Service Availability and Discovery Responsiveness

A User-Perceived View on Service Dependability

Dissertation

zur Erlangung des akademischen Grades
Dr. rer. nat. im Fach Informatik

eingereicht an der Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin am 9. Dezember 2014

von Herrn Dipl.-Inf. Andreas Dittrich

Präsident der Humboldt-Universität zu Berlin
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät
Prof. Dr. Elmar Kulke

Gutachter:

1. Prof. Dr. Miroslaw Malek
2. Prof. Dr. Alexander Reinefeld
3. Prof. Dr. Jörg Kaiser

Tag der mündlichen Prüfung: 20. März 2015

To Tante Maria.

Preface

Almost 2400 years ago Plato wrote the allegory of the cave, in which Plato's brother Glaucon and his mentor Socrates discuss the difference between the form and the sensation. Socrates describes a group of people who have lived their entire live inside a cave. The only knowledge about the outside world comes from shadows projected to the walls of the cave of people and objects passing by outside. These shadows are mere sensations of the true form outside the cave. Nonetheless, for the people inside the cave there is but this one perception which constitutes their reality.

In computer networks, every node has its own view on the network and the services therein, which is defined by the network components that allow the node to communicate with the rest of the network. While a global perspective on the network, the awareness of the form in Plato's words, is generally desirable it might be of limited value to a node with only its own perspective. When improving the quality of service provision it is thus important to keep the node-specific perspective in mind. Improving as many as possible node-specific perspectives will in turn improve the overall quality of service provision. This work focuses on the evaluation of node or user-perceived views on service quality as reflected in two different dependability properties, availability and responsiveness.

Berlin, March 2015 *Andreas Dittrich*

Acknowledgements

First and foremost, I would like to thank my doctoral advisor Miroslaw Malek for his continuous support and inspiration. He made me look further so often when I thought I had seen it all. I would also like to thank Felix Salfner for all the technical advice he was able to give me and for making me believe that I can actually finish this work. I am very grateful to Katinka Wolter for encouraging me during the last year and for demonstrating me again and again the beauty of math. This work was in parts carried out in the graduate school *METRIK* and I am thankful to have been able to present my work in front of experts from so many different fields. This helped me to strengthen and to focus. I am especially grateful to Joachim Fischer who always had an open ear from the moment I started this work. Of course I cannot name all my fellow colleagues I would like to thank for the many fruitful discussions during the work on this thesis.

Last but not least, I want to thank my family, my parents, my brothers and sister for always being there when I needed strength and for having the patience when I could not be there. Ce l'ho fatta Manu! Words cannot tell how much your support means to me. I am most grateful to Tante Maria, who made me believe that there is sunshine in every moment of our lives. She gave me the inspiration to start and the courage to continue.

Abstract

Services play an increasingly important role in modern networks, ranging from web service provision of global businesses to the Internet of Things. Dependability of service provision is thus one of the primary goals. For successful provision, a service first needs to be discovered and connected to by a client. Then, during actual service usage, it needs to perform according to the requirements of the client. Since providers and clients are part of a connecting Information and Communications Technology (ICT) infrastructure, service dependability varies with the position of actors as the ICT devices needed for service provision change. Service dependability models need to incorporate these user-perceived perspectives.

We present two approaches to quantify user-perceived service dependability. The first is a model-driven approach to calculate instantaneous service availability. Using input models of the service, the infrastructure and a mapping between the two to describe actors of service communication, availability models are automatically created by a series of model to model transformations. The feasibility of the approach is demonstrated using exemplary services in the network of University of Lugano, Switzerland. The second approach aims at the responsiveness of the service discovery layer, the probability to find service instances within a deadline even in the presence of faults, and is the main part of this thesis. We present a hierarchy of stochastic models to calculate user-perceived responsiveness based on monitoring data from the routing layer. Extensive series of experiments have been run on the Distributed Embedded Systems (DES) wireless testbed at Freie Universität Berlin. They serve both to demonstrate the shortcomings of current discovery protocols in modern dynamic networks and to validate the presented stochastic models.

Both approaches demonstrate that the dependability of service provision indeed differs considerably depending on the position of service clients and providers, even in highly reliable wired networks. The two approaches enable optimization of service networks with respect to known or predicted usage patterns. Furthermore, they anticipate novel service dependability models which combine service discovery, timeliness, placement and usage, areas that until now have been treated to a large extent separately.

Zusammenfassung

Die Bedeutung von Diensten in modernen Netzwerken nimmt stetig zu. Verlässliche Dienstbereitstellung ist daher eines der wichtigsten Ziele. Um erfolgreich einen Dienst erbringen zu können, muss dieser zuerst von einem Nutzer gefunden und verbunden werden. Während der Nutzung muss der Dienst eine Leistung entsprechend der Anforderungen des Nutzers erbringen. Da Anbieter und Nutzer Teil einer Informations und Kommunikationstechnologie (IKT) Infrastruktur sind, wird die Verlässlichkeit der Dienste je nach Position der Aktoren variieren, so wie sich die für die Bereitstellung nötigen IKT Geräte ändern. Dienstverlässlichkeitsmodelle sollten diese nutzerspezifischen Perspektiven berücksichtigen.

Wir stellen zwei Ansätze zur Quantifizierung nutzerspezifischer Dienstverlässlichkeit vor. Der erste, modellgetriebene Ansatz berechnet momentane Dienstverfügbarkeit. Aus Modellen des Dienstes, der Infrastruktur und einer Abbildung zwischen den beiden, welche die Aktoren der Dienstkommunikation beschreibt, werden durch eine Serie von Modelltransformationen automatisiert Verfügbarkeitsmodelle generiert. Die Realisierbarkeit des Ansatzes wird beispielhaft gezeigt anhand von Diensten im Netzwerk der Universität Lugano, Schweiz. Der zweite Ansatz behandelt die Responsivität der Dienstfindung, die Wahrscheinlichkeit innerhalb einer Frist Dienstinstanzen zu finden, unter der Annahme von Fehlern. Dies stellt den Hauptteil dieser Arbeit dar. Eine Hierarchie stochastischer Modelle wird vorgestellt, die nutzerspezifische Responsivität auf Basis von Messdaten der Routingebene berechnet. Umfangreiche Experimente wurden im Distributed Embedded Systems (DES) Funktestbett der Freien Universität Berlin durchgefürt. Diese zeigen die Probleme aktueller Dienstfindungsprotokolle in modernen, dynamischen Netzwerken. Gleichzeitig dienen sie der Validierung der vorgestellten Modelle.

Beide Ansätze zeigen, daß die Verlässlichkeit der Dienstbereitstellung in der Tat deutlich mit der Position von Nutzern und Anbietern variiert, sogar in hochverfügbaren Kabelnetzwerken. Die Ansätze ermöglichen die Optimierung von Dienstnetzwerken anhand bekannter oder erwarteter Nutzungsmuster. Zudem antizipieren sie neuartige Verlässlichkeitsmodelle, welche Dienstfindung, zeitige Bereitstellung, Platzierung und Nutzung kombinieren; Gebiete, die bisher im Allgemeinen getrennt behandelt wurden.

Contents

Acronyms

AMF	Availability Management Framework
ARP	Address Resolution Protocol
AutoIP	Automatic Private IP Addressing
BFS	Breadth-First Search
BPMN	Business Process Model and Notation
CDF	Cumulative Distribution Function
CMDB	Configuration Management Database
DCF	Distributed Coordination Function
DES	Distributed Embedded Systems
DNS	Domain Name System
DNS-SD	DNS-Based Service Discovery
ECDF	Empirical Cumulative Distribution Function
EE	Experimentation Environment
EP	Experiment Process
ETX	Expected Transmission Count
EU	Experimental Unit
FT	Fault Tree
FPP	Flooding Path Probability
FUB	Freie Universität Berlin
FzT	Fuzz Tree
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IP	Internet Protocol
JSON	JavaScript Object Notation
LLMNR	Local Link Multicast Name Resolution
MDD	Model-Driven Development
mDNS	Multicast DNS
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
MTU	Maximum Transmission Unit
NAS	Network-Attached Storage

NWB	Network-wide Broadcast
OFAT	One aFter AnoTher
OLSR	Open Link-State Routing
OU	Observational Unit
PDF	Probability Density Function
PBFS	Probabilistic Breadth-First Search
QUDG	Quasi Unit Disk Graph
RBD	Reliability Block Diagram
RMSE	Root Mean Squared Error
RPC	Remote Procedure Call
SD	Service Discovery
SDP	Service Discovery Protocol
SLP	Service Location Protocol
SCM	Service Cache Manager
SM	Service Manager
SSH	Secure SHell
SU	Service User
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
SOSE	Service-Oriented System Engineering
SSDP	Simple Service Discovery Protocol
SSH	Secure Shell
UDG	Unit Disk Graph
UDP	User Datagram Protocol
UML	Unified Modeling Language
UPSIM	User-Perceived Service Infrastructure Model
UPSAM	User-Perceived Service Availability Model
UPnP	Universal Plug-and-Play
USI	University of Lugano
UUID	Universally Unique IDentifier
VoIP	Voice-over-IP
WMHN	Wireless Multi-Hop Network
WMN	Wireless Mesh Network
WWW	World-Wide Web
XML	Extensible Markup Language
XML-RPC	Extensible Markup Language Remote Procedure Call
Zeroconf	Zero-configuration networking

Part I
Introduction

Part I of this thesis covers preliminaries to the topics of user-perceived dependability evaluation in service networks. Chapter 1 gives a concise introduction to the topic and states the scientific problems. The main contributions are listed.

Chapter 2 provides an extensive collection of related work. Background information about the main scientific areas is connected to the state-of-the-art as reflected in current research works.

The final Chapter 3 describes in detail the experiment framework *ExCovery*, which was developed during the work on this thesis to run the necessary experiments in service networks. *ExCovery* defines concepts for describing, measuring and storing experiment series in distributed systems.

Chapter 1
Introduction, Motivation, Problem Statement and Main Contributions

Abstract A brief introduction is given to dependability evaluation in service networks. Emphasis is put on the user-perceived scope as defined by the location of actor nodes in the network and the time of evaluation. The scientific problems of evaluating user-perceived service availability and user-perceived responsiveness of service discovery are stated. The chapter concludes with a summary of the approach to a solution of these problems and a list of the main contributions of this work.

1.1 Introduction

Information processing and communication have been converging rapidly in the last decades. Cheaper production cost of *Information and Communications Technology* (ICT) combined with increasing networking and computing capabilities have allowed mobile and embedded devices to become ubiquitous. They have entered traditional, static computing environments and turned upside-down the governing paradigms. On one hand, the introduced dynamics and flexibility open a lot of possibilities related to social networking and cloud computing, to name just a few. On the other, they create new challenges for dependability evaluation, which needs to deal with manifold devices that differ both in their functional capabilities and in their non-functional properties such as availability, responsiveness or performance. Not only are the network nodes heterogeneous but there is a high variability in the network link technologies and qualities. Furthermore, novel techniques are needed to quickly assess the state of these dynamic networks at any point in time. An illustration of such a network is Figure 1.1.

Service-Oriented Architecture (SOA) [81] proposes services as the basic building blocks of system design. The roots of SOA lie in the classic *World-Wide Web* with few, powerful instances providing services to many clients. Modern service networks may cover many diverse scenarios: Ad-hoc, decentralized *Wireless Multi-Hop Networks* (WMHN) or the so-called *Internet of Things* [91, 21].

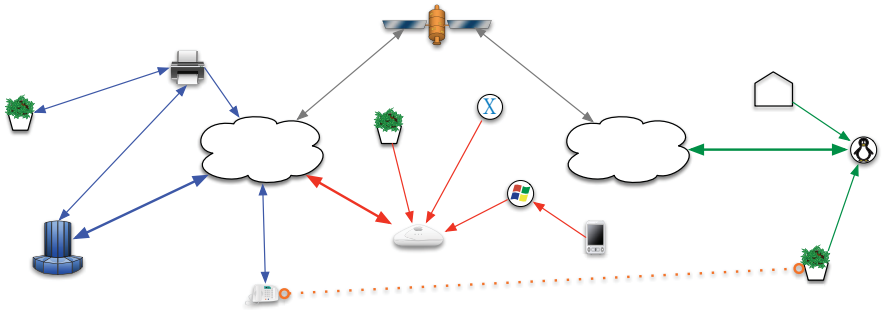


Fig. 1.1 An illustration of a heterogeneous network with diverse devices ranging from mainframes to embedded devices.

Meeting dependability requirements is crucial for successful service provision. It can be observed that information about the overall network dependability is often not sufficient to assess service dependability for any client within the network. This is due to non-functional properties like service dependability being highly dependent on the properties of the underlying ICT infrastructure. Moreover, network topologies change, components are upgraded or undergo maintenance after failure, services are migrated and so forth. These dynamics represent one of the main challenges of service dependability evaluation, especially during run-time, when changes need to be timely considered in the dependability models. Although services are usually well-defined within business processes, assessing dependability of the various processes in service networks remains uncertain. This is especially true for the *user-perceived* dependability of processes between a specific pair service requester and provider as every pair can utilize different ICT components for communication. The underlying infrastructure varies according to the position of the requesting client – represented by a person or even an ICT component – and the concrete providing service instance. Evaluation of user-perceived dependability should employ a model of the ICT infrastructure where service properties are linked to component properties.

A Case for Service Discovery

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. But since first generation service networks usually had static service compositions, discovery was only needed at deployment time, if at all. And while manual discovery is omnipresent during web browsing when clients use web search engines, this process is usually not seen as part of the actual service usage. But web service providers know the value of successful discovery and use considerable resources on search engine

optimization to improve their search ranking and thus make it more probable to be discovered by interested clients.

In the last decade, decentralized SD has become increasingly popular. Especially in ad-hoc scenarios – such as wireless mesh networks – SD is an integral part of auto-configuring service networks. Albeit the fact that auto-configuring networks are more and more used in application domains where dependability is a major issue, these environments are inherently unreliable. Because of a high variability of link quality in such networks, the dependability of SD is expected to change significantly with the positions of requester and provider.

Just as service usage, SD is an inherently time-critical process. A client using a service expects it to perform until a required deadline. The longer discovery takes to find the service, the less time the service has to perform. For modern network scenarios, where SD is expected to become ever more important, *Service Discovery Protocols* (SDPs) need to be able to reduce the time to discovery. They additionally need to be able to meet this discovery deadline with a high probability. This property is called *responsiveness*, the probability to perform within a deadline, even in the presence of faults, as defined in [144]. For SD responsiveness just as for service availability, evaluation needs to take into account the user-perceived scope: the location of the acting nodes and the time of discovery.

General Scope of the Work

This thesis targets two areas of user-perceived dependability evaluation in service networks. The first part of this work focuses on user-perceived service availability as in Definition 1.1.

Definition 1.1. Given an ICT infrastructure N with a set of providing service instances P and a set of service clients C, the user-perceived availability is the probability $A_{P,c}$ for a service provided by P to perform its required function when requested from a specific client $c \in C$. Service P is assumed to perform as required if all ICT components necessary for communication between P and c are available.

Since availability is an intrinsic property of the ICT layer, a mechanism is needed to reflect this characteristic in a service availability model. Following Definition 1.1, a service is available if all ICT components needed for interface connection and communication during service provision are available. A model that describes these ICT components can be called a *User-Perceived Service Infrastructure Model* (UPSIM) as in Definition 1.2.

Definition 1.2. Given an ICT infrastructure N with a set of providing service instances P and a set of service clients C, a providing service instance $p \in P$ and a service client $c \in C$ with $p, c \in N$, a user-perceived service infrastructure model $N_{upsim} \subseteq N$ is that part of N which includes all components, their properties and relations hosting the atomic services used to compose a specific service provided by p for c.

As availability decreases over time, the last maintenance of components will also impact the overall service availability. Moreover, the longer the service is expected to run, the more relevant is the possibility of eventual failure during its execution. One of the most common criteria when evaluating the quality of service providers is their interval availability, the uptime of a system over a reference period which is actually based on the instantaneous availability, the probability of a system to be available at a given time. More background information on service availability is given in Section 2.3.1. Part II provides a model-driven methodology to evaluate user-perceived instantaneous service availability based on the properties of the underlying ICT infrastructure.

The second part of this work focuses on the dependability of the discovery layer. More specifically, the responsiveness of active decentralized discovery. Active SD comprises operations where a client actively requests available service instances. In decentralized environments, these instances will directly respond to the requesting client. This request-response operation includes retries in case responses do not arrive in time. Responsiveness as introduced in [144] is defined as in Definition 1.3.

Definition 1.3. Responsiveness R is the probability of a system to operate successfully meeting a deadline, even in the presence of faults.

For discovery systems, the responsiveness reflects the probability of a client to receive a required number of responses until a given deadline. In Part III, we first provide insight into the responsiveness property in diverse SD scenarios under varying fault intensity. Second, we provide a hierarchy of stochastic models to evaluate user-perceived responsiveness of IP network based, active and decentralized discovery. The evaluation is again based on the properties of the underlying ICT infrastructure.

For simplification purposes, throughout the text *service client* and *providing service instance* are also referenced as *requester* and *provider*, respectively. A mapping of two specific instances requester and provider to the ICT infrastructure, that defines the user-perceived scope, is referred to as *service mapping pair*.

1.2 Problem Statement

The goal of this work is the development of two related methodologies that automatically generate evaluation models for user-perceived properties based on the current state of the network. The first methodology covers user-perceived instantaneous availability. The second methodology targets the user-perceived responsiveness of the service discovery layer including experimental validation.

1.2.1 User-Perceived Service Availability

A methodology is needed to support the model-driven assessment of user-perceived non-functional properties, such as instantaneous availability, based on a UPSIM (see Definition 1.2). To evaluate user-perceived service availability as in Definition 1.1, the methodology needs to merge the four dimensions – infrastructure, service, user and time – into a consistent model-driven evaluation. The methodology should include:

1. A model to describe ICT components including specific non-functional properties for availability evaluation (failure and repair rate, deployment time or time after last maintenance action) and a formalism to model networks as relational structures of those ICT components with the ability to assign roles (e.g. requester, provider) to specific components.
2. A model to describe services hierarchically as a composition of atomic services, including access times and durations of services.
3. A mapping of service elements to the relational structure that represents the ICT infrastructure, defining concrete service requesters and providers under evaluation and their redundant instances if available. In the case of redundant instances, a temporal order needs to be specified.
4. Generation of a specialized UPSIM according to Definition 1.2, which includes only those ICT components specific for the communication between a given pair requester and provider during execution of a previously described service.
5. Generation of a *user-perceived service availability model* (UPSAM) from this UPSIM according to the availability properties of the provided infrastructure at a given point in time.

Items 1, 2 and 3 should facilitate updates, as the infrastructure, its properties, the service description and user perspective will eventually change for different analyses. The complete methodology should be automated as much as possible to eliminate human errors during update or upgrade procedures. As a side goal, the methodology should be defined and implemented using well known standards and freely available tools to support external verification and to facilitate its dissemination.

Failure rates of ICT components are assumed to be given by hardware vendors or estimated using monitoring data, implying also software failures of service providers. Repair rates depend on the implemented maintenance strategy. Obtaining these values is out of the scope of this work. Modeling and predicting other external factors like network load is also not considered. The effects of such factors are assumed to be included in the failure and repair rates. This means we assume that all faults from classes *fail stop* to *byzantine* that happen after a defined deployment time are combined in the failure and repair rates of individual ICT components. An ordered fault classification can be found in [25] and is illustrated in Figure 1.2. We simplify the fault model by taking only constant failure and repair rates of ICT components into account. However, the methodology should be prepared such that given variable failure and repair rates, these could be included in the availability evaluation of individual ICT components.

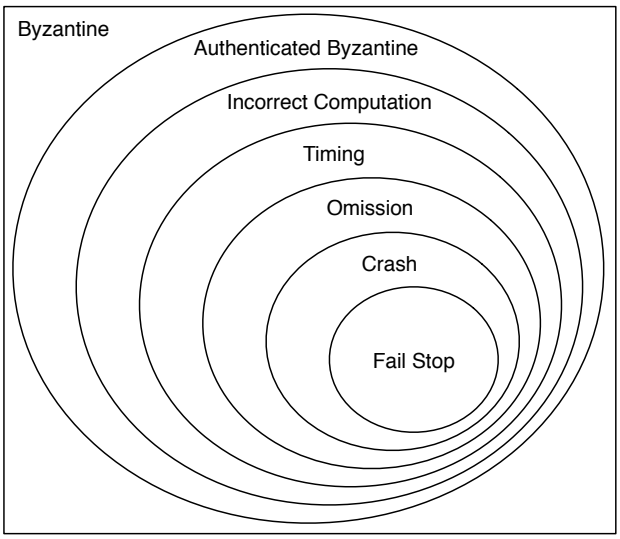


Fig. 1.2 An ordered classification of faults.

1.2.2 User-Perceived Service Discovery Responsiveness

A methodology is needed to quantify *user-perceived* responsiveness of active, decentralized SD. The user-perceived scope is defined by the position of requester and provider and the time of discovery. The methodology needs to use a stochastic model to evaluate responsiveness and an automated procedure that covers the following steps:

1. Define SD scenario that contains requester and provider, protocol and deadline for the SD operation.
2. Gather monitoring data from the network and prepare that data as input parameters of the models.
3. Instantiate specific models using these parameters and the scenario definition.
4. Evaluate user-perceived responsiveness by solving these model instances.

This work focuses on IP networks and their most common SDPs: Zeroconf [54, 53], SSDP [93] and SLP [106]. The focus lies on dynamic ad-hoc networks such as *Wireless Mesh Networks* (WMNs). Routing is done by the prevalent OLSR protocol [57]. The methodology should support evaluation of three different variants of SD responsiveness:

1. The responsiveness for different requester-provider pairs.
2. Average responsiveness of a specific provider for all requesters in the network.
3. A novel metric, the *expected responsiveness distance* should be investigated, to estimate the maximum distance from a provider where requesters are expected to discover it with a required responsiveness (see Definition 1.4).

Definition 1.4. Let t_D be the service discovery deadline, $R_{\text{req}}(t_D)$ the required responsiveness at time t_D, S a set of service providers and $C_d, d \in \mathbb{N}^+$ sets of service clients with d denoting the minimum hop distance of each client in C_d from all providers in S. Let $R_{\text{avg},d}(t_D)$ be the average responsiveness when discovering S from C_d. The expected responsiveness distance d_{er} is the maximum d where $R_{\text{avg},d}(t_D) \geq R_{\text{req}}(t_D)$ and $\forall d' \in \mathbb{N}^+, d' < d : R_{\text{avg},d'}(t_D) \geq R_{\text{req}}(t_D)$.

Considered faults are *fail stop*, *crash*, *omission* and *timing* (see Figure 1.2). Since the focus is on the real-time behavior of the SDPs in the network, all other faults are considered to be detected and recovered at higher layers and out of the scope of this work.

Extensive series of experiments should be run with two goals. First, to provide an insight into the behavior of responsiveness in modern service networks and to understand possible shortcomings of current SDPs. Second, to correlate the estimations of the proposed models with the actual measured responsiveness during the same period. This serves to validate the models.

1.3 Approach

Part II focuses on service availability. While the term user-perceived availability has been interpreted differently during the last 20 years, we define it as the availability of a service as provided by specified instances to specified service clients at a given point in time. Based on the approach initially introduced by Milanovic et al. in [157], we define a set of models and a model-driven methodology to automatically generate availability models and calculate the instantaneous availability for any given user perspective. Given a model of the network topology, a service description and a pair service requester and provider, a model-to-model transformation is applied to obtain a *User-Perceived Service Infrastructure Model* (UPSIM) as in Definition 1.2. The approach uses a subset of *Unified Modeling Language* (UML) [170] elements as well as UML profiles and stereotypes [171] to impose specific dependability-related attributes to ICT components. The ICT infrastructure and services are modeled independently using UML object and activity diagrams, respectively. Then, a mechanism is used to project the properties of ICT components to services through an XML mapping that correlates their respective models.

The methodology also provides a UML availability profile to obtain as output instead of an UPSIM a specific availability model expressed as *Reliability Block Diagram* (RBD) or *Fault Tree* (FT) to evaluate service availability for different user perspectives. To support one main contribution of this work, the evaluation of user-perceived *instantaneous* service availability, the probability of a service to be available at a specific point in time, the infrastructure model also includes the failure and repair rates and deployment times for all ICT components. The mapping model contains concrete ICT components for the agents service requester and provider, including possibly redundant components and their expected duration of usage.

Part III focuses on SD responsiveness. This property to date has seen no thorough evaluation and an extensive series of experiments is first done to get an idea about the characteristics of SD responsiveness in the target networks. The experiments are run in two different testbeds. The first is a virtual testbed where both topology and fault intensity can be controlled. The second is the *Distributed Embedded Systems* (DES) wireless testbed at Freie Universität Berlin, which provides realistic fault behavior. Both testbeds should guarantee a solid understanding on the state of SD responsiveness.

Second, a methodology is developed that considers the *user-perceived* responsiveness of given communication partners in active, decentralized SD. It estimates packet loss probabilities and transmission time distributions for each link on the communication paths between the partners and generates specific model instances to assess SD responsiveness. The hierarchy of stochastic models consists of mainly two parts. The higher level model describes the SD operation itself as a stochastic process using a regular discrete time Markov model. To map the discovery operation to a network under analysis, the individual SDP packets and their traversal through the network under analysis is done using semi-Markov models. They provide for each SD packet the probability to traverse the network and a distribution of the time to do so successfully. Their output is used to calculate the probabilities in the higher level model. In case no detailed knowledge about the lower network layers is available, the higher level SD model can also calculate responsiveness using measurement based distributions. Finally, the stochastic models are correlated with the actual results of the experimental evaluation to demonstrate their validity.

1.4 Main Contributions

The overall contributions of this dissertation are two different methodologies for user-perceived dependability evaluation in service networks. Both methodologies have the same foundation, to generate models for dependability evaluation bottom up based on the current monitored state of the network and to allow a time-dependent dependability evaluation using these models. Specific advances to the state of the art are mentioned in the following. More detailed contributions can be found in the individual chapter summaries.

- A fully model-driven methodology with state of the art tool support is provided which automatically generates and evaluates user-perceived instantaneous service availability models with a series of model to model transformations. The methodology contains proper specifications of the models and meta-models and the relations among them. All models were chosen focusing on visualization, such that while the methodology is able to work fully automated, human operators are able to understand the output models and manipulate the input models.
- For user-perceived instantaneous service availability evaluation, a concept of time for services and components is integrated into the methodology. This allows for a calculation of service availability at any point in time, based on the

individual age and wear of all components necessary for service provision between specified clients and providers.
- A comprehensive evaluation of service discovery responsiveness in experiments is presented. This evaluation goes beyond the state of the art in showing the dependence of responsiveness on the fault intensity, the number of actor nodes and the required time to successfully finish operation.
- This is the first methodology that enables generation and solution of stochastic models to calculate user-perceived responsiveness of service discovery. The model generation is hierarchical and models within the hierarchy can be exchanged or measurements used instead.
- The Monte Carlo method *Probabilistic Breadth-First Search* (PBFS) allows to efficiently estimate the reachability of network-wide broadcast protocols, which facilitates analysis of many other broadcast and multicast based protocols apart from service discovery.
- The *ExCovery* framework was developed to support experiments in distributed systems. *ExCovery* provides concepts that cover the description, execution, measurement and storage of experiments. These concepts foster transparency and repeatability of experiments for further sharing and comparison. *ExCovery* is released under an open source license to spur further development [8].

1.5 Structure of the Manuscript

This work is further structured as follows. We first summarize background information and related work in Chapter 2. The experiment framework *ExCovery* that was used to carry out the experiments on SD responsiveness is explained in detail in Chapter 3.

Part II covers the methodology to automatically evaluate user-perceived service availability. In Chapter 4, we define the input models and how the methodology uses them to calculate user-perceived steady-state and instantaneous service availability. Chapter 5 demonstrates the feasibility of the methodology by applying it to parts of the service network infrastructure at *University of Lugano* (USI), Switzerland. The service availability part is concluded by discussing further applications of the methodology and how to combine it with the later described service discovery models.

Part III of this work focuses on SD responsiveness. We present the results of extensive experimental evaluation in Chapter 6. Chapter 7 introduces the hierarchy of stochastic models to calculate responsiveness and also describes the Monte Carlo method PBFS. The part is concluded in Chapter 8, where the model results and experiment measurements are correlated. Advantages and shortcomings of the models are discussed.

Part IV closes the work and gives pointers for future research.

Chapter 2
Background and Related Work

Abstract Background information and an overview of related work is provided. We cover the main topics important to the work presented in subsequent chapters. The topics include the concepts of service oriented computing and service dependability metrics. We introduce models to evaluate these metrics with a special focus on wireless mesh networks.

2.1 Service-Oriented Computing

In the last decades, increasing requirements for both functional and non-functional properties have lead to a significant rise in system complexity. As a parallel trend, modern businesses are relying ever more on IT services. The whole concept of cloud computing is based on processes realized by a complex interaction of services [19]. Thus, businesses are heavily dependent on predictable service delivery with specific requirements for timeliness, performance and dependability. Failing to meet these requirements can cause a loss of profits or business opportunities and in critical domains, can have an even more severe impact. In order to tame complexity and enable efficient design, operation and maintenance, various modeling techniques have been proposed. They strive to help in better understanding IT systems by describing their components, predicting their behavior and properties through analysis, specifying their implementation and finally, enabling system validation and verification.

Computing and communication infrastructures have been converging rapidly in the last decade. Increased networking capabilities have allowed mobile and embedded devices to pervade areas of computing that used to have fixed environments which has helped to make them more flexible and dynamic. A plethora of new devices with different capabilities has entered traditional networks. At the same time, 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, protocols and interfaces are being developed and deployed that provide standardized methods for the different layers of service usage. *Service-Oriented Architecture* (SOA) describes a paradigm where services are the basic building blocks of system design. It recommends the design and implementation of interoperable services as discrete system components. SOAs are most commonly realized by means of web services [178] but in theory this is not a mandatory technological requirement. A business process model contains a clear specification of which services the system must provide to successfully accomplish a business goal. A concise overview on SOA is given in [119] while a comprehensive description of SOA and its principles can be found in [81]. The *Organization for the Advancement of Structured Information Standards* (OASIS) provided a reference model [141] in 2006 to enable the development of SOAs following consistent standards. Building these standards, *Service Component Architecture* (SCA) provides a programming model for building applications and solutions based on a SOA and is specified in [168]. Service-oriented system engineering [228] finally specifies the development phases of service-oriented methodology: specification, analysis and validation. As one of many examples, a rigorous model of SCA is presented in [77] that allows verification of such service component based systems.

2.1.1 Zero Configuration Networking

Supporting complex business processes, which possibly traverse multiple administrative boundaries, SOAs tend to be tedious to set up. 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 for example network addressing, service discovery, service description, application and presentation. 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 significant benefits. However, self-organizing networks are frequently deployed with wireless technology which is inherently unreliable. It is the goal of this work to investigate the dependability of decentralized service network protocols, specifically service discovery, in such networks.

Devices should be able to connect to the network and automatically configure all layers necessary for communication in the network, publishing, discovering and using service instances. This approach is called *Zero Configuration Networking* [51] and its basic requirements were formulated in [110], including addressing on the

network layer, decentralized name resolution, service discovery and description. All layers were supposed to be automatically configured and maintained, without administrative intervention. Possible protocol candidates to support the concepts were presented for example in [104]. Today various implementations exist to meet those requirements. In Internet Protocol (IP) [187] networks, which are the focus of this work, the most complete and prevalent ones are *Zeroconf* [55] and *Universal Plug and Play* (UPnP) [231]. Both have seen substantial backing through standardization and industry patronage and most devices targeting auto-configuring networks today support at least one of two. In the last years, with the rise of the Internet of Things [21], there has been a major shift in the targeting of auto-configuring service networks from home appliances to a globally connected, pervasive and heterogeneous network. Service-oriented concepts now also span such heterogeneous networks of smart embedded devices [99]. This creates major challenges for a manifold of properties within such networks, among them dependability [58]. The work at hand tries to address one part of this when investigating responsiveness of service discovery.

For the network layer, more precise requirements were defined in [243] to automatically configure IP Hosts. This lead to a standard for the dynamic configuration of link-local, non-routable IPv4 addresses which was based on the universally utilized address *Address Resolution Protocol* (ARP) [185], as described in [52]. The method today is known mostly under the term *AutoIP*. The dependability of AutoIP has been thoroughly investigated in research since its introduction and since it is not the focus of this work, no details will be given here. However, the models used in [39] to optimize retry strategies of the *AutoIP* protocol have been very influencing in investigating similar real-time Problems on the discovery layer, as presented in Chapter 7. In [39], the authors use Markov reward models [133] to intuitively describe the *AutoIP* protocol. We instead use both regular and semi Markov models to describe the discovery process in Chapter 7, but benefit from a similar intuitive transformation of the protocol standards to stochastic processes.

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 [172] or cost-effectiveness of network address configuration [39]. The performance and cost-effectiveness of service discovery using *Local Link Multicast Name Resolution* (LLMNR) [14] and *multicast Domain Name System* (mDNS) [54] with respect to network traffic generation and energy consumption is evaluated in [45]. This paper covers no effects of packet loss, an albeit common fault in wireless networks.

2.1.2 Service Composition

One important concept in SOA is composition, the possibility to combine the functionality of multiple services to provide more complex functionality as a composite service with a single interface. Service composition is supported first and foremost by the principles of composability and reusability [81] and has been a core focus

of SOA development and deployment from the very beginning [195]. A main challenge is the automatic composition of complex services from available resources and the verification and maintenance of such compositions, tackled among others in [167, 107, 155]. Additionally, compositions should be chosen with the goal of maximizing a specific quality metric [247], for example the availability of service delivery.

Following the all-encompassing work on composition in [155], if the individual services within a composition are indivisible entities regarding their functionality, they can be called atomic services. A composite service is composed of and only of two or more atomic services, while an atomic service can be part of any number of composite services.

Ideally, atomic service functionality should not be redundant, that is, every atomic service provides a different functionality. For instance, a composite service *email* could be divided into the atomic services *authenticate*, *send mail* and *fetch mail*. The indivisibility of atomic services is obviously in the eye of the beholder. Usually, their granularity is defined by the re-usability within the business process models. The atomic services *authenticate*, *send mail* and *fetch mail* could also be split into finer grained services. However, if the complete business process is well described with the current granularity in such a way that any of those three atomic services can be reused within other composite services without modifications, there is no need for further reduction. Throughout this work, we adopt and extend the service definition from Milanovic et al. [157], where complex services are described as a composition of atomic services:

Definition 2.1. Service is an abstraction of the infrastructure, application or business level functionality. It consists of a contract, interface, and implementation. *[...]* The service interface provides means for clients to connect to the service, possibly but not mandatory via network.

2.2 Service Discovery

In *Service-Oriented Architecture* (SOA), emphasis is put on the consideration of different ownership domains, so interoperability among services is an important aspect. There must be a way for potential partners to get to know of each other. This obligatory aspect is called visibility which is composed of awareness, willingness and reachability according to [141]. Among the principles introduced by SOA to support this aspect is discoverability. A comprehensive list and description of principles can be found in [81]. Discoverability means that structured data is added to service descriptions to be effectively published, discovered and interpreted. Communication of this data is done by means of *Service Discovery* (SD) and implemented by concrete *Service Discovery Protocols* (SDPs), which take care of announcing, enumerating and sorting existing service instances. Apart from [81], the basics of SD are explained for example in [198, 114] and, more recently, a survey has been given

in [150]. Service discovery is actually a special case of resource discovery in the network and a taxonomy exists with [232].

Service instances are often identified by a *Universally Unique Identifier* (UUID) which might also be a human-readable name. SD additionally provides a description of each enumerated instance, which contains the service type and holds more specific information for a client, for example, to locate the instance in the network, bind to it and use its provided service. Complete SD 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 by the SDP on the network. Enumerating and describing service instances is in fact frequently performed within one SD request packet and its response. The amount of description needed to use a service 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 IP 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. In the case of a printing service, for example, this could be the IP printing protocol and the information that this printer is able to print in color.

The service instances which have been enumerated by SD can be sorted in a subsequent step according to functional and non-functional requirements. This facilitates autonomous mechanisms like optimization of service compositions or fallback to correctly operating instances in case any one of the instances currently in use fails. These are capabilities which support the envisioned pervasive computing environments (see Section 2.1.1). In fact, decentralized SD, which is the main focus of analysis in Part III, was developed especially for pervasive computing. Decentralized SDPs regarding their concepts and capabilities have been classified in [249]. Unfortunately, the classification does not include the widespread *Zeroconf* protocol, which is, however, included in a survey by the same authors in [250]. A brief overview on decentralized SD systems can also be found in [79].

As we will show in Part III, the first generation of SDPs' extensive usage of multicast creates considerable challenges for the dependability of SD in multi-hop ad-hoc networks. No satisfying flooding mechanism exists to date in such lossy networks that combines both a high reliability and performance. For this reason, novel SD mechanisms have been proposed. Their improvements include a reduction of the propagation depth of discovery messages in the network [48], an inclusion of semantic information about the pervasive environment in the SD messages [220, 221], cache and retry timing optimizations [45] or cross layer solutions as in [138]. The authors of [89] introduce a middleware to abstract from the heterogeneity of different SD mechanisms and translate among them. A related approach exists in *Eureka*, where *Zeroconf* SD is used for component deployment [182]. Other approaches cover the transfer of the existing methods from IPv4 to IPv6 networks [122]. A survey with a comparison of many of these novel SDPs with a focus on ad-hoc networks is given in [154]. It needs to be noted, however, that none of the mentioned approaches provided intriguing benefits to gain significant industry backing.

2.2.1 Service Discovery Architectures

An abstract service class, such as *printing* is provided by concrete service instances in the network, for example *printer A, B* and *C*. A set of abstract service classes S can be provided on a set of concrete service providers P which then use a SDP to make the service known to requesting clients C. This process may be supported by a set of service registries R. Throughout this work, we will also call theses providing instances service providers and the clients trying to discover these instances service users. The three actor roles connected by SD can also be called user agent, service agent and directory agent [89]. They are also known as service consumer, service provider and service broker [43]. In this work, we will use the taxonomy of a general SD model developed by Dabrowski et al. in [66, 70], in which these roles are called *Service Manager* (SM), *Service User* (SU) and *Service Cache Manager* (SCM). We will use these specific terms only to improve readability in figures and stick instead to the intuitive terminology as given above. There is a slight difference however, as the authors of [70] distinguish between the actors themselves and software artifacts which act as agents in the process. An SM publishes its service on behalf of a service provider either autonomously or via a registry. It makes a service description available with information on how and where its service can be invoked: The provider identifier, a service type specification, an interface location or network address and optionally, various additional attributes. The SU discovers services on behalf of a user either by passively listening to announcements done by SAs or registries, by actively sending out queries to look for them, or by doing both.

As mentioned before, SD can happen in separate steps, enumerating discoverable instances first and then selectively retrieving the description. Also, not only services can be discovered, but administrative scopes, registries and service types, depending on the SDP. A registry caches service descriptions of multiple providers to maintain a list of present services that can be queried by clients. Registries are usually used to improve scalability. It should be noted that most SDPs implement also a local cache on clients and providers to reduce network load.

Two different SD architectures can be distinguished, as depicted in Figure 2.1: 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 two-party or decentralized architecture, there exist only clients (SUs) and providers (SMs) in the network which communicate directly among each other. A client that is interested in the functionality given by a specific service class discovers providers by a combination of passively listening to announcements and actively sending requests, with retries in specific intervals. All providers that may answer a query respond with a discovery response that is sent to the network or directly to the client. Depending on role and architecture, different communication types are used: unicast, multicast or broadcast. Two-party architectures are the main focus of the evaluation in Part III. The use of multicast generally causes higher load on the network than unicast. However, it may suppress requests from other clients by responding proactively and considerably simplifies distributed cache maintenance. The architecture is called three-party or centralized if there is one or more registry (SCM) present. Centralized does not imply a preceding administrative configuration

because a registry itself can be discovered at runtime as part of an SD process. There exist mixed forms that can switch among two- and three-party, called adaptive or hybrid architectures.

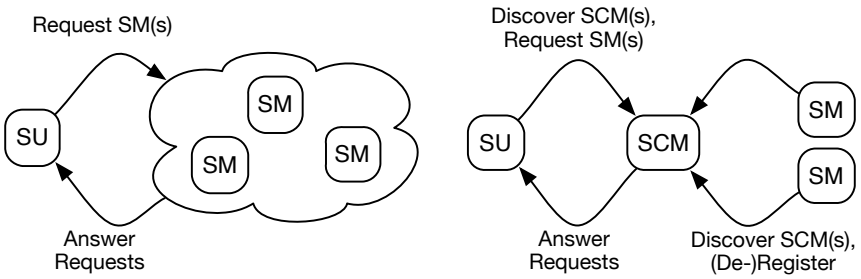


Fig. 2.1 Illustration of service discovery architectures: two-party (left) and three-party (right). SM: *Service Manager* or provider, SU: *Service User* or client, SCM: *Service Cache Manager* or registry

2.2.2 Service Discovery Protocols

The currently most common SDPs are presented and compared in [70, 198, 242]. Surveys on existing approaches to service discovery systems in ubiquitous computing environments can be found in [220, 250, 79]. They cover all well-known existing approaches to auto-configuring service networks. Such systems are the target environment of the experimental analysis in the work at hand. In IP networks, three SDPs are prevalent: *Service Location Protocol* (SLP) [103, 234, 106], *Simple Service Discovery Protocol* (SSDP) [93] and *Domain Name System based Service Discovery* (DNS-SD) [53]. DNS-SD as part of the *Zeroconf* protocol family is referred to by that name throughout the rest of this work. *Zeroconf* SD relies on a working DNS in the network, which is usually provided by *Multicast DNS* (mDNS) [54] support on all participating nodes. Especially SSDP and *Zeroconf* can be found in a plethora of embedded devices, such as printers, network-attached storage or cameras. The protocols transmit messages using the lightweight *User Datagram Protocol* (UDP).

Discovery protocols recover from timing and omission faults by retrying requests in certain intervals. The number of retries and the time between them vary among the protocols. *Zeroconf* and SLP specify an initial retry timeout and then double it every period. In SSDP, the requester may choose a timeout in a specified interval for every period. Values for the individual intervals are shown in Table 2.1. Quantitative analysis of specific properties to justify these strategies, responsiveness in particular, is practically non existent. They should be seen as best effort approaches. This is motivation for the analysis in Chapter 7, which in fact shows that static retry strategies struggle to perform reliably in dynamic networks. On the other hand, it is not trivial to find optimal retry strategies.

Table 2.1 Service discovery retry intervals for the studied protocols

	$t_{retry}(1)$	$t_{retry}(2)$	$t_{retry}(3)$	$t_{retry}(4)$	$t_{retry}(5)$
Zeroconf	$1s$	$2s$	$4s$	$8s$	$16s$
SLP	$2s$	$4s$	$8s$	$16s$	$32s$
SSDP (min/max)	$1s/5s$	$1s/5s$	$1s/5s$	$1s/5s$	$1s/5s$

A comparison of service discovery protocols, namely SLP, Jini (now known as Apache River [17]), salutation and SSDP can be found in [31]. A more recent and up-to-date collection of SDPs for energy and resource constrained embedded devices which also contains *Zeroconf* is presented in [236]. The authors also evaluate the mentioned protocols with respect to their adequacy in the target environment.

Protocol Communication Types

Depending on role and architecture, different communication types are used by the protocols: unicast, multicast or broadcast. Some SDPs include routing mechanisms, hence, overlay networks in their communication logic which is generally called SD with structured communication approach. Others leave this to the underlying layers, following an unstructured approach. Furthermore, the communication scheme used for discovery can be classified as passive (or lazy), active (or aggressive) or directed. In passive discovery, clients discover discoverable items only by listening to their unsolicited announcements. When doing active discovery, clients actively send out multi- or broadcast queries. In directed discovery, clients actively send unicast queries to a given registry or provider. There are many messages used by the SDPs to coordinate the distributed system, maintain a consistent state and optimize network traffic. All three common IP network SDPs follow an unstructured approach and do both active and passive discovery.

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. Sending replies via unicast on the other hand make sense if they contain information that is only valid for the requester. What messages are being sent by unicast or multi- and broadcast is basically a trade-off between network load and service data distribution and this trade-off is being evaluated differently in common SDPs. A sound compromise seems to be to resolve service types via multi- or broadcast and, if necessary, to resolve instance identifiers via unicast. This means to get a list of existing providing instances for a given type via multi- or broadcast and then ask a more precise description of a specific instance via unicast.

Zeroconf service discovery

Since Part III of this work focuses mainly on the *Zeroconf* SDP, we will give a more detailed introduction in this section. The *Zeroconf* stack works on top of the IP and has a very low overhead compared to other service network stacks, such as *UPnP*. It still provides complete auto-configuration of all layers up to service discovery. In recent years, *Zeroconf* became increasingly popular and by today is supported by numerous network services like printing, file or screen sharing and others. Linux and Macintosh operating systems including their mobile variants are deployed with *Zeroconf* technology enabled by default and implementations exist for virtually every operating system. The *Zeroconf* service network stack is described in detail in [55]. In short, *Zeroconf* handles the three lower layers of service networks [231] 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* which is better known as AutoIP and standardized in [52]. AutoIP introduces special types of *Address Resolution Protocol* (ARP) [185] 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 services. *Zeroconf* uses a multicast version of the *Domain Name System* (DNS) [163] called mDNS [54]. 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* (DNS-SD) mechanism [53]. 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 service instances with additional DNS TXT records although this is not of importance within the context of this work.

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 experiments in Chapter 6, consists of a single multicast request with multiple retries $1,\ldots,i$. The waiting time before a retry is 2^{i-1} seconds (see also Table 2.1). During that time the service client continues to wait for responses from service providers. Upon arrival of responses, it includes these known answers in subsequent requests to suppress duplicate responses. In [92], an approach is proposed which uses DNS-SD in wide area networks with centralized DNS servers to support discovery in service-oriented computing environments. The authors introduce specific service types for web services. Stolikj et al. have recently introduced a proxy concept to allow a 3-party architecture using *Zeroconf* SD [215]. So far, this approach remains a proof-of-concept and has seen no adoption in the official implementations.

Universal Plug-and-Play

The *Universal Plug-and-Play* (UPnP) protocol stack just as *Zeroconf* not only provides SD but defines layers for the complete auto-configuration of service networks including the service control and presentation. The full architecture is defined in [231]. For discovery, SSDP is used which works both in two and three-party mode. Messages are sent using the *Hypertext Transfer Protocol* (HTTP) over UDP. Requests are generally sent via multicast while responses are sent using unicast. Services are identified by a unique URI-tuple containing instance identifier and the service type, while the different types are standardized. Every UPnP service provider is accessible at a specific URL in the network where more details about the instance, other than its generic service type, can be obtained. Thus, using SSDP only instances of a specific service type can be enumerated, a more sophisticated discovery is not possible.

Service Location Protocol

Development on *Service Location Protocol* (SLP) started at the end of the last century. The latest standard for the protocol can be found in [106]. SLP has initially seen considerable industry support from companies like IBM, Hewlett-Packard, Sun Microsystems, Novell and Lexmark. SLP provides service clients with information about the existence, location, attributes and configuration of service instances. Within SLP, clients are modeled as *User Agents* and services are advertised by *Service Agents*. For scalability, a registry or *Directory Agent* may be used. In a two-party architecture, SLP uses multicast for requests and unicast for the responses while in three-party architecture, only unicast is used after a registry has been discovered. SLP messages are mostly text-based packets which are sent over UDP port 427. The search for services can be done by type or specific attributes rather than only by instance name. Services and their attributes are encoded in URLs in the form:

$$service :< srvtype >: // < addrspec >$$

In this URL, <*addrspec*> is a hostname or the dotted decimal notation of a hostname, followed by an optional colon and port number. A service scheme URL may be formed with any standard protocol name by concatenating `service:` and the reserved port name. There is also the possibility to have abstract and concrete service type as in:

$$service :< abstract-type >:< concrete-type >: // < addrspec >$$

Diverse other protocols

Several other SD protocols exist, although none of them have seen a considerable distribution in IP networks. We will list a short selection as reference. There exist complete application frameworks for short range wireless communication which include ad-hoc networking and discovery of existing devices and services. Most prominent are the *Bluetooth* system [35] and *ZigBee*, for which also a UDP/IP adaptation exists [223]. Once developed by Sun Microsystems, the *Jini* was a Java based technology for service-oriented computing. It is now known under the name *Apache River* and its specifications can be found in [17].

It has always been a tempting approach to embed SD within the routing layer to reduce the load of discovery communication. At the same time, SD would benefit from platform-specific optimizations on the lower network layers. This is of course at the expense of compatibility and a separation of concerns. Examples of such approaches can be found in [209] and, as a specific extension for OLSR, in [189]. The authors of [131] propose an overlay network for SD to improve flooding performance. In [143], diverse transparent cache strategies are proposed to improve the quality of SD in mobile networks with frequent partitioning of the network.

Finally, a lot of work has been done on improving the semantic expressiveness of SD requests and responses. Although this does not improve the real-time behavior of SDP operations, it might increase the probability for a client to get useful responses to a well-posed request. The fault model considered in this work does not include faults caused by unnecessary or omitted responses. These faults have to be recovered at a higher layer than the SDP itself and semantically enriched SD is one way to do that. An overview on personalized SD mechanisms in ubiquitous computing is given in [180]. How existing SD protocols could be extended to support semantically richer discovery is outlined in [239]. Prominent candidates for SD with increased semantic capabilities are, for example, *GloServ* [18], *EASY* [29] and the approach proposed by Paliwal et. al in [177].

2.3 Service Dependability

When evaluating the quality of service provision, various metrics have been the subject of interest. Metrics can be roughly categorized in performance and dependability related properties but depending on the focus of analysis and because of the interdependencies of non-functional properties, other categories can be justified as well. Computer systems do fail [97] and it is not the goal of this work to repeat the core research that has been done on dependability in the last decades. Instead we will provide works that give an overview and describe in more detail the metrics of interest to the research presented here. A concise summary of the concepts of dependable computing including a taxonomy can be found in [23]. We refer to [85] for a comprehensive definition of various dependability metrics. Among them are

reliability [83, 84] and performability [86], which form the base of availability (see Section 2.3.1) and its special case responsiveness (see Section 2.3.3).

In dependable computing the notion exists of faults which lead to errors in the system which in turn, if not handled properly, might cause system failures [113]. The fault is thus the initial state change, manifesting itself in an observable error in the system which by itself or in series, provokes system behavior not conforming to the requirements, hence, a failure. Dependable computing means dealing with the ever-occurring faults by reducing their number of occurrence (avoidance and removal during design and testing) and by limiting their impact (tolerance and evasion during runtime). An ordered fault classification can be found in [25]. Dealing with faults reduces the risk of system failure and their are various techniques to achieve this. For example, redundancy is a classical way of tolerating faults. Proactive fault management describes a novel approach to achieving a more dependable system by acting before a failure occurs [205]. All of those techniques require that faults can be detected and possibly located, which requires precise monitoring at runtime. A survey of online failure prediction methods, which can be used for proactive fault management can be found in [204]. An approach to adapt service-oriented systems proactively to improve dependability has recently been proposed in [151].

More focused on dependability of distributed systems in general, a superset of service networks, are the works in [63, 123]. Service-oriented systems have had their own share of dependability research. Fault taxonomies for SOA and for web service composition can be found in [44, 43] and [49], respectively. In [44, 43], also discovery faults are included. Important for discovery responsiveness are the class of discovery timing faults. Works on analyzing and maintaining quality of service throughout the service lifetime include [135, 46, 213]. Since determining the system state is crucial for precise dependability assessment, accurate monitoring systems are needed, possibly adapting to changes in the business process or in the services implementing such processes [26]. Since we cannot have perfect monitors in a dynamic distributed system, analysis also has to deal with the uncertainty in such monitored data, which can be considerable [95].

2.3.1 Availability

One important dependability metric in distributed systems is availability, which in general is the probability of a system to operate successfully. In case of service availability, this denotes the probability of a system to provide a specific service. System and service availability have been explained in numerous works already. It is the goal of this section to repeat only the most common characteristics in terms of the relationship of failure and repair rates and availability as this is important for the work at hand. For more in-depth information, the foundations of availability as a metric, which is based on reliability, can be found in [85]. A more detailed but nonetheless concise overview presents [98] which explains also common miscon-

ceptions about availability and reliability analysis. A study on the topic with a more specific focus on distributed systems like service networks can be found in [73].

Following Definition 2.1, there is a direct link between atomic services and the *Information and Communications Technology* (ICT) infrastructure, its hardware and software components required to provide the specified functionality. A service can be assumed to be available at the time it is requested if all network infrastructure components that are needed for communication between service requester and provider are available for the duration of service provision. This section will thus first explain different ways to calculate component and service availability.

Empirical analysis has shown that components are more prone to failure at the beginning and at the end of their life-cycle. In this sense, the failure rate (λ) over time results in the so-called bathtub failure curve. Vendors usually try to mitigate the effects of infant mortality by intensively testing the hardware before dispatching it so that customers receive the product at the stage of lowest failure probability, in which the failure rate is nearly constant until the wear-out stage, that delimits the end of product's life-cycle. This stage of constant failure rate corresponds to the major part of the life time of an electronic component. The failure *Probability Density Function* (PDF) $f(t)$ gives the relative frequency of failures at any given time t. For a constant failure rate, it can be approximated as an exponential function (see Equation 2.1).

$$f(t) = \lambda \cdot e^{-\lambda \cdot t} \tag{2.1}$$

The *Cumulative Distribution Function* (CDF) $F(t)$ represents the probability of a failure occurring before time t, and is given by the Equation 2.2.

$$F(t) = \int_0^t f(t) \cdot dt = 1 - e^{-\lambda \cdot t} \tag{2.2}$$

The complement of the CDF is therefore the probability of a component to perform its functions for a desired period of time without failure, better known as the reliability $R(t)$ of a component (Equation 2.3).

$$R(t) = 1 - F(t) = \int_t^\infty f(t) \cdot dt = e^{-\lambda \cdot t} \tag{2.3}$$

For repairable systems, the probability of a component to be alive at time t is given by the probability that no failure has occurred before t, which is the reliability $R(t)$ itself, and the probability that after the last failure, the component was repaired at time x, with $0 < x < t$, and has worked properly since then, $R(t-x)$. This probability is called availability $A(t)$ or more specifically, instantaneous availability. It can be expressed in terms of $R(t)$ and the probability of repair at instant x, given by $m(x) \cdot dx$ (see Equation 2.4).

$$A(t) = R(t) + \int_0^t R(t-x) \cdot m(x) \cdot dx \tag{2.4}$$

For constant failure rate λ and repair rate μ, the availability $A(t)$ can be expressed as in Equation 2.5. As can be noticed, for a repair rate tending to zero, $A(t)$ tends to reliability $R(t)$.

$$A(t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} \cdot e^{-(\lambda+\mu)\cdot t} \tag{2.5}$$

The interval availability $A_I(t)$ is the probability that a system is operational during a period of time (Equation 2.6).

$$A_I(t) = A(0,t) = \frac{1}{t-0} \cdot \int_0^t A(\tau) d\tau \tag{2.6}$$

Throughout this work, the function $A(t_1,t_2)$ is used interchangeably for interval availability, where t_1 and t_2 denote the start and end times of the evaluated interval. The steady-state availability is the probability that a system is operational when $t \to \infty$. As seen in Equation 2.7, it depends only on the failure and repair rates of components.

$$A = \frac{\mu}{\lambda+\mu} \tag{2.7}$$

Hardware vendors usually provide the *Mean Time To Failure* (MTTF) of their products. For repairable systems, the *Mean Time Between Failures* (MTBF) can be used instead, which means that a unit is not replaced but repaired after a failure. In either case, the *Mean Time To Repair* (MTTR) depends on the implemented maintenance processes. For instance, a network administrator may have to completely reinstall and reconfigure a server, or simply use a replacement machine ready for eventual failures. The first case can lead to a longer MTTR. Constant failure and repair rates can be obtained by calculating Equation 2.8 and 2.9, respectively.

$$MTTF = \frac{1}{\lambda} \tag{2.8}$$

$$MTTR = \frac{1}{\mu} \tag{2.9}$$

Equation 2.10 then calculates the steady-state availability of a component or a system using those values.

$$A_{comp} = \frac{MTTF}{MTTF+MTTR} = \frac{MTTF}{MTBF} \tag{2.10}$$

2.3.2 Availability Evaluation

Comprehensive collections of foundations, models, methods and tools that can be used for service availability assessment can be found in [156, 124, 147, 121]. In

[225], various stochastic models for availability assessment are explained and compared in case studies. More recently, the CHESS project [20] was founded by a joint of public-private partners of the European Union to support the development of an automated tool for dependability analysis. The CHESS project focuses on embedded systems and provides an Eclipse-based tool with its own modeling language.

The automatic generation of steady-state service availability models from service descriptions and infrastructure information is presented in [158, 148, 156, 157]. In fact, the authors also calculate instantaneous availability, assuming that all service components have been deployed at the same time, hence, they have the same age. Due to this limited concept of component and service time, the usefulness of the approach is clearly in steady-state analysis. The methodology relies on a configuration management database system to gather topology information at run-time, but prior evaluation during design time is also contemplated. The description of the ICT infrastructure could be completely automated. As a drawback, this requires an external tool, such as a network management tool, to provide the information. The automation has thus not yet been fully realized, however the approach has been implemented in [230]. In their methodology, the authors propose the usage of a *Depth-First Search* (DFS) algorithm [87] to discover all possible paths between service requester and provider. For each path a boolean equation is generated. The paths are then merged and simplified into a single equation using an external boolean solver application. This resulting equation is converted to an RBD model which is, again, evaluated with an external application.

A related approach with state-of-the-art tool support is presented in Part II of the work at hand. Its main methodology is based on the work by Milanovic et al. in [157] and designed to work fully automated. It addresses some of the shortcomings of the approach presented by Milanovic et al. by providing a fully model-driven workflow with proper specifications of the models and meta-models and the relations among them. This allows for a series of model-to-model transformations to generate e.g. the availability models, but also other output models are possible by design. In [230], a similar idea was pursued using an intermediate model representation. However, necessary details about the approach have not been published and the concept itself is limited when compared to the workflow provided in Part II. The approach in this work can be employed with minor modifications for different types of dependability evaluation: A different UML profile and the final model transformation needs to be specified. In addition to the model-driven design, our approach uses more expressive models for the infrastructure representation. The UML object diagram used is highly versatile, which allows to specify not only inherent properties of components but also structural information such as the level of redundancy. Finally, the approach presented in Part II defines for each component its failure rate, repair rate and its age. In addition to the definition of temporal order for the serial and parallel execution of services, this allows for a proper evaluation of instantaneous availability.

Other than in Part II, more information can be found in [2], where the general methodology is described, which has been extended to include the time of analysis to support instantaneous availability evaluation in [12]. A case study for steady-state availability evaluation can be found in [5]. Details on the implementation are

described in [165, 196]. The stochastic modeling and evaluation were supported by the Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE) [224], which is introduced and described to detail in [202]. SHARPE was also used for parts of the SD responsiveness evaluation in Part III of this work.

The authors of [245] provide a stochastic model to assess user-perceived web service availability and demonstrate that there can be significant differences between the system perspective and perspectives of individual users. In [211] a new status-based model to estimate user-perceived availability proposed. However, both works do not model the providing infrastructure in detail. The availability of a cell phone network from different mobile user perspectives and in different load conditions is analyzed in [40]. The dependencies of quality of service on the client side and on the server are well demonstrated in simulation. Also in [238], a model to evaluate user-perceived service availability is described. However, the approach has a different service model in mind. Services are perceived as available by a user if their specific resources are available upon request by that user. In contrast, Chapter II of this work defines a set of models and a methodology to automatically generate user-perceived availability models that calculate $A_{P,c}$ of service provider P for any given user c. The models do not take into account the quality during service usage but focus on the availability of the network infrastructure used during service communication between arbitrary pairs requester and provider. The user-perceived scope is thus defined by the network subgraph of such a pair and not by the quality requirements of a specific user.

Finally, two connected approaches warrant citation. The first work describes *Gravity*, a project which tackles autonomous adaptation of component based applications to optimize availability [47]. Components are modeled as services, thus the project is one way to implement a SCA. Related to the work at hand is the consideration of components with different availabilities, which leads to a dynamic availability model just as the models for user-perceived views as in Chapters 4 and 7. The second approach pursues the fact that traditional availability models are usually based on technical monitoring data of the systems they analyze. These models do not scale well to complex IT infrastructures without a high level of abstraction, which is also discussed in Section 5.5. Additionally, during planing and design of infrastructure changes, this information is generally not available or incomplete. Instead, a novel methodology is proposed in [94] based on IT reference models that define best practices for building IT infrastructures to meet certain requirements. Meeting such requirements may lead to certification by official authorities. The authors of [94] propose to quantify the grade of meeting the requirements, which to date remain purely qualitative. Given a selection of processes in the reference models that have been identified to be related to availability as in [10], one could use this approach to do a quantitative assessment of availability which is not based on monitoring data of a running system. To date, the approach has not been backed up with empirical data.

2.3.3 Responsiveness

If we are interested in knowing the probability that a system is able to perform as expected until a given deadline, we have to extend availability with a notion of real-time. Such metric as used in this work for user-perceived SD evaluation in Part III is called responsiveness. The foundations for responsive computing as a combination of fault-tolerant and timely execution have been laid out in [144, 145]. A precise definition can be found in [146] together with a consensus based framework to design responsive systems. Two types of responsiveness are distinguished in [146]. The first describes an optimization problem of a pair of parameters timeliness and availability. The second, which is the type covered in this work, combines timeliness and availability in a single measure. Informally, responsiveness can be defined as in Definition 2.2, which assumes responsiveness as a function of the availability of a system and the timeliness of its execution.

Definition 2.2. Responsiveness R is the probability of a system to operate successfully within a deadline, even in the presence of faults.

Both measures are time dependent and a combined metric for responsiveness can be expressed as in Equation 2.11.

$$R(t_0,t_D) = A(t_0,t_D) \cdot P(t_0,t_D) \tag{2.11}$$

The interval availability $A(t_0,t_D)$ (see Equation 2.6) is the probability of failure-free operation from time t_0 until deadline t_D and $P(t_0,t_D)$ is the probability to finish a given task successfully until t_D . Generally, we can omit t_0 and assume it as the time of evaluation of responsiveness $R(t_D)$. In the area of service-oriented systems, a task could reflect service provision to a specific client while the availability is the user-perceived availability of the ICT infrastructure for that client. Responsiveness can be extended for multiple tasks as in Equation 2.12 where n is the number of parallel tasks that need to finish successfully until the deadline.

$$R(t_0,t_D) = \frac{1}{n} \cdot \sum_{i=1}^{n} A_i(t_0,t_D) \cdot P_i(t_0,t_D) \tag{2.12}$$

Various works with a focus specifically on this combination of fault tolerance and real-time have been collected in [90]. Responsiveness remains a research issue until today. For example, the authors of [41] present a combined model to predict the availability and response times of services and investigate the dependability of both metrics in simulation. They do not provide a probabilistic evaluation of response times nor a single joint responsiveness metric as in Equation 2.11. The authors of [194] examine various restart strategies for web service reliable messaging with respect to the effective service response times and the cost of the strategy. Furthermore, they propose a metric to determine the quality of a specific strategy, which could be used to adapt retry strategies of current SDPs (see Section 2.2 and Table 2.1) to optimize the responsiveness of SD.

2.3.4 Performability, Resilience and Survivability

Very often, it is not only of interest whether a system is able to perform but also, whether it can perform with a certain quality as required by the user, given the presence of faults. This is the domain of performability, which combines both availability and performance evaluation in a single metric. Performability has been the focus of many works in the last four decades and we will only mention a quick selection of works, such as [28, 152] or, more recently, the summary of the concepts in [86]. Usually, performability is assessed as the probability to perform at a required level over a period of time. Systems can be described using reward-based models where each performance level will give a defined reward over time [56]. This way the expected accumulated reward over a period can be calculated. How to best construct such models is explained in [153] while an overview on tools and techniques for performability modeling is given in [111]. Related to the work at hand, there exist more recent model-driven approaches to evaluate the performability of composed services, which also allow the optimization of compositions with respect to performability [38, 96].

Availability and performability analysis usually handles the quality of service during regular operation, under the assumption of a specific fault model. While these models may include severe faults, the probability of the occurrence of such faults is usually assigned low. Resilience and survivability analysis on the other hand is focused on the probability of a system to withstand catastrophic events, for example terrorist attacks or natural disasters, since IT networks have become part of critical societal infrastructures. Although these metrics do not form the scope of this work, responsiveness and availability as calculated in Parts II and III could well be used as underlying measures to support a comprehensive resilience and survivability assessment. We refer to [129, 112] for a proper definition. In [214] a survey of the disciplines encompassed by resilience and examples for applications are presented.

2.3.5 Service Discovery Dependability

If discovery fails, a service cannot be available. Comprehensive service dependability evaluation thus needs to consider the discovery process. Nonetheless, SD has traditionally been neglected when evaluating service dependability. Due to the diversity of usage scenarios and the dynamics of modern networks, the dependability of SD is not trivial to predict. This problem is exemplified in unreliable networks with more complex fault behavior, such as self-organized *Wireless Mesh Networks* (WMNs), where the quality of links is constantly changing and heavily affected by external interference, fading effects and multi-path propagation. At the same time, self-organized networks belong to the type of environment where SD is expected to become ever more important.

As a time-critical operation, one key property of SD is *responsiveness* as defined in Section 2.3.3, the probability of successful operation within a deadline even in the

presence of faults. 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. 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 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 forced packet retransmissions result in an increased network load.

As an integral part of service usage, comprehensive service dependability evaluation should include SD 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 [86]. On the other hand, reducing the time to discover the service increases the risk of not finding it, in which case it would not be able to perform at all. Unfortunately, until now few works examine SD responsiveness and in service dependability evaluation, it has generally been neglected.

Dabrowski et al. evaluate different dependability properties of existing discovery protocols in [67, 68, 69, 72]. These include *update effectiveness*, the probability to timely restore a consistent state after failure, which resembles a specific case of responsiveness. They did not consider active SD responsiveness during regular operation. For an explanation of active versus passive discovery see Section 2.2.1. Furthermore, the widespread Zeroconf protocol is not considered. *Zeroconf* responsiveness has been evaluated in experiments in [6]. The research described in [71] is closely related to the topic of the work at hand. Here, the robustness of existing discovery mechanisms is evaluated under increasing failure intensity. However, also in [71] responsiveness is not covered in particular and the technologies used in the experimental evaluation cover SLP [106] and UPnP [231], but not *Zeroconf*. Different discovery protocols are compared with respect to their response times in [212], the work has a strong focus on the individual implementation overhead and does not consider faulty networks. The authors of the work at hand present results from responsiveness evaluation of IP discovery protocols in [6, 7] and provide a hierarchy of stochastic models to reproduce these results in [4]. Furthermore, parts of the results in [7] may be used as input data for the stochastic models to evaluate SD responsiveness in diverse scenarios instead of running time-consuming experiment series for each one of them. Responsiveness of SLP in particular is demonstrated in [61]. Part III of this work will provide a comprehensive summary of those works.

Other dependability properties are the robustness of SD, which has been examined with respect to discovery delay times in [172]. Preliminary measurements of multicast SD have been done in [118] for the campus wireless network at Columbia University, New York. They show that depending on the use cases of *Zeroconf* SD, the amount of multicast traffic caused can be considerable. The performance and cost-effectiveness of two different SDPs under different conditions of network traffic and energy consumption is evaluated in [45]. However, the paper covers no effects of packet loss. A stochastic model for the cost-optimization of the *Zeroconf* addressing layer protocol *AutoIP* has been presented and evaluated in [39]. Chapter 7 of this work proposes a related model for the discovery protocols.

Although security issues are not covered in this work, it should be mentioned that the lack of a single authority in decentralized service networks makes administration and control of the service instance identifier space more difficult. Additional measures need to be implemented to guarantee the trustworthiness of service discovery. An approach that introduces encryption and authentication on the SD layer is given in [65, 117]. For DNS-SD, the recent works in [127, 128] propose extension to support privacy. A method to progressively expose sensitive discovery information is presented in [251] but introduces significant delays in the SD operation, thus, reducing responsiveness.

Considered Fault Model

Impairments to SD dependability are generally the same as for generic distributed systems: Faults can roughly be categorized in network and node faults. They result in data loss, data delay, out-of-order data, duplicated data and varying timing errors from one data unit to another, such as jitter. UDP is an unreliable transport so recovery operations for the IP based SLP, SSDP and *Zeroconf* are done by the SD protocols themselves. Following the classification in [25], the different fault classes can be described for SD as follows (see also Figure 2.2):

Fail stop A node leaves the network and announces this to the rest of the service network. No more packets will be received by and sent to this node.

Crash A node leaves the network but is unable to announce it. No more packets will be received by and sent to this node.

Omission A certain percentage of packets gets lost either in nodes or in the network. This includes also the loss of all packets.

Timing Packets get delayed either in nodes or in the network. The delay may also be infinite in which case a packet is lost. Packets cannot arrive too early which would violate the causality.

Fail-stop faults may be classified as regular exhibition of network dynamics and are recovered by goodbye messages. Crash, omission and timing faults are recovered by request retries and timeouts as described in Section 2.2.2. The classes *incorrect computation*, *authenticated byzantine* and *byzantine* are not covered in this

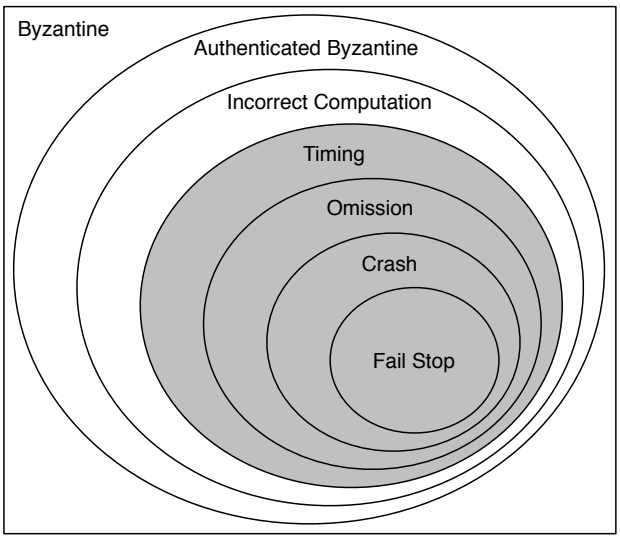


Fig. 2.2 An ordered classification of faults. Classes important for the temporal behavior of service discovery have been highlighted.

work and should be handled by application layer mechanisms like packet checksums and related integrity checks. While generally a dependency of faults has to be assumed, we abstract from this in the analysis in Chapter 7. Also, there are possibly arbitrary faults at lower layers which might manipulate packets such that byzantine faults at the discovery layer happen. We expect these errors to be negligible in the controlled environments covered in this work.

2.4 Modeling Service Networks

Throughout this work, various stochastic models are being introduced and evaluated. For a definition of the underlying formalisms we refer to the excellent compendium on stochastic modeling and analysis in [133]. The methodological foundations of network analysis as carried out in Parts II and III are provided with [42]. Since network topologies can almost always be represented in graph structures, common and well investigated algorithms can be utilized, many of which are explained in [87].

When designing and modeling ICT infrastructures, it is advantageous to make use of the principles of *Model-Driven Development* (MDD) [207], which can provide a better understanding of arising problems and their potential solutions through system abstraction. Models are abstractions of a system and/or its environment. Every model conforms to a meta-model, which defines the syntax of the model as an explicit specification describing the relevant concepts and relations between these

concepts [32]. The idea of obtaining one model from another characterizes a model to model transformation, where the input and output models can either share the same meta-model or have a completely different syntax. Usually, such transformations rely on identifying graph patterns as model elements and match them to given structures of the meta-model, as accurately described by Czarnecki et al. in an extensive survey on model transformation [64].

As a model formalism, the often used and standardized *Unified Modeling Language* (UML) consists of a set of graphic notations and relations to model structural and behavioral characteristics. A full specification of the current version of UML can be found in [170], complemented by [171]. In addition to a standard set of 14 different diagrams, UML has two important customization mechanisms: profiles and stereotypes [171]. These features enable designers to aggregate detailed information and better represent complex systems with diverse properties. Basically, UML profiles are mechanisms to customize the existing UML models by allowing the addition of permanent attributes according to the nature of the target model. Stereotypes specify new modeling elements with properties called stereotype attributes and are applied to existing UML elements, which then automatically inherit the respective attributes. Profiles can be composed of stereotypes and their attributes, describing model semantics with stereotypes and constraints. Thus, profiles and stereotypes are additional instruments to tailor UML models and their elements (e.g. classes, associations, objects) for specific needs.

To be applicable to UML diagrams, when designing a profile each of its stereotypes must extend a UML element. For instance, a stereotype S, extending the class element and containing the stereotype attribute A can only be applied to classes, which are then denominated stereotyped classes and inherit the attribute A. We will make use of these concepts in Part II of this work, where we use UML class, object and activity diagrams to model an ICT infrastructure and the services that run on it. We use profiles and stereotypes to make sure that all ICT components contain properties needed for a subsequent availability analysis. These properties contain static and dynamic properties, which are updated at runtime. Although not specified, in a full featured SOA these dynamic properties could also be gained by runtime monitoring, for example by a monitoring manager as proposed in [26].

The models needed to carry out this analysis are obtained by a model to model transformation from the UML models to well known availability models, specifically *Reliability Block Diagrams* (RBDs) and *Fault Trees* (FTs) [235]. While Part II provides a comprehensive overview of the complete methodology including an evaluation in case studies, different aspects of this work are described in more detail in [2, 5, 12]. An interesting extension of this methodology would be to use a *Fuzz Tree* (FzT) as output model. The first approach to include fuzzy logic in fault tree analysis has been presented in [216]. Today, high performing tools for modeling and evaluation exist [226].

A different definition of services is proposed by the service availability forum [210] in the *Availability Management Framework* (AMF). AMF specifies components as basic framework entities that consist of a set of software or hardware resources. In contrast to [148, 157], where infrastructure and services are mod-

eled independently, AMF components are intrinsic service providers, which can be grouped into bigger logical units called service units. Salehi et al. [203] developed a UML-based AMF configuration language to facilitate the generation, analysis and management of the AMF configurations. The language has been implemented by means of a UML profile. *Dependability Analysis Modeling* (DAM) [30] consists also of a UML profile for dependability modeling, extending the existing MARTE profile [169] to better represent non-functional properties. It correlates service and ICT components by defining the relation of a service and its underlying infrastructure, and describes them with a complete set of properties, although no transformation is provided by the methodology. The DAM project provides an extensive set of models with detailed attributes, but it is focused on modeling only and does not provide a transformation to reliability models for further evaluation. Furthermore, since the DAM methodology proposes a modeling of the system entirely using UML, changes to the mapping of service providers and ICT components have to be reflected within such a model. Even with clear annotation, this procedure can be tedious and error prone. Moreover, DAM is conceived as a complex UML Profile, an extended UML model, aiming at dependability modeling, thus excluding other non-functional properties.

A work inspirational to this paper describes dependability analysis using directed acyclic graphs [201]. Chapter 7 combines a network topology and a discovery operation in a Markov model that reflects such a graph.

2.5 Wireless Communication

Over the last two decades, wireless communication has become commonplace, introducing new paradigms for communication in distributed systems. WMNs, which consist of sets of nodes distributed within radio signal range of each other, have lately become increasingly diffused in a manifold of applications. Depending on the focus of application, a WMN may also be called *Wireless Sensor Network* (WSN), while *Wireless MultiHop Network* (WMHN) is an umbrella term for WMN and WSN. For reasons of simplicity, we will continue to use the term WMN in this work.

A relatively low cost of deployment, ad-hoc and automatic configuration are among the advantages of WMNs. However, inherent dependability issues such as complex fault behavior and fault dependencies due to the wireless communication continue to be a problem. A concise overview on the most important performance characteristics of the prevalent 802.11 networks [120] which are the target of this work is given in [78]. Generally, the gaps between theoretical and practical evaluation of WMN properties can be quite wide [161]. A survey of different types of wireless networks, their technological background and dependability properties and challenges exists in [16]. The same group also provides a survey with a focus specifically on WSNs [15], a more current work on that topic can be found in [246]. Simulating the behavior of wireless signals needs a proper model. In this work,

when estimating the propagation of radio signals, we refer to models based on the Jakes propagation loss model as in [248].

Routing

A diverse selection of routing metrics exists in wireless networks [179] which are being used by routing protocols. Most widely used for IP networking in WMNs is *Optimized Link State Routing* (OLSR) [57] and we will also use OLSR to model routing in WMNs in Chapter 7. OLSR nodes proactively search for routes and cooperatively create a spanning tree that covers the whole topology. A number of different metropolitan networks, such as in Athens, Berlin and Leipzig successfully employ OLSR. Our experimental evaluation relies on OLSR with multicast support [227] to support discovery protocols. OLSR has seen numerous optimizations, such as in [102], where a variant of OLSR is introduced that tries to optimize routing with respect to packet delay, packet loss and energy consumption constraints.

As opposed to proactive routing protocols, there also exist reactive methods, which look for routes only upon request, among the most common ones being *Dynamic Source Routing* (DSR) [126, 125] and *Ad-hoc On-demand Distance Vector* (AODV) routing [183]. Reactive routing is usually not used in the pervasive environments as targeted in this work and no more detail will be provided here. Instead we refer to [179].

We are aware of the fact that the routing topology has a significant impact on the performance of wireless networks and results valid for one protocol are not necessarily valid for another, an effect which is not only demonstrated in [190]. Our model in Section 7, however, is to a certain extent independent of the routing protocol used and only integrates the link quality metric as provided by OLSR. It is able to provide results for other protocols in a similar manner, as long as they create single path routes for unicast communication and the flooding mechanism for multicast and broadcast can be modeled with *Probabilistic Breadth-First Search* (PBFS) [11].

While many works on WMNs limit their evaluation on simple topology models, such as regular grids, the authors of [160] provide an algorithm to generate topologies which are close to the ones found in real world metropolitan networks. We also use this algorithm for validation in Section 7.4.

2.5.1 Packet Transmission Times

The responsiveness model in Chapter 7 of this work relies on an estimation of transmission times over the wireless link. Several approaches exist which model packet transmission delays at the 802.11 MAC level, e.g. [175, 193]. They are not considered in this work due to their complexity. An explicit method to calculate the network delay depending on the back-off process with retry limits, acknowledge-

ments, unsaturated traffic, and packet size is presented in [181] and could be integrated into our models. Compared to PBFS [11] it has not been validated with observations from real-world wireless networks, however. Bianchi [33] developed a Markov model to compute the 802.11 *Distributed Coordination Function* (DCF) saturation throughput. It assumes a known finite number of terminals, ideal channel conditions and all nodes in one collision domain. These assumptions do not hold for the WMNs targeted in this work. Instead of a detailed modeling of low-level MAC operations, the approach presented in Chapter 7 favors an integration of application layer protocols. The delay estimation is based on the *Expected Transmission Count* (ETX) metric [62] used by OLSR [82] with packet transmission delays as defined in the 802.11 standard [120]. This method is related to [166], but is considerably more efficient. On the other hand, the accuracy of this method depends on the accuracy of the ETX metric, or, on the accuracy of the link status detection metric in general, which has been examined in [162]. Here, more reliable metrics as proposed in [159] could be integrated.

Various extensions to OLSR exist, which extend the routing metric with information about the link delay. Such delay information could be propagated, for example, with a mechanism as proposed in [184]. It can then be used for a delay-centric routing, which optimizes routes through minimizing the end-to-end delay as in [137, 80, 136]. The delay information could further be used in a cross-layer optimization to improve the quality of service of multimedia applications [59]. Our work focuses on SDPs communicating on standard OLSR topologies. The mentioned delay-centric optimizations of OLSR show potential to increase the responsiveness of SD operations without altering the SDPs themselves and it seems worthwhile to investigate this in future research.

2.5.2 Network-Wide Broadcasts

A fundamental operation for WMNs are *Network-wide Broadcasts* (NWBs), which are required by many of the auto-configuring protocols, such as routing, when distributing information to all nodes in the network. Also SD is an operation that relies heavily on NWBs. However, due to the characteristics of wireless communication, NWBs are generally problematic. Current protocols for NWBs lead to the well-known broadcast storm problem, which was pointed out by Tseng et al. in [229]. It essentially describes excessive redundant transmissions that happen in a meshed network during basic flooding. Efficient flooding poses a great challenge so using broadcast or multicast should be considered carefully. NWBs have been the target of numerous optimizations with very different goals, among them OLSR's MPR selection scheme [57, 108], probabilistic flooding and flooding with retransmissions [37]. The authors of [229] and others proposed approaches focusing on removing redundant transmissions which emanate from a simplified, unweighted graph model. A standard for such an approach has been ratified with simplified multicast forwarding in [142], which is also implemented for OLSR [227]. Other approaches cover

the probabilistic reduction of the graph size to select forwarder strategies [206], which have been evaluated e.g. in [174]. However, removing redundancy in lossy networks also hurts reliability and can severely decrease NWB reachability [139].

Optimizing NWBs is a prime target when improving the overall performance and dependability of WMNs in general and specifically the responsiveness of SD, as it is important for the work at hand. To properly evaluate trade-offs between performance and reliability, researchers usually need to rely on extensive simulation and experimentation, since accurate analytic methods do not scale to realistic network sizes due to their complexity. Most existing optimizations neglect the real nature of WMNs and are based on simple graph models, which provide optimistic assumptions of NWB dissemination. *Unit Disk Graphs* (UDGs) are one way to obtain these unweighted graphs and are often employed for WMNs and NWBs. In an UDG, a link always exists when two nodes have an Euclidian distance of less than 1, otherwise not. Kuhn et al [132] propose an extension called *Quasi Unit Disk Graphs* (QUDGs), introducing an uncertainty that allows to model the impact of obstacles on wireless transmissions. On the other hand, models that fully consider the complex propagation characteristics of NWBs quickly become unsolvable due to their complexity.

Few approaches consider link qualities when optimizing NWBs as in [252] or [139, 140], where flooding with different forwarder selection strategies was proposed and evaluated. Evaluation revealed that in reasonably lossy networks, existing optimizations severely decrease NWB reachability when removing redundancy. In the context of this work in Part III, when trying to estimate several properties of multicast communication as used in SDPs, we are interested in calculating the reachability for NWB protocols, given a weighted topology as input. Oikonomou et. al [173] did a similar analysis for probabilistic flooding, using randomly generated graphs and not considering edge weights. Chen et. al [50] define the *Flooding Path Probability* (FPP), which corresponds to the reachability as defined in Section 7.3.1. Their provided algorithm to compute the FPP has a reduced but still exponential complexity. This lead to the development of PBFS ([11], see also Sections 7.3 and 7.4), a Monte Carlo method for evaluation of different measures of NWB protocols.

2.5.3 Wireless Service Discovery

For modern decentralized networks, such as WMNs with possibly mobile nodes, SD becomes ever more important as the number and position of providers may change dynamically. As can be seen in the model-based evaluation in [4] and in Part III of this work, SD responsiveness is difficult to predict in such networks and varies dramatically. Optimizing the responsiveness of SD in these environments should thus be a major focus in service network research.

A few SD mechanisms have been developed that target especially wireless adhoc scenarios, such as the cross-layer approach in [138] or [209], where discovery

communication is embedded into the routing layer. Related to that, [189] presents a *Multicast DNS* (mDNS) extension to OLSR, which provides as a backend for several SD protocols. More information about SD in pervasive computing environments can be found in Section 2.2.

2.6 Conclusion

This concludes the chapter with background information and an overview of related work important to the work presented in subsequent chapters. We covered the topics of service oriented computing, explaining service discovery and its different architectures. We also introduced some of the most common service dependability metrics, such as availability and performability. Furthermore, responsiveness was defined which will be used later to evaluate SD dependability. We introduced models to evaluate these metrics with a special focus on wireless mesh networks. The importance of including a user-perceived view into analysis was motivated. In the the next chapter, the experiment framework *ExCovery* is presented, which was developed to support the extensive series of experiments that have been conducted during this work.

Chapter 3
ExCovery – A Framework for Distributed System Experiments

Abstract The experimentation framework *ExCovery* for dependability analysis of distributed processes is presented. It provides concepts that cover the description, execution, measurement and storage of experiments. These concepts foster transparency and repeatability of experiments for further sharing and comparison. *ExCovery* was specifically developed to support the experiments on *Service Discovery* (SD) conducted for this work and presented in Chapter 6. As such, a case study is provided to describe SD as *Experiment Process* (EP). A working prototype for IP networks runs on the *Distributed Embedded Systems* (DES) wireless testbed at the Freie Universität Berlin.

3.1 Introduction

Experiments are a fundamental part of science. They are needed when the system under evaluation is too complex to be analytically described and they serve to empirically validate hypotheses. Experiments also play an important role in computer science when supporting or refuting theories inferred from observations or mathematical models. With increasing complexity of computer systems and networks, exploratory experiments are themselves the source of such theories. However, due to the diverse focuses it remains difficult to repeat, classify, evaluate and compare results from different experiments. This is also true for the diverse experiments on *Service Discovery* (SD) responsiveness that have been carried out to gather the results presented and discussed in Chapter 6.

A consistent *Experimentation Environment* (EE) helps to unify related experiments and thus, greatly improves the impact of individual results. In this chapter, we present *ExCovery*, an EE to support research on the dependability of distributed processes. A formal description to specify experiments has been developed, which forms the basis of *ExCovery*. It allows for automatic checking, execution and additional features, such as visualization of experiments. *ExCovery* is expected to foster

repeatability and transparency by offering a unified experiment description, measurement mechanism and storage of results.

To support the wireless testbed experiments discussed in Chapter 6, a case study hows to use *ExCovery* for experiments on SD. To support and validate research on SD responsiveness, such as in [4, 6, 7, 9, 11], was the original goal of *ExCovery* development. This chapter covers the abstract description of experiments using *ExCovery*, concrete setups and results can be found in Chapters 6 and 8. It should further be noted that code listings presented in this chapter have been shortened for illustrative purposes. A collection of complete code listings can also be found in the appendix. *ExCovery* has been published under an open source license and can be found at [8].

The main contents of this chapter have previously been published in [9]. More details on the implementation and additional experiment results can be found in [242]. The rest of this chapter is structured as follows. Section 3.2 covers the topics of scientific experimentation and design of experiments. *ExCovery* is presented in Section 3.3, its concepts illustrated with examplary experiment description code. The description of SD as a specific experiment process follows in Section 3.6.3. An overview of the current *ExCovery* prototype implementation is given in Section 3.7. Section 3.8 concludes the chapter.

3.2 The Art of Experimentation

The subject of an experiment can be characterized as a black box process as illustrated in Figure 3.1. Results stem from observations of the outputs, or responses, of an experiment process under certain conditions. The conditions are defined by the values of the inputs to this process, also called factors, some of which are controllable by the experimenter while others are not. During an experiment it needs to be identified which factors exist and how they influence the responses. The latter can be done by manipulating one factor at a time or by manipulating multiple factors in a factorial experiment. Usually, experiments are run in series to capture the variation among multiple runs of the same experiment, while the optimal number of repetitions depends on the amount of variation [34]. Such series of controlled experiments are called experimental system [197].

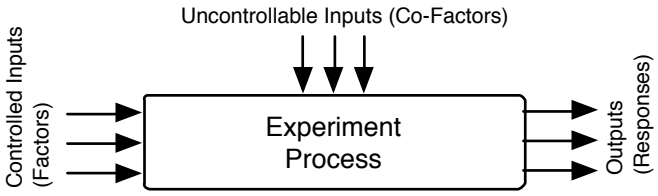


Fig. 3.1 Model of a generic experiment process

Experiments need to be reliable in a sense that they must be verifiable when repeated under similar conditions. A sound experiment design must therefore keep repetition in mind and the publication of experiment data must contain all necessary information to do so. Furthermore, experiments need to fulfill requirements for validity, a concept which is described in more detail in Section 3.2.4. A comprehensive guide on how to experiment and the caveats of running experiments can be found in [208].

3.2.1 Experiments in Computer Science

The role of experimentation in computer science has been the root of numerous debates, for example in [222, 88]. When computer science is seen as an engineering discipline, all subjects of inquiry are synthetic, as created by man. An often expressed opinion is that phenomena in computer science should thus be derived and explained following the construction of these subjects, instead of observing them as natural phenomena. However, the complexity of these subjects and their relationships in distributed system ever more often prohibits such deduction. Additionally, the subjects of computer science have become part of the world around us and interact with it, creating not entirely synthetic hybrids. As such, observation as in natural sciences can be justified just as the testing of hypotheses to assess and understand man-made systems.

When doing experiments in computer science, challenges exist mainly related to observation and repetition. It is not always clear which responses to observe and how to do that in such a way that the observation has no impact on the response itself. Modern computing systems provide ways to observe tens of thousands of parameters and observing them – measuring, recording and extracting – has to be done in the least invasive way. Also, repeating experiments can be difficult due to dependence on the original hardware and software platform. Where technology is proceeding at such rapid pace, experiments should be planned as independent as possible from specific hardware and software revisions they are running on, that is, if the revisions themselves are not the subject of experimentation.

3.2.2 Experiment Factors

Factors are the different sources of variation among individual experiments. Because there can be so many factors it is useful to classify them further according to [164].

Treatment factor Any substance or item whose effect on the data is to be studied [74]. It can have continuous values but usually has discrete levels that are to be applied to study its effect. Treatment factors can be further classified as *design factors*, intentionally varied during the experiment, *held-constant factors*, whose

impact is intentionally neglected by keeping their values fixed and *allowed-to-vary factors*, known to have a minor influence that can be compensated by applying randomization and replication (see also Section 3.2.4).

Nuisance factor On the other hand, nuisance factors have an unwanted effect on the output. They can be divided into controllable and uncontrollable nuisance factors. The former, often called *blocking factors*, can be fixed by the experimenter to reduce their impact on the response. The latter, also called *covariates*, cannot be set but their effect can be measured. Minimizing the impact of these effects on the experiment results can be done, for example, doing covariance analysis. Finally, *noise factors* cause random variations in the responses.

3.2.3 Experiment Design

A treatment is the entire description of what can be applied to the treatment factors of an *Experimental Unit* (EU), the smallest unit to which such treatment can be applied [24]. An *Observational Unit* (OU) then is the smallest unit on which a response will be measured. Experiment design defines which treatments are to be observed on which EUs [74]. Following [116], it can be divided into *treatment design*, a specification of the treatments used in an experiment, *error control design*, defining how specified treatments are to be applied to reduce unwanted variations, and *sampling and observation design*, which decides on the OUs and whether univariate or multivariate observations are to be taken.

Experiments tend to be time consuming. Proper experiment design strives to maximize the gained information per run by optimized structuring of factors and factor level variations over a required number of runs. This will be expressed in a higher precision of the response or in an increased significance of factor relationships. Reference experiment designs include completely randomized, randomized block, full factorial, fractional factorial, screening, response surface method, mixture, regression and space filling designs. A thorough explanation of experiment designs and for which objectives to apply them can be found in [74, 164].

3.2.4 Experiment Validity

Experiments need to fulfill requirements for internal and external validity. Internal validity demands that causal relationships between factors and responses are verified. External validity deals with the generalization of experiment results. To improve the validity of experiments, three interconnected principles are applied.

Replication Increases the number of experiment runs to be able to average out random errors and to collect data about the variation in responses. Care has to be taken in the selection of held-constant factors to allow for proper replication.

Blocking Means partitioning observations into groups, such that observations in each group are collected under similar conditions [74]. Statistical analysis requires that observations are independently distributed random variables [164].

Randomization The assignment of treatments to EUs as well as the temporal order and spacial choice of multiple runs takes care of the requirement of randomization. A design is called completely randomized when *all* treatment factors can be randomized. It generally strengthens the external validity if an experiment is run in a diversity of platforms (see also Section 3.2.5).

3.2.5 Experimentation Environment

To carry out the experiments on SD presented in Chapter 6 and 8, the *ExCovery* framework has been developed. The core of *ExCovery* 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. The main features of *ExCovery* are highlighted in this chapter and focus on the description of the conducted experiments and the analysis of their results. More background information on *ExCovery* can be found in [9] and [242]. To identify the demands *ExCovery* addresses, it is necessary to define what an *Experimentation Environment* (EE) is and what it needs to provide to an experimenter.

In the context of this work, an EE is defined as a set of tools with the purpose of describing, executing and evaluating experiments on a given subject, using a methodology specific to that subject. The actual setting in which the experiments and the EE are run is called platform. In general, an EE allows to perform a certain class of experiments in a controlled environment. It facilitates the identification and manipulation of factors and the observation of these manipulations on the responses. The amount of possible control depends on the characteristics of the EE. For covariates, the EE should allow to record them to be considered during analysis at a later stage. To foster the repeatability, correctness and transparency of experiments, an EE should use a well-defined description for setup, execution and evaluation of experiments. A common output format for measurements, logs and diverse meta information should be provided.

Last but not least, an EE should help an experimenter in designing and executing series of high quality experiments on the topic of interest. Experiments should be interruptible and resumable. For example, wireless testbeds allow users to experiment only in booked timeslots. When time is up an experiment should gracefully stop to be continued at a later stage. Experiments should execute as fast as possible. This is especially necessary for the experiments carried out in Chapter 6 as current SDPs use very long timeouts for some processes exchanging very few packets. This makes it difficult to execute enough repetitions to get reliable and valid results. Furthermore, in a distributed environment one has to consider robustness aspects for

the experimentation software itself. It should be able to handle node and connection failures, continuing the experiment where possible but stopping it where necessary, alerting the experimenter.

Network Experimentation Environments

ExCovery focuses on experiments to evaluate the dependability of distributed processes, such as network protocols that enable SD. Thus, it relies on network experimentation platforms like simulators and testbeds or mixed forms of both, for example virtualized testbeds or simulators with interfaces to real networks or real protocol implementations.

Simulators are software artifacts that simulate real-world processes by acting according to an abstract model. They can be discrete event-driven simulators, which calculate the state of the simulated object only when its state changes, or real-time simulators, which calculate the continuous behavior of the simulated object over time. There also exist mixed forms, for example, where an event-driven simulator is synchronized to a wall clock. While simulators have a perfect reproducibility of experiments, good scalability and generally a reduced execution time, their abstractions often struggle to capture the properties and behavior of real-world distributed systems [161]. A collection of caveats when simulating real-world effects with computer systems is explained in [200]. In this work, when reverting to simulators we used NS-3 [115, 13] which is common in the community.

A testbed on the other hand is made of real network devices. They allow less control over factors but measurements are the result of a realistic interplay of factors. As such, testbeds allow to represent a specific environment, for example a wireless mesh or a large scale internet network, very well. Testbeds usually provide means to manage experiment schedules and setup, data acquisition and storage. An approach to unify generic network experimentation across simulators, emulators and testbeds is proposed with NEPI [134, 192], an integration framework for network experimentation which creates a common model of experimentation that can be applied to many physical testbeds as well as to the NS-3 simulator and the *netns* emulator. Another approach focused on testbeds is Weevil [240], a model-driven experiment framework that was developed to automate large-scale experiments on distributed testbeds, with several enhancements regarding repeatability and scenario definition proposed in [241]. However, none of those approaches provides the full chain of description, execution, measurement and storage of experiments as does *ExCovery* with comparably low requirements on the platform they run on (see Section 3.3). *ExCovery* additionally has a special focus on time behavior analysis of network protocols, such as SDPs.

Choosing a wireless testbed which fulfills the requirements is no trivial task as behavior in reality can deviate dramatically from expected behavior. This fact is convincingly demonstrated for current testbeds in [161]. To carry out the experiments needed for this work in wireless networks, we employed the *Distributed Embedded Systems* (DES) testbed at *Freie Universität Berlin* (FUB). The DES testbed provided

the best compromise among usability, availability and versatility. Nodes run a recent version of the Linux operating system which facilitates software development and deployment. Second, the DES testbed is regionally close and the staff was always ready to provide technical help in unexpected situations. Third, the design of the testbed and its layout allow the generation of a manifold of different scenarios to experiment in. Comprehensive background information about the DES testbed can be found in [100, 101, 36]. In the last years, however, many other wireless testbeds have become available to the community. This is why *ExCovery* supports the DES testbed as a first platform, but is not restricted to it, it can be ported to other platforms for comparison in the future.

3.3 The Experimentation Environment *ExCovery*

We will now present the main concepts of the proposed experimentation environment *ExCovery* with its core, the formal abstract description of an experiment using the *Extensible Markup Language* (XML). An XML schema of such a description is provided with the framework code on request. A version specific to SD as an experiment process, which was utilized for the experiments in Chapters 6 and 8 can be found in Appendix A. The description includes definitions of the experiment with its input factors, the process to be examined, of fault injections or manipulations and diverse platform specific and informative declarations. *ExCovery* further provides a unified measurement concept that determines which and how data are stored for later analysis. An overview of the different concepts and the experiment work flow is illustrated in Figure 3.2, which shows the different steps when experimenting with *ExCovery*.

In the first preparation step, the experiment is designed by the experimenter following guidelines as mentioned in Section 3.2.3. The individual descriptions will be explained in Section 3.5. Second, platform setup is necessary to prepare the translation of descriptions to the target platform, such as a simulator or a testbed. This could include the deployment of executables and configuration files. The experiment is then executed by the experiment master as specified in the description. Each run is a sequence of actions performed on the participating nodes, described as the main process under evaluation and a set of injected faults or manipulations. The master and all nodes monitor and record dedicated parameters during each run, such as raw packet captures and the complete temporal sequence of *Experiment Process* (EP) actions and events. These data will be saved in a temporary location locally on the nodes. After experiment execution, the recorded data are collected and conditioned so that a common time base for all actions, events and packet measurements is established. Finally, data are stored into a single results database that contains all conditioned measurement data, created log files and the complete experiment plan with the exact sequence of treatments, as described in Sections 3.4 and 3.4.2).

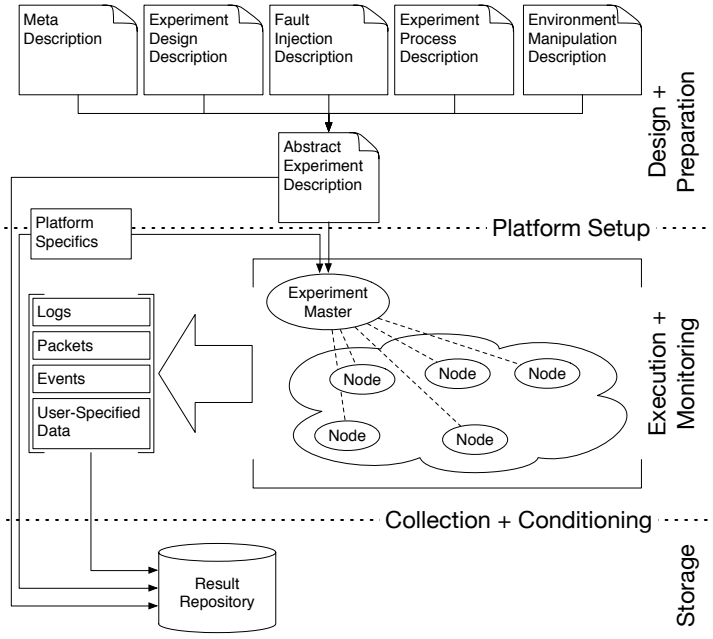


Fig. 3.2 Overview of *ExCovery* concepts and experiment workflow. [9]

Platform Requirements

To integrate a specific target platform in *ExCovery*, it must support several features. Most of the features are needed to establish a controllable environment or to compensate for missing control and to allow detailed measurements. As such, these are mainly issues for testbeds, such as the DES wireless testbed. Simulators generally can be integrated with less effort.

Experiment Management

There must be a separate, non-interfering and reliable communication channel between the experiment master and the nodes participating in the EP. In simulators, this is usually provided by a software interface while testbeds need to possess physically separate network interfaces. During experiments, full and privileged access to all nodes is mandatory. The platform needs to cleanly separate concerns of multiple users.

Connection Control

Full control over the network connections of the individual network nodes is needed. Network interfaces need to support activation and deactivation. Furthermore, it needs to be possible to manipulate packets sent over these interfaces based on defined rules, such as dropping, delaying, reordering, and modifying their content.

Measurement

There must be methods to capture packets with their exact local timestamps and their complete and unaltered content. To facilitate a comprehensive subsequent analysis, a packet tracking mechanism is required. In testbeds, this means tracking the routes of packets hop by hop, or attaching unique identifiers to packets [253]. Finally, the platform needs to support time synchronization among all participating nodes and quantification of the synchronization error.

3.4 Measurement and Storage Concept

This section clarifies the basic observations that are possible using *ExCovery*, how they can be observed and how this can help to unify related experiments. *ExCovery* follows the principle of collecting as much data as possible to support diverse analyses on the same experiment data at later time, emphasizing reusability and repeatability.

3.4.1 Measurement Data

What data is recorded by *ExCovery* and how is specified in the following section. Basic recordable data are the results of protocol operations as reflected by state changes on the participants and network messages sent among participants. *ExCovery* supports a plugin concept to extend these data with custom measurements [242].

Events

State changes on nodes in the context of *ExCovery* reflect *events* and occur, for example, when an experiment run is initialized or when a fault injection is started. Also any defined states of the EP signify events, such as *Discovery Request Sent* or *Discovery Response Received* in the case of SD. Events are associated with the node on which they occur. They contain a local time stamp and may have additional parameters, such as the identifier of the initialized experiment run. To control

the experiment execution, nodes can be synchronized using global events (see also Section 3.6).

Packets

Packets are the basic communication data of network protocols. As opposed to events, single packets are not easily identified: Their location changes as they traverse the network, retransmissions and network loops complicate the correct localization at any given time. Packets are recorded to facilitate verification of the recorded event list and to derive statistical connection parameters. A measured packet consists of a time stamp, representing the local occurrence of that packet, a unique identifier, a source and destination network address and the packet content itself.

Time

Events and packets have local time stamps of the node they were measured on. *ExCovery* defines mandatory measurements to be done before each run to estimate the time difference of each participant to a reference clock. This allows to construct a valid global time line of events and packets, avoiding causal conflicts due to local clocks deviating between experiment runs. This is especially needed when time synchronization protocols cannot be used because they interfere with the experiment processes.

Topology

To improve repeatability, an extensive description of the network topology is measured. This description includes the local information on each node about its direct neighbors and any other known nodes as well as the links between them. For each link, the quality as reflected in the *Expected Transmission Count* (ETX) metric is recorded (see Section 7.5). This measurement is done at the end of each repetition within a series of experiments. This way, the impact of fault injection or load generation will also be reflected in the routing layer quality metrics. Using these measurements, *ExCovery* allows to track network dynamics over extended periods of time.

3.4.2 Storage and Conditioning

ExCovery envisions four levels of storage for experiments, with defined data structures. This allows reusable data access functions among experiments. For more de-

tailed information on data conditioning and the current implementation of the storage concept refer to [9], [242].

The first storage level is the abstract experiment description itself, stored in an XML document. This document can be exchanged and loaded for execution and analysis. The second level is an intermediate storage for all concrete experiment data: experiment results and the software artifacts used during execution. The second level storage implementation is flexible and depends mainly on the capabilities of the execution platform. *ExCovery* does not impose a specific storage mechanism but requires that it is accessible during the subsequent collection and conditioning phase. Currently, *ExCovery* uses a special hierarchy on a file system to store second level data. Each node has its own temporary storage for recorded data, organized into data belonging to single runs and to the entire experiment. Each log file and measurement is stored corresponding to a run identifier and associated to the originating node. Time synchronization measurements are stored on the experiment master. Plugins have a separate storage location on the node where the custom measurements are done.

On the way to the third storage level, data are conditioned by first evaluating the synchronization measurements (see Section 3.4.1) and unifying the time base of all second level measurements. Then, the event list and captured packets are split up into single data items. Data from the second level plus the experiment description are then stored into a single package on the third level, which represents one complete experiment and is preferably stored as a database to unifiy and accelerate data access and extraction methods. Facilitating exchange of experiments, *ExCovery* currently stores the third level in a file based relational SQLite database. This allows to use the included XML description to recreate the exact run sequence of the experiment and to use SQL to access data stored within the database.

Table 3.1 shows a subset of the tables and their attributes at the third level. Stored data is classified as either experiment or run based. Run-based data is stored for every run of the experiment, while experiment-based data is only stored once per experiment. All run based measurements are identified by a run identifier (ID) and a node ID, which form the primary key of the related entities. A run ID represents one run and a node ID specifies one concrete node by its name.

Table 3.1 Tables and Attributes of Current Storage Concept

Table	Attributes
ExperimentInfo	ExpXML, EEVersion, Name, Comment
Logs	NodeID, Log
EEFiles	ID, File
ExperimentMeasurements	ID, NodeID, Name, Content
RunInfos	RunID, NodeID, StartTime, TimeDiff
ExtraRunMeasurements	RunID, NodeID, Name, Content
Packets	RunID, NodeID, commonTime, srcNodeID, Data
Events	RunID, NodeID, commonTime, Type, Parameters

The table `ExperimentInfo` represents the experiment as a whole and contains only one tuple made of the abstract experiment description, the version of *ExCovery* and a descriptive name and comment. `Logs` contains all raw log files and `EEFiles` the used *ExCovery* executables. In `ExperimentMeasurements`, specific named measurements are stored that are done once per experiment. As for run based data, `RunInfos` contains for each run and node the start time of the run and the offset of the node clock to the reference clock. Custom measurements are stored in `ExtraRunMeasurements`. The table `Packets` contains for each packet the common time stamp of detection, its originating node and the raw packet data. The `Events` table lists all recorded events and their parameters, identified by the run, the originating node and a common time stamp. This schema represents a preliminary approach to store data. Several future improvements are possible, for example by using a dimensional database model to store experiments in a data warehouse structure.

The fourth level describes the integration of multiple experiments into a single repository to facilitate comparison and analysis covering multiple experiments. To date, this level is not yet implemented in *ExCovery*.

3.5 Abstract Experiment Description

The core of experiments carried out with *ExCovery* consists of an abstract description in the *Extensible Markup Language* (XML). For reasons of brevity, this section explains only the most important settings included in the experiment description. A more thorough listing can be found in [9], [242]. 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, the time it was started or, in the case of this work, the SDP to be used. Figure 3.3 shows a rudimentary general section of an experiment description. Four abstract nodes are to be mapped by the processes described later. Parameters further classifying the experiment process are advisable as they shall help in recreating experiment conditions and in making them more transparent. For basic experiment classification, two parameters describing the discovery architecture and protocol are defined as key-value pairs. More information about SD architectures and protocols can be found in Section 2.2.

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, for example 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 3.4 shows an exemplary platform definition with four actor nodes to carry out the SD operation and five environment nodes (see also Section 3.5.1).

Factors The factor definition contains a list of factors and factor levels whose effect on the results is to be studied. The order of the factors defines how the factor

level combinations are applied, with the last factor changing its level most often. Concrete actor nodes are mapped to abstract roles of the EP, such as requesters and responders in the case of discovery. Furthermore, 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 in Figure 3.5. More information on the interpretation of factors when executing experiments can be found in Section 3.5.2.

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 as detailed in Section 3.6. Due to space constraints, no full description of the processes for the discovery actors and environment nodes is shown here. The full descriptions can be found in Appendix B.

```
<experiment_name>just a name</experiment_name>
<totalnodes>9</totalnodes>
<abstractnodes>
  <abstractnode id="R0"></abstractnode>
  <abstractnode id="P0"></abstractnode>
  <abstractnode id="P1"></abstractnode>
  <abstractnode id="P2"></abstractnode>
</abstractnodes>
<sd_level>user</sd_level>
<sd_protocol>zeroconf</sd_protocol> <!-- may also be "slp" -->
<sd_arch>2-party</sd_arch>
```

Fig. 3.3 Rudimentary general section of an experiment description with informative parameters about discovery process and listing of abstract nodes.

In the following, the individual elements of the abstract description are explained. How exactly they may be used within an experiment description is defined in the XML schema in Appendix A.

Factor Part of the *treatment* applied to the EU. Consists of a set of levels. Depending on the design, levels are applied *One aFter AnoTher* (OFAT) or randomized.

List of factors Contains all factors used, sorted. In an OFAT design the first factor varies least often during execution while the last factor changes every run.

Level Concrete value a factor can take, as input variable to the sub-processes of each run. Levels can be of different types. As such, they can control type and duration of fault injections (see Section 3.6.1) or represent mappings of abstract nodes to actors.

Set of levels All levels that should be applied during the experiment. Order of application is determined by the factor definition. There is only a single level if the factor should be kept constant during the whole experiment.

Replication factor Parameter defining an integer number of replications to be done for each treatment.

```
<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. 3.4 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. 3.5 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.

Abstract node Actor of the EP or of a node specific fault injector. Identified by a node identifier, such as a unique host name.

Environment node A node not participating as actor in any node specific process. Used e.g. to generate load.

Actor description Process prototype to be executed on one specific actor of the EP. Each abstract node is mapped to one actor description, multiple abstract nodes can instantiate the same actor description.

Experiment process Experiment operation that is to be executed and measured. Consists of actions performed on multiple nodes, synchronized by flow control functions that wait for a certain time or for certain events.

Manipulation process Main part of the *treatment*. Similar to EPs, represents a sequence of faults or impairments that should happen on a node.

Environment manipulation process Like experiment and manipulation processes but not node specific. Controls environment manipulations such as traffic generation.

3.5.1 Platform Definition

To instantiate an abstract experiment description on a concrete platform, such as a wireless testbed, a definition of this platform is required. In this definition, abstract and environment nodes are mapped to concrete, physical nodes of the experiment platform. *ExCovery* identifies nodes by their host name and IP address. The host name should be constant during an experiment run. When an IP addresse changes due to reconfiguration of a network interface, for example after an injection of such a fault, an event is generated to signal this.

Figure 3.4 illustrates a compact version of a platform specification. Four actor nodes and five environment nodes exist. Actor nodes map to an abstract node id that has been previously defined (see Figure 3.3). All nodes have a unique identifier and a network address that can later be used to analyze the recorded event and packet lists.

3.5.2 Experiment Execution from Factor Definition

To execute the overall experiment and its individual runs from the abstract description, *ExCovery* generates treatment plans from replications, the factors and their levels. Plans are OFAT if no custom factor level variation plan is given. The various random values used in *ExCovery* are generated using pseudo-random generators. This allows for perfect repeatability of random sequences when initialized with the same seed. Which seed is used for initialization is clearly defined in the experiment description so that all random sequences can be reproduced. Software for the design of experiments exists that can help with the generation of such treatment plans, this

is why a specific plan generator is outside the scope of this work. However, *ExCovery* supports plan generators by providing all necessities, such as factor, level and repetition representations.

ExCovery uses four internal functions for the experiment flow. Experiments are initialized by calling `experiment_init` on every participant, which takes care of the necessary preparations before all individual experiment runs. Each run is then initialized by `run_init`. There further exist the respective exit functions `run_exit` and `experiment_exit`.

Figure 3.5 shows the definition of several factors and their levels. It maps the abstract nodes listed in the general section of the description (see Figure 3.3) to actor roles `requester` and `responder`, 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.

Experiment Phases

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. Each run consists of a preparation phase, an execution or measurement phase which is the core of the experiment and a cleanup phase. During preparation, the whole environment of the EP must be reset to a defined initial working condition. Software agents are initialized. In testbeds, for example, network packets generated in previous runs must be dropped on all participants. Preliminary measurements can be done to compensate for incomplete control over the environment, such as clock offsets for all participants. 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 timestamp and written to a database. The database represents one full experiment and can be shared to allow transparent reusability and repeatability. It 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. All steps will be repeated during each run, this has to be considered when estimating the total time an experiment needs to finish.

3.6 Process Description

ExCovery provides common mechanisms to control execution of the defined processes. Two types of processes can be differentiated, depending on whether they

relate to abstract nodes or to the environment. Abstract node processes are mapped to real nodes during experiment execution, such as protocol actions or fault injection processes. Environment processes are performed by all nodes, such as dropping packets on all network interfaces to reset the environment. Every process is described as a sequence of actions. Processes run concurrently so one needs to consider timing and desired or necessary dependencies. *ExCovery* defines methods for synchronization to provide basic flow control.

wait_for_time Process waits for a fixed delay in seconds.

wait_for_event Process waits until the specified event is registered on any participant. An event can be specified by its name, location and any of its parameters. The location is either a single abstract node or a subset of nodes specified by an actor role. Event parameters can be of diverse types. If omitted, they default to "any". A time-out in seconds can be set.

wait_marker Creates a time stamp used by the next `wait_for_event` call, which considers only events occurring after that time stamp.

event_flag Used to create local events to let process actions depend directly on each other.

Besides these flow control functions, there are process specific actions, environment actions and manipulation actions. Each action can have a list of parameters, allowing the description of manifold scenarios. In Section 3.6.3, SD as an EP is described to illustrate this. Manipulation processes are described in Section 3.6.2.

Figure 3.6 shows a code fragment where the different processes are defined, without the actual sequences of actions that will be described later. Among the node processes, the role `responder` is defined and as possible actor nodes, the abstract nodes `fact_nodes` from the factor list are referenced. Environment processes do not need a definition of nodes.

```
<processes max_run_time=120>
  <env_process>
    <!-- list of environment process actions -->
  </env_process>
  <node_processes name="...">
    <process_parameters>
      <actor_node_map>
        <factorref id="fact_nodes"/>
      </actor_node_map>
    </process_parameters>
    <actor id="responder" name="SM">
      <!-- list of node process actions -->
    </actor>
  </node_processes>
</processes>
```

Fig. 3.6 Template for the description of node and environment processes.

3.6.1 Manipulation Processes

ExCovery has a concept for intentional manipulations done on participant nodes and on their network environment. Manipulations cover direct fault injections that cause failures in a target area. Fault provocation is used when direct injection is not desirable or possible and characterizes actions that are known to provoke faults in a target area. The main faults considered are communication faults. *ExCovery* provides a simplified fault model to allow for the description of basic fault behavior with a set of common parameters. Fault injection processes can have common parameters describing their temporal behavior: *duration*, *ratio* and *randomseed*.

Duration Specifies the amount of time a fault should be applied to the target.
Ratio Specifies a percentage of a given period in which a fault is active. The fault is active in one continuous block
Randomseed The fault activation time in the period is chosen randomly using this parameter as seed.

Fault Injections

Mechanisms for fault injection are explained in the following. In addition to the common parameters, injections can have custom parameters to further define their behavior. It should be noted that all injected faults add up to already existing communication faults in the target platform. Also, whenever the term *packet* is used it refers to packets belonging to the EP.

Interface fault No messages are transmitted or received on the specified interface in the specified direction as long as this fault is active. Direction can be receive, transmit, both, or chosen randomly.
Message loss Defines probability for every packet to be dropped. Direction is analogous to the interface fault.
Message delay Applies a constant delay to every packet.
Path loss, path delay Path loss and delay are message loss and delay faults, selectively affecting only the communication between the target and a given second node.

Environment Manipulations

Environment manipulations are applied on a global level and involve possibly all specified environment nodes. Manipulations include the previously defined fault injections. Additionally, the following manipulations can be applied.

Traffic generator Creates network load between a given number of node pairs. Each pair bidirectionally communicates at a given data rate (see also Figure 3.5).

Pairs can be randomly chosen from the acting nodes, non-acting nodes or all nodes. Pairs vary between runs as determined by a switch amount parameter.

Drop all packets All experiment nodes stop receiving, sending and forwarding EP packets.

Every fault injection and environment manipulation but the traffic generator is started only once and without a given duration, needs to be explicitly stopped. Given is just the default list supported by *ExCovery*. *ExCovery* provides also a generic function with an arbitrary list of parameters that are given to the acting nodes to be executed [242]. However, an experimenter should preferably extend *ExCovery* by defining a plugin with new functions and their implementation.

3.6.2 Manipulation Process Description

Manipulation processes are defined in the experiment description as a series of actions and events. This list is executed sequentially and can contain flow control functions as described in Section 3.6. A node manipulation process is created for each abstract node it is specified for while the environment manipulation process is implicitely supported on all nodes. The specific actions activate or deactivate the faults and manipulations. One event is generated by each action to signal its start or stop, respectively. Parameters of these actions can be constant or varied during experiment execution. Variation is realized by references to factors instead of fixed values.

Start interface fault Starts interface fault with the given parameters. Generates `fault_start_interface` event.

Start fault message loss Starts message loss fault with the given parameters. Generates `fault_start_messageloss` event.

Start fault message delay Starts message delay fault with the given parameters. Generates `fault_start_messagedelay` event.

Start traffic generator Traffic generator is started by this action.

Start dropping all packets Starts the corresponding manipulation in order to purge all experiment process packets from the network.

The manipulation actions can be used to extend the EP description or to define separate manipulation processes. This depends on whether manipulations shall be synchronous with the EP or autonomous. Figure 3.7 shows a shortened listing of a traffic generation process. After generating a `ready_to_init` event, it uses the factors from Figure 3.5 to choose and configure traffic generation by a set of environment nodes, switching one pair of nodes in every run. The manipulation remains active until `done` is registered.

```
<env_process>
  <env_actions>
    <event_flag>
      <value>ready_to_init</value>
    </event_flag>
    <env_traffic_start>
      <bw>
        <factorref id="fact_bw" />
      </bw>
      <choice>0</choice>
      <!-- this causes identical randomization in replications -->
      <random_switch_amount>1</random_switch_amount>
      <random_switch_seed>
        <factorref id="fact_replication_id" />
      </random_switch_seed>
      <random_pairs>
        <factorref id="fact_pairs" />
      </random_pairs>
      <random_seed>
        <factorref id="fact_pairs"/>
      </random_seed>
    </env_traffic_start>
    <wait_for_event>
      <event_dependency>done</event_dependency>
    </wait_for_event>
    <env_traffic_stop />
  </env_actions>
</env_process>
```

Fig. 3.7 Illustrative example of environment process for traffic generation.

3.6.3 Experiment Process Description

Remaining is a description of the main experiment process, whose events and packets are to be observed and measured. The description provides a temporal and causal sequence of actions on the participating nodes as introduced in Section 2.2, facilitating flow control functions from Section 3.6. In line with the topic of this work, we will explain how to describe generic SD as an EP to be used within *ExCovery*, to support the experiments in Chapters 6 and 8. The description can contain actors for service providers, service clients and service registries. For reasons of brevity, these roles will be referred to as SM, SU and SCM, which is in line with the terminology used in [70]. For each actor a number of instances can be created to represent all participants of the SD process.

The model developed in [70] defines a set of main operations for a generic SD process, namely “Configuration Discovery and Monitoring”, “Registrations and Extension”, “Service-Description Discovery and Monitoring”, and “Variable Discovery and Monitoring”. These will be considered in the abstract SD process description, with an optional list of parameters to define specific variants. The description does not intend to model a *Service Discovery Protocol* (SDP) specific behavior in detail, but to give an abstract description of a SD scenario. The details of executing the description are left to the SDP implementation, so that multiple implementations which adhere to the same SD concepts can be compared in experiments. However,

the executing SDPs are allowed to generate custom events which will be recorded by *ExCovery*. Actions that can be executed on participating SD nodes are described as follows.

Init SD Mandatory action to allow participation of a node in the SD. Represents "Configuration Discovery and Monitoring". Discoverable items such and scopes and SCMs are discovered and a unique identity is established on each node. This action reads as parameter the role as either SCM, SU or SM. Optional custom parameters further configure the used SDP. When the SCM parameter is used, the node generates a `scm_started` event. If an SM registers its service on an SCM node, a `scm_registration_add` event is generated with the registering node's identification as parameter. Analogously, when a registration is revoked or changed, the respective events `scm_registration_del` and `scm_registration_upd` are generated. In a hybrid architecture, SU and SM agents keep looking for SCMs and emit `scm_found` events when a SCM has been discovered. When SD initialization is complete, `sd_init_done` is emitted.

Exit SD Stops a previously started role and all assigned searches and publishings, emitting `sd_exit_done` upon completion. To participate again in the SD process, a node needs to re-run init.

Start searching SU and SM nodes initiate a continuous SD process for a given service type, generating the event `sd_start_search`. Refers to the group of "Service-Description Discovery and Monitoring" functions. *ExCovery* does not distinguish among *passive*, *aggressive*, or *directed discovery* (using an SCM). A service is considered discovered when its complete description has been received. The event `sd_service_add` will be emitted with the found service's identifier as parameter. Analogously, when a service becomes unavailable, the event `sd_service_del` is generated

Stop searching A previously started search is stopped and event `sd_stop_search` generated. Includes de-registration of any notification request on SCMs.

Start publishing Starts publishing an instance of a given service type, generating a `sd_start_publish` event. Refers to the group of "Registrations and Extension" functions, such as registration on a SCM.

Stop publishing Gracefully stops publishing a given service type. Comprises further actions like aggressively sending revocation messages or SCM de-registration. Generates `sd_stop_publish` event upon completion.

Update publication Updates a previously published service description. Covers underlying functions related to registration on SCMs. Generates an event `sd_service_upd` with the service identifier as parameter before the update is executed.

An example SD scenario depicted in Figure 3.8 shows a single active SD in a two-party architecture with a timeline for each actor SU1 and SM1. Actions are shown as white circles, events as black circles. Unlabeled events inherit the label of the preceding action. In the preparation phase, SU and SM initialize themselves. This phase ends a fixed time after `sd_start_publish` from SM1 is registered, to let unsolicited announcements of SM1 pass. SU1 then starts a search, beginning the

execution phase. After a time t_R the service is discovered and `sd_service_add` is generated on SU1. The scenario finishes here, in the clean-up phase searches and publications are stopped and the SD system shut down.

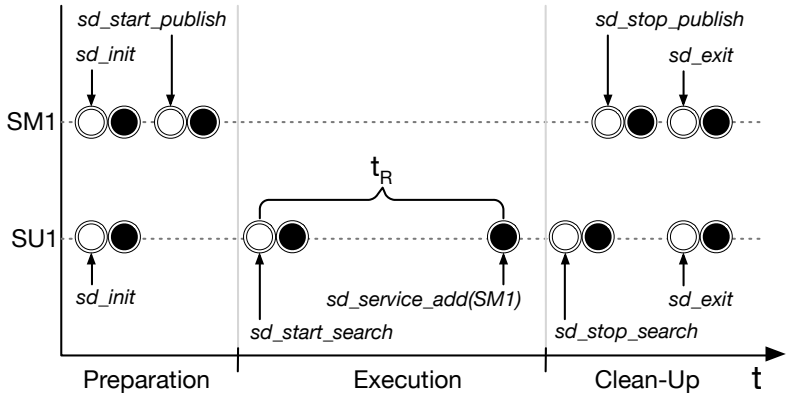


Fig. 3.8 Visualization of a one-shot discovery process.

Figures 3.9 and 3.10 show descriptions of two-party SD processes for SU and SM roles using the introduced actions and events. The SM role in Figure 3.9 basically starts publishing and continues until a `done` event is registered. The SU role in Figure 3.10 waits first for all SMs to emit their `sd_start_publish` event, next for the environment to register the `ready_to_init` event. It will then start searching and finish either when all SMs have been discovered, having generated their respective `sd_service_add` events or when the deadline of 30 seconds has been reached. In either case, `done` is generated and the clean-up phase begins.

```
<actor id="actor0" name="SM">
  <sd_actions>
    <sd_init />
    <sd_start_publish />
    <wait_for_event>
      <event_dependency>"done"</event_dependency>
    </wait_for_event>
    <sd_stop_publish />
    <sd_exit />
  </sd_actions>
</actor>
```

Fig. 3.9 SD process in a two-party architecture. Publisher role.

The code listings of all processes in this section have been deliberately shortened for reasons of readability. Full versions can be found in Appendix B.

```
<actor id="actor1" name="SU">
 <sd_actions>
  <wait_for_event>
   <from_dependency>
    <node actor="actor0" instance="all"/>
   </from_dependency>
   <event_dependency>"sd_start_publish"
    </event_dependency>
  </wait_for_event>
  <wait_for_event>
   <event_dependency>"ready_to_init"
    </event_dependency>
  </wait_for_event>
  <sd_init />
  <wait_marker />
  <sd_start_search />
  <wait_for_event>
   <from_dependency><node actor="actor1" instance="all"/>
   </from_dependency>
   <event_dependency>"sd_service_add"</event_dependency>
   <param_dependency><node actor="actor0" instance="all"/>
   </param_dependency>
   <timeout>"30"</timeout>
  </wait_for_event>
  <event_flag><value>"done"</value></event_flag>
  <sd_stop_search />
  <sd_exit />
 </sd_actions>
</actor>
```

Fig. 3.10 SD processes in a two-party architecture. Requester role.

3.7 Prototype Implementation

An *ExCovery* prototype has been implemented using the Python programming language [191] with the aim of being reusable on diverse platforms. It abstracts the handling of the XML experiment description and the resulting run sequence and parameter variations in separate classes that can be instantiated by programs to analyze, visualize, trace or export experiment related data.

Some requirements must be met to reuse this implementation in addition to all requirements mentioned in Section 3.3. For testbed platforms, several additional requirements for the software packages exist. Nodes need to run the Linux operating system with a kernel of version 2.6.34 or higher, which was released on May 16, 2010. Packet capturing software for the *pcap* format should be available as well as the components of the netfilter software. Privileged access is mandatory on all participating nodes. The current *ExCovery* implementation further depends on the software tools *tc*, *iptables*, *route*, *kill*, *killall*, *traceroute* and *tshark*. All of these are broadly used standard tools on Linux systems.

As the first platform, *ExCovery* supports the wireless DES testbed at Freie Universität Berlin [36]. An implementation for the SD process in Section 3.6.3 exists using the Zeroconf SDP [55, 54, 53] and the *Service Location Protocol* (SLP) [103, 106]. This section gives a quick overview of the prototype, a comprehensive description of the implementation can be found in [242, 61].

Software Components

In accordance to the developed concepts the prototype is composed of one controlling entity (master) and a set of controlled entities (nodes) as depicted in Figures 3.2 and 3.11. Master and nodes are connected in a centralized client-server architecture with a dedicated communication channel. They communicate synchronously using *Extensible Markup Language Remote Procedure Calls* (XML-RPC) [244].

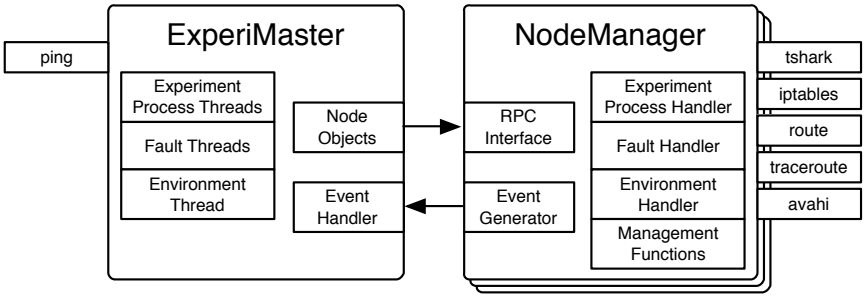


Fig. 3.11 Execution components of the provided implementation, here only for Zeroconf.

The controlling *ExperiMaster* maintains a list of objects corresponding to the active nodes in the experiment on which actions will be executed. A node object presents the functions of one node to the master program via XMLRPC and uses locking to allow only one access at a time. Which action is executed at which time is specified in process descriptions loaded from the experiment description file. The master creates an EP thread and a fault thread for each abstract node in the description. A single thread is created for the environment manipulations. The actions performed by this thread and the management actions performed by the main program can be executed concurrently on all nodes.

The *NodeManager* is the central component of the nodes participating in experiments. It handles *Remote Procedure Calls* (RPCs) from *ExperiMaster*. Basically, the *NodeManager* provides an interface to the *ExperiMaster*. Components on a node use the event generator to signal the occurrence of events, as defined in Section 3.4.1. Basic procedures exposed via RPC are actions for management, fault injection, environment manipulation and the EP actions as defined in Sections 3.3 and 3.6.3. The implementation of these functions can be delegated to sub-components, e.g., the EP actions in the context of this work refer to SD actions that are implemented by the Avahi [22] software package in the case of Zeroconf. For SLP, the reference implementation provided by OpenSLP project group [176] is used by means of a Python wrapper.

To allow analysis of properties outside the scope of the *ExCovery* processes, for example packet loss and delay, a network packet tagger is provided. It remains running in the background on each node. The tagger adds an option to the header of each selected IP packet and writes a 16 bit identifier to it, incrementing the identifier with

each packet. This approach can cause problems when IP packets are fragmented, due to the packet size being larger than the *Maximum Transmission Unit* (MTU) of the network. In the context of this work, however, SD packet sizes used for testing are generally far below the MTU. Additionally, *ExCovery* includes a set of Python scripts to collect, condition and store experiment results in a database.

The presented concept and implementation generally support multiple SDPs. They need to provide a Linux implementation which provides an interface to fundamental SDP operations, as represented by the actions in Section 3.6.3. For the prototype, the Zeroconf SDP suite Avahi [22] was modified to allow the association of request and response pairs. This allows analysis of response times not only at the SD operation level (t_R in Figure 3.8) but at the level of individual SD request and response packets, which by default is not supported in Zeroconf SDPs. These response times are different when packets are being lost and requests are retried. Zeroconf is based on multicast DNS [54] and the ID field of the used DNS records was chosen to identify responses to a request. This field is suitable and least invasive because it "must be ignored on reception" and queries "should not" use the ID field. The identifier is initialised with a random value in each run and then incremented for each retransmission and each new query. A set of functions exist for extraction and analysis of event and packet based metrics.

A set of specific analysis functions was added to the framework code as presented in [9] to support the results presented in Sections 6.5 and 6.4. Additionally, diverse enhancements were developed to improve monitoring during execution of experiments, given the long durations of the individual experiments. Finally, the framework code for the execution of experiments was redesigned to reduce the duration of experiments, especially when they are resumed after partial completion. Since they are not the focus of this work, the implementation of these changes will not be discussed in detail.

3.8 Conclusion

This concludes the presentation of the *ExCovery* framework. *ExCovery* has been tried and refined in a manifold of SD dependability experiments over the last two years that were carried out on the wireless DES testbed at Freie Universität Berlin. Specifically, *ExCovery* was developed for the experiments on *Service Discovery* (SD), which are presented in Chapters 6 and 8. The described abstract description of SD as experiment process was used in those experiments.

Part II
User-Perceived Service Availability

Service-Oriented Architecture (SOA) has emerged as an approach to master growing system complexity by proposing services as basic building elements of system design. However, it remains difficult to evaluate dependability of such distributed and heterogeneous functionality as it depends highly on the properties of the enabling *Information and Communications Technology* (ICT) infrastructure. This is especially true for the user-perceived dependability of a specific pair service client and provider as every pair may utilize different ICT components.

In Chapter 4, we provide a model for the description of ICT components and their non-functional properties based on the *Unified Modeling Language* (UML). Given a service description, a network topology model and a pair service client and provider, we propose a methodology to automatically identify relevant ICT components and generate a *User-Perceived Service Infrastructure Model* (UPSIM). We further provide a model-driven methodology to automatically create availability models of such views, called *User-Perceived Service Availability Models* (UPSAM). The methodology supports the generation of different availability models, exemplarily providing reliability block diagrams and fault-trees. These can be used to calculate user-perceived steady-state and instantaneous service availability. For instantaneous availability evaluation, the age of the ICT components, the order and time of their usage during service provision are taken into account, providing a proper concept of service time.

Chapter 5 demonstrates the feasibility of the methodology by applying it to parts of the service network infrastructure at *University of Lugano* (USI), Switzerland. We then show how this methodology can be used to facilitate user-perceived service dependability analysis. Using the UPSAM, we calculate the availability of an exemplary mail service in diverse scenarios. We conclude this part by discussing further applications of the methodology and explaining how to combine it with the service discovery models in Chapter 7.

Chapter 4
Modeling User-Perceived Service Dependability

Abstract This chapter presents an approach to generate models for the evaluation of user-perceived service properties. It provides a model for the description of a network of ICT components and their non-functional properties based on the *Unified Modeling Language* (UML). Given a set of input models that describe the service network topology, its services and actors, the proposed methodology automatically identifies relevant ICT components and generates a *User-Perceived Service Infrastructure Model* (UPSIM) which can be further transformed into *User-Perceived Service Availability Models* (UPSAMs) for steady-state and instantaneous availability analysis. So far, the methodology can create *Reliability Block Diagrams* (RBDs) and *Fault Trees* (FTs). How these models can be utilized is demonstrated in case studies in Chapter 5.

4.1 Introduction

This chapter builds on and combines previously published work on the topic which is described in the following. In order to assess the service dependability from different user perspectives, an extraction of relevant network parts is presented in [2]. Given a model of the network topology, a service description and a pair service requester and provider, a model-to-model transformation is applied to obtain a *User-Perceived Service Infrastructure Model* (UPSIM) as in Definition 1.2. The approach in [2] uses a subset of *Unified Modeling Language* (UML) [170] elements as well as UML profiles and stereotypes [171] to impose specific dependability-related attributes to ICT components. The ICT infrastructure and services are modeled independently using UML object and activity diagrams, respectively. Then, a mechanism is used to project the properties of ICT components to services through an XML mapping that correlates their respective models. The methodology in [2] is based on the work by Milanovic et al. in [156, 157], which provides the core concepts of transforming the three dimensions service, infrastructure and actors into

availability models of the service. More background information about this work can be found in Section 2.3.2.

In [5], a methodology is presented that provides a UML availability profile to obtain as output instead of an UPSIM a specific availability model expressed as *Reliability Block Diagram* (RBD) to evaluate service availability for different user perspectives. The case study in [5] uses an implementation of that methodology that is extensively described in [165]. Extending [2] and [5] with a time dimension has been done in [12] to support instantaneous service availability evaluation, the probability of a service to be available at a specific point in time. The infrastructure model from [2] is extended to include all ICT components, their failure and repair rates and deployment times. The mapping contains concrete ICT components for the service requester and provider, including possibly redundant components and their expected duration of usage. Using these models, the methodology automatically generates an availability model from those parts of the ICT infrastructure needed during service provision for the specified user-perceived view. The implementation of this approach is described in [196]. This provides a basic but solid concept of service time, an thus allows for true instantaneous availability evaluation. This will become even more important when variable and dependent failure and repair rates are introduced. In [157], although instantaneous availability is calculated, all components are considered to have been deployed at the same time. Additionally, the service execution has no duration but happens in one instant of time. The approach in [157] is thus most suitable for steady-state evaluation.

The remainder of the chapter is organized as follows. Section 4.2 defines the input models to be used in the service dependability evaluation. The methodology to evaluate user-perceived service dependability is presented in Section 4.3. Section 4.4 describes how to use this methodology to generate availability models. In chapter 5, the feasibility of the approach is demonstrated in representative case studies by applying it to an exemplary services within parts of the service network infrastructure of University of Lugano, Switzerland.

4.2 Model Specification

The methodology consists of generating a UPSIM respectively UPSAM from a service description, a network topology and a mapping between them. The UPSAM is then evaluated using an external tool for availability analysis. In this section, the different types of input models are defined and explained.

ICT infrastructures are able to support a variety of services, while a single service description can be similarly applied to a diversity of networks. For this reason, the presented methodology supports independent modeling of infrastructure and services, relying on a third model to provide a relation between them. This approach implies also that changes on the network topology or service description should be reflected only in their respective models. Our approach uses a subset of UML elements as it is standardized, easily expandable, well supported by numerous tools and

thus, widely accepted in both industry and academia. Moreover, UML *profiles* and *stereotypes* are applied to impose specific attributes to ICT components, resulting in a coherent model with a standardized description:

- *Class diagrams* are used to describe structural units of the network (e.g.: routers, clients, servers), their properties and relations in distinct classes.
- *Object diagrams* describe a deployed network structure/topology composed of class instances, namely objects with all properties of the parent class, and links as instances of their relations. Object diagrams are used to describe both the complete network structure as well as the UPSIM.
- *Activity diagrams* are used for the service description and represent the service as a flow of actions.

Figure 4.1 depicts the context of a UPSIM as a UML class diagram. The following subsections explain the different parts of the diagram in detail.

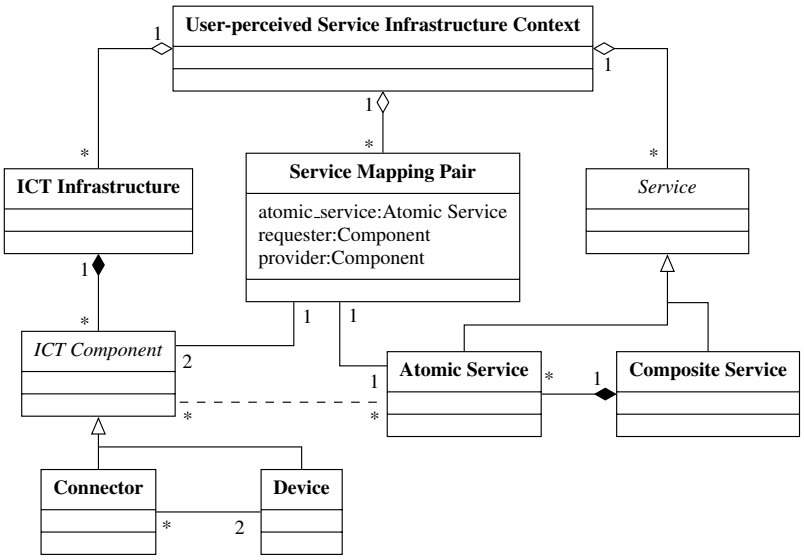


Fig. 4.1 Context of a user-perceived service infrastructure model.

4.2.1 ICT Infrastructure Model

A network topology can be represented as a bidirectional graph, in which network devices and their links are respectively characterized by nodes and edges. But in practice, service dependability rests upon the characteristics of individual components and networks are generally composed of heterogeneous nodes. This is why in

opposition to graphs, a separate model would better represent the individual characteristics of network components.

The class *ICT infrastructure* in the left part of Figure 4.1 aggregates a set of interconnected *ICT components*. Network nodes and the communication between them exhibit very distinct properties. For this reason, *ICT components* are subdivided into *Device* and *Connector* types. Following standard UML notation, Figure 4.1 shows that every *Connector* must be associated to two *Devices*, which may have any number of *Connectors*. *Devices* and *Connectors* are respectively modeled as classes and associations in a UML class diagram. The ICT infrastructure model is then presented in a UML object diagram, where network nodes are instance specifications of those classes, and communication is represented by the corresponding links, which are instances of associations. To ensure that two different instances of the same class also have the same properties, every class may only have static attributes.

The same set of attributes can be applied to common model elements through *profiles* [171] to facilitate dependability analysis that requires specific properties to be present for each model element. A *stereotype* containing these specific properties can be employed to guarantee that every *ICT component* inherits them and thus, meets the requirements of the analysis. Following this approach, we provide an availability profile, which is depicted in Figure 4.2. Focusing on both steady-state and instantaneous availability (see Section 2.3.1), the profile includes the stereotype attributes (1) failure rate, (2) repair rate and (3) component deployment time, at which a component is expected to have its maximum availability. Although any date and time format could be applied, epoch time has been chosen for deployment time to simplify subsequent steps. Additionally, the redundant components property specifies internal redundancy, which can be used to implicitly define a large set of ICT components into a single object in the infrastructure model. ICT components are divided into devices and links and are represented in the UML class diagram in Figure 4.2 as classes and associations, respectively.

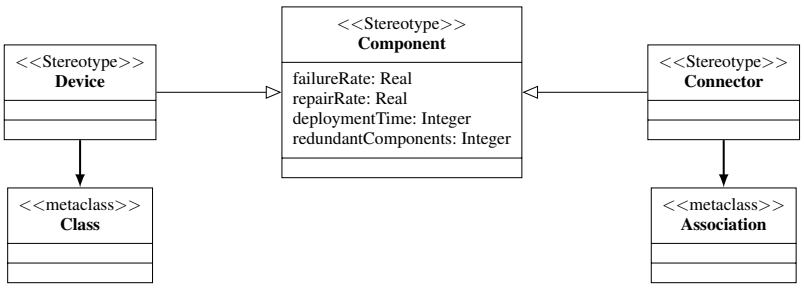


Fig. 4.2 Elementary availability profile

Furthermore, a network profile (see Figure 4.3) has been developed to guarantee the uniformity of the infrastructure. It defines an abstract stereotype *Network Device* to capture common properties of identified network components: *Router*,

Switch, *Printer* and *Computer*. The latter one is abstract and specializes into *Client* and *Server*. *Communication* is a stereotype dedicated to the association between classes, and has channel and throughput attributes. In practice, it corresponds to communication links between devices. With this profile, a set of attributes can consistently be imposed to stereotyped classes for later model transformation. Albeit only an illustrative representation in this case study, the network profile of Figure 4.3 can support a variety of attributes depending on the application.

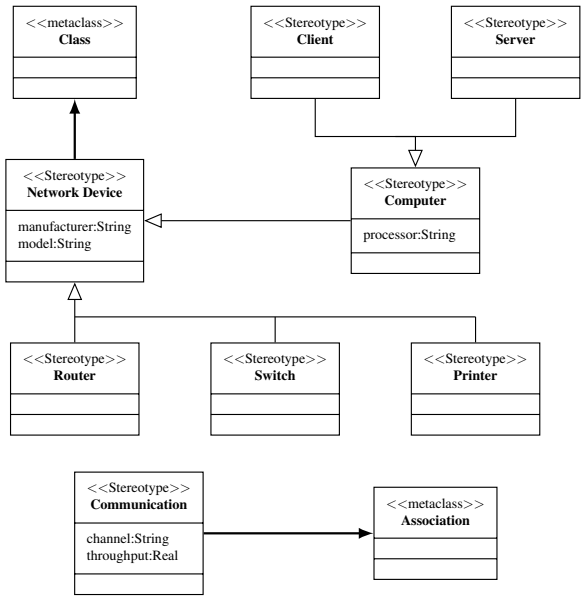


Fig. 4.3 Network profile with types of elements and their basic properties.

4.2.2 Service Model

Services in the context of this work are described as a sequence of complex actions provided by a composition of atomic service instances. *Composite services* are modeled with UML activity diagrams using *atomic services* as building blocks, as depicted in the right part of Figure 4.1. A composite service consists of initial and final nodes, *atomic services* and join and fork figures. Figure 4.4 presents the UML activity diagram of a simple composite service. It is assumed that each atomic service is being executed – in series or in parallel. In this particular diagram, atomic services *A* and *B* are executed in parallel. Instead of using decision nodes, separate decision branches are modeled as separate services.

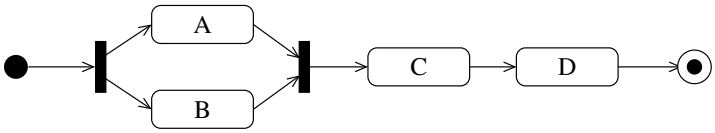


Fig. 4.4 Example of composite service model as UML Activity Diagram.

In this methodology, the availability is measured by the probability to traverse the activity diagram from start to end nodes, just like in common availability models. To accomplish that, every composing atomic service must be successfully executed, that is, there must be at least one path between a service requester and one of its providers in which all network components are available. As an example, the composite service model in Figure 4.4 shows that atomic services *A* and *B* are executed in parallel, *C* is executed right after both of them completed, *D* follows after *C*. The represented composite service is successful if and only if all its composing atomic services are successful.

According to Milanovic et at. [157], atomic services are abstractions of the ICT infrastructure, application or business level functionality. At this point, atomic services are still considered abstract functionalities and are not yet related to a set of concrete ICT components. This relation exists later during service execution, where atomic services map to a set of ICT components including requester, provider and connecting ICT components and inherit their properties.

4.2.3 Network Mapping Model

Since service properties strongly depend on the underlying infrastructure, a correct mapping between services and the ICT components that enable them is required for further analysis. To obtain the set of potentially required ICT components for each atomic service, the service model is projected to the ICT infrastructure by a separate mapping, represented by the dashed line in Figure 4.1 between classes *Atomic Service* and *ICT component*. Atomic services are instantiated by a service mapping pair when defining requester and provider. The mapping, provided as an XML file (see Figure 4.5), contains a unique description of the service mapping pair requester and provider for every atomic service. The service mapping pair in the center of Figure 4.1 gives the initial and final boundaries of the ICT infrastructure used by a specific atomic service. Other related ICT components depend on the possible paths between those communication end points. The methodology thus uses a path discovery algorithm to identify these paths as initially proposed in [148].

Contrarily to the *service mapping pair* proposed in [2] and as proposed in [12], this methodology defines a mapping model that allows multiple service providers per atomic service, which enables the modeling of redundant parts (i.e. multiple DNS servers) located in different areas of the network. This feature requires an additional annotation to describe the priority of access, as the redundant parts can be

```
<atomicservice id="A">
  <requester id="req1"></requester>
  <provider id="prov1"></provider>
</atomicservice>
```

Fig. 4.5 Example of the XML code representing the mapping of a single atomic service.

accessed in parallel or in series after an eventual failure. For the latter case, further annotations are needed to define serial access times of redundant components. Additionally, this methodology supports redundancy modeling using independent components with possibly different properties, instead of identical replicas as in [2]. Service and infrastructure descriptions are time-independent and, alone, are not able to provide an estimated execution time for atomic services. This becomes instead a parameter of the mapping. Having such information, it is possible to estimate for which interval each atomic service availability should be evaluated. The service model (see Section 4.2.1) plays an important role by describing which atomic services are executed in parallel or in series.

An example of the enhanced mapping model is shown in Figure 4.6. Multiple providers within an atomic service description are always considered to be redundant, that is, at least one of the listed instances must be available for the requester to achieve a successful service provision. Every instance may be able to provide the atomic service with a different estimated duration in seconds, and the priority (0 being the highest) plays a role for the definition of start and end execution times for each instance access.

```
<atomicservice id="C" requester="req1" timeout="10">
  <provider="prov1" duration="3" priority="0">
  <provider="prov2" duration="4" priority="1">
  <provider="prov3" duration="8" priority="1">
</atomicservice>
```

Fig. 4.6 Mapping of atomic service with parameters for instantaneous availability evaluation.

In the example of Figure 4.6, three providers *prov1, prov2, prov3* are able to deliver a specified atomic service C to the requester *req1*. Supposing this atomic service is invoked at time $t = 0s$, the example models the following behavior: component *req1* requests a service from component *prov1*, which has the highest priority. If service provision fails, *req1* requests the same service from components *prov2* and *prov3* simultaneously – both have priority 1 – after the timeout. Therefore, next requests are invoked at time $t = 10s$ and take 4 and 8 seconds, respectively. The atomic service is considered finished only after every provider has finished. The duration of atomic service C is then 18 seconds, that is the latest estimated end time minus the earliest estimated start time of its redundant providers. This way, the next atomic service will be invoked at time $t = 18s$.

Although there is a well-defined serial and parallel order of execution in the system behavior, availability analysis will always consider them as parallel since they represent redundancy. Details are described in Sections 4.4.1 and 4.4.2, where the example of Figure 4.6 is evaluated.

4.2.4 Input Model Considerations

Separating the infrastructure model, the service description and the mapping allows to efficiently handle dynamic system changes by updating only individual models where necessary. The reasons for system changes are manifold: user mobility, network topology changes due to new or failing components, service migration and so on. For instance, the UPSIM can be generated for different user perspectives with only minor changes to the mapping of atomic services while the abstract service model and network model remain untouched. In a mobile scenario, where users can be at different positions within the network but still use the same service, the network model needs to be updated while the service description and mapping remain the same. Migrating a service from one provider to another requires updating only the mapping while substituting a service – replacing one service composition with another one that provides the same functionality – requires changing only the service description and mapping but not the network model.

As input models, this work adopts the Unified Modeling Language (UML) for infrastructure and service descriptions as it is standardized and widely used, especially for design purposes. For the mapping model, XML is chosen due to its versatility. The decisions for UML and XML guarantee that the models remain human-readable and visualization was a relevant factor driving those decisions. However, the main contribution of the methodology proposed in Section 4.3 lies in the evaluation of user-perceived service availability, given that the ICT infrastructure is accordingly described with availability properties. Therefore, those inputs can also be provided using different formalisms, keeping intact the main purpose of the methodology but improving, for instance, the scalability of its application. One possible improvement has already been proposed in Chapter 7 (see also [4]), where the topology is gathered directly from the routing layer to be used in the responsiveness evaluation of service discovery in wireless mesh networks. The routing layer is also able to provide information about the quality of links, as it keeps statistical data about successful packet transmission among nodes.

4.3 Methodology

Given the input models from Section 4.2, a model-to-model transformation is applied to obtain the UPSIM and UPSAM, depending on the desired type of dependability evaluation. This section describes a methodology to do so and is based on

the methodology provided by Milanovic et al. in [157], although the order of steps has changed and the steps have been adjusted and extended to match the presented model-driven workflow. Especially the concept of service and component time is completely new, which will be applied in Steps 9 and 10. More detail can be found in Sections 4.4.1 and 4.4.2. Open-source tools are chosen for the implementation. Eclipse [217] is a multi-language software development environment with numerous extension plug-ins. This includes the UML2-compliant [170] modeling tool Papyrus [218] and the model transformation plug-in VIATRA2 [219] with embedded support for UML models. VIATRA2 complements the Eclipse framework with a transformation language based on graph theory techniques and abstract state machines. It also provides its own model space, so it can import the input models to an intermediate representation where they can be manipulated before a subsequent model generation.

As output, two different types of models are generated. The UPSIM is presented as UML object diagram and represents a subset of the original ICT infrastructure that includes all components relevant for the specific service mapping pair. All redundant paths between requester and provider are included. A methodology to generate the UPSIM is presented in Figure 4.7. Instead of generating the UPSIM, we can also output a *Reliability Block Diagram* (RBD) or *Fault-Tree* (FT) for the same subset of the network. Figure 4.8 presents an overview of the methodology, which differs only in the last step when generating the output model. All steps are described in detail below. Steps 1 to 4 provide the input models specified in Section 4.2 for later transformation and output model generation. The three input models are in the left part of Figures 4.7 and 4.8. All model tranformations and their auxiliary measures are depicted within the center rectangle labeled *VIATRA2* and described in Steps 5 to 7. The UPSIM generated in Step 8 and the UPSAM generated in Step 9 can be seen at the bottom of the figures.

1. Identify ICT components and create the respective UML classes for each class type. According to the subject of a subsequent dependability analysis, a UML profile can be applied to classes in this step. Results in a class diagram containing the description of every ICT component. See also Section 4.2.1.
2. Model the ICT infrastructure using UML object diagrams with instances of the classes from Step 1. Results in an object diagram of the complete infrastructure. See Section 4.2.1.
3. Identify and iteratively describe services using UML activity diagrams with atomic services as building blocks (actions). Results in a collection of service models with no correlation to the infrastructure. See Section 4.2.2.
4. Generate service mapping pairs by mapping atomic services from Step 3 to respective requester and provider ICT components from the infrastructure object diagram (Step 2). Results in an external XML file (see Figure 4.5). See Section 4.2.3.
5. Import ICT infrastructure and service UML models to the VIATRA2 model space using its native UML importer. VIATRA2 creates entities for model elements and their relations. Also, atomic services are transformed into entities of the model space.

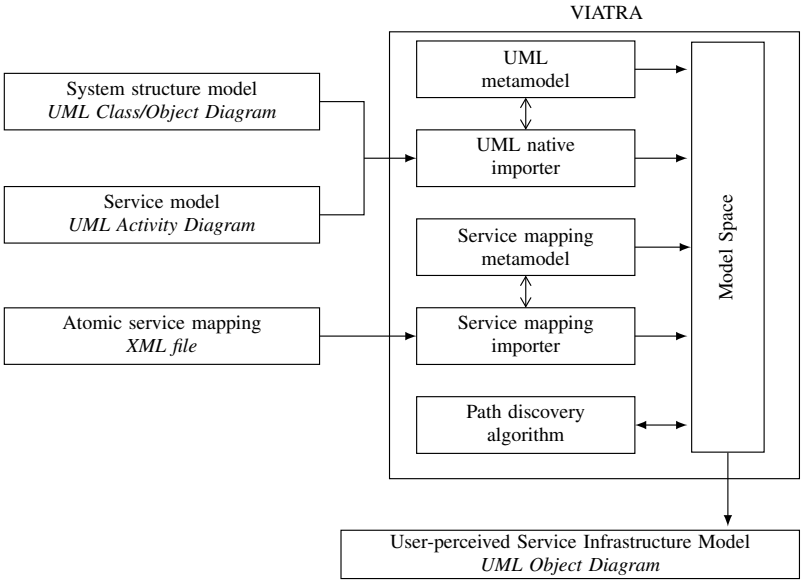


Fig. 4.7 Implementation of the model transformation to obtain the *User-Perceived Service Infrastructure Model* (UPSIM).

6. Import service mapping pairs to the VIATRA2 model space using a custom service mapping importer.
7. Discover all acyclic paths between requester and providers. Given a composite service, its atomic services and their service mapping pairs, an algorithm discovers all paths between the ICT components contained in these mappings. Resulting paths are stored separately in the model space for further manipulation. This step is described in 4.3.1.
8. Generate output model: UPSIM. Paths extracted from Step 7 are merged into a single network topology, corresponding to the user-perceived service infrastructure. The UPSIM is obtained as a UML object diagram. See Section 4.3.2.
9. Generate output model: UPSAM.
   a. Generate atomic UPSAMs. For each atomic service, Paths extracted from Step 7 are merged into a single network topology, corresponding to the user-perceived service infrastructure. The atomic UPSAM is obtained as an RBD or FT from that infrastructure.
   b. Generate composite UPSAM. According to the service model from Step 3, the atomic UPSAMs are combined into a single RBD or FT.
10. Calculate the user-perceived availability with the *Symbolic Hierarchical Automated Reliability and Performance Evaluator* (SHARPE) [224] using the composite UPSAM from Step 9.

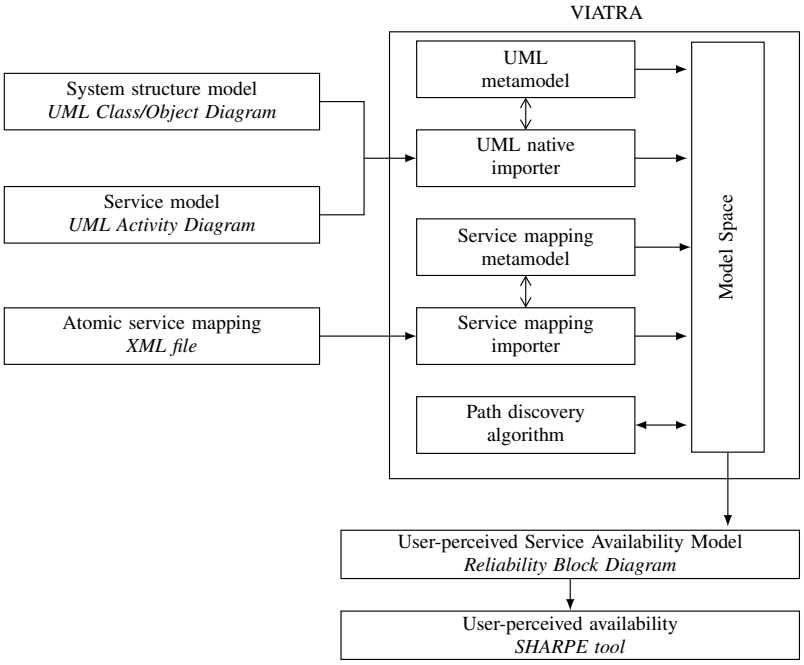


Fig. 4.8 Implementation of the model transformation to obtain the *User-Perceived Availability Model* (UPSAM).

Steps 1 to 3 are done manually using a UML modeling tool like Papyrus and kept unaltered as long as the ICT infrastructure and services descriptions do not change. They may also be obtained automatically using data from a configuration management database. The mapping in Step 4 is a simple XML structure where changes will eventually be performed in order to analyze different user-perspectives on a service. This can be done manually or automated. Steps 5 and onward are then fully automated. Extensive details about the implementation of all steps can be found in [165].

VIsual Automated model TRAnsformations tool

The approach relies on a model-to-model transformation using VIATRA2, that receives multiple input models conforming to specific source meta-models and produces a single target model conforming to a target meta-model (see Figure 4.7). A meta-model describes the abstract syntax of a model, its elements, their properties and relationships and modeling rules [64]. In VIATRA2, a meta-model is described using a specific syntax provided by the VIATRA textual meta-modeling language. Models and meta-models are stored in the *Visual and Precise Metamodeling* (VPM) model space, which provides a flexible way to capture languages and models from

various domains by identifying their entities and relations. To import the UML models into this model space, VIATRA2 provides a UML native importer and a UML meta-model. In order to import the service mapping pair into the model space, a custom service mapping meta-model and importer plug-in was developed based on the hierarchy presented in Figure 4.5. The task of the services mapping importer is to parse the XML file, traverse the content tree and find appropriate VPM entities in the meta-model corresponding to the type of each element. It is implemented in the Java programming language and added to the project workspace as an Eclipse plug-in. Additionally, the VIATRA2 textual command language provides a flexible syntax to access the VPM model space. It is based on mathematical formalisms and provides declarative model queries and manipulation [233]. This language is especially useful in this methodology to implement the path discovery algorithm in Section 4.3.1.

4.3.1 Path Discovery Algorithm

As stated in Section 4.2.3, the service mapping pair gives the initial and final boundaries of the ICT infrastructure used by a specific atomic service. In order to identify the ICT components potentially required for service provision, all possible paths between the service requester and provider must be traced. Multiple paths significantly increase the availability of an atomic service, as they provide redundancy. This approach implies that topology changes on the ICT infrastructure do not require adjustments to the service model or the atomic service mapping, given that requester and provider are still running on the same ICT components. For every service mapping pair, the algorithm discovers a set of paths between the respective requester and provider, and stores the visited entities in a reserved tree structure inside the model space.

The complexity of such algorithms grows significantly with the size of the ICT infrastructure. In order to find all possible paths, every node must be visited through all available edges. For this reason, the time complexity of the algorithm is even more sensitive to the number of edges, reaching $O(n!)$ for a fully interconnected graph of n nodes. The complexity is a serious problem in networks with a high connectivity, such as wireless mesh networks and the reason why the methodology for evaluation of service discovery responsiveness presented in Chapter 7 uses a probabilistic path discovery. However, cabled networks usually contain few if any loops, while most clients are located in tree-like structures with a low number of edges. The path discovery algorithm used in this methodology was first described in [2] and is based on the DFS algorithm [87], with a path tracking mechanism to avoid live-locks in cycles. It takes a pair of service requester and provider from the mapping model, identifies both in the graph representation of the network – obtained from the UML object diagram in Section 4.2.1 – and traces the paths between them to be subsequently merged into a subgraph. This process is repeated for every provider described in the mapping model. This assures a precise solution at the cost

of a higher runtime, which proved to be acceptable for the case study networks in Chapter 5. However, the methodology might as well use *Probabilistic Breadth-First Search* (PBFS) as described in Section 7.3 to achieve better scalability.

4.3.2 User-perceived Service Infrastructure

The final step of the methodology is the generation of an output model by merging the paths obtained from the algorithm. Since all the atomic services within a given composite service are executed, their paths are also merged into one UML object diagram which corresponds to the partial infrastructure required for proper service delivery. The instance specifications of the UPSIM object diagram have the same signature as in the original ICT infrastructure. Therefore, they maintain the same set of properties as the classes they instantiate. It is thus guaranteed that a subsequent service dependability analysis will find specific required properties for every element of the user-perceived ICT infrastructure. One such analysis could assess service availability based on those properties, which is exactly what the UPSAM does. Details on the generation of the UPSAM can be found in Section 4.4.

4.4 User-perceived Availability Model Generation

Since all the atomic services within a given composite service may be executed, the paths found in Section 4.3, Step 7 are merged into one model which corresponds to the partial infrastructure required for proper service delivery of a given service pair. The UPSAM is a transformation of that partial infrastructure. The instance specifications of the components within that partial infrastructure have the same signature as in the original ICT infrastructure from Section 4.2.1. Therefore, they maintain the same set of properties as the classes they instantiate. It is thus guaranteed that a subsequent availability analysis will find specific required properties for every element of the UPSAM.

Each atomic service has its own set of paths. All ICT components forming the path are translated into serialized blocks inside the RBD, given that all of them must be working in order to traverse the path. If an ICT component has n redundant components, the RBD will have n parallel blocks with the same characteristics. This corresponds to the redundant components property of the profile (see Figure 4.2). An atomic service is available if all ICT components on at least one of its paths are available. This introduces path redundancy inside the service network, and is represented within the RBD by placing blocks related to these paths in parallel. Identical blocks within those parallel paths are then merged into a single block.

Let us demonstrate this using Figure 4.9 as an example of an ICT infrastructure model, ignoring the links for simplification purposes. The following paths are identified from component A to G:

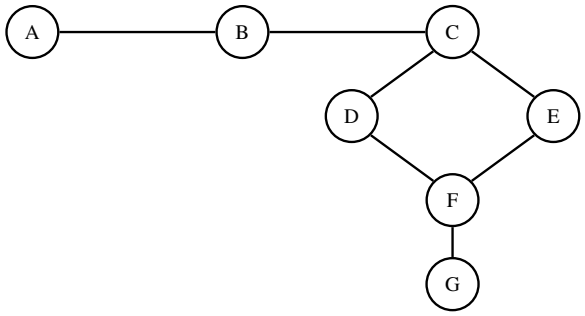


Fig. 4.9 ICT Infrastructure model example

$$A \to B \to C \to D \to F \to G$$
$$A \to B \to C \to E \to F \to G$$

Components A, B, C and F, G are represented as a series of blocks, as they are common for both paths. Blocks D and E are in series within their respective paths but parallel to each other. Thus, they are represented as a pair of parallel blocks in between the two sequences obtained previously. Given all components have no redundancy, only component B is triplexed (`redundantComponents=2`). Knowing that B has two redundant components for a total of three components, it is represented as three parallel blocks B. The resulting RBD is presented in Figure 4.10. Note that for steady-state evaluation, the order of the blocks does not affect the resulting availability. The individual RBDs of the atomic services are then connected in a similar manner to an overall availability model of the composite service, according to the defined service model (see Section 4.2.2).

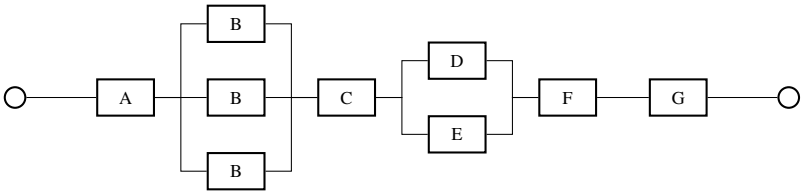


Fig. 4.10 Reliability Block Diagram of the example ICT infrastructure model

Existing methodologies for steady-state availability analysis mostly use RBDs to represent their output models. If the focus of modeling is on failure instead of success conditions, FTs are used. Both models provide comparable analysis for different points of view and are supported by the proposed methodology.

4.4.1 Instantaneous Availability Evaluation

In order to evaluate user-perceived *instantaneous* service availability, the time dimension must be added to the problem. So far, we have only been dealing with steady-state availability, where time $t \to \infty$. In such scenarios, every component is known to have reached a constant and stable availability, so that composite service availability can be evaluated as if composing atomic services were invoked at the same time $t \to \infty$. While steady-state availability has its applications, it cannot capture the behavior of services over time as it cannot consider different execution times of atomic services and the age of their providing components. For example, if after failure a hardware component is replaced with an identical, but new unit, steady-state analysis would result in the same service availability as before the replacement. When evaluating the service availability at time $t_x < \infty$, it is important to know the estimated execution time of each atomic service, since they may be invoked at different instants and the availability of ICT components varies over time. Moreover, every component may have been deployed at different points in time. Some components may have already reached a steady-state condition, while others are in transient state.

Consider the graph in Figure 4.11 and the atomic service mapping of Figure 4.6. The path discovery is executed three times, once per provider. Taking provider *prov1* as example, the algorithm is able to discover two paths starting from component *req1* (Figure 4.12(a)). In order to merge all discovered paths, identical vertices and edges are merged. The resulting subgraph is shown in Figure 4.12(b), and can be directly transformed into an availability model.

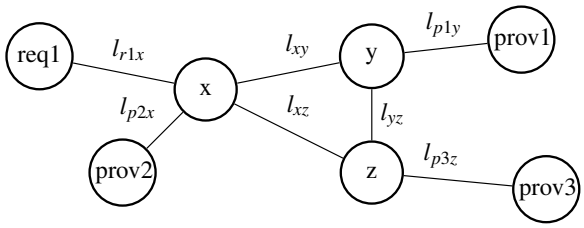


Fig. 4.11 Simple network topology to demonstrate path discovery.

For the purpose of instantaneous availability evaluation, the resulting subgraphs are transformed into individual availability models and connected to a composite model that calculates the user-perceived instantaneous service availability. A key contribution is that the availability of individual components is shifted in time according to their deployment time and to the estimated atomic service duration. This approach provides a realistic and consistent evaluation for transient scenarios, and can be divided into two steps:

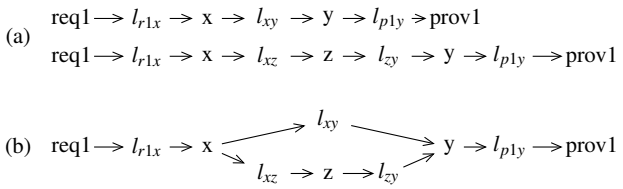


Fig. 4.12 Paths between *req1* and *prov1* (a) and merged subgraph (b).

1. The *Model generation* step generates the service availability model, which is composed of the availability of individual components arranged according to their roles in the service provision.
2. The *Access time definition* step complements the service availability model by identifying the exact instant at which each component is invoked within its life-cycle.

The model generation for the atomic services is similar to the one described for steady-state availability, while the access time definition warrants more detailed explanation in Section 4.4.2. The resulting user-perceived service availability model, represented in the FT in Figure 4.13, is composed of three main stages: (1) Composite service, (2) atomic services and (3) ICT components necessary for provision of the atomic services.

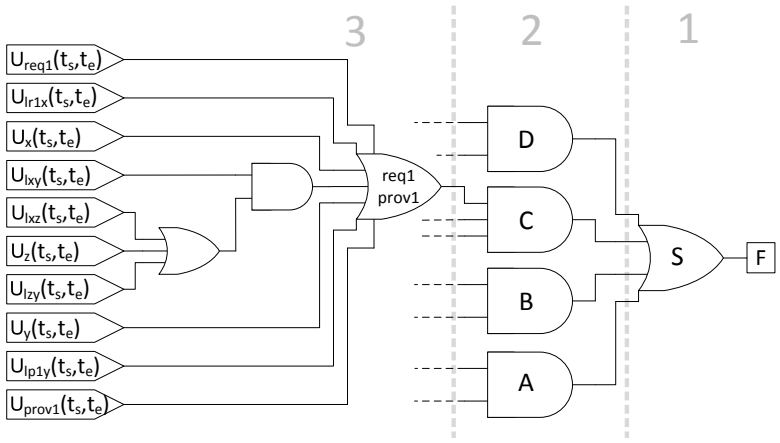


Fig. 4.13 Partial *fault-tree* (FT) of the provided example.

We will now explain the three stages in detail using the example FT in Figure 4.13. According to the service model in Figure 4.4, the condition for the composite service *S* to fail is that at least one of the atomic services fails. Therefore, the first stage of this model can be represented in the rightmost part of the tree by a single OR logic gate, where the number of input ports corresponds to the number

of atomic services. In stage two, every atomic service is represented by an AND logic gate with every provider connected to an input port. This logic gate represents the condition that all providers must fail to result in an atomic service failure. This information is provided by the mapping model in Figure 4.6, which contains three providers *prov*1, *prov*2 and *prov*3. Stage three in Figure 4.13 does not have a fixed pattern. Its logical circuits depend exclusively on the subgraphs from Section 4.3.1. Components essential for service provision of a specific provider are connected to OR logic gates, while redundant components are connected to AND logic gates. The third stage shows the logic circuits of the resulting subgraph in Figure 4.12(b), corresponding to communication between requester *req1* and provider *prov1* in Figure 4.6.

The same problem can be modeled using an RBD, depicted in Figure 4.14. The layers 1, 2 and 3 correspond to the equivalent stages in Figure 4.13: Composite service *S*, atomic service *C* and component *req*1 using component *prov*1 when requesting *C*. According to the RBD and FT formalisms, logical AND gates represent parallel blocks while logical OR gates represent serial blocks. The models in Figures 4.13 and 4.14, in addition to failure and repair rates of individual ICT components, are sufficient to evaluate the user-perceived steady-state service availability. In the next section, the access time definition is explained that allows these models to evaluate instantaneous availability of composite services.

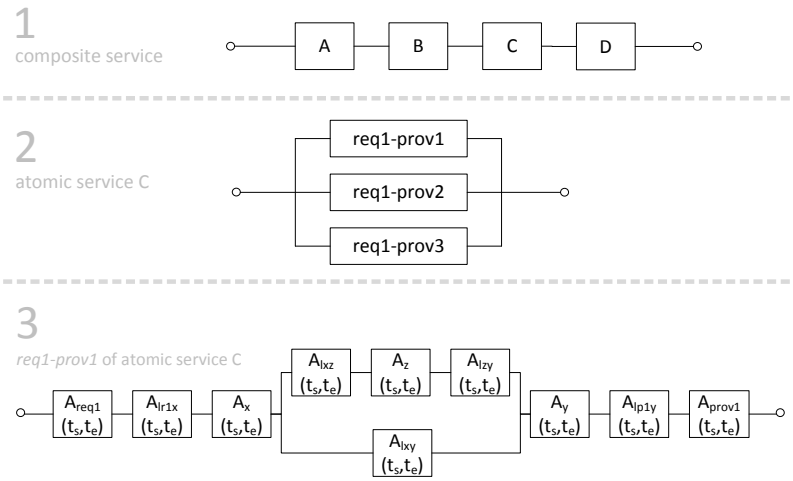


Fig. 4.14 Partial *reliability block diagram* (RBD) of the provided example.

4.4.2 Access Time Definition

In a non-steady-state scenario, we want to evaluate how availability decreases over time. The deployment time defines the instant at which individual components were fully available. Furthermore, using the additional information proposed in the mapping model – duration and priority – it is possible to calculate when every single atomic service provision is expected to start and finish. The assumption that every service provision starts at the same time is only valid for steady-state evaluation, where components have reached a stable availability – a proposition hardly realistic in dynamic and heterogeneous networks.

As seen in Section 4.4.1, the user-perceived service availability depends on the interval availability of many components, according to their roles in their respective atomic services. Another option would be to use the instantaneous availability of those components at a specific instant within this interval. Picking the correct instant is not trivial, however. Using the start of the interval would lead to an overly optimistic estimation while using the end time might be too pessimistic, especially for longer running atomic services. Picking any specific instant within the interval would need a sophisticated atomic service model with access durations for every component and dependencies among them. It is unclear what such a model would look like, knowing that components will change for every user-perspective, and how it could be reasonably validated to justify its usage. This is why this methodology proposes to use interval availability, the average availability over the whole duration of an atomic service. The exact range of the interval availability evaluation, however, is estimated according to the instant each component is invoked, taking its deployment time as reference. Initial references can be defined as follows:

- t_0 of an individual component is independently defined as its own deployment time.
- t_{a0} of a composite service is defined as t_0 of the youngest component.

Furthermore, every atomic service is invoked at different instants, with t_{a0} as reference. Their initial times are set according to the service and mapping models, which denote the serial/parallel configuration of each atomic service, and provide their estimated duration. As mentioned in Section 4.2.3, the estimated duration of each atomic service is given by the latest estimated end time minus the earliest estimated start time of its redundant providers.

As an example, consider the FT model in Figure 4.13. The relevant network components, required for the provision of atomic service *C* from provider *prov1* to requester *req1*, were deployed at different instants. In this example, components *prov1* and l_{p1y} are assumed the youngest and their deployment time is therefore taken as reference time t_{a0}. The composite service *S* will be invoked at $t = t_{a0} + 3600$, that is, one hour after the youngest component was deployed. The atomic service *C*, mapped in Figure 4.6, has an estimated duration of 18 seconds, as already described in Section 4.2.3. Using similar analysis, the estimated duration of atomic services *A* and *B* are set to 10 and 20. This way, it is possible to identify the time intervals of each atomic service relative to the reference t_{a0}:

$$
\begin{aligned}
t_{invocation\_start,A} &= t_{a0} + 3600\ seconds \\
t_{invocation\_end,A} &= t_{a0} + 3600 + 10\ seconds \\
t_{invocation\_start,B} &= t_{invocation\_start,A} \\
t_{invocation\_end,B} &= t_{a0} + 3600 + 20\ seconds \\
t_{invocation\_start,C} &= t_{invocation\_end,B} \\
t_{invocation\_end,C} &= t_{a0} + 3600 + 38\ seconds
\end{aligned}
$$

The invocation start time of C is equal to the latest invocation end time of A and B, as it will be invoked only after both A and B have finished. The same applies to each service provision within atomic services. Absolute start and end times of the availability intervals of individual components i necessary for service provision p are given by:

$$
\begin{aligned}
t_{s,i} &= t_{invocation\_start,p} - t_{deployment,i} \\
t_{e,i} &= t_{invocation\_end,p} - t_{deployment,i}
\end{aligned}
$$

Let the deployment time of component x be ten days, hence, 864000 seconds before t_{a0}. This way $t_{s,x}$ and $t_{e,x}$ in the service provision *req1-prov1* within atomic service C are obtained as follows:

$$
\begin{aligned}
t_{s,x} &= t_{a0} + 3600 + 20 - (t_{a0} - 864000) = 867620 \\
t_{e,x} &= t_{a0} + 3600 + 38 - (t_{a0} - 864000) = 867638
\end{aligned}
$$

The interval unavailability $U_x(t_s,t_e) = 1 - A_x(t_{s,x},t_{e,x})$ for component x in the example of Figure 4.13 should be calculated using Equation 4.1.

$$
A_x(t_{s,x},t_{e,x}) = \frac{1}{t_{e,x} - t_{s,x}} \cdot \int_{t_{s,x}}^{t_{e,x}} A_x(\tau) d\tau \tag{4.1}
$$

For constant failure and repair rates, $A_x(\tau)$ is given by Equation 2.5. A constant failure rate is a reasonable approximation after the wear-in period of hardware components, in which faults become random and independent events along most of the component lifetime. For the case study in Chapter 5, we will use this approximation for all availability calculations. It should be mentioned that in scenarios where this approximation is not valid and non-constant failure rates are given, a different instantaneous availability equation could be derived using these non-constant rates and used without further modifications within the proposed methodology.

Chapter 5
Case Study of User-Perceived Service Availability

Abstract This chapter demonstrates the feasibility of the methodology previously introduced in Chapter 4 with a representative case study. We apply the methodology to parts of the network infrastructure of *University of Lugano* (USI), Switzerland. This infrastructure consists of six interconnected routers and switches in its core with servers directly connected to it, and tree-like peripheral networks composed of clients and printers. First, a *User-Perceived Service Infrastructure Model* (UPSIM) is generated for two different clients of an exemplary mail service. User-perceived steady-state and instantaneous availability models are created for diverse scenarios. We show and discuss how availability changes in these scenarios and how the methodology enables assessment of these different results with very low effort.

5.1 Introduction

In this chapter, the feasibility of the methodology from Chapter 4 is demonstrated by generating the *User-Perceived Service Infrastructure Model* (UPSIM) and various different *User-Perceived Service Availability Models* (UPSAMs) of an exemplary *Send mail* service in a real network and from different user perspectives. The topology depicted in Figure 5.1 is based on the network of *University of Lugano* (USI), Switzerland. The network core, consisting of six interconnected routers and switches in its core, is nearly identical to the real infrastructure while the tree-formed peripheral parts connected to the core, composed of clients and printers, have been reduced for demonstration purposes. Servers are directly connected to the bottommost switches. Every component has a unique ID and a specified type in the format $id : type$. For better visualization, links lack labels. They are, however, referenced throughout this chapter as the concatenated IDs of the devices they connect. For instance, the line between *t1* and *e1* represents a communication link identified by *t1_e1*. Links in this network are categorized either as wired or wireless, respectively represented by full or dashed lines. In Sections 5.2 and 5.3, only wired links are taken into account.

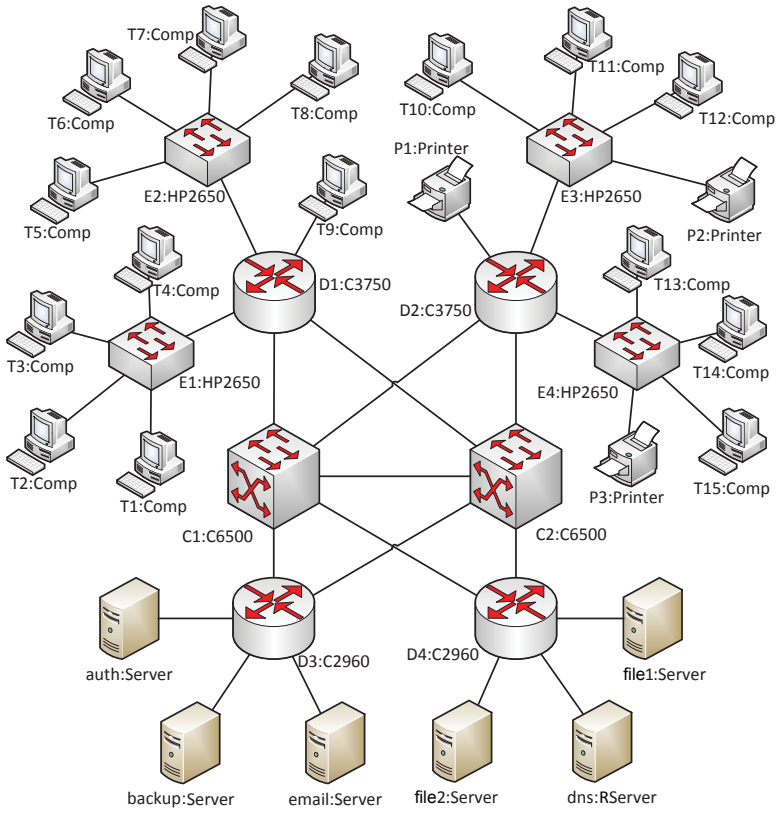


Fig. 5.1 Network infrastructure based on university campus network. Lines between devices represent network links.

5.2 User-Perceived Service Infrastructure

A number of atomic services are provided in the network, such as *Check authentication*, *Dispatch email*, *Print document*, *Request backup* and so forth, whereas each service has at least one provider. Atomic services can compose composite services, for example email, printing or backup, which are requested by different clients located at various positions within the network. The following sections explicitly reflect the steps of the proposed methodology defined in Section 4.3.

5.2.1 Identification and Modeling of ICT Components

The ICT infrastructure is modeled with focus on service dependability assessment, more specifically, steady-state and instantaneous availability. In order to use the

UPSIM for user-perceived service availability assessment, we developed the simple UML profile in Figure 4.2, which contains relevant properties for such analysis. This profile is then applied to the infrastructure. Each ICT component, device or connector, has intrinsic dependability attributes such as a failure rate, a repair rate and a number of internal redundant components. In this case study, we assume that these rates are constant, and that the failure rate encapsulates all possible failure causes within the scope of a component, e.g. hardware or software. As mentioned in Section 4.4.2, this assumption does not necessarily hold. In scenarios where functions of non-constant rates are given, a different equation to calculate instantaneous availability of individual components could be derived and used without further modifications within the proposed methodology.

As can be seen in the availability profile in Figure 4.2, although *Device* and *Connector* inherit the same attributes from *Component*, they are distinguished in order to be applied – respectively and exclusively – to *Class* and *Association* elements.

The different types of ICT components identified in the infrastructure model (Figure 5.1) are modeled in a set of stereotyped classes depicted in Figure 5.2, which shows the UML class diagram containing the description of the devices and their connections as, respectively, classes and associations. This diagram contains all network elements with their predefined availability properties. At this point, we evaluate all constituting devices and their possible connections. For instance, *D1* and *D2* are different instances of the same device labeled *C3750*, which is known to be a switch. Therefore, the corresponding class is created, with *Component* and *Switch* stereotypes applied from the availability and network profiles. When all devices are represented as classes, their possible links are represented by associations stereotyped as *Component* and *Communication*.

As another example for the ICT component description in UML, in Figure 5.2 the type `RServer` represents a server containing an internal redundancy of one extra component (`redundantComponents=1`) which signifies that there are actually two servers with one server to fail over. Type `C2960` represents a switch. The complete list of ICT components including relevant availability data is presented in Table 5.1.

5.2.2 ICT Infrastructure Modeling

The infrastructure object diagram (see Figure 5.3) is built with instances of classes and associations (namely *instanceSpecification* and *links*) from the previous step, and reflects the topology of the network in Figure 5.1. Since attributes are statically defined in classes, those instances are already well-defined components. Also, given that links are instances of associations, connections are possible only when those associations exist.

In the UML object diagram in Figure 5.3, each node is represented by a unique identification and the respective type, in the format `id:Type`. Types *HP2650*, *C3750*, *C6500* and *C2960* are switches, the other types should be self-explanatory.

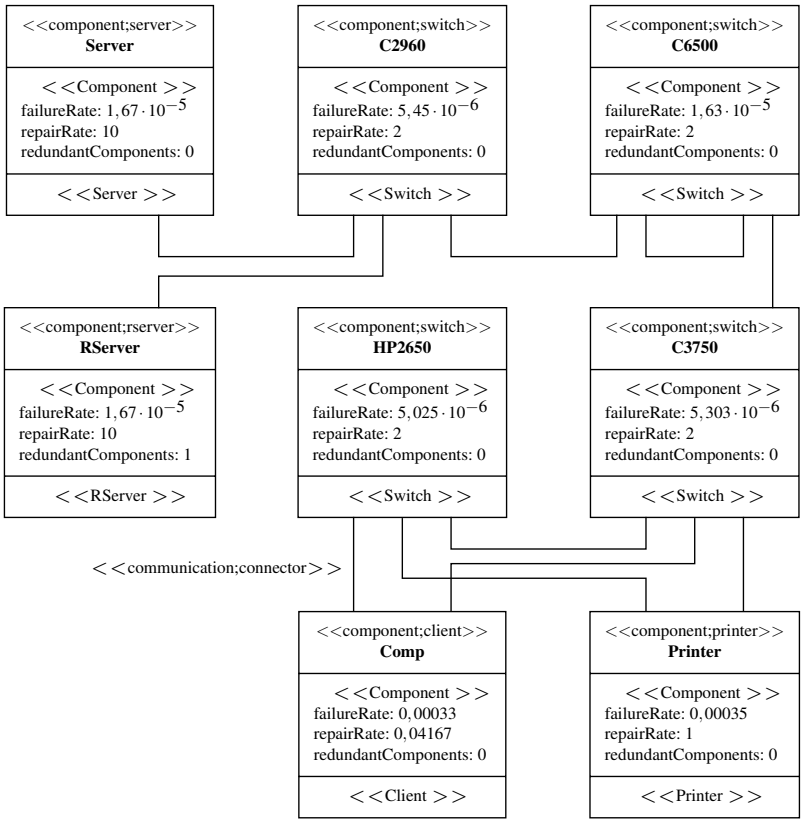


Fig. 5.2 Predefined network elements classes.

Table 5.1 Specification of ICT components

Type	Manufacturer	Model	Failure rate $(1/h)$	Repair rate $(1/h)$	RC*
C2960	Cisco	Catalyst 2960-48FPD-L	$5,45 \cdot 10^{-6}$	2	0
C6500	Cisco	Catalyst 6500	$1,63 \cdot 10^{-5}$	2	0
C3750	Cisco	Catalyst 3750G-24TS	$5,303 \cdot 10^{-6}$	2	0
HP2650	Hewlett-Packard	ProCurve 2650	$5,025 \cdot 10^{-6}$	2	0
Server	Dell	PowerEdge T620	$1,67 \cdot 10^{-5}$	10	0
RServer	Dell	PowerEdge T620	$1,67 \cdot 10^{-5}$	10	1
Comp	HP Compaq	DC7800	0,000333	0,04167	0
Printer	Canon	IR3245N	0,001389	2	0
Wired link	*n/a*	*n/a*	$7,69 \cdot 10^{-6}$	0,25	0
Wireless link	*n/a*	*n/a*	0,0833	33,333	0

*redundantComponents

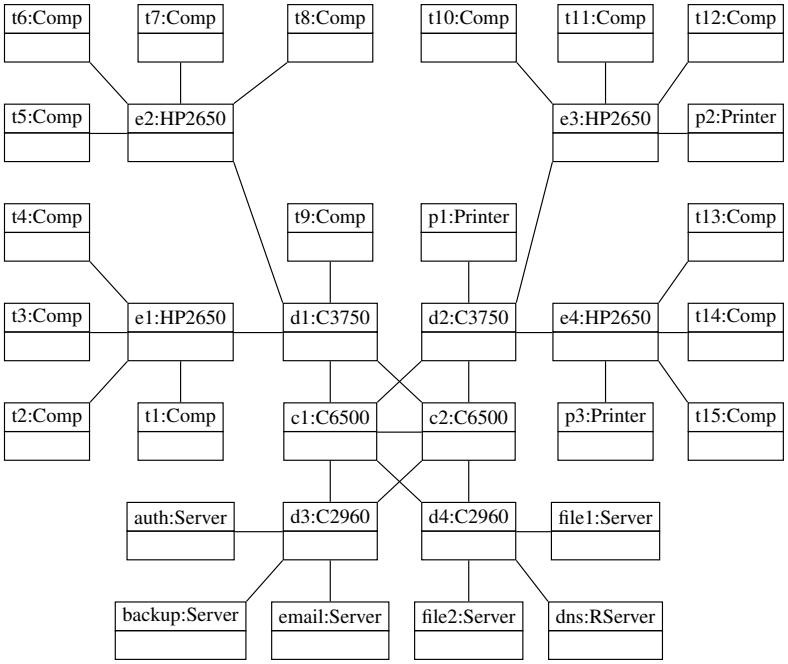


Fig. 5.3 Network infrastructure presented in UML Object Diagram

Given that *RServer* has *redundantComponents* = 1, the *dns* is then known to have redundancy although represented by a single node.

5.2.3 Identifcation and Modeling of Services

The exemplary *Send mail* service of this case study consists of resolving the *Mail Exchanger* (MX) address via the *Domain Name System* (DNS) and then sending an email message over that MX by means of the common *Simple Mail Transfer Protocol* (SMTP). During SMTP communication, the MX checks the credentials provided by the client with an external authentication server. This service represents a widespread use-case in present-day service networks. In detail, the service is composed of three atomic services, used in sequential order. Following is a description of the atomic services:

1. *Resolve mail server address* – A client first uses the DNS to resolve the MX record, an IP address of the mail server.
2. *Dispatch email via SMTP* – The client connects to the MX using that address. It will then send an email message over that MX by means of the common SMTP.

3. *Check authentication* – During SMTP communication, the MX will check authentication credentials provided by the client using an external authentication server.

The UML activity diagram representing the *Send mail* flow of actions of the composite service is depicted in the activity diagram in Figure 5.4.

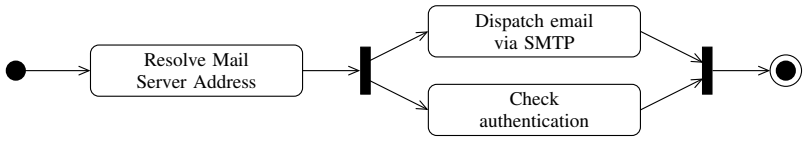


Fig. 5.4 Send mail service represented in UML activity diagram

The *Send mail* service description (see Figure 5.4) remains generic and abstract. It is a composition of exclusively atomic services, which are not yet mapped to ICT components. This means that in case of changes to the ICT infrastructure, the service description remains identical. Thus, the same service description can be used to describe a service for arbitrary pairs in any network that provides the atomic services.

5.2.4 Service Mapping Pair Generation

Each atomic service from the service description is now mapped to infrastructure elements by means of service mapping pairs, which associate each atomic service with requester and provider ICT components. Mapping is the key mechanism to support dynamicity as it allows to change service requesters and providers with minimal effort. According to the service description of Figure 5.4, the service mapping should contain at least three pairs with their atomic service as unique key. Additional service mapping pairs could be listed in the mapping file to support other services. However, they will be ignored when the corresponding atomic service is irrelevant for the analyzed service.

To illustrate the proposed methodology for a specific user-perspective, we explicitly select requesters and providers in the service mapping pairs of all atomic services: In this case study, the ICT components *t1* and *backup* were chosen as clients to compare two views on a composite service as perceived by different clients. Components *dns*, *email* and *auth* play the roles of dns server, mail server and authentication server. The mappings between atomic services and the ICT infrastructure (Step 4 of the methodology) for the clients *t1* and *backup* are given in Tables 5.2 and 5.3, respectively. It can be seen that only minor changes to the input models are necessary to change the user-perceived view on a service: Only the requesting instance in the mapping is changed, the network model and service description remain untouched.

Table 5.2 Service mapping pairs of the *Send mail* service for client *t1*.

Atomic Service	Requester	Provider
Resolve mail server address	t1	dns
Dispatch email via SMTP	t1	email
Check authentication	email	auth

Table 5.3 Service mapping pairs of the *Send mail* service for client *backup*.

Atomic Service	Requester	Provider
Resolve mail server address	backup	dns
Dispatch email via SMTP	backup	email
Check authentication	email	auth

5.2.5 Model Space Import

Next, we are running the VIATRA2 native UML2.2 importer on all files containing the set of input diagrams (profiles, class diagram, object diagram and activity diagram). As described in Section 4.3, the import of the service mapping pairs to the VIATRA2 model space is accomplished with a custom-developed service mapping metamodel and importer plug-in. The XML file is parsed and the content tree traversed to identify elements. All import steps are completely automated.

5.2.6 Path Discovery for Service Mapping Pairs

In this step, the elements of each service mapping pair are matched to ICT components of the infrastructure (see Figure 5.1). The algorithm sees the infrastructure as a graph and iteratively extracts all possible paths between two vertices requester and provider. For instance, for the first service mapping pair of Table 5.2, the discovery for client *t1* identifies eight acyclic ways to reach *dns*:

$$t1 \to e1 \to d1 \to c1 \to c2 \to d4 \to dns$$
$$t1 \to e1 \to d1 \to c1 \to c2 \to d4 \to dns$$
$$t1 \to e1 \to d1 \to c1 \to d2 \to c2 \to d4 \to dns$$
$$t1 \to e1 \to d1 \to c1 \to d3 \to c2 \to d4 \to dns$$
$$t1 \to e1 \to d1 \to c2 \to d4 \to dns$$
$$t1 \to e1 \to d1 \to c2 \to c1 \to d4 \to dns$$
$$t1 \to e1 \to d1 \to c2 \to d2 \to c1 \to d4 \to dns$$
$$t1 \to e1 \to d1 \to c2 \to d3 \to c1 \to d4 \to dns$$

These paths are then added to the VIATRA2 model space for further manipulation. The path discovery algorithm has been implemented using the VTCL language provided by VIATRA2. This step is completely automated.

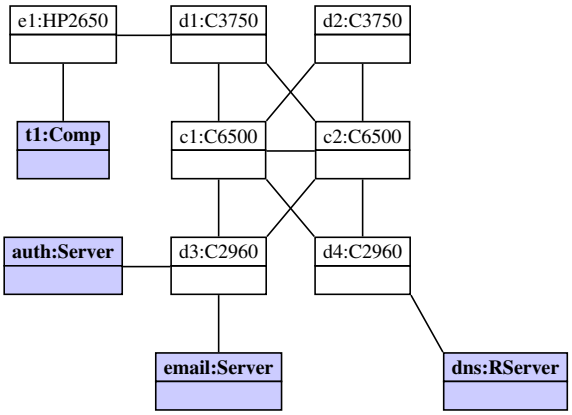
5.2.7 User-Perceived Infrastructure Model Generation

The final step comprises of matching the elements of the paths obtained in the previous step (Section 5.2.6) to the complete infrastructure given by the second step (Section 5.2.2). This step is fully automated and behaves like a filter on the complete topology, where only nodes which appear at least once in the discovered paths are preserved. Multiple occurrences are ignored. The output model is a UML object diagram containing only that fragment of the infrastructure required for the execution of the *Send mail* service from client *t1*, all redundant paths taken into account. This user-perceived service infrastructure model is shown in Figure 5.5a. The types of *InstanceSpecifications* are adopted from the input infrastructure object diagram. Therefore, the service and network properties (failure rate, repair rate, etc.) are automatically inherited from the instantiated classes. Services and dependability properties are now correlated, facilitating extraction for further analysis. To generate the UPSIM for the second perspective, the *Send mail* service from client *backup* using the same atomic service providers, we only have to make minor adjustments to the service mapping. Figure 5.5b shows the UPSIM from this user perspective.

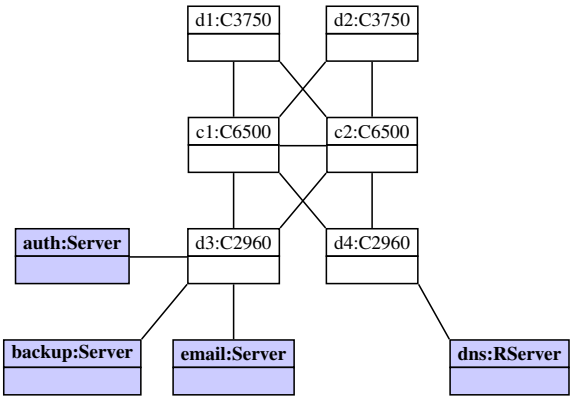
The generated UPSIM can be used to visualize the set of ICT components and their connections relevant for a particular pair requester and provider. This alone is very helpful in case of service problems, as it provides a quick overview on which ICT components might be the cause. More important, the UPSIM can be used to facilitate analysis of various user-perceived dependability properties (e.g.: availability, performability, responsiveness). For example, it can be used to assess user-perceived steady-state availability for a service, that is, the ratio of operational time over the livetime of a service, observed by a particular user. Such analysis can be performed by transforming the UPSIM to a RBD or FT, in which entities correspond to components of the UPSIM. The availability for individual components can be calculated using the component attributes failure rate and repair rate, as seen in Formula 2.10. The service availability can then be computed using the RBD or FT. We present this complementary transformation to RBDs in Section 5.3. More details about the process can be found in [5].

5.3 User-Perceived Steady-State Availability

We will now demonstrate the evaluation of user-perceived availability using the methodology from Section 4.4 with the exemplary *Send mail* service. We simplify the fault model by taking only the steady-state availability of ICT components into



(a) Requested by client *t1*.



(b) Requested by client *backup*.

Fig. 5.5 UPSIM of *Send mail* service for two different user perspectives.

account. This means we assume that all faults from classes *fail stop* to *byzantine* are combined in the steady-state availability of the individual ICT components. An ordered fault classification can be found in [25]. We also disregard service discovery: The DNS server address is known a priori to the client as is the authentication server address to the MX. In the following, we will generate the UPSAM for the first atomic service of the *Send mail* service, *Resolve mail server address*. Generation for the subsequent atomic services is omitted but will adhere to the exact same procedure.

As described in Section 4.3, the ICT infrastructure is represented by a UML object diagram, where each node is an instance of a specific ICT component class described in a UML class diagram. The links between nodes are also represented as instances of associations from the UML class diagram. For simplification, in this

specific evaluation associations are given the maximum availability of 1 – meaning that they are always available – for the sole purpose that their respective RBD blocks can be omitted in illustrations without affecting the steady-state availability. The full topology object diagram is shown Figure 5.3.

For UPSAM generation, instead of generating the UPSIM as described in Section 5.2.7, the paths discovered in the step described in Section 5.2.6 are merged and transformed into a single RBD. For requester *t1*, this RBD is shown in the upper part of Figure 5.6. This procedure basically corresponds to Step 9a of the methodology detailed in Section 4.3. It reduces common nodes of different paths and excludes those which do not affect the overall availability of the service. For instance, in order to pass through *d2*, nodes *c1* and *c2* must be available, in addition to the common nodes *t1*, *e1*, *d1*, *d4* and *dns*. However, their availability implies that there is already at least one path guaranteed to be available between *d1* and *d4*. This is because associations have an availability of 1 and there are associations between *c1* and *d1*,*d4* as well as between *c2* and *d1*,*d4*. For this reason, the node *d2* does not affect the overall steady-state availability and is excluded from the UPSAM. Furthermore, redundant components are expanded: The *dns* component is converted into a pair of parallel blocks *dns1* and *dns2*.

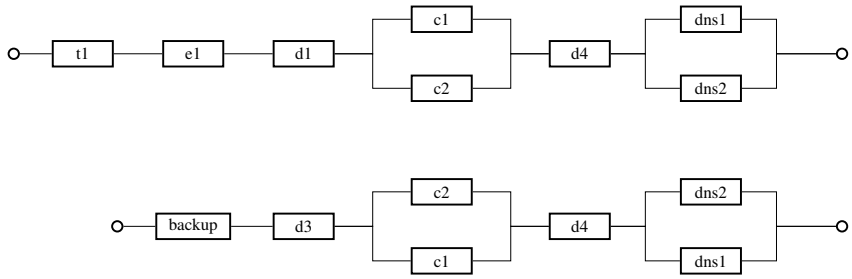


Fig. 5.6 *User-perceived service availablity models* (UPSAM) of atomic service *Resolve mail server address* for requesters *t1* and *backup*.

Figure 5.6 shows the UPSAM for requester *t1* side by side with the analogously created UPSAM for requester *backup*. We see only minor differences in the two models because to reach *dns*, both requesters have to use almost the same part of the network. Both have to traverse the network core, only the entry points are different. The next atomic service *Dispatch email via SMTP* paints a different picture. To reach the mail exchanger, requester *backup* does not need to traverse the network core, drastically reducing the number of blocks in the reliability block diagram. The UPSAMs for these user perspectives are depicted in Figure 5.7.

Next, a composite UPSAM is created from the atomic UPSAMs according to the service description in Figure 5.4. Although the *Send mail* service described in the activity diagram contains a parallel execution, every single atomic service must be concluded in order to accomplish the execution of the composite service. For this reason, the resulting UPSAMs of the individual atomic services are put in series to

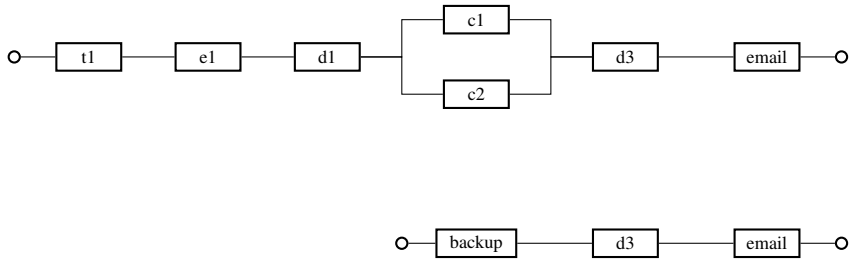


Fig. 5.7 *User-perceived service availablity models* (UPSAM) of atomic service *Dispatch email via SMTP* for requesters *t1* and *backup*.

compose the overall UPSAM of the composite service *Send mail*, as presented in Figure 5.8. This corresponds to Step 9b of the methodology detailed in Section 4.3. For the sake of clarity, the atomic service RBDs have been combined into single blocks in the figure.

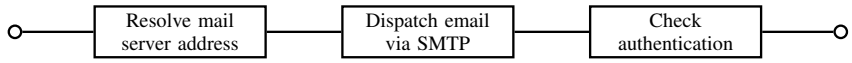


Fig. 5.8 Service availablity model of the *Send mail* service

As the last step, we use the SHARPE tool [224] to solve the obtained UPSAM to calculate the steady-state availability for the composite service *Send mail*. Results are shown in Table 5.4. We included results for the same service as requested by client *backup* to show how two different user perspectives on the same service differ in their availability.

Table 5.4 Service availability of *Send mail* service from different user perspectives

Service	Requester *t1*	Requester *backup*
Resolve mail server address (atomic)	0.999912118	0.999992884
Dispatch email via SMTP (atomic)	0.999910452	0.999993942
Check authentication (atomic)	0.999993942	0.999993942
***Send mail* (composite)**	**0.999816521**	**0.999980768**

In fact, altough the availability is reasonably high for both clients, it is an order of magnitude higher when the same service is requested by client *backup* instead of client *t1* (1.6 hours downtime per year for client *t1* versus 10 minutes for client *backup*). This justifies the approach of considering user-peceived service availability. These differences are expected to be of a much higher magnitude in more heterogeneous networks with a significant variability in availability of the various component types, especially when taking into account different link qualities. In Section 5.4, we introduce several types of heterogeneity, such as varying link quality and

component age to provide a more complete picture of user-perceived service availability.

5.4 User-Perceived Instantaneous Availability

This section demonstrates the proposed methodology for instantaneous availability evaluation applied to the previously described illustrative *Send email* service. Three atomic services compose the service, as depicted in Figure 5.4. This part of the case study slightly modifies server positions from Figure 5.1 and introduces a wireless link. These changes are introduced solely to exemplify conceptual differences of the types of analysis. For example, network links now have a realistic availability to integrate varying link quality into the analysis. We also include external redundancy represented by redundant providers of an atomic service at different positions in the network, as opposed to internal redundancy in previous sections, which means identical redundant providers at the same position in the network. This also leads to a UML object diagram updated from Figure 5.3.

Failure and repair rates are again set according to their types as in Table 5.1. But to allow instantaneous availability assessment, the deployment times are provided individually for each component in Table 5.5. Times are represented in epoch time, where 0 denotes midnight on the 1^{st} of January, 1970. For each link, this corresponds to the time of deployment of the youngest component connected to its edges. For instance, link *c1_d3* was deployed on epoch time 1366027200, which is the same as for *d3* and later than 1346511000 for *c1*. Reasonable values that could reflect a real world example were chosen for all components. In an actual network infrastructure, both tables could be updated at run-time using monitoring information and a configuration management database.

Two distinct scenarios will now be evaluated. First, the variance of instantaneous availability from different user perspectives is demonstrated in Section 5.4.1). The impact of changing the age or number of components is investigated in Section 5.4.2. The resulting RBDs and FTs are too complex to be visually presented in a reasonable manner in this section and are thus left out. Only the results of their evaluation are shown instead.

5.4.1 Different User Perspectives

In this scenario, three different clients *t1*, *t2* and printer *p2* are requesting *Send email*. As can be seen in Figure 5.9, component *t1* is connected to *e1* via a wired link. The mapping model of this scenario is shown in Table 5.6. As opposed to *t1*, *t2* is connected to *e1* via a wireless link. The rest of the infrastructure remains unchanged, so the only difference from client *t1* is a less reliable link. The third client, printer *p2*, connects to the network from a completely different position. The change of user

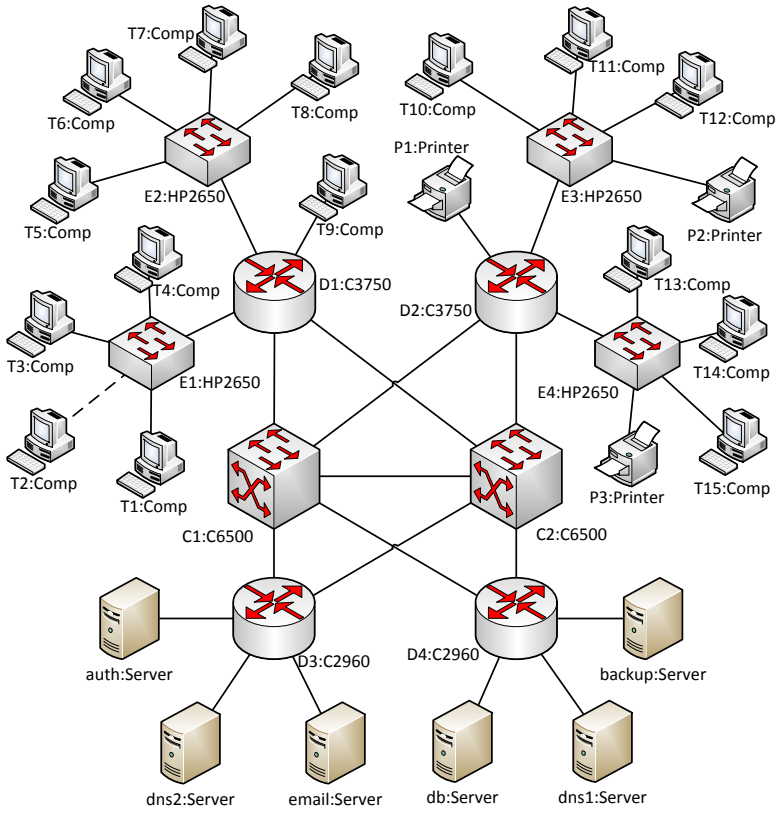


Fig. 5.9 Network infrastructure based on university campus network, slightly modified server positions for case study. Lines between devices represent network links, a dashed line constitutes a wireless link.

perspective for *t2* and *p2* is achieved with only minor modifications to the mapping model, changing the requester component of the atomic services *Resolve address* and *Dispatch email* in Table 5.6. The methodology then automatically generates different availability models.

The reference time t_0 of this evaluation corresponds to the deployment time of the newest component potentially required during service provision, component *auth* at epoch time 1366108200 for all three clients. Instantaneous availability $A(t)$ is then calculated over t until it reaches a steady-state condition. Although this network contains younger components, these were not identified by the path discovery as potentially required during service provision and have no impact on this analysis. The resulting diagram for the instantaneous availability of *Send email* when invoked at time t is presented in Figure 5.10. *Send email* is not fully available at t_0 because not all components were deployed at that exact time. This means that while some components may have reached a steady-state condition, others will be in a transient

Table 5.5 Deployment times of individual ICT components.

ID	Deployment time (s)	ID	Deployment time (s)
t1	1366043100	e3	1366016400
t2	1366024200	e4	1367073300
t3	1368788400	d1	1366038000
t4	1368896400	d2	1346511000
t5	1366013700	d3	1366027200
t6	1366459200	d4	1346511000
t7	1366545600	c1	1346511000
t8	1367778000	c2	1346511000
t9	1369040400	p1	1366027200
t10	1366016400	p2	1366099200
t11	1366026600	p3	1368361800
t12	1366026600	backup	1368446400
t13	1367073300	db	1365850800
t14	1367073900	dns2	1366027200
t15	1367247600	auth	1366108200
e1	1366038000	email	1366050000
e2	1366013700	dns1	1355572800

Table 5.6 Mapping model of client *t1* for *Send email*. For other clients simply all occurences of *t1* have to be changed to the new client.

Atomic Service	Requester	Timeout	Provider	Prio	Durat.
Resolve address	t1	10s	dns1	0	2 sec
			dns2	1	2 sec
Dispatch email	t1	10s	email	0	5 sec
Check auth.	email	2s	auth	0	2 sec

state. Over time, the availability decreases until also the most recently deployed components, in this case *auth* and *d3_auth*, reach their individual steady-state availability. Figure 5.10 shows a comparison of the instantaneous availability of *Send email* when invoked by clients *t1* (dotted line), *t2* (full line) and *p2* (dashed line). Some corner values are shown in Table 5.7 with the instantaneous availability $A(t_{a0})$ at the reference time, the steady-state availability A and difference of the two.

Table 5.7 Availability numbers of *Send email* service for different user perspectives.

Requester	$A(t_{a0})$	A	$A(t_{a0}) - A$	
t1	0.9955	0.9916	0.0039	$\rightarrow 5.616\ min/day$
t2	0.9922	0.9891	0.0031	$\rightarrow 4.464\ min/day$
p2	0.9990	0.9989	0.0001	$\rightarrow 0.175\ min/day$

The steady-state availability for *t1* is 0.9916 and that of *t2* is 0.9891. This translates to 3 days of downtime per year for *Send email* when requested by *t1* versus

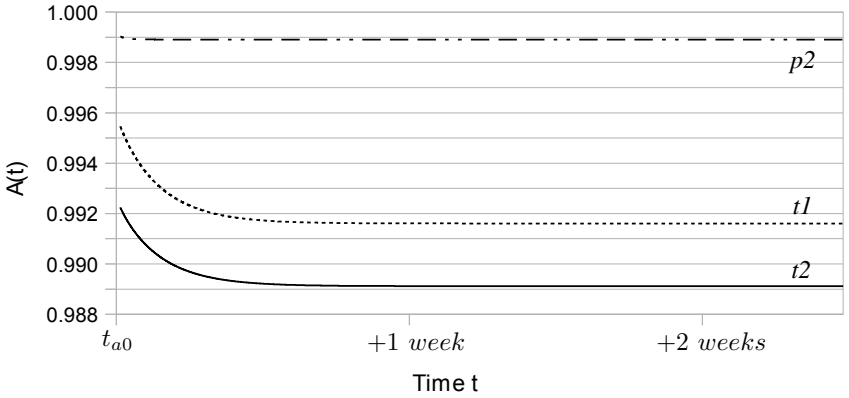


Fig. 5.10 Instantaneous availability over time of *Send email* service for three different clients.

almost 4 days when requested by *t2*. In both scenarios, the links between clients *t1* and *t2* and the device *e1* have a time redundancy when the DNS service is requested. If the first request to *dns1* fails, the links will be tried again by requesting *dns2*. So with respect to *t1_e1* and *t2_e1*, the DNS service will be successful when they are available either during the first or the second request. This can be modeled as a system of parallel availability blocks. During a subsequent request to the *email* server, the links are again accessed, which can be modeled as a block in series to the previous parallel system.

The evaluation of links *t1_e1* and *t2_e1* according to their access order in this system of two parallel blocks in series with a single block, results in the steady-state availability of 0.99997 and 0.9975, corresponding to the one for wired and wireless links, respectively. The ratio of these values resembles the ratio of the composite service steady-state availabilities for *t1* and *t2*, since the only difference between them is the link from the clients to *e1*. $A(t)$ of *p2* is much higher than the one of *t1* and *t2*. It is also notably more stable: While the difference between $A(t_{a0})$ and A is minor for the printer, it sums up to a few minutes per day for the two client computers, as can be deducted from Table 5.7. The scenario shows that the proposed methodology is able to capture diverse availabilities of the same service, depending on which client is using it and also, that the variation of availabilities is different over time.

5.4.2 Adding and Replacing Equipment

The second scenario evaluates how $A(t)$ changes when a set of network components is added or replaced. Evaluation is again done from the perspective of *t1* with reference time t_{a0}. This time, *dns2* is absent during the first week. In the mapping model, this fact is reflected by having no redundant provider for the first atomic service.

The next event is a renewal of equipment in the lab room where components *t1, t2, t3, t4* and *e1* were located (see Figure 5.9).

In Figure 5.11, the availability with a single DNS server reaches a lower steady-state availability of 0.984 during the first week, against 0.9916 with redundant DNS providers. The addition of *dns2* after one week increases the availability of the composite service considerably. However, adding *dns2* alone does not show a significant effect on instantaneous availability, as all but one component necessary to reach *dns2*, link *dns2_d3*, are also required to access components *dns1* and *email*. All of the are already close to a steady-state condition. Thus, $A(t)$ for *Send email* decreases only minimally over a few days to steady-state availability before the lab room equipment is exchanged. Following the replacement, $A(t)$ reaches 0.9997 as new components are known to be fully available. The individual component availability then decreases until they again reach a steady-state condition, bringing the overall service availability to the same level as before. Results are summed up in Table 5.8, steady-state values for *A* represent the lowest values within the evaluated period.

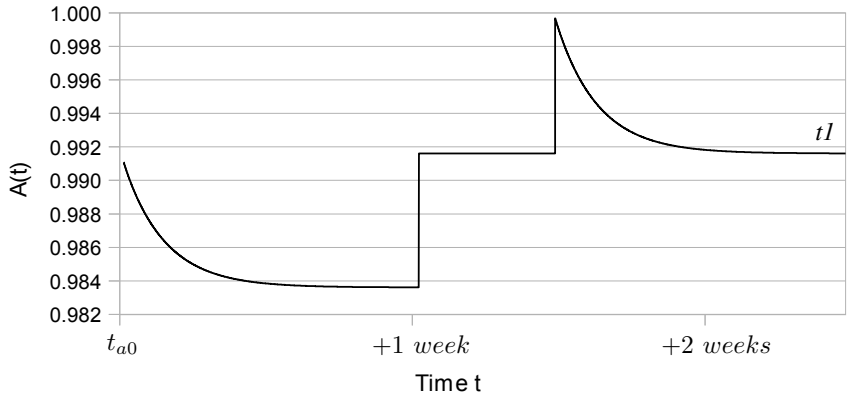


Fig. 5.11 Instantaneous availability over time of *Send email* service for client *t1* when changing equipment.

Table 5.8 Availability of *Send email* for *t1* when changing components.

Requester	$A(t_{a0})$	A	$A(t_{a0}) - A$
only *dns1*	0.991091	0.983619	0.007472 $\rightarrow 10.76\ min/day$
dns1, *dns2*	0.991609	0.991605	0.000004 $\rightarrow 0.3\ sec/day$
new lab	0.999674	0.991612	0.008063 $\rightarrow 11.61\ min/day$

When exactly the availability of a service will reach steady-state depends on the individual characteristics of the deployed components. Usually, the instantaneous availability will tend to steady-state availability after a duration in the order

of few weeks without changes in the network. In a regular network with a reasonable amount of components and dynamics, it is very rare to have weeks without any changes to the ICT infrastructure. This means that at any time, there will be at least some user-perspectives in a transient state, which justifies the decision to evaluate instantaneous availability.

5.5 Conclusion

The evaluation of availability over time concludes this part about user-perceived service availability evaluation. Dependability assessment in modern service networks remains challenging. It has been demonstrated that service availability indeed depends considerably on the properties of the providing ICT infrastructure. This is expected to be true for dependability properties other than availability as well. The methodology presented in Chapter 4 facilitates the automated calculation of such user-perceived properties. The separation of input models allows for quick updates in dynamic networks. However, parts of the methodology have limits regarding their scalability. Future research should focus on reducing the complexity of those parts. For example, we will show in Chapter 7, how in specific networks the exact path discovery may be replaced by a stochastic method.

The presented service models still do not include the *Service Discovery* (SD) process. It could be included as a first service in series with the original service description. The duration of this SD service would be the deadline until SD is required to have successfully enumerated all providers of subsequent services in the service model. The availability of this SD service thus reflects the probability of SD to finish successfully until the deadline, or in other words, its responsiveness. Part III of this work covers the metric SD responsiveness and the models presented in Chapter 7 allow to calculate it.

Part III
Service Discovery Responsiveness

In service networks, *Service Discovery* (SD) plays a crucial role as a layer where providers can be published and enumerated. This part focuses on the responsiveness of the discovery layer, the probability to operate successfully within a deadline, even in the presence of faults.

We approach SD responsiveness by evaluating it in two different experiment environments in Chapter 6. Both setups evaluate common protocol implementations to provide a realistic view on the responsiveness of active SD. First, a virtual testbed is set up with a simplified communication fault model that includes only packet loss. This is done to isolate the impact of packet loss on responsiveness as it is expected to have a major influence. Next, similar experiments are run using the previously introduced experiment framework *ExCovery* on the *Distributed Embedded Systems* (DES) testbed at Freie Universität Berlin. In the DES testbed, instead of injecting faults, varying load conditions provoke realistic fault behavior.

Following the experiments, a hierarchy of stochastic models for decentralized SD is proposed in Chapter 7. It is used to describe the active discovery of a single service using three popular SD protocols. A methodology to use the model hierarchy in wireless mesh networks is introduced. Given a pair requester and provider, a discovery protocol and a deadline, it generates specific model instances and calculates responsiveness. This process is supported by the Monte Carlo method *Probabilistic Breadth-First Search* (PBFS), which estimates various metrics for the propagation of discovery packets. Furthermore, a new metric *Expected Responsiveness Distance* d_{er} is introduced, to estimate the maximum distance from a provider where requesters can still discover it with a required responsiveness. Using monitoring data from the DES testbed, it is shown how responsiveness and d_{er} of the protocols change depending on the position of nodes and the link qualities in the network. Chapter 8 uses data from experiments to validate the developed models. Two ways of solving the models are evaluated. First, the full model hierarchy is solved based on low level monitoring data from the routing layer. Second, only the application level discovery model is solved using a history of SD communication measurements. Results show that the discovery model estimations correlate almost perfectly with the real responsiveness. In the absence of SD communication measurements, the full model hierarchy provides reasonable estimations given that the low level monitoring data is accurate.

Both the extensive series of experiments and the analytical model results give valuable insight into the responsiveness of SD that can help to improve future research on the dependability of service usage as a whole.

Chapter 6
Experimental Evaluation of Discovery Responsiveness

Abstract As a time-critical operation, an important metric of *Service Discovery* (SD) is responsiveness – the probability of successful discovery within a deadline, even in the presence of faults. To get an idea about the responsiveness of SD in realistic scenarios, this chapter provides an evaluation from a comprehensive set of experiment series. We present results of the evaluation of decentralized SD, specifically active SD using *Zeroconf*. To identify the main impairments to SD responsiveness, two different experiment setups are chosen. First, we examine SD responsiveness under the influence of packet loss in a controlled virtual platform. We show that responsiveness decreases significantly already with moderate packet loss and becomes practicably unacceptable with higher packet loss. Second, we examine SD in the *Distributed Embedded Systems* (DES) wireless testbed at Freie Universität Berlin. We demonstrate how the responsiveness changes depending on the distance of actors and the load on the network. The experiments reveal packet loss, packet delay and the position of SD actors as main factors influencing SD responsiveness. 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.

6.1 Introduction

Rather than developing an analytical model, which will be presented in Chapter 7, this chapter approaches the responsiveness of *Service Discovery* (SD) in a series of experiments. The experiments serve the following purposes:

1. To provide a general overview on SD responsiveness using a virtual experiment environment with a simplified fault model. For several realistic discovery scenarios, the responsiveness of active SD using *Zeroconf* [55] is shown depending on

the deadline t_D of the discovery operation, the packet loss rate and the required number p of providers P to be discovered.
2. To allow a deep insight into SD responsiveness in WMNs, similar scenarios are examined in experiments carried out in the *Distributed Embedded Systems* (DES) wireless testbed at Freie Universität Berlin. Responsiveness is evaluated depending on the distance of actor nodes and the load in the network. 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 presented in Chapter 7. As such, the gained results could be used to verify the validity of employing these models when optimizing the responsiveness of SD configurations in WMNs.
3. To 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 analyses.
4. To show the long-term behavior of the DES testbed and demonstrate the effects 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 forced interruptions during execution of experiments.

The focus of the evaluation is on active SD (see Section 2.2.2), where a client actively sends out discovery requests, retrying in case not enough valid responses were received until a timeout. The responsiveness in active discovery reflects the probability $R(t_D)$ that p valid responses to a request sent at time t_0 are received until the deadline $t_D > t_0$. The number p denotes the required number of service providers in a specific scenario.

This chapter provides results from two different sets of experiments that evaluate responsiveness of active decentralized SD in unreliable networks. All systems in experiments ran common Linux operating systems and service network stacks based on publicly available reference implementations. They are thus representative for systems in real life applications. Two different experiment setups are used:

1. A virtualized testbed, based on the XeN hypervisor technology [188, 27]. The service network is automatically configured using the de-facto standard technologies proposed by the *Zeroconf* working group [243, 105, 55].
2. The *Distributed Embedded Systems* (DES) wireless testbed at the Freie Universität Berlin. The *ExCovery* framework is used to run experiments, which provides a unified description, execution, measurement and storage of distributed system experiments and assures repeatability. *ExCovery* is described in detail in Chapter 3.

We present and discuss the results of the experiments and show how SD responsiveness is affected by the loss rate in the network, the load in the network as well as

the position and number of requesters and providers. The presented research complements the existing work on SD dependability (see Section 2.3.5) by providing a comprehensive experimental evaluation of responsiveness with a realistic fault model.

This chapter represents a concise presentation of the results published in [6, 7] and [61]. The rest of the chapter is structured as follows. The SD scenarios evaluated in the experiments are described in Section 6.2. The experiment setup and configurations can be found in Section 6.3. Sections 6.4 and 6.5 present and discuss the results of analysis. The chapter concludes with Section 6.6.

6.2 Service Discovery Scenarios

For the experiments, three representative SD scenarios were chosen. The scenarios reflect common use cases of SD, such that the experiments should give a reasonable overview of the responsiveness of decentralized discovery mechanisms with different fault intensity. Table 6.1 summarizes the parameters of the experiment scenarios.

Table 6.1 Summary of simulation parameters for the three scenarios for virtual testbed and the wireless DES testbed.

Scenario	*I*		*II*		*III*	
Testbed	Virtual	DES	Virtual	DES	Virtual	DES
Number of network nodes	2	115,119	2...51	115,119	2...51	51
Number of service clients	1	1	1	1	1	1
Number of service providers	1	1,1	1,20,50	1,1	1...50	1...30
Maximum discovery time	1,7s	1,7s	0...20s	0...20s	20*s*	20*s*
Packet loss (%)	0...90	n/a	20,40	n/a	10,20,30,40	n/a
Number of load generators	n/a	0...50	n/a	0,40	n/a	10
Bitrate per load generator	n/a	72kbit/s	n/a	72kbit/s	n/a	0...1,7Mbit/s
Observed discovery operations	10000	24000	6000	4000	24000	4000

6.2.1 Scenario I – Single Service Discovery

The goal of the first scenario is to measure the responsiveness when discovering a single service. The service network consists of one client and one provider. This might resemble the most common scenario for SD: One client needs to use one specific service in the network, such as the printing on a specific printer or the backing up to a *Network-Attached Storage* (NAS). The scenario can be considered as the baseline: Only one answer needs to be received and the requesting application has different requirements in terms of how long it is able to wait for a response. The client is allowed to wait for a positive response up to $t_D = 1$ second in one case, and

up to $t_D = 7$ seconds in the other. These deadlines were chosen as they correspond to two retry events in the *Zeroconf* discovery protocol, which was used in the experiments. As can be seen in Table 2.1, the first retry is triggered at $t_D = 1$ second and the third retry at $t_D = 7$ seconds. Due to delays on the nodes and in the network, we can be sure that in the first case only responses to a single request arrived while in the second case three requests have been sent.

To see how results vary in unreliable networks, measurements are taken with increasing fault intensity. In the virtual testbed, packet loss rates are varied from 0 to 90 percent. In the wireless testbed, additional load is generated increasing from 0 to 50 streams with a data rate of 72 kilobits per second each. We additionally analyze providers at two different positions in the WMN to investigate the impact of distance on responsiveness.

6.2.2 Scenario II – Timely Service Discovery

Many service networks are populated with multiple instances of the same service type to support redundancy. A client needs to discover as many instances as possible and will then choose one that optimally fits its requirements or failover to another provider later in case the chosen one crashes. For consistent conditions, in this scenario full coverage is required, so all available providers need to be discovered. There is one service client, P service providers with $P \in \{1, 10, 50\}$ and discovery is successful when all $p = P$ provided service instances have been discovered. The goal is to measure how responsiveness increases with time. It should be obvious that the faster a required responsiveness can be reached, the better.

In the virtual environment, measurements are carried out with a packet loss rate of 20 and 40 percent, respectively. In the testbed, we use two different load setups, 0 to 40 streams with the same data rate of 72 kilobits as in Section 6.2.1. Additionally, instead of trying to discover multiple providers, we fix $P = 1$ and analyze providers at two different positions in the WMN to demonstrate the impact of distance on $R(t)$.

6.2.3 Scenario III – Multiple Service Discovery

When dealing with diverse faults in the network, having more redundant instances of a given service type should generally increase SD responsiveness when looking for a fixed number of instances. In the case of fixed coverage however, which means the ratio of discovered services needed for successful operation remains constant, this is not necessarily the case. Fixed coverage is needed, for example, for monitoring tasks or in general, when complete knowledge about available service providers is demanded. A more thorough investigation of the improvements to responsiveness when deploying more service providers is justified. The impact of additional com-

munication overhead when discovering an increasing number of providers remains unclear.

This is why in the third scenario, we investigate the difference in responsiveness when discovering $p = P$ out of P service instances (full coverage). In the virtual testbed, P is growing from 1 up to 50 providers. Measurements are carried out with a packet loss rate of $10, 20, 30$ and 40 percent. The experiments in the DES testbed were run in the dense cloud of building T9 of the DES testbed (see Section 6.3.2) as the increasing load would quickly cause bridging links between the buildings to break down, partitioning the network and rendering results useless. This reduced the total number of network nodes. The maximum number of providers here is $max(P) = 30$. Also, instead of setting a packet loss rate, we are increasing the load in the network until a saturation point where responsiveness does not decrease anymore.

6.3 Experiment Setup

In this section, the setups for the two experiment platforms, the virtualized environment and the wireless DES testbed, are explained, to carry out the experiments needed for the scenarios described in Section 6.2. We focus on the most important parameters. More detailed information about the experiments can be found in [6, 7], the *ExCovery* framework is described in Chapter 3.

As service discovery protocol, the wide-spread *Zeroconf* protocol suite based on multicast DNS [54] and DNS-based service descriptions [53] is used. A full description of the protocol can be found in [55]. *Zeroconf* implements a two-party architecture (see also Section 2.2.1) and all messages, requests and responses, are sent via multicast. Comparable experiments were carried out using *Service Location Protocol* (SLP). Since the results for SLP confirm the findings shown here for *Zeroconf*, they are not included in this Chapter and can be found in [61].

Discovery requests are run from a single dedicated discovering node – or service client. All other nodes act as service providers forwarding and/or responding to discovery requests. 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 P providers, an operation to discover p providers is successful, when p providers have received at least one request packet and from each of these p providers one response has arrived at the requester within deadline t_D. The probability of a request to arrive at p providers and of p responses from those providers to arrive until t_D at the requester denotes the responsiveness $R(t_D)$ of the discovery operation. The minimum of multicast messages to be sent is $p+1$, not including duplicate transmissions in the network due to the multicast flooding. Consequently, the maximum number of messages for each request and subsequent retry is $P+1$. It needs to be pointed out that the actual number of transmissions inside the mesh network is considerably higher and grows with an increasing number of collisions.

Discoveries were aborted and considered failed if no responses arrived until an experiment run deadline of $max(t_D) = 20$ seconds in the virtual testbed, respectively $max(t_D) = 30$ seconds in the wireless testbed. This value was chosen because in *Zeroconf*, the time between retries doubles after each retry to reduce network load (see also Section 2.2.2 and Table 2.1). So after 15 seconds, we have reached a total number of five SD requests and the next one would be sent after 31 seconds. Waiting time was extended by additional 5 seconds in the XeN environment to ensure delivery of all responses. In the wireless testbed, the extension was 15 seconds. Since no considerable delay was to be expected in the virtual testbed, the shorter deadline is justified. In both cases, the timeout means that for *Zeroconf* we will have the responsiveness after four retries at $max(t_D)$.

6.3.1 Virtual Testbed

All experiments in the virtual testbed were carried out on a machine running the Xen hypervisor [188]. Service provider and client nodes ran as unprivileged guest domains [27]. Provider nodes had the same base system to boot from, which was a minimal installation of the Debian Linux operating system [75]. At boot time, they were only running the *Zeroconf* daemons to configure the service network and do SD. A dedicated client node additionally ran a *Secure SHell* (SSH) daemon for remote execution of discovery operations. Memory requirements for the guest systems were very low, the technical specifications of the systems are listed in Table 6.2. A schematic of the virtual testbed setup is shown in Figure 6.1.

Table 6.2 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

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 SD communication. On the *Internet Protocol* (IP) [187] layer, the topology reflects a fully connected, single-hop network. All packets sent among nodes pass the bridge between them. On this bridge, packet loss was realized by randomly dropping packets at a

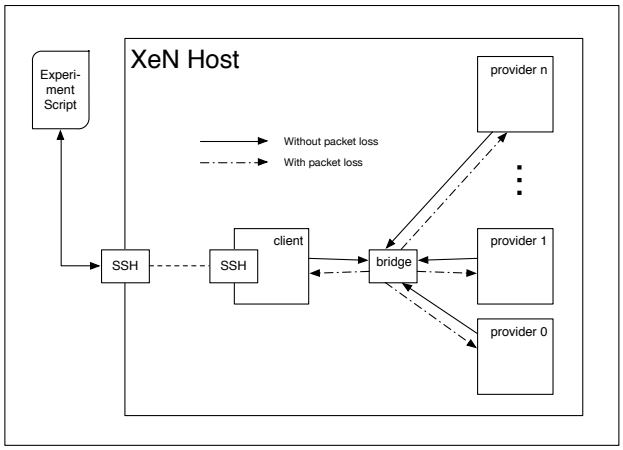


Fig. 6.1 Overview of the virtual testbed setup.

given rate. The rate was the same for every packet, no additional faults were injected on the bridge. Packet dropping was done independently in the forwarding direction of every individual interface connected to the virtual bridge. The setup causes multicasts or broadcasts to be potentially lost at the interface to every node they were transmitted to. This effectively prevents all-or-nothing behavior where a broadcast is either received by all or by no host at all. A multicast can be lost on the way to one node without affecting the probability of getting lost on the way to any of the other nodes. The SSH daemon on the client node ran on a second interface which was not connected to the virtual bridge so that it was unaffected by the fault injection and traffic on this interface did not interfere with service network traffic.

To automatically configure the service network, software developed by the Avahi project [22] was used. Avahi is an implementation of the protocols recommended by the *Zeroconf* working group for automatic configuration of IP service networks [243]. An *Automatic private IP addressing* (AutoIP) [52] daemon set a unique IP address within the 169.254.0.0/16 subnet and an mDNS [54] daemon handled service name resolution for DNS-based service discovery [53]. Due to the fact that Avahi was used for auto-configuration, all nodes could run from copies of the same system disk image and no manual administration was necessary after booting. During measurements, no nodes joined or left the network so no reconfiguration of the network layers occurred which could interfere with the observed SD operations.

In the XeN environment, 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.

Justification of Experiment Setup

The chosen network model is a simplified version of a fully connected, wired Ethernet network. In such reliable networks, packet loss is usually not an issue. Typical unreliable networks, especially wireless networks as in the second testbed, have a complex fault behavior. Most faults occurring in these networks, such as bursts of packet loss, delay or jitter have complex functional dependencies. However, the intention behind using this virtual testbed is to control the network faults during experiments as much as possible to isolate their impact.

Packet loss is expected to be among the highest-impact faults in SD. Thus, the experiments run in this testbed focus on packet loss. The fault model includes only random packet loss at a given rate, independent of the traffic occurring on the link and whether preceding packets are lost as well. Packet delivery time, which is in the order of hundreds of microseconds in this testbed, can be neglected compared to the time between discovery retries, which is in the order of seconds (see Table 2.1). Since an average packet loss is assumed, service discovery responsiveness will most probably be worse in networks with the same packet loss rate but more complex fault characteristics. This analysis hence provides an upper bound for responsiveness which will be compared to the results in the wireless DES testbed with highly complex fault characteristics.

6.3.2 Wireless Testbed

As the second platform to execute experiments, the wireless DES-Testbed at *Freie Universität Berlin* (FUB) was used. To run the experiments, the *ExCovery* framework for distributed system experiments was employed. *ExCovery* was specifically developed for this work and supports the DES testbed as the first experiment platform. More in-depth information on *ExCovery* follows in Chapter 3.

The DES testbed consists of roughly 130 uniform nodes equipped with IEEE 802.11 hardware. The nodes are spread indoors and outdoors over three adjacent campus buildings at FUB. An overview of the testbed topology is depicted in Figure 6.2 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 [36, 100, 101].

The DES testbed was chosen for various reasons. First of all, as opposed to the virtual environment, it allows to evaluate processes in a network with realistic fault behavior. Second, the DES testbed allows to generate manifold topologies due to

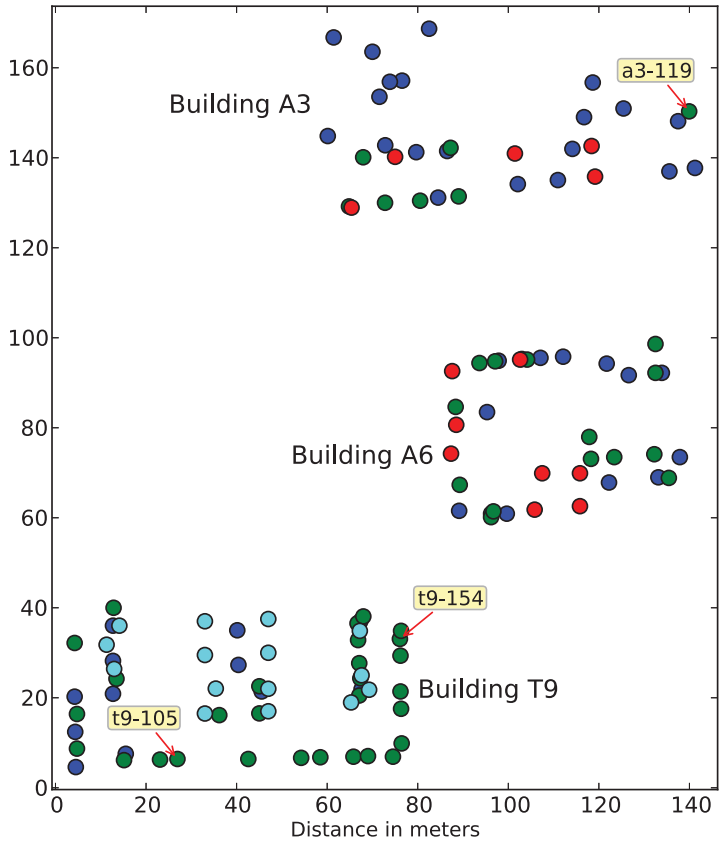


Fig. 6.2 Overview of the wireless DES testbed. Three buildings can be distinguished, nodes colors define their building floors.

its wide distribution of nodes on campus and the ability to manipulate their wireless signal range. This gave us the possibility to study in detail the effects of node distance on SD responsiveness. Finally, the nodes run a relatively modern Linux distribution which simplifies the development and deployment of new software.

Multicast provides considerable challenges in wireless mesh networks, which 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. Just as for the experiments in the virtual testbed, Avahi [22] was used to implement SD, albeit in a slightly newer version 0.6.29. However, the results presented in Sections 6.4 and 6.5 provide insight also in the behavior of other SD protocols, which implement the same operations, as will be discussed later.

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. Since the setup is slightly more complex than in the XeN testbed, we will detail a few of those steps below. It should be noted, however, that the *ExCovery* framework takes care of most of the steps during the actual execution of experiments.

Experiment Initialization

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 are chosen randomly to exchange UDP [186] packets at a given data rate. UDP is 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.

Experiment Measurement and Cleanup

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 the deadline of 30 seconds 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 timeouts of current SD protocols and the overhead involved, one experiment run usually takes between 45 and 90 seconds to complete. Runs can be rendered invalid due to 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. This means that the actual number of SD operations that were carried out was considerably higher than the number shown in Table 6.1.

Scenario-Specific Setups

The first two series of experiments cover the discovery of a single provider by a single client as in the scenario from Section 6.2.1 and in parts, from Section 6.2.2. In both series, the requesting node was *t9-105*. In one series the provider was *t9-154* to cover scenarios where both nodes are in the same building, thus, in one well connected cloud. In the second series the provider was *a3-119* at a maximum hop distance to *t9-105* to study the effects on SD responsiveness in scenarios where nodes are connected with sparse and possibly weak links. All other nodes participated in

Table 6.3 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	10
Bitrate per generator (kbit/s)	72	72	0…1725

randomly distributed load generation at various levels. The position of nodes within the network is depicted in Figure 6.2.

A separate series of experiments for the scenario from Section 6.2.3 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. The topology of these experiments is illustrated in Figure 6.3. 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.

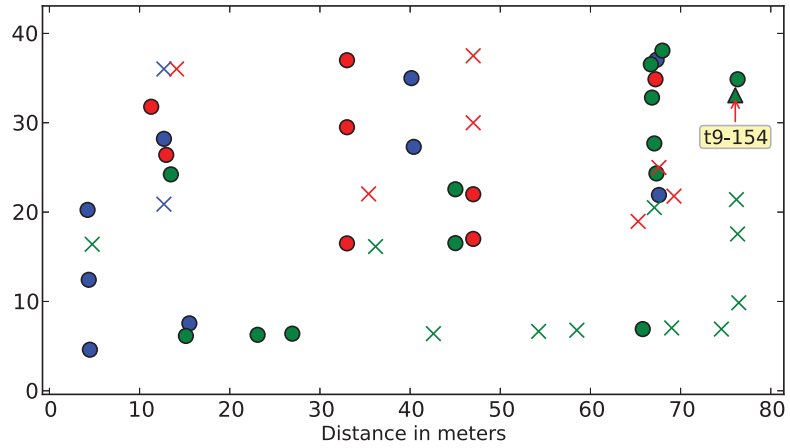


Fig. 6.3 Overview of building *t9* of the wireless DES testbed. Nodes colors define their building floors. Circle nodes are providers, crosses denote environment nodes for load generation. The requester *t9-154* has been labeled.

The most important configuration parameters of the three experiment series are summed up in Table 6.3.

6.4 Experiment Results

We will now present and discuss the results for the behavior of $R(t_D)$ under varying influences, as described in the three scenarios in Section 6.2. Each scenario was repeatedly run and observed in a series of experiments. In the virtual testbed experiments, the total number of experiment runs amounted to $60,000$. This equals the number of discovery operations for all scenarios combined, hence it is equal to the number of discovery requests without retries. The number of accumulated discovery responses reached $599,046$ over all runs. 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. A subset of these results was investigated for the defined SD scenarios, as shown in Table 6.1. The total number of experiment runs in the wireless testbed series was 32004 of which 26670 provided valid results for analysis. Due to the overhead involved in running those experiments, the series took several weeks to complete. The conditions for the validity of runs are explained in Section 6.3.2.

To foster repeatability, the raw measured data of all experiments in the XeN testbed has been uploaded to the AMBER Data Repository [149]. Interested researchers are invited to validate the results and perform further investigations. The full experiment data of the DES experiments, including experiment description, packet and event logs and all other measurements as explained in Chapter 3 are available for examination on request.

6.4.1 Scenario Results – Single Service Discovery

This analysis investigates in more detail how $R(t_D)$ for $t_D \in \{1,7\}$ seconds decreases with an increasing fault intensity in the network. According to the results in the virtual testbed, discovery of single service providers within the deadline $t_D = 7$ seconds proved to be reasonably responsive in networks with low packet loss. As illustrated in Figure 6.4, even with 30% packet loss, the responsiveness of single service discovery was well above 0.9 for the given deadline. Keeping in mind the focus of this scenario with lax requirements for the deadline, this is most probably also true for the required responsiveness. Considering a consumer context, such as home office or entertainment, a higher than 90% probability of success seems sufficient.

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 expected to be worse, which is not promising for use cases with stricter requirements. However, it is valid to conclude that given a deadline $t_D = 7$ seconds, with up to 20 percent packet loss and no additional negative effects, single service discovery is sufficiently responsive.

The picture is a different one for $t_D = 1$ second. Here, at a packet loss rate of 30%, a service will only be discovered every second try. The curve for $R(1)$ perfectly represents the expected behavior for discovery without retries. Since for SD

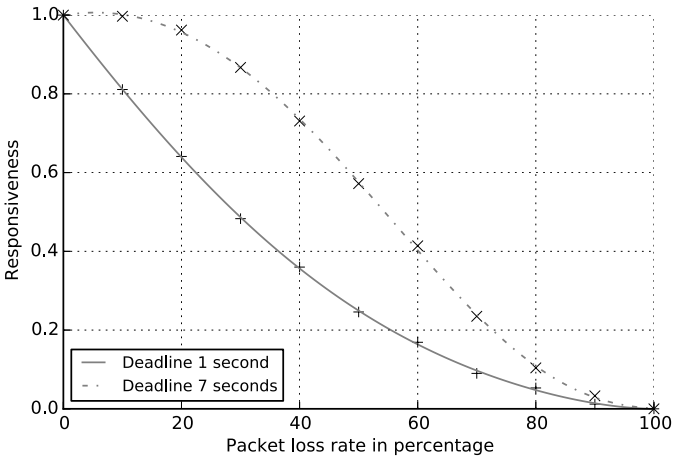


Fig. 6.4 Responsiveness of single service discovery with deadline $t_D = 1,7$ seconds as a function of packet loss rate.

to be successful, both the request and the subsequent response need to arrive at their destinations. Given a packet loss rate of P_{loss} in each direction, both packets arrive with a probability of $1 - P_{loss}$. So $R(1)$ may also be calculated using Equation 6.1. Values calculated by Equation 6.1 are listed in Table 6.4. They almost perfectly match the results from the experiment series in Figure 6.4. This validates the chosen experiment setup. With a more realistic fault model and an arbitrary number of retries, the calculation of responsiveness gets considerably more complex. Chapter 7 will present a hierarchy of stochastic models to do so.

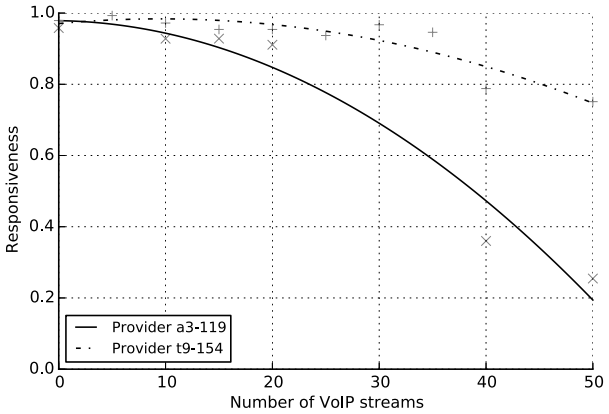
$$R(1) = (1 - P_{loss})^2 \tag{6.1}$$

Table 6.4 Expected responsiveness with a given packet loss rate for discovery without retries.

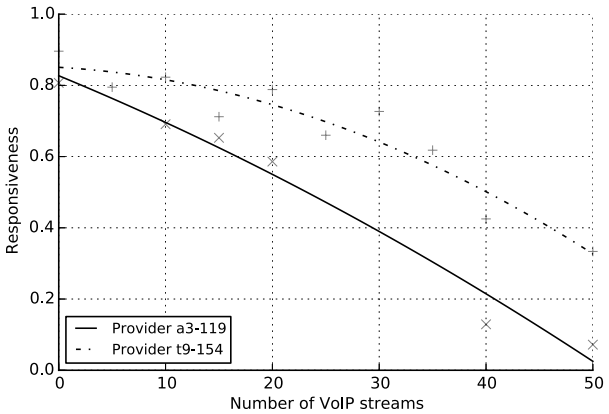
Packet Loss Rate	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
$R(1)$	1	0.81	0.64	0.49	0.36	0.25	0.16	0.09	0.04	0.01	0

In the wireless testbed, the results paint a comparable picture, although the differences are worth being explored. For this, $R(t_D)$ with $t_D \in \{1,7\}$ was calculated for each load level. The results are illustrated in Figure 6.5. The figures show for the two providers *t9-154* and *a3-119* how $R(t_D)$ decreases with increasing load. The curves in the figures are the regression functions over all data points of a specific provider.

Comparable is for both providers that until a certain point, the responsiveness decreases but does not drop to quickly. Depending on the distance from the requester,



(a) Deadline $t_D = 7$ seconds.) for two different providers.



(b) Deadline $t_D = 1$ second.) for two different providers.

Fig. 6.5 Responsiveness over increasing load without retries for two different providers.

however, there is a turning point after which the load gets too high and responsiveness rapidly goes below acceptable values. This is around 40% for $R(7)$ in the virtual testbed results. In the wireless setup, for short distances as provider *t9-154* this corresponds to 40 VoIP streams. For long distances as provider *a3-119*, responsiveness already drops with 20 concurrent streams. This effect is even stronger with shorter deadlines, as illustrated in Figure 6.5b. The curves for $R(t_D)$ do not continue along the regression slope forever. At higher fault intensities, the responsiveness effectively drops to zero because the whole service network breaks down, an effect that is also visible in the following results.

From all results can be deducted that the slope of the regression curve generally gets steeper with shorter deadlines. This means that the fault intensity has a

higher impact on $R(t_D)$ the less time there is to complete the discovery operation. In wireless networks, one reason for this behavior is the impact of delay on $R(t_D)$. However, this effect is not existent in the virtual testbed and warrants more in-depth evaluation in future research.

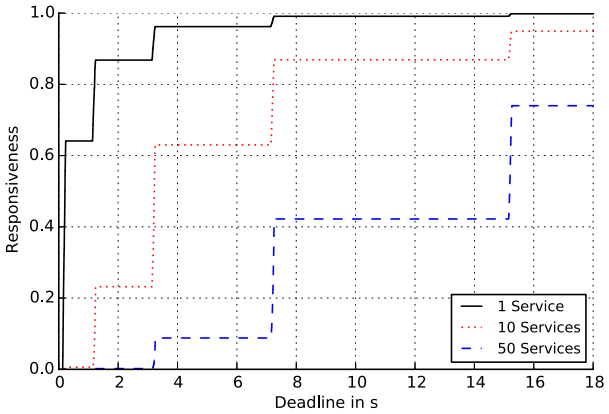
6.4.2 Scenario Results – Timely Service Discovery

The results in this section present responsiveness with increasing deadlines. The analysis of the virtual testbed data is illustrated in Figure 6.6. The characteristic steps in the curves mark the request retry events. Conforming to the fault model, a certain amount of requests and responses gets lost, so after each discovery request the responsiveness curve plateaus until the retry timeout, when a new request is being sent.

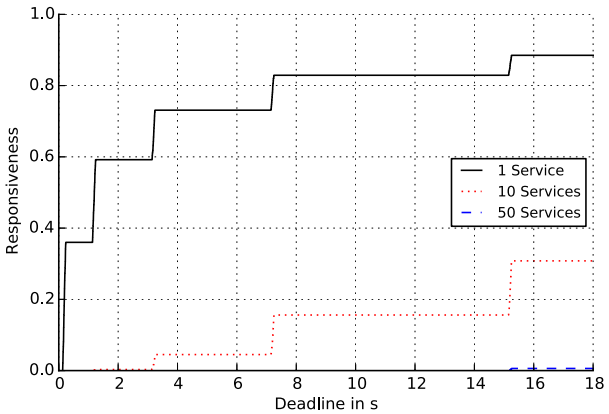
With 20% packet loss, the responsiveness of discovering a single service approaches 1 within the observed interval (Figure 6.6a). This corresponds to the conclusions from single SD at 7 seconds waiting time (see Figure 6.4). In fact, in this scenario discovering a single service reaches a responsiveness of almost 0.9 already after the first retry, after roughly one second. We can conclude that with low packet loss, current SD protocols work sufficiently well when discovering single service instances.

However, Figure 6.6a also 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 use cases when there is enough time to wait for discovery responses and, especially in the case of 50 instances, low responsiveness is acceptable. Those cases are expected to be rare in regular service networks. If short response times are needed, as they are for the use cases anticipated in the Internet of Things, we need a high responsiveness in shortest time. It can be seen in Figure 6.6a that both curves require a long discovery time, thus, a high deadline t_D to reach reasonable responsiveness.

With 40 percent packet loss, discovery of multiple services becomes highly improbable. Figure 6.6b illustrates this. There is a chance of less than $1/3$ to find all services instances in time when there are 10 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 such a comparable dramatic effect on responsiveness as with multiple services. This dramatic decrease in responsiveness when discovering more service instances will be further investigated in the next scenario in Section 6.4.3. The observed behavior is backed up by analytical evaluation. If we extend Equation 6.1 to the discovery of multiple service instances, with n being the number of instances, Equation 6.2 calculates the responsiveness at $t_D = 1$ second, just before the first retry would get sent.



(a) Packet loss 20%



(b) Packet loss 40%

Fig. 6.6 Responsiveness over time for a different number of providers.

This means that with a rate of $P_{loss} = 0.4$, $R(1)$ for $n = 10$ and $n = 50$ is $3.6 \cdot 10^{-3}$ and $4.85 \cdot 10^{-12}$, respectively.

$$R(1) = (1 - P_{loss})^{1+n} \tag{6.2}$$

It can be expected 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, thus, a high number of discovery responses. Without a more realistic network and fault model and a sound cost function for packet transmission it remains rather difficult to determine an optimal trade-off between responsiveness and discovery packet load.

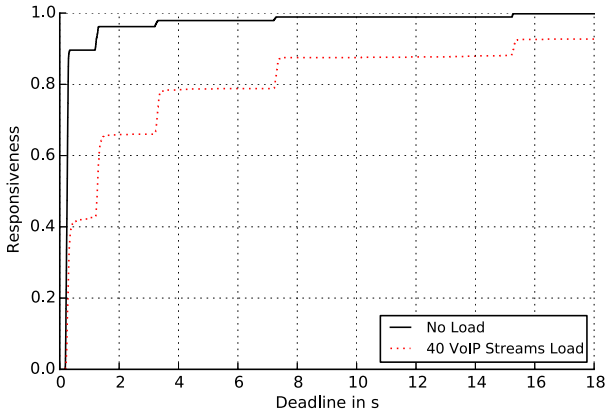
Using the results from experiments in the wireless testbed, we now examine how $R(t_D)$ increases with t_D, depending on the distance between requester and provider and the load in the network. To be able to isolate the effect of node positions on the results, only data from experiment Series I and II was used (see Section 6.3.2 and Table 6.3). 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 6.2. All DES testbed nodes participated in routing network packets.

Figures 6.7 and 6.8 illustrate the results. Figure 6.7 shows the responsiveness of the discovery operation over time, equivalent to Figure 6.6, for providers *t9-154* and *a3-119*. Also in the testbed the two curves in each graph reflect the responsiveness for different load conditions. As expected, 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 $2.8 Mbit/s$, 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. However, packet loss is not the only factor impacting $R(t_D)$. Especially at higher loads, the delay of individual packets becomes apparent, smoothening the steps in the curve. This effect is also visible in Figure 6.7b for provider *a3-119*.

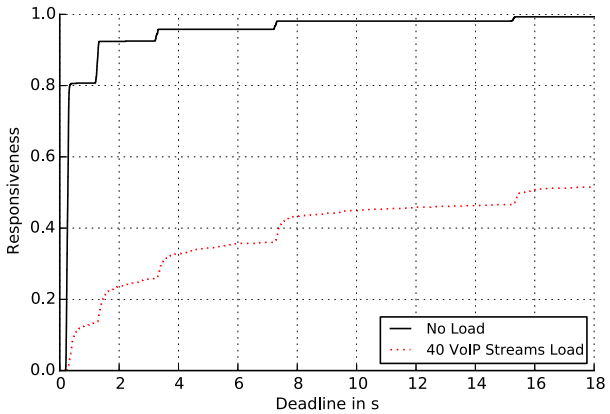
The results are in line with the ones from the virtual testbed. The current SD protocol retry configuration works well when conditions are close to perfect but their responsiveness decreases significantly when conditions deteriorate. Their static retry strategies struggle in unreliable networks. As can be seen later, the analytical models in Chapter 7 hint at possible advantages in conditions when SDPs (namely SLP [106] and SSDP [93]) use the more reliable unicast instead of multicast for certain messsages.

Packet Based Analysis

The discovery operation of a single service consists of multiple pairs request and response. Figure 6.8 shows the responsiveness of such individual packet 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 (Figures 6.7a and 6.8a, respectively 6.7b and 6.8b). 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 Figure 6.8 can be used, for example, as lower level input data in the analytical responsiveness models presented in Section 7.2.1



(a) Provider *t9-154*



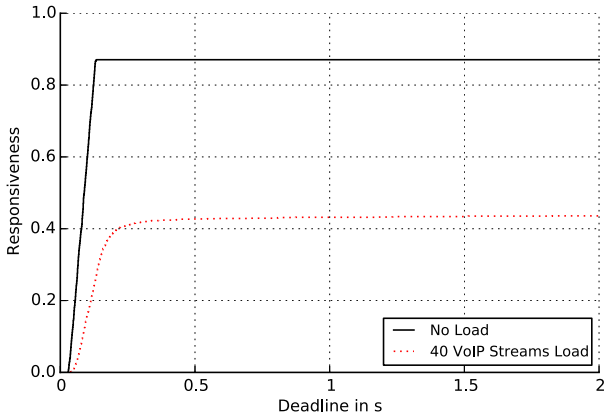
(b) Provider *a3-119*

Fig. 6.7 Responsiveness over time for two different providers under varying load. Complete service discovery operation including retries based on events at the discovery layer.

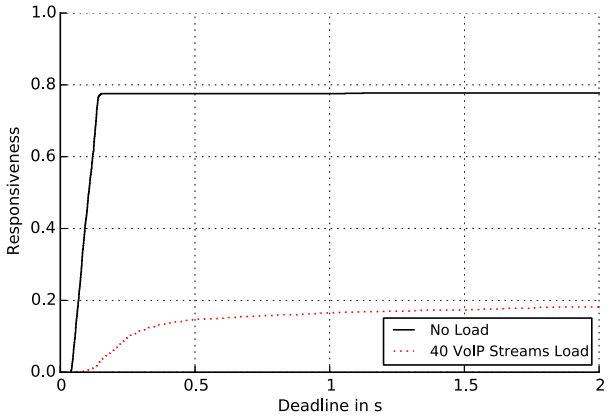
or when evaluating different protocols that use similar types of packets. We will also use them to validate the stochastic models in Chapter 8.

6.4.3 Scenario Results – Multiple Service Discovery

For the last scenario, we investigate the behavior of SD responsiveness when discovering a fixed percentage of service instances over an increasing number of service instances. Full coverage is required so 100% of service instances need to be dis-



(a) Provider *t9-154*



(b) Provider *a3-119*

Fig. 6.8 Responsiveness over time for two different providers under varying load. Packet-based analysis for individual discovery request and response pairs.

covered for successful operation. All instances belong to the same service type – a single discovery request should discover all instances if no packets are lost. 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. It could already be deduced from the previous scenario that SD responsiveness decreases dramatically when discovering ten or more services in lossy networks. Figure 6.9 shows an articulate explanation for this behavior.

With low packet loss rates responsiveness decreases almost linearly with the number of nodes in the network. This is the reason why with 20% packet loss we

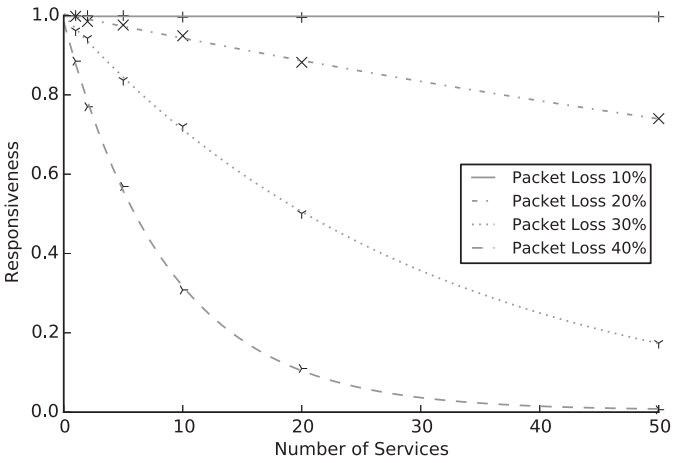


Fig. 6.9 Responsiveness of Zeroconf service discovery after four retries (within 20s)

still have a relatively high probability of almost 80% to discover all 50 service instances in time (see Figure 6.6a). If more packets get dropped the quality of SD worsens rapidly to an exponential decrease with the number of nodes in the network. In practice, SD that requires a high coverage becomes unusable when dealing with a higher number of services. Current SD methods are usually not operating in such scenarios yet but the envisioned Internet of Things will demand that.

For the results of the wireless experiments, the data used for this analysis are from experiment Series III (see Section 6.3.2 and Table 6.3). One requester and 30 providers are deployed in the dense mesh cloud of building *t9*. This is done to make sure that the distance of providers to the requester has less impact on the results. Instead, we want to see how discovery performs under varying load conditions. Because there are fewer nodes for load generation, the data rate per generator pair is increased for a fixed number of 10 node pairs. Environment nodes and providers are distributed randomly in the network. The topology can be seen in Figure 6.3, which also shows the position of the different types of nodes, requester, provider and environment.

In the analysis, it is examined for different load levels how the responsiveness decreases with an increasing number of providers needed for successful discovery. Results are depicted in Figure 6.10. 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 $1.7 Mbit/s$. With the given distribution of nodes, this is also the saturation level. Results do not get significantly worse with higher load. At the same time, the network does 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, mainly in the preparation and cleanup phase of every single experiment run.

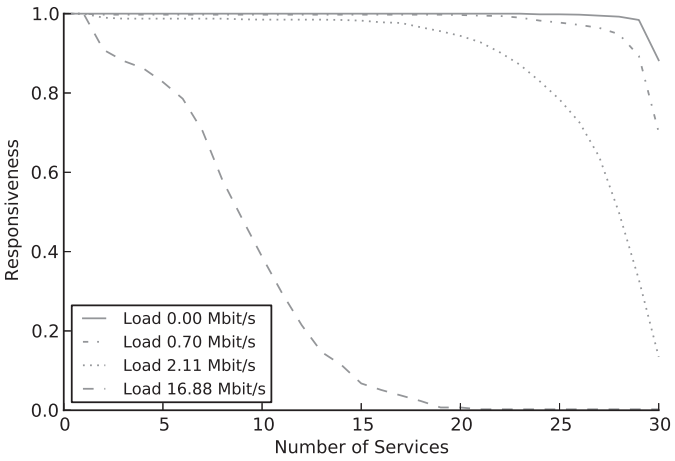


Fig. 6.10 Responsiveness over number of needed service providers for different load levels.

The results demonstrate that deploying more services is not generally a promising solution to increase responsiveness in wireless mesh networks. It confirms the findings from the virtual testbed experiments, albeit there is no linear decrease of responsiveness with lower load. One can see that at very high loads, discovering more than 20 nodes is impossible. Although not shown in Figure 6.10, there is an almost identical set of discovered nodes over all runs at that load level. This is because some nodes, although not partitioned from the rest of the network, are effectively blocked off SD by environment nodes that are generating load at very high data rates in their vicinity. Further research on the transmission mechanisms (unicast or multicast) and the retry intervals is needed to optimize responsiveness in such scenarios.

6.5 Experiment Results of Testbed Behavior

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 Section 3.2.2 and more in detail in [9]. Such stable and known conditions could only be achieved in the XeN-based virtual testbed. In the DES testbed, which is 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 took several weeks to complete, a better knowledge of the testbed behavior is

necessary. Although we cannot determine the source of all nuisance factors, we can measure them and show their effect.

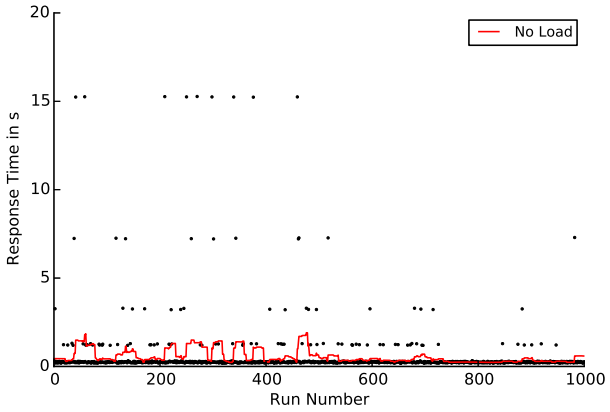
This section presents general measurements of the behavior of the DES testbed for the duration of the experiments. These cover the variation of response times under similar conditions (see Section 6.5.1), the varying quality of individual links (see Section 6.5.2) and the clock drift of the network nodes (see Section 6.5.3) 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 shown in Section 6.4. It should be pointed out that results can only be representative for the scenarios which have been 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.

6.5.1 Response Times over All Runs

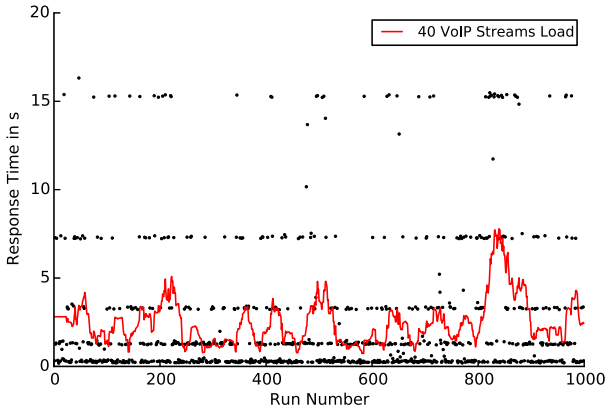
Figures 6.11 and 6.12 illustrate the effects of internal an external faults on the results for a part of four sets of experiments from Series I and II (see Table 6.3). The response times of discovery operations, which in this case are the times for the first response to arrive at the requester, are plotted over 1000 experiment runs. The requesting node is *t9-105*, the providers are *t9-154* (Figure 6.11) and *a3-119* (Figure 6.12). 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 6.2. For each node pair, results are shown without additional load in the network (upper Figures 6.11a and 6.12a) and with an additional load of 40 *Voice-over-IP* (VoIP) streams with 72 kbit/s each, which amounts to roughly 2.8 Mbit/s traffic overall in the network (lower Figures 6.11b and 6.12b).

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 of responses. This corresponds to the default retry intervals of the *Zeroconf* discovery protocol as already visible in Figure 6.7. *Zeroconf* starts with a timeout of one second and then doubles this timeout on every retry (see Table 2.1). 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 6.12b, 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 has been examined in more detail in Section 6.4, it can be noted that the response times are not independent



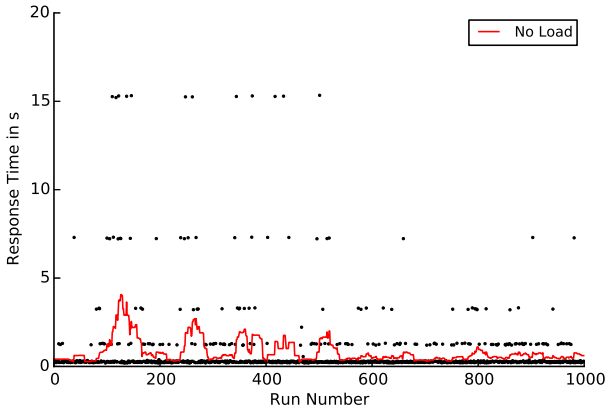
(a) Without additional load in the network



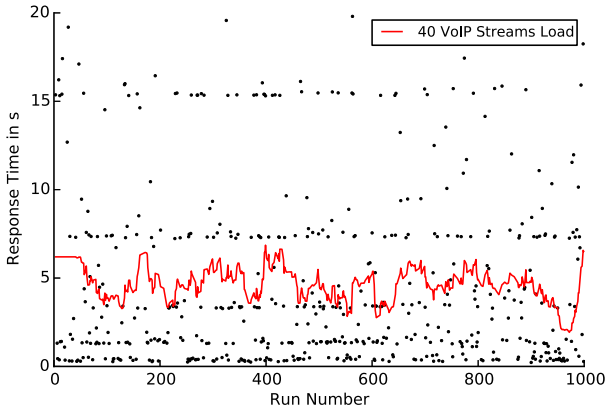
(b) With additional load of 40 VoIP streams with 72 kbit/s each

Fig. 6.11 Response times in testbed over 1000 experiment runs. Requester is *t9-105*, provider is *t9-154* at short distance in the same cloud as the requester.

over time. There are periods of consistently higher response times, such as in Figure 6.12a between runs 120 and 180 or in Figure 6.11b 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. Choosing 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



(a) Without additional load in the network



(b) With additional load of 40 VoIP streams with 72 kbit/s each

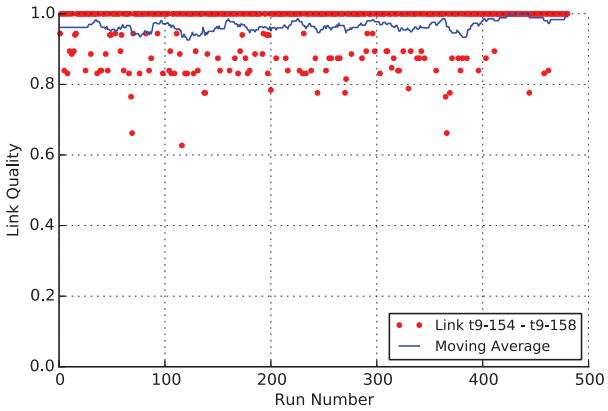
Fig. 6.12 Response times in testbed over 1000 experiment runs. Requester is *t9-105*, provider is *a3-119* at maximum distance from the requester.

data, as does the *ExCovery* framework (see Chapter 3), to be able to detect and visualize such effects.

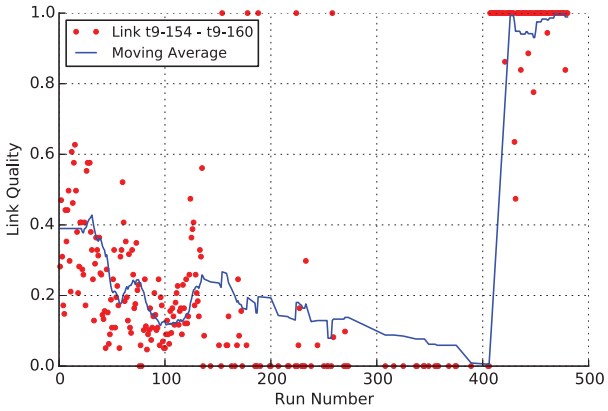
6.5.2 Topology Changes over Time

The variations in overall response times hint at changing link qualities over time. This why the *ExCovery* framework supports full topology recordings including link qualities at the end of each run in a series of experiments. More details on these

measurement can be found in Section 3.4.1. We will now present an analysis of those measurents which is illustrated in Figure 6.13.



(a) Neighbor *t9-158*



(b) Neighbor *t9-160*

Fig. 6.13 Link qualities of the two neighbors of *t9-154* over 500 experiment runs, equivalent to 8 hours experiment time.

The figure shows variations in link quality over time for two different neighbors of node *t9-154*, which has been extensively used during the experiments. Node positions in the DES testbed can be seen in Figure 6.14. On the *y*-axis are link qualities as a ratio of *Open Link-State Routing* (OLSR) hello messages between the two nodes that did not get lost. On the *y*-axis is the experiment run number. The red dots are individual OLSR link quality measurements before each run while the line represents a moving average of 20 runs. One can see that the neighbor node *t9-158* generally provides a very good quality over the observed period. Most of the time

the hello packet loss is close to zero. Node *t9-160* shows a very different behavior. During the first 250 runs, the link quality to *t9-154* is very weak. Also, the variations in quality are very inconsistent. Until run 400 the link is basically down. Then, the link quality suddenly becomes almost perfect and remains so until the end of the observed period. This erratic behavior of course has an impact on the routing within the network and thus, on the propagation of multicast packets which are the base of SD.

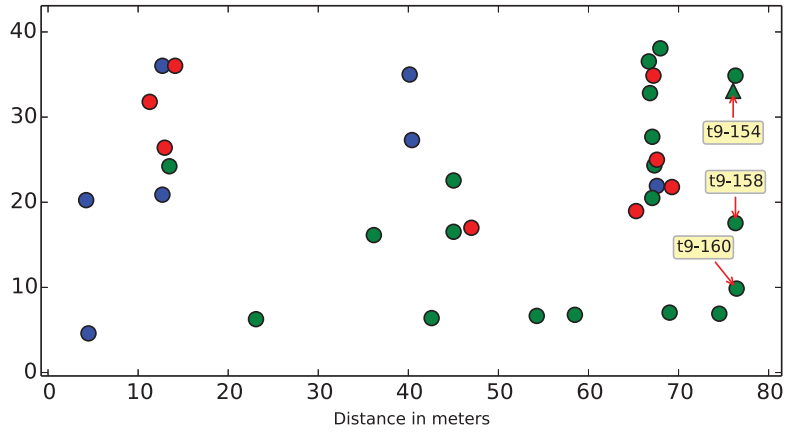


Fig. 6.14 Overview of building *t9* of the wireless DES testbed. Nodes colors define their building floors. The node *t9-154* and its neighbors *t9-158* and *t9-160* have been labeled.

6.5.3 Node Clock Drift over Time

While Sections 6.5.1 and 6.5.2 show 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 begin to drift apart.

Figure 6.15 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 expectedly drift differently from the others, the drift of the individual nodes even varies over time. This 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.

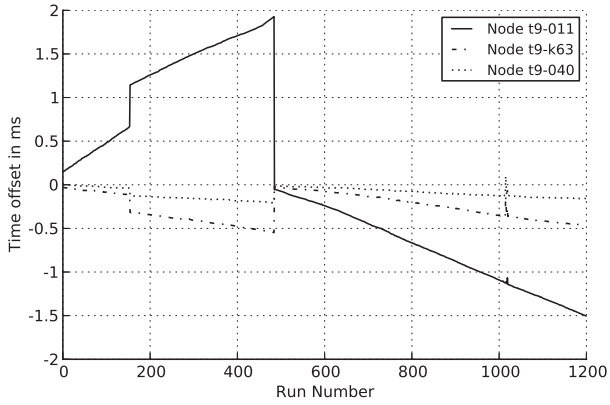


Fig. 6.15 Absolut clock differences between different providers and requester *t9-154* over a duration of 1200 experiment runs, approximately 20 hours.

Clock synchronization is highly important for the analysis of real-time problems, such as SD. 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.

6.6 Conclusion

In this chapter, we examined decentralized *Service Discovery* (SD) in experiments with different fault intensities to get an initial idea about the behavior of SD responsiveness. Two experiment environments were setup to observe three common

scenarios. A virtual testbed with a fault model that includes only packet loss and the *Distributed Embedded Systems* (DES) wireless testbed at Freie Universität Berlin. Analysis in the DES testbed was 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. This will be demonstrated in Chapter 8.

Second, an analysis of the long-term testbed behavior and the effects of internal and external faults has been 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 has been shown that the effect of these faults is considerable and has to be taken into account when interpreting results gathered in the testbed.

Chapter 7 provides a hierarchy of stochastic models to evaluate user-perceived responsiveness of SD. Parts of the results presented in this work are used to validate the presented models in Chapter 8. Comparable experiments were carried out using *Service Location Protocol* (SLP). They confirm the results shown here for *Zeroconf* and have been left out for reasons of brevity. More information about these SLP experiments can be found in [61].

Chapter 7
Modeling Service Discovery Responsiveness

Abstract The focus of this work is the responsiveness of the discovery layer, the probability to operate successfully within a deadline, even in the presence of faults. This chapter proposes a hierarchy of stochastic models to evaluate responsiveness of decentralized discovery. The models are used to describe the discovery of a single service using three popular protocols. Furthermore, a methodology to use the model hierarchy in *Wireless Mesh Networks* (WMNs) is introduced. Given a pair requester and provider, a discovery protocol and a deadline, the methodology generates specific model instances and calculates responsiveness. To estimate the behavior of multicast communication, which is fundamental to all current protocols, the methodology uses the Monte Carlo method *Probabilistic Breadth-First Search* (PBFS). PBFS accurately approximates multicast behavior in WMNs and it does so with low time and space complexity compared to existing approaches. Finally, this chapter introduces a new metric, the expected responsiveness distance d_{er}, to estimate the maximum distance from a provider where requesters can still discover it with a required responsiveness. Using monitoring data from the DES testbed at Freie Universität Berlin, it is shown how the models can be used to calculate responsiveness and d_{er} of current discovery protocols depending on the position of nodes and the link qualities in the network.

7.1 Introduction

This chapter provides a hierarchy of stochastic models to evaluate responsiveness of decentralized *Service Discovery* (SD) in unreliable networks. It provides a methodology to apply these models to *Wireless Mesh Networks* (WMNs). The stochastic model hierarchy and the Monte Carlo method *Probabilistic Breadth-First Search* (PBFS) to estimate the reachability of multicast packets have been discussed in [4, 11]. This chapter refers to these publications where necessary and provides a comprehensive overview of how the different contributions are interconnected.

The remainder of this chapter is structured as follows. A hierarchy of stochastic models to evaluate SD responsiveness is introduced in Section 7.2, followed by the description and validation of the multicast estimation method PBFS in Sections 7.3 and 7.4, respectively. An overview of a methodology that creates and solves the models can be found in Section 7.5. The case study in Section 7.6 shows the responsiveness of SD in different scenarios as calculated by the models. Results are explained and interpreted. Section 7.7 concludes the work.

7.2 A Model for Service Discovery

When doing SD, the number of requesters and providers may vary. For instance, multiple clients might request a single service. One client could discover all existing providers in the network to choose one meeting best its requirements. We will now focus on decentralized SD by a single client as it is the basis of all other SD operations. Any such SD operation can be described with a generic family of Markov models, which includes three types of states:

- A single state named req_0 defines the beginning of the SD operation, when the initial request has been sent.
- Two absorbing states *ok* and *error* define the successful or unsuccessful end of the SD operation.
- A set containing every state between the first two types where not all required responses have been received and the final deadline has not been reached.

The Markov model family is parametric in two parameters: number of retries and required coverage. The maximum number of retries n describes the first dimension of the model family. Beginning from the initial state req_0, the retries define a chain of retry states $req_i,\ i = 1...n$, that stand for "retry i has been sent". From each retry state the model can transition into *ok* in case a sufficient number of responses was gathered. If not it will transition to the next retry state and eventually from req_n to *error*. The second parameter required coverage describes how many responses need to be received before an SD operation is called successful and is the second dimension of the model family. Each of the retry states $req_i,\ i = 1...n$ becomes a set of states $req_{ij},\ j \geq 1$, depending on the success ratio when doing retry i. The size of this set may be arbitrary but as an example, if three services need to be found for successful operation, there could be three states $req_{ij},\ j = 0,1,2$ for every retry i that stand for "retry i has been sent and j responses have been received so far". Figure 7.1 exemplary depicts a model in this family with two retries and three states to reflect the different success ratios.

Estimating the transition probabilities within this Markov model family is not trivial. In the following, we propose a hierarchy of stochastic models where the probabilities of these high level discovery models are calculated by low level models based on link quality data measured in the network. The hierarchy consists of the following three layers:

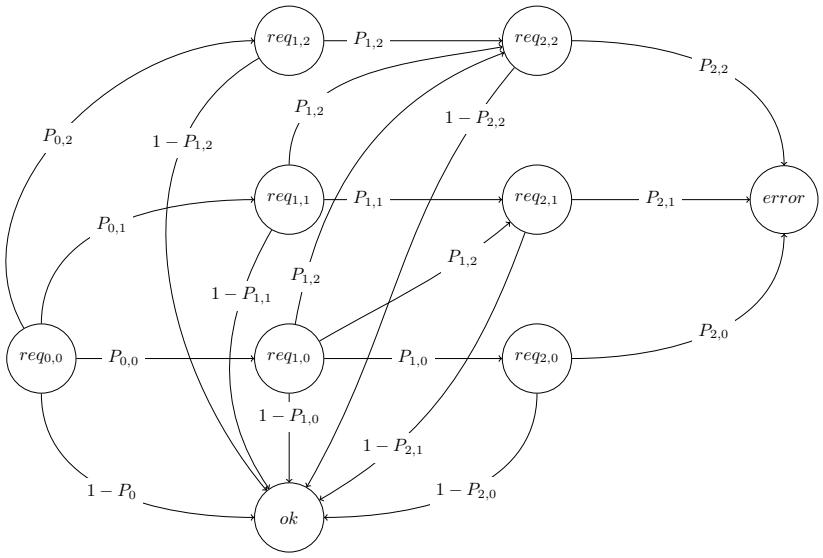


Fig. 7.1 Markov chain for service discovery model family.

1. **Service Discovery Operation** (see Section 7.2.1) – A discrete time Markov model reflecting the overall SD operation at the application protocol level, including discovery requests and their retries. Calculates the SD responsiveness for a specific scenario.
2. **Retry Operation** (see Section 7.2.2) – A semi-Markov model describing individual SD requests and their respective responses. Used to calculate the probabilities in the upper level discovery model, can be replaced by packet based measurements applicable to the evaluated scenario.
3. **Network Mapping** (see Section 7.2.3) – Extensions to the retry operation model to map the abstract retry operation to a concrete network under analysis. Uses PBFS to estimate parameters of multicast communication (see Section 7.3.

7.2.1 Service Discovery Model

To demonstrate the model hierarchy, we instantiate a specific model for discovery of a single service within a deadline $t_D = 5s$ using the *Zeroconf* protocol. The number of retries n can be derived by examining the retry strategy of the SDP under analysis (see Table 2.1). Given the times in seconds $t_{retry}(i), i \in \mathbb{N}^+$ between retries $i-1$ and i with $t_{retry}(0) = 0$. The total time $t_{total}(r)$ after the beginning of a discovery operation when sending retry r is calculated according to Equation 7.1.

$$t_{total}(0) = 0, \quad t_{total}(r) = \sum_{i=1}^{r} t_{retry}(i), \quad r \in \mathbb{N}^{+} \tag{7.1}$$

For Zeroconf, $t_{total}(2) < t_D < t_{total}(3)$. So, $n = 2$ retries will be sent. The resulting regular Markov model instance is depicted in Figure 7.2: In short, retries continue to be sent until a response is received, triggering a transition to the *ok* state. If no response is received after retry n has been sent and before t_D, the operation is considered failed by transitioning to state *error*. So, the discovery operation is successful as soon as the first response packet arrives at the requester. Transitions between the retry states will only happen if no response has been received until the specific retry timeout.

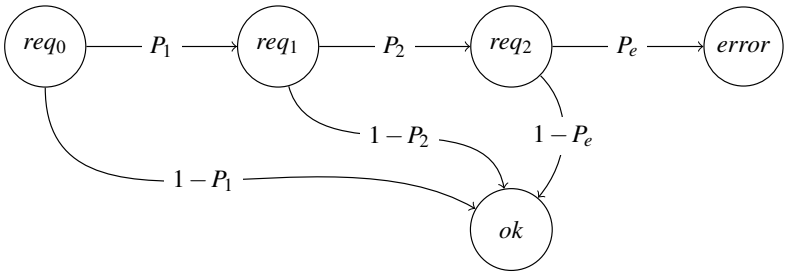


Fig. 7.2 Markov chain for single service discovery

There are two related probabilities in Figure 7.2: $P_r, 1 \leq r < n$ is the probability that no discovery response was received between $t_{total}(r) - t_{retry}(r)$ and $t_{total}(r)$. P_e is the probability that no response was received between $t_{total}(n)$ and t_D. Arrival times of responses to a specific request r can be considered as a random variable X_r. Equation 7.2 describes the cumulative distribution of this variable, the probability that a response to request r has arrived by time t or, the *responsiveness* $R_r(t)$ of a single request-response operation for request r.

$$F_{X_r}(t) = P\{X_r \leq t\} = R_r(t) \tag{7.2}$$

Knowing this, Equation 7.3 calculates the probability that a response to request r arrives in a specific time interval.

$$P\{t_x \leq X_r \leq t_y\} = R_r(t_y) - R_r(t_x) \tag{7.3}$$

Functions $Pr : \mathbb{N}^{+} \to [0,1]$ and $Pe : \mathbb{N}^{+} \to [0,1]$, as defined in Equations 7.4 and 7.5, can now calculate P_r and P_e such that $Pr(r) = P_r$ and $Pe(n) = P_e$.

$$Pr(r) = \prod_{i=1}^{r} \left(1 - \frac{R_i(t_{total}(r)) - R_i(t_{total}(r-1))}{1 - R_i(t_{total}(r-1))}\right) \tag{7.4}$$

$$Pe(n) = \prod_{i=1}^{n} \left(1 - \frac{R_i(t_D) - R_i(t_{total}(n))}{1 - R_i(t_{total}(n))} \right) \tag{7.5}$$

Since Equation 7.5 is a special case of Equation 7.4, only Equation 7.4 is explained in detail. A retry is forced when no response packet arrived until the retry timeout, so the product multiplies the individual probabilities for non-arrival of responses to each request that has been sent so far. The probability for the response to request i to arrive *within the specific interval* is described by the quotient. The numerator describes the unconditional probability for a response to arrive within the specified interval. But, deducing from the structure of the Markov model, it cannot have arrived before. That condition is given in the denominator. The quotient, thus, gives the probability that a response to request i is received in the specified time interval, provided it has not arrived before. It is then subtracted from 1 to get the probability of non-arrival.

Missing is a specification to calculate the functions $R_r(t)$. One way would be to measure response times of request-response pairs and fit a distribution to them, such as the data shown in Sections 6.4.2 and 6.5.1. We will use such data to validate the discovery model in Chapter 8. Additionally, for the cases where no comparable data is available, we provide an analytical solution, using a retry operation model.

7.2.2 Retry Operation Model

When discovering a single service, each retry step relates to a request-response pair, described by the semi-Markov process in Figure 7.3. In state *Rq*, a request has been sent. When it arrives at the destination, the provider will send a response, triggering a transition to state *Rp*. As soon as this response arrives back at the requester, the model enters state *ok*. If one of the messages gets lost, it will transition to state *error*. This will cause the SD process to reach a timeout and force a retry, which again will follow that pattern.

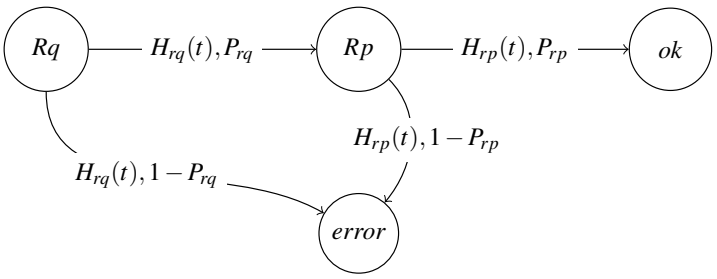


Fig. 7.3 Semi-Markov chain for a single request-response pair

In case messages arrive (with a probability of P_x), they have a certain distribution of arrival times. This is described by $H_x(t)$, the sojourn time distribution for state x. The cumulative distribution function of time to absorption in state *ok* now calculates $R_r(t)$ as in Equation 7.2. Since t is relative to the beginning of the SD operation, $R_r(t)$ is in fact parametrized by the location $t_{total}(r)$, the time at which retry r is being initiated. The retry operation model is independent of a concrete network infrastructure. It has no knowledge of how to calculate the probabilities and transition time distributions. Providing concrete values of P_x and $H_x(t)$ for specific SD pairs on demand is the purpose of the network mapping model.

7.2.3 Network Mapping Model

Mapping requests and responses to the network under analysis means providing models that calculate P_x and $H_x(t)$ in the retry operation model (see Figure 7.3) by taking into account the details of the used communication mechanism, unicast or multicast. This mapping is dependent on the concrete network infrastructure. In this case, we provide models for a WMN running OLSR [57]. Different networks could need diverse models, but, provided they estimate P_x and $H_x(t)$, these could be used in the proposed model hierarchy as well.

Unicast Model

A unicast message follows the shortest path according to the routing metric. In OLSR, every node periodically calculates that shortest path and saves the next hop for every destination. A unicast message is sent to the next hop node which then decides to forward according to its own next hop information for the destination. We use an algorithm that calculates the unicast path hop by hop based on the next hop information on each node. If no such global information is available, using the shortest path known to the first node remains a viable, albeit less accurate solution.

Since there is only one path with n nodes and $m = n-1$ links, this can be modeled as a simple semi-Markov chain of n states. Each state $k_i, i = 1...n-1$ stands for "message forwarded by node i", state k_n means "message arrived at node n". The links between nodes i and $i+1$ become transitions $k_i \rightarrow k_{i+1}$. Further, there is a transition from $k_i, i = 1...n-1$ to *error* to account for packet loss. State transition probabilities $P_{k_i,k_{i+1}}$ are calculated from the currently monitored packet transmission probabilities of the link between nodes i and $i+1$ (see Section 7.5), taking into account that unicasts will be retransmitted up to seven times if not acknowledged by the receiving node. The estimation of sojourn time distributions $H_{k_i,k_{i+1}}(t)$ is described in Section 7.2.4. The resulting unicast chain is then integrated into the retry operation model in Figure 7.3 – for a unicast response, for example, by merging states k_i and Rp, k_n and *ok* as well as the two error states. The rest of the chain replaces transition $Rp \rightarrow ok$.

Multicast Model

In theory, modeling the traversal of a multicast discovery packet should consider all possible paths between source and destination. Multicasts are in essence *Network-Wide Broadcasts* (NWBs) on a subgraph of the complete network, which contains all nodes taking part in the service network. This redundancy has been taken into account in the methodology for service availability evaluation in Part II but finding all paths between two nodes is *NP*-complete, a prohibitive complexity in networks with high connectivity, such as WMNs. Since the vast majority of those paths has a very low probability of traversal and their impact on the responsiveness of the multicast communication would be minor, this work instead proposes a different method to estimate the reachability of multicast communication. *Probabilistic Breadth-First Search* (PBFS) is a Monte Carlo method to derive an estimation of the multicast path length between given nodes that communicate using flooding or variations thereof. Because of its extensive use when estimating multicast communication within the SD models, PBFS needs a detailed description, which can be found in Section 7.3.

In PBFS, node neighbors are only considered if the edge between a node and its neighbor succeeds a random roll against its transmission probability, as monitored by the routing layer (see Section 7.5). This way, each run of PBFS realistically simulates how a multicast packet would traverse the WMN. PBFS is sampled a sufficient number of times to approximate with which probability the destination node could be reached. This reflects P_x in the retry operation model, for example, P_{rq} for a multicast request. We additionally store the probability for each path length in case of arrival to later estimate the distribution of sojourn time $H_x(t)$ in Section 7.2.4.

7.2.4 Transmission Time Distributions

Estimations for sojourn time distributions in the network mapping models are based on the (re-)transmission and potential back-off periods defined in the 802.11 standard [120]. For transmission times over links, we assume the lowest data rate, which is correct for multicasts. Unicasts, however, will transmit at higher data rates if possible, reducing the time for individual retry transmissions. The estimation thus presents an upper bound for the transmission time as dependent on the data rate. The estimation also ignores additional contention due to internal traffic or external interference, which affects the upper limit of transmission times. To account for this, a certain percentage of packets is assumed to arrive *after* the estimated maximum transmission time for both uni- and multicasts. The bounds calculated from these assumptions are fitted to an exponential distribution for the transmission time. For the unicast model, this is done for each transition $k_i \rightarrow k_{i+1}, i = 1...n-1$ and provides $H_{k_i,k_{i+1}}(t)$. In the multicast model, one distribution function is generated for each possible path length given by PBFS. The distributions are then weighted with the corresponding probability for their length and combined in a single function $H_x(t)$.

7.3 Multicast Reachability Estimation

As previously described in Section 2.2.2, all SDPs use multicast communication for crucial parts of the discovery process. However, due to the characteristics of wireless communication, NWBs are generally problematic in WMNs. Being able to estimate NWB dissemination in such networks is thus fundamental when estimating packet traversal probabilities and traversal times, as needed for the lower-level models in Sections 7.2.3 and 7.2.4. Most existing dissemination models neglect the real nature of WMNs and are based on simple graph models, which provide optimistic assumptions of NWB dissemination. On the other hand, models that fully consider the complex propagation characteristics of NWBs quickly become unsolvable due to their complexity.

In this section, we present the details of the Monte Carlo method PBFS to approximate the reachability of NWB protocols. PBFS has been described in more detail in [11]. Without reverting to expensive empirical or analytical methods, PBFS can be used to realistically approximate various metrics, such as reachability or average path length. PBFS simulates individual NWBs on graphs with probabilistic edge weights, which reflect link qualities of individual wireless links in the WMN, and estimates reachability over a configurable number of simulated runs. This approach not only has a very low time and space complexity compared to existing approaches, but further provides additional information, such as the distribution of path lengths. Furthermore, it is easily extensible to NWB schemes other than flooding. The applicability of PBFS is validated both theoretically and empirically in Section 7.4.

7.3.1 Network Model

A WMN can be modeled as an undirected graph $G = (V, E)$, where each vertex $v \in V$ corresponds to a WMN node and each edge $e \in E$ corresponds to a wireless communication link between two nodes. In *Quasi Unit Disk Graphs* (QUDGs) [132], there is a communication link between two nodes if their Euclidian distance is less than d (with $0 < d < 1$) and none if it is greater, otherwise unspecified. Thus, a link may or may not exist within this uncertainty range. This neglects that links itself are rarely perfect or completely non-existent, but their quality varies in between. For example, there might be good links which succeed in transmitting packets 85% of the time, while others succeed only 10% of the time. Thus, in the model proposed in this section wireless links are not just binary but approximated with a transmission probability $p_{i,j}$, the probability a transmission from node i to node j succeeds (see Equation 7.6). If $p_{i,j} = 0$, the edge is not included in the graph.

$$p_{i,j} : E \rightarrow [0,1] \tag{7.6}$$

In practice, $p_{i,j}$ usually reflects a measurement-based approximation. Link qualities are never constant but vary over time due to small-scale fading effects, micro-

mobility of the environment and interference. Node outages may also happen. The graph model assumes independence between the qualities of different links, any dependence is expected to be already included in the approximation of $p_{i,j}$. For the application in the context of this work, SD in WMNs (see case study in Section 7.6), static networks with no node mobility are assumed. However, the method described in Section 7.3.4 can also be adjusted to support network topology dynamics reflected in varying values for $p_{i,j}$.

7.3.2 Reachability Metrics for Network-wide Broadcasts

To evaluate the performance of NWBs within the weighted graph model, we define reachability as the main metric, which captures an important aspect of the dependability of a NWB. We distinguish between the following two reachabilities:

Individual reachability $r_{a\to b}$ – The probability that a node b is reached by a NWB initiated by node a. In practice, $r_{a\to b}$ is averaged over multiple runs as a single run will only give a binary value.

Global reachability r_a – The percentage of nodes that receives a NWB sent by node a. Let R be the number of nodes reached by a NWB initiated from node a. The global reachability is defined as $r_a = \frac{R}{|V|}$ and derived from a single NWB or averaged over multiple NWBs. For a general statement about the performance of a protocol or a given topology this should be averaged over a large number of runs. Another way to derive the global reachability would be to average all individual reachabilities.

Global and individual reachabilities depend on the given network topology with its transmission probabilities, NWB protocol, NWB source node and, for individual reachability, the destination node. Source independent results can be achieved by considering NWBs from every node in the network and averaging over all of them.

7.3.3 Calculation of Reachability

There exist purely analytical methods to calculate the reachability metrics based on the described weighted graph model. For example, Chen et al. [50] have shown how the individual reachability $r_{a\to b}$ can be obtained by summing up the probabilities of the family of edge sets that contain a path from a to b, as in Equation 7.7: $2^E_{a\to b}$ is the set of all subsets of edges that contain a path from a to b. Calculating this formula has an exponential complexity due to the iteration of the power set.

$$r_{a\to b} = \sum_{E' \in 2^E_{a\to b}} \left(\prod_{(i,j)\in E'} p_{i,j} \prod_{(k,l)\in E\setminus E'} (1 - p_{k,l}) \right) \tag{7.7}$$

Another possible approach is to enumerate all given paths between a and b and calculate the probability of not reaching node b over any of these paths. Enumerating all simple paths between two nodes is NP-hard for arbitrary graphs. Furthermore, as these paths are not independent and can contain common subpaths, this approach requires factoring out these common subpaths, which is again NP-hard, a prohibitive complexity in highly connected WMNs. However, for certain networks with a low connectivity, hence, a low in-degree of the nodes, this approach is viable and we are taking it into account in Part II when presenting an approach to automatically generate user-perceived service availability models.

Chen's algorithm for the *Flooding Path Probability* (FPP) [50] also computes the individual reachability $r_{a\to b}$. It resorts to computing the joint probability distribution that a packet is received by a subset of the vertex cut in each step of its algorithm. The complexity is $O(N(C\Delta^+ + 2^C\Delta^-))$, where C is the size of the largest vertex cut used by the algorithm, Δ^+ is the maximum out-degree and Δ^- the maximum in-degree of the graph. Therefore, calculating the FPP is efficient if the network has only a small largest vertex cut, which is not true for high connectivity WMNs.

Given their complexity, these existing approaches are only feasible in simple networks. To support the reachability evaluation of NWBs as used by discovery protocols in WMNs, a more efficient method is needed. Equation 7.7 will be used in Section 7.4.1, however, to validate the proposed PBFS in exemplary small networks.

7.3.4 Probabilistic Breadth-First Search

We will now introduce the Monte Carlo method *Probabilistic Breadth-First Search* (PBFS) that models flooding as a slight modification of *Breadth-First Search* (BFS) [87] in a weighted graph as described in Section 7.3.1. In contrast to regular BFS, in PBFS, for a node i its neighbor j in the graph is only considered if the edge between i and j succeeds a random roll against its transmission probability $p_{i,j}$. Consequently, PBFS can be seen as a flooding process in a graph whose edges are weighted with independent transmission probabilities, abstracting from wireless effects such as contention and collisions. This way, each run of PBFS simulates how a NWB packet, for example a discovery request, would traverse the WMN. It does so without resorting to complex network simulations or testbed experiments as in Chapter 6, and with considerably lower complexity than the purely analytical evaluations in Section 7.3.3. The algorithm has previously been published in [11] and is presented in Figure 7.4.

The global reachability r_{src} is ratio of the number of nodes that were marked (*reached*) to the number of all nodes. In real networks, for example, the probabilistic edge weights $p_{i,j}$ in the graph can be obtained by doing link probing for the ETX metric as described in Section 7.5. The PBFS procedure in Figure 7.4 samples a sufficient number of times to approximate with which probability a destination node b is reached by a NWB from node a (the individual reachability $r_{a\to b}$). This reflects P_x in the retry operation model from Section 7.2.2, for example, P_{rq} for a multicast

```
 1: procedure PBFS(G, src)                                  ▷ G provides nodes and p_{i,j}.
 2:     mark[n] ← False ∀ n ∈ G                             ▷ BFS visited marker.
 3:     NoT[n] ← 1 ∀ n ∈ G                                  ▷ Number of transmissions setup.
 4:     reached ← 0                                         ▷ How many nodes were finally reached.
 5:     nodes ← [src]                                       ▷ Queue of nodes to be processed.
 6:     mark[src] ← True
 7:     while nodes ≠ ∅ do
 8:         i ← nodes.dequeue()
 9:         for j ∈ G.neighbors(i) do
10:             p_success ← 1 − (1 − p_{i,j})^NoT[i]
11:             if mark[j] = False ∧ random() < p_success then
12:                 nodes.enqueue(j)
13:                 mark[j] ← True
14:                 reached ← reached + 1
15:             end if
16:         end for
17:     end while
18: end procedure
```

Fig. 7.4 Demonstration of the basic *Probabilistic Breadth-First Search* (PBFS) algorithm, which mimics flooding in a WMN.

request. Additionally, PBFS approximates over how many hops b is reached, providing a list of tuples with the probability for any path length b is reachable by the flooding process initiated by a. This list can be used to estimate the transmission time distribution functions $H_x(t)$ in the retry operation model as described in Section 7.2.4. For reasons of brevity, path length distributions have not been included in Figure 7.4. It should be mentioned that while throughout this work, only undirected graphs are considered, PBFS can be used just as well on directed graphs.

PBFS has several advantages over the existing analytical approaches. First, it is computationally less complex as the FPP [50] resorts to computing the joint probability distribution that a packet is received by a subset of the vertex cut in each step of its algorithm. PBFS instead has the complexity $O(|E|+|V|)$ of BFS [60], which allows to calculate reachability even for large graphs. PBFS also provides over how many hops a node is reached and with which probability. Furthermore, it does that not only for one node pair but for all reachable nodes from a specific NWB source node. PBFS can work with dynamic values of $p_{i,j}$. Provided a function that calculates $p_{i,j}$ over time or a series of ETX measurements, each run of PBFS could use different values for $p_{i,j}$ as input. Additionally, it is possible to modify PBFS to model modifications of basic flooding and other NWB approaches, as has been demonstrated in [11]. For example, PBFS allows for different configurations of transmissions done by each node (see code in Figure 7.4, Lines 3 and 10).

7.4 Multicast Reachability Validation

As PBFS relies on repeated randomized sampling runs, it reflects a Monte-Carlo method to approximate reachability. In a nutshell, more runs increase the accuracy of its approximation. An estimation of a sufficient sampling number for PBFS, its deviation from the exact theoretical reachability and its confidence follows in Sections 7.4.1 and 7.4.2.

7.4.1 Theoretical Validation

For theoretical validation, PBFS is evaluated in three exemplary small networks, where the complexity still allows a precise analytic assessment of reachability using Equation 7.7. These networks are depicted in Figure 7.5. All three network graphs are depicted within one Figure. The first network G_1 consists of nodes a, b, c and d and connecting links. The second G_2 adds nodes e and f to G_1, while the third network G_3 consists of the whole graph in Figure 7.5. Edge labels correspond to $p_{i,j}$.

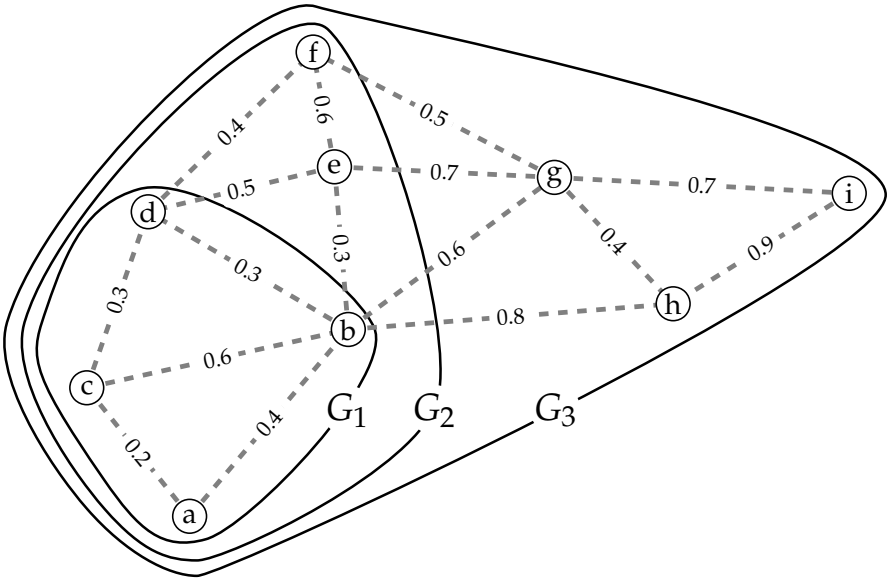


Fig. 7.5 Three example networks for theoretical validation: $\{a,b,c,d\}$, $\{a,b,c,d,e,f\}$ and $\{a,b,c,d,e,f,g,h,i\}$ including all links connecting the nodes. [11]

Evaluation results are listed in Table 7.1. It lists the values of three different reachabilities when calculated using Equation 7.7 and when approximated by PBFS

over a certain number of runs x. The run-time of the algorithms is included in parentheses as a reference. The results demonstrate two phenomena: First, the exponential increase in runtime when using Equation 7.7 is visible already for these simple networks. Since execution times are based on non-optimized implementations, they have to be taken with a grain of salt. But the effect of combinatorial explosion using Equation 7.7 is obvious. Second, one can see that PBFS converges relatively fast to the correct reachability value. Already after only 1000 runs, the relative error is around 5% and below 0.5% after 100000 runs with a confidence of 99%.

Table 7.1 PBFS deviation (with 99% confidence after 1000 trials) with different number of runs x, implementation runtime included in parentheses for illustration.

Network	Nodes / Edges	Reachability evaluated	Equation 7.7	PBFS $x_1 = 10^3$	PBFS $x_2 = 10^4$	PBFS $x_3 = 10^5$
G_1	4 / 5	$r_{a\to d}$	0.22824	±5.10%	±1.58%	±0.47%
			(0.004 s)	(0.020 s)	(0.199 s)	(1.990 s)
G_2	6 / 9	$r_{a\to f}$	0.19682	±4.71%	±1.45%	±0.48%
			(0.116 s)	(0.023 s)	(0.230 s)	(2.275 s)
G_3	9 / 16	$r_{a\to i}$	0.39104	±6.30%	±1.90%	±0.59%
			(31.004 s)	(0.032 s)	(0.315 s)	(3.179 s)

The standard deviation of a Monte Carlo method is determined as in Equation 7.8. Here, the value of x represents the number of iterations in PBFS while e is the absolute error.

$$\sigma = \sqrt{x} \cdot e \tag{7.8}$$

Given the characteristics of PBFS, we have to assume a constant standard deviation of the calculated reachabilities. This means that the absolute error decreases with an increasing number of runs while the standard deviation remains the same (see Equation 7.9).

$$\sigma = \sqrt{x_1} \cdot e_1 = \sqrt{x_2} \cdot e_2 = \cdots = \sqrt{x_n} \cdot e_n \tag{7.9}$$

So with an increasing number of runs we see the error decreasing by a factor as in Equation 7.10

$$\frac{e_n}{e_m} = \sqrt{\frac{x_m}{x_n}} \quad , m > n \tag{7.10}$$

In fact, this expected behavior is reflected by a linear decrease of the absolute error in the logarithmic plot in Figure 7.6. Together with the convergence of reachability it validates PBFS as a proper Monte Carlo method to approximate the behavior of NWBs.

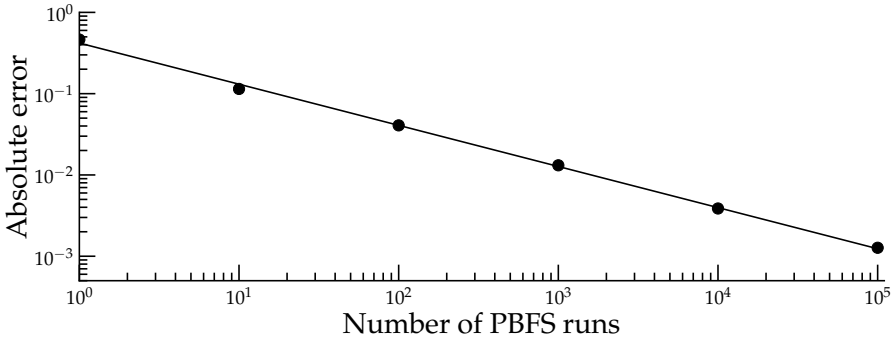


Fig. 7.6 Absolute PBFS error (y-axis) over number of runs (x-axis). Obtained by comparing PBFS results with Equation 7.7 for network G_3 in Figure 7.5. Axes have a logarithmic scale. [11]

7.4.2 Experimental Validation

To justify the feasibility of PBFS when reproducing properties of real world WMNs, this section provides an experimental validation. We compare the global and individual reachability approximations given by PBFS with results measured in simulations and a wireless testbed. We focus on basic flooding as NWB across different network topologies because it is the main method used to date and as such the basis of the multicast model in Section 7.2.3. Furthermore, hop length distributions for exemplary node pairs are shown and compared, which are needed for the calculation of transmission time distributions as in Section 7.2.4. PBFS estimations are expected to have a higher deviation, as it abstracts from short-term effects not reflected in the measured transition probabilities $p_{i,j}$.

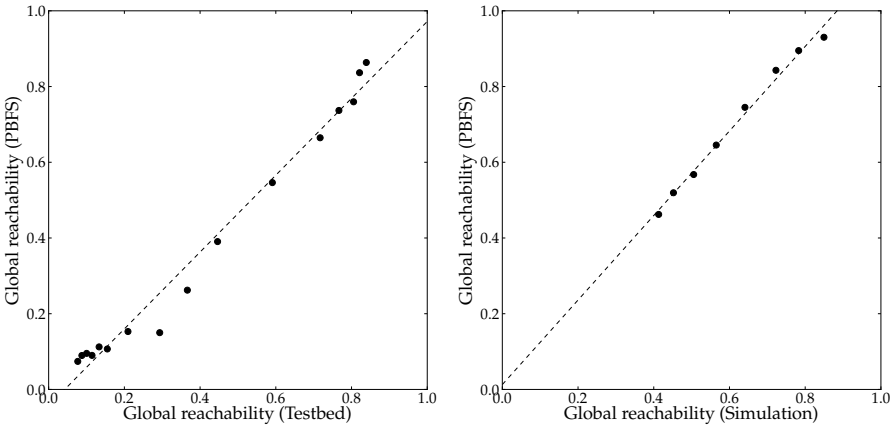
Experiment Setup

Flooding is implemented within the Click Modular Router framework [130], which allows to run it either in a wireless testbed or in simulators. For real-word experiments, the DES-Testbed at Freie Universität Berlin is used as it was the case already in Chapter 6. This testbed consists of around 130 wireless nodes equipped with IEEE 802.11 hardware that are spread over multiple campus buildings of FUB. We refer to [36] for a complete description of the testbed. As simulator, ns-3.14 [115, 13] with extensions for Click is employed. The wireless model is based on the Jakes propagation loss model [248], nodes form an NPART topology [160] consisting of 275 nodes. Different scenarios are obtained by varying the transmission power in the testbed as well as in simulations. Measuring transmission probabilities $p_{i,j}$, which are the input to PBFS, is done by a link prober that sends neighbor discovery packets and calculates the ETX [62]. Flooding packets are sent as link-layer broadcasts with a fixed 1 Mbps data rate and payload sizes of 400 bytes, the same size as the probing packets. In each scenario one node was chosen to start 1000 NWBs. The

achieved global and individual reachabilities are then evaluated and compared with the ones obtained by running PBFS using the same $p_{i,j}$ from the link prober.

Experiment Results

In Figure 7.7, the global reachability for flooding is shown, contrasting results obtained by PBFS with measurements from the testbed and simulation (Figures 7.7a and 7.7b, respectively). One data point corresponds to the reachability in a single scenario. Linear regression results in the dashed line and shows a very strong linear correlation with a correlation coefficient r greater than 0.99 and very small p-values in both comparisons. This indicates a strong presumption against the null hypothesis, meaning that there is no relationship between the two data sets. The complete results of the linear regression are listed in Table 7.2. PBFS slightly over-estimates reachability when compared to simulations, an effect that is not shown in testbed results. This bias in simulations hints at a systemic property in the simulation setup that needs further investigation.



(a) PBFS vs. Testbed (b) PBFS vs. Simulation

Fig. 7.7 Comparison of global reachability predicted by PBFS (y-axis) versus achieved in reality and by simulation (x-axis). A point constitutes the reachability for a single scenario [11].

Figure 7.8 shows the individual node reachabilities (top graphs) with their corresponding empirical cumulative distribution functions of the absolute difference between predicted and achieved reachabilities for all nodes (bottom graphs). Again, a data point reflects the reachability of a single node as estimated by PBFS versus the results from testbed or simulation. The shaded area in the top graphs constitutes an absolute error $\pm 10\%$ of the ideal line with slope 1. Linear regression with all data points results in the solid line.

Table 7.2 Values of the linear regression when correlating PBFS with experiment results. The regression line is of the form $Y = a + b \cdot X$.

Experiment	slope b	intercept a	r-value	p-value	standard error
PBFS vs. Testbed	1.015	-0.043	0.99	$2.038 \cdot 10^{-13}$	0.038
PBFS vs. Simulation	1.116	0.013	0.995	$2.578 \cdot 10^{-7}$	0.044

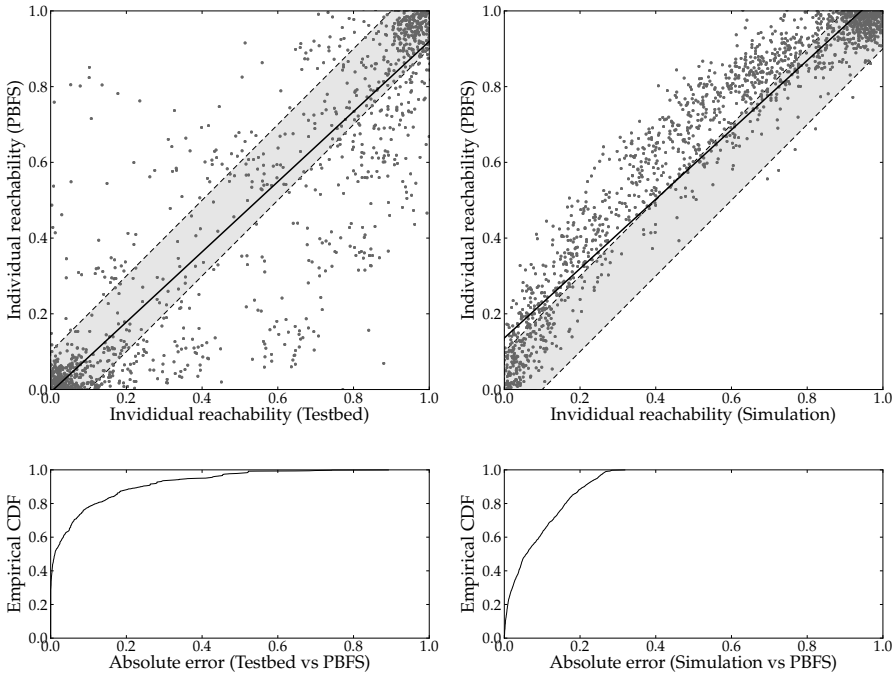
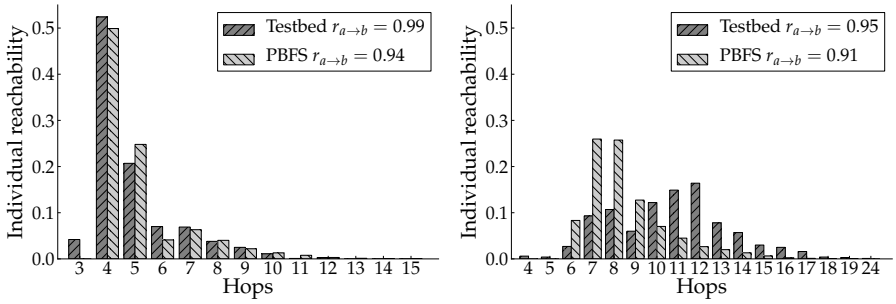


Fig. 7.8 Comparison of individual node reachabilities predicted by PBFS (y-axis) and achieved in reality or simulations (x-axis) across different scenarios (top). Corresponding CDF plots for the absolute differences shown in the bottom graphs. [11]

While there are outliers when compared with both simulation and testbed results, PBFS reasonably approximates the measured reachability. Simulation shows a much narrower distribution, although the points show a characteristic curve which again hints at some systematic property of the simulation model that warrants further research. Investigating this, however, is out of the scope of this work. In the bottom graphs, the corresponding CDF plots for the absolute differences are shown. The CDF plots show that in both setups, the PBFS approximation does not differ more than 0.2 from the measured reachability for 80% of all data points.

Last but not least, PBFS provides the distribution of path length probabilities for each destination node of a NWB as needed for the transmission time estimations in Section 7.2.4. A comparison of these distributions between PBFS and testbed

measurements is shown in Figure 7.9. In general, PBFS is able to estimate these distributions reasonably accurate as exemplary shown in Figure 7.9a. This is despite a 5% difference in reachability in this case. However, there are few nodes, especially at higher distance from the NWB source, that show misfits, usually with an overestimated path length as can be in Figure 7.9b.



(a) Node t9-155 (b) Node a6-028

Fig. 7.9 Exemplary path length distributions for NWBs to nodes t9-155 and a6-028. [11]

7.4.3 Discussion Validation

Validation shows that PBFS quickly converges to the theoretically correct value. Additionally, it approximates the behavior of real-life testbeds very well. Thus, it provides a valid method to approximate the input values for the lower level network models used in this work (see Section 7.2.3). Although not the scope of this work, it should be mentioned that PBFS generally promises to advance research on optimizing NWBs. The inevitable inaccuracies shown in the validation regarding global and individual reachabilities, but also regarding the distribution of multicast path lengths stem from basically two facts:

1. The calculation of link transmission probabilities has an intrinsic error based on its own accuracy and cannot capture short-term effects visible in experiment runs.
2. Transmission probabilities are not independent, especially during flooding (broadcast storm problem

Nonetheless, PBFS provides a valid approximation of reachability for NWBs and it does this with a high efficiency. This allows to use PBFS to estimate the stochastic behavior of NWBs when evaluating the responsiveness of SDPs, which rely heavily on NWBs when doing multicast communication. The results in the case study presented in Section 7.6 could not have achieved a comparable accuracy

without resorting to considerably more complex methods. In fact, the complexity of those models would have prohibited similar analysis.

7.5 Model Generation and Solution

After describing the various layers of the stochastic model hierarchy and the methods to measure or calculate their input parameters, it is now time to put the models into use. So, to calculate the SD responsiveness for given pairs of requester and provider in a network, the model layers described in Sections 7.2 need to be generated bottom-up using the following steps:

1. Define a scenario which consists of (1) the SD communication partners requester and provider, (2) the discovery protocol and (3) a deadline for the SD operation.
2. Generate low level network mapping models for individual requests and responses between the SD pair requester and provider based on the communication mechanisms of the protocol, uni- or multicast (see Sections 7.2.3, 7.2.4 and 7.3.4).
3. Integrate the network mapping models from Step 2 in the semi-Markov chain for the retry model (see Section 7.2.2). This chain calculates the responsiveness of an individual retry over time.
4. Calculate the number of retries n based on the defined protocol and deadline. This defines the structure of the high level discovery model (see Section 7.2.1).
5. Estimate the state transitions probabilities in the discovery model, using Equations 7.4 and 7.5. In these equations, $R_n(t)$ is the cumulative probability for absorption at time t in state *ok* in the retry model from Step 3.

The discovery model can then be solved. The steady-state probability of arrival in state *ok* in this model is the probability that an SD operation as specified in the scenario is successful, given the current monitored state of the network. The methodology has been implemented in a Python [191] framework that carries out all necessary steps. More complex stochastic analysis is performed using the SHARPE tool [224].

Monitoring

All monitoring data is gathered on demand from the routing layer. This approach is least invasive and can be used in every network where the routing layer provides the needed data. OLSR nodes use probe messages to measure link qualities for every neighbor. Given the forward delivery ratio d_f and reverse delivery ratio d_r, the *Expected Transmission Count* (ETX) [62] is defined as in Equation 7.11. This information allows to construct a complete network graph with edges weighted by their ETX value. In the graph, nodes are annotated with meta information from their local

OLSR routing table that includes the next hop for every other reachable node in the network.

$$\mathrm{ETX} = \frac{1}{(d_f \cdot d_r)} \tag{7.11}$$

7.6 Case Study

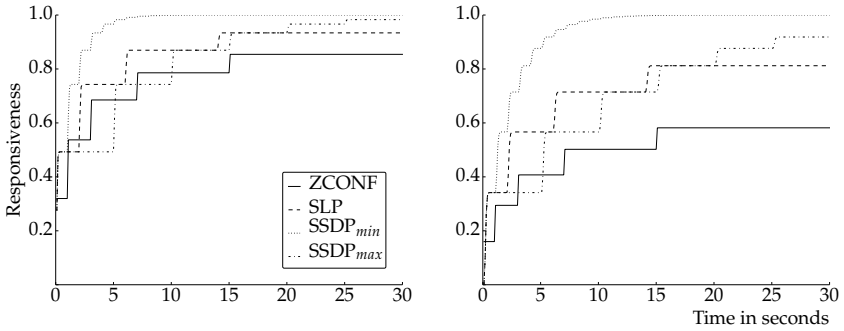
To demonstrate how the proposed methodology can be used to estimate the responsiveness in common use cases of SD, three of the protocols explained in Section 2.2.2 – *Zeroconf*, *Simple Service Discovery Protocol* SSDP and *Service Location Protocol* SLP – are now evaluated in three different scenarios using measured data from a real-life WMN, the *Distributed Embedded Systems* (DES) testbed at *Freie Universität Berlin* (FUB). For the sake of traceability, node identifiers in this text reflect the actual hostnames in the testbed. Due to space limitations, we refer to a complete description of the testbed in [36].

In this case study, OLSR [57] was used in version 0.6.5.2. It provides a valid reference for the real world application of the methodology. All monitoring was done by OLSR. Topology data was gathered with OLSR's *JavaScript Object Notation* (JSON) plug-in and then integrated into the network model using the Python framework. The testbed was configured and data gathered with different transmission power levels to obtain different topologies. Retry intervals of the SDPs are set according to the standards as described in Section 2.2.2 and Table 2.1. Since SSDP does not have fixed intervals, it is assessed in two different configurations reflecting the minimum and maximum interval as defined in the standard.

7.6.1 Scenario 1 – Single Pair Responsiveness

First, the responsiveness of a single pair requester and provider is evaluated over time. In order to investigate also how the responsiveness changes with the distance between nodes, two different pairs were chosen. One pair *(t9-105, t9-154)* is within the main cloud *t9*, a dense and well-connected part of the WMN consisting of 56 nodes (see Figure 7.10a). The other pair *(t9-105, a3-119)* covers almost the maximum distance in the network (see Figure 7.10b). In both cases, node *t9-105* is the requester. The results clearly show that as the distance between requester and provider increases, overall responsiveness decreases.

The difference in responsiveness among the protocols is apparent. With increasing deadlines the responsiveness of the *Zeroconf* protocol is consistently lower compared to the other protocols. This is because *Zeroconf* uses multicast for both requests and responses. Multicast packets will not be resent seven times before considered lost, so the danger of packet loss is much higher. The positive effects of mul-



(a) Within same cloud, provider *t9-154* (b) Maximum distance, provider *a3-119*

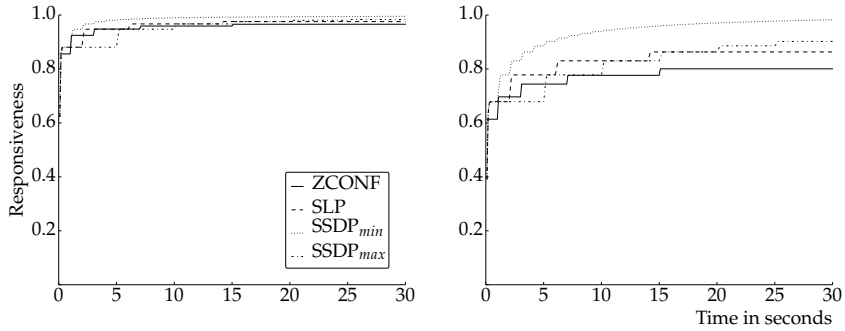
Fig. 7.10 Discovery responsiveness over time for different providers requested from *t9-105*. [4]

ticast responses for multiple communication pairs, as pointed out in Section 2.2.1, cannot be considered in this analysis. Also not included are the effects of additional load on the network caused by discovery. Since retries are considered independent events, lower retry intervals will lead to a higher responsiveness. While this assumption can be justified for retry intervals in the order of seconds – discovery packets are only a few bytes in size – it cannot hold for ever-lower intervals. So, although SSDP with a minimum interval ranks consistently best, the increased load might not be in the best interest of the service network as a whole. More in-depth research is needed on that matter. However, it can be deducted that with low deadlines, the chosen retry interval is more relevant for responsiveness than the communication mechanism (i.e., unicast vs. multicast). In general, current SDPs struggle to achieve a high responsiveness in WMNs, even over short distances.

7.6.2 Scenario 2 – Average Provider Responsiveness

The second scenario covers the average responsiveness of a single provider over time when requested from an arbitrary client in the network. To demonstrate how the models capture topology changes, this scenario uses data measured in two different topologies that were generated with different radio power settings. The focus lies on provider *t9-154* from Section 7.6.1, which is well centered within the network so it provides a good reference to see the effects of overall link quality on responsiveness. Figure 7.11 shows the results.

The main observation is that the average responsiveness when discovering node *t9-154* is quite high due to its prominent, almost optimal position in the network. With high quality radio links, depicted in Figure 7.11a, all protocols quickly reach a responsiveness of over 90%. Responsiveness is considerably decreased for lower quality wireless connections (see Figure 7.11b). With deadlines above 15 seconds,



(a) High radio power (b) Low radio power

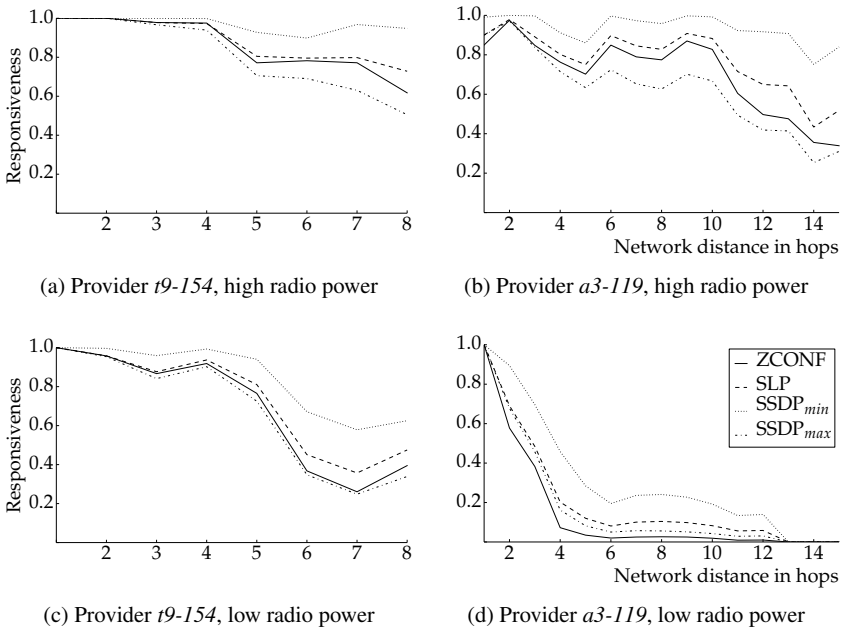
Fig. 7.11 Average responsiveness over time for provider *t9-154* in different topologies. [4]

there is a consistent ranking of the SDPs, with *Zeroconf* again having the lowest responsiveness. The ranking is identical for different link qualities, only the overall values are different. Due to different retry strategies of the protocols, however, this behavior is not consistent for lower deadlines. This underlines the findings from Scenario 1: With deadlines close to the individual retry intervals, the chosen interval is more relevant for responsiveness than the communication mechanism. In summary, it can be said that purely multicast based SD as in *Zeroconf* is justified when positive effects for multiple communication partners are expected. For single discovery operations among few partners, responding via unicast like in SSDP and SLP provides higher responsiveness because of its more reliable communication mechanism. Among SSDP and SLP, the specific retry strategy until the deadline is the main factor impacting responsiveness.

7.6.3 Scenario 3 – Expected Responsiveness Distance

The last scenario covers the *expected responsiveness distance* d_{er} from Definition 1.4. The responsiveness of two different providers, *t9-154* and *a3-119*, is calculated when requested from every client in the network. Then, the responsiveness is averaged for requesters at the same distance of these providers. Again, the used data was measured in two different topologies that were generated with different radio power settings. The discovery deadline is set to five seconds. Results are illustrated in Figure 7.12.

The ranking among the protocols is not the same as in the previous scenarios. This is due to the chosen, realistically short deadline of five seconds. The retry strategy until this deadline has an important impact and the maximum retry timeout for SSDP simply did not force enough retries to account for lost messages. It can also be recognized in Figure 7.12d, that badly placed providers risk a very low d_{er} with



(a) Provider *t9-154*, high radio power

(b) Provider *a3-119*, high radio power

(c) Provider *t9-154*, low radio power

(d) Provider *a3-119*, low radio power

Fig. 7.12 Average responsiveness over number of hops for providers *t9-154* and *a3-119* in two different topologies. [4]

decreasing link quality. The d_{er} for the different protocols with a required responsiveness $R_{req} = 0.8$ is summarized in Table 7.3. It should be noted that the maximum d_{er} depends on the eccentricity ε of the provider node, the greatest distance from any other node.

Table 7.3 Expected responsiveness distance d_{er} of the studied protocols with a deadline of five seconds (ε = provider eccentricity, R_{req} = required responsiveness, *RPS* = radio power setting). A higher d_{er} is generally desired.

Provider	ε	R_{req}	*RPS*	Zeroconf	SLP	SSDP (min)	SSDP (max)
t9-154	8	0.8	high	4	5	8	4
			low	4	5	5	4
a3-119	15	0.8	high	3	4	13	3
			low	1	1	2	1

As can be seen in Figure 7.12, the average responsiveness is not always decreasing over distance. This happens because hop count as the chosen distance metric does not necessarily reflect the quality of a path. In fact, longer paths might be of higher quality. The hop distance is, however, an intuitive metric that in this case

presents the lower bound for d_{er}. If needed, a more realistic, quality-based distance metric should be used to increase accuracy.

7.7 Conclusion

The case study on SD responsiveness as calculated by the presented models concludes this chapter. Results demonstrate that responsiveness varies dramatically depending on the position of nodes in the network and the overall link quality. The results further indicate that, with short deadlines close to the individual retry intervals, the right retry timing strategy is more important than the communication mechanism. With longer deadlines, using the more reliable unicast instead of multicast consistently improves responsiveness. In either case, the fixed strategies of current SD protocols struggle to achieve a high responsiveness in these dynamic and inherently unreliable networks. In Chapter 8, a comprehensive experimental validation of the models is presented.

Chapter 8
Correlating Model and Measurements

Abstract After demonstrating the *Service Discovery* (SD) responsiveness models in Chapter 7, the accuracy of the model estimations needs to be validated. This is done by comparing the model results to results measured in experiments. Several representative scenarios for active single SD are chosen with varying fault intensity. Corresponding to previous experiments, the first set of scenarios covers SD behavior with increasing load. In the second set the radio signal power on the nodes is decreased to reduce link quality. The empirical cumulative distribution functions of the SD response times in experiments, which reflect the responsiveness of SD are compared to the model outputs based on low level network measurements taken during the same period. We show that wherever direct SD packet measurements can be used, the model output has a very low error. When monitoring data from the routing layer is used, the error of the model has a consistently higher error depending on the quality of this input data.

8.1 Introduction

The focus of this chapter is the validation of the different layers of the model hierarchy which was introduced in Chapter 7. We use data gathered in experiments in the DES testbed, which was in parts already presented in Chapter 6, and correlate estimations given by the model with actual measured data in corresponding experiments. In experiments, three types of data are being recorded that are useful when correlating model results: Events, packets and topology (see also Sections 3.4 and 6.3.2).

Events The events reflect state changes in the *Service Discovery* (SD) process, such as “SD operation started” and “SD response received”. These time-stamped events allow to calculate the responsiveness of SD operations on the application layer, which should correlate to the steady-state probability of arrival in state *ok* in the discovery model presented in Section 7.2.1.

Packets Events are caused by packet transmissions on the network communication layer. A request initiated on the SD layer will cause a request packet to be sent, with eventual retries until a sufficient number of responses has been received. Each of the arriving responses will in turn cause an event on the SD layer. Since *ExCovery* adds unique identifiers to SD requests (see Section 3.7), each arriving response packet can be matched with the original request packet. The responsiveness of these pairs request and response can also be calculated. It corresponds to the cumulative probability for absorption in state *ok* in the retry model (see Section 7.2.2). As such, packet measurements can be used to validate the discovery model isolated from the lower models in the hierarchy.

Topology In each run, a full snapshot of the topology is recorded as reflected in the local views of each node. Every snapshot contains all known nodes, their neighbors and link qualities to these neigbors as estimated by the *Open Link-State Routing* (OLSR) layer. This allows to construct a directed graph of the network, with edges weighted by their transmission probability. This graph can be created for each node or over all nodes combined, creating the worst, average or best case.

Two different types of correlations are being done, both focusing on the discovery model which estimates the responsiveness of active SD operations. First, we use the topology data to solve all models bottom up as described in Section 7.5. In networks with proactive routing, snapshots of the current topology are usually readily available and leveraging this data is an attractive alternative to actively taking SD measurements. However, it needs to be checked whether the abstractions in the lower layer models combined with the uncertainty in input data correlate with the actual measured SD responsiveness. Second, we will compare model estimations using only packet measurements as input data to validate the discovery model isolated from the rest of model hierarchy. This can be seen as the baseline and reflects the case where no monitoring data is available to solve the lower level models but a history of past SD operations can be analyzed to predict the next SD operations. Both types of estimations, which will be referred to as *full model* and *discovery model* estimations, will be correlated to the actual measured responsiveness for the same period. We will compare responsiveness over time and calculate the error of the estimation in various conditions.

The rest of this chapter is structured as follows. In Section 8.2, the experiment setup and input data preparation for the models is explained. Section 8.3 shows and discusses the correlation of model and experiment results. Section 8.6 concludes the chapter.

8.2 Experiment Setup

Separate series of experiments were run for correlation since not all experiments presented in Chapter 6 were run with full topology recording enabled. The goal of this chapter is to correlate the active single discovery model from Chapter 7 so a

consistent database of measurements was generated for this purpose. We will show here the results for two pairs of requester and provider within the building *t9*. A description of the DES testbed can be found in Section 6.3.2. The first pair consists of nodes *t9-154* and *t9-k21a*, two nodes positioned at opposite ends of the building, as illustrated in Figure 8.1.

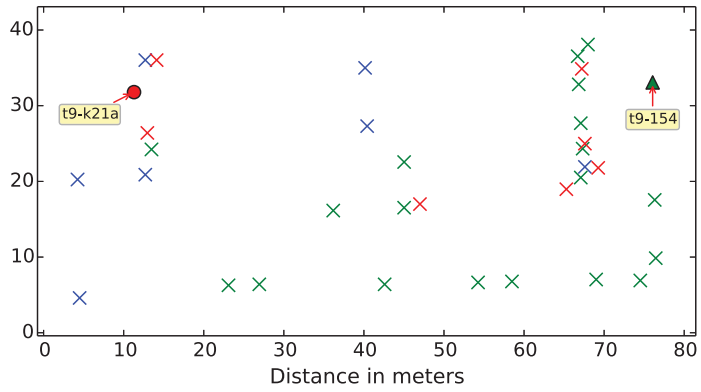


Fig. 8.1 Overview of building *t9* with nodes involved in experiments highlighted. Node colors define building floors.

The second pair consists of the two neighboring nodes *t9-154* and *t9-149*. This pair was chosen to show differences in the full model estimation between single and multi-hop communication, to abstract from effects of multicast propagation. The node positions are illustrated in Figure 8.2. A relatively long link is between the two nodes.

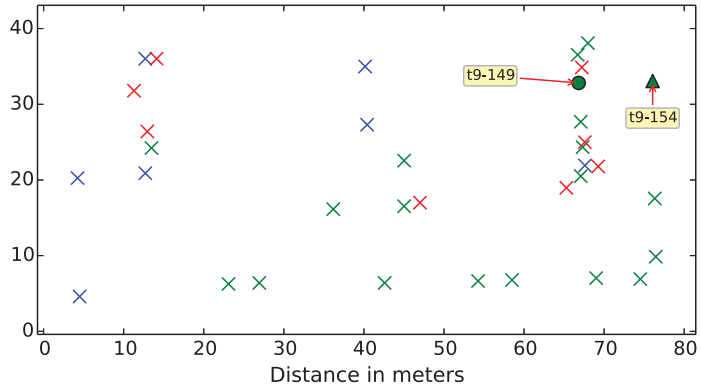


Fig. 8.2 Overview of building *t9* with neighboring nodes involved in experiments highlighted. Node colors define building floors.

For each pair, a set of scenarios was defined to compare model estimations and measurements with varying fault intensity. This was done because the abstractions in the models assume a simplified fault model, especially on the lower network layers. It will be shown how valid these abstractions are. Each scenario was measured in 1000 discovery operation, which proved to be a sound compromise between experiment runtime and ruling out temporal anomalies on the testbed. This number of runs translated to between 12 and 20 hours of experiment time. The individual scenarios are explained in the following sections.

In this Chapter, we will again focus on *Zeroconf* SD, which is purely multicast based and most sensitive to faults on the network. Multicast packets will not be retried on the lower network layers. Also, this puts the focus on the multicast estimations of the network mapping model (see Section 7.2.3). The unicast estimations basically transforms information as given by the OLSR routing layer to a semi-Markov model. While it remains interesting to show the quality of the responsiveness calculation based on our transformation, instead of proving the validity of the transformation itself, it would rather demonstrate the accuracy of the OLSR link quality data. As improving OLSR is not the target of this work, we will thus focus on the multicast-based *Zeroconf* SD.

8.2.1 Varying Data Rate Scenario

In the first scenario, just as shown in Chapter 6, in addition to the SD actors a random set of nodes was chosen to exchange traffic with varying data rates. We chose 44 nodes or 22 node pairs to make sure the traffic was reasonably distributed within the topology. Load on the network will cause collisions due to the shared wireless communication medium and the number of collisions will increase the higher the data rate becomes. This in turn will force retransmissions on the physical layer or lead to packet loss which will decrease the responsiveness. This scenario reflects a realistic use case, where a client tries to discover a single service, such as a backup file server, while other network nodes communicate at the same time.

Five different levels have been chosen for the 22 bidirectional UDP [186] stream between the load generating node pairs causing a combined data rate of 0.52, 0.70, 0.79, 0.88, and 1.05 megabits per second on the network. At present, load generation is started at the beginning of each run and stopped during the cleanup phase at the end of the run. This is because *ExCovery* does not yet support processes that continue over a number of runs. While this behavior is not problematic when considering the impact during the measurement phase, it will have an effect on the link quality as estimated by OLSR. Since OLSR measures this quality also during phases when no load is generated [62], the estimated link quality will be higher on average than the actual quality during measurements. This would be reflected in the SD responsiveness as calculated by the models.

8.2.2 Varying Radio Power

Although the first scenario provokes a realistic fault behavior, we decided to add a second scenario with the objective to manipulate link qualities in a consistent way. This way OLSR would correctly estimate the link qualities within its bounds and the input data to the stochastic models would be more accurate. Environment nodes this time only help with routing but do not produce additional traffic. Instead, the radio power of all nodes is varied to change the quality of the wireless links. The power gain values for the transmission antennas wer set to values of 15, 18, 20 and 25 dB. The single hop experiments between nodes *t9-154* and *t9-149* have only been carried out in this scenario as a consistent load impacting the single link could not be guaranteed in the first scenario. Specific power values needed to be chosen considering the characteristics of this link. We measured that 11 and 15 dB provided a good range in which the quality changed without dropping the link.

8.2.3 Input Data Preparation

The stochastic models work purely on the basis of network data. When the full model hierarchy is calculated bottom up, OLSR link quality data is used. If only the discovery model is calculated, it uses SD packet based response time distributions as input. However, the event based responsiveness on the application layer contains also the overhead for queuing and computing on the SD actor nodes themselves.

To solve this issue, the overhead was calculated by taking the time differences of the empirical *Cumulative Distribution Functions* (CDF) of packet and event based measurements. Event based measurements were only considered before the first retry, to make sure that in both cases, only a single request and a single response were been sent. Then, the average time difference was calculated in each scenario. Table 8.1 shows the result for the different experiment scenarios. As a reference, also values for the experiments presented in Chapter 6 and experiments for which no results have been shown in this thesis are included.

As can be seen in Table 8.1 the offset is not constant but increases with higher load or lower link quality. Two factors are mainly responsible for this behavior. First, the actor node has additional processing overhead when the load generation uses a higher data rate. These packets need to be routed and also the actor node helps with that. Second, the comparison between the empirical CDF packets and events is not perfect. The discrete points in time do not match completely between the two distributions so an approximation is necessary. Given that there are less data points in the event based CDF, as more SD operations need to be retried the higher the load and the lower the link quality, this approximation tends to overestimate the offset. However, these offsets provide a reasonable bound with which the model calculations can be shifted in time. This can be seen in Figure 8.3, which shows a comparison of measured responsiveness and as calculated using the models. The

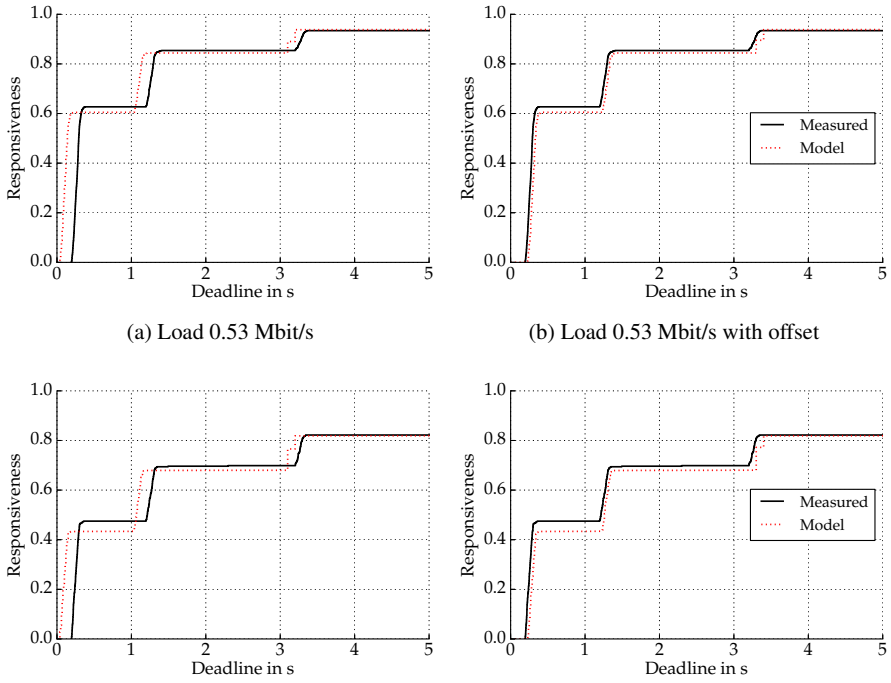
Table 8.1 Offset between packet and event CDF due to timestamp difference. The offset reflects the processing overhead on the SD requester node.

	Experiment	Offset (ms)
Load (Mbit/s)	0.53	184.9538
	0.70	210.0539
	0.79	293.4857
	0.88	337.7214
	1.05	379.0667
Power (dB)	15	201.1710
	18	187.4545
	20	179.8312
	25	175.7707
One hop (dB)	11	175.1428
	15	171.6270
t9-105 (# VoIP Streams)	0	166.1595
	5	170.6339
	10	169.9149
	15	175.5955
	20	171.6539
	25	184.9500
	35	188.9077
	40	210.0517
	50	217.9131
a3-119 (# VoIP Streams)	0	177.2267
	5	178.8163
	10	183.2445
	15	186.2169
	20	190.4982
	40	295.1937
	50	331.5833
	Max	**295.1938**
	Min	**166.1596**
	Avg	**193.6029**

graphs on the left side show the comparison without calculating the offset. On the right side the model results are drawn including the correction.

8.3 Correlation of Model and Experiments

We will now show the results of the correlation, first for the experiments in varying load conditions, then with varying signal strength. The responsiveness as estimated by the models is correlated with the measurements for a corresponding data set. For



(a) Load 0.53 Mbit/s

(b) Load 0.53 Mbit/s with offset

(c) Power 18dB

(d) Power 18dB with offset

Fig. 8.3 Comparison between responsiveness without correction of event and packet based measurement offsets. On the right side the corrected results.

both sets of experiments, the correlation is done visually, showing both responsiveness curves in the same graphs, and using statistical correlation metrics. We will use well known metrics, such as the maximum and the average of the absolute error e, the *Root Mean Squared Error* (RMSE), the variance of the error σ_e and the Pearson product-moment correlation coefficient r. These metrics a widely used to quantify the correlation of given data sets. However, as is often the case with single number metrics, none of them is able to draw a complete picture in all scenarios. This is the reason why all metrics are calculated in all scenarios, together with a visual correlation. For a comprehensive explanation and interpretation of these metrics we refer to [237, 109]. The RMSE is basically the square root of the average squared error of the model outputs and is calculated as in Equation 8.1. The functions CDF_{exp} and CDF_{mod} calculate the responsiveness from experiments and from the models, respectively. Since CDF_{exp} is an empirical distribution, the values of i represent times at which there actually exists a value for $CDF_{exp}(i)$.

$$RMSE = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} \left(CDF_{exp}(i) - CDF_{mod}(i)\right)^2} \tag{8.1}$$

The Pearson coefficient is calculated as in Equation 8.2. The values of r will always lie between -1 and 1 and can be interpreted as the quality of the linear correlation between two data sets. In Equation 8.2, $(CDF_{exp}(i) - avg(CDF_{exp}))(CDF_{mod}(i) - avg(CDF_{mod}))$ is positive exactly when $CDF_{exp}(i)$ and $CDF_{mod}(i)$ are both higher or both lower than their respective means and in turn, r will be positive. In the case of $r = 1$, this means that a linear equation perfectly describes the relationship between CDF_{exp} and CDF_{mod}. So whenever CDF_{exp} increases, also CDF_{mod} increases by a similar value. A value of $r = 0$ says there is no linear relationship between the two. More information and a thorough discussion of the Pearson coefficient can be found in [199, 76]. In our case, with two monotonously increasing functions we strive for a coefficient as close to $r = 1$ as possible. The p-value is of less significance, which corresponds to the probability that random sampling would result in the same r if there were in fact no correlation between CDF_{exp} and CDF_{mod}. The p-value was zero within the precision of the calculation in all scenarios.

$$r = \frac{\sum_{i=1}^{n} (CDF_{exp}(i) - avg(CDF_{exp}))(CDF_{mod}(i) - avg(CDF_{mod}))}{\sqrt{\sum_{i=1}^{n} (CDF_{exp}(i) - avg(CDF_{exp})^2} \sqrt{\sum_{i=1}^{n} (CDF_{mod}(i) - avg(CDF_{mod})^2}} \tag{8.2}$$

8.4 Experiments with Variable Load

The first set of experiments was done while increasing the load of environment nodes in the network. Related to Figure 6.7, we can see the results of all three series in Figure 8.4.

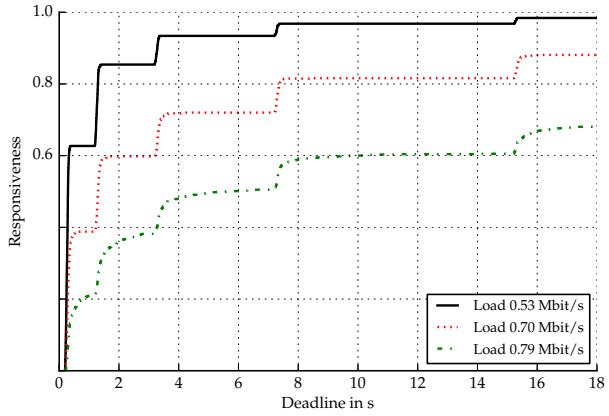


Fig. 8.4 Experimental CDF for load experiments using data rates of 0.53, 0.70 and 0.79 Mbit/s.

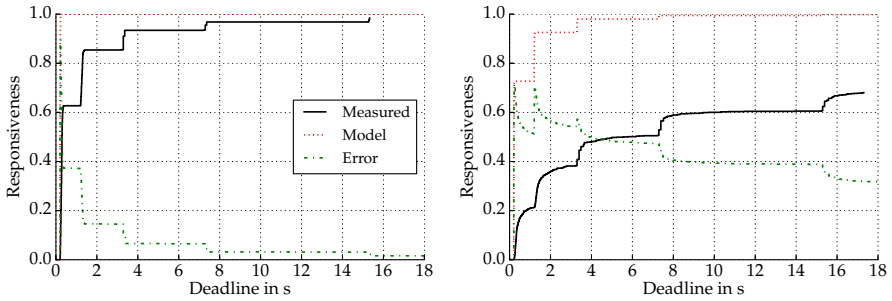
It can be seen that the responsiveness generally decreases with increasing load. More retries are needed at higher loads to achieve similar responsiveness. It can additionally be seen that the steps in the CDF after doing retries are less pronounced, as the additional packets provoke contention on links and nodes and the delay of packets gets higher. More detailed information can be found in Chapter 6.

8.4.1 Full Model Hierarchy

The results for the correlation of experiment and full model are shown in Figure 8.5. Here, the experiment results reflect the black line corresponding to the line for the same load in Figure 8.4. The dotted red line is the estimation of the full model hierarchy calculated bottom up. The dashed green line reflects the absolute error.

The model results do not correlate very well with the measured results. A decreasing error is to be expected when comparing two cumulative distributions, in fact, both of them will converge to a responsiveness of 1. However, the error is very high with a RMSE of between 0.21 and 0.47 depending on the load and a low Pearson coefficient r. The value of r is actually higher with increasing load because the model estimations only then start to show the characteristic step function but r remains very low with 0.86. The Pearson coefficient needs to be very close to 1 to reflect a strong positive correlation between two data sets.

One can see that with low load as in Figure 8.5a, the models basically predict a near perfect responsiveness already after the first request. The reason for this behavior is the input data to the network mapping model (see Section 7.2.3), which calculates transmission times and delays of SD multicast messages based on the link quality as given by OLSR. It turns out that according to OLSR, the links are almost perfect. As mentioned earlier, OLSR uses probe messages to monitor the quality of links and these link qualities continue to be sent, even when no load is being generated between experiment runs. A solution would be to run load generation as a continuous process outside the abstract experiment description that *ExCovery* reads. However, since the load generating pairs are being switched randomly each run, this external process would need to be synchronized with experiment execution. Alternatively, *ExCovery* could be extended to support such global processes. As a short term remedy, we will show results of the full model hierarchy when we increase the fault intensity by decreasing the wireless signal quality on the nodes. This is constant for the duration of an experiment series and allows OLSR to have reasonably static conditions. It remains to be noted that the full model hierarchy depends directly on the quality of the input data and results inherit the uncertainties in this data. Figure 8.5 is an impressive illustration of this argument. As a reference, Table 8.2 shows the correlation scores for the full model hierarchy in this scenario.



(a) Load 0.53 Mbit/s (b) Load 0.79 Mbit/s

Fig. 8.5 Comparison between measured Responsiveness and results of the full model hierarchy for different loads. The green line denotes the absolute error between the two distributions.

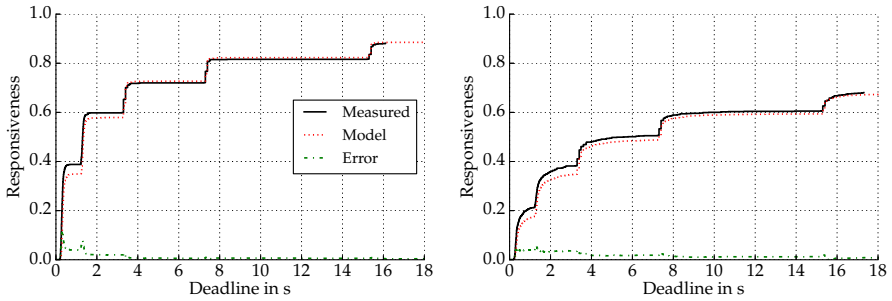
Table 8.2 Table of errors and correlation metrics for full model hierarchy in load scenario.

Experiment	$max(e)$	$avg(e)$	$RMSE$	σ_e	r
0.53 Mbit/s	0.8952	0.1374	0.2106	0.0936	0.3053
0.70 Mbit/s	0.8856	0.3068	0.3573	0.1363	0.5762
0.79 Mbit/s	0.7147	0.4563	0.4715	0.1101	0.8623

8.4.2 Discovery Model

In this section, we will show the correlation between responsiveness measurements and model estimations based on recorded packet transmission times. As has been explained earlier, the highest model in the model hierarchy, the discovery model uses as input the cumulative distribution of response times for individual pairs request and response. This distribution is used to calculate the transition probabilities as in Equations 7.4 and 7.5. When the full model hierarchy is calculated bottom up, this distribution is calculated using the retry model (see Section 7.2.2). Instead, packet based measurements can be employed, converted into such distributions as shown in Figure 6.8. This will be done in this section as well. It provides a way to show the correlation of the discovery model isolated from the lower parts of the model hierarchy.

The results are illustrated in Figure 8.6 where exemplary curves for data rates of 0.53 and 0.79 Mbit/s are shown. In Figure 8.6, the black line again is the actual measured responsiveness, the red dotted line is the model estimation and the green line the absolute error. Two observations can be made. First, the correlation of the estimations is almost perfect with a Pearson coefficient higher than 0.99 in all experiments. The RMSE is 0.03 and lower. On average, the estimated responsiveness differs by only 0.02 within the observed interval until the deadline at 18 seconds. The maximum error is higher, but occurs only right at the beginning of the interval. This hints at optimization potential when calculating the computing overhead on the



(a) Load 0.53 Mbit/s (b) Load 0.79 Mbit/s

Fig. 8.6 Comparison between measured Responsiveness and discovery model results for different loads. The green line denotes the absolute error between the two distributions.

Table 8.3 Table of errors and correlation metrics for discovery model in load scenario.

Experiment	$max(e)$	$avg(e)$	$RMSE$	σ_e	r
0.53 Mbit/s	0.2079	0.0153	0.0300	0.0043	0.9949
0.70 Mbit/s	0.1144	0.0189	0.0288	0.0018	0.9975
0.79 Mbit/s	0.0544	0.0198	0.0248	0.0007	0.9988

node. However, the two distributions correlate very well and this validates the discovery model and especially its calculation of the transition probabilities for such a scenario. Table 8.3 sums up the values of the different error and correlation metrics.

8.5 Experiments with Variable Radio Power

In the second set of experiments, in each series a different radio signal power was used on each of the mesh nodes. This changed the range of the wireless signal and in turn made it more vulnerable to interference and other types of faults. The results of the three series for antenna gains of 15, 18 and 20 dB are shown in Figure 8.7.

It can be seen when comparing Figure 8.7 and 8.4 that the overall responsiveness of both sets is similar. In fact, the antenna gain was chosen to resemble similar conditions for packet loss. However, contrary to the results from the load experiments in Section 8.4 the slope of the CDF is a lot less smooth and the steps in the distribution due to the retries are considerably more pronounced. The reason for this is that although the overall signal quality is lower and produces a comparable packet loss on the paths between requester and provider, the nodes do have to deal with less traffic as no background load is generated. Additionally, collisions happen only due to the multicast flooding of SD packets or because of external interference. Both causes lead to a lower delay in the network, hence, sharper steps in the CDF.

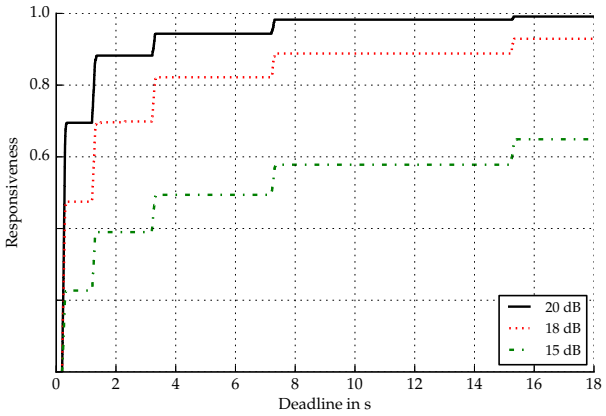


Fig. 8.7 Experimental CDF for power Experiments using antenna gains of 15, 18 and 20 dB.

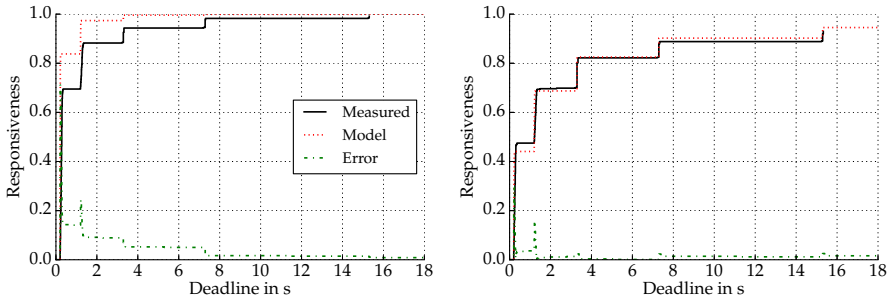
8.5.1 Full Model Hierarchy

The results for the correlation of experiment and results of the full model hierarchy are shown in Figure 8.8. The experiment results reflect the black line corresponding to the line for the same load in Figure 8.4. The dotted red line are the estimations of the model calculated bottom up. The dashed green line reflects the absolute error. Two exemplary graphs for antenna gains of 20 and 18 dB are shown.

As opposed to the results in Section 8.4.1, the two curves correlate considerably better. Especially with lower signal quality, the OLSR monitoring data seems to better pick up the actual transmission quality of the network and thus, the estimation of the full model hierarchy is consistently more accurate. With high signal quality the models overestimate responsiveness. What exactly causes this behavior cannot yet be fully explained. The behavior hints at the smaller OLSR probe packets being less likely to be dropped than SD packets over higher quality links. More thorough investigation is needed here. The RMSE is between 0.04 and 0.18 and the Pearson coefficient is above 0.98 in two out of three series. The correlation in all three scenarios is promising, given that no additional measurements are needed to produce such results but a current snapshot of the network is sufficient. All errors and correlation metrics are summarized in Table 8.4.

Table 8.4 Table of errors and correlation metrics for full model hierarchy in power scenario.

Experiment	$max(e)$	$avg(e)$	$RMSE$	σ_e	r
20 dB	0.7110	0.0696	0.1107	0.0443	0.8817
18 dB	0.2955	0.0216	0.0373	0.0071	0.9822
15 dB	0.2786	0.1767	0.1827	0.0148	0.9913



(a) Gain 20 dB (b) Gain 18 dB

Fig. 8.8 Comparison between measured responsiveness and results of the full model hierarchy with different antenna gain. The green line denotes the absolute error between the two distributions.

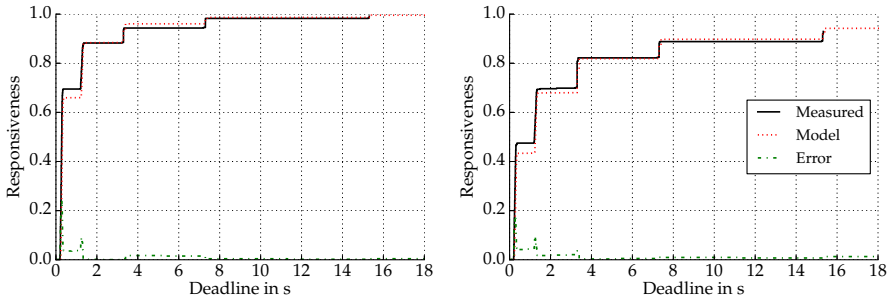
8.5.2 Discovery Model

Finally, we will show the correlation between responsiveness measurements with different antenna gain and model estimations based on recorded packet transmission times. Results are illustrated in Figure 8.9. The observations from Section 8.4.2 are valid here as well. The correlation of the estimations is again almost perfect with $r > 0.99$ in all experiments. The RMSE is 0.035 and lower. On average, the estimated responsiveness different by only 0.02 within the observed interval until the deadline at 18 seconds. Also the maximum error is comparable to the load experiments so the discovery model works equally well in both presented scenarios. This is strong evidence for the validity of the discovery model.

Another observation can be made in both the full model and discovery model estimations. The retry steps in the CDF are steeper due to lower packet delays and the maximum error shows spikes right after these retries. It shows that estimating the exact arrival times is non-trivial and the models struggle to capture it properly. However, even in those worst cases, the maximum error of the discovery model is still between 0.24 and 0.09. In the latter case that means that predicting the responsiveness at any given time will be at maximum 10% off and on average only 2% in all cases. Table 8.5 presents a summary of the obtained errors and correlation metrics.

Table 8.5 Table of errors and correlation metrics for discovery model in power scenario.

Experiment	$max(e)$	$avg(e)$	$RMSE$	σ_e	r
20 dB	0.2426	0.0163	0.0353	0.0059	0.9922
18 dB	0.1687	0.0220	0.0334	0.0036	0.9957
15 dB	0.0905	0.0221	0.0295	0.0013	0.9964



(a) Gain 20 dB (b) Gain 18 dB

Fig. 8.9 Comparison between measured responsiveness and results of the discovery model with different antenna gain. The green line denotes the absolute error between the two distributions.

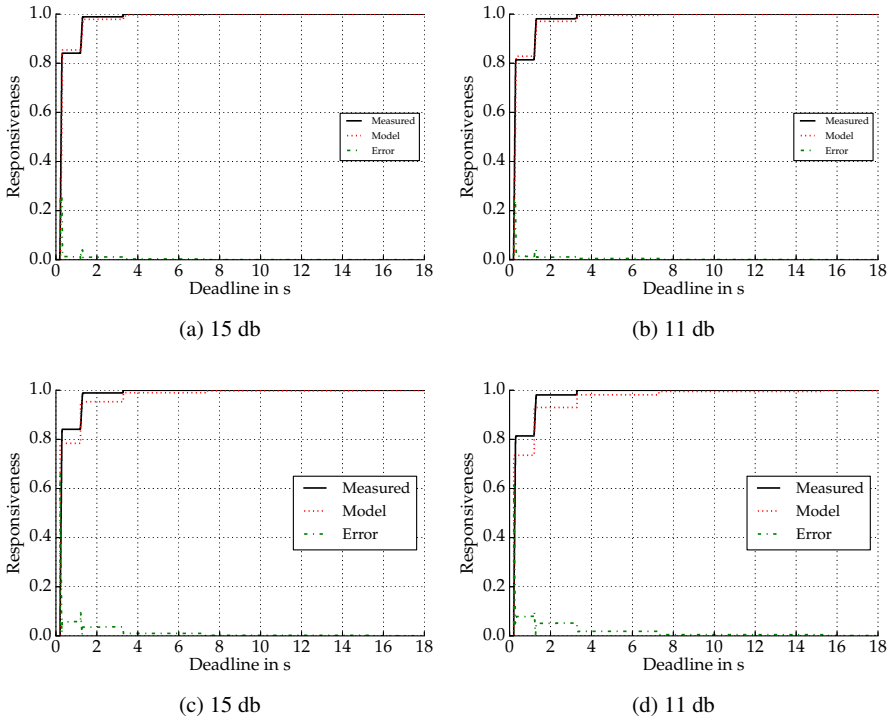
8.5.3 One Hop Experiments

A separate set of experiments was run with only two neighboring nodes, to isolate the SD operation from the rest of the network. The antenna gain was chosen to be at 15 and 11 dB. In fact, it was rather difficult to choose correct values as the operating system allows to set only integer values. This coarse grained resolution led to either perfect or non existent links if chosen too high or two low. At the same time, the gain needed to be chosen such that one value provided consistently better quality then the other. Although both created a reasonably reliable link, 15 and 11 dB fulfilled these requirements. Results are illustrated in Figure 8.10.

It can again be seen, that both the discovery model and the full model hierarchy correlate well with the measured responsiveness. As before, the discovery model fares consistently better as it relies on concrete SD packet measurements instead of abstractions on the lower layers. For reference, Table 8.6 contains the numerical results for this scenario.

Table 8.6 Table of errors and correlation metrics for one hop experiments in the power scenario.

Experiment	model used	$max(e)$	$avg(e)$	$RMSE$	σ_e	r
15 dB	full model	0.6679	0.0267	0.0621	0.0293	0.8229
	discovery model	0.2589	0.0102	0.0349	0.0080	0.9888
11 dB	full model	0.6142	0.0348	0.0646	0.0242	0.8511
	discovery model	0.2417	0.0105	0.0327	0.0068	0.9902



(a) 15 db (b) 11 db

(c) 15 db (d) 11 db

Fig. 8.10 Comparison between measured responsiveness and results of the full model hierarchy for different values of signal strength.

8.6 Conclusion

We correlated model estimations for responsiveness with the actual measured responsiveness during the same time interval. Two different estimations were calculated. The first uses the full model hierarchy bottom up as presented in Section 7.5. It uses input data about the links and their quality from the routing layer and uses it to approximate the distribution of transmission times and probabilities over the paths between the SD actors. This is done with the help of *Probabilistic Breadth-First Search* (PBFS), which is described in Section 7.3. The second estimation is done by solving only the discovery model using measurements of previous SD packets.

Both types of estimations may be feasible, depending on the application scenario. Using the full model hierarchy is advised if the network provides such low level data. Generally, this is true for wireless mesh networks with proactive routing. At the cost of computing resources for solving all models bottom up, network bandwidth is saved. Also, this estimation can be performed upon entering a network, no history of measurements is needed. The second approach should be used if no low level network measurements are available. In this case, a history of SD operations should

be kept to be used as input data to the models. The more frequent SD operations are and the less dynamic the network is, this estimation will be more precise.

The estimations of the full model hierarchy depend highly on the quality of the input data. Due to the experiment setup, the generated load did not have the same impact on the routing layer monitoring packets than it had on the SD packets. This uncertainty is inherited by the models which in these cases led to an overestimation of responsiveness. As soon as the low level monitoring data more closely reflects the quality of the SD communication, the model hierarchy estimations correlate better with the actual measurements. In these cases, the *Root Mean Squared Error* (RMSE) lies between 0.18 and 0.03 and the Pearson coefficient r is above 0.99 in two out of three cases while being 0.88 in the third. Still, more research is needed to improve the quality of the input data. This in turn will automatically improve the quality of the models.

The correlation of the discovery model is exceptionally good. In fact, it correlates nearly perfectly in all considered scenarios. The maximum error occurs only right after the requests and subsequent retries. This is due to the computing overhead on the client node being estimated outside the discovery model. A more precise estimation will reduce this error. Still, the absolute error is between 0.05 and 0.24 in all scenarios. The RMSE is very low between 0.025 and 0.035 and r is well above 0.99 in all scenarios. This validates the discovery model as a representation of the active single SD process.

All error and correlation metrics are summarized in Table 8.7.

Table 8.7 Table of errors and correlation metrics in all scenarios.

Scenario	Experiment	model used	$max(e)$	$avg(e)$	$RMSE$	σ_e	r
Load	0.53 Mbit/s	full model	0.8952	0.1374	0.2106	0.0936	0.3053
		discovery model	0.2079	0.0153	0.0300	0.0043	0.9949
	0.70 Mbit/s	full model	0.8856	0.3068	0.3573	0.1363	0.5762
		discovery model	0.1144	0.0189	0.0288	0.0018	0.9975
	0.79 Mbit/s	full model	0.7147	0.4563	0.4715	0.1101	0.8623
		discovery model	0.0544	0.0198	0.0248	0.0007	0.9988
Power	15 dB	full model	0.2786	0.1767	0.1827	0.0148	0.9913
		discovery model	0.0905	0.0221	0.0295	0.0013	0.9964
	18 dB	full model	0.2955	0.0216	0.0373	0.0071	0.9822
		discovery model	0.1687	0.0220	0.0334	0.0036	0.9957
	20 dB	full model	0.7110	0.0696	0.1107	0.0443	0.8817
		discovery model	0.2426	0.0163	0.0353	0.0059	0.9922
Power 1-Hop	11 dB	full model	0.6142	0.0348	0.0646	0.0242	0.8511
		discovery model	0.2417	0.0105	0.0327	0.0068	0.9902
	15 dB	full model	0.6679	0.0267	0.0621	0.0293	0.8229
		discovery model	0.2589	0.0102	0.0349	0.0080	0.9888

Part IV
Conclusions

Chapter 9
Conclusions and Outlook

Abstract The results of the two main parts of this work are summarized and put into context. We first cover user-perceived service availability evaluation as presented in Part II. We then comment on the research on service discovery responsiveness in Part III and give pointers to future research.

9.1 User-Perceived Dependability

Services play an increasingly important role in modern networks, ranging from web service provision of global businesses to the Internet of Things. Dependability of service provision is thus an important goal but the assessment of specific dependability properties remains challenging. Ever more often, a system-view of dependability does not properly reflect the variable distribution of dependability within dynamic networks and is thus only of statistical relevance. Since providers and clients are part of a connecting *Information and Communications Technology* (ICT) infrastructure, service dependability varies with the position of actors. The ICT devices needed for service provision change for each configuration. Service dependability models need to incorporate these user-perceived perspectives. We present two approaches to quantify user-perceived service dependability. Both approaches demonstrate that the dependability of service provision indeed differs considerably depending on the position of service clients and providers, even in highly reliable wired networks. The following sections hold concluding remarks for each problem area that has been covered in the preceeding parts of this work.

We first focus on service availability. Using input models of the service, the infrastructure and a mapping between the two to describe actors of service communication, availability models are automatically created that calculate user-perceived instantaneous availability. The feasibility of the approach is demonstrated using exemplary services in the network of University of Lugano, Switzerland. The contributions of this approach as presented in Part II are being summarized in Section 9.2.

The second and main part of this thesis aims at the responsiveness of *Service Discovery* (SD), the probability to find service instances within a deadline even in the presence of faults. For successful provision, a service first has to be discovered by a client so successful SD is a precondition for successful service usage. We present a hierarchy of stochastic models to calculate user-perceived responsiveness of SD based on monitoring data from the routing layer. Extensive series of experiments have been run in the Distributed Embedded Systems (DES) wireless testbed at Freie Universität Berlin to assess responsiveness of current *Service Discovery Protocols* (SDPs) and to validate the presented models. The contributions of Part III can be found in in Section 9.3.

9.2 Service Availability Evaluation

A methodology is provided in Part II to facilitate the evaluation of user-perceived service dependability, that is, the dependability valid for a specific pairs service requester and provider. The approach is based on the work by Milanovic et al. [157] but we focus explicitly on instantaneous service availability and provide an automated, fully model-driven methodology that evaluates user-perceived service availability. In addition, the same methodology could be used to evaluate other dependability properties, given that different attributes necessary for evaluation of such properties are attributed to the ICT component model. All described features are achieved by transforming a set of input models that describe:

1. The ICT infrastructure, including non-functional properties related to the analysis of interest for every entity of the infrastructure.
2. An abstract service as a composition of atomic services, which are indivisible with respect to their functionality.
3. A mapping of atomic services to ICT components that enable these services, hence, the actors of service communication.

For the input models, the *Unified Modeling Language* (UML) was adopted for infrastructure and service descriptions as it is standardized and widely used, especially for design purposes. For the mapping model, the *Extensible Markup Language* (XML) was chosen due to its versatility. Visualization of the models has been an important factor as well and these decisions guarantee that the models remain human-readable. The separation into individual models and distinct steps for generating and using the models allows the methodology to be well suited for dynamic environments. Changes to intrinsic properties of network devices, such as failure rate, redundant components, manufacturer and others, can be performed directly in the device class description and so reflect to all objects in the ICT infrastructure model. That way, the methodology provides a straight-forward way to enable different types of user-perceived dependability analysis based on a *User-Perceived Service Infrastructure Model* (UPSIM), depending on the properties for each class of components. The methodology uses a hierarchical service model with a compos-

ite service on the highest level, which is composed of atomic services. The atomic services in turn map to ICT infrastructure components. With only minor changes to this mapping model, all properties can be evaluated for different requesters and providers, reflecting different user-perceived views.

Parts of the results of the research on service availability presented in this work have been published in [2, 3, 5, 10, 12].

9.2.1 Output Models

The presented methodology provides an in major parts automated generation of different output models for specified user-perceived views, such as a UPSIM for subsequent analysis, or *Reliability Block Diagrams* (RBDs) and *Fault Trees* (FTs) for availability evaluation. The methodology uses specific instances of the previously described input models, which define the current state of the network, the service under analysis and the user-perceived scope, and finds all ICT components relevant for service provision of this pair, preserving their context within the network. It then generates a UPSIM that can be used in subsequent user-perceived dependability analysis. We exemplify a transformation into a RBD, which is solved to obtain the steady-state availability of the given service. Given deployment times of the ICT components in the infrastructure model and usage durations in the service model, one main contribution of this methodology is a transformation into a RBD to assess instantaneous service availability at a specific point in time. The approach defines a sound concept of service and component time to allow such analysis: In addition to the usage durations the exact component access times during provision are included in the service model and taken into account for evaluation, which results in a more accurate estimation, especially for longer service execution times. Instead of an RBD, the methodology also supports and equivalent transformation to a FT. These output models for a client-specific infrastructure providing a composite service constitute a *User-Perceived Service Availability Model* (UPSAM).

9.2.2 Case Study

To demonstrate the proposed methodology, we apply it to parts of the service network infrastructure at *University of Lugano* (USI), Switzerland. We show how to generate the UPSIM for an exemplary email service and demonstrate how such a UPSIM could be used to evaluate different user-perceived properties of the printing service for given service requesters and providers. As a side effect, with the UPSIM the methodology provides a practical way to automatically identify and visualize dependability-relevant ICT components, to give a quick overview on where the service problem might be caused. Using the generated UPSAM, we show how the steady-state availability of the same service can differ considerably even in such

a high-availability network when requested from two different users. The methodology is thus able to provide a fine-grained view on service availability as experienced from different points of the network. By using an UPSAM for instantaneous availability evaluation, not only different views have been compared but also the impact on user-perceived service availability over time when modifying network components. As an example, the impact of redundant atomic service providers on the composite service availability was calculated. Additionally, individual ICT components were exchanged to demonstrate the effect of component age on availability. The results show availability as a dynamic property and indicate how the methodology can improve network design by estimating the impact of new component deployment. After deployment, they can be used to detect bottlenecks in the network or to evaluate the impact of planned changes to the ICT infrastructure.

9.2.3 Outlook

Although the demonstration network is based only on parts of the service network at USI, the proposed methodology is scalable and applicable to complex, dynamic networks as well. We showed how different types of dynamics affect only specific models so that in most scenarios, the majority of models remains unchanged. Also, the methodology implementation is automated whenever no human input or decision is necessary and it is thus possible to quickly regenerate the output models in case of dynamic changes to the network or its services. More research is needed to demonstrate the applicability of the methodology to complex infrastructures such as cloud computing, which do not provide the operator with enough information about individual components necessary for service provision. Additionally, the path discovery algorithm and creation of the composite reliability block diagram need optimization when applied to networks with a high degree of connectivity, such as wireless mesh networks. Combining sets of components with a low variability in user-perceived availability to reduce the size of the topology graph should be considered. Finally, extending the methodology to evaluate different time-dependent dependability properties like performability remains an open issue. With respect to instantaneous availability, future work should focus on improving the accuracy of the resulting availability estimation especially with respect to short term effects on the infrastructure. This includes examining extensions to support variable failure and repair rates and a proper load model.

9.3 Evaluation of Service Discovery Responsiveness

Part III provides an extensive evaluation of the responsiveness of SD, the probability to successfully discover service providers within deadlines. To the best of our knowledge, no work has been done that covers an evaluation of this impor-

tant property both by experiments and using stochastic models. Results of the research on SD responsiveness presented in this work have in parts been published in [3, 1, 4, 6, 7, 9, 11, 8].

9.3.1 Experimental Evaluation

First, we examined the dependability of decentralized active SD in experiments, to observe the behavior of SD responsiveness in three common scenarios. This has been done in two experiment environments. First, a hypervisor based virtual testbed and second, the *Distributed Embedded Systems* (DES) wireless testbed at *Freie Universität Berlin* (FUB). In the DES testbed, the *ExCovery* experiment framework was employed to run all series of experiments. Responsiveness of *Domain Name System* (DNS) based discovery was evaluated under varying fault intensity and with up to 50 service instances. Furthermore, we analyzed the impact of the deadline of the SD operation, the distance of nodes, the load in the network and the required number of providers to be discovered. Analysis in the DES testbed was 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. The analysis provides an extensive evaluation of SD responsiveness with a realistic fault model. Presented results are applicable to various types of optimizations in wireless mesh networks.

The results in the virtual testbed show that the responsiveness of the used SDPs decreases dramatically with moderate packet loss of around 20%. It decreases non-proportional when more service instances need to be discovered. At higher packet loss rates the decrease becomes exponential with the number of nodes to be discovered such that SD becomes practically impossible. Responsiveness can be improved if a low coverage is acceptable. This can be the case, for example, if redundant service instances are introduced to the network and only a fixed number of service instances needs to be discovered, regardless of the total number of instances in the network. However, this assumption is only valid in networks with low packet loss. In less reliable networks, redundant service instances and the caused overhead in communication could in fact worsen responsiveness.

In addition to the experiments on SD, an analysis of the long-term behavior of the DES testbed and the effects of internal and external faults was 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 has been shown that the effect of these faults is considerable and has to be taken into account when interpreting results. Experiment analysis based on average measurements should be done very carefully. Choosing the measurement history window size too big can easily lead to over or underestimating of the testbed link quality.

Experiment Environment *ExCovery*

The experiment framework *ExCovery* developed for this work has been tried and refined in a manifold of SD dependability experiments over the last two years, which were carried out on the wireless DES testbed at FUB. The core of *ExCovery* is a formal experiment description that supports automated checking, execution and additional features, such as visualization of experiments. In addition to that, the framework offers unified concepts for experiment execution, measurement and storage of results. As such, *ExCovery* is expected to foster experiment repeatability, comparability and transparency. To support this claim and to facilitate transparency and repeatability, all experiment descriptions and results of this work are made available for interested researchers on request. *ExCovery* is available for download in a public repository under the permissive MIT license [8].

ExCovery was specifically developed for the experiments on SD presented in Chapters 6 and 8 but has generic support for experiments on the dependability of distributed processes. We provided an abstract description of SD as experiment process. The description covers the specification of the individual processes of an experiment and their actors: Fault injection, environment manipulation and the main process under experimentation (in this case SD) are expressed as interdependent series of actions and events. Execution takes care of controlling the individual nodes during experiment runtime, to make sure each run of an experiment has a clean and defined environment and each node acts according to the experiment description. *ExCovery* manages series of experiments and recovers from failures by resuming aborted runs. Measurements are taken both on the level of experiment process actions and events, to record the behavior as seen by the application, and on the level of raw network packets. Measurements are stored in a unified database format to facilitate sharing and comparison of results. *ExCovery* has already been employed in a number of works from other authors and is still under active development.

9.3.2 Stochastic Modeling

This work proposes a stochastic model family to evaluate the user-perceived responsiveness of active decentralized SD. The family consists of a hierarchy of Markov and semi-Markov processes that are parametrized to allow instantiation for diverse SD scenarios and use current network monitoring data as input. To put the models into use, a methodology is introduced that works specifically in wireless mesh networks with proactive routing. Upon request, the methodology generates and solves model instances bottom up for specific SD scenarios. Especially the abstractions at the lower layers lead to a comparably low complexity, which allows for diverse aggregate calculations, such as the average responsiveness of one provider for all clients in the network. The methodology relies on the Monte Carlo method *Probabilistic Breadth-First Search* (PBFS) to estimate the behavior of SD multicast communication. PBFS was developed during the work on this dissertation and approx-

imates the reachability and other metrics of network-wide broadcasts by random sampling the packet traversal through a probabilistic graph.

Using monitoring data from the DES testbed, responsiveness is evaluated for the three most prevalent SDPs in IP networks: *Service Location Protocol* (SLP), *Simple Service Discovery Protocol* (SSDP) and *Zeroconf* discovery. First, the responsiveness for different pairs of requester and provider is compared. Second, the average responsiveness of a single provider, depending on the topology, is analyzed. Results demonstrate that responsiveness varies dramatically depending on the position of nodes in the network and the overall link quality. Thus, they confirm the findings from the experiments. The results further indicate that, with short deadlines close to the individual retry intervals, the right retry timing strategy is more important than the communication mechanism. With longer deadlines, using the more reliable unicast instead of multicast consistently improves responsiveness. In either case, the fixed strategies of current SDPs struggle to achieve a high responsiveness in these dynamic and inherently unreliable networks. Finally, a new metric *expected responsiveness distance* d_{er} is introduced, estimating the maximum hop distance from a provider at which nodes can discover it with a required responsiveness. To deploy a responsive service with a minimum number of nodes, every requester in the network should be within the d_{er} of at least one provider. The d_{er} of two different providers is evaluated and the results underline the importance of position when placing service instances.

Validation shows that the highest layer discrete time Markov model, which represents the SDP itself, is able to estimate the actual measured responsiveness with a very low error at any given time. It uses input data in the form of a distribution of response times of previous discovery operations. Model estimations and experiment measurements consistently converge to the same value and correlate with a root mean squared error of 0.03 and below in all cases. The Pearson product-moment correlation coefficient r is well above 0.99 in all cases. This validates the model as a proper representation of the discovery operation. The lower layer models, which calculate the probability and time distribution of traversal for specific network packets, are able to estimate the actual behavior with a low error under the condition that the input data properly reflects the current network quality. It has been shown that the used *Expected Transmission Count* (ETX) metric, as given by the routing layer, has its shortcomings and cannot always cope with the dynamics of wireless link qualities. The quality of the ETX metric increases the more static the network is, thus reducing the error of the estimations given by the presented stochastic models. We can conclude that given a precise metric of the link quality, the complete hierarchy of stochastic models is able to estimate the responsiveness for any given SD communication pair with a high accuracy.

9.3.3 Outlook

Responsiveness evaluation of SD in dynamic and decentralized networks remains challenging. With increasing demand for real-time systems, responsiveness optimization will become one of the main issues. With self-configuring networks entering dependability critical domains, our experimental and analytical evaluation has shown that distributed SD has to be used with caution, especially in wireless scenarios where packet loss cannot be neglected. Applying advanced cache optimizations to reduce transmissions as proposed in [45] could improve responsiveness in such scenarios. Optimizing retry strategies seems to be indeed one promising next step. However, further investigations are needed since not all improvements would be compliant with existing standards that are already deployed in current network devices. Some adaptations such as accepting non-authoritative discovery responses from any node in the service network might improve discovery responsiveness but reduce the reliability of service discovery as a whole.

Future research could extend the presented hierarchy of stochastic models to support centralized and hybrid SD architectures. Also, the higher layer discovery model is not time-dependent and needs to be regenerated for different deadlines if the number of retries changes. The probability P_e to transition to the error state after the last retry needs to be calculated for each deadline. However, the discovery model provides an intuitive representation of the SDP communication and can be calculated with low complexity. These advantages are important when optimizing SDPs for fixed deadlines as given by the application layer. Nevertheless, the time dependence of the lower layer models could be combined in a single discovery layer model which uses the deadline as a parameter and calculates the distribution of arrival times in the absorbing states. This in turn could be used to assess responsiveness. Such model could then support the development of novel SDPs that, for example, support variable retry intervals depending on the state of the network.

Final Remarks

The main contributions of this dissertation are two different methodologies for user-perceived dependability evaluation in service networks. Both methodologies have the same foundation, to generate models for dependability evaluation bottom up based on the current monitored state of the network and to allow a time-dependent dependability evaluation using these models. This way, they enable optimization of service networks with respect to known or predicted usage patterns.

The presented solutions can be used in service networks to analyze the quality of service deployments for different client perspectives. Service discovery is an integral parts of such deployments. Furthermore, they help to improve them by estimating the quality when placing additional instances, moving existing instances and adding or replacing network equipment. Since the user-perspective is valid for both the clients and providers, both parties may use the models given that they have sufficient input data to generate and solve them.

Finally, the methodologies developed in this work are one step away from anticipating novel service dependability models which combine service discovery, timeliness, placement and usage, areas that until now have been treated to a large extent separately. Such models will become the basis for future adaptive and autonomous networks such as the Internet of Things and its successors.

References to Works of the Author

1. Dittrich, A.: Self-aware adaptive service networks with dependability guarantees. In: K. Bollue, D. Gückel, U. Loup, J. Spönemann, M. Winkler (eds.) Joint Workshop of the German Research Training Groups in Computer Science, p. 88. Verlagshaus Mainz, Aachen, Germany (2010). Extended abstract
2. Dittrich, A., Kaitovic, I., Murillo, C., Rezende, R.: A model for evaluation of user-perceived service properties. In: 27^{th} International Parallel & Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW), pp. 1508–1517. IEEE Computer Society, Boston, MA, USA (2013). doi:10.1109/IPDPSW.2013.163
3. Dittrich, A., Kowal, J., Malek, M.: Designing survivable services from independent components with basic functionality. In: International Workshop on Dependable Network Computing and Mobile Systems (DNCMS), pp. 33–38. IEEE Computer Society, Naples, Italy (2008). doi:10.13140/2.1.1438.3684
4. Dittrich, A., Lichtblau, B., Rezende, R., Malek, M.: Modeling responsiveness of decentralized service discovery in wireless mesh networks. In: K. Fischbach, U.R. Krieger (eds.) MMB & DFT, *Lecture Notes in Computer Science*, vol. 8376, chap. 7, pp. 88–102. Springer International Publishing (2014). doi:10.1007/978-3-319-05359-2_7
5. Dittrich, A., Rezende, R.: Model-driven evaluation of user-perceived service availability. In: M. Vieira, J.C. Cunha (eds.) Dependable Computing, *Lecture Notes in Computer Science*, vol. 7869, pp. 39–53. Springer Berlin Heidelberg (2013). doi:10.1007/978-3-642-38789-0_4
6. Dittrich, A., Salfner, F.: Experimental responsiveness evaluation of decentralized service discovery. In: 24^{th} International Parallel & Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW), pp. 1–7. IEEE Computer Society, Atlanta, GA, USA (2010). doi:10.1109/IPDPSW.2010.5470861
7. Dittrich, A., Solis Herrera, D., Coto, P., Malek, M.: Responsiveness of service discovery in wireless mesh networks. In: 20^{th} Pacific Rim International Symposium on Dependable Computing (PRDC). IEEE Computer Society, Singapore (2014). doi:10.1109/PRDC.2014.38
8. Dittrich, A., Wanja, S.: *ExCovery* – a framework for distributed system experiments. Software repository (2014). Version 0.2, Published under the MIT License. URL http://adittrich.github.io/excovery
9. Dittrich, A., Wanja, S., Malek, M.: *ExCovery* – a framework for distributed system experiments and a case study of service discovery. In: 28^{th} International Parallel & Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW), pp. 1314–1323. IEEE Computer Society, Phoenix, AZ, USA (2014). doi:10.1109/IPDPSW.2014.147
10. Goldschmidt, T., Dittrich, A., Malek, M.: Quantifying criticality of dependability-related it organization processes in cobit. In: 15^{th} Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 336–341. IEEE Computer Society, Shanghai, China (2009). doi:10.1109/PRDC.2009.60

11. Lichtblau, B., Dittrich, A.: Probabilistic breadth-first search – a method for evaluation of network-wide broadcast protocols. In: 6^{th} IEEE/ACM/IFIP International Conference on New Technologies, Mobility and Security (NTMS). IEEE Computer Society, Dubai, UAE (2014). doi:10.1109/NTMS.2014.6814046
12. Rezende, R., Dittrich, A., Malek, M.: User-perceived instantaneous service availability evaluation. In: 19^{th} Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 273–282. IEEE Computer Society, Vancouver, Canada (2013). doi:10.1109/PRDC.2013.49

References

13. The NS-3 network simulator. Webpage (2012). URL http://www.nsnam.org
14. Aboba, B., Thaler, D., Esibov, L.: Link-local multicast name resolution (LLMNR). RFC 4795 (Informational) (2007)
15. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer Networks **38**(4), 393–422 (2002). doi:10.1016/S1389-1286(01)00302-4
16. Akyildiz, I.F., Wang, X., Wang, W.: Wireless mesh networks: a survey. Computer Networks **47**(4), 445–487 (2005). doi:10.1016/j.comnet.2004.12.001
17. Apache Software Foundation: Apache River - Jini^{TM} Network Technology Specifications, 2.2.2 edn. (2013)
18. Arabshian, K., Schulzrinne, H.: Gloserv: global service discovery architecture. In: 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MOBIQUITOUS), pp. 319–325 (2004). doi:10.1109/MOBIQ.2004.1331738
19. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A.I.S.M.Z.: Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, UC Berkeley Reliable Adaptive Distributed Systems Laboratory (2009)
20. ARTEMIS Embedded Computing Systems Initiative: CHESS project. Webpage (2012). URL http://www.chess-project.org
21. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer Networks **54**(15), 2787–2805 (2010). doi:10.1016/j.comnet.2010.05.010
22. The avahi project. Software (2014). URL http://avahi.org
23. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1), 11–33 (2004). doi:10.1109/TDSC.2004.2
24. Bailey, R.A.: Design of Comparative Experiments, *Cambridge Series in Statistical and Probabilistic Mathematics*, vol. 25. Cambridge University Press (2008)
25. Barborak, M., Dahbura, A., Malek, M.: The consensus problem in fault-tolerant computing. ACM Computing Surveys **25**(2), 171–220 (1993). doi:10.1145/152610.152612
26. Baresi, L., Guinea, S.: Towards dynamic monitoring of WS-BPEL processes. In: Service-Oriented Computing - (ICSOC) 2005, *Lecture Notes in Computer Science*, vol. 3826, pp. 269–282. Springer Berlin Heidelberg (2005). doi:10.1007/11596141_21
27. Barham, P.R., Dragovic, B., Fraser, K.A., Hand, S.M., Harris, T.L., Ho, A.C., Kotsovinos, E., Madhavapeddy, A.V.S., Neugebauer, R., Pratt, I.A., Warfield, A.K.: Xen 2002. Tech. Rep. UCAM-CL-TR-553, University of Cambridge, 15 JJ Thomson Avenue, Cambridge, UK (2003)
28. Beaudry, M.: Performance-related reliability measures for computing systems. IEEE Transactions on Computers **27**(6), 540–547 (1978). doi:10.1109/TC.1978.1675145

29. Ben Mokhtar, S., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: Easy: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. Journal of Systems and Software **81**(5), 785–808 (2008). doi:10.1016/j.jss.2007.07.030. Software Process and Product Measurement
30. Bernardi, S., Merseguer, J., Petriu, D.: An UML profile for dependability analysis and modeling of software systems. Tech. Rep. RR-08-05, University of Zaragoza (2008)
31. Bettstetter, C., Renner, C.: A comparison of service discovery protocols and implementation of the service location protocol. In: 6^{th} EUNICE Open European Summer School: Innovative Internet Applications (2000)
32. Bézivin, J., Gerbé, O.: Towards a precise definition of the OMG/MDA framework. In: 16^{th} Annual International Conference on Automated Software Engineering (ASE), pp. 273–280. IEEE Computer Society Press (2001). doi:10.1109/ASE.2001.989813
33. Bianchi, G.: Performance analysis of the IEEE 802.11 distributed coordination function. IEEE Journal on Selected Areas in Communications **18**(3), 535–547 (2000). doi:10.1109/49.840210
34. Bittanti, S., Campi, M.C., Prandini, M.: How many experiments are needed to adapt? In: A. Chiuso, S. Pinzoni, A. Ferrante (eds.) Modeling, Estimation and Control, *Lecture Notes in Control and Information Sciences*, vol. 364, pp. 5–14. Springer Berlin Heidelberg (2007). doi:10.1007/978-3-540-73570-0_2
35. Bluetooth SIG, Inc.: Specification of the Bluetooth System, 4.1 edn. (2013)
36. Blywis, B., Günes, M., Juraschek, F., Hahm, O.: Properties and topology of the DES-testbed. Tech. Rep. TR-B-11-02, Freie Universität Berlin, Computer Systems & Telematics, Takustraße 9, 14195 Berlin, Germany (2011)
37. Blywis, B., Reinecke, P., Günes, M., Wolter, K.: Gossip routing, percolation, and restart in wireless multi-hop networks. In: Wireless Communications and Networking Conference (WCNC), pp. 3019–3023. IEEE Computer Society (2012). doi:10.1109/WCNC.2012.6214322
38. Bocciarelli, P., D'Ambrogio, A.: Model-driven performability analysis of composite web services. In: SIPEW, pp. 228–246 (2008). doi:10.1007/978-3-540-69814-2_15
39. Bohnenkamp, H., van der Stok, P., Hermanns, H., Vaandrager, F.: Cost-optimization of the IPv4 zeroconf protocol. In: International Conference on Dependable Systems and Networks, pp. 531–540. IEEE Computer Society, Los Alamitos, CA, USA (2003). doi:10.1109/DSN.2003.1209963
40. Bondavalli, A., Lollini, P., Montecchi, L.: Analysis of user perceived qos in ubiquitous umts environments subject to faults. In: U. Brinkschulte, T. Givargis, S. Russo (eds.) Software Technologies for Embedded and Ubiquitous Systems (SEUS), *Lecture Notes in Computer Science*, vol. 5287, chap. 17, pp. 186–197. Springer Berlin Heidelberg (2008). doi:10.1007/978-3-540-87785-1_17
41. Bosse, S., Schulz, C., Turowski, K.: Predicting availability and response times of IT services. In: 22^{nd} European Conference on Information Systems (ECIS). Tel Aviv, Israel (2014)
42. Brandes, U., Erlebach, T. (eds.): Network Analysis – Methodological Foundations, *Lecture Notes in Computer Science*, vol. 3418. Springer Berlin Heidelberg (2005). doi:10.1007/b106453
43. Brüning, S., Weissleder, S., Malek, M.: A fault taxonomy for service-oriented architecture. In: Informatik-Berichte, vol. 215. Humboldt-Universität zu Berlin, Institut für Informatik (2007)
44. Brüning, S., Weissleder, S., Malek, M.: A fault taxonomy for service-oriented architecture. In: 10^{th} High Assurance Systems Engineering Symposium (HASE), pp. 367–368. IEEE Computer Society (2007). doi:10.1109/HASE.2007.46
45. Campo, C., García-Rubio, C.: DNS-based service discovery in ad hoc networks: Evaluation and improvements. In: P. Cuenca, L. Orozco-Barbosa (eds.) Personal Wireless Communications, *Lecture Notes in Computer Science*, vol. 4217, pp. 111–122. Springer Berlin Heidelberg (2006). doi:10.1007/11872153_10

46. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web **1**(3), 281–308 (2004). doi:10.1016/j.websem.2004.03.001
47. Cervantes, H., Hall, R.: Autonomous adaptation to dynamic availability using a service-oriented component model. In: 26^{th} International Conference on Software Engineering (ICSE), pp. 614–623. IEEE Computer Society (2004). doi:10.1109/ICSE.2004.1317483
48. Chakraborty, D., Joshi, A., Yesha, Y., Finin, T.: Toward distributed service discovery in pervasive computing environments. Transactions on Mobile Computing **5**(2), 97–112 (2006). doi:10.1109/TMC.2006.26
49. Chan, K.S.M., Bishop, J., Steyn, J., Baresi, L., Guinea, S.: A fault taxonomy for web service composition. In: 3^{rd} International Workshop on Engineering Service Oriented Applications (WESOA), *Lecture Notes in Computer Science*, vol. 4907, pp. 363–375. Springer (2007). doi:10.1007/978-3-540-93851-4_36
50. Chen, P., Johansson, K.H., Balister, P., Bollobás, B., Sastry, S.: Multi-path routing metrics for reliable wireless mesh routing topologies. Tech. Rep. TRITA-EE 2011:033, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden (2011)
51. Cheshire, S.: Zero configuration networking (zeroconf). Webpage (2009). URL http://www.zeroconf.org
52. Cheshire, S., Aboba, B., Guttman, E.: Dynamic configuration of IPv4 link-local addresses. RFC 3927 (Proposed Standard) (2005)
53. Cheshire, S., Krochmal, M.: DNS-based service discovery. RFC 6763 (Proposed Standard) (2013)
54. Cheshire, S., Krochmal, M.: Multicast DNS. RFC 6762 (Proposed Standard) (2013)
55. Cheshire, S., Steinberg, D.H.: Zero Configuration Networking - The Definitive Guide, 1^{st} edn. O'Reilly Media (2005)
56. Ciardo, G., Marie, R.A., Sericola, B., Trivedi, K.S.: Performability analysis using semi-markov reward processes. IEEE Transactions on Computers **39**(10), 1251–1264 (1990). doi:10.1109/12.59855
57. Clausen, T., Jacquet, P.: Optimized link state routing protocol (OLSR). RFC 3626 (Experimental) (2003)
58. Controneo, D., di Flora, C., Russo, S.: Improving dependability of service oriented architectures for pervasive computing. In: 8^{th} International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS), pp. 74–81 (2003). doi:10.1109/WORDS.2003.1218068
59. Cordeiro, W., Aguiar, E., Moreira, W., Abelem, A., Stanton, M.: Providing quality of service for mesh networks using link delay measurements. In: 16^{th} International Conference on Computer Communications and Networks (ICCCN), pp. 991–996 (2007). doi:10.1109/ICCCN.2007.4317947
60. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2 edn. McGraw-Hill Higher Education (2001)
61. Coto, P.: Responsiveness analysis of service location protocol in wireless mesh testbed. Master's thesis, Università della Svizzera Italiana (USI), Lugano, Switzerland (2014)
62. Couto, D.S.J.D., Aguayo, D., Bicket, J., Morris, R.: A high-throughput path metric for multi-hop wireless routing. In: 9^{th} annual international conference on Mobile computing and networking, pp. 134–146. ACM, San Diego, USA (2003). doi:10.1145/938985.939000
63. Cristian, F.: Understanding fault-tolerant distributed systems. ACM Communications **34**(2), 56–78 (1991). doi:10.1145/102792.102801
64. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal **45**(3), 621–645 (2006). doi:10.1147/sj.453.0621
65. Czerwinski, S.E., Zhao, B.Y., Hodes, T.D., Joseph, A.D., Katz, R.H.: An architecture for a secure service discovery service. In: 5^{th} Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)International Conference on Mobile Computing and Networking (MobiCom), MobiCom '99, pp. 24–35. ACM, Seattle, Washington, USA (1999). doi:10.1145/313451.313462

66. Dabrowski, C.E., Mills, K.L.: Analyzing properties and behavior of service discovery protocols using an architecture-based approach. In: Working Conference on Complex and Dynamic Systems Architecture (2001)
67. Dabrowski, C.E., Mills, K.L.: Understanding self-healing in service-discovery systems. In: Workshop on Self-healing systems (WOSS), pp. 15–20. ACM (2002). doi:10.1145/582128.582132
68. Dabrowski, C.E., Mills, K.L., Elder, J.: Understanding consistency maintenance in service discovery architectures during communication failure. In: 3^{rd} International Workshop on Software and Performance (WOSP), pp. 168–178. ACM, Rome, Italy (2002). doi:10.1145/584369.584396
69. Dabrowski, C.E., Mills, K.L., Elder, J.: Understanding consistency maintenance in service discovery architectures in response to message loss. In: 4^{th} Annual International Workshop on Active Middleware Services, pp. 51–60 (2002). doi:10.1109/AMS.2002.1029690
70. Dabrowski, C.E., Mills, K.L., Quirolgico, S.: A model-based analysis of first-generation service discovery systems. Tech. Rep. ADA523379, NIST National Institute of Standards and Technology, Gaithersburg, MD, USA (2005)
71. Dabrowski, C.E., Mills, K.L., Quirolgico, S.: Understanding failure response in service discovery systems. J. Syst. Softw. **80**(6), 896–917 (2007). doi:10.1016/j.jss.2006.11.017
72. Dabrowski, C.E., Mills, K.L., Rukhin, A.: Performance of service-discovery architectures in response to node failures. Tech. Rep. ADA511255, NIST National Institute of Standards and Technology, Gaithersburg, MD, USA (2003)
73. Dai, Y.S., Xie, M., Poh, K.L., Liu, G.Q.: A study of service reliability and availability for distributed systems. Reliability Engineering & System Safety **79**(1), 103–112 (2003). doi:DOI: 10.1016/S0951-8320(02)00200-4
74. Dean, A., Voss, D. (eds.): Design and Analysis of Experiments. Springer Texts in Statistics. Springer New York (1999). doi:10.1007/b97673
75. Debian – the universal operating system. Webpage (2013). URL http://www.debian.org
76. Derrick, T.R., Bates, B.T., Dufek, J.S.: Evaluation of time-series data sets using the pearson product-moment correlation coefficient. Medicine and science in sports and exercise **26**(7), 919—928 (1994)
77. Ding, Z., Chen, Z., Liu, J.: A rigorous model of service component architecture. Electronic Notes in Theoretical Computer Science **207**, 33–48 (2008). doi:10.1016/j.entcs.2008.03.084
78. Duda, A.: Understanding the performance of 802.11 networks. In: 19^{th} International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 1–6. IEEE Computer Society (2008). doi:10.1109/PIMRC.2008.4699942
79. Edwards, W.K.: Discovery systems in ubiquitous computing. IEEE Pervasive Computing **5**(2), 70–77 (2006). doi:10.1109/MPRV.2006.28
80. Enneya, N., Oudidi, K., Elkoutbi, M.: Enhancing delay in manet using olsr protocol. International Journal of Communications, Network and System Sciences **2**(5), 392–399 (2009). doi:10.4236/ijcns.2009.25044
81. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design, 1^{st} edn. The Prentice Hall Service Technology Series from Thomas Erl. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
82. Esposito, P.M., Campista, M.E.M., Moraes, I.M., Costa, L.H.M.K., Duarte, O.C.M.B., Rubinstein, M.G.: Implementing the expected transmission time metric for olsr wireless mesh networks. In: 1^{st} IFIP Wireless Days (WD), pp. 1–5 (2008). doi:10.1109/WD.2008.4812900
83. Eusgeld, I., Fechner, B., Salfner, F., Walter, M., Limbourg, P., Zhang, L.: Hardware Reliability, chap. 9, pp. 59–103. Vol. 4909 of Eusgeld et al. [85] (2008). doi:10.1007/978-3-540-68947-8_9
84. Eusgeld, I., Fraikin, F., Rohr, M., Salfner, F., Wappler, U.: Software Reliability, chap. 10, pp. 104–125. Vol. 4909 of Eusgeld et al. [85] (2008). doi:10.1007/978-3-540-68947-8_10
85. Eusgeld, I., Freiling, F.C., Reussner, R. (eds.): Dependability Metrics, Advanced Lectures, *Lecture Notes in Computer Science*, vol. 4909. Springer Berlin Heidelberg (2008). doi:10.1007/978-3-540-68947-8

86. Eusgeld, I., Happe, J., Limbourg, P., Rohr, M., Salfner, F.: Performability, chap. 24, pp. 245–254. Vol. 4909 of Eusgeld et al. [85] (2008). doi:10.1007/978-3-540-68947-8_24
87. Even, S.: Graph Algorithms, 2^{nd} edn. Cambridge University Press (2011)
88. Feitelson, D.G.: Experimental computer science: The need for a cultural change. Tech. rep., The Hebrew University of Jerusalem (2006). Manuscript
89. Flores-Cortés, C.A., Blair, G.S., Grace, P.: An adaptive middleware to overcome service discovery heterogeneity in mobile ad hoc environments. IEEE Distributed Systems Online **8**(7) (2007). doi:10.1109/MDSO.2007.41
90. Fussell, D.S., Malek, M. (eds.): Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems. Kluwer Academic Publishers (1995). doi:10.1007/978-1-4615-2271-3
91. Gershenfeld, N., Krikorian, R., Cohen, D.: The internet of things. Scientific American **291**(4), 76–81 (2004)
92. Giordano, M.: Dns-based discovery system in service oriented programming. In: P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld, M. Bubak (eds.) Advances in Grid Computing (EGC), *Lecture Notes in Computer Science*, vol. 3470, pp. 840–850. Springer Berlin Heidelberg (2005). doi:10.1007/11508380_86
93. Goland, Y.Y., Cai, T., Leach, P., Gu, Y.: Simple service discovery protocol/1.0 – operating without an arbiter. Internet Draft (1999). URL ftp://ftp.pwg.org/pub/pwg/ipp/new_SSDP/draft-cai-ssdp-v1-03.txt
94. Goldschmidt, T., Malek, M.: Ranking of dependability-relevant indicators for availability enhancement of enterprise information systems. Tech. rep., Humboldt-Universität zu Berlin (2012)
95. Gorbenko, A., Kharchenko, V., Tarasyuk, O., Chen, Y., Romanovsky, A.: The threat of uncertainty in service-oriented architecture. In: RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems (SERENE), pp. 49–54. ACM, New York, NY, USA (2008). doi:10.1145/1479772.1479781
96. Grassi, V., Mirandola, R., Sabetta, A.: A model-driven approach to performability analysis of dynamically reconfigurable component-based systems. In: 6^{th} International Workshop on Software and Performance (WOSP), pp. 103–114. ACM, New York, NY, USA (2007). doi:10.1145/1216993.1217011
97. Gray, J.: Why do computers stop and what can be done about it? Tech. Rep. TR-85.7, Tandem (1985)
98. Grottke, M., Sun, H., Fricks, R.M., Trivedi, K.S.: Ten fallacies of availability and reliability analysis. In: T. Nanya, F. Maruyama, A. Pataricza, M. Malek (eds.) Service Availability, *Lecture Notes in Computer Science*, vol. 5017, pp. 187–206. Springer Berlin Heidelberg (2008). doi:10.1007/978-3-540-68129-8_15
99. Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., Savio, D.: Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. IEEE Transactions on Services Computing **3**(3), 223–235 (2010). doi:10.1109/TSC.2010.3
100. Günes, M., Blywis, B., Juraschek, F.: Concept and design of the hybrid distributed embedded systems testbed. Tech. Rep. TR-B-08-10, Freie Universität Berlin, Computer Systems & Telematics, Takustraße 9, 14195 Berlin, Germany (2008)
101. Günes, M., Blywis, B., Juraschek, F., Schmidt, P.: Practical issues of implementing a hybrid multi-nic wireless mesh-network. Tech. Rep. TR-B-08-11, Freie Universität Berlin, Computer Systems & Telematics, Takustraße 9, 14195 Berlin, Germany (2008)
102. Guo, Z., Malakooti, S., Sheikh, S., Al-Najjar, C., Malakooti, B.: Multi-objective olsr for proactive routing in manet with delay, energy, and link lifetime predictions. Applied Mathematical Modelling **35**(3), 1413 – 1426 (2011). doi:10.1016/j.apm.2010.09.019
103. Guttman, E.: Service location protocol: automatic discovery of ip network services. IEEE Internet Computing **3**(4), 71–80 (1999). doi:10.1109/4236.780963
104. Guttman, E.: Autoconfiguration for IP networking: Enabling local communication. IEEE Internet Computing **5**(3), 81–86 (2001). doi:10.1109/4236.935181
105. Guttman, E.: Zeroconf host profile applicability statement. Internet Draft (2001)
106. Guttman, E., Perkins, C., Veizades, J., Day, M.: Service location protocol, version 2. RFC 2608 (Proposed Standard) (1999). Updated by RFC 3224

107. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: 14^{th} Australasian Database Conference (ADC), pp. 191–200. Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2003)
108. Härri, J., Bonnet, C., Filali, F.: OLSR and MPR: mutual dependences and performances. In: K. Agha, I. Guérin Lassous, G. Pujolle (eds.) Challenges in Ad Hoc Networking, *IFIP International Federation for Information Processing*, vol. 197, pp. 67–71. Springer US (2006). doi:10.1007/0-387-31173-4_8
109. Hartung, J., Elpelt, B., Klösener, K.H.: Statistik, Lehr- und Handbuch der angewandten Statistik, 15 edn. R. Oldenbourg Verlag München Wien (2009)
110. Hattig, M.: Zeroconf requirements. Internet Draft (2000)
111. Haverkort, B.R.H.M., Niemegeers, I.G.: Performability modelling tools and techniques. Performance evaluation **25**(1), 17–40 (1996). doi:10.1016/0166-5316(94)00038-7
112. Heegaard, P.E., Trivedi, K.S.: Network survivability modeling. Computer Networks **53**(8), 1215–1234 (2009). doi:10.1016/j.comnet.2009.02.014. Performance Modeling of Computer Networks: Special Issue in Memory of Dr. Gunter Bolch
113. Heimerdinger, W., Weinstock, C.: A conceptual framework for system fault tolerance. Tech. rep., Carnegie Mellon University (1992)
114. Helal, S.: Standards for service discovery and delivery. IEEE Pervasive Computing **1**(3), 95–100 (2002). doi:10.1109/MPRV.2002.1037728
115. Henderson, T.R., Roy, S., Floyd, S., Riley, G.F.: NS-3 project goals. In: Workshop on NS-2: the IP network simulator, WNS2 '06. ACM, New York, NY, USA (2006). doi:10.1145/1190455.1190468
116. Hinkelmann, K., Kempthorne, O.: Design and Analysis of Experiments, Introduction to Experimental Design, vol. 1, 2^{nd} edn. John Wiley and Sons, Inc. (2008)
117. Hodes, T., Czerwinski, S., Zhao, B., Joseph, A., Katz, R.: An architecture for secure wide-area service discovery. Wireless Networks **8**(2-3), 213–230 (2002). doi:10.1023/A:1013772027164
118. Hong, S.G., Srinivasan, S., Schulzrinne, H.: Measurements of multicast service discovery in a campus wireless network. In: Global Telecommunications Conference (GLOBECOM), pp. 1–6. IEEE Computer Society (2009). doi:10.1109/GLOCOM.2009.5426121
119. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Computing **9**(1), 75–81 (2005). doi:http://doi.ieeecomputersociety.org/10.1109/MIC.2005.21
120. IEEE Standards Association: IEEE standard for information technology – telecommunications and information exchange between systems – [...]. IEEE Std 802.11-2012 pp. 1–2793 (2012). doi:10.1109/IEEESTD.2012.6178212
121. Immonen, A., Niemelä, E.: Survey of reliability and availability prediction methods from the viewpoint of software architecture. Software and System Modeling **7**(1), 49–65 (2008)
122. Jaehoon Jeong Jungsoo Park, H.K.: Service discovery based on multicast DNS in IPv6 mobile ad-hoc networks. In: The 57^{th} IEEE Semiannual Vehicular Technology Conference (VTC 2003-Spring), vol. 3, pp. 1763–1767. IEEE Computer Society (2003)
123. Jalote, P.: Fault Tolerance in Distributed Systems. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1994)
124. Johnson Jr., A.M., Malek, M.: Survey of software tools for evaluating reliability, availability, and serviceability. ACM Computing Surveys **20**(4), 227–269 (1988). doi:10.1145/50020.50062
125. Johnson, D., Hu, Y., Maltz, D.: The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4. RFC 4728 (Experimental) (2007)
126. Johnson, D.B., Maltz, D.A., Broch, J.: Dsr: The dynamic source routing protocol for multihop wireless ad hoc networks. In: C.E. Perkins (ed.) Ad Hoc Networking, chap. 5, pp. 139–172. Addison-Wesley, Pittsburgh, PA 15213-3891 (2001)
127. Kaiser, D., Waldvogel, M.: Adding privacy to multicast DNS service discovery. Tech. Rep. KN-2014-DiSy-003, Distributed Systems Laboratory, Department of Computer and Information Science, University of Konstanz (2014)

128. Kaiser, D., Waldvogel, M.: Efficient privacy preserving multicast DNS service discovery. In: Workshop on Privacy-Preserving Cyberspace Safety and Security. IEEE Computer Society (2014). To appear
129. Knight, J.C., Strunk, E.A., Sullivan, K.J.: Towards a rigorous definition of information system survivability. DARPA Information Survivability Conference and Exposition, **1**, 78 (2003). doi:http://doi.ieeecomputersociety.org/10.1109/DISCEX.2003.1194874
130. Kohler, E.: Click for measurement. Tech. Rep. TR060010, UCLA Computer Science Department (2006)
131. Kozat, U.C., Tassiulas, L.: Network layer support for service discovery in mobile ad hoc networks. In: International Conference on Computer Communications INFOCOM, vol. 3, pp. 1965–1975. IEEE Computer Society (2003). doi:10.1109/INFCOM.2003.1209218
132. Kuhn, F., Wattenhofer, R., Zollinger, A.: Ad hoc networks beyond unit disk graphs. Wireless Networks **14**(5), 715–729 (2008). doi:10.1007/s11276-007-0045-6
133. Kulkarni, V.G.: Modeling and Analysis of Stochastic Systems, 2 edn. Texts in Statistical Science. Chapman & Hall/CRC (2010)
134. Lacage, M., Ferrari, M., Hansen, M., Turletti, T., Dabbous, W.: NEPI: Using independent simulators, emulators, and testbeds for easy experimentation. SIGOPS Operating Systems Review **43**(4), 60–65 (2010). doi:10.1145/1713254.1713268
135. Lee, C.: On quality of service management. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA 15213 (1999)
136. Lee, K.J., Kim, M.S., Cho, S.Y., Mun, B.I.: Delay-centric link quality aware olsr. In: The 30^{th} Conference on Local Computer Networks, pp. 690–696. IEEE Computer Society (2005). doi:10.1109/LCN.2005.49
137. Li, H., Cheng, Y., Zhou, C., Zhuang, W.: Minimizing end-to-end delay: a novel routing metric for multi-radio wireless mesh networks. In: International Conference on Computer Communications INFOCOM, pp. 46–54. IEEE Computer Society, Rio de Janeiro, Brazil (2009). doi:10.1109/INFCOM.2009.5061905
138. Li, L., Lamont, L.: A lightweight service discovery mechanism for mobile ad hoc pervasive environment using cross-layer design. In: 3^{rd} International Conference on Pervasive Computing and Communications Workshops (PerCom), pp. 55–59. IEEE Computer Society (2005). doi:10.1109/PERCOMW.2005.8
139. Lichtblau, B., Redlich, J.P.: Network-wide broadcasts for wireless mesh networks with regard to reliability. In: 19^{th} Symposium on Communications and Vehicular Technology in the Benelux (SCVT), pp. 1–6. IEEE Computer Society (2012). doi:10.1109/SCVT.2012.6399410
140. Lichtblau, B., Redlich, J.P.: Link quality based forwarder selection strategies for reliable network-wide broadcasts. In: 14^{th} International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1–6. IEEE Computer Society (2013). doi:10.1109/WoWMoM.2013.6583480
141. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Oasis reference model for service oriented architecture 1.0. Official OASIS Standard (2006)
142. Macker, J.: Simplified multicast forwarding. RFC 6621 (Experimental) (2012)
143. Macker, J., Dean, J., Taylor, I., Harrison, A.: Multicast service discovery profiles for deployment within dynamic edge networks. In: Military Communications Conference (MILCOM), pp. 1104–1109. IEEE Computer Society (2010). doi:10.1109/MILCOM.2010.5680087
144. Malek, M.: Responsive systems: The challenge for the nineties. Microprocessing and Microprogramming **30**, 9–16 (1990)
145. Malek, M.: Responsive systems: A marriage between real time and fault tolerance. In: M.D. Cin, W. Hohl (eds.) Fault-Tolerant Computing Systems, *Informatik-Fachberichte*, vol. 283, pp. 1–17. Springer Berlin Heidelberg (1991). doi:10.1007/978-3-642-76930-6_1
146. Malek, M.: A consensus-based framework and model for the design of responsive computing systems. In: Foundations of Dependable Computing, *The Springer International Series in Engineering and Computer Science*, vol. 283, pp. 3–21. Springer US (1994). doi:10.1007/978-0-585-27377-8_1

147. Malek, M., Hoffmann, G.A., Milanovic, N., Brüning, S., Meyer, R., Milic, B.: Methoden und werkzeuge zur verfügbarkeitsermittlung. In: Informatik-Berichte, 219. Institut für Informatik, Humboldt-Universität zu Berlin (2007)
148. Malek, M., Milic, B., Milanovic, N.: Analytical availability assessment of IT services. In: T. Nanya, F. Maruyama, A. Pataricza, M. Malek (eds.) Service Availability, *Lecture Notes in Computer Science*, vol. 5017, pp. 207–224. Springer Berlin Heidelberg (2008). doi:10.1007/978-3-540-68129-8_16
149. Marco Vieira Naaliel Mendes, J.D., Madeira, H.: The AMBER data repository. Tech. rep., University of Coimbra, Coimbra, Portugal (2008). URL http://amber-dbserver.dei.uc.pt:8080
150. Meshkova, E., Riihijärvi, J., Petrova, M., Mähönen, P.: A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. Computer Networks **52**(11), 2097 – 2128 (2008). doi:10.1016/j.comnet.2008.03.006
151. Metzger, A., Sammodi, O., Pohl, K.: Accurate proactive adaptation of service-oriented systems. In: J. Cámara, R. de Lemos, C. Ghezzi, A. Lopes (eds.) Assurances for Self-Adaptive Systems, *Lecture Notes in Computer Science*, vol. 7740, pp. 240–265. Springer Berlin Heidelberg (2013). doi:10.1007/978-3-642-36249-1_9
152. Meyer, J.F.: On evaluating the performability of degradable computing systems. IEEE Transactions on Computers **29**(8), 720–731 (1980). doi:10.1109/TC.1980.1675654
153. Meyer, J.F., Sanders, W.H.: Specification and construction of performability models. In: 2^{nd} International Workshop on Performability Modelling of Computer and Communication Systems (1993)
154. Mian, A.N., Baldoni, R., Beraldi, R.: A survey of service discovery protocols in multihop mobile ad hoc networks. IEEE Pervasive Computing **8**(1), 66–74 (2009). doi:10.1109/MPRV.2009.2
155. Milanovic, N.: Contract-based web service composition. Ph.D. thesis, Institut für Informatik, Humboldt-Universität zu Berlin, June (2006)
156. Milanovic, N.: Models, methods and tools for availability assessment of IT-services and business processes. Habilitation, Technische Universität Berlin (2010)
157. Milanovic, N., Milic, B.: Automatic generation of service availability models. IEEE Transactions on Services Computing **4**(1), 56–69 (2011). doi:10.1109/TSC.2010.11
158. Milanovic, N., Milic, B., Malek, M.: Modeling business process availability. In: Congress on Services - Part I, pp. 315–321. IEEE Computer Society (2008). doi:10.1109/SERVICES-1.2008.9
159. Milic, B.: Distributed biconnectivity testing in wireless multi-hop networks. Dissertation, Humboldt-Universität zu Berlin (2010)
160. Milic, B., Malek, M.: NPART - node placement algorithm for realistic topologies in wireless multihop network simulation. In: 2^{nd} International Conference on Simulation Tools and Techniques (Simutools), Simutools '09, pp. 9:1–9:10. ACM (2009). doi:10.4108/ICST.SIMUTOOLS2009.5669
161. Milic, B., Malek, M.: Properties of wireless multihop networks in theory and practice. In: S. Misra, I. Woungang, S. Chandra Misra (eds.) Guide to Wireless Ad Hoc Networks, Computer Communications and Networks, pp. 1–26. Springer London (2009). doi:10.1007/978-1-84800-328-6_1
162. Milic, B., Malek, M.: Accuracy of link status detection in wireless multi-hop networks. In: 13th international conference on Modeling, analysis, and simulation of wireless and mobile systems (MSWIM), pp. 122–131. ACM, New York, NY, USA (2010). doi:10.1145/1868521.1868543
163. Mockapetris, P.V.: Domain names - implementation and specification. RFC 1035 (Internet Standard) (1987). Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604
164. Montgomery, D.C.: Design and Analysis of Experiments, 7^{th} edn. John Wiley and Sons, Inc. (2009)
165. Murillo, C.: Model-driven evaluation of user-perceived service availability. Master thesis, Università della Svizzera Italiana (USI), Lugano, Switzerland (2013)

166. Naimi, A.M., Jacquet, P.: One-hop delay estimation in 802.11 ad hoc networks using the OLSR protocol. Tech. Rep. RR-5327, INRIA (2004)
167. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: 11^{th} International Conference on World Wide Web (WWW), pp. 77–88. ACM, New York, NY, USA (2002). doi:10.1145/511446.511457
168. OASIS Committee: Service Component Architecture Assembly Model Specification Version 1.1 (2011)
169. Object Management Group: Modeling and Analysis of Real-Time Embedded Systems (MARTE) Profile Specification, 1.1 edn. (2011). URL http://www.omgmarte.org
170. Object Management Group: Unified Modeling Language Infrastructure, 2.4.1 edn. (2011). URL http://www.omg.org/spec/UML/2.4.1
171. Object Management Group: Unified Modeling Language Superstructure, 2.4.1 edn. (2011). Version 2.4.1. URL http://www.omg.org/spec/UML/2.4.1
172. Oh, C.S., Ko, Y.B., Kim, J.H.: A hybrid service discovery for improving robustness in mobile ad hoc networks. In: International Conference on Dependable Systems and Networks (DSN). IEEE Computer Society (2004)
173. Oikonomou, K., Koufoudakis, G., Aissa, S.: Probabilistic flooding coverage analysis in large scale wireless networks. In: 19^{th} International Conference on Telecommunications (ICT), pp. 1–6 (2012). doi:10.1109/ICTEL.2012.6221290
174. Oikonomou, K., Stavrakakis, I.: Performance analysis of probabilistic flooding using random graphs. In: International Symposium on a World of Wireless, Mobile and Multimedia Networks WoWMoM, pp. 1–6. IEEE Computer Society (2007). doi:10.1109/WOWMOM.2007.4351694
175. Oliveira, R., Bernardo, L., Pinto, P.: Modelling delay on IEEE 802.11 MAC protocol for unicast and broadcast nonsaturated traffic. In: Wireless Communications and Networking Conference (WCNC), pp. 463–467. IEEE Computer Society, Kowloon, Hongkong, China (2007). doi:10.1109/WCNC.2007.90
176. OpenSLP project: openSLP service location protocol. Webpage (2014). URL http://www.openslp.org
177. Paliwal, A.V., Shafiq, B., Vaidya, J., Xiong, H., Adam, N.: Semantics-based automated service discovery. IEEE Transactions on Services Computing **5**(2), 260–275 (2012). doi:10.1109/TSC.2011.19
178. Papazoglou, M.P.: Web Services: Principles and Technology, 1^{st} edn. Pearson Education Limited (2008)
179. Parissidis, G., Karaliopoulos, M., Baumann, R., Spyropoulos, T., Plattner, B.: Routing metrics for wireless mesh networks. In: S. Misra, S. Misra, I. Woungang (eds.) Guide to Wireless Mesh Networks, Computer Communications and Networks, pp. 199–230. Springer London (2009). doi:10.1007/978-1-84800-909-7_8
180. Park, K.L.P., Yoon, U.H., Kim, S.D.: Personalized service discovery in ubiquitous computing environments. IEEE Pervasive Computing **8**(1), 58–65 (2009). doi:10.1109/MPRV.2009.12
181. Park, P., Di Marco, P., Fischione, C., Johansson, K.: Delay distribution analysis of wireless personal area networks. In: 51^{st} Annual Conference on Decision and Control (CDC), pp. 5864–5869. IEEE Computer Society (2012). doi:10.1109/CDC.2012.6426060
182. Pauls, K., Hall, R.S.: Eureka - a resource discovery service for component deployment. In: Component Deployment, pp. 159–174 (2004)
183. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental) (2003)
184. Peter Hjärtquist, A.L.: Multimetric olsr and ett. In: 5^{th} OLSR Interop and Workshop. Vienna, Austria (2009)
185. Plummer, D.: Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. RFC 826 (Internet Standard) (1982). Updated by RFCs 5227, 5494
186. Postel, J.: User Datagram Protocol. RFC 768 (Internet Standard) (1980)
187. Postel, J.: Internet protocol. RFC 791 (Internet Standard) (1981). Updated by RFCs 1349, 2474, 6864

188. Project, X.: The Xen® hypervisor. Webpage (2013). URL http://www.xenproject.org
189. Proto, F.S., Pisa, C.: The OLSR mDNS extension for service discovery. In: 6^{th} Conference on Sensor, Mesh and Ad Hoc Communications and Networks, Workshops (SECON), pp. 1–3. IEEE Computer Society (2009). doi:10.1109/SAHCNW.2009.5172965
190. Puccinelli, D., Gnawali, O., Yoon, S., Santini, S., Colesanti, U., Giordano, S., Guibas, L.: The impact of network topology on collection performance. In: P. Marrón, K. Whitehouse (eds.) Wireless Sensor Networks, *Lecture Notes in Computer Science*, vol. 6567, pp. 17–32. Springer Berlin Heidelberg (2011). doi:10.1007/978-3-642-19186-2_2
191. Python Software Foundation: The python programming language. Webpage (2014). URL https://www.python.org
192. Quereilhac, A., Lacage, M., Freire, C., Turletti, T., Dabbous, W.: NEPI: An integration framework for network experimentation. In: 19^{th} International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 1–5. IEEE Computer Society (2011)
193. Raptis, P., Vitsas, V., Paparrizos, K.: Packet delay metrics for IEEE 802.11 distributed coordination function. Mobile Networks and Applications **14**(6), 772–781 (2009). doi:10.1007/s11036-008-0124-7
194. Reinecke, P., Wolter, K.: Adaptivity metric and performance for restart strategies in web services reliable messaging. In: 7^{th} International Workshop on Software and Performance (WOSP), pp. 201–212. ACM, New York, NY, USA (2008). doi:10.1145/1383559.1383585
195. Ren, Z., Jin, B., Li, J.: A new web application development methodology: Web service composition. In: WES, pp. 134–145 (2003)
196. Rezende, R.: User-perceived instantaneous service availability evaluation. Master thesis, Università della Svizzera Italiana (USI), Lugano, Switzerland (2013)
197. Rheinberger, H.J.: Experiment, Differenz, Schrift: zur Geschichte epistemischer Dinge. Basilisken-Presse, Verlag Natur and Text (1992)
198. Richard, G.G.: Service advertisement and discovery: Enabling universal device cooperation. IEEE Internet Computing **4**(5), 18–26 (2000). doi:10.1109/4236.877482
199. Rodgers, J.L., Nicewander, W.A.: Thirteen ways to look at the correlation coefficient. The American Statistician **42**(1), 59–66 (1988). doi:10.1080/00031305.1988.10475524
200. Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P.: Design and analysis of computer experiments. Statistical Science **4**(4), 409–423 (1989)
201. Sahner, R.A., Trivedi, K.S.: Performance and reliability analysis using directed acyclic graphs. IEEE Transactions on Software Engineering **SE-13**(10), 1105–1114 (1987). doi:10.1109/TSE.1987.232852
202. Sahner, R.A., Trivedi, K.S., Puliafito, A.: Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package. Kluwer Academic Publishers, Norwell, MA, USA (1996)
203. Salehi, P., Hamoud-Lhadj, A., Colombo, P., Khendek, F., Toeroe, M.: A UML-based domain specific modeling language for the availability management framework. In: 12th International Symposium on High-Assurance Systems Engineering (HASE), pp. 35–44. IEEE Computer Society (2010). doi:10.1109/HASE.2010.21
204. Salfner, F., Lenk, M., Malek, M.: A survey of online failure prediction methods. ACM Computing Surveys **42**(3), 10:1–10:42 (2010). doi:10.1145/1670679.1670680
205. Salfner, F., Malek, M.: Architecting dependable systems with proactive fault management. In: A. Casimiro, R. de Lemos, C. Gacek (eds.) Architecting Dependable Systems VII, *Lecture Notes in Computer Science*, vol. 6420, pp. 171–200. Springer Berlin Heidelberg (2010). doi:10.1007/978-3-642-17245-8_8
206. Sasson, Y., Cavin, D., Schiper, A.: Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In: Wireless Communications and Networking (WCNC), vol. 2, pp. 1124–1130 (2003). doi:10.1109/WCNC.2003.1200529
207. Selic, B.: The pragmatics of model-driven development. IEEE Software **20**(5), 19–25 (2003). doi:10.1109/MS.2003.1231146
208. Seltman, H.J.: Experimental Design and Analysis. Howard Seltman (2013)

209. Seno, S.A.H., Budiarto, R., Wan, T.C.: A routing layer-based hierarchical service advertisement and discovery for MANETs. Ad Hoc Networks **9**(3), 355–367 (2011). doi:10.1016/j.adhoc.2010.07.003
210. Service Availability Forum: Application Interface Specification, 6.1 edn. (2011). URL http://www.saforum.org
211. Shao, L., Zhao, J., Xie, T., Zhang, L., Xie, B., Mei, H.: User-perceived service availability: A metric and an estimation approach. In: International Conference on Web Services (ICWS), pp. 647–654. IEEE Computer Society (2009). doi:10.1109/ICWS.2009.58
212. Siddiqui, F., Zeadally, S., Kacem, T., Fowler, S.: Zero configuration networking: Implementation, performance, and security. Computers & Electrical Engineering **38**(5), 1129 – 1145 (2012). doi:10.1016/j.compeleceng.2012.02.011. Special issue on Recent Advances in Security and Privacy in Distributed Communications and Image processing
213. Stantchev, V., Malek, M.: Addressing dependability throughout the SOA life cycle. IEEE Transactions on Services Computing **4**(2), 85–95 (2011). doi:10.1109/TSC.2010.15
214. Sterbenz, J.P.G., Hutchison, D., Çetinkaya, E.K., Jabbar, A., Rohrer, J.P., Schöller, M., Smith, P.: Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. Computer Networks **54**(8), 1245–1265 (2010). doi:10.1016/j.comnet.2010.03.005. Resilient and Survivable networks
215. Stolikj, M., Verhoeven, R., Cuijpers, P.J.L., Lukkien, J.J.: Proxy support for service discovery using mDNS/DNS-SD in low power networks. In: 15th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1–6. IEEE Computer Society, Sydney, Australia (2014). doi:10.1109/WoWMoM.2014.6918925
216. Tanaka, H., Fan, L.T., Lai, F.S., Toguchi, K.: Fault-tree analysis by fuzzy probability. IEEE Transactions on Reliability **R-32**(5), 453–457 (1983). doi:10.1109/TR.1983.5221727
217. The Eclipse Foundation: Eclipse development environment. Software (2013). URL http://www.eclipse.org
218. The Eclipse Foundation: Papyrus UML modeling tool. Software (2013). URL http://www.eclipse.org/modeling/mdt/papyrus
219. The Eclipse Foundation: VIATRA2, VIsual Automated model TRAnsformations. Software (2013). URL http://www.eclipse.org/gmt/VIATRA2
220. Thompson, M.S.: Service discovery in pervasive computing environments. Ph.D. thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA (2006)
221. Thompson, M.S., Midkiff, S.F.: Service description and dissemination for service discovery in pervasive computing environments. International Journal of Ad Hoc and Ubiquitous Computing **12**(4), 193–204 (2013). doi:10.1504/IJAHUC.2013.052862
222. Tichy, W.F.: Should computer scientists experiment more? Computer **31**(5), 32–40 (1998). doi:10.1109/2.675631
223. Tolle, G.: A UDP/IP adaptation of the ZigBee application protocol. Internet Draft (2008)
224. Trivedi, K.S.: SHARPE (symbolic hierarchical automated reliability and performance evaluator). Software (2010). URL http://sharpe.pratt.duke.edu
225. Trivedi, K.S., Kim, D.S.: System availability assessment using stochastic models. Applied Stochastic Models in Business and Industry **29**(2), 94–109 (2013). doi:10.1002/asmb.951. Special Issue: Advanced Reliability and Maintenance Modeling (APARM 2010)
226. Tröger, P., Becker, F., Salfner, F.: FuzzTrees – failure analysis with uncertainties. In: 20^{rh} Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 263–272. IEEE Computer Society, Singapore (2014). doi:10.1109/PRDC.2013.48
227. Tromp, E.: Basic Multicast Forwarding Plugin for Olsrd, 1.7.0 edn. (2010)
228. Tsai, W.T.: Service-oriented system engineering: a new paradigm. In: IEEE International Workshop on Service-Oriented System Engineering (SOSE), pp. 3–6. IEEE Computer Society (2005). doi:10.1109/SOSE.2005.34
229. Tseng, Y.C., Ni, S.Y., Chen, Y.S., Sheu, J.P.: The broadcast storm problem in a mobile ad hoc network. Wireless Networks **8**(2), 153–167 (2002). doi:10.1023/A:1013763825347
230. Ullrich, F.: Automatisierte verfügbarkeitsmodellierung von it-diensten. Master thesis, Humboldt-Universität zu Berlin (2010)

231. UPnP Forum: Universal Plug and Play Device Architecture 1.1 (2008)
232. Vanthournout, K., Deconinck, G., Belmans, R.: A taxonomy for resource discovery. Personal Ubiquitous Computing **9**(2), 81–89 (2005). doi:10.1007/s00779-004-0312-9
233. Varró, D., Balogh, A.: The model transformation language of the VIATRA2 framework. Science of Computer Programming **68**(3), 187–207 (2007). doi:10.1016/j.scico.2007.05.004
234. Veizades, J., Guttman, E., Perkins, C., Kaplan, S.: Service location protocol. RFC 2165 (Proposed Standard) (1997). Updated by RFCs 2608, 2609
235. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault tree handbook. Tech. Rep. NUREG-0492, U.S. Nuclear Regulatory Commission, Washington, DC, USA (1981)
236. Villaverde, B.C., De Paz Alberola, R., Jara, A., Fedor, S., Das, S.K., Pesch, D.: Service discovery protocols for constrained machine-to-machine communications. IEEE Communications Surveys Tutorials **16**(1), 41–60 (2014). doi:10.1109/SURV.2013.102213.00229
237. Wackerly, D.D., Mendenhall, W.I., Scheaffer, R.L.: Mathematical Statistics with Applications, 7 edn. Thomson Learning, Inc., Belmont, CA, USA (2008)
238. Wang, D., Trivedi, K.S.: Modeling user-perceived service availability. In: M. Malek, E. Nett, N. Suri (eds.) Service Availability, *Lecture Notes in Computer Science*, vol. 3694, pp. 107–122. Springer Berlin Heidelberg (2005). doi:10.1007/11560333_10
239. Wang, R.C., Chang, Y.C., Chang, R.S.: A semantic service discovery approach for ubiquitous computing. Journal of Intelligent Manufacturing **20**(3), 327–335 (2009). doi:10.1007/s10845-008-0213-2
240. Wang, Y.: Automating experimentation with distributed systems using generative techniques. Ph.D. thesis, University of Colorado, Boulder, CO, USA (2006)
241. Wang, Y., Carzaniga, A., Wolf, A.L.: Four enhancements to automated distributed system experimentation methods. In: 30^{th} International Conference on Software Engineering (ICSE), pp. 491–500. ACM (2008). doi:10.1145/1368088.1368155
242. Wanja, S.: Experimentation environment for service discovery. Master's thesis, Humboldt-Universität zu Berlin, Berlin, Germany (2012)
243. Williams, A.: Requirements for automatic configuration of IP hosts. Internet Draft (2002). URL http://tools.ietf.org/html/draft-ietf-zeroconf-reqts-12
244. Winer, D.: Extensible markup language remote procedure calls. Specification (2003). URL http://xmlrpc.scripting.com/spec.html
245. Xie, W., Sun, H., Cao, Y., Trivedi, K.S.: Modeling of user perceived webserver availability. In: International Conference on Communications (ICC), vol. 3, pp. 1796–1800. IEEE Computer Society (2003). doi:10.1109/ICC.2003.1203909
246. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Computer Networks **52**(12), 2292–2330 (2008). doi:10.1016/j.comnet.2008.04.002
247. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: 12^{th} International Conference on World Wide Web (WWW), pp. 411–421. ACM, New York, NY, USA (2003). doi:10.1145/775152.775211
248. Zheng, Y.R., Xiao, C.: Simulation models with correct statistical properties for rayleigh fading channels. IEEE Transactions on Communications **51**(6), 920–928 (2003). doi:10.1109/TCOMM.2003.813259
249. Zhu, F., Mutka, M., Ni, L.: Classification of service discovery in pervasive computing environments. Tech. Rep. MSU-CSE-02-24, Michigan State University, East Lansing, MI, USA (2002)
250. Zhu, F., W. Mutka, M., M. Ni, L.: Service discovery in pervasive computing environments. IEEE Pervasive Computing **4**, 81–90 (2005)
251. Zhu, F., Zhu, W., Mutka, M.W., Ni, L.M.: Private and secure service discovery via progressive and probabilistic exposure. IEEE Transactions on Parallel and Distributed Systems **18**(11), 1565–1577 (2007). doi:10.1109/TPDS.2007.1075
252. Zhu, T., Zhong, Z., He, T., Zhang, Z.L.: Exploring link correlation for efficient flooding in wireless sensor networks. In: 7^{th} USENIX Conference on Networked Systems Design and Implementation (NSDI), pp. 4–4. USENIX Association, Berkeley, CA, USA (2010)

253. Zseby, T., Henke, C., Kleis, M.: Packet tracking in planetlab europe – a use case. In: T. Magedanz, A. Gavras, N. Thanh, J. Chase (eds.) Testbeds and Research Infrastructures. Development of Networks and Communities, *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 46, pp. 265–274. Springer Berlin Heidelberg (2011). doi:10.1007/978-3-642-17851-1_22

Appendix A
XML Schema of Service Discovery Experiment Process

In the listing below is a sample XML schema for service discovery as an experiment process. One can see that few adjustments to the `actions` parts allow to use a similar schema for arbitrary network protocols that are to be examined under the influence of the same factors to be varied. See also Chapter 3.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://sd_experiments"
        targetNamespace="http://sd_experiments"
        elementFormDefault="qualified">

<xs:element name="sd_experiment_description" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="meta" type="xs:anyType" />
      <xs:element name="abstractnodes" type="abstractnodes" />
      <xs:element name="factorlist" type="factorlist" />
      <xs:element name="processes" type="processes" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="abstractnodes">
  <xs:sequence>
    <xs:element name="abstractnode" type="abstractnode" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="abstractnode">
  <xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="factorlist" >
  <xs:sequence>
    <xs:element name="factor" type="factor" maxOccurs="unbounded"/>
    <xs:element name="replicationfactor" type="replicationfactor" maxOccurs="1"
    ↪ />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="factor">
  <xs:sequence>
    <xs:element name="levels" maxOccurs="1" />
    <xs:element name="description" maxOccurs="1" type="xs:anyType"/>
```

```
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" />
  <xs:attribute name="type" type="xs:string" />
  <xs:attribute name="usage" type="xs:string" />
</xs:complexType>

<xs:complexType name="replicationfactor">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="id" type="xs:string" />
      <xs:attribute name="type" type="xs:string" />
      <xs:attribute name="usage" type="xs:string" />
    </xs:extension>
   </xs:simpleContent>
</xs:complexType>

<xs:complexType name="levels">
  <xs:sequence>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="processes">
  <xs:sequence>
    <xs:element name="node_processes" type="node_processes" />
    <xs:element name="env_process" type="env_process" />
  </xs:sequence>
  <xs:attribute name="max_run_time" type="xs:int" />
</xs:complexType>

<xs:complexType name="node_processes">
  <xs:sequence>
    <xs:element name="process_parameters" minOccurs="1" maxOccurs="1"
    ↪ type="process_parameters"/>
    <xs:element name="actor" minOccurs="1" maxOccurs="unbounded" type="actor"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="process_parameters" >
  <xs:sequence>
    <xs:element name="actor_node_map" type="actor_node_map" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="actor_node_map">
  <xs:choice>
    <xs:element name="factorref" type="factorref"/>
    <xs:sequence>
      <xs:element name="actormap" type="actormap" minOccurs="1"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>

<xs:complexType name="actormap" >
  <xs:sequence>
    <xs:element name="instance" minOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" />
</xs:complexType>

<xs:complexType name="instance">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="id" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```
<xs:complexType name="description">
  <xs:simpleContent>
    <xs:extension base="xs:string"></xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="actor">
  <xs:sequence>
    <xs:element name="sd_actions" maxOccurs="1" />
    <xs:element name="manipulation_actions" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" />
  <xs:attribute name="name" type="xs:string" />
</xs:complexType>

<xs:complexType name="factorref" >
  <xs:attribute name="id" type="xs:string" />
</xs:complexType>

<xs:complexType name="sd_actions" />
<xs:complexType name="manipulation_actions" />
<xs:complexType name="env_process">
  <xs:sequence>
    <xs:element name="description" type="description" />
    <xs:element name="env_actions" type="env_actions" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="env_actions">
  <xs:sequence>
    <xs:element type="flow_control_actions" minOccurs="0" maxOccurs="unbounded"
    ↪ />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="flow_control_actions" />
  <xs:choice></xs:choice>
</xs:complexType>

<!-- ACTIONS //-->
<xs:complexType name="traffic">
  <xs:sequence>
    <xs:element name="pairs" />
    <xs:element name="bw" />
    <xs:element name="choice" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="wait_for_event">
  <xs:sequence></xs:sequence>
</xs:complexType>

</xs:schema>
```

Appendix B
ExCovery Process Description

This listing contains experiment and environment process descriptions as used for the experiments conducted in this work. See also Chapters 6 and 3.

```
<!-- ... -->
<processes max_run_time="120">
 <node_processes name="">
  <process_parameters>
   <actor_node_map>
    <factorref id="fact_nodes"/>
   </actor_node_map>
  </process_parameters>

  <actor id="responder" name="publisher">
   <sd_actions>
    <sd_init />
    <sd_publish />
    <wait_for_event>
     <event_dependency>done</event_dependency>
    </wait_for_event>
    <sd_unpublish />
    <sd_exit />
   </sd_actions>
  </actor>

  <actor id="requester" name="searcher">
   <sd_actions>
    <wait_for_event>
     <from_dependency>
      <node actor="responder" instance="all"/>
     </from_dependency>
     <event_dependency>sd_start_publish</event_dependency>
    </wait_for_event>
    <wait_for_event>
     <event_dependency>ready_to_init</event_dependency>
    </wait_for_event>
    <sd_init />
    <wait_for_event>
     <event_dependency>ready_to_search</event_dependency>
    </wait_for_event>
    <wait_marker />

    <sd_start_search />
    <wait_for_event>
     <from_dependency>
      <node actor="requester" instance="all"/>
```

```
      </from_dependency>
      <event_dependency>sd_service_add</event_dependency>
      <param_dependency>
        <node actor="responder" instance="all" />
      </param_dependency>
      <timeout>30</timeout>
    </wait_for_event>
    <event_flag><value>done</value></event_flag>

    <sd_stop_search />
    <sd_exit />
  </sd_actions>
 </actor>
</node_processes>

<env_process>
 <description>The sd process is treated with a varying network
 ↪ traffic</description>
 <env_actions>
   <wait_for_event>
     <from_dependency>
        <node actor="responder" instance="all"/>
     </from_dependency>
     <event_dependency>sd_start_publish</event_dependency>
   </wait_for_event>
   <wait_marker />

   <env_start_drop_sd />
   <wait_time>
     <value>5</value>
   </wait_time>
   <event_flag>
     <value>ready_to_init</value>
   </event_flag>

   <env_traffic_start>
     <bw>
       <factorref id="fact_bw" />
     </bw>
     <choice>0</choice>
     <random_switch_amount>1</random_switch_amount>
     <random_switch_seed>
       <factorref id="fact_replication_id" />
     </random_switch_seed>
     <!-- this causes the randomization in replications -->
     <random_pairs>
       <factorref id="fact_pairs" />
     </random_pairs>
     <random_seed>
       <factorref id="fact_pairs"/>
     </random_seed>
     <!-- this causes the randomization in repeatitions of the same
     ↪ pairs-factor to be the same
     this means the randomly selected nodes -->
   </env_traffic_start>

   <wait_for_event>
     <from_dependency>
       <node actor="requester" instance="all"/>
     </from_dependency>
     <event_dependency>sd_init_done</event_dependency>
   </wait_for_event>
   <wait_time>
     <value>1</value>
   </wait_time>
   <env_stop_drop_sd />
```

```
      <event_flag>
        <value>ready_to_search</value>
      </event_flag>

      <wait_for_event>
        <event_dependency>done</event_dependency>
      </wait_for_event>
      <env_traffic_stop />
    </env_actions>
  </env_process>
</processes>
<!-- ... -->
```

Index