Service Migration in Dynamic and Resource-Constrained Networks

Nickelsen, Anders

Publication date:
2011

Document Version
Early version, also known as pre-print

SERVICE MIGRATION IN DYNAMIC AND RESOURCE-CONSTRAINED NETWORKS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF
ELECTRONIC SYSTEMS
OF
AALBORG UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Anders Nickelsen
January 14, 2011

Supervisor:
Assoc. Prof. Hans-Peter Schwefel, Aalborg University, Aalborg, Denmark
Ass. Prof. Rasmus L. Olsen, Aalborg University, Aalborg, Denmark

The assessment committee:
Dr. Ernö Kovacs, NEC, Heidelberg, Germany
Assoc. Prof. Andrea Bondavalli, Università degli Studi di Firenze, Florence, Italy
Assoc. Prof. Tatiana K. Madsen, Aalborg University, Aalborg, Denmark (Chairman)

Moderator:
Assoc. Prof. Lars Bo Larsen, Aalborg University, Aalborg, Denmark

Date of defense: March 8, 2011

Report: R02-2011
ISSN: 0908-1224
ISBN: 978-87-92328-54-0

Dedicated to
my family for their unconditional support.

Abstract

Migration is the process of moving an active application from one device to another. The objective of migration is to improve the user's experience of the application. Migration is becoming increasingly useful as users have access to many, different computing devices (such as workstations, laptops and mobile devices) that are able run the same applications, such as access to e-mail, edit documents, browse the Internet or stream multi-media content. However, application performance and migration performance depends on the state of the underlying network. Network dynamics and resource-constraints may therefore influence the user's experience of a migrating application.

Automatic triggering of migration aims at improving the user experience quality without requiring the attention of the user when the network changes. In order to automatically trigger automatic successfully, thorough understanding is needed of the network behavior, the triggering function and the migration orchestration procedures. This thesis presents a study of the core functions to support automatic application migration. The migration functions are presented as a part of an overall migration platform developed in the OPEN research project.

First, properties of network state estimation methods are studied. Ubiquitous networks are dominated by mobility, varying traffic conditions and communication paths spanning wired and wireless networks. To provide reliable service in such dynamic networks estimation is vital to trigger migration. The thesis presents network estimation methods based on hidden Markov models and Bayesian Networks (BN) and evaluates their performance in terms of estimation accuracy and response time to network changes.

Secondly, a migration platform is developed to automatically trigger migration based on network state estimates. The complexity of decision policy generation is analyzed and the performance impact of taking future decisions into account is studied. An instantaneous generation approach is compared to a model-based Markov Decision Process (MDP)-based approach. Both generation approaches aim at optimizing the trade-off between a high average user experience quality and a low penalty of failed and delayed migrations. The impact of in-accurate network state estimation on the performance is evaluated.

Thirdly, the design and evaluation of a lean migration orchestration protocol for resource-constrained links is presented. The protocol is evaluated in scenarios, resource-constrained ad-hoc links between devices are used. Performing migration over such links is challenging due to throughput limitations and the potential short time-windows during which the communication is possible. The RFID-based technology Near Field communication (NFC) is used in a case study where the performance of the orchestration protocol is evaluated.

Fourthly, an analysis is presented of the complexities of having dynamic elements in the migration platform, such as devices or network joining and leaving. We investigate an extended decision framework where the decision concerns both where to migrate and how to migrate based on available networks. Including many dimensions in the decision policies increase the complexity of the policy generation, and we investigate methods to reduce the complexity by means of approximation.

In summary, this thesis presents solutions to support functions in a platform for automatic migration in scenarios with dynamic and/or resource-constrained networks. It is shown how model-based approaches to both network estimation and triggering can improve the user's experience of a migratory application. Moreover, the lean migration protocol proved useful in scenarios with resource-constrained networks which broadens the range of cases where migration can be successfully applied. Finally, the steps toward modeling migration dynamics demonstrated the need for approximate methods, which were shown to be useful for automatic triggering.

Acknowledgements

For their work and support I would like to thank first my supervisors Hans-Peter Schwefel and Rasmus L. Olsen. Hans-Peter in particular for the always encouraging and challenging discussions - both as a PhD student and as a master student. The opportunity to work together with Hans-Peter beyond my master study was the main reason that I started my PhD study and the main reason I was able to complete it. I thank Rasmus for always being ready to support me the countless times I have been knocking at his door with problems to discuss, and in particular for always meeting me and my problems with a smile. Rasmus has helped me get through many valleys of frustration during my study.

I would also like to thank all of my colleagues from the ICT project OPEN and the IST project HIDENETS with whom I have had really fruitful discussions and received feedback from as well. In particular I would like to thank Ernö Kovacs and the people in the "Context-aware Services" research group for your hospitality at NEC, Heidelberg in Germany during my stay in the summer 2009. A special thanks goes to Miquel Martin for the always interesting and detailed discussions on the things that matter.

Also I would like to thank my family and friends, for without the support from you, I would not have been able to withstand the work load.

Finally, a special thanks also goes to all my colleagues at the Networking and Security section throughout the years at Aalborg University for making work life so much fun and filled with new, exciting perspectives on life and research.

Contents

1

Introduction and problem description

The purpose of this chapter is to provide the reader with a basic understanding of the problem domain, overview the contribution of the thesis and to describe the structure of the thesis. Initially an introduction to the problem domain of service migration in dynamic and resource-constrained networks is given by examples. The benefits of service migration between devices are described, followed by an overview of the necessary steps in the migration process. From this overview, the challenges of performing migration in dynamic and resource-constrained networks are described. The challenges lead to the problem statement and the scope of the thesis, along with an overview of the thesis.

1.1 Motivation

An important aspect of ubiquitous environments is to provide users with the possibility to freely move about and continue interaction with applications through a variety of interactive devices (including smart phones, PDAs, tablets, desktop computers, digital television sets, public displays, intelligent watches, etc.). In recent years, interactive devices are transforming from specific single function devices to general execution platforms. This generalization allows for the application clients to run on any available device, which in turn enables the user to interact with the service through the preferred terminal and user interface. Thus, the interaction with a specific service is no longer bound to a specific device. However, in such environments one big potential source of frustration is that people have to start their session over again from the beginning at each interaction device change. Moreover, the different devices may give the user different experiences with the application, and the user may have certain preferences toward doing specific tasks on specific devices. In addition, these different devices have increased access to several different networking interfaces, both cellular long- and medium-range technologies and ad-hoc short-range technologies. Choosing which networks to use for specific applications in specific situations is not a trivial choice to make for the user.

Service migration can overcome these limitations and support continuous interaction with services and applications. Migration enables the interactive applications to follow users and adapt to *context changes* while preserving their state. In particular, the knowledge of context, which is any information that can be used to characterize a situation of an entity [1], is important to ensure that applications are migrated at the right time and place. In addition to moving applications between devices, the migration process *adapts the application* to utilize the available resources on the new device or in the new network in order to provide an optimal user experience of a *continuous* application. To summarize:

$$\text{Migration} = \text{Device Change} + \text{Adaptation} + \text{Continuity}.$$

1.2 General migration scenario

As an example, Figure 1.1 illustrates a migratory video-streaming service that is initially run on a mobile device with a small screen and a wireless Internet connection. The video-streaming client is migrated to a large, high resolution display with a fixed Internet connection.

Figure 1.1: General migration scenario.

A migration can be *triggered* in two ways. Manual triggers are generated by the user. In the example, the user selects the large display in a menu of available migration targets on the mobile device. Automatic triggers are generated by the migratory application based on changes in the context. Examples of such context changes are when the large display is discovered in the network, when the available network capacity degrades or if the mobile device runs low on battery.

A migration *preserves the state* of the application (current point in video playback or volume settings in the example), such that the application does not need to be restarted on the target device and the user can continue the work flow after migration.

We consider client/server-based applications, as illustrated in Figure 1.2, where an application is running on the user's device (the client) connected to the remote service (the server) that is provided via the Internet. Examples of this application type are: mail service, document editing, web browsing, multimedia streaming, real-time communication (audio/video) and gaming. The migration moves and adapts the *client* between devices that are available to the user. It is as such only the client that is considered migratory in this work. In order to support existing services, the migration has to be transparent as seen from the remote server.

1.3 Migration platform

We propose a middleware with migration functions between the migratory application and the device. The purpose of the migration middleware is to

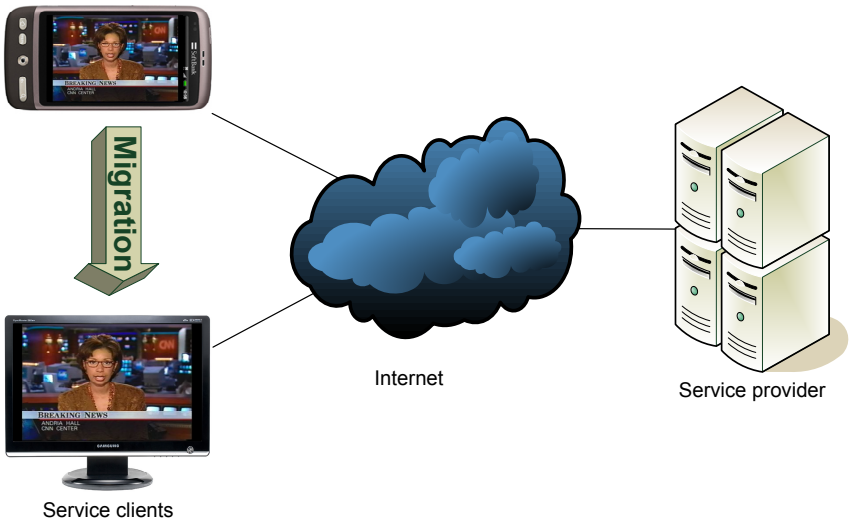

Figure 1.2: The migrating services considered in this work are provided through the Internet. The migration of the clients must be transparent to the provider.

encapsulate and unify functions that are common between migratory applications. This allows the application developer to focus on giving the user the best application experience and not focus on implementing migration functions in each application. The functions contained in the middleware have the following responsibilities.

- Collect context information
- Automatically trigger migration upon context changes
- Orchestrate the migration between devices, i.e. move the application state
- Adapt application to fit target device and new context
- Make migration transparent toward remote servers
- Ensure security in migration

1.4 Challenges of migration

The primary focus is on the performance of the decision problems included in an automatically triggered migration. For a migration to succeed, the following decision functions need to be included.

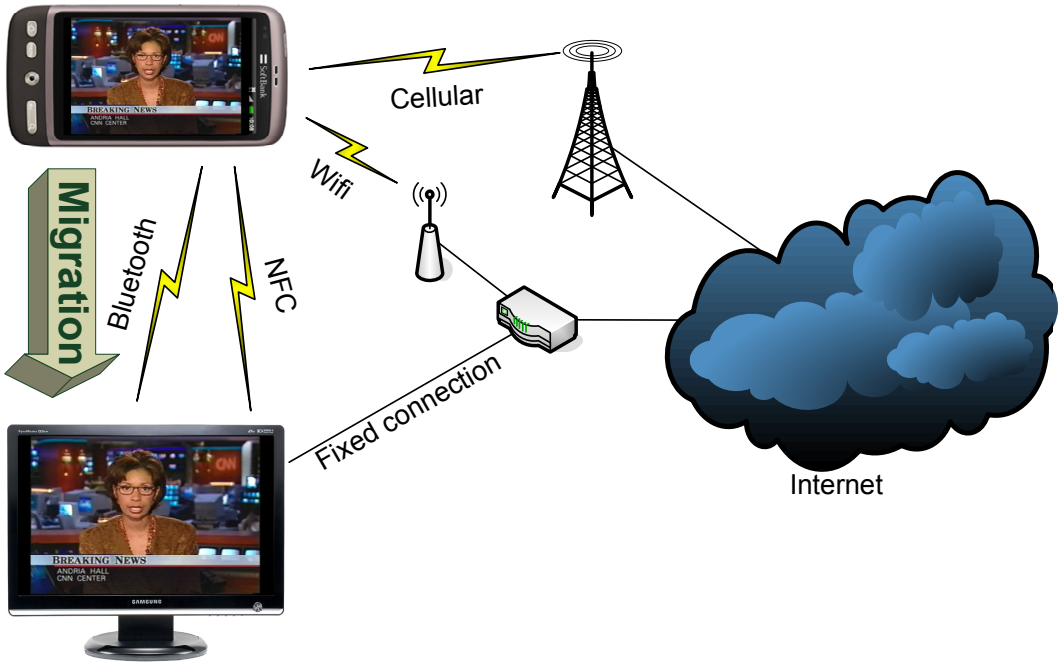

Figure 1.3: The devices have several available methods to connect to the Internet. The methods have different properties in terms of resources and behavior.

- when to migrate (context change detection)
- where to migrate (target device selection)
- how to migrate (orchestration method selection)
- what to migrate (application component selection)
- how to adapt (application adaptation)

The two last points are included for generality only and are not considered in this work, as we define applications as atomic entities. If applications are constructed by components, a choice can be made where to migrate individual application components to multiple devices.

Common to all of the three considered decision problems is that the sets of choices (context states, available target devices, available orchestration methods) become very large in scenarios with several users, devices and migratory applications. Furthermore, the availability of a choice may be dynamic, for instance, caused by user mobility or device resource-constraints. This may potentially cause the models that form the basis for the decision problems to be intractable by the middleware. Hence, for each decision problem, it must

be considered how the decision problem complexity can be kept low, which is a focus of this work.

1.4.1 When to migrate

To decide *when to migrate*, context information needs to be collected to estimate the state of the network, to detect when it changes. Many parameters may influence the performance of the application, however, in this work, we focus on the time dynamics of the network connections between user devices and between application client and server. The end-to-end connection between client and remote server may span several different network technologies with different properties. The remote server is provided through the Internet. The Internet access may be provided via both wired and wireless links. This is illustrated in Figure 1.3. Wireless network connections are dynamic by nature, since they are based on shared medium access. User mobility can cause instability of the wireless links and other wireless users and noise can cause the links to be unreliable, resulting in dropped packets. In addition, devices may be directly interconnected via ad-hoc short- to medium range wireless technologies. Such links may be more stable and reliable, since they are used by fewer users simultaneously, but they also typically provide less capacity and shorter connectivity windows, since connectivity is based on physical closeness of devices.

The state of the network is not always directly observable, so estimation methods need to be applied in order to obtain a representation of the network state. Due to dynamics, estimation will be inaccurate and take time, which may impact the other decisions that depend on the network state.

1.4.2 Where to migrate to

To decide *where to migrate to*, the decision mechanism needs to understand what device give the best user experience of the application. Therefore, a model is needed of the quality of the user experience delivered by an application on a specific device in a certain state of the network. This model is based on information about application requirements, device capabilities and application behavior within a dynamic network. A part of this model is called a *configuration*. The configuration specifies on which device the application runs. When a specific configuration is active, a certain user experience is delivered, depending on the network state. The model of the user experience

quality specifies how the user experience quality behaves in the dynamic network state in different configuration. When the network state changes it is evaluated through the model if another configuration delivers a better user experience. If so, a migration to the new configuration must be triggered.

1.4.3 How to migrate

To decide *how to migrate*, a model is needed of the migration process and its performance in dynamic and resource-constrained networks. Since the migration is performed over the similar types of networks as the application is running, network dynamics may also affect the performance of the migration process. As seen in Figure 1.3, multiple networks may be available for the migration, so the model is used to decide which migration method performs the best in the current state of the network. Furthermore, some ad-hoc network links may only have connectivity for a limited time. This is the case, for instance, when physical closeness is required to obtain a link. In this case, the chosen orchestration method should be optimized to succeed within the allotted window of time.

1.5 Problem statement and objectives

Based on the previously described challenges of performing automatic migration in dynamic and resource-constrained networks, we address the following problem in this dissertation:

How to design estimation, decision and orchestration functions in migration middleware to deliver maximum application user experience quality in dynamic and resource-constrained networks?

As the networks between the devices have an important role both in the application performance and in the migration performance, we focus on the set of problems that are related to dynamics in the network state. The following sub-problems are addressed in the following chapters of this dissertation.

- How to make accurate network state estimation in migration scenarios?
- How to collect network state information at decision enforcement points?
- How to model user experience quality and impact of network dynamics?

- How to choose between multiple configurations and orchestration methods with inaccurate networks state estimation?
- How to orchestrate migration when network is resource-constrained?

The objectives and scope of this dissertation are summarized below:

- Develop a migration platform that can perform automatic migration
- Analyze how the properties of dynamic networks affect network state estimation performance; develop specific estimation methods for the migration framework
- Analyze how the properties of dynamic networks affect migrations triggering policies and analyze how state estimation inaccuracy influences the triggering policies
- Develop a lean migration orchestration protocol for resource-constrained networks
- Analyze how dynamics of available migration target devices and orchestration methods cause by network dynamics influence triggering policies

Delimitations

To focus the scope of this dissertation, we make the following delimitations:

Central architecture A fundamental design decision in developing the migration platform is the architecture type which can be either centralized or distributed. We choose to only focus on the centralized architecture, as it demonstrates the required functionalities of a migration platform, without introducing additional complexities, which are in this scope unnecessary.

Security Secure migration is a vital part of the migration concept, and functions must be included in the framework to solve issues such as identification, authorization, trust management and confidentiality, etc. However, as the performance of such security functions is not directly influenced by network dynamics, they are considered out of our scope.

Mobility management Since the users may choose to migration applications between networks, both vertically and horizontally, mobility solutions must be included in the migration platform. We do not focus on

the mobility solutions, as they have been addressed in other publications, which are referenced as solutions in the mobility management design.

User experience modeling The performance of the migration functions is measured by the average quality of the user experience when using the migratory application. In this work, the quality is defined on an abstract scale, but can be compared to subjective quality scales such as the Mean Opinion Score (MOS) scale.

1.6 Summary of contributions

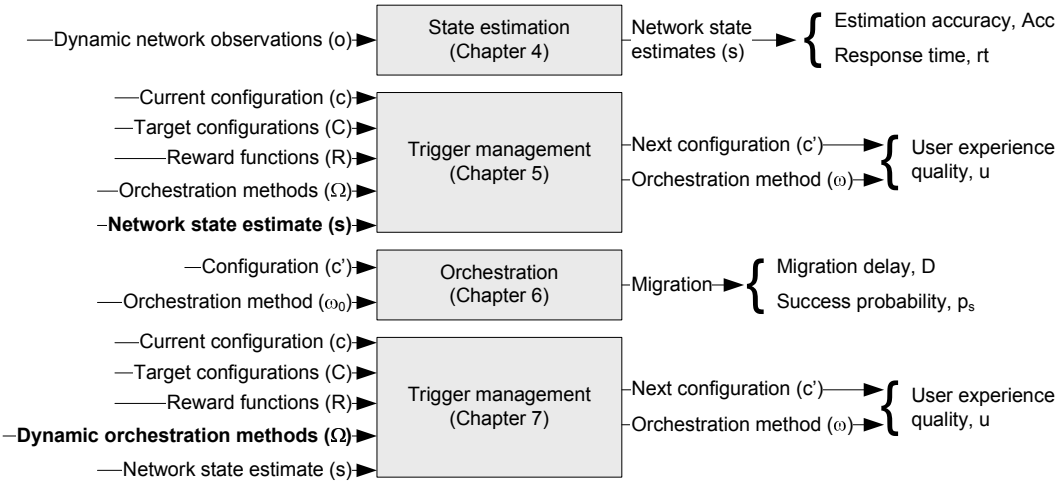

Figure 1.4: Overview of the migration functions and how they are described in this thesis.

Figure 1.4 depicts the organization of the chapters of this thesis based on the core functions of the migration platform. The contributions of each chapter are summarized below.

Chapter 3 Design of the migration platform as a middleware. The application-platform interface and user-platform interface are specified. The result is a flexible middleware platform that can handle migration of applications across devices and networks - both triggered manually and automatically based on collected context information.

Chapter 4 Analysis of methods for collecting network context information and design of a model-based state estimation component for the migration platform. In particular the challenges of estimating hidden states

of the network based on observable parameters are addressed. Several methods are studied in terms of estimation accuracy and response time, including simple threshold-based heuristics, Hidden Markov Models and Bayesian Networks. Based on a scenario with both hidden and observable network parameters, the trade-off is investigated between the accuracy and the response time of a state estimation function. We employ Bayesian Networks to model the mapping between the parameters

Chapter 5 Analysis of methods to trigger migration automatically to optimize the user experience in dynamic networks. We design a decision framework for the migration platform that can account for both the quality of the user's experience and the penalty of waiting during orchestration. Based on this framework, we investigate methods to generate optimal triggering policies that balance this trade-off, in order to make decisions in dynamically changing environments. The migration triggering process is modeled using a Markov Decision Process (MDP), to make decisions that maximize the long-term experience quality.

Chapter 6 We develop a lean orchestration protocol for an ad-hoc resource-limited network connection. An NFC connection is used as case study as is has a short connectivity window. Based on experimental performance measurements of an NFC link, we study how much application state information can be transferred in a connectivity window. The lean orchestration protocol is developed to make the most use out of the available time for orchestration. The evaluation shows how big amounts of state data can be transferred during orchestration in scenarios where only the NFC link is available.

Chapter 7 We investigate a scenario with several orchestration methods. The complexity of the decision problems increases rapidly as more orchestration methods are introduced. Furthermore, the orchestration methods have dynamic availability such that they may appear and disappear over time. We study how the interplay between trigger management and orchestration can handle this dynamic availability. We compare two policy generation approaches. The first one includes all dimensions of the scenario and can be generated offline before running the system so decisions can be enforced fast. The second one is calculated online and includes only the available orchestration methods at the time when the choice is made. We compare the trade-off between the accurate but complex offline world model and the less accurate but less complex online subset model. The results show that the subset model can be utilized

successfully when adapted to the dynamics of the orchestration method availability.

A list of names of the used symbols in the thesis is found on page 129. The contributions described in the chapters have been presented in the following publications:

- Anders Nickelsen, Rasmus L. Olsen, Hans-Peter Schwefel – "Model-based decision framework for autonomous application migration", in preparation
- Anders Nickelsen, Fabio Paterno, Agnese Grasselli, Kay-Uwe Schmidt, Miquel Martin, Bjørn Schindler, Francesca Mureddu – "OPEN – Open pervasive environments for interactive migratory services", in *Proc. of 12th International Conference on Information Integration and Web-based Applications & Services*, Paris, France, 2010
- Anders Nickelsen, Miquel Martin, Hans-Peter Schwefel – "Service migration protocol for NFC links", in *Proc. of Networked Services and Applications - Engineering, Control and Management (EUNICE)*, Trondheim, Norway, 2010
- Anders Nickelsen, Rasmus L. Olsen and Carmen Santoro – "Open pervasive environments for interactive migration services (OPEN)", in *Proc. of 1st Future Internet Symposium (FIS)*, Vienna, Austria, 2008
- Anders Nickelsen, Jesper Grønbæk, Thibault Renier and Hans-Peter Schwefel – "Probabilistic Network Fault-Diagnosis using Cross-Layer Observations", in *Proc. of 23rd Internation Conference on Advanced Information Networking and Applications (AINA)*, Bradford, UK, 2009

Topics in the following publications support the work on the contributions in this thesis

- Anders Nickelsen, Hans-Peter Schwefel – "Emulation of Wireless Multi-Hop Topologies with Online Mobility Simulation", in *International Journal On Advances in Telecommunications*, ISSN 1942-2601, vol. 2, no. 1, 2009, 27:36, http://www.iariajournals.org/telecommunications/
- Anders Nickelsen, Morten N. Jensen, Erling Matthiesen and Hans-Peter Schwefel – "Scalable emulation of dynamic multi-hop topologies", in *Proc. of 4th International Conference on Wireless and Mobile Communications (ICWMC)*, Athens, Greece, 2008

- Yaoda Liu, Hans-Peter Schwefel, Frank Y. Li, Anders Nickelsen – "Circuit Elimination based Link State Routing in Mobile Ad-hoc Networks", in *Proc. of 4th IEEE International Symposium on Wireless Communication Systems (ISWCS)*, Trondheim, Norway, 2007
- Rasmus Løvenstein Olsen, Anders Nickelsen, Jimmy Jessen Nielsen, Hans-Peter Schwefel, Martin Bauer – "Experimental analysis of the influence of context awareness on service discovery in PNs", in *Proc. of 15th IST Mobile & Wireless Communication Summit*, Mykonos, Greece, 2006

References

[1] A.K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

2

Background and state of the art

This chapter describes existing approaches to migration platforms. First existing solutions for service migration are described, ranging from traditional agent based systems to new service migration frameworks. Context information is a large part of autonomous migration and so the background of context management is studied as well. Finally, available methods to model stochastic parameters are described, as they are used for both estimation and decision making in this work.

2.1 Migratory applications and frameworks

2.1.1 Dynamic, adaptive and reconfigurable systems

Applications and systems may not always operate in the conditions they were designed to be in. Users may alter run-time conditions deliberately or the evolution of new technologies may supersede the evolution of the application or system. From this realization originates the concept of *dynamic*, *adaptive* and *reconfigurable* systems [14] [1]. In such a system, an application is composed of blocks of components, which can be reconfigured to suit new purposes or adapt to new conditions. They can adapt dynamically, with activation and deactivation of components. Still, they do not specifically target application state preservation, so the application may need to be re-initialized upon reconfiguration. The systems are based on CORBA [23] or OSGI [2], which are both service/component management framework. Each dynamic, adaptive system is self-contained, such that it can make the configurations itself. There is no common infrastructure, which on one hand is beneficial when deploying systems in new domain because they do not depend on infrastructure presence. On the other hand, none of the applications can take the others into account in the decision making, so any decisions made can at most be optimal for the application, which is not necessarily optimal for the user, which may have a couple of applications running, or an operator, which have many applications running. The dynamic, adaptive systems are context-aware, such that they can adapt to changes in the context of the user or the system. However, in most cases, the decision about which configuration to use next is taken either manually by the user or the developer [8], or heuristically by the system, by choosing to use the most recent set of components to compose the application [14].

2.1.2 Proxy-based UI/web adaptation

Migration of user interfaces is also an emerging research field. Based on similar motivations as this work, namely that the user has increased access to terminal, user interface migration solutions pursues this realization by utilizing the terminals as application-client-size view-ports to a server-side application. Migration between terminal then becomes a matter of adapting the view-port to the capabilities of the terminal [25] [19] [24]. Typically, the client-side application is hosted in a cross-platform browser, in order to support cross-platform

adaptability. The application itself is then hosted from a web-service. Adaptation is performed by intercepting the communication between server and client at a proxy, restructuring the elements of the web application according to some policy and relaying the result on to the client to be presented to the user. The restructuring policies are statically defined to match distinct terminal types and the restructuring process is performed by reverse engineering data objects within the application content (typically HTML). Connectivity-wise, this approach is attractive because it operates transparently as seen from the server-side, and requires no adaptation of existing services. Its limitation is the reverse engineering process, since the adaptation requires full access to the application-level content (the HTML) and complete understanding of the content (the ontology of HTML objects). Also, clearly, this method of user interface migration only applies to web applications written in HTML.

2.1.3 Mobile agents

Mobile agents are independent programs moving between devices and platform to achieve a predefined goal. [3]. There exists different types of mobile code; *remote evaluation*, *code on demand* and *mobile agents* [9]. The reference solutions is client-server, where a client component communicates with a server component, which has the *know-how* to reach the goal and the *resources*. Both the remote evaluation and the code on demand solutions employ two component; in remote evaluation, the client has the know-how and the server has the resources, which is vice versa in code on demand, where the client has the resources and the server the know-how. With mobile agents, only one component exists, which has know-how of the goal and is transmitted between resources, until the goal is met. In all cases, the component with the know-how moves between platforms of resources. With migration, the concept is opposite. In migration, the user is the ultimate resource, and the applications have to move to where the user is present. If the current terminal, through which the user is interacting with the application, is different from the previous, adaptation is required. Know-how about adaptation has to be present where it is performed, that is, either on the terminal, or on a dedicated adaptation/reconfiguration entity.

The authors of the seminal paper on code mobility [3] evaluated the impact of the paradigm 10 years later, and concluded that code did not move as much around as anticipated [4]. The main reason, they conclude, is that the paradigm was too complex to completely take over the role of client-server based communication. The cost of overcoming the complexity of developing,

managing mobile agents and keeping them secure did not prove worth the benefit of mobility.

Strong vs. weak agent migration In agent terminology, two categories of migration have been defined; *strong migration* and *weak migration* [7] [12]. Strong migration is defined as the procedure where everything is migrated between devices - in particular code and state. The entire code and state need to be migrated for the migration to be strong. Weak migration is defined as any migration that is not strong. Details of the definitions are illustrated in Figure 2.1. In this work, the migration is characterized as weak, as it is only the application state that is migrated between device between to instances of the application code.

Agent systems standards The Foundation for Intelligent Physical Agents (FIPA) is a body for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems [6]. FIPA specifications represent a collection of standards which are intended to promote the interoperation of heterogeneous agents and the services that they can represent. The specifications can be viewed in terms of different categories: agent communication, agent transport, agent management, abstract architecture and

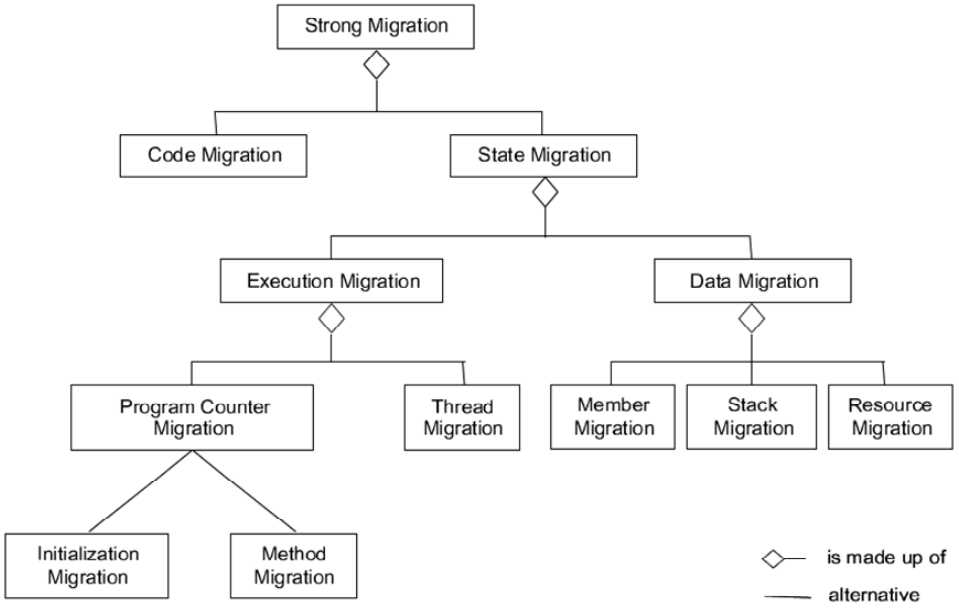

Figure 2.1: Definition of *strong migration* after [12]. *weak migration* is defined as anything not being strong.

applications. Of these categories, agent communication is the core category at the heart of the FIPA multi-agent system model.

Java Agent Development Environment (JADE) JADE is a development framework for distributed, multi-agent systems that complies to the FIPA standards [13]. It is a graphical user-interface (GUI) based integrated development environment (IDE) that creates software containers for agents. A set of containers is defined as a platform. The JADE framework contains agent support functions such as agent management and directory containers. The JADE framework is a candidate system for the deployment of the migration platform described in this work, since it contains solutions to many of the communication and management functions needed within the platform. However, it provides neither context management or decision functions necessary for our analysis, and can therefore not serve as a reference solution.

2.2 Context management

The context management framework employed in this work is called CMF and was developed in the course of the MAGNET and MAGNET Beyond projects [27] and the OPEN project [26]. The CMF handles gathering and management of context information in a distributed system [34]. Through standardized interfaces, providers of context information can insert pieces of information into the system where consumers of context information can query for them. Dissemination of data in the CMF can be optimized to reduce overhead in the communication channels in the same way as ad-hoc routing backbones are optimized for broadcasting of messages [18].
The CMF supports both proactive and reactive retrieval schemes. In proactive mode, a notification to a consumer is sent whenever a value of a context information is changed and in reactive mode, the consumer queries whenever recent context information is needed. A detailed analysis of the context management system has revealed that the probability of a mismatch between the reported value and the actual value of a piece of context information depends on the delay in the network [22] [21]. This is an important property when interacting with the context management system, however, it is not considered directly in this work. We focus on how inaccurate state estimates affect decision performance in a case-study of network parameter estimation. Also considering the mismatch probability as an inaccuracy source would be possible in the scope of the estimation framework in future work.

2.3 Stochastic modeling approaches

The primary challenge of this work is to make decisions about migration under uncertainty. The network state is not always observable, and must be inferred from observable parameters that have a stochastic relation to the network state. The hidden Markov model and Bayesian Network modeling approaches in described in the following are widely recognized and used methods to infer hidden parameters from observable parameters. The Markov Decision Process is models the effect of making choices in stochastic model, which is relevant for the decision problems in migration.

2.3.1 Hidden Markov models

A hidden Markov Model is a Markov chain where the observed state (Y_t) may not be equal to the true state (X_t) [29]. The parameters of the model are the initial state distribution, $\pi(i) = P(X_1 = i)$, the transition model, $A(i,j) = P(X_t = j, X_{t-1} = i)$, and the observation model $P(Y_t, X_t)$. By inference, a sequence of observations can be used to estimate a true state sequence. For inference, a Forward-Backward algorithm can be used. The Forward-Backward algorithm produces as a byproduct the *a posteriori* Marginal distribution, which leads to Marginal Posterior Mode estimate, which is the most probable state of X_t, given the observed sequence Y until t [32]. We will use the hidden Markov model approach to continuously estimate the most likely state of hidden network parameters from network observations in Chapter 4, 5 and 7.

HMMs have been widely used for estimation of hidden states. The most recognized application case is the speech recognition example presented in [29]. More related to network, the HMMs have been to estimate end-to-end loss nature in hybrid wired/wireless networks [17], model an internet communication channel for performance analysis [32], network traffic categorization [36], network delay prediction [37] and fault prediction and diagnosis [33] [5].

2.3.2 Bayesian Networks

Bayesian Networks is a graphical model that unite probability theory and graph theory. Probabilistic graphical models are graphs in which nodes represent random variables, and the (lack of) arcs represent conditional independence assumptions. Hence they provide a compact representation of joint

probability distributions. Using BNs to estimate hidden state under uncertainty has been studied and applied in industrial control systems [11] [38] and medical diagnosis [20] [15]. In general, good state estimation accuracy can be achieved and the BNs provide a useful framework for modeling complex systems based on learning data and expert knowledge. Compared to deterministic rule-based systems (e.g. [16]) BNs are accentuated by their ability to cope with unreliable observations [38] [35] while achieving useful performance typically requiring less training data than other popular AI methods like neural network methods [39] [38]. Ongoing research in BNs is focused on applying, often application specific, approximate inference methods while maintaining accuracy [30] [35].
BN fault diagnosis has also gained attention in network management. In work of [35] [31] the diagnosis strengths of the BN methods has been applied in diagnosis of faults in complex network infrastructures based on correlation of network fault notifications. The authors in [10] train a BN to learn the normal state of a network based on observations of network traffic in a network routing device. They show how the trained BN is capable of detecting network anomalies but do not diagnose their cause. Similar to [10] our BN approach is based on observations drawn from network traffic as opposed to explicit context change notifications.

2.3.3 Markov Decision Processes

A Markov Decision Process is a discrete time stochastic control process, based on a Markov chain [28]. At each time step, the process is in some state s, and the decision maker may choose any action a that is available in state s. The process responds at the next time step by randomly moving into a new state s', and giving the decision maker a corresponding reward $R_a(s, s')$. The probability that the process chooses s' as its new state is influenced by the chosen action. Specifically, it is given by the state transition function $P_a(s, s')$. Thus, the next state s' depends on the current state s and the decision maker's action a. But given s and a, it is conditionally independent of all previous states and actions; in other words, the state transitions of an MDP possess the Markov property.

The core problem of MDPs is to find a policy for the decision maker: a function π that specifies the action $\pi(s)$ that the decision maker will choose when in state s. Note that once a Markov decision process is combined with a policy in this way, this fixes the action for each state and the resulting combination behaves like a Markov chain. The goal is to choose a policy $\pi*$

that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon.

The standard family of algorithms to calculate this optimal policy requires storage for two arrays indexed by state: value V, which contains real values, and policy p which contains actions. At the end of the algorithm, p will contain the solution and V(s) will contain the discounted sum of the rewards to be earned (on average) by following that solution from state s. The algorithm has the following two kinds of steps, which are repeated in some order for all the states until no further changes take place. They are

$$\pi(s) := \operatorname*{argmax}_a \left\{ \sum_s P_a(s,s')(R_a(s,s') + \gamma V(s')) \right\}$$

$$V(s) := \sum_s P_{\pi(s)}(s,s') \left(R_{\pi(s)}(s,s') + \gamma V(s') \right)$$

where γ is the discount rate and satisfies $0 > \gamma > 1$. It is typically close to 1.

A calculation approach called *value iteration*, which will be used in this work, combines the steps into one calculation

$$V(s) = \max_a \left\{ \sum_s P_a(s,s')(R_a(s,s') + \gamma V(s')) \right\}$$

This update rule is iterated for all states s until it converges with the left-hand side equal to the right-hand side.

References

[1] M. Aksit and Z. Choukair. Dynamic, adaptive and reconfigurable systems overview and prospective vision. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 84–89, 2003.

[2] O.S.G. Alliance. *Osgi service platform, release 3*. IOS Press, Inc., 2003.

[3] A. Carzaniga, G.P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of the 19th international conference on Software engineering*, page 32. ACM, 1997.

[4] A. Carzaniga, G.P. Picco, and G. Vigna. Is Code Still Moving Around? Looking Back at a Decade of Code Mobility. In *Software Engineering-Companion, 2007. ICSE 2007 Companion. 29th International Conference on*, pages 9–20. IEEE, 2007.

[5] A. Daidone, F. Di Giandomenico, A. Bondavalli, and S. Chiaradonna. Hidden Markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. *IEEE Proc. of the 25th Symposium on Reliable Distributed Systems (SRDS), Leeds, UK, October 2006*, 2006.

[6] FIPA. Foundation for Intelligent Physical Agents (FIPA). http://www.fipa.org/, 2011.

[7] C.L. Fok, G.C. Roman, and C. Lu. Mobile agent middleware for sensor networks: An application case study. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 382–387. IEEE, 2005.

[8] J. Fox and S. Clarke. Exploring approaches to dynamic adaptation. In *Proceedings of the 3rd International DiscCoTec Workshop on Middleware-Application Interaction*, pages 19–24. ACM, 2009.

[9] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding code mobility. *Software Engineering, IEEE Transactions on*, 24(5):342–361, 2002.

[10] CS Hood and C. Ji. Probabilistic network fault detection. *Global Telecommunications Conference*, 3, 1996.

[11] E. Horvitz and M. Barry. Display of information for time-critical decision making. *Proc. of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 1995.

[12] T. Illmann, F. Kargl, M. Weber, and T. Kruger. Migration of mobile agents in java: Problems, classification and solutions. In *Proc. of the Int. ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA00)*. Citeseer, 2000.

[13] JADE. Java Agent Development Environment. http://jade.tilab.com/, 2011.

[14] H. Klus, D. Niebuhr, and A. Rausch. A component model for dynamic adaptive systems. In *International workshop on Engineering of software services for pervasive environments: in conjunction with the 6th ESEC/FSE joint meeting*, page 28. ACM, 2007.

[15] L. Leibovici, M. Paul, A.D. Nielsen, E. Tacconelli, and S. Andreassen. The TREAT project: decision support and prediction using causal probabilistic networks. *International Journal of Antimicrobial Agents*, 30:93–102, 2007.

[16] G. Liu, AK Mok, and EJ Yang. Composite events for network event correlation. *Proc. of the 6th Int. Symposium on Integrated Network Management*, pages 247–260, 1999.

[17] J. Liu, I. Matta, and M. Crovella. End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment. *Proc. of Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.

[18] Y. Liu, H.P. Schwefel, F.Y. Li, and A. Nickelsen. Circuit-Elimination based Link-State Routing in Mobile Ad-hoc Networks. In *Wireless Communication Systems, 2007. ISWCS 2007. 4th International Symposium on*, pages 185–189. IEEE, 2007.

[19] K. Luyten and K. Coninx. Distributed user interface elements to support smart interaction spaces. In *Multimedia, Seventh IEEE International Symposium on*, page 8. IEEE, 2005.

[20] B.N. Nathwani, K. Clarke, T. Lincoln, C. Berard, C. Taylor, K. Ng, R. Patil, M.C. Pike, and S.P. Azen. Evaluation of an expert system on lymph node pathology. *Human Pathology*, 28(9):1097–1110, 1997.

[21] R.L. Olsen. *Enhancement of wide-area service discovery using dynamic context information*. Institute of Electronic Systems, Aalborg University, 2008.

[22] R.L. Olsen, A. Nickelsen, J. Nielsen, H.P. Schwefel, and M. Bauer. Experimental analysis of the influence of Context Awareness on Service Discovery in PNs. *15th IST Mobile & Wireless Communication Summit, Mykonos, Greece*, 2006.

[23] R. Orfali and D. Harkey. *Client/server programming with Java and CORBA*. John Wiley & Sons, Inc. New York, NY, USA, 1998.

[24] F. Paternò, C. Santoro, and A. Scorcia. Preserving Rich User Interface State in Web Applications across Various Platforms. *Engineering Interactive Systems*, pages 255–262, 2008.

[25] F. Paternò, C. Santoro, and A. Scorcia. User interface migration between mobile devices and digital tv. In *HCSE'08*, page 292. Springer-Verlag, 2008.

[26] F. et al Paterno. Open pervasive environments for migration interactive services – OPEN. http://www.ict-open.eu, 2010.

[27] Ramjee (Ed.) Prasad. My personal Adaptive Global Net (MAGNET), 2010.

[28] M.L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. *IMA Journal of Management Mathematics*, 1994.

[29] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSp Magazine*, 3(1):4–16, 1986.

[30] I. Rish, M. Brodie, and S. Ma. Accuracy vs. efficiency trade-offs in probabilistic diagnosis. *Eighteenth national conference on Artificial intelligence*, pages 560–566, 2002.

[31] I.B. Rish, M.S.M. Odintsova, N. Beygelzimer, A. Grabarnik, G. Hernandez, K.I.B.M.T.J.W.R. Center, and NY Hawthorne. Adaptive diagnosis in distributed systems. *Neural Networks, IEEE Transactions on*, 16(5):1088–1109, 2005.

[32] K. Salamatian and S. Vaton. Hidden markov modeling for network communication channels. *ACM SIGMETRICS Performance Evaluation Review*, 29(1):101, 2001.

[33] F Salfner. Predicting failures with hidden Markov models. *Proc. of 5th European Dependable Computing Conference (EDCC-5)*, pages 41–46, 2005.

[34] L. Sanchez, J. Lanza, R. Olsen, M. Bauer, and M. Girod-Genet. A generic context management framework for personal networking environments. In *Mobile and Ubiquitous Systems-Workshops, 2006. 3rd Annual International Conference on*, pages 1–8. IEEE, 2007.

[35] M. Steinder and A.S. Sethi. Probabilistic fault localization in communication systems using belief networks. *IEEE/ACM Transactions on Networking (TON)*, 12(5):809–822, 2004.

[36] C. Wright, F. Monrose, and G.M. Masson. HMM profiles for network traffic classification. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 9–15. ACM, 2004.

[37] T. Yensen, J.P. Lariviere, I. Lambadaris, and R.A. Goubran. HMM delay prediction technique for VoIP. *Multimedia, IEEE Transactions on*, 5(3):444–457, 2003.

[38] Z. Yongli, H. Limin, and L. Jinling. Bayesian networks-based approach for power systems fault diagnosis. *Power Delivery, IEEE Transactions on*, 21(2):634–639, 2006.

[39] R. Zhang and A.J. Bivens. Comparing the use of bayesian networks and neural networks in response time modeling for service-oriented systems. *Proc. of workshop on Service-oriented computing performance: aspects, issues, and approaches*, pages 67–74, 2007.

3

Service migration framework

This chapter describes the details of a migration scenario that are used to create a model of a migratory application, a model of the migration process, a migration platform and its deployment. Details of the migration scenarios from Chapter 1 are elaborated. The chapter provides an overview of the specific features characterizing a migratory application. Furthermore, it describes the necessary functions to perform migration, how they are defined in a migration platform and how the platform is realized as middleware in a migration infrastructure. The work described in this chapter is joint work carried out as a part of the EU FP7 research project called OPEN (Open Pervasive Environments for iNteractive migratory services – see ww.ict-open.eu or [6] for more information about the project).

3.1 Migration scenario

A user is viewing a video-stream of a news cast on a mobile device. The video is streamed from an Internet service. The mobile device is connected to the Internet through a wireless LAN connection. The user's experience of watching the video on the mobile device has a certain quality. The experienced quality is influenced by the continuity of the stream, the size of the display and the allowed mobility of the device. At one point a problem in the network causes glitches in the video stream because packets are lost due to the network problem. Glitches are annoying which causes a reduction in the quality of the user experience. However, a large display in close range of the user is able to handle packet losses better, since is has better processing capabilities. The large display is not as mobile as the mobile device, but is has a larger screen to display the video stream on. Because the user prefers continuity over mobility in news video streams, the migration platform infers that the large display will give a better quality user experience when there are problems with the network. Moreover, since the type of network problem is expected to last longer than the news cast, it is even worth to wait for a migration to complete from the mobile device to the large display. Based on this information, the migration platform decides to trigger the migration. The user experiences that the video on the mobile device pauses. A short while after, the video continues on the large display from the position on the mobile device and the video player on the mobile device closes.

3.2 Model of a migratory application

In the following, the concepts of a migratory application are described. Figure 3.1 illustrates how the concepts are related. A migratory application is specified by 1) the functions it provides, for instance, video output and touch-based input, 2) its internal business logic and 3) its state. Functions are abstract representations of application functionality. In the example in Figure 3.1, the functions are *network*, *display* and *input*. An application is specified by a set of functions. Functions may depend on other functions in order to run. If a function cannot be implemented, the application cannot run. Functions are implemented by components and several components may implement the same function, for instance, video output on a mobile device is implemented by a component that may be different to one that implements video output on a fixed large screen. Similar to function interdependencies,

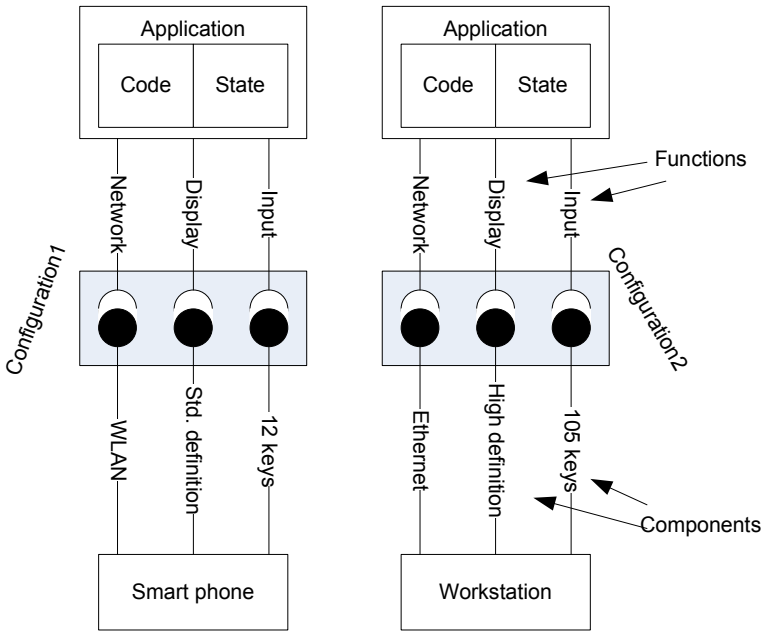

Figure 3.1: Example of a migratory applications, that is defined by *functions*, which are implemented by *components*. Configurations specify which components implement which functions, and several configurations of the same application may exist.

components can depend on other components. For instance, the display component for the mobile device may require a specific component to implement a network function. A part of the component specification is the location of the component, i.e. which device it runs on. Components also have requirements to the device and the context that must be met to allow the component to run. Requirements can be about specific contextual parameters, such as a minimum size of an available screen on a device.
Configurations specify which functions are implemented by which components and define the quality of user experience the combination will produce. One application can be realized in several different configurations, where the set of functions are implemented by different components. A *utility function* for each configuration quantifies the user experience quality in the specific context conditions, for instance, for different values of network packet loss or for different locations the user may be in.

When an application instance is running, it may change its state. Every migratory application includes an interface to export an object that contains

enough information about the application state to allow the object to be imported in another application instance and let that instance resume from the state of the original application.

3.3 The migration process

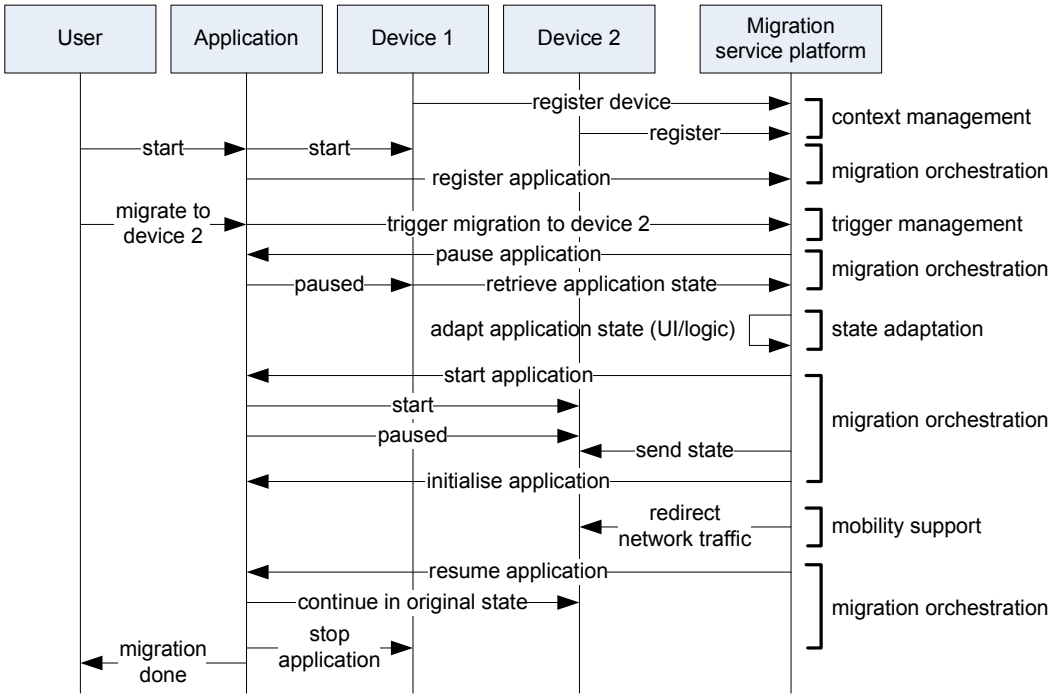

Figure 3.2: Migration procedure involving the user, the application, the two devices that the application migrate between and the underlying migration service platform.

The procedure for a migration is described in Figure 3.2. Initially, all devices and applications are registered, including device and network capabilities as well as application's requirements to devices and networks. Registration only occurs once, also for multiple migrations. Then, migration can be triggered, either manually by the user or automatically by the application or as a reaction to contextual changes. After the trigger, the original application instance is paused to enable state retrieval to maintain state persistence between application instances. The application state is adapted to suit the target device or network. Then, a new application instance is started on the target device, the adapted state is inserted, and the new application instance can

continue the session of the original. Any persistent network connections to remote servers are redirected to the target device.

Model of the migration process

As a general reference for this dissertation, a model is defined of the migration process. The definitions are illustrated in Figure 3.3. The context is defined discretely as a set of N states, $\mathcal{S} = \{s_1, ..., s_N\}$. The states are abstractions of the network parameters, which are considered relevant if they impact the performance of the application or the migration process. The network state estimate is represented by $\hat{s}$. The best configuration of the application is automatically evaluated periodically by the migration middleware. A set of M configurations, $\mathcal{C} = \{c_1, ..., c_M\}$, constitutes the set of possible next configurations. The middleware has to decide which configuration c' to change to from the current configuration c. Moreover, a set of K orchestration methods, $\Omega = \{\omega_1, ..., \omega_K\}$ defines the set of available choices for orchestration. One configuration must be available and chosen in order for the migration to proceed. If $c \neq c'$, a migration $c \rightarrow c'$ using ω, is attempted. The migration duration, D, is random and characterized by a distribution $f_{D,c\rightarrow c',\omega}(\tau)$ that depends on the size of the state object that needs to be transferred, the properties of the network type of ω and the network state. If the migration is successful, the application is in configuration c' after time D. If the migration fails, with probability $(1 - p_s)$, (p_s is migration success probability) the duration until the system returns to configuration c may be characterized by a different distribution $f_{fail,c\rightarrow c'}(\tau)$.

Migration delay Migration delay, D, is the time it takes to orchestrate the steps during migration (pause source, extract state from source, insert state in target, start target, stop source). The primary component of D is the time it takes to transfer the application state object between the devices and the server. The application state is transferred over the network and therefore D is influenced by network state. D is assumed geometrically distributed over the network states with parameter p_d. Different orchestration methods may have different delay distributions depending on the capabilities of the used technology. For instance, the mean delay of an ω using a WLAN 802.11 connection may be much smaller than the mean delay of an ω using an NFC connection, due to the bandwidth differences of the technologies.

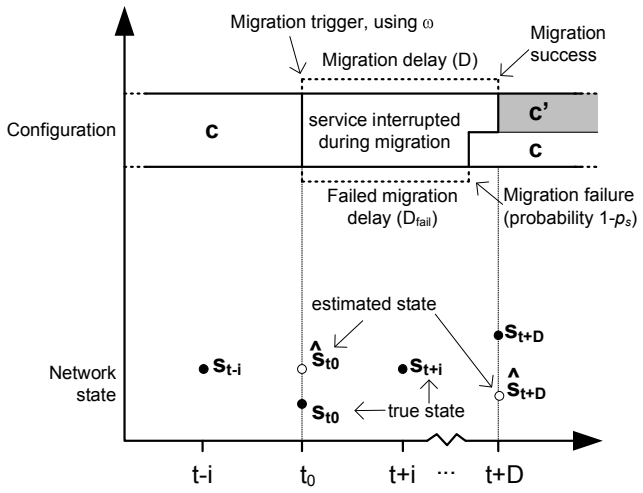

Figure 3.3: Time-line of a migratory system. Network states (s) are estimated and used to decide which configurations (c') to enforce when significant changes in the network state occur. The migration may fail with probability $(1 - p_s)$ after D_{fail} time-steps or succeed after D time-steps.

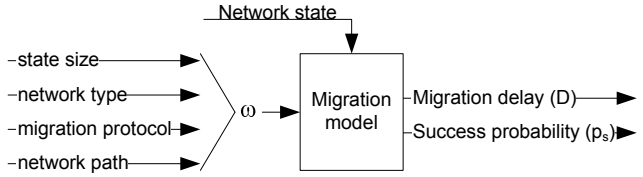

Figure 3.4: Migration elements and network state impact migration delay, D and success probability, p_s.

A difference of ω, besides network technologies, can be different application state sizes to be transferred. A smaller application state object may be transferred faster. However, a smaller application state object may compromise the continuity of the application work flow, since the user may have to redo some steps after migration is complete to re-initialize the application into its original state, since not all state information was transferred. The application re-initialization delay must be considered part of the migration, but a part that is not affected by network state, as opposed to the state transfer delay. It can be considered a constant value of time that is added to the transfer delay that is dependent on the transferred application state size instead of the network state.

Migration success probability The migration success probability describes for each ω the probability of the migration succeeding for each of the network states. A failure is modeled as one of the steps of an orchestration not succeeding. For instance, if the messages in a step are communicated using XML-RPC over TCP, a migration failure happens if one of the TCP connections fail, for instance if SYN packets are not successfully transmitted. Then, the success probability, p_s is calculated as the probability of not loosing any TCP SYN packets during the orchestration, $p_s(p_{loss,s}) = (1 - p_{loss,s}^5)^5$ (assuming one SYN packet per protocol step, and 5 retransmissions of SYN packets) If ω uses non-standard migration protocols, as in the case of NFC, p_s is modeled differently, since the number of TCP connections may be different, or the orchestration method may use a different protocol than TCP. In this case, it may be modeled by a connectivity window, where the migration fails if the duration of the window is shorter than the time required to transfer the state.

3.4 Design of migration support platform

To perform automatic migration successfully, several tasks must be performed by the migration platform. Components to solve these tasks are included in the migration platform, as seen in Figure 3.5 and the interactions during migration between the ones addressed in this work are seen in Figure 3.6. The tasks and corresponding components are described in the following.

1. Discover and register applications, configurations and devices that support migration (context management)
2. Collect network context information, e.g. performance measurements from devices and networks, and network capabilities of available devices (context management).
3. Estimate the state of the network based on collected information and estimate the quality of the user experience of the application in each configuration (network estimation in context management).
4. Infer which configuration gives the best user experience in the current context and trigger migration if the best configuration is different from the current one (trigger management).
5. Find available methods to orchestration the migration and choose the optimal given the current context and the inferred target configuration (trigger management)

6. Orchestrate the migration to move and adapt the application between the devices (migration orchestration)
7. Adapt application state during migration (application state adaptation)
8. Redirect network traffic from source to target in order to support mobility (mobility support)

Note that steps 4 and 5 can be performed jointly or in sequence. In this work they are performed jointly by the trigger management component, see Chapter 5 for more details.

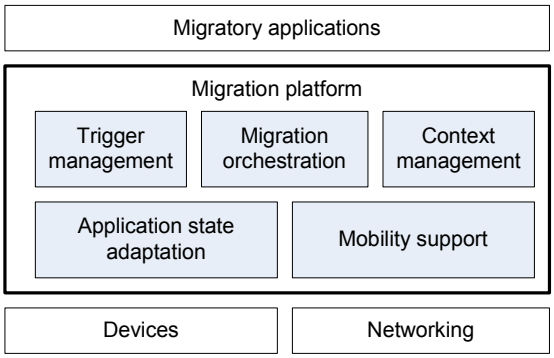

Figure 3.5: Architecture of the migration platform.

3.4.1 Primary components

In the following are descriptions of the components in the platform that are treated in this work. Their input and performance parameters are depicted in Figure 3.7.

Context management

The context of the user describes the user, the user's activity, the network state or devices. Context information indicates when and where to migrate and is the basis for the decisions in the trigger management component. The context management component in the platform ensures access for other modules, applications and services to distributed, dynamic information of various types within the network, and offers search, discovery, access and distribution functionality of context information. In the migration middleware, an existing

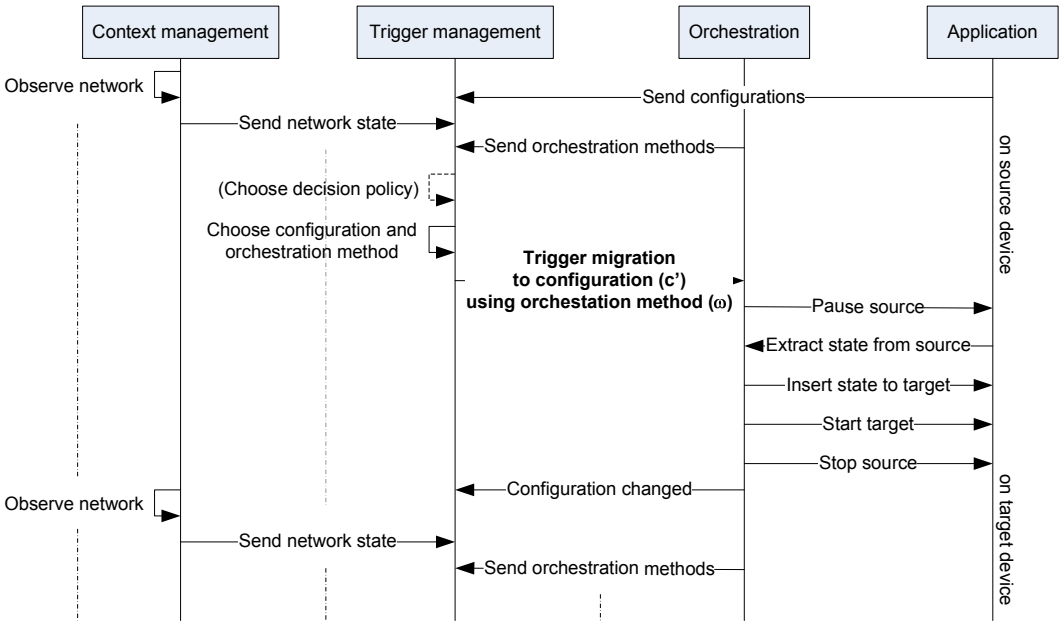

Figure 3.6: Interactions between migration functions internally in the platform.

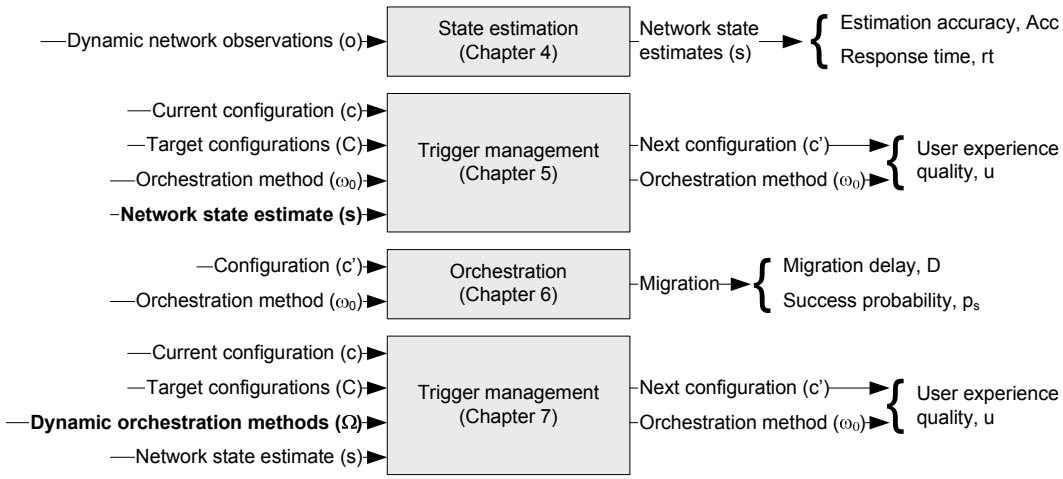

Figure 3.7: Inputs and outputs of the functions in focus in this work. The state estimation function represents the context management component, as this is the main focus in terms context management in this work.

context management framework (CMF) [1] is used. Information is collected by *context agents* installed on involved entities and collected in a storing and processing unit on the migration server. The CMF provides an interface to both query context information when needed and subscribe to changes, such that,

e.g., the decision function can be notified when a relevant change in context occurs, in order to react to the change.

The context management component delivers estimates of the network state to the decision function in the trigger management component. Two metrics, estimation accuracy and response time (to context changes) define the performance of the estimator. This is described in detail in Chapter 4.

Trigger management

The purpose of the *trigger management* component is to decide the best configuration of the applications in order to fulfill the requirements of the user and the application. The component chooses a configuration from a set of configurations based on a decision policy. The configurations are registered in the context management system by the applications on the devices when they join the migration domain. The decision policy can either be generated before the entire migration system is run or generated during run-time based on available configurations. As the trigger management component makes decisions to improve the quality of the user experience, the performance of the trigger management component is measured in the average experience quality delivered to the user during an application run. Details of the component are described in Chapter 5.

Migration orchestration

During the orchestration process, the application is paused to maintain consistency of the state of the application. This means that the user cannot interact with the application during migration. Therefore, the orchestration process has to be fast for the migration to be least intrusive to the user's experience. The process also has to transfer the entire state of the application, such that the user does not need to repeat steps in the application work flow. However, in some scenarios, the methods available for orchestration may be resource-limited, and a trade off between a fast orchestration process and a consistent state is present; either the orchestration is completed quickly, but the amount of transferred state information is reduced, forcing the user to repeat steps in the work flow, or enough state information is transferred to allow a continuity in the work flow, but the orchestration process takes longer time. The performance of the orchestration component is defined by these two metrics; migration delay and success probability. These are described in detail in Chapter 5 and analyzed in detail in Chapter 6.

The orchestration component is responsible for:

- Registration and de-registration of application components.
- Receive the migration trigger and perform the migration.
  - Manage the application instances on source and target device (start, pause, stop).
  - State persistence: retrieves and transfers the state to the target device;

The interaction between the application and the orchestration component is managed using a protocol on the device where the application is running.

3.4.2 Secondary components

The following components are required in the migration platform but not treated further in this work. Background on the solutions can be found in deliverables of the OPEN project [6].

Application state adaptation

The platform supports adaptation of both the user interface and the application logic during migration. When migration is triggered to an interactive device other than a desktop, the migration server adapts the application user interface by building the corresponding logical description through a reverse engineering process and using it as a starting point for creating the implementation adapted to the accessing device. Adaptation of the application logic is done by changing the internal wiring of the components. For instance, if the target device of migration includes the possibility of decoding video in hardware, instead of software, an additional or a different component may be utilized for decoding after migration. The orchestrator component instructs the adaptation component(s) when to adapt the application components and state.

Mobility support

In scenarios where devices or applications change network as a part of the migration process the change must be transparent to the application server (and

client) in order for them not to require a reconfiguration of e.g. IP addresses. The network may be changed during run-time for several reasons, including congestion in the current network or device mobility between networks. As described in Chapter 1, the migration procedure must be *transparent* to any remote service, in order to support existing remote services that do not support migration. To solve this, the migration procedure is made transparent by using a mobility function in the migration platform. The mobility function is deployed as a mobility anchor point in the application data stream as SOCKS-based proxy server [2] to redirect data during migration. The orchestrator component instructs the mobility anchor point when to redirect which traffic flows.

3.5 Deployment of the migration middleware

The migration platform is realized as a middleware between the migratory application and the execution platforms, i.e. devices and networks. The platform contains functionality to enable migratory applications to migrate between different execution platforms. By realizing the platform as middleware, the migration function can be shared between multiple migratory applications, such that the application developers can focus on application development and not migration development.

The platform employs a client/server infrastructure to centralize information collection and decision making. The deployment of the middleware server, the mobility anchor point (for mobility support) and the client parts is illustrated in Figure 3.8. The migration server contains the shared functions and is the point where most decisions are made, since the server is the central point of information. The server instance may reside on any device, so long as it is reachable by the clients. Typical deployment cases are in the user's home, in an enterprise or operator's infrastructure or it may even be accessible via the Internet as a public web-service. The migration client is typically the end user terminal (but may also be a large public display), on which migratory applications are running. The client-side middleware implements the part of the migration functionality that is common across applications, and interact with migration server. Applications interact with the middleware through a specifically defined interface (see details in [3]). For example, an application can implement the application-side functions and publish them as XML-RPC methods and call the middleware-side functions through XML-RPC as well. By using a general framework like XML-RPC to implement the interface, the platform does not pose any requirements to the technology that the applications

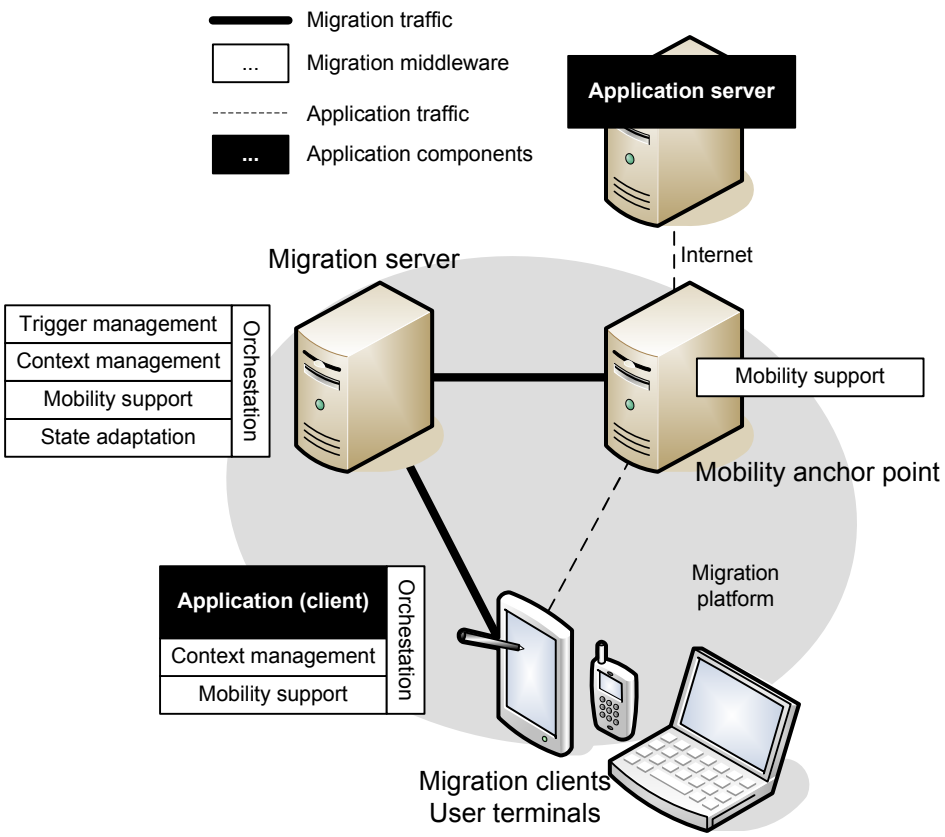

Figure 3.8: The migration middleware deployment; a client part resides on the user terminal where also the application client-part is running and a server-part resides in a central migration server within the migration domain. Traffic is tunneled through a mobility anchor point to make the migration seamless for the application server.

are implemented. The platform can as such support any kind of technology that can publish XML-RPC methods.

The platform is deployed as a centralized architecture as this type of architecture can handle the collection of information about the available devices and their capabilities as well as access to context information such as network performance and alike. Without a common platform to handle information collection, the applications need to implement it themselves. Moreover, they each need to implement capabilities to manage migration triggers and orchestrate migration. On one hand, the centralized architecture faces many relevant challenges of migration in dynamic and resource-constrained networks without having to deal with additional coordination and dynamics challenges as

introduced by a distributed architecture without a dedicated central controlling entity. On the other hand, the need for a central controlling entity limits the scenarios in which the middleware can support migration. We assume that scenarios with a central entity are sufficient to study the migration properties and therefore consider the distributed architecture out of scope of this work. Full ecosystems of the hardware and the software that would enable the proposed architecture are becoming increasingly available. In the market today, large companies develop themselves many devices and software platforms that can facilitate a centralized architecture. By introducing the migration platform in their ecosystem, they will be able to control the interaction between the devices. The developers are then in a position to provide a optimal migration experience, as they do not have to rely on involvement from other parties. Examples of such companies and their ecosystems are listed in Table 3.1.

Company	*Products*
Apple	iPhone, iPad, Apple TV, MacBook, iMac
Google	Android, Chrome, Google Phone
Microsoft	Windows, Windows Mobile, Windows Phone
Samsung	mobile devices, TVs and set-top boxes, laptops, netbooks
Sony	mobile devices, TVs, netbooks, home entertainment

Table 3.1: Examples of players able to provide migration features with-in their own product ecosystem.

.

A prototype of the migration platform has been implemented in an experimental setup to make measurements for the work reported in [2]. In the experimental setup, an link emulator was used to emulate wireless network properties such as packet loss and delay in dynamic network topologies [4]. The measurements were of migration delays and success probabilities, which can be used directly as input in the models presented in this dissertation. The emulator emulates link properties based on a pre-generated link models, as defined by a scenario with predefined mobility. The emulator has been extended to support real-time, online generation of link models, based on the case where changes in communication caused by link properties may change mobility patterns [5]. This features has not been used in migration scenarios yet, but could be used to evaluate performance of the migration platform in situations where slow or failed migration affect the user's mobility pattern. One use-case would be if a migration fails due to network problems and the user moves to find better network and retry the migration. The emulator supports experimental evaluation of such a use-case.

3.6 Dynamics in the migration domain

Dynamics in the application execution environment occur in the following dimensions:

Context parameters Many dynamic context parameters can affect the performance of the migratory application or the migration process. Network state is the example of context in this work, but device resources like processing power or remaining battery time are also relevant dynamic context parameters.

Device availability A device may join and leave the migration domain. When a device enters the domain, device capabilities and available components are registered with the migration server.

Component availability Components may be available or unavailable depending on whether the resources they require are available from a device. One component using a resource may render another component unavailable because it depends on the same resource. In this way the context influences the component availability.

Configuration availability As new device type and thereby new components may be introduced into the migration domain, new configurations may as well, specifying how an application can make use of the new device type. A new configuration is installed directly in the existing configuration set of an application which is installed in the migration domain.

Application availability New applications may be installed in the migration domain or un-installed by users or operators

Due to the dynamics described above, the migration system must be able to handle dynamics in the execution environment and be scalable in terms of available configurations and applications.

3.7 Summary

This chapter described the proposed migration platform that contains support functions for migratory applications. The functions are collected in a common platform to relieve the application developers of the task of developing migration abilities for each application. The platform enables automatic triggering

of migratory applications based on context changes. The functions in the platform are realized as middleware in a network infrastructure. The infrastructure contains the devices running the applications, a migration server to coordinate and orchestrate the migration process and adapt the migratory application to suit the device capabilities, and a mobility anchor point to redirect network connections when migrating between different networks. The middleware can be used by migratory applications during the migration process or the applications can implement subsets of the migration functions themselves. To use the middleware functions, migratory applications use a specifically defined interface to the platform. For analysis in the following chapters, models were presented of a migratory application and the migration process.

References

[1] M. Bauer, R.L. Olsen, M. Jacobsen, L. Sanchez, M. Imine, and N. Prasad. Context management framework for MAGNET Beyond. In *Workshop on Capturing Context and Context Aware Systems and Platforms, Proceedings of IST Mobile and Wireless Summit*. Citeseer, 2006.

[2] K. Hoejgaard-Hansen, H-P. Schwefel, and H. C. Nguyen. Session mobility solution for client-based application migration scenarios. In *Proc. of The 8th International Conference on Wireless On-demand Network Systems and Services*, 2011.

[3] Martin, M. et al. D4.2: Migration Service Platform Design. Technical report, ICT-OPEN EU FP7 project, 2009.

[4] A. Nickelsen, M.N. Jensen, E.V. Matthiesen, and H.P. Schwefel. Scalable emulation of dynamic multi-hop topologies. In *Wireless and Mobile Communications, 2008. ICWMC'08. The Fourth International Conference on*, pages 268–273. IEEE, 2008.

[5] A. Nickelsen and H.P. Schwefel. Emulation of Wireless Multi-Hop Topologies with Online Mobility Simulation. *International Journal on Advances in Telecommunication*, 2(1):27–36, 2009.

[6] Paterno, F. et al. www.ict-open.eu, 2010.

4

Probabilistic network state estimation

This chapter presents the solutions to the context management component in the migration framework, with particular focus on functions to estimate network state. Dynamic networks are unreliable, which challenges state estimation. The chapter first motivates the need for network state estimation in migration scenarios. Then a scenario is described to illustrate the challenges of making accurate and fast state estimation. Two approaches are analyzed as solution candidates. A Hidden Markov model-based and a Bayesian Network-based approach are investigated as case studies. Finally, their performance is evaluated by simulation of a migration scenario.

4.1 Motivation

The decisions made during migration, such as automatic triggering and choice of orchestration method, are reactions to context changes. Context as a general concept means any piece of information that can be used to describe the current situation of the user and/or the application. In this work, we focus on context information about the state of the network, i.e. parameters that describe the state of the network. As we assume that the migratory applications are client-server based and thus utilize one or more networks to make an end-to-end connection, the state of the network influences the performance of the application. Such influence must be considered when making decision about the migration, and therefore context information has to be available to the decision maker.

Estimating network state is complicated as observations from network traffic may be *unreliable* i.e. missing, obsolete, noisy and ambiguous [12]. An approach to overcome unreliable observations and provide robustness to changes is to use and correlate several observations across multiple protocol stack layers. This is, however, not trivial as more observations can lead to *contradicting* observations and increased ambiguity. To target these challenges we perform a study of estimation methods to apply in the migration framework. We employ two model-based methods, namely Hidden Markov models and Bayesian Networks, as they are able to correlate multiple observations to multiple network state parameters. We compare their performance in terms of accuracy to a simpler threshold based heuristic method called the optimal threshold (OT) where observations and network states are directly mapped.

Most migration decisions are made centrally on the migration server, so network state information has to be available at the migration server at the time a decision is made. Network traffic is not routed through the migration server, which makes it difficult to observe the state of the network directly on the server. As described in Chapter 3, the migration framework utilizes the context management component to

- Estimate the state of context information on entities involved in the migratory application
- Deliver context information to where it is needed, in particular trigger management on the migration server

We use a state-of-the-art system for the context management component called the Context Management Framework (CMF) (cf. Chapter 2). The

CMF handles delivery of context information such that the decision makers can query the CMF for network state information when needed. The focus is on the problems of retrieving and estimating the network state before it is distributed via the CMF. An overview of external interfaces of the estimation function is illustrated in Figure 4.1. The performance metrics are *estimation accuracy* and the time it takes to detect a change in context, called *response time*. These are used both for comparison of estimation methods and further as input parameters to the decision maker in the later chapters to analyze how different settings affect migration performance.

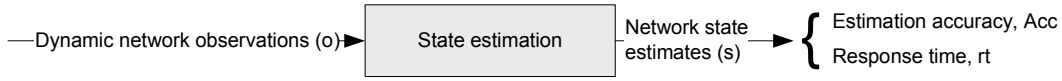

Figure 4.1: Chapter overview.

4.2 Scenario

The scenario represents domain knowledge and defines a set of hidden network parameters and available observations. The scenario used in this chapter is based on the migration scenario described in Chapter 3. The details of the scenario are illustrated in Figure 4.2.

The migratory application uses TCP to establish an end-to-end connection between the application client and the remote server. The performance of an application that uses TCP is sensitive to packet loss, which causes a drop in the connection throughput, because of TCP congestion control mechanisms. TCP reacts strongly to packet loss as it infers *congestion* of the network route and, due to its congestion avoidance mechanisms, reduces the transmission rate to maximize the throughput. In the scenario, the end-to-end connection spans wireless and wired networks. Here, packet loss can be caused by either *congestion* in a router due to excessive cross-traffic or by a low quality of the *wireless link* due to radio interference or mobility. These are the two causes of packet loss considered in this work. It is these two parameters that are estimated by the estimation function. In the end-node, observation points based on network traffic are identified that contain information useful to infer about the hidden network state. The TCP end-point of an application connection offers several potential observations which are all affected by the network state. In this work, the focus is delimited to consider **round-trip time** (RTT), **packet**

retransmission rate and **frame retransmission rate** as observable variables. Packet loss cannot be avoided and therefore influence on application performance must be mitigated by avoiding the causes of packet loss. In this scenario the recovery action decided by trigger management is to migrate to another device that has better abilities to handle packet loss.

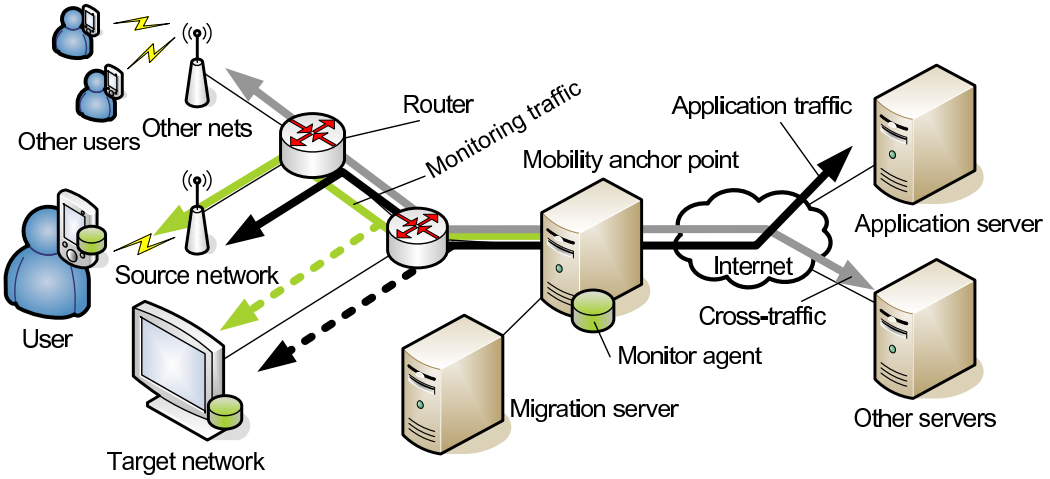

Figure 4.2: Scenario overview. The devices involved in the migration connect to the infrastructure using wireless and wired links connections. Packets may be lost on the end-to-end connection due to congestion which may occur in a router or due to an unreliable wireless link.

4.3 Network state model

We consider a discrete state-based model of the network, as illustrated in Figure 4.3. The state space is defined based on the parameters that influence the application performance. The application depends on throughput of the TCP connection. The throughput can either be *sufficient* or *insufficient*. This binary perspective on the state space is inspired from the fault tolerance domain, where a network fault is either present or not present. This fault perspective on the parameter states allows us to limit the state space of the parameters that influence TCP throughput, in particular congestion and the quality of the wireless link. We define the *route (R)* parameter by two states; *normal* or *congested*. Packets are dropped in the router in both states, but the packet loss rate varies. The packet loss rate in the *congested* state causes *insufficient throughput*. The state space of the *wireless link (WL)* parameter is defined as

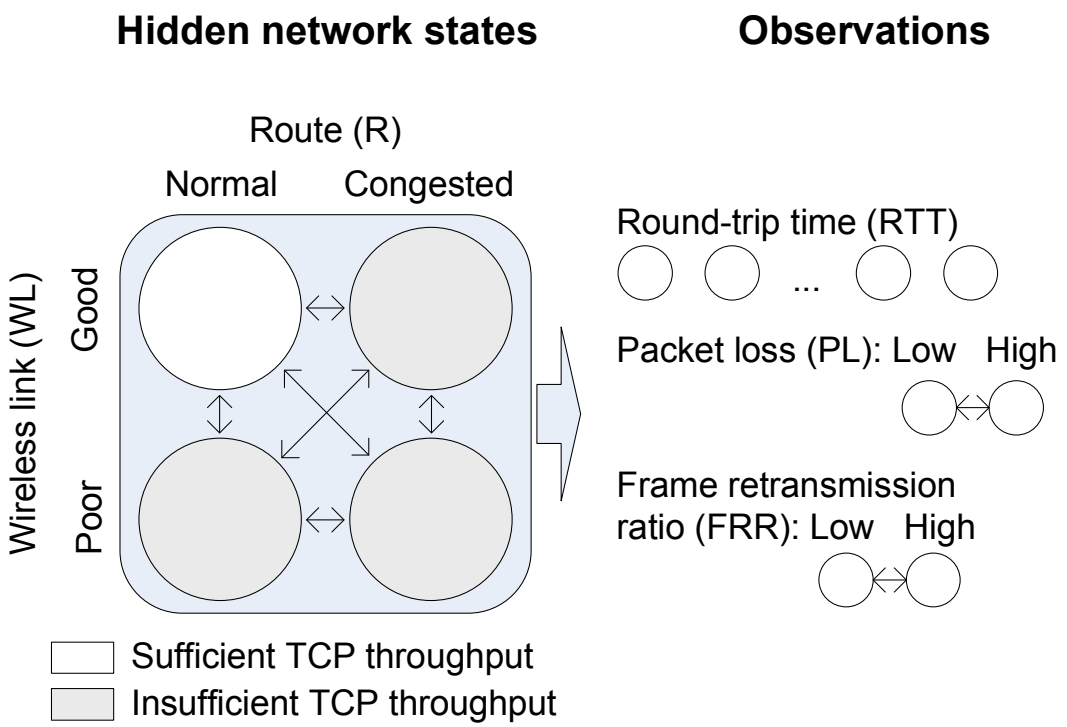

Figure 4.3: State space of the paramaters considered in the network state model.

good or *poor*, also with the *poor* state causing insufficient throughput for the application.

We define both the observed frame retransmission ratio (FRR) and packet loss (PL) by two states; *low* and *high*. The round-trip time (RTT) is a continuous parameter that is discretized into a set of more states than two. In all cases the thresholds are defined by the specifics of the scenario, and in case of the round-trip time, also the number of states.

4.4 Estimation function in migration framework

The detailed structure of the estimation function is illustrated in Figure 4.4. The function consist of a observation processing step and an estimation step before it is delivered using the CMF.

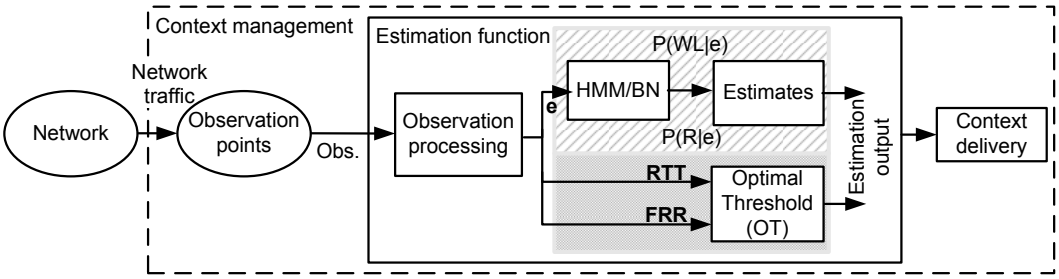

Figure 4.4: Overall estimation framework.

4.4.1 Observation processing step

The Bayesian networks and the threshold approaches are not inherently capable of handling causality in (infinite) time. Instead, functions for processing observations and extracting timely features are utilized in the estimation component. Examples of such functions are simple mean or variance estimates or more advanced auto-regressive functions [3]. In this work mean estimators are used, sampled in discrete steps using a moving average. In this way, an observation may be missing if network measurements, e.g. acknowledgments for RTT, are not available within the time window. It is assumed that mean estimates in this approach can provide useful statistics to infer about network states.

4.4.2 State estimation step

The state estimation step calculates the most likely state of the hidden network parameters based on the processed observations. The estimation performance is directly determined by how efficiently a method can estimate the state of the network.

A state estimate may either be *true* or *false* related to the true network state. The mapping between state estimates and the true network state is depicted in Figure 4.5 and enable definition of the two metrics: *Accuracy (Acc)* is the amount of correct estimates relative to all estimates. *Reactivity time (RT)* measures the reaction time in milliseconds from a state change occurs in the network until it is correctly estimated.

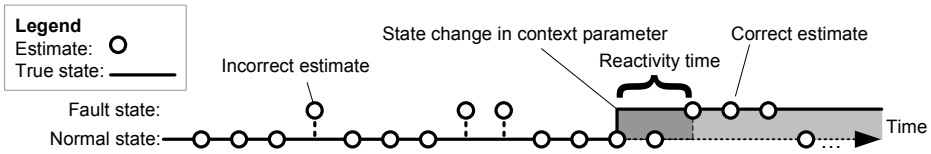

Figure 4.5: State estimate definitions.

4.4.3 Context delivery

The delivery step is handles by the CMF and can be performed in several ways; proactive or reactive. In proactive mode, the state estimate is *pushed* to the decision maker in the form of a notification upon a change. In reactive mode, the decision maker *polls* for the most recent estimate of the state when needed for a decision. In this work, the trigger management component operates periodically and therefore context estimates is assumed polled in a reactive manner.

4.5 State estimation approaches

In the following, the three estimation approaches are described and their performance is evaluated in specific scenarios.

4.5.1 Optimal threshold state estimator

To estimate network states a simple alternative to the model-based methods is to use single optimal thresholds on observations. In its simplest form, the optimal threshold (OT) approach does not combine multiple observations and thus only direct mappings between observations and hidden variables are possible.

We define the optimal thresholds γ_{rtt}, γ_{pl} and γ_{frr} (see Table 4.3 for the specific values used). The criterion used in this work is to define and parametrize the optimal thresholds is based on the *minimum probability of error* (MPE) decision rule. This rule ensures optimal thresholds that minimize the amount of wrong state estimates. This approach is chosen as we do not consider any aspects that would give reason to weigh false positives different from false negatives (see Figure 4.5). Knowledge of the penalty from initiating an unnecessary migration could require another decision rule. This is, however, outside the scope of this work.

The threshold estimator is used as reference and compared to the performance of the HMM and the BN in the following sections.

4.5.2 Hidden Markov model

The hidden Markov model (HMM) models the behavior of an unobservable parameter and the stochastic relation between this behavior and any observable parameters. The HMM can in general be used to model any number of hidden and observable parameters with an arbitrary state space size. We assume the state of the hidden route parameter (normal/congested) to behave according to a first-order Markov model. This allows us to model the relation between the route state and the observed packet loss using the Hidden Markov model. From this model, it is then possible to estimate the state of the hidden route parameter using an observed sequence of packet losses measurements.

Design of hidden Markov model state estimator

The HMM used as a case study for performance evaluation is illustrated in Figure 4.6. The environment is defined by the route parameter and thereby as the set of the two states, $\mathcal{S} = \{s_1 = N, s_2 = C\}$. We model the behavior of the environment as a Markov model. $P(s_i, s_j)$ denotes the probability of a

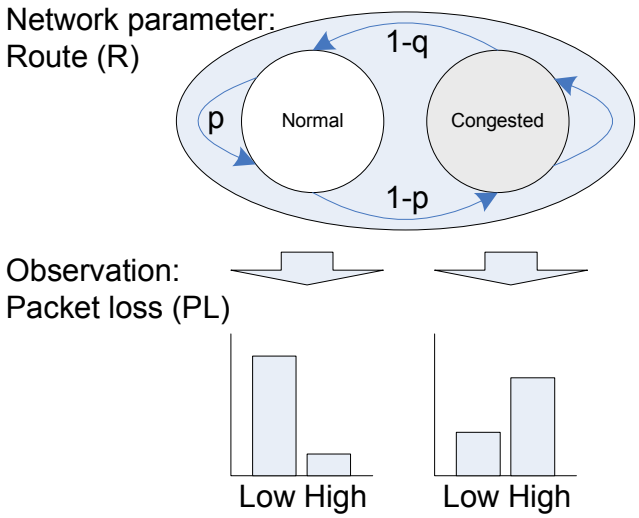

Figure 4.6: Hidden Markov model of the hidden route state and the observable packet loss state

transition from state s_i at time t to state s_j at time $t+1$. The probability of staying in a state in time-step ($P(s_i, s_i)$) is defined for the normal state by p and for the congested state by q. For network observations we use packet loss ratio with two possible states (low, high), representing measurable packet loss in a congested network, $obs = \{o_1 = L, o_2 = H\}$. The separation between o_1 and o_2 is defined by γ_{pl}. During run-time, the Forward-Backward algorithm is used to calculate the probability distribution over the set of states given the sequence of previous and current observations. By using a Marginal Posterior Mode (MPM) estimate [10] (without look-ahead window), the most likely state of the environment can be calculated. Transition probabilities of the HMM are the same as the environment model P. Observation probability distributions, $P(obs|\mathcal{S})$, depend on how the environment affects the observable parameters. Concrete values for both matrices are specified later in the evaluation example.

Performance evaluation results and analysis

We limit the analysis to two sub-scenarios of the overall scenario. In scenario 1, we simulate a network that has a low rate of state changes. In scenario 2, we simulate a network with a high rate of changes. The differences between the two networks are expressed in the state transition probabilities. In scenario 1, the probability of changing state is very low (0.1-0.5 changes per second on average). We simulate with a estimation time interval of 100ms, and the transition probabilities are seen in Table 4.1. In scenario 2, it is opposite, such that the probability of changing state is very high (9.5-10 changes per second on average), as seen in Table 4.2.

$s(t) \rightarrow s(t+1)$	N	C
N	0.99	0.01
C	0.05	0.95

Table 4.1: Transition probabilities $P_1(s(t), s(t+1))$ (scenario 1)

$s(t) \rightarrow s(t+1)$	N	C
N	0.01	0.99
C	0.95	0.05

Table 4.2: Transition probabilities $P_2(s(t), s(t+1))$ (scenario 2)

The probability distributions of the observations in each network state, b, (observation matrix) are defined as (columns = low/high packet loss, row 1 = normal route, row 2 = congested route):

$$b = \begin{bmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{bmatrix}$$

The distributions reflect the situation where high packet loss is primarily observed when the network is congested.

Since the threshold approach assumes a slowly changing environment, it is expected to perform well with in the first scenario and poorly in the second. We simulated 30 runs in 10000 steps and obtained sequences of true network states $s(t)$ and observations $o(t)$ from the HMM, and estimated network states $\hat{s}(t)$ from the OT and HMM estimators. The performance results are shown in Figure 4.7. From the figure we see that overall the HMM has a better accuracy, and that a high rate of state changes impact the estimation accuracy of both approaches. An ideal state estimator is shown for reference.

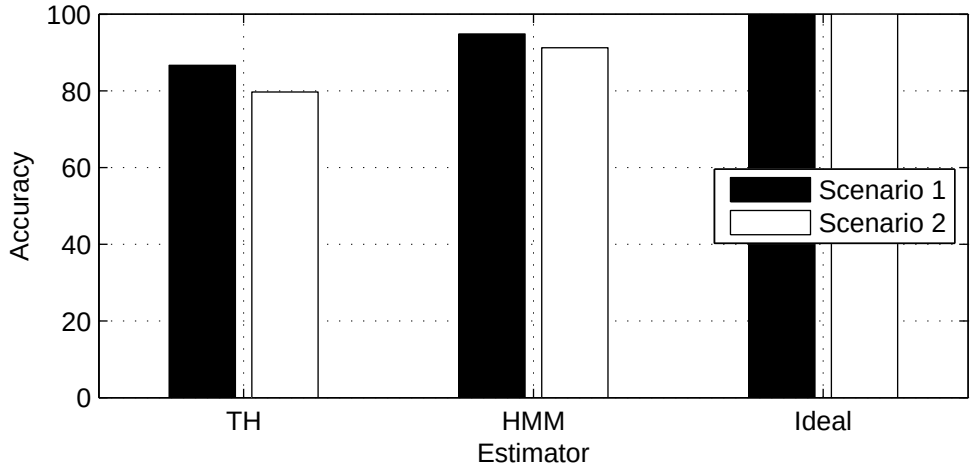

Figure 4.7: Accuracy of the estimators, including the theoretical ideal estimator.

4.5.3 Bayesian Network

A BN [8] describes causal probabilistic relations and enables inference of hidden states under uncertainty. Our BN model is derived from the basic network model in Figure 4.3. The BN estimation performance has been evaluated, in simulation, by the metric of estimation accuracy. Even as a basic BN is considered, our results show how utilizing multiple observations in the BN has the potential to improve performance compared to the OT. Finally, we evaluate the robustness of the BN toward changed network conditions. These results show how the BN using multiple observations is more robust to changes in network delay compared to the OT.

Design of Bayesian Network state estimator

A Bayesian Network (BN) is a graphical model that relates stochastic variables of a domain by their causal relations. Formally, a BN $N = (\mathcal{G}, \mathcal{P})$ consists of two basic entities [4]: a *directed acyclic graph* (DAG) $\mathcal{G} = (X, E)$ where X is a set of nodes $X = \{X_1, \ldots, X_n\}$ and E is a set of edges connecting the nodes. $\mathcal{P}$ is a set of conditional probability distributions (CPD). Each node X_i represents a variable with a finite set of states and each edge represents a causal relation between two variables. A strength of BNs is that independence between variables can be utilized such that only a part of the conditional probabilities present in $P(X)$ needs to be specified in $\mathcal{P}$. This makes it practical to construct and parametrize such models. Most importantly, BNs also make inference in P(X) computationally feasible for large models [13]. More on the background of BNs may be found in Chapter 2 and [4].

Developing a BN for state estimation is a three-step process: (1) *obtaining domain knowledge*, (2) *developing a BN structure* and (3) *obtaining probabilities*. This process is described in the following sections where we also introduce the optimal threshold approach for comparison to the BN.

Intermediate model To construct the BN structure $\mathcal{G}$, variables and their causal relations are mapped from the scenario to a graphical representation. In general, good patterns for construction and methods for verification are still an open issue in the research domain of BNs [9]. The approach in this work has been to construct an intermediate model describing basic system components and in which order these components influence each other, i.e. their causal relations. Next, the intermediate model has been formed into a BN. The intermediate model is depicted in Figure 4.8 and has been specified from the following structured method: (1) Variables have been identified that represent system components where the network parameters originate. E.g., *congested* is a state of the *route* component where route is the end-to-end route. (2) Next, observable variables have been identified where useful information about the states of the unobservable system components can be obtained. These are RTT and packet/frame retransmission rate as described previously. (3) Intermediate variables have been specified that describe system behavior and relations between observations and the system components. As an example the condition of the component *wireless link* affects the observation *packet-retransmission rate* through the intermediate variables *packet loss*. (4) Finally, variables are represented as nodes in $\mathcal{G}$ and edges are identified from causal relations between variables. In the following a short description of intermediate nodes are given. More details can be found in [6] and [7].

Cross traffic load - The cross-traffic load is the load on the bottleneck drop tail queue router (R1 in Figure 4.10) which is assumed to have a constant service rate.
Upstream - The transmitted data is modeled by *Upstream*, that represents the actual rate of outgoing packets from the sender. It is assumed that there is always data to be sent from the application.
TCP - The *TCP* node covers the transmission rate control mechanisms of TCP. TCP controls the upstream based on RTT and detection of packet loss.
Throughput - The rate of successfully acknowledged data.

Unconnected nodes in the model are interpreted as causally independent variables. Some important independence assumptions are:
Upstream→Cross traffic load - It is assumed in this model, that *Upstream* $\ll$ *Cross traffic load*. Thus, load on the bottleneck link is not significantly influenced by the upstream of the sending application. This also means that the influence of *Upstream* on *Congestion* is considered insignificant.
Wireless link→Route - The two components of the network have no (significant) influence on each other, thus the nodes are regarded as being independent.

Bayesian Network model of a TCP connection The model in Figure 4.8 cannot be characterized as a BN, as it clearly contains a cycle *Upstream→Packet loss → TCP → Upstream*. For this type of problem a dynamic BN could be introduced (see [5]). However, the focus in this work is to use a regular BN and thus the cycle must be eliminated. Moreover, some of the intermediate nodes may be disregarded for simplicity. The main steps to achieve this are:

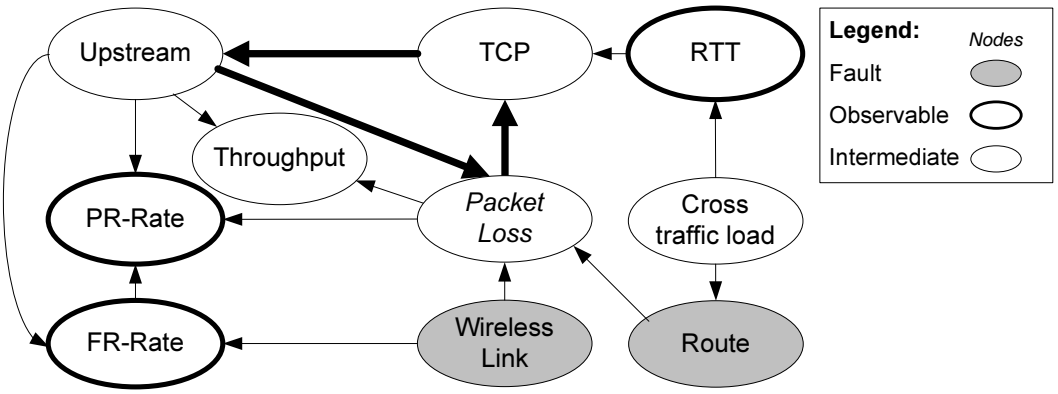

Figure 4.8: Intermediate model.

1) The *TCP*-node is removed to eliminate the cycle. This means that the impact from the congestion and flow control mechanisms of TCP are left unmodeled.
2) Being controlled by TCP the *Upstream*-node is eliminated from the model by converting packet retransmission rate and frame retransmission rate ($\frac{retransmissions}{s}$) into ratios ($\frac{retransmissions}{all-transmissions \cdot s}$) between sent and retransmitted packets/frames (PRR and FRR). Thus, upstream is contained in these observations.
3) Maintaining *Throughput* as an observation is an option. However, immediate TCP actions (e.g. reducing window size) can have a delayed impact on throughput observations. As the BN does not express such causal relations in time having throughput as an observation can be difficult.
4) The state of the *route* is directly defined by the *Cross traffic load* node. Thus, these two nodes are joined.
5) To maintain the relation that both route congestion and a poor wireless link lead to packet loss, the intermediate variable representing packet loss remains in the model.

The final outcome is the BN of Figure 4.9 defined as the *basic model*. PRR is defined by *high*, *medium* and *low*. Compared to FRR (which only depends on wireless link conditions) the additional state enables different expressions of when a single or two faults have occurred. For an overview of the states see Table 4.3.

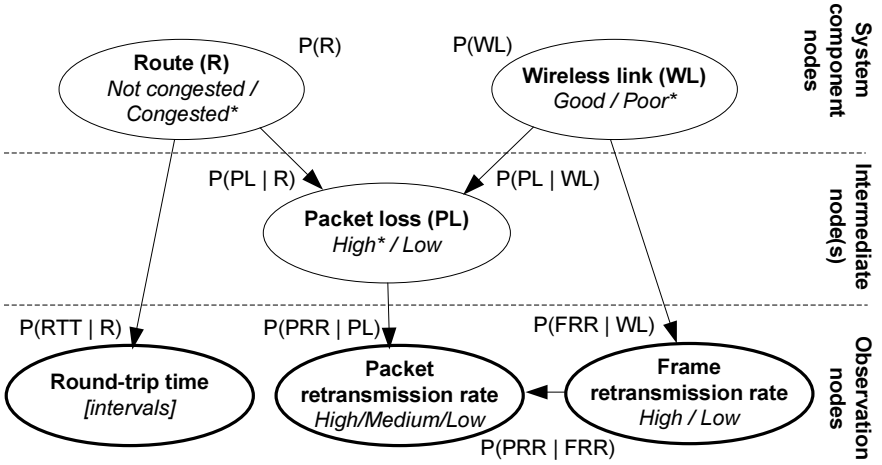

Figure 4.9: The basic model BN.

Realization of the BN Data is needed to parametrize the BN and the OT to perform estimation. The components have been realized in an ns-2 simulation environment that implements the network model of Figure 4.10.

Initially, data from the observation points can be generated by simulation as done in this work, or in practice, read from a network log file or monitored real-time from a communication process. Next, the observation point data is processed into evidence e. After processing, the evidence is propagated in the BN and network state estimates are inferred. For the OT the same processed observations are used and state estimates are given directly from evidence.

Inference in the BN is done by calculating the posterior probabilities $P(R|e)$ and $P(WL|e)$. Posterior probabilities enable an estimate of which state a variable is in and we assume that a fault state is diagnosed if $P(R = \text{fault-state}|e) > 0.5$. Inference can be performed using either exact inference [13], which is NP-hard or alternatively approximate inference methods [11]. The small BN model considered, implies only little computational overhead. Based on 9000 inference cases the average inference time is 0.5 ms (standard laptop running Linux) and thus exact inference is applied.

4.5.4 Case study

In the case study scenario, as displayed in Figure 4.10, *TCP source 1* transmits data to *TCP sink 1* over a wireless link and a wired network. We define π_{loss} to represent the loss rate of the wireless link. Congestion in the buffer of $R1$ is induced by cross-traffic from *TCP source 2*. We define λ_{load} to represent the rate of a Poisson process generating incoming connections to *TCP source 2*. The incoming connections correspond to file transfers with sizes according to a Pareto distribution ($\mu = 10\,KB,\ \beta = 1.5$ [1]). The two network model parameters π_{loss} and λ_{load} are used to define the two parameters in the networking scenario. First, a normal state has been defined as a starting point, where the delivered throughput of the TCP connection is sufficient for the application. Next, the parameters have been tuned empirically in a simulation environment to cause a 50% throughput reduction, which then represent *fault states*. Both fault states are defined to be permanent once they are entered.

The state-spaces of the individual observation nodes have been defined based on the optimal thresholds γ_{rtt} and γ_{frr}. This is done to pursuit good observations for the BN and ensure comparability with the OT approach. As the OT approach does not specify a threshold for PRR, for consistency, optimal thresholds based on the MPE decision rule have been applied for γ_{prr} as well.

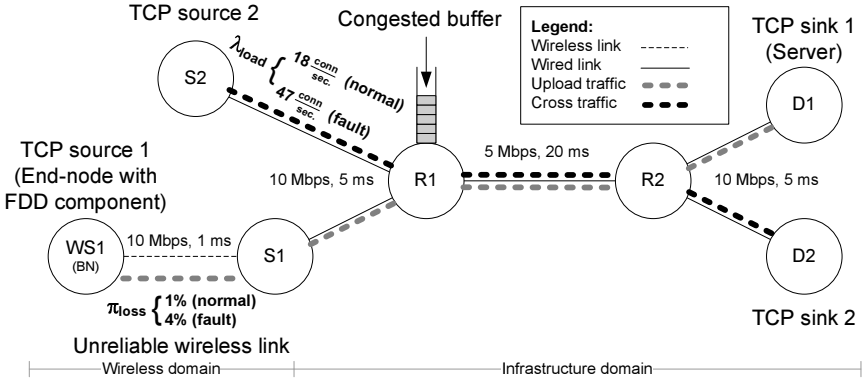

Figure 4.10: Network model of scenario 2.

The optimal thresholds are derived from two marginal distributions of each observation; one distribution where no fault occurs and one where it does. As all observation nodes have a different number of states, the optimal thresholds are applied differently for each observation node in the BN. The RTT has 12 states. The granularity has been found experimentally from minimum and maximum expected RTT values by evaluating which number of states give the best separation of the distributions. In this case, γ_{rtt} is separating state 3 and 4, based on *Route* fault occurrences. In the PRR case, the observation is influenced by both faults which can be generalized into representing 0, 1 or 2 faults occurring (3 states). Thus, these states are separated by two optimal thresholds, $\gamma_{prr,1}$ and $\gamma_{prr,2}$. To obtain these, an additional marginal distribution is used for PRR when both faults occur. The FRR observation node has two states separated by γ_{frr} based on *Wireless Link* fault occurrences.

The conditional probability distributions (CPDs) in the BN are elicited based on *learning*. By using the Expectation-Maximization (EM) algorithm it is possible to estimate the parameters of a probabilistic model [2]. In the

	States	Thresholds
RTT	12	62-105 ms Interval: 4.3
PRR	low/medium/high	0.02, 0.05
FRR	low/high	0.029
PL	low/high	
R	P(R)	0.5
WL	P(WL)	0.5

Table 4.3: BN node and parameter setup.

BN, the parameters are the specific probabilities of the causal relations, e.g. $P\ (RTT \mid R)$.

Two approaches can be used when learning with EM-algorithm; online and offline learning [5]. Offline learning uses pre-recorded sets of data to estimate fixed parameters of the model. In online learning, the parameters are continuously estimated when the BN is in operation. This makes it possible to adapt the model to local network conditions [3]. As offline learning is considered in this work, handling different network scenarios depends on the BN's robustness to such changes.

Data sets for learning have been generated from the simulation containing information from both observations and network states. The probabilities of the network states, i.e. route and wireless link, represent the prior *belief* of parameter behavior. Here, they have been specified equally as 50%/50% as we assume to have no prior belief of whether normal states are more dominant than fault states. All basic model configuration parameters are shown in Table 4.3. Learned conditional probabilities are depicted in Table 4.4 and Figure 4.11.

Performance evaluation results

This section presents the results from evaluating the BN in comparison to the OT approach for networks state estimation. Performance is evaluated by comparison of relevant metrics using a basic setup. Robustness is evaluated by comparing metrics when introducing unmodeled changes in the setup in the following section.

For the performance evaluations the following sub-scenarios were used.
Congested route scenario (1): A transition from normal to fault state in the *route*, the *wireless link* remains in normal state.

	P(PL | R, WL)		P(PRR | FRR, PL)			P(FRR | WL))		
	Low	High	Low	Medium	High		Low	High
0,0	0.96	0.04	0.76	0.21	0.03	0	0.83	0.17
1,0	0.39	0.61	0.36	0.05	0.59	1	0.46	0.54
0,1	0.56	0.44	0.14	0.51	0.35			
1,1	0.10	0.90	0.21	0.10	0.69			

$0 \rightarrow low$, $1 \rightarrow high$

Table 4.4: CPDs for PL, PRR and FRR.

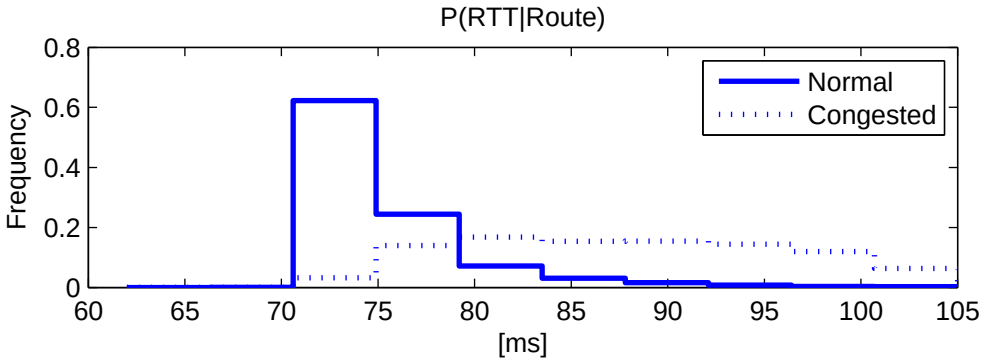

Figure 4.11: CPD for RTT.

Poor wireless link scenario (2): A transition from normal to fault state in the *wireless link*, the *route* remains in normal state.
The BN was trained offline with learning data from all normal and fault states where one and both faults occurred simultaneously. The data sets used for training were independent and dedicated for training purposes. Each scenario was simulated 30 times using ns-2. The accuracy metric was calculated as one mean of all estimates from all 30 runs, whereas the remaining metrics are averages per run. In general, the OT approach may have a slightly smaller amount of estimates than the BN in a run (cf. Section 4.5.4).

Both the BN and the OT approaches were evaluated based on the same set of observations provided by observation processing. Observation processing window sizes were fixed to ensure comparable reactivity times. The size of the window for the mean estimators is set to 300 ms equally for all observations. Fixing this value is done to decrease the size of the parameter space and ensure that state estimation is provided within hundreds of milliseconds. It should be noted that the value of the window size influences the performance of the BN based estimation significantly [6]. Collecting observations and performing state estimation is done periodically. The choice of sampling period is determined by the dynamics of the observations. In this work, sampling is done with a sampling period of $T = 100\ ms$. From early simulation and test results, this setting proves reasonable to capture the dynamics of RTT and retransmission ratios. The simulation results are listed in Table 4.5.

As seen from *accuracy*, the approaches were similar in performance when estimating the route state, and the BN was more accurate when estimating the wireless link state. In the following we consider the cases where the same

	Acc. [%]	**RT [ms]**	*Scenario*
OT	87 (±1.6)	413 (±102)	1
BN	86 (±0.6)	397 (±30)	
OT	73 (±1.4)	323 (±83)	2
BN	77 (±0.7)	323 (±30)	

Table 4.5: Performance results (Ci=95%).

observations yielded different diagnoses outcomes for the BN and the OT, first illustrated by an example comparing a single estimation case.

An observation of RTT state equal to 4 yields R = *congested* from the OT due to the definition of γ_{rtt}. As the OT does not use the additional observations, this holds for all evidence where RTT state is 4. The BN, however, makes use of all observations. For the evidence state vector $e = [RTT = 4, PRR = 1, FRR = 0]$ a FRR in *low state* suggests that the observed medium packet loss is caused by R = *congested*. The BN makes this conclusion in correspondence with the OT. However, for $e = [4, 1, 1]$ (high FRR) the BN estimates R = *not congested* as the medium packet retransmission ratio is more likely to be caused by observed events on the wireless link. This illustrates the impact of additional, unrelated observations. The BN is said to *explain away* the RTT observation [4].

In both scenarios, multiple evidence vectors were observed that caused a estimation contradiction between the BN and the OT. We investigated how often this evidence occurred and the amount of true estimates in relation to false estimates for the BN and correspondingly the OT. The results for the *poor wireless link* scenario are shown in Figure 4.12 with "-1" corresponding to a missing observation in the evidence (i.e. the moving average window was empty at sample time).

From the figure we see that in most cases the 3 evidence vectors that cause contradicting estimates help the BN to estimate the true state (more white than black space). For instance, in two of these cases the BN makes use of low RTT observation ($< \gamma_{rtt}$) to infer that a high PRR must be caused by a poor WL although a low FRR is currently observed. Altogether, these results explain the increased wireless link state estimation accuracy in Table 4.5. We, however, also see that the vectors do not occur very often, only 6.3% in total (568 out of 9012 cases). Clearly, it is the occurrence probability of these vectors that determines the improvement of the BN over the OT.

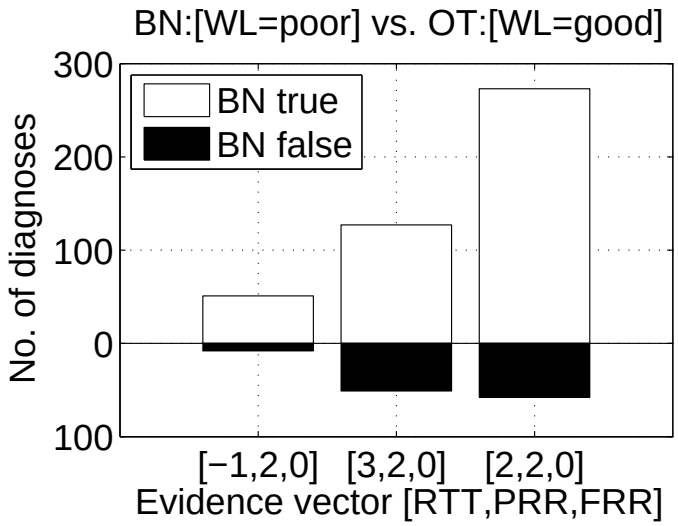

Figure 4.12: Occurrences of e that lead to contradicting *wireless link state* diagnoses.

Considering similarly the *congested route* scenario, the occurrence of the contradicting evidence vectors is 1.8% which is also indicated by the similar accuracy for the BN and OT. Here, only the two vectors $e = [3, 2, 0]$ and $e = [4, 1, 1]$ lead to contradicting estimates. The vector $e = [3, 2, 0]$ was also seen in the *poor wireless link* scenario which means that the BN estimates both R = *congested* and WL = *poor*. While a scenario with both parameters causing throughput drop has not been studied it has been included in the training set. These results suggests this evidence set is likely when both state-changes occur simultaneously.

The contradicting cases show that the constructed BN based on its structure and leaned probabilities is capable of using additional observation to improve estimation performance. Influence of additional observations in the BN is obtained when RTT is in the range of 2-4 and some medium or high FRR is observed. In other, and clearly a dominating amount of cases, low or high observations of RTT and FRR *alone* are strong enough to make correct estimates and the added observations provide no additional useful (or harmful) information for the BN.

Robustness evaluation results

The probability of evidence occurring is clearly dependent on the networking environment of the estimation component. Commonly, network properties may change due to mobility, long term changes in the infrastructure network traffic load etc. To introduce such changes, the delay on the link *R1*↔*R2* (cf. Figure 4.10) was changed in the *congested route* scenario (2.1). The delay was respectively increased ($\Delta d_1 = 4.3\ ms$) and decreased ($\Delta d_2 = -4.3\ ms$). These changes were introduced without re-training the BN or OT. Recall that the BN RTT state interval is $4.3ms$, meaning that the mapping between observation value and node state was shifted. Results for the two cases are shown in Table 4.6.

From the table we see that the change degraded the accuracy of both approaches, as expected. Most impact was seen from the increased delay. The probability of false positives show that for Δd_1 OT estimated *R=congested* constantly. Although the BN also produced many false positive, it was less than the OT. This illustrates that the BN is more robust than the OT to network changes, due to the use of additional observations. However, as emphasized in the previous section, RTT observations strongly influence the BN estimation performance as well. A similar tendency is indicated when the delay is decreased. The results demonstrate how the multiple observations also can be useful to increase robustness to changes from the learned network conditions. Moreover, they show the basic BN as sufficiently general to be used in other scenarios. Increasing BN complexity may yield better performance but may lower robustness as the BN becomes more specific to the learned scenario. Investigating this trade-off is left for future research.

Missing Observations The BN differs from the HMM and OT in one other important property; it can utilize prior belief and additional observations to make an estimate even when observations in the evidence are missing. The OT, only using one observation, cannot provide an estimate without an observation

	Acc. [%]	*Scenario*
OT	49 (±0.35)	Δd_1 (1)
BN	52 (±0.004)	
OT	79 (±1.4)	Δd_2 (1)
BN	80 (±0.7)	

Table 4.6: Performance results (Ci=95%).

sample. Potentially, the previous estimated value could be used again or the normal state could be selected by default. In this work no assumptions have been made on what would be the best approach, as this may be dependent on the cost of making a wrong estimate. Consequently, cases where the OT does not lead to an estimate have simply not been included in the statistics of Table 4.5 and Table 4.6. This may be disadvantageous to the BN in terms of accuracy. In our results, nonetheless, the BN performance benefits from the property. This is seen in both performance scenarios, where the BN estimates route congestion correctly in 130 out of 150 cases without an RTT observation, and poor wireless link in 6 out of 7 cases without an FRR observation.

4.6 Conclusion

Automatic migration triggering depends on having access to information about the network state. The network state is not always directly observable, and therefor solutions to infer the network state from observable parameters have to be developed in the migration platform. The state estimation function in the context management component was developed to obtain and process observations. From the processed observations two model-based estimation methods are used to estimate states of hidden context parameters in the network.

A hidden Markov model (HMM) was adapted to the migration framework and used to demonstrate how the estimation function can operate. It was shown how to measure estimation performance in the form of estimation accuracy. The designed HMM is used in the subsequent chapters as an integrated part of the migration framework.

As one step further in the investigation, we propose to use a Bayesian Network that is capable of inferring the state of multiple network parameters from multiple network observations. We presented how a BN model of a heterogeneous communication network enables state estimation of a congested router and a poor wireless links hidden in the network. The estimation based on multiple observations of round-trip time, packet and frame retransmission ratios. The model uses discrete observation statistics and does not consider temporal relations between faults and/or observations. Through comparison to a simple estimation heuristic, using an optimal threshold on a single observation, the effects of having multiple observations was demonstrated. Improvements in accuracy are seen when estimating wireless link state. Additionally, the BN's ability to handle missing observations was discussed. Finally, the BN

was shown to be more robust to deviations in the network as an effect of using multiple observations.

As a general abstraction for evaluating the migration performance, the estimation methods could be abstracted into the described estimation performance parameters. In this manner, a general function could model the estimation methods in migration framework by their estimation properties. This would enable evaluation of the migration performance benefits and tradeoffs of both estimation methods.

References

[1] E. Altman and T. Jiménez. Ns simulator for beginners. Technical report, Univ. de Los Andes, Mérida, Venezuela and ESSI, 2003.

[2] D. Heckerman. A tutorial on learning with Bayesian networks. Technical report, Microsoft Research, Advanced Technology Division, March 1995.

[3] CS Hood and C. Ji. Probabilistic network fault detection. *Global Telecommunications Conference*, 3, 1996.

[4] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag New York, Inc., 2001.

[5] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002.

[6] A. Nickelsen and J. Grønbæk. Probabilistic fault detection in network communication. Technical report, Aalborg University, May 2006.

[7] A. Nickelsen, J. Grønbæk, T. Renier, and HP Schwefel. Probabilistic Fault-Diagnosis in Mobile Networks Using Cross-Layer Observations. In *Proceedings of AINA*, volume 9, pages 225–232, 2009.

[8] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, California, 1988.

[9] KW Przytula and D. Thompson. Construction of Bayesian networks for diagnostics. *Aerospace Conference Proc.*, 5, 2000.

[10] K. Salamatian and S. Vaton. Hidden markov modeling for network communication channels. *ACM SIGMETRICS Performance Evaluation Review*, 29(1):101, 2001.

[11] M. Steinder and A.S. Sethi. The present and future of event correlation: A need for end-to-end service fault localization. *World Multi-Conf. Systemics, Cybernetics, and Informatics*, pages 124–129, 2001.

[12] M. Steinder and A.S. Sethi. Probabilistic fault localization in communication systems using belief networks. *IEEE/ACM Transactions on Networking (TON)*, 12(5):809–822, 2004.

[13] M. Suojanen, S. Andreassen, and KG Olesen. A method for diagnosing multiple diseases in MUNIN. *IEEE Transactions on Biomedical Engineering*, 48(5):522–532, 2001.

5

Automatic migration trigger management

This chapter presents the challenges of making trigger decisions in the migration framework. Decisions are made periodically about target configurations and orchestration methods. The decisions are made based on the estimated network state, described in Chapter 4. We propose a decision framework for making decisions based on estimated network state and investigate two different approaches to decision making. One, instantaneous approach that maximizes the immediate user experience of the application and one, model-based that takes the effect of future decision into account to maximize long-term application experience. Their performances, in terms of average long-term user experience quality, are compared, and the effect of estimation inaccuracy on migration performance is investigated.

5.1 Motivation

Automatic triggering is necessary in situations where the user cannot trigger migration manually. This can be the case when there is only a short window of time to make the decision, for instance, in a mobility scenario, where the application fails if it is not migrated before the user moves out of network range. Another case can be when the decision space is too complex for the user to handle, and the best choice is not obvious to the user from the available context information. In this chapter, we investigate how to generate automatic migration triggers based on the state of the network with the aim to automatically migrate to configurations that improve the user's experience.

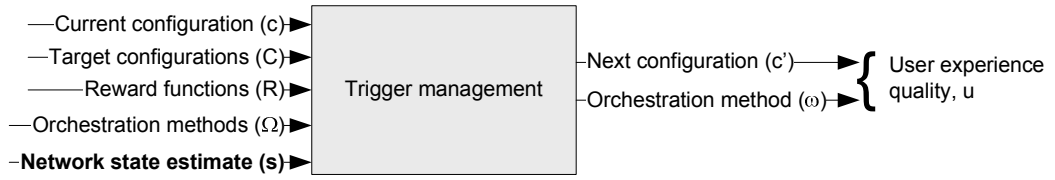

Figure 5.1: Input and output of the trigger management component in the migration framework.

The challenges of the trigger management component are illustrated in Figure 5.1. When running, the component receives as input a set of configurations, C, reward functions for the configurations, $R(c, s)$, a set of orchestration methods, Ω, the current configuration, c and the estimated network state, $\hat{s}$. From this input, the task of the component is to choose the next configuration c' and the orchestration method, ω. The performance of the component is defined by the average quality of the user experience, u, of the chosen configurations in the dynamic network.

The network state is dynamic and changes influence which configuration is currently the best. Also, orchestration of the migration process takes time and interrupts the user's work-flow while in progress. The triggering function must balance the trade-off between choosing configurations that provide improved user experience and are fast and successful to avoid waiting on migration. Also, as described in Chapter 4, the network state is not always directly observable, and the triggers are therefore based on the estimated network state. If the state estimation is inaccurate, unnecessary migrations may be triggered, which interrupts the user and may end up in a worse configuration than originally.

The decision making part of the component is achieved by decision policies. A decision policy, $c', \omega = \pi(c, s)$, defines the best next configuration given the

current configuration and network state. We use two different approaches as case studies:

- a greedy, instantaneous approach (INS) that optimizes the user experience until the next decision is made
- a Markov Decision Process (MDP)-based approach which predicts the effect of future decisions and optimizes for the long-term average experience quality.

Based on a simulated migration scenario, we evaluate the effect on the average experience quality of: 1) the estimation approach, 2) the policy generation approach and 3) the network dynamics. The estimators are based on the threshold and Hidden Markov model approaches from Chapter 4. From the evaluation we analyze the dependencies between the performance of the decision framework and the estimation approach, the policy generation approach and the extent of the interruption of the migration procedure. Through analysis of the performance results, we illustrate the usefulness of taking future decisions into account as done by the MDP approach.

In this chapter we assume only one orchestration method in Ω, called ω_0. In Chapter 7 we use a set of multiple orchestration methods up to ω_K that can be dynamically available.

5.2 Design of trigger management

The trigger management component contains two functions to trigger migration automatically based on decision policies; policy generation and policy enforcement, as illustrated in Figure 5.2. Policy generation is used before the migration platform runs, and the policy enforcement is used to enforce the generated decision policies when running the platform.

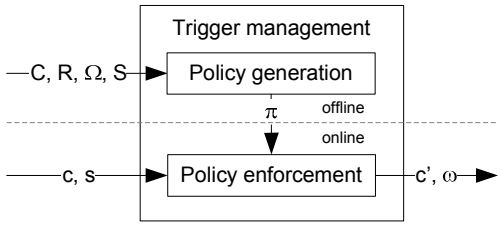

Figure 5.2: Overview of the functions of trigger management.

5.2.1 Policy generation

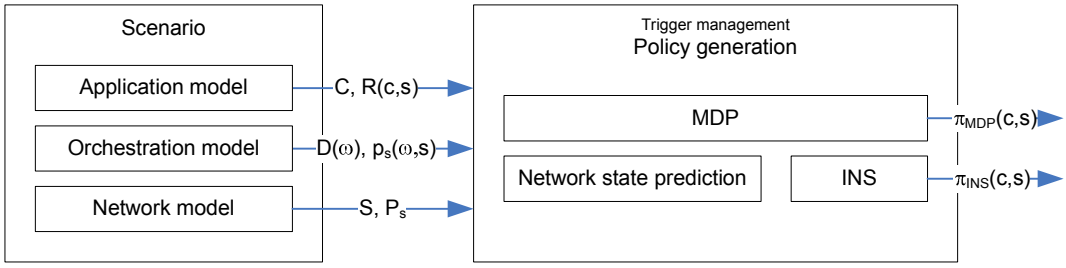

Figure 5.3: Policy generation function of trigger management. In this work, the generation step is done offline, before running the migratory system.

The policy generation function is illustrated in Figure 5.3. The function takes the migration model parameters defined in Chapter 3 as input. Then it generates a decision policy to be used when the migration platform is running. The decision policy is generated before the system is running, also refered to as *offline* generation. We use a decision policy that jointly determines both target configuration and orchestration method. This enables the choice of either parameter to influence the other and the function is thereby able to generate the combined optimal choices of both parameters.

The two policy generation approaches evaluated in this work are examples. The MDP method predicts future decision based on a predicted network state. The INS methods does not predict the network state inherently since it does not consider future decisions. We include a prediction step in the INS method to allow the method to optimize for the predicted network state after the migration delay.

5.2.2 Policy enforcement

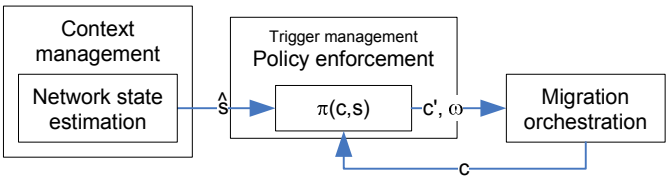

Figure 5.4: Policy enforcement function of trigger management.

The structure of the policy enforcement function is illustrated in Figure 5.4. To enforce any decision policy online, the enforcement function needs to know

the policy, the current configuration of the application c and the estimated network state $\hat{s}$. When the system is online, the network state, $\hat{s}$, is estimated in the *state estimation* component delivered by the context management component.

5.2.3 Reward

The quality of the user's experience is modeled by reward functions that map physical properties into quality of experience. Derivation of reward functions has already been covered in detail in existing literature, so this is not covered in this work. Examples of reward functions mapping network properties into quality of multi-media applications can be studied in [4] [2] [5]. Reward is defined as a combination of the satisfaction of experiencing a certain application quality and the dissatisfaction of the interruption from migration.
Reward of an application configuration in a certain system state is characterized by $R(c, s)$. $R(c, s)$ is given by the application developer as a distribution of reward of the network state space.

The dissatisfaction is characterized by two parameters of an orchestration method ω; the migration success probability, p_s and the migration delay, D. These two parameters influence the user experience negatively and can therefore be used to characterize how performing a migration should be penalized. We represent the penalty of migration by a function of the waiting time $\eta(D)$. The function can have different shapes (linear, logarithmic, quadratic, exponential) representing different user dissatisfaction patterns. In our work, we define the function linearly as $\eta(D) = -\alpha \cdot D$, where α is a constant and represents the reduction in reward per time-step experienced by the user while waiting on the migration to finish. A linear function is simple to include in the mathematical models, and represents a fairly realistic user behavior in terms of dissatisfaction when waiting for migration to complete.

5.3 Policy generation functions

We evaluate and compare the performance of the different estimation methods and policy generation approaches by simulating the migration framework. First, we systematically analyze the differences between two policy generation methods to learn where they are different, and how the differences impact the decision quality. Then, we analyze two example scenarios of the entire framework to learn how the different parameters of the full framework impact the

decision quality. We evaluate the three state estimation methods of Chapter 4 methods combined with the two policy generation approaches in two different scenarios, which reflect two different network conditions.

Since only one orchestration method is considered in this chapter, such that $\Omega = \omega_0$, the decision polices do not include choice of w.

5.3.1 Instantaneous decision policy generation

The instantaneous approach chooses the configuration c' that generates the highest expected reward for the predicted state estimate after the migration $c \to c'$, such that

$$\pi_{INS}(c, \hat{s}(t+D)) = \underset{c' \in C}{\operatorname{argmax}} E[r(c, c', \hat{s}(t+D))]. \tag{5.1}$$

Because of the migration delay, D, the state estimate at time $t+D$ is predicted based on the currently estimated state to allow the policy generation methods to account for migration delay. Recall, D is the random migration delay in discrete time-steps distributed according to the probability mass function $f_{D,c\to c'}(\tau)$. We denote the predicted state estimate after a migration that lasts for D time-steps as $\hat{s}(t+D)$. To predict the most likely estimated state at time $t+D$, the behavior of the Markov model (that models the network behavior) is used, such that

$$\hat{s}(t+D) = \Pi \cdot P^D$$

where Π is an 1xN vector, where all elements are zero, except element i that is equal to one if s_i is the most likely state.

The reward expression $r(\cdot)$ represents the reward of migrating from c to c' when the predicted state after migration is $\hat{s}(t+D)$. The expression includes the migration success probability and migration delay as follows

$$\begin{aligned} &E\left[r(c, c', \hat{s}(t+D))\right] = \\ &\quad p_s \sum_{\tau=0}^{\infty} \left[R(c', \hat{s}(t+\tau)) - \eta(\tau)\right] f_{D,c\to c'}(\tau) + \\ &\quad (1-p_s) \sum_{\tau=0}^{\infty} \left[R(c, \hat{s}(t+\tau)) - \eta(\tau)\right] f_{fail,c\to c'}(\tau) \end{aligned}$$

The INS approach is *greedy*. It is also conservative, in the sense that it maximizes the reward for the immediate state after migration (positive), but accounts for all migration penalty during migration (negative).

5.3.2 Markov Decision Process (MDP) decision policy generation

The MDP approach can calculate an optimal decision policy, $\pi_{MDP}(c, s)$. The MDP approach predicts the effect of future decisions on the long-term average reward and generate a policy with the decisions that optimize that. We use value iteration with a finite window of 1000 future decisions to find the optimal policy. The window size value was found through experiments and is large enough for the generated policies to converge, meaning that a larger window will only affect computation time, but not change the policy. See Chapter 2 or [3] for more details on the MDP framework, dynamic programming and value iteration.

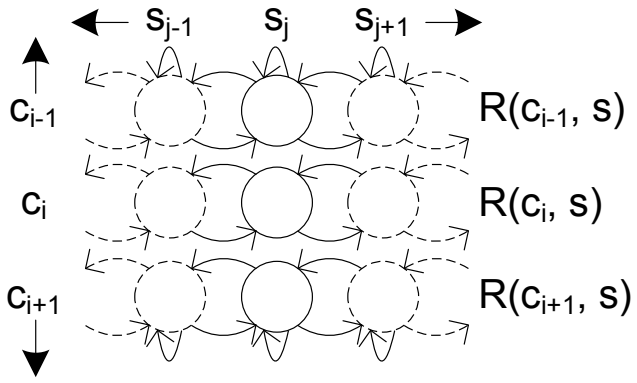

Figure 5.5: A model of the system without the ability to migrate between configurations.

The general form of the Markov model used in the MDP approach is depicted in Figure 5.5. For each available configuration a reward function $R(c, s)$ is assigned that maps a reward value to each network state for a given configuration. There exist different reward functions for the different configurations.

In Figure 5.6 the scenario of migrating between configurations $c \rightarrow c'$ is depicted (with one possible orchestration method). The actions of the MDP map directly to the available configurations, which can be chosen at each epoch. To model the orchestration process, an intermediate delay state is introduced between the configurations. When the migration is triggered, the

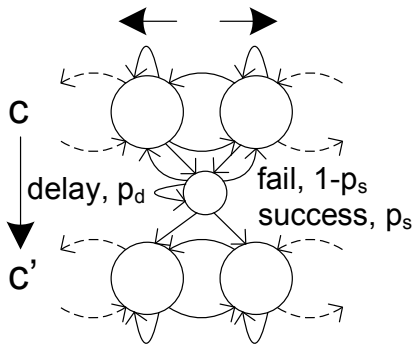

Figure 5.6: Model of the migratory system which shows how a migration $c \rightarrow c'$ may change the active configuration and thereby the reward function if successful.

system enters the delay state with probability 1. The reward model of the delay state is defined by $\eta(D)$, which means that in the linear case of this framework, the reward of the delay state is simply α. In the delay state the migration may be further delayed with probability p_d, fail back to c with probability $(1-p_s)\cdot(1-p_d)$ or succeed to c' with probability $p_s \cdot (1-p_d)$. The definition of p_d is based on the migration delay D. When D is geometrically distributed then $p_d = 1 - \frac{1}{\bar{D}+1}$, where $\bar{D}$ is the mean migration delay.

5.4 Case studies and numerical results

5.4.1 Simulation model

The network is modeled as a network with two states, representing a normal and a congested network, $\mathcal{S} = \{N, C\}$. P, the transition behavior within $\mathcal{S}$, is specified by p, the probability of a transition from N to N, and q, the probability of the transition from C to C, as specified in Table 5.1. The application is a video streaming application with two available configurations on two difference devices; D1) high definition on large display device, and D2) standard definition on mobile device with smaller screen), $\mathcal{C} = \{D1, D2\}$. The reward values of the two configurations, R, are listed in Table 5.1. These values express that the user gets a higher perceived quality with increasing resolution on the large display device (i.e. R=4 for (c,s)=(1,1) compared to R=3 for (c,s)=(2,1)). Moreover, the impact of packet loss is larger when $c = D1$ than $c = D2$ because the higher resolution requires more available bandwidth on the large display.

For migration penalty values, we use $p_s = 0.9$ and use a geometrically distributed delay D with parameter $p_d = 0.95$. This characterizes a mean migration delay of 2 seconds, when time-unit per time-step is 100ms (equal to $\bar{D} = 20$). The success probability and delay mean are sampled from experimental work on migration prototypes, described in detail in [1].

For the dissatisfaction function $\eta(D) = \alpha \cdot D$, we use $\alpha = 0.01$, which has been determined a reasonable value through experiments, as it depends on a combination of the scale of the rewards and the mean delay. The parameter values are summarized in Table 5.1.

For network state estimation, we use the observation models and HMM state estimator as described in Chapter 4.

block	*description*	*name*	*value*
network model	transition probabilities	p	0.01...0.99
		q	0.01...0.99
policy generation	reward	R(c,s)	$\begin{bmatrix} 4 & 1 \\ 3 & 2 \end{bmatrix}$
	waiting penalty	α	0.01
	mean delay	$\bar{D}$	20
	success probability	p_s	0.9
MDP	window size	w	1000

Table 5.1: Model parameters of the different components used in the evaluation.

We generated policies based on the different models and for each policy simulated 30 runs of the system model in 10000 steps and obtained sequences of configurations, of true and estimated network states $\hat{s}$. From this the overall user experience quality was calculated using the reward functions.

5.4.2 Policy generation approach impact on policies

To understand the differences between the INS and the MDP approached, we analyze which policies they generate under same conditions. We use the full parameter spectrum of the network states, defined by p and q in P (cf. Table 5.1), since it represents the scope of network parameters of the example that characterize external, uncontrollable network state behavior. In order to analyze the relative performance of the methods we compare the MDP and INS approaches in the full spectrum. We use a forward prediction step in the

INS approach for the comparison, to be able to use p and q in both policy generation approaches.

Figure 5.7 plots indexes (of the 16 uniquely generated policies) as z-values over the full spectrum of p and q for both approaches in the case of ideal migration ($\bar{D} = 0, p_s = 1$). There exist many regions in the parameter space where the policies are not equal. As an example, z=1 is the policy

$$\pi(c, s) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

which means that $c' = 1$ is chosen for any configuration (vertical) in any network state (horizontal).

With INS (Figure 5.7, top), the regions are divided into four quadrants at $p = 0.5$ and $q = 0.5$. At the upper left (z=16) and lower right (z=1) quadrants, the policies specify to go and stay in D2 or D1 configurations, respectively, indifferent of the system state. To the lower left (z=11), the policy specifies to choose D1 configuration when in congested network state and D2 when in normal network state. The upper right region (z=6) chooses opposite to the lower left.

With the MDP (Figure 5.7, lower), the regions are separated by the line $p = q$. Below this line ($p > q$), the decision is to either to change to D1 always (z=1) or in some cases (z=2) to stay in D2 when in congested network state. Above the line ($p < q$), several different policies are used that will eventually all change to D2 always (z=8,12,16). In the small region ($p > 0.9$ and $q > 0.9$) the MDP behaves similar to the INS by choosing D1 in normal network state and D2 in congested network state (z=6).

The differences in the overall policy distributions is due to the way the two policy generation methods optimize their policies. Even though they both optimize for utility, the INS methods only regards one future decision, whereas the MDP regards many. The different distributions show that it has an impact on the policy whether the future decisions are considered.

Figure 5.8 shows the policies over the full spectrum of p and q for both approaches in the case of non-ideal migration ($\bar{D} = 20, p_s = 0.9$). The INS policy distribution clearly changes compared to ideal migration. Instead of 4 quadrants, now 9 regions exist, with the original 4 regions still existing though smaller. In a square-formed region in the center, a new policy is seen (z=4), which is the policy that chooses to stay in the current/initial configuration at all times. The additional 4 new policies between each pair of former quadrants are combinations of the original policy pairs. For example, this means that

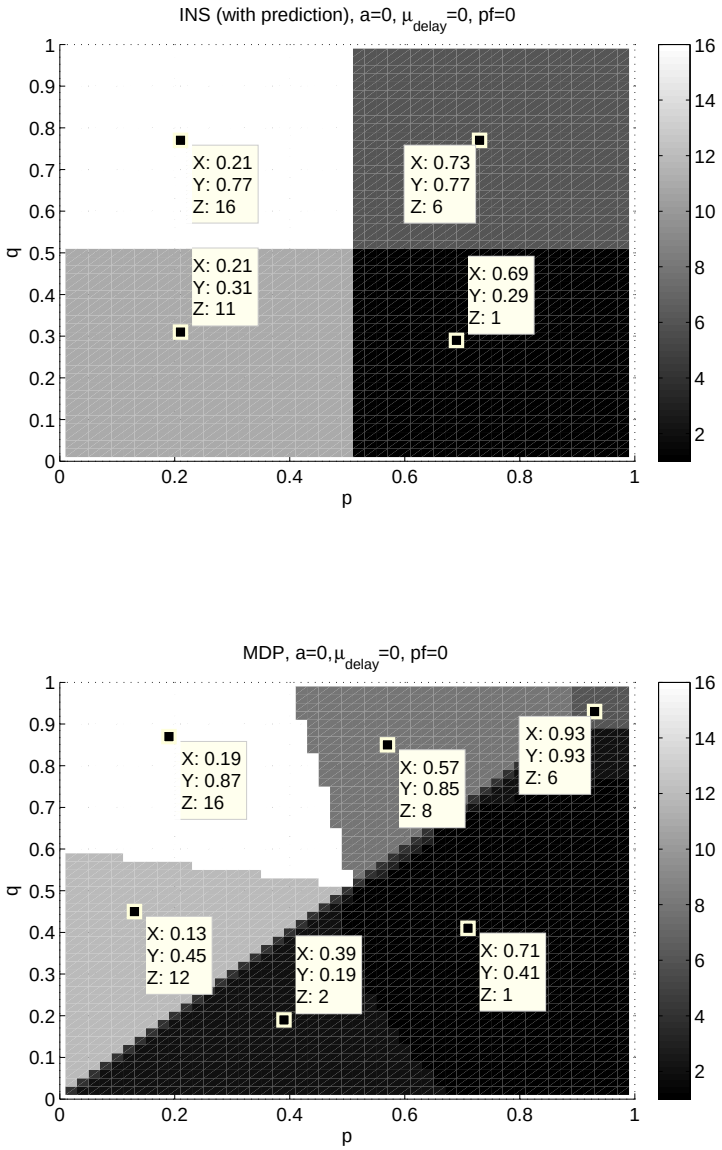

Figure 5.7: Decision policies generated by INS and MDP approaches for different network parameter settings with zero migration penalty.

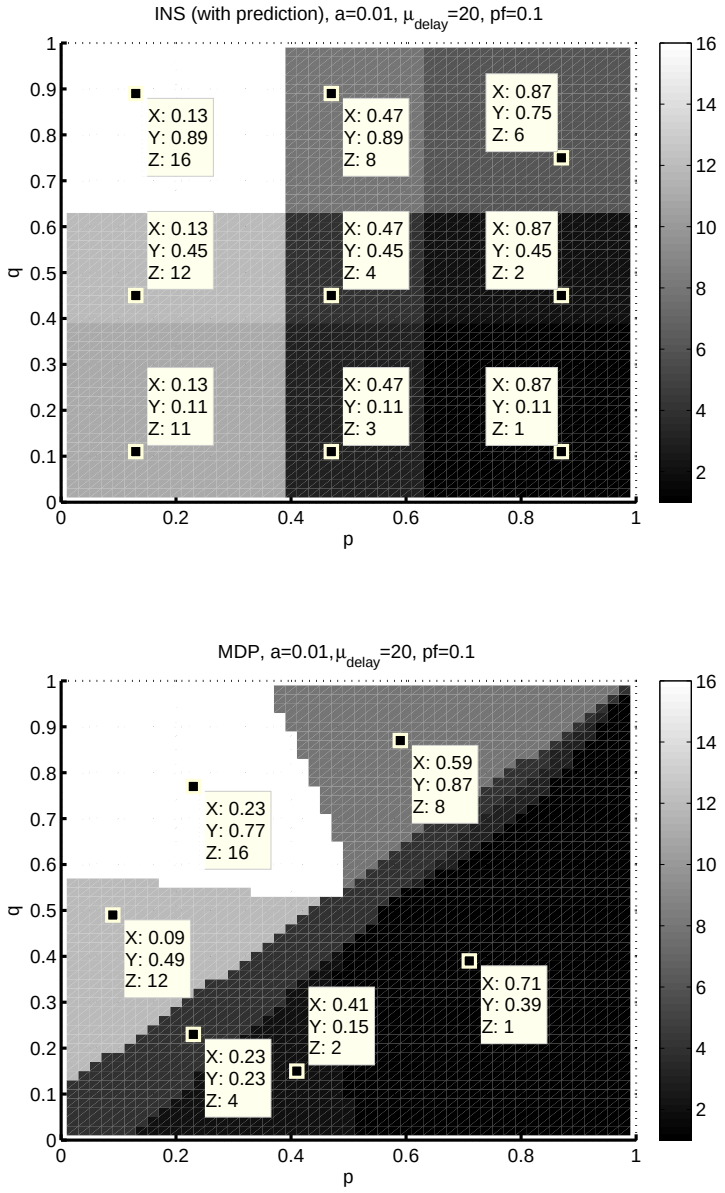

Figure 5.8: Decision policies generated by INS and MDP approaches for different network parameter settings with non-zero migration penalty.

the policy in the middle lowest square (z=3) combines choosing D1 always (the policy to the right, z=1) and choosing D2 in normal network state (the policy to the left, z=11) in the way that it chooses D1 always except if in D2 and normal network state. The MDP policy distribution is very similar to the non-ideal case, except that the small region upper right is gone and there is a new region around $p = q$, that contains the same policy as the center square (z=4) in the INS figure, namely not to change configuration when network state changes. The no-change policy is found in both INS and MDP distributions when the migration is non-ideal which can be explained by the added success probability and long migration delay, which decreases the expected average utility that both methods optimize for. In the border-line regions of both cases, a higher penalty means a higher probability of decreased utility when migrating, and as a consequence, migration is not chosen in the specific cases.

5.4.3 Policy generation approach impact on performance

To understand the impact on the quality of the different policies, we simulated runs of the decision framework using all policies from both approaches in the entire parameter space, for ideal (zero penalty) and non-ideal migration respectively. For each approach we calculated the average utility in each simulation with a specific policy, and calculated means over the repetitions. In these simulations we used an ideal state estimator. The difference in mean average quality between the approaches at each point (p, q) is shown in Figure 5.9 for ideal migration and in Figure 5.10 for non-ideal migration. Black means no difference (because INS and MDP policies are the same) and the brighter the tone, the larger the difference. The range of average utility varies between 1.6 and 4 for the different points (p, q). We repeated simulations to obtain significant differences within 95% confidence intervals.

In the ideal migration case of Figure 5.9, we see that in two regions ($p < 0.5$ and $q < 0.5$) and ($p > 0.5$ and $q > 0.5$) the policies of the MDP approach generate higher average utility values than the policies of the INS approach. In the non-ideal migration case, as shown in Figure 5.10, differences are detected in 2 regions; in the lower left and upper right regions. The corner regions are similar to those in ideal migration, but smaller in area and with larger average utility differences.

The evaluation results from the example show that in some cases the use of the MDP approach gives an advantage. The advantage depends on the penalty of the migration, which is a key property of the MDP method.

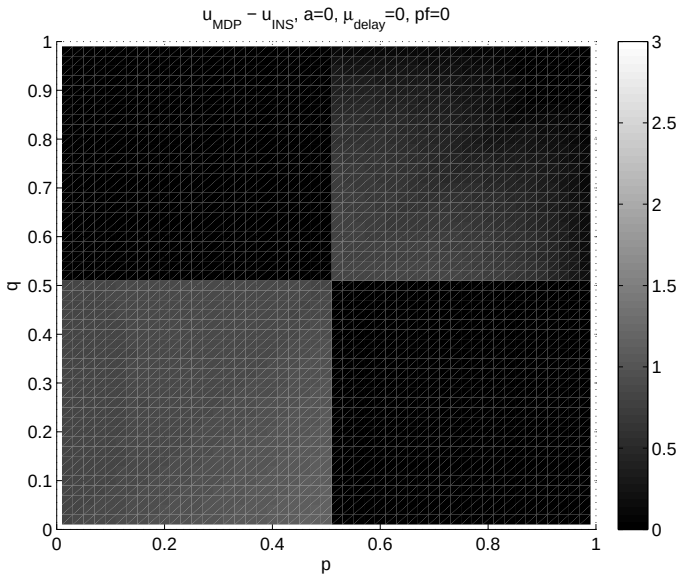

Figure 5.9: Average utility difference between policies generated by INS and MDP for different settings of p and q, with zero penalty and ideal state estimation.

5.4.4 Inaccurate state estimation impact on performance

We evaluate the performance of the combined state estimation and decision algorithms by calculating the average quality achieved during a simulation run and the number of configuration changes. We use the two specific scenarios of (p, q) described in Chapter 4 to be able to compare performance between approaches.

In scenario 1, we simulate a network that has a low rate of state changes. In scenario 2, we simulate a fast-changing network with a high rate of changes. The differences between the two networks are expressed in the state transition probabilities. In scenario 1, the probability of changing state is very low (0.1-0.5 changes per second on average), which is seen in Table 5.2. In scenario 2, it is opposite, such that the probability of changing state is very high (9.5-10 changes per second on average), as seen in Table 5.3.

We use both scenarios for evaluation to illustrate the robustness of the model-based approach. To get a clear understanding of the impact of state estimation inaccuracy, we used the instantaneous approach without state prediction, denoted INS', in this evaluation.

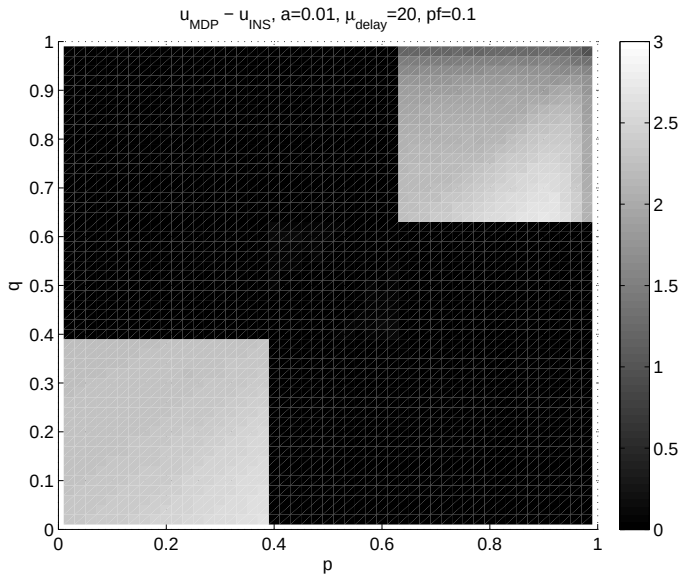

Figure 5.10: Average utility difference between policies generated by INS and MDP for different settings of p and q, with non-zero penalty and ideal state estimation

$s(t) \to s(t+1)$	N	C
N	0.99	0.01
C	0.05	0.95

Table 5.2: Transition probabilities $P_1(s(t), s(t+1))$ (scenario 1) (copy of Table 4.1)

$s(t) \to s(t+1)$	N	C
N	0.01	0.99
C	0.95	0.05

Table 5.3: Transition probabilities $P_2(s(t), s(t+1))$ (scenario 2) (copy of Table 4.2)

The policy of the instantaneous approach is the same in both scenarios as it only depends on the reward distribution and not the transition probabilities. The policy is defined as

$$\pi_{INS'}(c, s) = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}.$$

The interpretation of the policy is that if normal network state is observed (column 1) then $c' = D1$ is decided. If high packet loss is observed (column 2) then $c' = D2$ is decided.

The optimal policy generated by the MDP in scenario 1 is

$$\pi_{MDP,\text{s1}}(c, s) = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}.$$

The output of this policy is equal to that of the instantaneous approach, as it will choose the configuration with the maximum quality for each network state. This can be seen when comparing to the reward-values in Table 5.1. For scenario 2, the MDP generates a policy opposite to the previously used

$$\pi_{MDP,\mathrm{s2}}(c,s) = \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}.$$

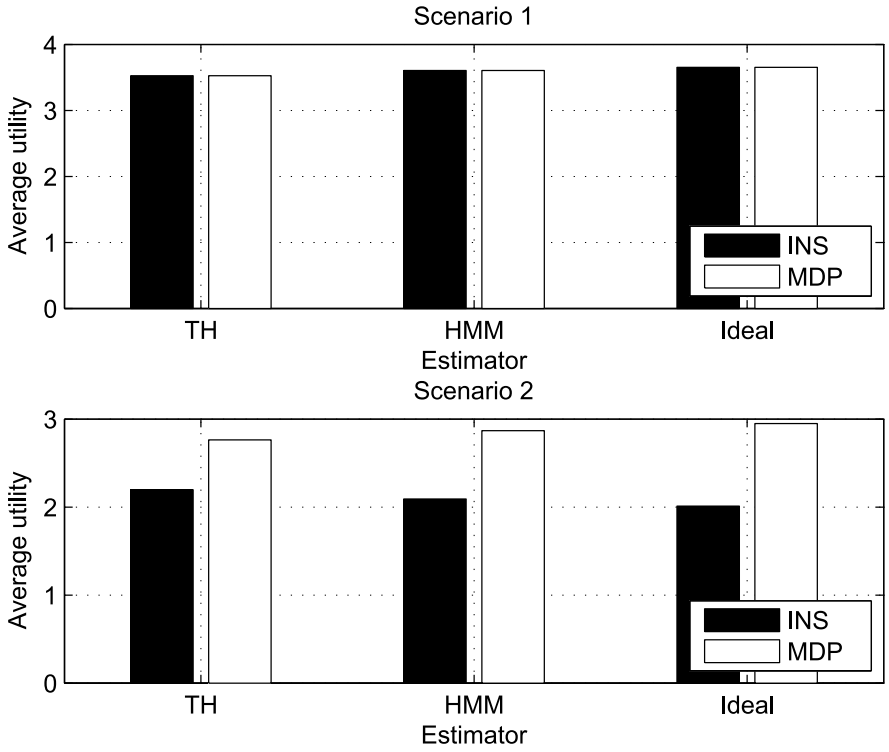

Figure 5.11: Average utility comparison between the three estimation methods and the two policy generation methods in both network scenarios.

Based on the sequences of chosen configurations and true states, we calculated the average utility achieved during the simulation runs. The mean average utilities in both scenarios are shown in Figure 5.11, where also the theoretical ideal estimator is shown. The difference are significant in all cases within a 95% confidence interval. In scenario 1, only the state estimation method makes a difference since the policies are equal. In scenario 2, the policies are opposite. Since the MDP policy is optimal, the average quality increases with increasing estimation accuracy. However, the instantaneous policy is opposite the optimal and always makes the worst decision. This explains

the decrease in average quality with the increasing estimator accuracy. The fact, that the INS approach produces the lowest average utility with the ideal estimator in scenario 2 is due to the behavior of the INS policy. In scenario 2, the choices of the INS policy are exactly opposite of the optimal policy and therefore INS performance benefits from any inaccuracy in the state estimator.

5.5 Conclusion

We have proposed a model-based policy generation approach for the migratory platform based on a Markov Decision Process (MDP). The MDP-approach generates an optimal decision policy while considering the penalty of performing the migration (success probability and delay). One key property of the MDP approach is the ability to consider the effect of future decisions into the current choice. With the MDP-approach, decisions are made based on the state of the network. As the state is not always observable, we apply the HMM described in Chapter 4 for network estimation based on observable parameters.

An example system was simulated to evaluate the model-based approach by comparison to a simpler instantaneous approach. The comparison using a model of a slowly changing network showed that the introduction of the HMM alone gives benefits, as the average quality achieved during was slightly higher for the MDP-approach than the instantaneous approach. As the network model changed to include more rapid changes, the MDP-approach also performed better than the instantaneous approach. Also here the MDP-approach produced the highest quality and followed the dynamics of the network more precisely. The average user experience quality of the policy generation approach was compared to a simple instantaneous approach that does not consider future decisions. With our results we are able to quantify the gain in performance of considering future decisions.

References

[1] K. Højgaard-Hansen, H. C. Ngyuen, and H-P. Schwefel. Session mobility solution for client-based application migration scenarios. In *To appear in Proceedings of The 8th International Conference on Wireless On-demand Network Systems and Services (WONS2011)*, 2010. Submitted.

[2] C.E. Luna, L.P. Kondi, and A.K. Katsaggelos. Maximizing user utility in video streaming applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(2):141–148, 2003.

[3] M.L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. *IMA Journal of Management Mathematics*, 1994.

[4] H.P. Schwefel, S. Praestholm, and SV Andersen. Packet Voice Rate Adaptation Through Perceptual Frame Discarding. In *IEEE Global Telecommunications Conference, 2007. GLOBECOM'07*, pages 2497–2502, 2007.

[5] S. Shenker. Fundamental design issues for the future Internet. *IEEE Journal on Selected Areas in Communications*, 13(7):1176–1188, 1995.

6

Optimized orchestration over resource-constrained links

In this chapter we analyze how resource-constrained ad-hoc network links between devices can be used in the migration process. Migration over such ad-hoc links is challenged by link properties such as being short-lived and having low bandwidth. We use Near Field Communication (NFC) as a case study and present a design of a lean orchestration protocol that utilizes an NFC link for orchestrating the migration. We present experimental analysis of the NFC throughput performance and study how that impacts the migration performance, measured by migration completion time.

6.1 Motivation

The migration framework assumes to have resourceful network links (Ethernet, WLAN or GPRS/UMTS) and a central server available to orchestrate the migration process (cf. Chapter 3). However, using ad-hoc network links between devices and in particular RFID based Near Field communication (NFC) can be advantageous for multiple reasons: (1) The physical closeness required to create an NFC link can be interpreted as migration trigger; (2) binding the migration to the physical closeness can increase the user-trust in the migration procedure, as hijacking of sessions by remote devices can be prevented; (3) fast connection setup times and low interference probability due to its short range can be advantageous; (4) low power consumption can make NFC the technology of choice in migration scenarios triggered by low battery of the source device. In addition, migrating via the NFC link may be the only option in scenarios of interruption of coverage of mid- to long-range (cellular) technologies.

Migration over ad-hoc NFC links is however challenged by throughput limitations and the potential short time-windows during which the communication is possible. Preliminary experiments have shown that users expect the entire migration process to finish within 3-5 seconds when using NFC. These results are based on scenarios assuming the user is familiar with the use of RFID and NFC, as for instance door locks or ticketing (which require a small window to complete). In the scenarios, the users were not guided as to how the migration process was progressing. Such guidance may increase the allotted window of time for migration, however, this aspect is not investigated further in this work. The consequence is that NFC may only be feasible for small application state sizes, and we propose a protocol targeted at maximizing the feasible state size.

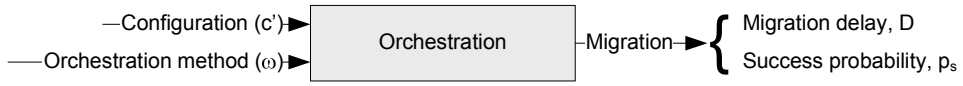

Figure 6.1: Input and output of the orchestration component in the migration framework.

In this chapter, we propose a migration protocol optimized for resource-constrained links, such as NFC. This migration protocol is in general viewed as another orchestration method that can be chosen by the trigger management component as input for the orchestration component, as illustrated in Figure 6.1. Based on a model of the orchestration process, the orchestration component can estimate orchestration performance metrics such as migration

duration and success probability. In this work, we derive the performance metrics experimentally for the proposed migration protocol.

From an experimental implementation of the protocol, we determine the range of state sizes that can be transferred within the available communication windows. From this investigation, we present general performance measurements, which are also applicable in other contexts than service migration. In addition, we determine the potential of using multiple wireless technologies simultaneously for orchestration. By use of the lean orchestration protocol, we compare orchestration scenarios with both WLAN and NFC available, to study if use of two simultaneous technologies provide increased throughput or whether they interfere with each other in the test setup. In the test setup, we did not find frequency interference between the two technologies. However, we did find that the mobile device seems to be challenged by managing two network links simultaneous, causing the aggregated throughput to degrade to lower than when only using WLAN.
As we focus on what performance implications NFC has on migration, a detailed security analysis and investigations of complex user interactions are out of the scope of this work.

We present a migration scenario in Section 6.2 to illustrate how use of NFC links challenges the migration architecture. Performance results of NFC usage are presented in Section 6.5.2 to obtain approximate magnitudes of boundary conditions using NFC for migration. In Section 6.4 the proposed protocol for using NFC in migration is described and implementation and evaluation of the protocol are presented in Section 6.5.

6.2 Scenario and orchestration model

Figure 6.2 illustrates a migration scenario which includes an additional ad-hoc link, compared to the general scenario presented in Chapter 3. The migration scenario from the mobile device (source) to the large display (target) includes the following activities:

1. the user puts the mobile device within NFC connection range of the large display to trigger the migration.
2. the middleware triggers the migration based on the connection event and orchestrates the migration procedure, which includes
   (a) pausing the application on the mobile device

(b) extracting state information

(c) transferring the state from the source device to the migration server

(d) initializing the middleware components on the target device

(e) transferring the state to the target (large display in Figure 6.2)

(f) inserting the state into a new application instance

(g) resuming this application in the original state

3. the middleware terminates the original paused application on the source device.

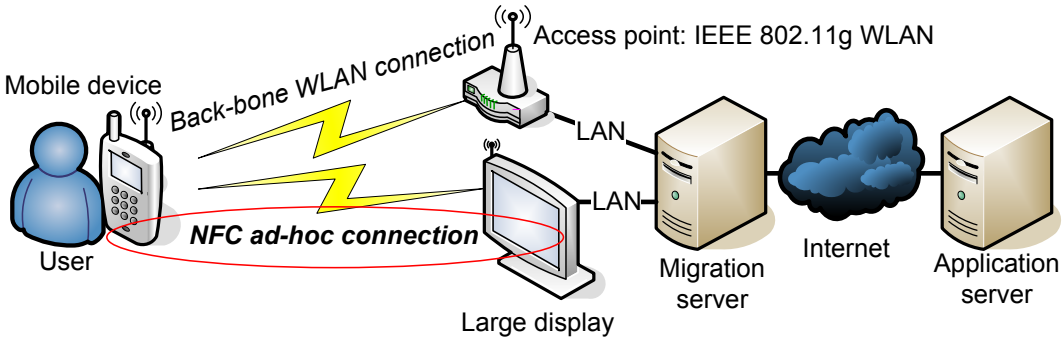

Figure 6.2: A scenario of migrating the client-part of the application from the mobile device to the large display.

In case of a video streaming application, the required state information to be transferred is a URL, a time offset and potentially media relevant information regarding codecs, etc. which were received/negotiated in the beginning of the stream, for instance as a session description protocol (SDP) profile [6]. Because the video stream is not re-initialized when migrated, such information is not re-exchanged between client and server, and must thus be transferred as state. A SDP-profile is exchanged in clear-text, and its size can range from 230 bytes for a compressed video profile [14], over 860 bytes for a raw audio profile [5] and upward for more complex sessions.

6.3 Near-Field Communication background

Traditional RFID communication consists of a passive RFID tag and an active RFID reader. The reader generates a radio frequency (RF) field to request a response from the tag and the tag uses the energy in the reader's RF field

to respond. One NFC entity integrates both RFID tag and reader and thus allows two NFC entities to communicate peer-to-peer. The NFC specification ([4]) allows for entities to work as traditional passive RFID tag and reader for compatibility, however, only the peer-to-peer mode is considered here, in *active* mode, where all entities generate RF fields.

NFC devices must be prepared for neighbor discovery, similar to the inquiry phase of Bluetooth and the frequency scanning phase of ad-hoc mode WLAN. An NFC device can have one of two roles; *initiator* or *target*. The initiator uses its RF field to contact targets. The target only senses for initiator RF fields and does not turn on its own RF field unless requested by an initiator as part of communication. A protocol using NFC must include at least one initiator and one target to have successful communication.

NFC has previously been studied as a part of different application types; for service discovery in Smart Spaces [1], for tracking habits in home health-care [7] or as information carrier in marketing such as in All-I-Touch [9] and [8]. Similarly, NFC performance parameters are mostly user-experience oriented; trustworthiness [11] and usability [2] [12]. Likewise, [13] and [10] look into the security implications of NFC communications.

6.4 Design of lean orchestration protocol

The scenario for the lean orchestration protocol is depicted in Figure 6.2. Some time period before the user puts the mobile device close to the large display to activate NFC neighbor discovery, the WLAN connection disappears so that migration needs to be performed via the NFC link. The lifetime of the NFC link is limited by the window of time the user holds the mobile device close to the large display. The goal of the proposed migration protocol is to maximize the probability that the application state can be successfully transferred within the available time window. In order to maximize use of the time window, the protocol aims at starting the transfer as early as possible. When the devices have discovered each other, the state is prepared in the source device and the size of the state is exchanged at first. After that, the state is transferred. Finally, when the state has been transferred successfully, the application in the source device is terminated.

In a scenario with a back-bone connection (such as WLAN), migration decisions are made by the migration server and therefore clients spend much time waiting before transferring the state due to communication with the server.

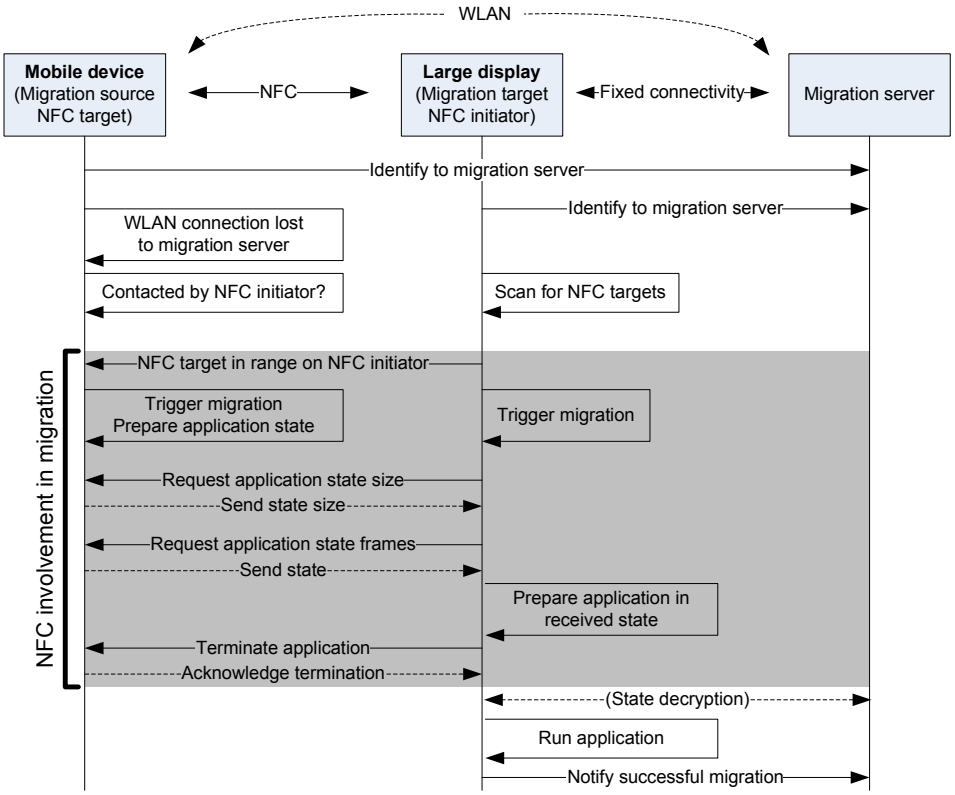

Figure 6.3: Fast and low-overhead migration procedure utilizing the NFC link.

Also, when migration is controlled by the server, the transfer of the state introduces overhead as the transfer is split into an upload from the source device to the server followed by a download from the server to the target device. In our scenario where the WLAN connection has disappeared, there cannot be direct communication between source device and server. This would have to be relayed by the target device, which would be even more time consuming than normally.

In order to optimize the state transfer, delay and overhead must be reduced and interactions between source device and server cannot be performed. This requires that the decision to transfer the state needs to be made solely on the target. Moreover, the amount of signaling messages between the target and the source must be kept low due to the NFC round-trip delay. Our proposed protocol to achieve this optimization is illustrated in Figure 6.3 and works as follows. Both devices are assumed to have registered previously with the migration server via the WLAN connection to establish a trust relationship. The large display is configured as NFC initiator and the mobile device as NFC target. The large display actively searches for the mobile device. When the user *swipes* the mobile device close to the large display to trigger migration, the NFC initiator detects the NFC target and triggers the migration procedure on the large display.

The large display requests the application state size from the mobile device. To deliver this result, the mobile device must know which application to migrate and its state size. Several selection schemes can be employed, such as choosing the application which has the active/focused window, or the user can have selected an application to migrate manually before the swipe, e.g. when notified about the loss of connectivity. The assumption here is that the mobile device knows which application to migrate and that the application state size is known. The large display then downloads the state object from the mobile device via the NFC link. Once the download has finished, the large display sends a termination command to the mobile device, which terminates the original application. To activate the application on the large display the application must be resumed in the downloaded state. In summary, our lean protocol relies on decentralization of decisions from the server to the migration target. The major decisions that have been decentralized are:

- Trigger the migration when 'NFC-in-range' event is detected
- Use NFC for the migration protocol and state transfer
- Use pre-defined rules or user-interaction to allow the source device to identify which application to migrate

As the role of a device in the NFC communication must be set before communication, a role selection algorithm must be in place. Selection could be based on static rules, where all devices select a certain role or have pre-assigned roles. The rule in a given scenario must ensure that roles are distributed in order to allow communication. The selection could also be based on a random-hopping scheme, where devices hop between the two roles and listen for presence of opposite roles; initiators request target responses and targets listen for initiator requests. Due to the scarceness of time in the scenario, we employ a static selection decided by the device type: Mobile devices are NFC targets and static devices are NFC initiators. The rationale is that mobile devices are typically power-constrained where as static devices are assumed to have an external power supply is in the case of the large display. As the target role requires less power than an initiator the target role is assigned to the most power-constrained device.

6.5 Experimental results

This section presents an experimental performance study of an NFC link in order to obtain ranges of several performance metrics. These ranges are important for the design of the migration protocol for resource-constrained links. We investigate relevant migration parameters: neighbor discovery time, transmission delay (round-trip), and throughput. The neighbor discovery time directly subtracts from the time budget available for orchestrating migration and transferring state. The round-trip time message delay impacts duration of exchanging messages during orchestration. The throughput is used to estimate how much state data can be transferred during the remaining time budget.

6.5.1 Experimental setup

The setup consisted of two HTC 3600 smart phones running Windows Mobile 5 and with an 'SDiD 1010' NFC dongle [3] attached. The NFC dongles were configured in active mode with 424 kbit/s PHY data rate. A file transfer application was deployed on both devices and one application instant acted as NFC initiator while the other acted as NFC target. Since the maximum frame size was 186 bytes, fragmentation was implemented. All experiments were repeated 30 times and the results are shown with 95% confidence intervals.

6.5.2 NFC performance measurements and analysis

Neighbor discovery time was measured by placing the devices in range and let the initiator search for targets. The initiator blocks execution while searching for targets. The measured interval indicates the blocking time from the moment the application started searching until the target was discovered and the initiator could continue. The mean discovery time was measured to be 59 ± 0.8 ms.

Transmission delay was measured by sending the smallest possible frame size (16 bytes header, no payload) back and forth between the NFC entities. The delay is defined as a round-trip delay, i.e. as the time from starting the send procedure on one device until receiving the response on the same device. For two nodes, the average delay was measured to be 106 ± 1.8 ms.

Throughput was measured by sending multiple frames continuously to mimic a large application state object. The total size of the transferred file was varied between 10-660 bytes and the time was measured from the first event until the final acknowledgment was received. Results from the experiment for different file sizes are shown in Figure 6.4. The figure shows that the throughput increases rapidly with increasing file sizes on the left end and then it converges to a value around 3.26 kbit/s for values above 300 bytes. The throughput degradation for small file sizes is expected as the ratio between overhead and payload is relatively high.

6.5.3 Orchestration protocol results

To evaluate the feasibility of the lean migration protocol, it was implemented on the device described in the setup. One of the mobile phones acted as the

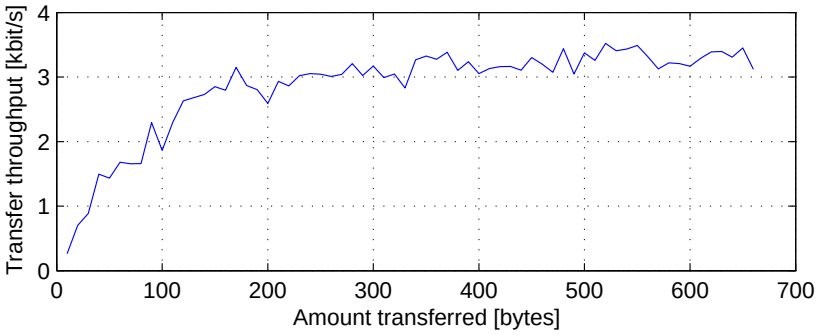

Figure 6.4: Throughput measurements over NFC link

large display. The goal of the study was to quantify how much time is available for actual state transfer during the lean migration protocol. An additional goal was to understand how much state information can be transferred within a typical swipe, by comparing the bandwidth results described above with the time results. The proposed migration orchestration messages GET_STATE_SIZE, GET_STATE and TERMINATE were implemented as simple string messages and the transferred state size was varied similar to the throughput measurements. The results of one experiment for each application state size in the range between 1 and 700 bytes are shown in Figure 6.5.

The results for the migration duration show a linear behavior over state size S; a least-squares fit yields the relation $delay = 2.1\frac{ms}{byte} \cdot S + 386ms$ which is also depicted in Figure 6.5. The minimum delay at $S = 0$ is due to the required 3 message exchanges in the protocol. This can be seen as signalling overhead. A detailed analysis of the time stamps shows that each exchange requires 106ms plus some processing delay. The inverse of the slope of the line $\frac{1}{2.1}\frac{bytes}{ms} = 3.8kbit/s$ is in the same order of magnitude as the throughput values observed in Section 6.5.2.
The observed linear behavior can be used to estimate the limit on state sizes that are feasible to transfer in a certain time window. For instance, assuming a maximum time window of 5 seconds for the transfer, the relation yields a

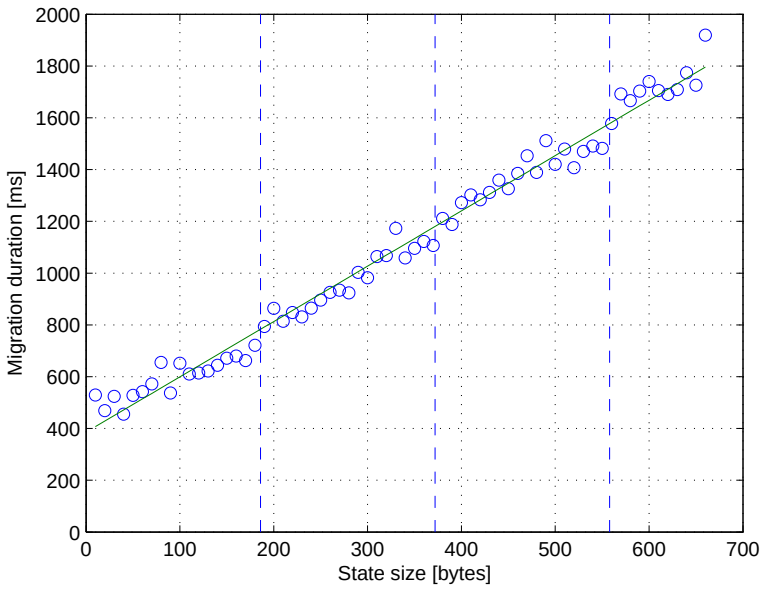

Figure 6.5: Duration of a migration consisting of orchestration signaling and state transfer

maximum S=2197. Based on this upper limit, the migration framework can decide whether to initiate migration when only the NFC link is available as orchestration method. We consider this orchestration method decision problem an extension to the configuration decision problem analyzed in Chapter 5. Chapter 7 presents analyses of added model complexity and migration performance when choosing both target configuration and orchestration method in each decision epoch in the trigger management component.

6.5.4 Using multiple, diverse wireless links for orchestration

In the scenario above, the assumption is that only one of the wireless links is available for orchestration at any time. In some cases, though, it may be that both technologies are available simultaneously. Then two options apply; either the platform uses both links for orchestration, or chooses one of them. In this section we test the hypothesis, that aggregating a WLAN link and an NFC link will provide more throughput than just WLAN alone. In Chapter 7 we also study the impact of choosing between dynamically available orchestration methods.

To evaluate the potential throughput gain of combining WLAN and NFC for orchestration, we construct a scenario where both are available simultaneously and transfer data using both connections simultaneously. We used the same experimental scenario as described above, and considered the following parameters:

- Without or with NFC communication; to evaluate only WLAN or combined WLAN+NFC
- Data origin; whether the files where downloaded from the mobile device (direct download) or from the migration server (indirect download)

The average throughput obtained over 10 repetitions of the experiments for the different combinations of the parameters are plotted in Figure 6.6. The figure shows that the throughput when downloading directly from the mobile device is slightly lower than when downloading from the migration server. It also shows that the throughput is nearly halved when the NFC link becomes active, and that the throughput of the combined links also suffer when downloading directly.

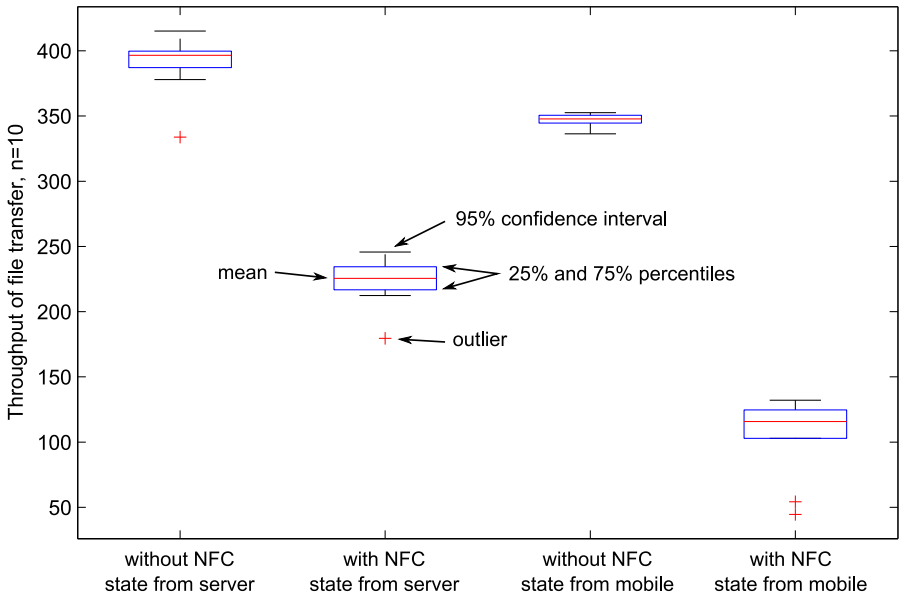

Figure 6.6: Average obtained throughput (kB/s) during data transfer over diverse wireless link, for different settings of the NFC link (on/off) and the state object origin (server/mobile device). 10 repetitions.

These results show that the expected throughput gain of aggregating the diverse wireless links cannot be obtained by using them independently. The handling of multiple wireless interfaces challenges the mobile device enough to reduce the throughput instead of maintaining or increasing it. Preliminary experiments have not shown radio interference between the used frequency domains. The consequence of this result is that combination of wireless technologies should be designed carefully, if a throughput gain is expected. The throughput gain maybe achieved by using the fast setup-time of the NFC link to negotiate optimal communication parameters for the WLAN link, such that a data transfer over WLAN may start sooner. It is, however, not recommended to use the two technologies in simultaneous, independent combination in the experimental platform.

6.6 Conclusion

This chapter presents a solution for orchestrating migration over resource-constrained links, with the specific example of RFID based near-field communication (NFC) technology. The main challenge of migrating an application over resource-constrained links is to make best use of the limited time in which the link is available. The primary steps of migration that can be optimized time-wise are device discovery, decision to trigger and message exchange during orchestration, including transferring the actual state of the application from the source device to the target device. In a general migration scenario, all these steps are coordinated by a central migration server in the network. Through an experimental setup we have shown that in some cases, migration over ad-hoc NFC links is possible and feasible. This is achieved by the primary migration decisions and control of orchestration procedure from the server to the target device. By measuring the overhead of the optimized migration orchestration protocol and the performance of NFC it is shown that the platform can be used for migrating application state sizes up to 2000 bytes within an NFC swipe window of 5 seconds.

References

[1] Z. Antoniou and S. Varadan. Intuitive mobile user interaction in smart spaces via nfc-enhanced devices. In *ICWMC '07*, 2007.

[2] I. Cappiello, S. Puglia, and A. Vitaletti. Design and initial evaluation of a ubiquitous touch-based remote grocery shopping process. In *NFC '09*, 2009.

[3] Wireless Dynamics. SDiD. http://www.sdid.com/products1010.shtml, 2010.

[4] ECMA. *340: Near Field Communication: Interface and Protocol (NFCIP-1)*. European Association for Standardizing Information and Communication Systems, http://www.ecma.ch/ecma1/STAND/ecma-340.htm, 2004.

[5] M. Elkotob and K. Andersson. Analysis and measurement of session setup delay and jitter in VoWLAN using composite metrics. In *MUM'08*, pages 190–197, 2008.

[6] M. Handley, V. Jacobson, and C. Perkins. SDP: Session description protocol (RFC-4566). *Request for Comments, IETF*, 2006.

[7] Rosa Iglesias, Jorge Parra, Cristina Cruces, and Nuria Gómez de Segura. Experiencing nfc-based touch for home healthcare. In *PETRA '09*, 2009.

[8] S. Karpischek, F. Michahelles, F. Resatsch, and E. Fleisch. Mobile sales assistant - an nfc-based product information system for retailers. In *NFC '09*, 2009.

[9] F. Kneissl, R. Rottger, U. Sandner, J.M. Leimeister, and H. Krcmar. All-i-touch as combination of nfc and lifestyle. In *NFC '09*, 2009.

[10] G. Madlmayr, J. Langer, C. Kantner, J. Scharinger, and I. Schaumuller-Bichl. Risk analysis of over-the-air transactions in an nfc ecosystem. In *NFC '09*, 2009.

[11] M. Massoth and T. Bingel. Performance of different mobile payment service concepts compared with a nfc-based solution. In *ICIW '09*, 2009.

[12] H. Mika, H. Mikko, and Y.-o. Arto. Practical implementations of passive and semi-passive nfc enabled sensors. In *NFC '09*, 2009.

[13] P. Schoo and M. Paolucci. Do you talk to each poster? security and privacy for interactions with web service by means of contact free tag readings. In *NFC '09*, 2009.

[14] M. Ulvan and R. Bestak. Analysis of Session Establishment Signaling Delay in IP Multimedia Subsystem. *Wireless and Mobile Networking*, pages 44–55, 2009.

7

Interplay between trigger management and orchestration

The previous chapters have addressed the challenges of automatic migration triggering between multiple devices. In both chapters one network was assumed available for orchestrating the migration. The purpose of this chapter is to present an extended analysis that combines the parameters of the previous scenarios and solutions. The chapter analyzes the migration performance when the number of available orchestration methods in the trigger management function is increased beyond one. Furthermore, the orchestration methods cannot be assumed to be permanently available, i.e. their availability is considered dynamic. Since it is the task of the migration orchestration component to know which methods are available, the purpose of the analysis is to understand how this knowledge from the orchestration component can be utilized in the trigger management component to improve overall migration performance.

7.1 Motivation

Chapter 5 presented the analysis of how knowledge about an orchestration method is used in the trigger management component when generating decision policies that improve performance of the overall migration system in a dynamic network. The assumptions of Chapter 5 are:

- one orchestration method always available for migration
- orchestration parameter values, i.e. success probability and migration delay, are known before generating decision policies

Chapter 6 presented a design and a performance analysis of an optimized orchestration method specifically targeted at resource-constrained networks, with NFC as the case-study. One of the properties of the scenario was that multiple orchestration methods were available at different times; at one point a WLAN-method and at another point an NFC-method.
The availability of multiple orchestration methods at different time poses a new dimension of challenges to the decision framework. Not only does it need to decide which configuration to use, it must also consider how to achieve this, i.e. what orchestration method to apply.

Figure 7.1: Focus is on the set of orchestration methods as input to the trigger management component and how their dynamic availability affect the user experience of the migratory system.

This chapter presents an analysis of interplay between trigger management and migration orchestration to understand how to include multiple orchestration methods in the decision framework and how to use knowledge about available orchestration methods while the system is running. This perspective on the trigger management component is illustrated in Figure 7.1. We show how the complexity of the offline policy generation approach increases in the decision framework as more orchestration methods and their availability are introduced. The results show that by only using the available subset of orchestration methods when generating a policy, as opposed to including all possible orchestration methods, the complexity of the models is reduced without affecting decision performance.

7.2 Scenario

Figure 7.2 illustrates a reference migration scenario for this chapter with multiple configurations and multiple orchestration methods. The application is running in configuration c and can be migrated to multiple configurations (devices), $C' = \{c'_1, c'_2, ..., c'_M\}$. Multiple orchestration methods, $\Omega = \{\omega_1, \omega_2, ..., \omega_K\}$, are available to carry out the migration.

Multiple networks types (NFC, WLAN, ...) can be available for migration. Furthermore, the application may be able to use different state-persistence techniques (checkpointing, shared memory/variables, complete application state extraction, virtualization). ω represents specific migration parameters for migration, e.g. network type, state size, etc. At each decision epoch, the decision framework determines the optimal configuration and orchestration method given the current network conditions. All configurations are assumed available, and since the availability of the orchestration methods is dynamic, the input to the decision framework is the entire set of target configurations, C', and the *subset* of available orchestration methods, $\Omega_{ss} \in \Omega$. The output of the decision framework is then the tuple (c', ω).

A specific example of the above reference scenario can be explained by extending the scenario from Chapter 6. The user is running the client-part of an application on a mobile device, while a server side is running on an Internet server. The client-side application is, before migration, using the WLAN to connect to the server-side. The mobile device and the application

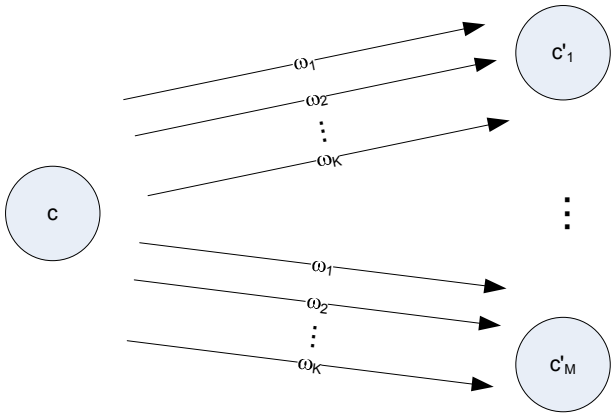

Figure 7.2: Example of scenario featuring migration from configuration c to c' with multiple possible orchestration methods. The individual methods may not always be available.

are registered on the migration server located within the migration domain. A large display device is also registered with the migration server and available to receive migrations. At one point, the user holds the mobile device close to the large display and trigger an NFC discovery event. At this point the set of available orchestration methods changes to include both the WLAN and the NFC links.

7.3 Challenges and contributions

From the reference scenario we identify the following problems, which are addressed in this chapter.

- How to include multiple orchestration methods in the MDP model to generate decision policies?
- How to handle dynamic orchestration method availability when generating policies?

The contributions of this chapter are as follows.

- Adaptation of the decision framework to generate decision policies with multiple orchestration methods.
- Identification of the increased complexity of the models used to generate decision policies, when using multiple orchestration methods and their dynamic availability.
- Reduction of complexity by adapting the policy generation function to work with subsets of choices.
- Performance comparison between two policy generation approaches, all choices vs. subsets.

Available orchestration methods are assumed able to migrate to any target configuration. With for instance NFC this is an approximation, since it can most likely only be used to migrate between the two devices connected by NFC. This assumption can be relaxed by considering combinations of orchestration methods and target configurations as available/unavailable, instead of just orchestration methods. This enables the exclusion of specific combinations of orchestration method and target configuration to mimic more realistic scenarios. For example, combinations of the NFC orchestration method and

configurations on other devices than connected to via NFC would be permanently unavailable. Such an extension is not considered in this work.

In Chapter 5, orchestration properties were assumed independent of the network state in the sense that the migration delay had the same distribution when the network was in normal state as when the network was in congested state. This assumption is rather strong. For instance, in the case where the configuration is changed because the network performance degrades and the migration is performed over the network. The degraded network performance will also influence the migration delay and success probability. In this chapter we redefine the success probability to be influenced by network state. We assume this is satisfactory to demonstrate the contribution of this chapter regarding dynamics of multiple orchestration methods.

7.4 Modeling approaches

The level of model complexity depends on how many dimensions are modeled. In the following, two types of models are described; a general model that includes all states in all dimensions and a more specific model that considers some of the state-space dimensions fixed.

7.4.1 World model

A *world model* models every relevant aspect of the migratory system. The world model must include

- every possible configuration that can be experienced when running the system
- every possible orchestration method that can be experienced when running the system
- configuration availability
- orchestration method availability
- network impact on reward in a certain configuration or when migrating using a certain orchestration method

Each of the above parameters adds a dimension to the world model state space, summing to 5 dimensions. In the general form, the number of states, $|S|$, in the

world model of all parameters above can be calculated by state-space products as

$$|S| = N \cdot 2M^2 \cdot 2K.$$

where N is the number of networks states, M is the number of configurations and K is the number of orchestration methods. Already with a simple scenario (as used later, cf. Section 7.6) with 2 configurations, 3 orchestration methods and 4 network states, the state space of the world model contains 192 states. This shows, that even for a simple scenario with few choices in each dimension, the state space gets large when considering availability of configurations and orchestration methods. The size of the transition matrix used in the MDP also quickly explodes. In real migration scenarios, the amount of choices is much larger than the above simple example, and in principle infinite, as all possible target configurations ought to be considered, meaning that any device capable of acting as target device should be included. Hence, a method is needed to reduce the number of configurations and orchestration methods to include when generating the decision policy.

7.4.2 Subset model

By making some assumptions of the availability of the configurations and the orchestration methods, it is possible to reduce the number of states in the model in small scenarios. A small scenario is a scenario, where the number of possible configurations and orchestration methods are known beforehand, bounded and available during the use of the system. Examples are for instance a home domain or an office domain.

The most specific partial model only includes a subset of available orchestration methods. This is the smallest model, in terms of state number, and also the most restricted, in the sense that it assumes only the subset of orchestration methods to be available during the use of the system. This means that it is not possible to choose unavailable orchestration methods, even though they may become available after a specific policy is activated.

7.4.3 Applied approximation approach

In this work, we use a combination of the two above model types to investigate the trade-off of using subset models. The motivation is that all possible choices cannot be known when the system is designed, and as such, the system needs to be able to adapt and make decisions based on the currently available choices,

which may then change in the future, either because of network dynamics or changes in the scenario (a new user enters, new applications are installed, new networks are set up, etc). Moreover, the time it takes to generate the policies increases exponentially with the number of configurations and orchestration methods, since all possibilities have to be accounted for in all future steps to generate the optimal decision policy. Therefore, a reduction of the number of states by reducing the number parameters will improve both offline and in particular online policy generation performance.

We compare two approaches to policy generation that do not model orchestration availability, i.e. they are not full world models. One approach, called the *world model approach*, uses a model that includes all orchestration methods in the scenario, and assumes them to be available at all times. The other approach, called the *subset model approach*, uses a set of subset models, which each consider a combination of available and unavailable orchestration methods (for details, see Figure 7.7).

The interplay element of the analysis is added in the policy enforcement function. In the world model approach, the one policy is applied always, independent of the set of available orchestration methods. In the subset model approach, the policy is changed to match the set of available orchestration methods, as reported by the orchestration component. In principle, the subset policies should be regenerated whenever the set of available orchestration methods changes, however, we pre-generate all policies (as it is possible in the small case study scenario) and just select the appropriate policy accordingly during run.

7.5 Integration of trigger management and migration orchestration

This section presents the design of the extended MDP decision framework where the choice contains a target configuration and an orchestration method. From the reference scenario, the relevant problem are identified and the solutions are described subsequently.

7.5.1 Extended decision framework

The extended decision functions are is shown in Figure 7.3 (policy generation) and Figure 7.4. Compared to the functions of Chapter 5, the extended versions include generation of subset policies and interactions between migration orchestration and trigger management.

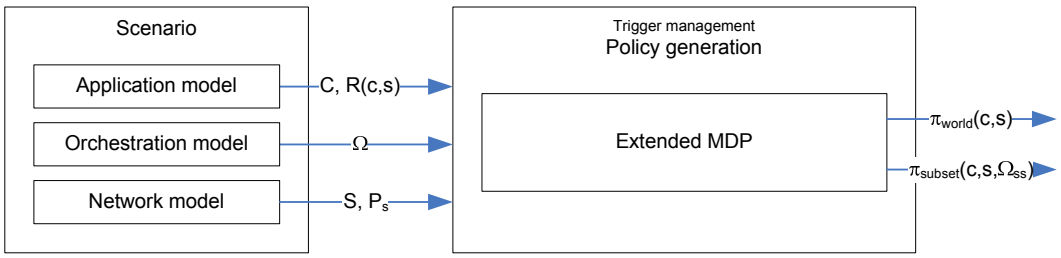

Figure 7.3: The extended policy generation function, where a set of orchestration methods, Ω, as delivered as input for offline generation of two types of policies: *world policy* π_{world} and *subset policy* π_{subset}.

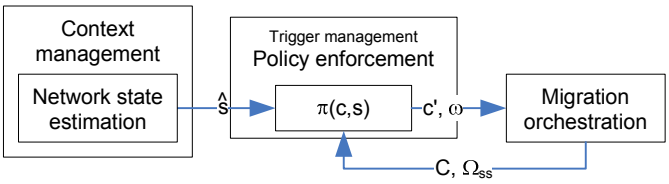

Figure 7.4: The extended policy enforcement function, where the orchestration components delivers the set of available orchestration methods when running the platform, such that the matching π_{subset} can be used.

The interaction between the migration orchestration component and the trigger management component contains an interface to exchange knowledge about available orchestration methods. These are initially collected in the orchestration component, since this component needs the knowledge on how to apply them. Based on a model of the orchestration process, the orchestration component is able to deduce the properties of each orchestration method that are relevant for the trigger management component and deliver these as a collection of K orchestration methods, $\Omega = \{\omega_1, \omega_2, ..., \omega_K\}$.

The migration orchestration component contains an availability estimation function that keeps Ω_{ss} updated, so when an orchestration method becomes unavailable, it is also removed from the set and the set is re-sent to the trigger management component. The estimation process is performed in the migration

orchestration component and delivered as an input to the decision policy enforcement function in trigger management. In practice, estimation can neither be accurate nor instantaneous (cf. Chapter 4), as the orchestration component needs to monitor the status of each device and network by exchanging messages (like heart-beat monitoring). However, it is assumed that the availability estimation is ideal (accurate and instantaneous).

7.5.2 Multiple orchestration methods in the MDP

The MDP state space is constructed as combinations of current configuration (c), target configuration (c'), orchestration method (ω) and network state (e). As an optimization, the states that have $c = c'$ can be interpreted as/joined into one state, in which no migration occurs. This is also the only type of state where migration can be triggered, since migration will be ongoing in all other states (from c to c' using ω). The general representation of the MDP is seen in Figure 7.5.

The possible actions to take in the non-migrating states are extended compared to Chapter 5 such that an action is defined by target configuration and orchestration method as $A = \{c', \omega\}$.

Subset policies are generated by solving the MDP with a limited transition matrix, where transitions probabilities have been adopted to not trigger a migration if an unavailable orchestration method is used. Such a transition matrix is generated for each combination of the orchestration methods.

7.6 Performance evaluation

A simulation setup in MATLAB is used to evaluate the performance difference in of the two modeling approaches to decision making. Whereas in Chapter 5, the migration behavior was analyzed for a full spectrum of different network parameters (p, q), here we choose one network scenario to analyze. In order to study the effect of orchestration properties on migration, we require a scenario with many migrations, such that the orchestration methods are used often, and their differences are possible to measure. Many of the scenarios from Chapter 5 are not relevant, because they only contain one or two initial migrations, before settling into one configuration for the rest of the run. Therefore, in the design of the scenario we must ensure that

- multiple migrations occur

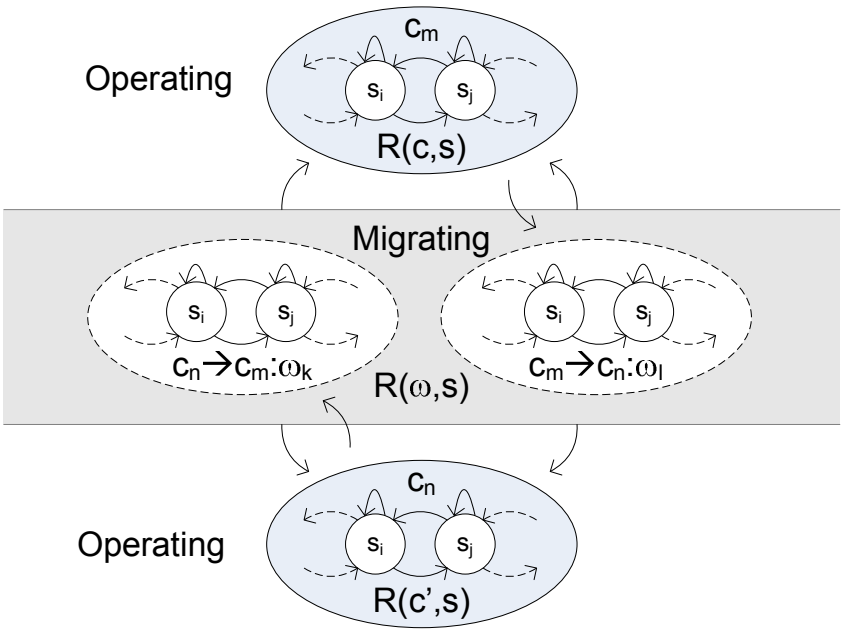

Figure 7.5: General model of the MDP extended with multiple orchestration methods. Compared to the original MDP, two decision have to be made upon migration; which target configuration c to select and which orchestration method ω to use for migrating to c. The reward, R, represents the quality of the user's experience of using a migratory application. R depends on whether the application is in a certain configuration or whether the application is being migrated and if, then which migration orchestration method is used.

- all configurations are visited
- all orchestration methods make sense to use at some point

Another difference from the scenarios in Chapter 5 is the changing availability of the orchestration methods. We simulate this as a two-state Markov chain for each orchestration method, the two state representing available or unavailable, respectively. Given a specific number of orchestration methods, their availability states are combined into one Markov chain, where each state is a unique combination of (un)available orchestration methods. We use this modeling technique to ensure that the behavior of the availabilities are independent of any other behavior in the system.

7.6.1 Methodology

We compare the performance of the two approaches by simulating different settings of the mean ω sojourn time μ_ω, which influences how often the set of available orchestration methods changes. A large μ_ω means slow changes, and a small μ_ω fast changes. Comparing the decision performance for different settings of μ_ω is interesting as orchestration method availability is not modeled by any of the decision approaches. The MDP generates a decision policy based on the assumption that the available orchestration methods remain available in the future, which is controlled by the size of the look-ahead window, w. With a small μ_ω, the availability changes fast, which causes an MDP with a large w to generate policies based on wrong assumptions, since it assumes the currently available Ω_{ss} to be available for too long.

The world model policy assumes all orchestration methods available at all times. The faster the availability changes, the better the world model policy makes decisions, as slow changes means a high probability of wrong decisions.

The active subset model policy is changed when the set of available orchestration methods changes, where it is assumes to be the optimal policy based on a finite window of future decisions. The MDP assumes that the model is stationary during all decision epochs of the future window. The window is large enough for the starting state not to influence the policy (cf. Section 7.6.3) - the policy has converged. However, the smaller μ_ω becomes, the less accurate is the stationarity assumption of the MDP, and the performance is expected to degrade.

The following method is applied to perform the evaluation:

- Specify system model with multiple orchestration methods
- Calculate transition matrix (world model)
- Create transition matrices for each subset of ω availability, Ω_{ss}, by disabling migration using unavailable orchestration methods (migration transitions are redirected to original state)
- Solve MDPs for all subsets to generate subset policies
- Simulate migration scenarios with dynamically available ω subset with the two different decision strategies
  - World model policy: Always use policy calculated from Ω
  - Subset model policies: Change policy according to Ω_{ss}.

- Calculate average reward and compare performance

7.6.2 System model

To meet the requirement specified above, the scenario is designed as follows.

- 4 network states, $S = \{s_1, s_2, s_3, s_4\}$
- 2 configurations, $C = \{c_1, c_2\}$
- 3 orchestration methods $\Omega = \{\omega_1, \omega_2, \omega_3\}$

The network model has 4 states compared to 2 states of the network models used previously. 4 states are necessary to make all orchestration methods optimal choices at some point. This kind of model could model the scenario of the BN evaluation, with route and wireless link as parameters. 4 states then corresponds to $S = \{\text{no faults}, \text{congestion}, \text{poor wireless link}, \text{two faults}\}$. The model is assumed to have uniformly distributed steady-state probabilities. We use 3 orchestration methods to retain a choice when the availability is dynamic. If only two are used, there is no choice to make when only one orchestration method is unavailable. The resulting MDP is seen in Figure 7.6. In total, this scenario results in 48 states.

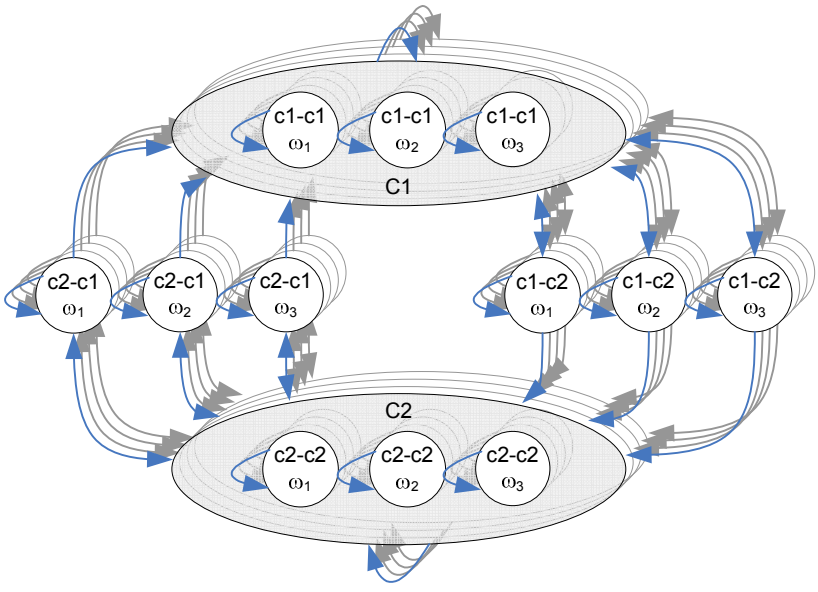

Figure 7.6: MDP of the evaluation scenario with 4 network states, 2 configurations, 3 orchestration methods.

The reward of a configurations depends on the network state, defined by

$$R(c,s) = \begin{bmatrix} 4 & 1 & 1 & 1 \\ 3 & 2 & 2 & 2 \end{bmatrix}$$

The above matrix ensures that the generated policies will contain migrations in both directions ($c_1 \rightarrow c_2$ and $c_2 \rightarrow c_1$). In state s_1 the policy is the to migration to c_1 and in states s_2, s_3, s_4 the policy is to migrate to c_2. To ensure that every orchestration method is used, the success probabilities are also dependent on the network state, such that

$$P_s(\omega, e) = \begin{bmatrix} 0.8 & 0.9 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.9 & 0.8 \\ 0.8 & 0.8 & 0.8 & 0.9 \end{bmatrix}$$

This success probability matrix makes ω_i most suitable in environment state s_{i+1}. Note that state s_1 is the only one in which migration $c_2 \rightarrow c_1$ occurs. For this migration only one orchestration method is used, and when the success probabilities are equal, ω_1 is always used, because of the lowest index.

The migration delay of all ω is modeled by μ_d and $p_d = 1 - (\frac{1}{1+\mu_d})$.All migration delay distributions are equal and with a non-zero mean. Having all mean delays equal means that the delay will not impact the choice of orchestration method, however, it will influence the tendency to trigger a migration (since the mean delay is not zero). The delay is used to include the penalty of migration, but to simplify control over the choice of the orchestration methods, and leave this control to the success probability parameters.

The penalty to the reward from the migration procedure is modeled by $\eta(\bar{D}) = -\alpha\bar{D}$. In this scenario, $\bar{D} = \mu_d$ and $\alpha = 0.01$ for all orchestration methods. It is possible to have different penalties for different orchestration methods, e.g. η_ω, but this is not considered in this work.

parameter	description	default value
μ_e	mean environment state sojourn time	300
μ_d	mean migration delay	30
μ_ω	mean ω combination duration	300
t	simulation steps	800000
n	number of experiments	128
W_{mdp}	MDP window size	3000

Table 7.1: Parameters and default values of the interplay simulation setup.

The availability of the 3 orchestration methods (ω) is modeled using the Markov chain depicted in Figure 7.7. The Markov chain model all possible combinations of the 3 orchestration methods and their availability, and it is assumed that only one method can change availability at a time. The subsets of available ωs are

$$\Omega_{\text{ss}} = \begin{bmatrix} [0 & 0 & 0] \\ [0 & 0 & 1] \\ [0 & 1 & 0] \\ [0 & 1 & 1] \\ [1 & 0 & 0] \\ [1 & 0 & 1] \\ [1 & 1 & 0] \\ [1 & 1 & 1] \end{bmatrix}$$

The behavior of the orchestration method availability is modeled such that only one change can happen at a time. The mean time between changes is the same for all orchestration methods, μ_ω, which results in equal transition probability, $1 - p_\omega$, where the relation between μ_ω and p_ω is modeled as a geometric distribution.

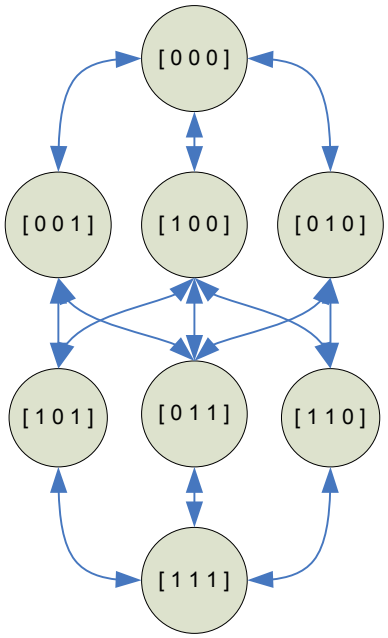

Figure 7.7: Markov chain of the availability of ω. Each state represents a unique combination of available ωs, when 3 methods exist, and assuming that only one method can change availability at time.

7.6.3 Results and analysis

Policy convergence

The optimal policies generated by an MDP are influenced by w, which determines how many future decisions to take into account when solving the MDP. Therefore, also the time it takes to solve the MDP is influenced and increases with an increased w. In order to understand where the convergence level is of the policies (i.e. which window size can be used consistently between all subset-policies during the performance evaluation), the scenario was simulated with increasing values of w. The results are shown in Figure 7.8.

The results show that all subset-policies are influenced by the size of w and that they have different convergence speeds. Based on the results, $w = 3000$ is chosen as the default value for all subset-MDPs in order to ensure that all subset-policies converge.

System behavior

The entire system Markov chain has 48 states for this scenario, so in order to depict the behavior sensibly, the states have been aggregated into a smaller state space.

In Figure 7.9, a slice of a simulation run is illustrated with the smaller state space distributed over three sub-figures. From this it is possible to observe how the system behaves during a run. The upper sub-figure shows the *migratory system state*. The states here distinguish between being in a configuration (c_1, c_2) or being in a migrating state. If the system is in a migrating state, this is characterized by the triple $\{c, c', \omega\}$. The middle figure show the realization of the network state and the lower figure shows the realization of Ω_{ss}.

Figure 7.9 allows us to interpret that the system is exercised in most of its states during this relatively short slice of the simulation (total simulation time run is 800000). Several migrations occur using several different orchestration methods. Only ω_1 is not used in this slice. It is, however, used at other times in the simulation run. It is also seen that the migration occurs after an environment state change, e.g. just after t=59100, where a migration from C2 to C1 is triggered by a change in environment state. The quick state change between t=59500 and t=59600 where a migration from C1 to C2 is triggered, but the system goes back into C1 is the display of a migration failure. Since the system state has not changed during migration, a new migration is triggered immediately and successfully completed after a new delay period. In this case,

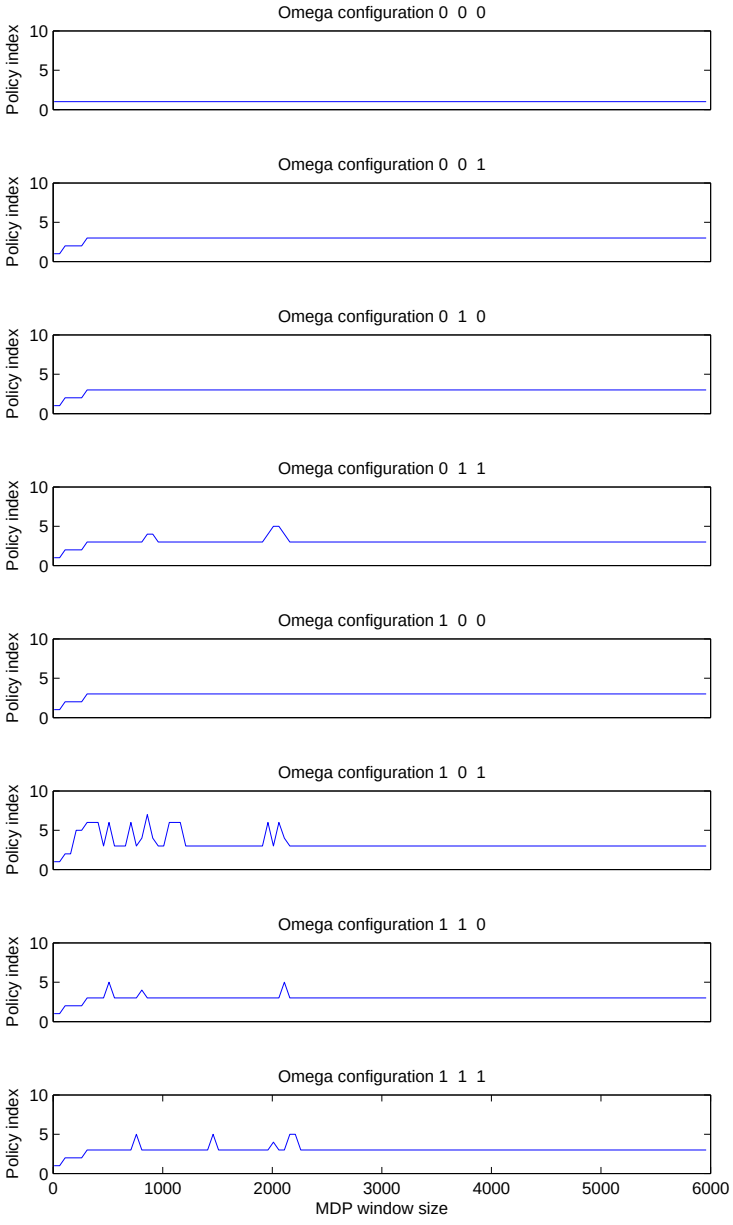

Figure 7.8: Analysis of the MDP w impact on policy generation to show when the policies converge, and thereby determine a sensible size of w.

ω_2 is used for migration, as it is the only one available, so even though it may not be the globally optimal choice given the environment state of the system, it is the locally optimal choice in this specific decision epoch.

This can be observed just before 59900, where another migration from C1 to C2 is seen. Since ω_3 is available it is chosen by the decision policy as a better choice than ω_2 that failed previously.

Performance comparison

The results of the performance measurements are depicted in Figure 7.10. The figure shows the mean average reward gained by the two approaches respectively for values of μ_ω from 10 to 1000 (in steps of 50). The figure shows an increasing performance difference between the two approaches with the increase in μ_ω. This is as expected, as the increased μ_ω reduces the dynamic of the orchestration method availability and thus makes the assumption of the

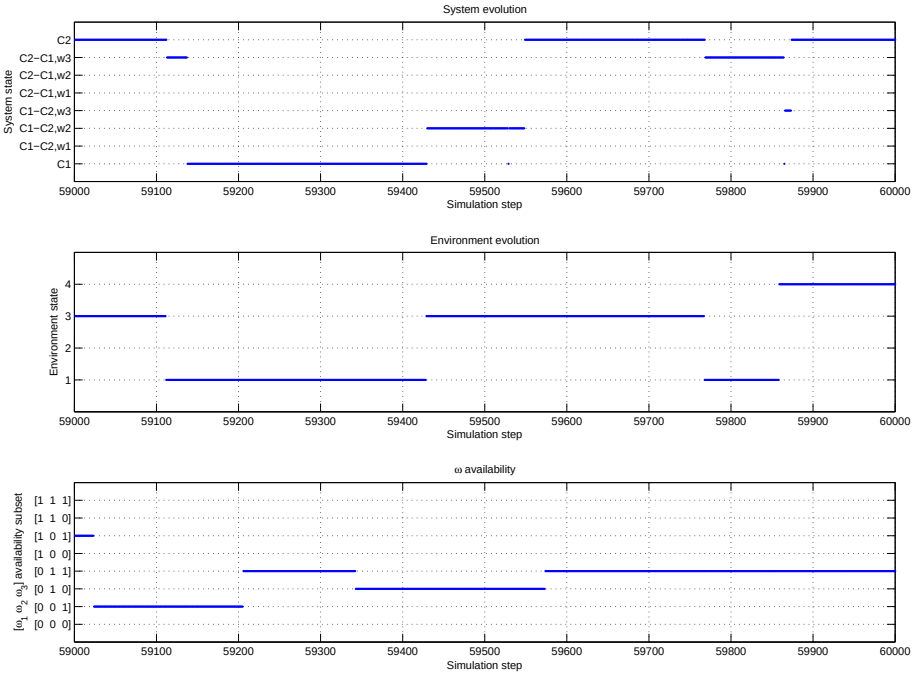

Figure 7.9: Results from one simulation of the system with the world model policy.

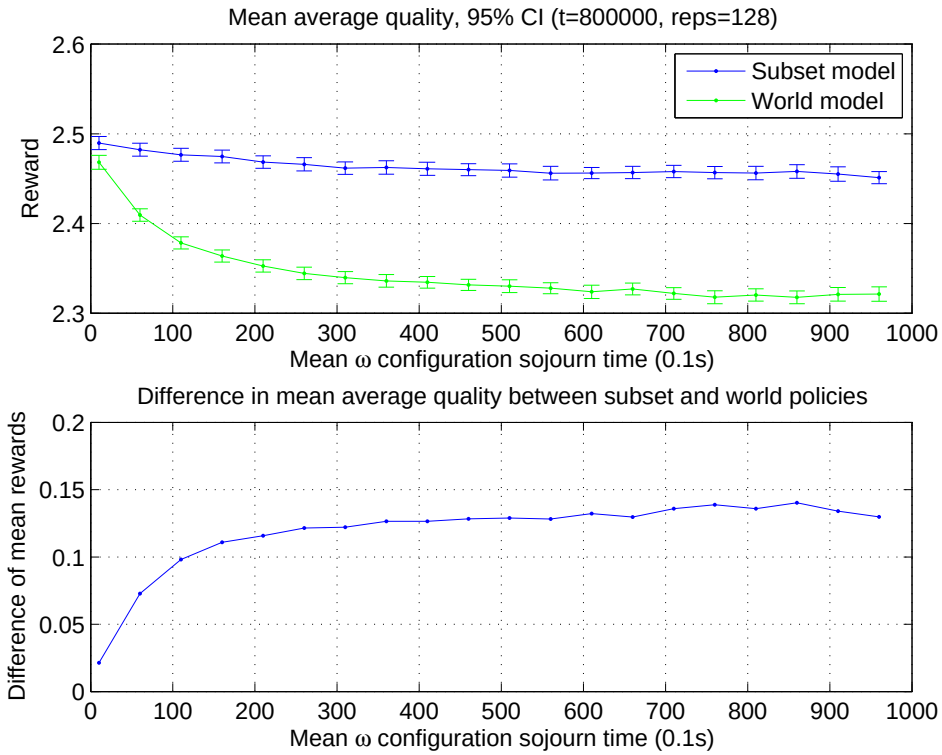

Figure 7.10: Results from varied μ_ω with the two decision policy approaches: Subset model has specific policies for combinations of available omegas, world model has only one policy, assuming all omegas are available all the time. Comparison metric is the average reward generated during a run of a migrating application, migrating between two configurations with up to 3 different omegas to choose from.

subset-MDP less approximate. Similarly, the reduced dynamic makes the error of the world model worse. This is the consequence of the situations where the optimal orchestration method becomes unavailable. Then the world model policy will constantly specify to use the unavailable orchestration method, and no migration occurs. With an increased μ_ω, the duration of the period increases where the world model policy makes this error and thus the mean average reward degrades.

7.7 Conclusion

In many migration scenarios, several network connections will be available to orchestrate the migration. As the underlying network connections may be unstable, the availability of the orchestration methods may change dynamically in a migration scenario. The MDP policy generation approach can be extended to handle multiple dynamically available orchestration methods, however, a world model that includes every dynamic dimension becomes intractable for any real-life scenarios. By generating decision policies that only include a subset of orchestration methods, and enforcing the policy that matches the set of available orchestration methods, the complexity of the decision policies can be made feasible. From an example scenario, we have shown that when using the subset policies the performance of the decision framework is better than when using a world model policy.

8

Conclusions and outlook

In this chapter, a short summary of the key points of each chapter is given, and a conclusion is provided with respect to the problem statement. Furthermore, the chapter provides an outlook of what will be the next step of the work presented and other perspectives with respect to migratory applications and the migration framework.

8.1 Summary

Service migration enables interactive applications and services to move between devices, while they preserve their state, so user's work flow is uninterrupted. By reacting to changes in the context of the application and adapting to them, the quality of the user experience of the application can be enriched by migration. In many situations, migration is triggered manually by the user. However, in scenarios with dynamic context the time available to trigger a migration may be too short for the user to react to, or the amount of information needed to make an informed choice may be too much for the user. Therefore, the migration needs to be triggered automatically in these scenarios. In this work, we focus on automatically triggered service migration within dynamic and resource constrained network scenarios. Network scenarios are important as network state dynamics influence both the application performance and the migration process and it is often difficult to control network properties in favor of migratory applications or the migration process.

Chapter 3 describes a centralized migration framework to enable automatic triggering and migration orchestration in scenarios with dynamic context. The framework includes core migration functions to

- estimate the network state as example of context information
- generate decision policies for choosing target device of migration and procedure for performing the migration
- enforce the decisions and orchestrate the migration

The specific method to orchestrate migration is also presented.

Chapter 4 describes solutions to the problem of estimating the state of the network under uncertainty. Through simulation evaluations, we show how model-based estimation approaches such as hidden Markov Models and Bayesian Network enable accurate, fast and robust network state estimation in migration scenarios. The platform enables delivery of context information, in this case network state information, and migration elements such as configurations and descriptions of orchestration methods, to where decisions are enforced, which is necessary for successful migration.

In Chapter 5 the estimated network state is used to decide which configuration the migratory application should be in. This decision is a trade-off between choosing application configurations and orchestration methods that a) deliver a good user experience and b) ensure fast and successful migration.

By applying model-based decision approaches such as the Markov Decision Process (MDP) to generate decision police, we show by simulation how the overall user experience is improved. Moreover, it is shown that inaccurate state estimation influences the resulting decision policies.

Chapter 6 describes a lean orchestration protocol for resource-constrained networks. It is investigated how well a resource-constrained network type such as Near-Field Communication (NFC) performs for orchestrating the migration. The performance of the protocol is evaluated experimentally and the results show that migration over an NFC link is possible for small application state sizes.

In Chapter 7 it is investigated how the migration framework performs when elements of the framework are affected by network dynamics. This is exemplified by introducing dynamic availability of the orchestration methods. The extension increases the complexity of the models used to generate policies, and two approaches are investigated to reduce the complexity. Both approaches ignore dynamics in policy generation, but one handles the dynamics in policy enforcement by using policies based on subsets of available orchestration methods. By simulation it is shown that changing the policy to match the available set of orchestration methods improves the user experience quality, compared to a model which assumes that all orchestration methods are available.

8.2 Conclusions

Migration triggering is a complex process involving network state estimation, choice of target configurations and orchestration methods and potentially application adaptations. Automatic triggering is beneficial in migration scenarios that include dynamic and resource-constrained networks, as performance of both application and migration depends on network state. In addition, networking scenarios are important as they demonstrate well the challenges of other dynamic behaviors that may characterize the context of the migratory applications.

There are three main user types that interact with the migration platform: application users, application developers and platform developers. The problems studied in this work are mainly faced by platform developers. However, the choice of solutions directly affect application users, as the results show, so the choices must balance the trade-off of implementation complexity and user experience.

Platform developers need much system knowledge to construct the models for state estimation and policy generation. With simple scenarios (small network, few configurations and orchestration methods) the effort invested in gathering domain knowledge to build the models may not be justified in enriched user experience. Simple heuristics are therefore enough to achieve sufficient performance.
In static scenarios, where new migration elements (configurations, orchestration methods) are not added, the world-model policy generation approach (including availability) may be the best choice for accurate decisions. As shown, the state space will quickly grow, so automatic methods to generate the models will be needed. When the state-spaces become very large, even if generated with automatic methods, the solution time and the storage size of the policies will be an obstacle. On the other hand, if migration elements can be added during the system lifetime, the subset-model approach must be used, in order to (over time) generate policies that include the added artifacts.

Furthermore, it is shown how estimation accuracy heavily influences migration performance. It is recommended to use the model-based approaches in scenarios where multiple hidden network parameters, such as route congestion and wireless link quality, may influence the application performance. In particular, when there is access to multiple observable parameters that are related to the hidden parameters, it is recommended to use the model-based methods, as they are shown to improve estimation performance, to improve migration performance. The results show that using multiple observations makes the models robust to parameter inaccuracies, so it is recommended to use the Bayesian Network in particular when multiple observations are available and if it is difficult to get accurate parameters.

The benefit of the model-based policy generation approach depend very much on the predictability of the network state and the relation between the network state dynamics and the migration procedures. Taking future decisions into account improves the migration performance only if such decisions are to be made. If the network dynamics are too fast for a migration procedure to complete as a reaction to a state change, then a migration will never be triggered, and the modeling effort is wasted. Therefore, it is recommended to use the MDP approach in migration scenarios where network state (in general environment dynamics) changes with longer interval than the migration procedure takes to complete, at minimum.

Performance of the orchestration procedures is shown to impact the migration experience. Optimization of the duration and success probability of the procedure is important in order to deliver a good migration experience.

We have shown how the procedures can be optimized in scenarios with only resource-constrained link available for migration, extending the domain where successful migrations can be delivered. The trade-off is the continuity of the application, as only small amounts of state information may be successfully transferred during migration, which may require the user to repeat steps in the application work flow. In order to keep applications continuous, the recommendation is to use the lean orchestration protocol in scenarios where the mobile device would otherwise be disconnected.

Overall, the proposed migration platform supports migratory applications in the described scenarios. These scenarios are general enough to show a broad scope of application types that can be migrated. In the end, the framework has some limitations. Access to the migration server is required, which means that the framework can only be deployed in settings with some sort or existing network infrastructure. Moreover, the decision policies are generated offline, which means that all applications, devices and networks need to be known beforehand. This is a challenge, in particular for large deployments, as for instance corporate domains. Moreover, the offline generated policies are based on reward models of the user experience with an application under certain network conditions. Methods to obtain such models have been studied in the literature, however, the perceived experience is very subjective, which is very tied to both applications, network properties and the particular users and their preferences. The fact that such models have to be known beforehand by application developers may limit the level of model details, as they may use very specific models to be able to deploy any applications at all. This will reduce the benefit of migrating applications, when not all context changes that affect application performance can be reacted to. Nevertheless, the framework supports the whole spectrum of details, and when more detailed reward models are derived, such that new context events can be considered in the decision policies, the migration framework can be upgraded accordingly.

8.3 Outlook

Migration framework Orchestration methods are chosen centrally at the migration server. However, when orchestrating over a resource-constrained link the choice of orchestration method has to be made locally on the device. In the studied case of NFC, time for orchestration is so short that the choice never will be made centrally. However, there clearly exist border-line cases, where the difference between making the choice locally and centrally impacts migration performance.

Also scenarios with multiple applications and users are interesting because they will share the device resources and thereby introduce a new decision trade-off where the best user experience may cover the average experience of many users and their applications instead of just one of each as studied here.

State estimation This work is based on network state as the context influencing the application performance. Other context parameters such as other resources, mobility and other users are interesting extensions of the scope.

Moreover, the delivery process in the context management framework may itself cause inaccurate state estimates, called *mismatch probability*. The effect and optimization of this factor should be studied and implemented in the estimation approaches.

Decision making In this work, migration elements such as configuration and orchestration methods are assumed known beforehand. This may not always be the case. Online learning of reward models, orchestration methods properties should be studies as a part of adaptive policy generation. Also, it is assumed in the policy generation that the network states are observable, and then the estimation task was separated into the estimation function. Partially observable MDPs can solve the same problem jointly in one function. The trade-off between added accuracy vs. increased model complexity should be studied as policy generation optimization.

Interplay Interplay between migration orchestration and trigger management components were studied to optimize the enforcement part of trigger management. In the same manner interplay with other controllable parts of the platform, e.g. QoS policies in the network or performance monitoring schemes, should be studies for optimization of the overall platform performance.

List of Symbols

.

a : Actions

s : Network state

$\hat{s}$: Estimated network state

o : Observation

c : Current configuration

c : Next configuration

ω : Orchestration method

N : Number of network states

S : Set of network states

O : Set of observation

M : Number of configurations

C : Set of configurations

K : Number of orchestration methods

Ω : Set of orchestration methods

A : Set of actions

α : Migration penalty

p_s : Migration success probability

p_d : Parameter of geometric migration delay distribution

$R(c, s)$: Reward function

$\bar{D}$: Mean migration delay

p, q : Network model transition probabilities

$\pi(c, s)$: Decision policy

w : MDP look-ahead window size