Experimental Responsiveness Evaluation of Decentralized Service Discovery

Andreas Dittrich and Felix Salfner
Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
Email: {dittrich|salfner}@informatik.hu-berlin.de

Abstract—**Service discovery is a fundamental concept in service networks. It provides networks with the capability to publish, browse and locate service instances. Service discovery is thus the precondition for a service network to operate correctly and for the services to be available. In the last decade, decentralized service discovery mechanisms have become increasingly popular. Especially in ad-hoc scenarios – such as ad-hoc wireless networks – they are an integral part of auto-configuring service networks. Albeit the fact that auto-configuring networks are increasingly used in application domains where dependability is a major issue, these environments are inherently unreliable. In this paper, we examine the dependability of decentralized service discovery. We simulate service networks that are automatically configured by Zeroconf technologies. Since discovery is a time-critical operation, we evaluate responsiveness – the probability to perform some action on time even in the presence of faults – of domain name system (DNS) based service discovery under influence of packet loss. We show that responsiveness decreases significantly already with moderate packet loss and becomes practicably unacceptable with higher packet loss.**

I. Introduction

Computing and communication infrastructures have been converging rapidly in the last decade. Increased networking capabilities allow mobile and embedded devices to pervade areas of computing that used to have fixed environments and help to make them more flexible and dynamic. A plethora of new devices with different capabilities is entering traditional networks. On the other hand, we experience a ubiquity of connectivity in traditional computing environments. This brings the need for a unified architecture to connect all devices and leverage the services they provide.

A network service in general is an abstract functionality that is provided over the network. It can be leveraged by using the methods of an interface on a specific instance providing that service in the network. Historically, services have existed in computer networks since the first networks were constructed. The difference today is that in service-oriented computing (SOC) protocols and interfaces are being developed and deployed that provide standardized methods for the different layers of service usage. Within every service-oriented domain various layers of service usage need to be defined and configured for successful operation of the service network. These layers incorporate network addressing, service discovery, service description, application and presentation [1].

The first service networks have been centrally administrated. In the last decade new technologies have emerged with methods to automatically configure the various layers of service usage. These methods are an integral part in self-organizing ad-hoc environments where service networks (and the service instances within) are shared by different administrative domains with no central authority. In such environments service auto-configuration provides a significant benefit. In application domains where dependability is of an essence (e.g., in catastrophe management) *dependable* auto-configuring service networks are required. However, self-organizing networks are frequently deployed with wireless technology which is inherently unreliable. It is the goal of this paper to investigate dependability of decentralized service discovery in such networks.

The domains of service-oriented applications are diverse but they share common concepts. One key concept that every service network needs to provide is service discovery. Service discovery involves service providers publishing and service clients searching and locating service instances. If a service cannot be discovered, its existence remains unknown and clients cannot use it —it is unavailable for the client. Thus, dependable service discovery is a prerequisite for dependable service networks. Since discovery is a time-critical operation, to be dependable it needs to provide reliable methods for publishing and discovering service instances within time constraints.

In this paper, we focus on *responsiveness*. Responsiveness is the probability of successful operation within deadlines even in the presence of faults [2]. For service discovery, responsiveness answers the following question:

> For a service client C in a service network N, what is the probability of successfully discovering m out of n total service instances of service type T within time interval t?

To date, no analytical models exist to evaluate responsiveness in auto-configuring networks. Rather than developing an analytical model, this paper provides results from simulation experiments that evaluate responsiveness of decentralized service discovery in unreliable networks. We set up a virtualized testbed and used de-facto standard technologies proposed by the *Zeroconf* working group ([3], [4]) to automatically config-

978-1-4244-6534-7/10/$26.00 ©2010 IEEE

ure the service network. All systems in simulation ran common Linux operating systems and Zeroconf network stacks and are thus representative for systems in real life applications.

The rest of this paper is structured as follows. In Section II we provide references to related work and other approaches to dependability evaluation in auto-configuring service networks. Details on service discovery with a focus on decentralized environments are given in Section III. The experimental setup and scenarios used in this paper are described in Section IV. In Section V we show results for the experimental scenarios and discuss them. The paper is concluded and future work is discussed in Section VI.

II. Related Work

An introduction to fault tolerance in distributed systems is covered in [5]. Responsiveness as a dependability property is best described in [2]. A survey on existing approaches to service discovery systems in ubiquitous computing environments can be found in [6]. It covers all well-known existing approaches to auto-configuring service networks.

Dependability evaluation in auto-configuring service networks has been carried out on various dependability properties, e.g., robustness of service discovery with respect to discovery delay times [7] or cost-effectiveness of network address configuration [8]. The performance and cost-effectiveness of service discovery using local link multicast name resolution (LLMNR) [9] and multicast domain name system (mDNS) [10] with respect to network traffic generation and energy consumption is evaluated in [11]. This paper covers no effects of packet loss.

The research described in [12] is closely related to the topic of this paper. Here, the robustness of existing discovery mechanisms is evaluated under increasing failure intensity. However, responsiveness is not covered in particular. Also, the technologies used in experimental evaluation cover Service Location Protocol (SLP) [13] and Universal Plug-and-Play (UPnP) [14] but not Zeroconf, which is the focus of this paper. A detailed description of Zeroconf can be found in [15].

Contemporary approaches to service networks that demonstrate the need for a responsive service discovery pave the way for many new scenarios. For example, in [16] a model is proposed for survivable services-oriented applications based on numerous redundant services. To show the diversity of those approaches [17] should be mentioned, which describes a resource discovery service for component deployment. Scenarios like this provide a further motivation for this paper.

III. Service Discovery

The objective of service discovery is to enumerate instances of a given service type. Instances are identified by a human-readable name or a universally unique identifier (UUID). Service discovery additionally provides a description of each enumerated instance which holds more specific information for a client to locate the instance in the network, bind to it and use its provided service. It is thus a two-step resolution process that first resolves a service type to a number of instance identifiers and then resolves an instance identifier to an instance description.

The resolution process is not necessarily carried out in two steps on the network. Enumerating and describing service instances is in fact frequently performed within one service discovery request and its response. However, especially in decentralized service networks it is important to distinguish between the two steps.

The amount of description varies depending on the information already included within the service identifier and prior knowledge of service clients. In general, a service client needs at least the network location of a providing instance, for example the network address and port to connect to. On top of that, a description can also provide a communication protocol and information specific for the requested service type.

A. Decentralized service discovery

In centralized service networks, discovery is handled by three different entities: the service client, service provider and service registry. Once a provider connects to the network, it publishes its presence to the registry and provides the registry with a description that is sufficient for a client to connect to the provider. Clients query the registry by sending a unicast discovery request for a service type. The registry answers the queries by sending a unicast discovery response which holds all instances of the requested service type that have registered. Subsequent service communication is done directly between service client and instance.

However, the existence of a (single) designated registry is problematic with respect to scaling, fault tolerance and has to be known a-priori. In decentralized service discovery the resolution of service types and possibly even instance identifiers is done collaboratively by the entire network. The first is mandatory, the latter can also be done in direct communication between service client and provider to reduce network load.

In a decentralized service network, a client that is interested in the functionality given by a specific service type queries the network or a subset of network nodes by sending a discovery request via broadcast or multicast. All nodes that may answer the query respond with a discovery response that is sent to the network via multi- or broadcast or directly to the client via unicast.

Replying by multi- or broadcast is useful to reduce multiple identical responses. Also, it updates information about present service instances on all nodes receiving the response and might suppress subsequent requests by other clients for the same service type. Replies sent by unicast make sense if they contain information that is only valid for the requester.

What messages are being sent by unicast or multi-/broadcast is basically a trade-off between network load and service data distribution and this trade-off is being evaluated differently in common discovery protocols. A sound compromise seems to be to resolve service types via multi- or broadcast and, if necessary, to resolve instance identifiers via unicast.

When doing decentralized discovery, the absence of the registry as an authoritative entity is the main difference to centralized discovery. Every service provider is authoritative when queries resolve to its own instance identifiers or identifier descriptions. This means that in centralized networks a discovery request is successful when the registry responded to a request. In decentralized networks, depending on the needs of the requesting client, in some scenarios all service providers need to respond for successful operation. This introduces delays to discovery. Responsiveness, being the probability to perform some action on time even in the presence of faults, is hence influenced by this delay. Packet loss is a second major impact factor, especially in wireless environments. Another side-effect of packet loss is that packet retransmissions result in an increased network load.

Although security issues are not covered in this paper, it should be mentioned that the lack of a single authority in decentralized service networks makes administration and control of the instance identifier space more difficult. Additional measures need to be implemented to guarantee the trustworthiness of service discovery.

B. Zeroconf service discovery

So far, although the various existing approaches mentioned in Section II follow the same general concept for service discovery (described in Section III-A) they remain technically incompatible. Hence, for this paper, one architecture for service discovery had to be chosen. Because of its low overhead, versatility and compatibility with existing, widely deployed services [1], Zeroconf was chosen as a protocol stack for the service network in the following experiments.

The Zeroconf stack works on top of the Internet Protocol (IP) and has a very low overhead compared to other service network stacks. It still provides complete auto-configuration of all layers up to service discovery. A thorough evaluation of service network stacks and their overhead can be found in [1]. In recent years, Zeroconf became increasingly popular and by today is provided by numerous network services like printing, file or screen sharing and others. Linux and Macintosh operating systems are delivered with Zeroconf technology enabled by default and implementations exist for virtually every operating system.

The details of the Zeroconf service network stack are described in detail in [18]. In short, Zeroconf handles the three lower layers of service networks [14] and uses specific protocols to automatically configure them and to provide their functionality.

1) *Addressing* – To take part in the network, every node needs a unique network address. The protocol used for auto-configuration is the ubiquitous Automatic Private IP Addressing (APIPA) which is better known as AutoIP and standardized in [19]. AutoIP introduces special types of Address Resolution Protocol (ARP) [20] messages called ARP probes.
2) *Name resolution* – Service identifiers need to be resolved to network addresses for clients to be able to connect and bind to the services. Zeroconf uses a multicast version of the domain name system (DNS) [21] called mDNS [10]. This protocol can configure names for service instances and resolve them to network addresses.
3) *Discovery* – To reduce the number of different protocols, Zeroconf uses a DNS-based service discovery mechanism (DNS-SD) [22]. All service instance identifiers as well as service types are handled as DNS names and as such can be resolved by mDNS on the lower layer. DNS-SD is merely an extension to DNS that provides additional record types for service discovery.
4) *Description* – The description needed to connect a service includes the network address and port. This functionality is also provided by DNS-SD. DNS-SD can provide a more complete description of a service instance although this is not of importance within the context of this paper.

In Zeroconf, most discovery requests and responses are sent via multicast to ensure a high distribution of the data. A single service discovery, as carried out in the following experiments consists of a single multicast request with multiple retries $1, \ldots, i$. The waiting time before a retry is 2^{i-1} seconds. During that time the service client constantly waits for responses from service providers. Upon arrival of responses, it includes these known answers in subsequent requests to suppress duplicate responses.

TABLE I
TECHNICAL SPECIFICATION OF SIMULATION SYSTEMS

	Xen host	Zeroconf nodes
Processor type	Intel® Xeon® X5365	n/a
Processor frequency	3000 Mhz	n/a
Cores	$2 * 4$	1
Memory	16 GB	48 MB
Operating system	Linux openSUSE 11.0	Linux Debian 5.0.3
Architecture	x86_64	x86_64
Kernel version	2.6.25.20-0.5-xen	2.6.26-2-xen-amd64
Xen version	3.2.1_16881_04-4.3	n/a
Avahi version	n/a	0.6.23-3lenny1

IV. Experimental setup

All experiments were carried out in a virtual testbed on a machine running the Xen hypervisor [23]. Service nodes were run as unprivileged guest domains with the same base system to boot from. Technical specifications of the systems are listed in Table I.

The service network for the nodes was realized by connecting them to a virtual network bridge on the Xen host. This network was solely used for service discovery communication. The topology reflects a fully connected, single-hop network where all packets sent among nodes pass the bridge between them. On this bridge, packet loss was realized by randomly dropping packets at a given rate. The rate was the same for every packet. No additional faults were injected on the bridge.

The service nodes were running a minimal installation of the Debian Linux operating system [24]. At boot time, they were only running the base system and Zeroconf daemons to configure the service network and do service discovery. On a dedicated client node, a secure shell (SSH) daemon was additionally run for remote discovery execution. This daemon ran on a second interface which was not connected to the virtual bridge so that traffic on this interface did not interfere with service network traffic. Memory requirements for the guest systems were very low (c.f. Table I).

To automatically configure the service network, software developed by the Avahi project [25] was used. Avahi is an implementation of the protocols recommended by the Zeroconf working group for automatic configuration of IP service networks. An AutoIP [19] daemon set a unique IP address within the 169.254.0.0/16 subnet and a mDNS [10] daemon handled service name resolution for DNS-based service discovery [22]. Due to the fact that Avahi was used for auto-configuration, all nodes could run from copies of the same disk image and no manual administration was necessary after booting.

Discovery requests were run from a single dedicated discovering node – or service client. All other nodes acted as service providers responding to discovery requests. Discovery times were measured on the client directly before the request was sent and directly after responses were received to measure user-perceived responsiveness. Thus, time synchronization between nodes in the service network was not necessary.

No nodes joined or left the network so no reconfiguration of the network layers occurred during measurements which would interfere with discovery operation. Discovery of a service was considered successful when a certain percentage of instances had been discovered. Discoveries were aborted (considered failed) after 20 seconds waiting time. This value was chosen because in Zeroconf, the time between retries doubles after each retry to reduce network load. So after 15 seconds we have reached a total number of five discovery requests and the next one would be sent after 31 seconds. Waiting time was extended by additional 5 seconds to ensure delivery of all responses.

As mentioned in Section III, after service discovery a client should have enough information to contact a service instance. Hence discovery in our case meant resolving the IP address and port for every service instance.

A. *Justification of experimental setup*

The chosen network model is a simplified version of a fully connected, wired Ethernet network. On those networks, packet loss usually is no issue. We are aware that unreliable networks – especially wireless networks – have a complex behavior and most of the faults occurring in those networks (e.g.: bursts of packet loss, delay, jitter) have a strong functional dependency. However, we wanted to control the unreliability of the network as much as possible and, therefore, included random packet loss at a given rate, independent of the traffic occurring on the link and whether the preceding packet was lost. Our results hence yield insight on average behavior. Packet delivery time, which is in the order of milliseconds, can be neglected compared to the time between discovery retries, which is in the order of seconds.

TABLE II
SUMMARY OF SIMULATION SCENARIO PARAMETERS

Scenario	1	2	3
Number of service providers	1	1,20,50	1...50
Maximum discovery time	10s	0...20s	20s
Packet loss (%)	0...90	20,40	10,20,30,40
Observed discovery operations	10,000	6,000	24,000

Packet loss was identified as the highest-impact fault on service discovery. So, in these experiments we focused on packet loss. Since we assume a mean packet loss service discovery responsiveness will most probably be worse in networks with more complex fault characteristics. Our analysis hence provides an upper bound.

Since service instances are discovered on a single node, in our network model it is sufficient to drop packets only on links directly attached to it. In fact, packet dropping was done independently in forwarding direction of every individual interface connected to the virtual bridge. This setup causes multi- or broadcasts to be potentially lost on every interface they were transmitted to in order to prevent all-or-nothing behavior where a broadcast is either received by all or by no host at all.

B. *Simulation scenarios*

To judge the applicability of decentralized discovery mechanisms, three scenarios were chosen that reflect common use cases of service discovery. The parameters of the three scenarios are summarized in Table II.

1) Scenario 1: Single service discovery: The first goal is to measure the responsiveness of single service discovery. The service network consists of one client and one provider. The client has lax requirements and is allowed to wait up to ten seconds for a positive response. This is a common scenario for service discovery and can be considered as the baseline. Only one answer needs to be received and there is enough time to wait for it. To see how results vary in unreliable networks measurements have been taken with packet loss rates ranging from zero to 90 percent.

2) Scenario 2: Timely service discovery: Most service networks are populated with multiple instances of the same service type. The client needs to discover as many instances as possible and will then choose one that optimally fits its requirements. The faster discovery is successful, the better. In this scenario full coverage is required. There is one service client, n service providers ($n = 1, 10, 50$) and discovery is successful if all n provided service instances have been discovered. The goal is to measure how responsiveness increases with time. The faster we reach a high value the better. Measurements are carried out with a packet loss rate of 20 and 40 percent.

3) Scenario 3: Multiple service discovery: When dealing with packet loss in the network, in general more instances of the same service type should increase responsiveness of service discovery when looking for a fixed number of instances. In the case of fixed coverage – which means the ratio of

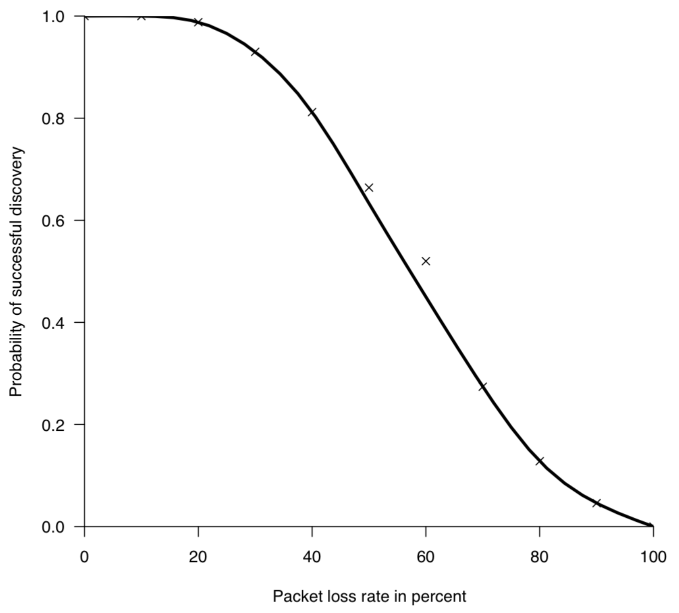

Fig. 1. Responsiveness of single service discovery with 10s deadline as a function of a packet loss rate

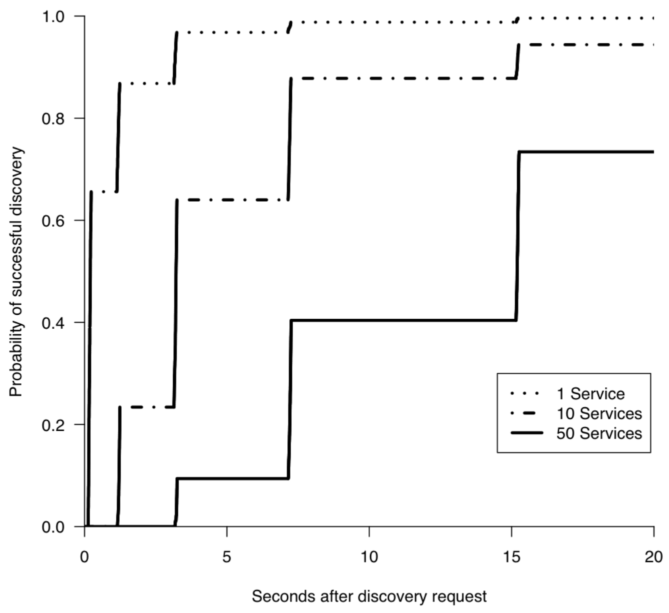

Fig. 2. Responsiveness of service discovery with 20% packet loss

discovered services needed for successful operation remains constant – this is not necessarily the case. In this scenario we investigate the difference in responsiveness when discovering n out of n service instances (full coverage) with n growing up to 50. Measurements are carried out with a packet loss rate of $10, 20, 30$ and 40 percent.

V. Experimental Results

In this section the results for the three scenarios described in Section IV-B are presented and discussed. The total number of discovery requests sent in simulation was 60,000. The number of accumulated discovery responses reached 599,046. Requests were split equally over loss rates ranging from 0% to 90% in 10% steps and over the number of provided service instances equal to $1, 2, 5, 10, 20$ and 50. The number of investigated discovery operations for each scenario are shown in Table II. The raw data of all these measurements has been uploaded to the AMBER Data Repository [26] and interested researchers are invited to perform further investigations.

A. Scenario results: Single service discovery

Discovery of single service instances within ten seconds proved to be reasonably responsive in networks with low packet loss. As illustrated in Figure 1, even with 30% packet loss, the responsiveness of single service discovery was well above 0.9 which is acceptable in this scenario. The service client in this scenario has lax requirements from which can be concluded that any responsiveness higher than 0.9 should suffice. With higher packet loss rates responsiveness decreases rapidly, dropping to 0.63 at 50 percent packet loss. In real world networks with more complex fault behavior – for example wireless networks – the results are probably worse, which is not promising for other scenarios with stricter requirements.

However, it is valid to conclude that with up to 20 percent packet loss and no additional worsening effects, single service discovery is sufficiently responsive.

B. Scenario results: Timely service discovery

The analysis of the data for this scenario is illustrated in Figures 2 and 3. With 20 percent packet loss the line for discovering a single service is close to a responsiveness of one. This follows the conclusions from single service discovery at 10 seconds waiting time (Figure 1). In fact, discovering a single service reaches a responsiveness of almost 0.9 already after the first retry in this scenario. With the given techniques and low packet loss, current Zeroconf service discovery mechanisms work sufficiently well when discovering single service instances.

However, Figure 2 shows that responsiveness decreases rapidly if more services need to be discovered. Both discovering 10 and 50 service instances reaches a somewhat acceptable (≈ 0.95 and ≈ 0.7) responsiveness at the end of the timescale after four retries with 20 percent packet loss. This corresponds to scenarios where there is enough time to wait for discovery responses and, especially in the case of 50 instances, low responsiveness is acceptable. If short response times are needed, we need a high responsiveness in shortest time and it can be seen in Figure 2 that both curves require a long discovery time to reach reasonable responsiveness.

The characteristic steps in the curves mark the request retry events. Whenever a discovery request is sent, It can be ex-

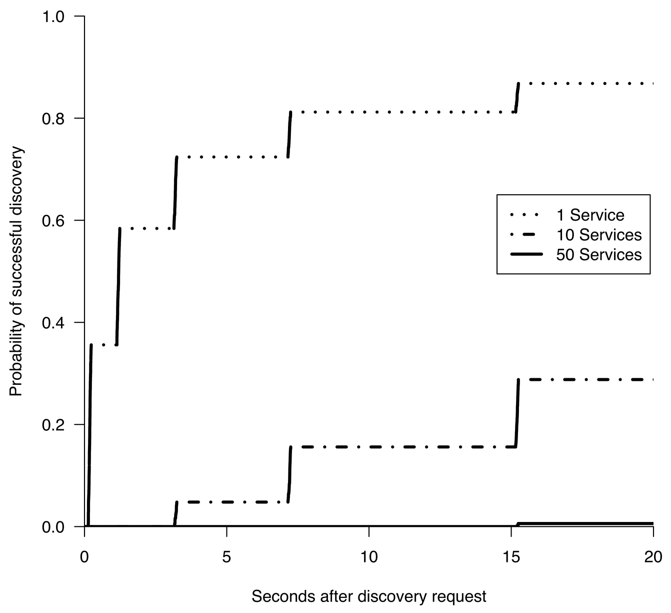

Fig. 3. Responsiveness of service discovery with 40% packet loss

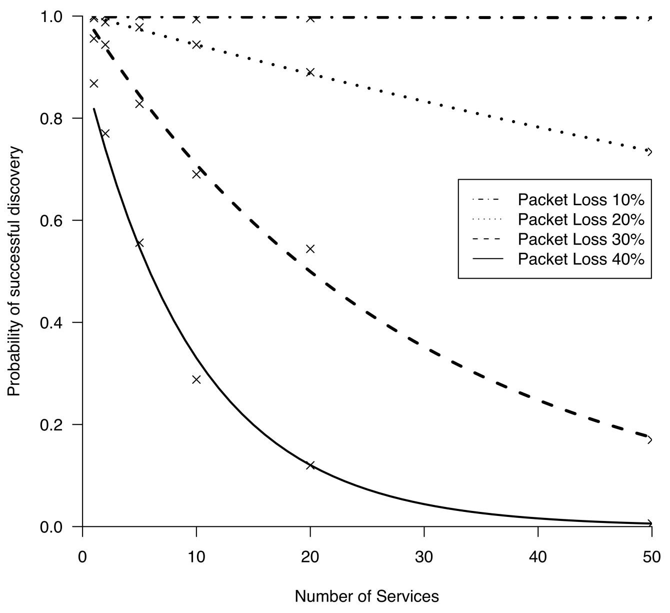

Fig. 4. Responsiveness of Zeroconf service discovery after four retries (within 20s)

pected that a shorter time between retries should significantly increase responsiveness (at a given time t). However, this would also increase network load and might have other adverse effects on the network especially when dealing with a high number of service instances. Without a more realistic network model and a sound cost function for packet transmission it seems difficult to determine an optimal threshold.

With 40 percent packet loss, discovery of multiple services almost completely stops working. Figure 3 illustrates this. There is a chance of less than one third to find all services instances in time when there are ten instances in the network. Discovering 50 service instances is practically impossible. In contrast, discovering single services remains partly usable so doubling the packet loss rate did not have a comparable dramatic effect on responsiveness as with multiple services. This dramatic decrease in responsiveness when discovering more service instances will be investigated in the next scenario.

C. Scenario results: Multiple service discovery

In this scenario we investigate responsiveness for discovering a fixed percentage of service instances when the number of service instances increases. Full coverage is required so 100% of service instances need to be discovered for successful operation. All instances belong to the same service type – a single discovery request should discover all instances if no packets are lost. It could already be deducted from the last scenario that service discovery responsiveness decreases dramatically when discovering ten or more services in lossy networks. Figure 4 shows an articulate explanation for this behavior.

With low packet loss rates responsiveness decreases linearly with the number of nodes in the network. This is the reason why with 20 percent packet loss we still have a pretty high probability of almost 80 percent to discover 50 service instances in time (see Figure 2).

If more packets get dropped this worsens rapidly to an exponential decrease of responsiveness with the number of nodes in the network. In practice, service discovery with high coverage becomes unusable when dealing with a higher number of services. Current Zeroconf discovery methods are usually not operating in such scenarios.

We can conclude that with given discovery methods, we reach a threshold in unreliable networks when we try to discover an increasing number of services. The exponential decline of responsiveness seen in the last scenario illustrates this threshold.

Responsiveness can be improved if a low coverage is acceptable. This can, for example, be the case if redundant service instances are introduced to the network and, e.g., a fixed number of service instances needs to be discovered (regardless of the total number of instances in the network). However, following the conclusions from the multiple service discovery scenario, this assumption is only valid in networks with low packet loss. In less reliable networks redundant service instances could in fact worsen responsiveness.

Applying the improvements proposed in [11] to multicast DNS could also improve responsiveness in the described three scenarios. However, further investigations are needed since not all improvements are compliant with the existing standards. Changes such as accepting non-authoritative discovery responses from any nodes in the service network might improve discovery responsiveness but reduce the reliability of service discovery as a whole.

VI. Conclusions

Service discovery is an elementary concept in service networks. It provides networks with the capability to publish, enumerate and locate service instances. A working service discovery is thus the precondition for a service network to operate correctly and for the services to be available. If a client cannot discover the presence of a service in the network, it can never be available to the client.

In this paper, we examined dependability aspects of decentralized service discovery concepts in unreliable networks. We simulated a service network that was automatically configured by Zeroconf technologies, which are frequently used in real life applications.

Since discovery is a time-critical operation, we focused on responsiveness which is the probability of successful operation within deadlines. We evaluated responsiveness of domain name system (DNS) based service discovery under influence of packet loss and with up to 50 service instances.

The empirical results show that the responsiveness of the used service discovery mechanisms decreases dramatically with moderate packet loss of around 20%. It decreases further the more service instances need to be discovered. At high packet loss rates the decrease becomes exponential with the number of nodes such that discovery becomes practically impossible.

In summary, with self-configuring networks entering dependability-critical domains, our experimental evaluation has shown that distributed service discovery has to be used with caution, especially in wireless scenarios where packet loss cannot be neglected. Finally, with increasing demand for real-time systems, the responsiveness optimization will become the main issue.

References

[1] A. Dittrich and J. Kowal, "Architektur für selbstkonfigurierende Dienste auf Basis stark ressourcenbeschränkter eingebetteter Systeme," Master Thesis, Institut für Informatik, Humboldt-Universität zu Berlin, July 2008.

[2] M. Malek, "Responsive systems: The challenge for the nineties," *Microprocessing and Microprogramming*, vol. 30, pp. 9–16, 1990.

[3] A. Williams, "Requirements for automatic configuration of IP hosts," Draft, September 2002, draft-ietf-zeroconf-reqts-12.

[4] E. Guttman, "Zeroconf host profile applicability statement," Draft, July 2001, draft-ietf-zeroconf-host-prof-01.

[5] P. Jalote, *Fault Tolerance in Distributed Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

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

[7] C.-S. Oh, Y.-B. Ko, and J.-H. Kim, "A hybrid service discovery for improving robustness in mobile ad hoc networks," in *Proceedings of the IEEE 2004 International Conference on Dependable Systems and Networks (DSN 2004)*. IEEE Computer Society, June 2004, Short Paper.

[8] H. Bohnenkamp, P. van der Stok, H. Hermanns, and F. Vaandrager, "Cost-optimization of the ipv4 zeroconf protocol," *Dependable Systems and Networks, International Conference on*, vol. 0, p. 531, 2003.

[9] B. Aboba, D. Thaler, and L. Esibov, "Link-local Multicast Name Resolution (LLMNR)," RFC 4795 (Informational), Internet Engineering Task Force, Jan. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4795.txt

[10] S. Cheshire and M. Krochmal, "Multicast DNS," Internet-Draft, 2006. [Online]. Available: http://tools.ietf.org/html/draft-cheshire-dnsext-multicastdns-06.txt

[11] C. Campo and C. García-Rubio, "DNS-based service discovery in ad hoc networks: Evaluation and improvements," in *PWC*, 2006, pp. 111–122.

[12] C. Dabrowski, K. Mills, and S. Quirolgico, "Understanding failure response in service discovery systems," *J. Syst. Softw.*, vol. 80, no. 6, pp. 896–917, 2007.

[13] 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

[14] "Universal plug and play device architecture 1.1," UPnP Forum, Tech. Rep., 10 2008.

[15] E. Guttman, "Autoconfiguration for ip networking: Enabling local communication," *IEEE Internet Computing*, vol. 5, no. 3, pp. 81–86, 2001.

[16] A. Dittrich, J. Kowal, and M. Malek, "Designing survivable services from independent components with basic functionality," in *International Workshop on Dependable Network Computing and Mobile Systems (DNCMS 08)*, Naples, Italy, October 2008, pp. 33–38.

[17] K. Pauls and R. S. Hall, "Eureka - a resource discovery service for component deployment," in *Component Deployment*, 2004, pp. 159–174.

[18] S. Cheshire and D. H. Steinberg, *Zero Configuration Networking – The Definitive Guide*, 1st ed. ORéilly Media, Inc., December 2005.

[19] S. Cheshire, B. Aboba, and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," RFC 3927 (Proposed Standard), Internet Engineering Task Force, May 2005. [Online]. Available: http://www.ietf.org/rfc/rfc3927.txt

[20] D. Plummer, "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826 (Standard), Internet Engineering Task Force, Nov. 1982, updated by RFCs 5227, 5494. [Online]. Available: http://www.ietf.org/rfc/rfc826.txt

[21] P. Mockapetris, "Domain names - implementation and specification," RFC 1035 (Standard), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[22] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," Internet-Draft, Aug. 2006. [Online]. Available: http://tools.ietf.org/html/draft-cheshire-dnsext-dns-sd-04.txt

[23] (2009) The xen® hypervisor. [Online]. Available: http://xen.org/

[24] Debian – the universal operating system. [Online]. Available: http://www.debian.org/

[25] The avahi project. [Online]. Available: http://avahi.org/

[26] J. D. Marco Vieira, Naaliel Mendes and H. Madeira, "The AMBER data repository," University of Coimbra, Coimbra, Portugal, Tech. Rep., May 2008.