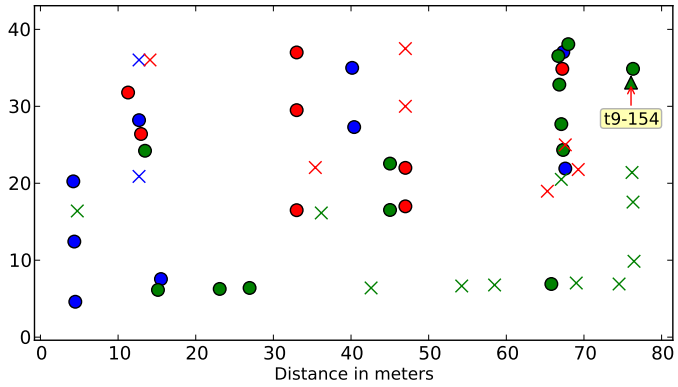


Fig. 4. Overview of building *t9* of the wireless DES testbed. Nodes colors define their building floors. Circle nodes are providers, crosses denote environment nodes for load generation. The requester *t9-154* has been labeled.

TABLE I
EXPERIMENT CONFIGURATION OF EACH SERIES

Experiment	Series I	Series II	Series III
Number of nodes (max)	115	119	51
Nodes in buildings	*t9*, *a3*, *a6*	*t9*, *a3*, *a6*	*t9*
Number of requesters	1 (*t9-105*)	1 (*t9-105*)	1 (*t9-154*)
Number of responders	1 (*a3-119*)	1 (*t9-154*)	1 . . . 30
Number of load generators	0 . . . 50	0 . . . 50	20
Bitrate per generator (kbit/s)	72	72	0 . . . 13824

are in one well connected cloud. In the second series the provider was *a3-119* to study the effects on SD responsiveness in scenarios where nodes are connected with sparse and possibly weak links. All other nodes participated in randomly distributed load generation at various levels. The position of nodes within the network is depicted in Figure 3.

A separate series of experiments was carried out only in the dense cloud of building *t9*. Requester *t9-154* was trying to discover up to 30 service providers while the remaining nodes generated load in the network. This reflects a scenario where a client wants to discover as many providers of a given service as possible to select the best one among them. The topology of these experiments is illustrated in Figure 4. Nodes can be distinguished by their form and color. Again, the colors define the three building floors the nodes reside on. The shapes define the node type: Circle nodes are providers, cross nodes are environment nodes used for load generation.

The most important configuration parameters of the three experiment series are summed up in Table I. The full experiment data, including XML descriptions, log files, packet captures and discovery event measurements are available on request for the research community to foster the transparency and repeatability of the presented results.

The results focus on two different sets of analyses of SD responsiveness. First, general measurements of the behavior of the testbed for the duration of the experiments are presented. These cover the variation of response times under similar conditions (see Section VI-A) and the clock drift of the network nodes (see Section VI-B) over time. The results should illustrate the behavior of the DES testbed over time and help to interpret the results of the discovery process analyses, which are done in the second set of analyses and presented in Section VII. It should be pointed out that results can only be representative for the scenarios which can be realized with the DES testbed. Uniform grid topologies, for example, cannot be created with the testbed. Thus, the significance of the results for such scenarios needs to be inspected before drawing conclusions.

VI. EXPERIMENT RESULTS – TESTBED

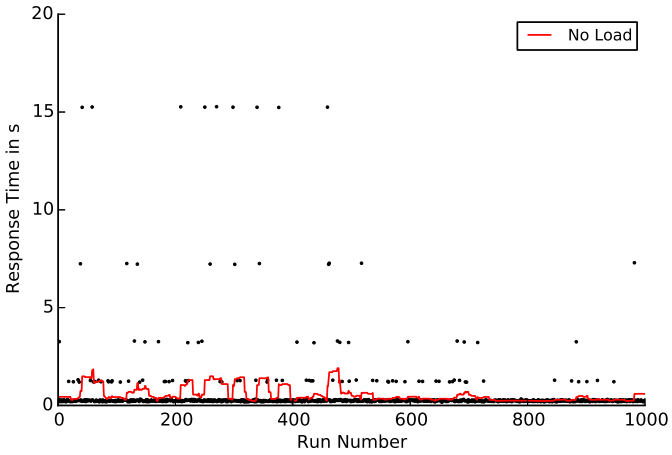
Ideally, a testbed should provide a controlled environment over a series of experiments. More precisely, held-constant factors should be known and accounted for in a later analysis and the effect of allowed-to-vary factors should be minimized. Nuisance factors with an unwanted or unknown effect on the results should be eliminated as much as possible. The different types of factors are described in [6].

Such stable and known conditions can usually only be achieved in simulations. In physical testbeds, especially wireless testbeds which are highly sensitive to external interference, the different nuisance factors are both difficult to determine and to measure. The probability of unwanted effects on the results is even higher the longer it takes to run the experiments. Given that the experiments carried out for this paper took several weeks to complete, a better knowledge of the testbed behavior was necessary. Although we cannot determine the source of all nuisance factors, we can measure them and show their effect.

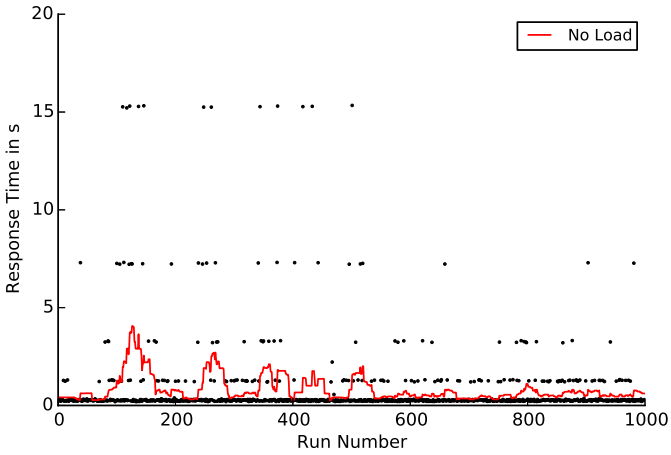
A. Response Times over All Runs

Figure 5 illustrates the effects of internal an external faults on the results for a part of four sets of experiments from Series I and II (see Table I). The response times of discovery operations, which in this case are the times for the first response to arrive at the requester, are plotted over 1000 experiment runs. The requesting node is *t9-105*, the providers are *t9-154* (Figures 5a and 5c on the left side) and *a3-119* (Figures 5b and 5d on the right side). Nodes *t9-105* and *t9-154* are in one cloud within the same building while *t9-105* and *a3-119* cover the maximum hop distance in the network. The node positions in the mesh network are shown in Figure 3. For each node pair, results are shown without additional load in the network (Figures 5a and 5b in the upper row) and with an additional load of 40 voice-over-IP streams with 72 kbit/s each, which amounts to roughly 2.8 Mbit/s traffic overall in the network (Figures 5c and 5d in the lower row).

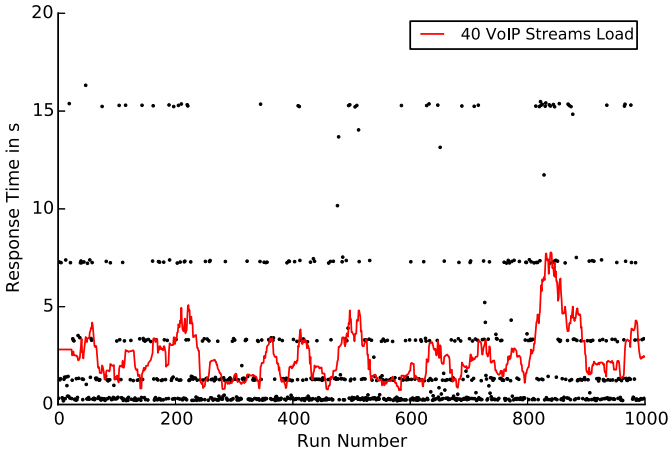
In the graphs, the dots reflect response times of individual discovery operations while the line denotes a moving average over 20 operations. Depending on the additional load on the network, the 1000 runs cover a period of $15-20$ hours so the graphs are showing long-term effects in the testbed. One can see that the response times generally increase with the load in the network and the distance of requester and provider. At times $1, 3, 7$ and 15 seconds we see a significant accumulation



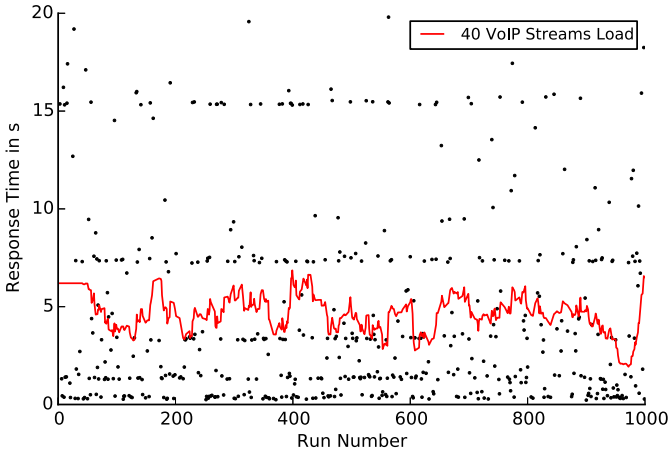
(a) Short distance, provider *t9-154*



(b) Maximum distance, provider *a3-119*



(c) Short distance, provider *t9-154*



(d) Maximum distance, provider *a3-119*

Fig. 5. Response times in testbed over 1000 experiment runs. Requester in all scenarios is *t9-105*, providers *t9-154* and *a3-119*. Figures 5a and 5b show results without additional load in the network, Figures 5c and 5d show results with an additional load of 40 voice-over-IP streams with 72 kbit/s each.

of responses. This corresponds to the default retry intervals of the *Zeroconf* discovery protocol, which starts with a timeout of one second and then doubles this timeout on every retry. The accumulation of responses very close to the retry intervals hints at packet loss having the decisive impact on response times: Either packets arrive in time or they get lost. With higher loads and distances, however, also packet delay comes into play which is especially visible in Figure 5d, where the accumulation at the retry intervals is less pronounced and response times generally have a wider distribution.

While the responsiveness over load and distance will be examined more in detail in Section VII, it can be noted that the response times are not independent over time. There are periods of consistently higher response times, such as in Figure 5b between runs 120 and 180 or in Figure 5c between runs 820 and 900. Given that these intervals span roughly an hour of experiment time, it seems highly improbable that these effects are random statistical accumulations. Instead, they hint at external causes that degrade the overall quality of the wireless testbed. Furthermore, those events happen very frequently. Finding the root cause of such anomalies is out of the scope of this work. The results are meant to demonstrate instead that any analysis based on average measurements should be done very carefully. Chosing the measurement history window size too big can easily lead to over- or underestimating the testbed quality. Also, it is important to measure and store all historical data, as does the *ExCovery* framework, to be able to detect and visualize such effects.

B. Node Clock Drift over Time in Testbed

While Section VI-A shows the effects of external factors on the experiment results, there are also internal factors that need to be measured. In this section, we focus on the clocks on the individual nodes. All nodes in the network synchronize their clocks before running experiments but to not interfere with the measurements, this is not done anymore during experiment execution and their clocks drift apart.

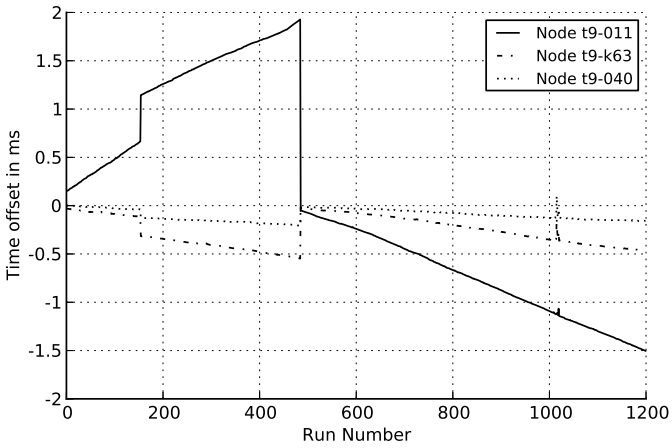


Fig. 6. Absolut clock differences between different providers and requester *t9-154* over a duration of 1200 experiment runs, approximately 20 hours.

Figure 6 shows the variations of clock offsets as measured over 1200 experiment runs (approximately 20 hours) in Series III. Each of the three lines represents the offset of a different node to requester *t9-154*. The lines abruptly change at the same positions of run 175 and 480. At run 175, the experiment was interrupted for an extended period so the clocks continued to drift apart before resuming experiments. This leads to the jump in the lines in direction of the clock drift relative to the reference node *t9-154*. At run 480, the experiments needed to be interrupted because the time slot in the testbed ended. The nodes were rebooted multiple times for experiments carried out by other DES testbed users. Before restarting the series, the nodes were manually synchronized, bringing the offset to reference node *t9-154* close to zero. Between those characteristic steps, the clocks steadily drift apart from each other. However, not only does every node drift differently from the others, even the drift of the individual nodes varies over time, as can be seen for node *t9-011*, which runs faster than the reference node *t9-154* until experiment run 480 but after that runs slower.

Clock synchronization is highly important for the analysis of real-time problems, such as service discovery. Given the observed behavior, it is obvious that the current testbed synchronization methods are not sufficient to guarantee consistent behavior over multiple hours of testbed usage. This justifies the approach of *ExCovery* to measure the time difference between the nodes on every single run. After the experiments are done, the stored event and packet times are corrected using these offsets before being imported to the result database for further analysis. This reduces the absolute synchronization error of all nodes to the precision of the measured timestamps, which improves measurement accuracy and reduces the likelihood of causal problems during subsequent analysis. A future improvement to *ExCovery* would be to optionally support time synchronization tasks during the initialization phase of each experiment run.

VII. Experiment Results – Service Discovery

After showing the variations of testbed behavior in Section VI, this Section focuses on the discovery process itself. A discovery operation consists of an initial request and a series of retries if an insufficient number of responses is received until a timeout. For each provider, only a single response packet needs to arrive at the requester. Given a total number of m providers, an operation to discover n providers is successful, when n providers have received at least one request packet and from each of these n providers one response has arrived at the requester within deadline t_D. Thus, the minimum of multicast messages to be sent would be $n + 1$, not including duplicate transmissions in the network due to the multicast flooding. Consequently, the maximum of messages for each request and subsequent retry is $m + 1$. It needs to be pointed out that the actual number of transmissions inside the mesh network is considerdably higher and grows with an increasing number of collisions.

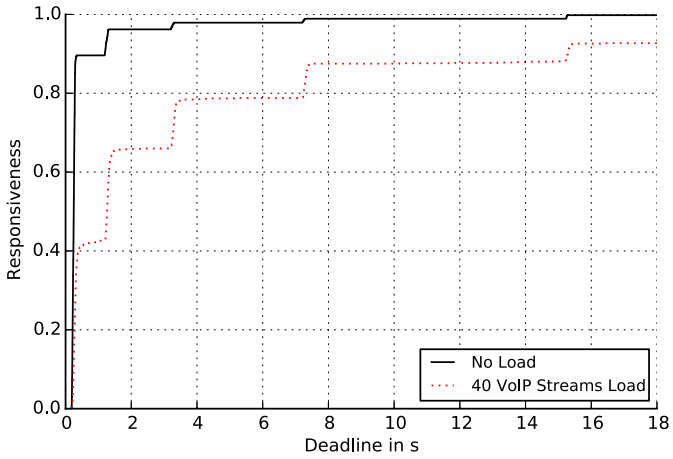
The probability of a request to arrive at n providers and of n responses from those providers to arrive until t_D at the requester denotes the responsiveness $R(t_D)$ of the discovery operation. In this section, we will investigate the behavior of $R(t_D)$ under varying influences. The SD protocol is *Zeroconf*, which uses DNS packets sent via multicast for both requests and reponses.

A. Responsiveness over Time

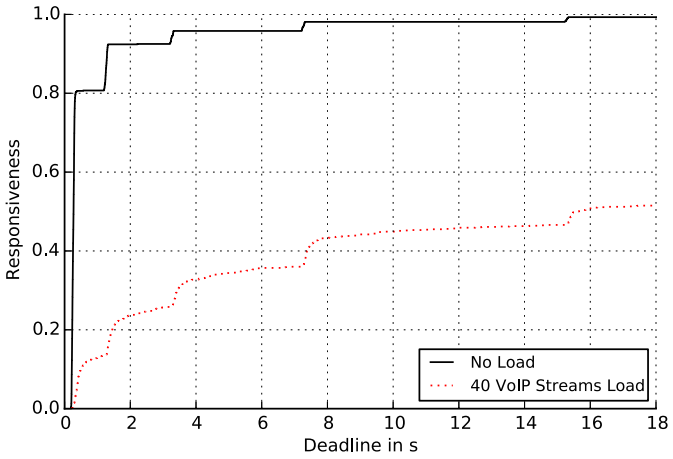
The first analysis covers the responsiveness $R(t_D)$ of SD over time. It examines how $R(t_D)$ increases with t_D, depending on the distance between requester and provider and the load in the network. To be able to isolate the effect of node positions on the results, only data from experiment Series I and II was used (see Section V and Table I). Thus, the discovery operations were carried out by only two actors, one requester *t9-105* and one provider: *t9-154* for a short distance and *a3-119* for the maximum distance in the network. The respective node positions are depicted in Figure 3. All DES testbed nodes participated in routing network packets.

Figure 7 illustrates the results. The upper graphs show the responsiveness of the discovery operation over time, for providers *t9-154* and *a3-119*. The two curves reflect the responsiveness for different load conditions. Responsiveness increases with every request being sent, which corresponds to the steps in the curves. It can be seen that $R(t_D)$ is generally lower with higher distance. It can further be noted that additional load in the network has a dramatic effect on the packet loss rates and considerably decreases $R(t_D)$. The 40 VoIP streams are randomly distributed in the network, changing each experiment run. They impose a combined load of $2.8Mbit/s$, which does not seem much compared to the maximum theoretical data rates in the network. Still, even with $t_D = 18s$ there is only a 50% chance of discovering provider *a3-119* under these conditions.

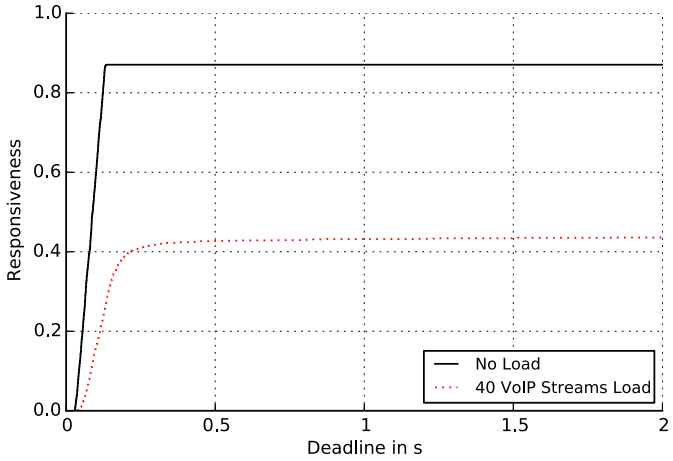
The results are in line with the ones presented in [17]. Current service discovery protocols work well when conditions



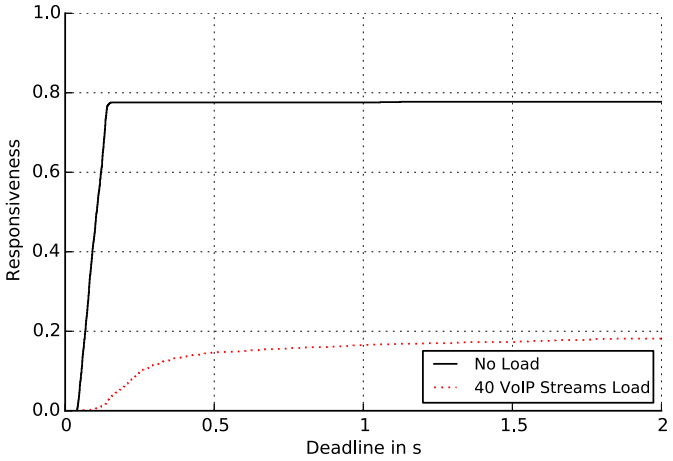
(a) Provider *t9-154*



(b) Provider *a3-119*



(c) Provider *t9-154*



(d) Provider *a3-119*

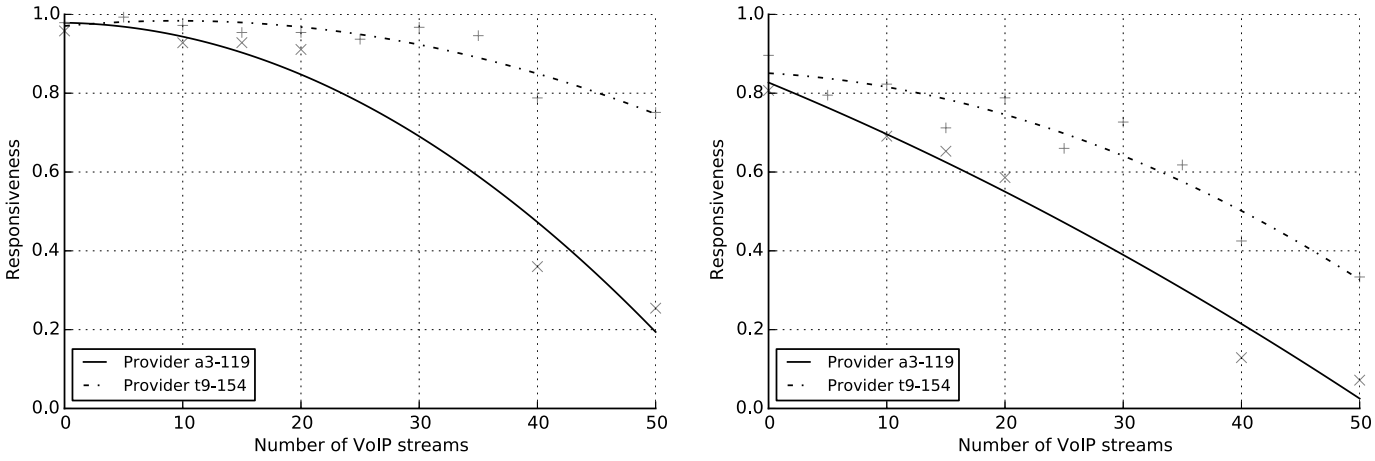
Fig. 7. Responsiveness over time for two different providers under varying load. Figures 7a and 7b show the complete service discovery operation, Figures 7c and 7d individual request/response pairs.

are close to perfect but their responsiveness decreases significantly when conditions deteriorate. Their static retry strategies struggle in unreliable networks. However, packet loss is not the only factor impacting $R(t_D)$, especially at higher loads, the delay of individual packets becomes important, smoothening the steps in the curve, an effect that is also visible in Figure 5d. The analytical models in [5] hint at possible advantages in conditions when SDPs (namely SLP [23] and SSDP [24]) use the more reliable unicast instead of multicast for certain messsages. To validate these results in future work, *ExCovery* is extendable to support these protocols.

The lower graphs of Figure 7 show the responsiveness of individual request/response pairs within an SD operation. Here, $R(t_D)$ denotes the probability that a response to a given request arrived within t_D. The characteristics of the curves confirm the findings on $R(t_D)$ for the complete discovery operation, with $R(1)$ being roughly the same for the corresponding graphs of each provider. The results are included here because they are not based on the events as measured on the discovery layer but instead use raw packet captures done by *ExCovery*. As such, *ExCovery* supports diverse analysis types using the same result database. Packet response times as in Figures 7c and 7d can be used, for example, as lower level input data in analytical responsiveness models as in [5] or when evaluating different protocols that user similar types of packets.

B. Responsiveness over Load

This analysis investigates in more detail how the responsiveness decreases with the level of additional load in the network. For this, fixed deadlines t_D were chosen and $R(t_D)$ calculated for each load level. The results are illustrated in Figure 8 which shows for the two providers *t9-154* and *a3-119* how $R(t_D)$ decreases with increasing load. Figure 8a shows the results for $t_D = 7s$, which corresponds to the time at which the third retry will be triggered, reflecting the responsiveness when doing two



(a) Responsiveness at deadline $t_D = 7s$

(b) Responsiveness at deadline $t_D = 1s$

Fig. 8. Responsiveness over increasing load without (Figure 8b) and after 2 retries (Figure 8a) for two different providers.

retries. Figure 8b shows the results for $t_D = 1s$, just before the first retry would get sent. The curves are the regression function over all data points of a specific provider.

Due to space constraints, only these two results are shown but from all results can be deducted that the slope of the regression curve generally gets steeper with shorter deadlines. This means that the load has a higher impact on $R(t_D)$ the less time there is to complete the discovery operation. One reason for this behavior is the impact of delay on $R(t_D)$. However, this effect is not sufficiently understood and warrants more in-depth evaluation in future research.

C. Responsiveness over Number of Providers

The final analysis targets how SD responsiveness changes with the number of needed service providers. The data used for this analysis were from experiment Series III (see Section V and Table I). One requester and 30 providers were deployed in the dense mesh cloud of building *t9*. This was done to make sure that the distance of providers to the requester had less impact on the results. Instead we wanted to see how discovery performs under varying load conditions. Because there were less nodes for load generation, the data rate per generator pair was increased for a fixed number of 10 node pairs. Environment nodes and providers were distributed randomly in the network. The topology can be seen in Figure 4, which also shows the position of the different types of nodes, requester, provider and environment. This configuration reflects basically any SD scenario where a client tries to enumerate as many service providers of a given service class as possible to select the best n providers according to its requirements.

In the analysis, it was examined for different load levels how the responsiveness decreases with an increasing number of providers needed for successful discovery. Results are depicted in Figure 9. The load levels reflect the combined data rate of all 10 load generation streams. This means that at the highest traffic level, each stream had a data rate of $1.7Mbit/s$. With the given distribution of nodes, this was also the saturation level. Results did not get significantly worse with higher load. At the same time, the network did not get permanently partitioned which means that packets exchanged among the nodes to generate the routing topology managed to get transmitted at a sufficient rate.

The results demonstrate that deploying more services is not generally a valid solution to increase responsiveness in wireless mesh networks. They confirm the preliminary findings in [17] that with high load, hence, increased fault intensity, the responsiveness of service discovery decreases exponentially with the number of services needed. This effect is not visible with low failure intensity. One can see that at very high loads, discovering more than 20 nodes was impossible. Although not shown in Figure 9, this was an almost identical node set over all runs at that load level. This is because some nodes, although not partitioned from the rest of the network, were effectively blocked off SD by environment nodes that were transmitting at very high data rates in their vicinity. Since packets for topology maintenance managed to get transmitted at sufficient rates this hints at profound deficiencies of the *Zeroconf* protocol and maybe even current discovery protocols in general. Further research on the transmission mechanisms (unicast or multicast) and the retry intervals is needed to optimize responsiveness in such scenarios.

VIII. Conclusion and Outlook

The *ExCovery* experiment framework was employed to run several series of experiments to examine the responsiveness of service discovery (SD), the probability to discover service providers within a deadline, even in the presence of faults. Experiments were run in the wireless DES testbed at Freie Universität Berlin. The analysis provides an extensive evaluation of discovery responsiveness with a realistic fault model.

First, it is demonstrated how to use *ExCovery* for experiments in the DES testbed and how the broad range of measure-

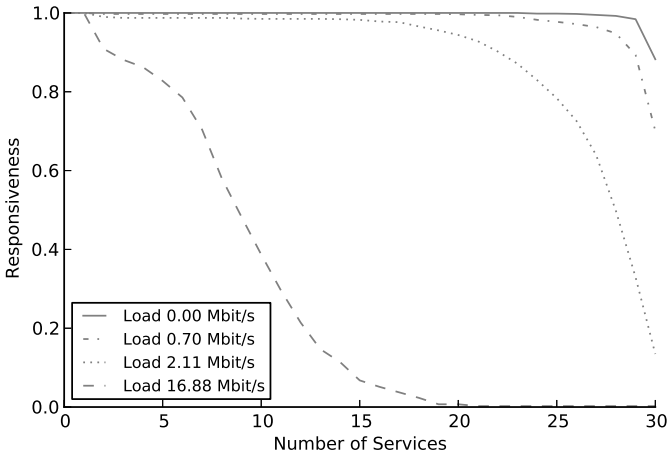


Fig. 9. Responsiveness over number of needed service providers for different load levels.

ments stored during runs enables diverse types of analysis. The core of *ExCovery* is a formal experiment description that supports automated checking, execution and additional features, such as visualization of experiments. *ExCovery* is expected to foster experiment repeatability, comparability and transparency as it offers a unified experiment description, measurement mechanism and storage of results. To facilitate transparency and repeatability, all experiment descriptions and results of this work are made available for interested researchers on request.

Second, an analysis of the long-term testbed behavior and the effects of internal and external faults is carried out. Internal faults contain node crashes or clock drifts, external faults comprise all types of wireless interference or also forced interruptions during experiment execution. It is shown that the effect of these faults is considerable and has to be taken into account when interpreting results.

Third, for several realistic discovery scenarios the responsiveness is evaluated depending on the deadline of the discovery operation, the distance of nodes, the load in the network and the required number of providers to be discovered. Analysis is performed both for the individual discovery packets as well as the complete discovery operation, which includes retries in case packets do not arrive in time. The former allows to infer conclusions for other application protocols which use similar packets and can provide input for existing and future analytical models which use lower network level measurements. As such, the presented results are applicable to various types of optimizations in wireless mesh networks.

REFERENCES

[1] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, 1^{st} ed., ser. The Prentice Hall Service Technology Series from Thomas Erl. Upper Saddle River, NJ, USA: Prentice Hall PTR, August 2005.

[2] M. Malek, "Responsive systems: A marriage between real time and fault tolerance," in *Fault-Tolerant Computing Systems*, ser. Informatik-Fachberichte, M. D. Cin and W. Hohl, Eds. Springer Berlin Heidelberg, 1991, vol. 283, pp. 1–17.

[3] I. Eusgeld, J. Happe, P. Limbourg, M. Rohr, and F. Salfner, *Performability*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 4909, ch. 24, pp. 245–254.

[4] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Computer Networks*, vol. 47, no. 4, pp. 445–487, 2005.

[5] A. Dittrich, B. Lichtblau, R. Rezende, and M. Malek, "Modeling responsiveness of decentralized service discovery in wireless mesh networks," in *MMB & DFT*, ser. Lecture Notes in Computer Science, K. Fischbach and U. R. Krieger, Eds. Springer International Publishing, March 2014, vol. 8376, ch. 7, pp. 88–102.

[6] A. Dittrich, S. Wanja, and M. Malek, "*ExCovery* – a framework for distributed system experiments and a case study of service discovery," in *28th International Parallel & Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW)*. Phoenix, AZ, USA: IEEE Computer Society, May 2014, pp. 1314–1323.

[7] C. E. Dabrowski, K. L. Mills, and S. Quirolgico, "A model-based analysis of first-generation service discovery systems," NIST National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. ADA523379, October 2005.

[8] G. G. Richard, "Service advertisement and discovery: Enabling universal device cooperation," *IEEE Internet Computing*, vol. 4, no. 5, pp. 18–26, Sep-Oct 2000.

[9] M. S. Thompson, "Service discovery in pervasive computing environments," Ph.D. dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA, July 2006.

[10] F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," *IEEE Pervasive Computing*, vol. 4, pp. 81–90, 2005.

[11] W. K. Edwards, "Discovery systems in ubiquitous computing," *IEEE Pervasive Computing*, vol. 5, no. 2, pp. 70–77, 2006.

[12] C. E. Dabrowski and K. L. Mills, "Understanding self-healing in service-discovery systems," in *Workshop on Self-healing systems (WOSS)*. ACM, 2002, pp. 15–20.

[13] C. E. Dabrowski, K. L. Mills, and J. Elder, "Understanding consistency maintenance in service discovery architectures during communication failure," in *3rd International Workshop on Software and Performance (WOSP)*. Rome, Italy: ACM, 2002, pp. 168–178.

[14] ——, "Understanding consistency maintenance in service discovery architectures in response to message loss," in *4th Annual International Workshop on Active Middleware Services*, 2002, pp. 51–60.

[15] I. Eusgeld, F. C. Freiling, and R. Reussner, Eds., *Dependability Metrics, Advanced Lectures*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 4909.

[16] S. Cheshire and D. H. Steinberg, *Zero Configuration Networking - The Definitive Guide*, 1^{st} ed. O'Reilly Media, December 2005.

[17] A. Dittrich and F. Salfner, "Experimental responsiveness evaluation of decentralized service discovery," in *24th International Parallel & Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW)*. Atlanta, GA, USA: IEEE Computer Society, April 2010, pp. 1–7.

[18] S. Wanja, "Experimentation environment for service discovery," Master's thesis, Humboldt-Universität zu Berlin, Berlin, Germany, February 2012.

[19] B. Blywis, M. Günes, F. Juraschek, and O. Hahm, "Properties and topology of the DES-testbed," Freie Universität Berlin, Berlin, Germany, Tech. Rep. TR-B-11-02, March 2011. [Online]. Available: http://edocs.fu-berlin.de/docs/receive/FUDOCS_document_000000009836

[20] S. Cheshire and M. Krochmal, "Multicast DNS," RFC 6762 (Proposed Standard), Internet Engineering Task Force, Feb. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6762.txt

[21] ——, "DNS-Based Service Discovery," RFC 6763 (Proposed Standard), Internet Engineering Task Force, Feb. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6763.txt

[22] J. Postel, "User Datagram Protocol," RFC 768 (Internet Standard), Internet Engineering Task Force, August 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt

[23] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," RFC 2608 (Proposed Standard), Internet Engineering Task Force, Jun. 1999, updated by RFC 3224. [Online]. Available: http://www.ietf.org/rfc/rfc2608.txt

[24] Y. Y. Goland, T. Cai, P. Leach, and Y. Gu, "Simple service discovery protocol/1.0 – operating without an arbiter," Internet-Draft, October 1999. [Online]. Available: ftp://ftp.pwg.org/pub/pwg/ipp/new_SSDP/draft-cai-ssdp-v1-03.txt