ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory
Communication Systems Group

Master's Thesis - Winter Term 2005/2006

Mobility Prediction in MANETs

Theus Hossmann
hossmath@ee.ethz.ch
MA-2006-08

April 30, 2006

Tutor:	Károly Farkas	farkas@tik.ee.ethz.ch
Supervisor:	Prof. B. Plattner	plattner@tik.ee.ethz.ch

Abstract

As mobile devices, such as laptops, PDAs or mobile phones, are getting more and more ubiquitous and are able to communicate with one another using technologies like wireless LAN (WLAN), the paradigm of *wireless mobile ad hoc networks (MANETs)* is gaining popularity. MANETs impose new challenges to the design of applications and network protocols because of their self-organizing, mobile and error-prone nature. Mobility prediction is a tool to deal with the problems emerging from the nodes' mobility by predicting future changes in the network topology. This is crucial for different tasks such as routing and distributed server selection.

This thesis presents an approach to mobility prediction based on *pattern matching*. Each node monitors the *Signal to Noise Ratio (SNR)* of its links to obtain a time series of past measurements. When a prediction is requested, the node tries to detect situations similar to the current one in the history of its links by computing the normalized cross-correlation function of the recent past with the collected training data. The found matches are then used as a base of the prediction. As an application of the SNR prediction, an extension to the formerly developed *Priority Based Selection (PBS)* algorithm was defined. PBS is used for distributed server selection in mobile ad hoc networks by computing and maintaining a Dominating Set of the network graph. The extension introduces a *link stability criterion*, which requires that a Client accepts only a node as Server to which it has a link predicted to be stable for a certain time.

In order to evaluate the developed prediction algorithm, it has been implemented in the network simulator ns-2. Simulations have shown that the predictions are highly accurate. Furthermore, the application to server selection has been proved successfully, as the stability of the computed Dominating Sets increased significantly using the link stability criterion.

Preface

With this Master's Thesis I will finish my studies at the Department of Information Technology and Electrical Engineering (D-ITET) [1] at the Swiss Federal Institute of Technology (ETH) in Zürich [2]. This thesis was performed at the Computer Engineering and Networks Laboratory [3] from October 2005 until April 2006.

I had the luck to finish my studies with an interesting Master's Project in a fascinating area. It was an absorbing but great time and I have learnt a lot on how science works. I want to express my sincere gratitude to everybody who made this thesis possible, especially to

- *Károly Farkas*, for the great guidance and support during the whole project;
- *Prof. Dr. Bernhard Plattner*, for making this project possible;
- *Eduardo Silva*, for the enjoyable teamwork and letting me win countless GPL Arcade Volleyball games [4];
- *My parents*, for the generous support during my whole studies;
- *Marinha*, for her inspiring ideas.

Zürich, 03.05.2006

Theus Hossmann

Contents

List of Figures

List of Tables

1

Introduction

This chapter provides an introduction to this thesis. In Section 1.1, the official task description of the thesis project is presented, including a working plan which was followed during the whole project. Section 1.2 introduces and motivates the work in more detail and explains the structure of this report.

1.1 Task Description

Mobile ad-hoc networks (MANET) are self configuring wireless networks which are not dependent on any centralized infrastructure, may comprise heterogeneous devices (mobile phones, PDAs, laptops, etc.) and span over several hops. With the increasing number of mobile devices, providing the computing power and connectivity to run applications like multiplayer games or collaborative work tools, MANETs are getting more and more important as they meet the requirements of today's users to connect and interact spontaneously.

Because of the lack of centralized infrastructure, service provisioning (specification, lookup, deployment and management of services) is a major challenge in MANETs. Distributed server selection is an important part of service provisioning. What makes server selection in MANETs difficult is the freedom of the nodes to join and leave the network whenever they want to. In order to prevent the network from choosing a node as a server, which is about to leave, predictions about the future behavior of each of the participating nodes is crucial for electing a stable set of servers.

SIRAMON (Service provIsioning fRAMework for self-Organized Networks) [6] is a framework, currently being developed at the Computer Engineering and Networks Laboratory [3] of ETH Zurich, with the goal of coping with the challenges of service provisioning in MANETs.
In a preceding master thesis [5] a distributed, zone based server selection algorithm, called *Priority Based Selection (PBS)* was implemented for SIRAMON. PBS elects servers based on node weights, representing how well a node is suited to act as a server. These weights are assigned to the nodes depending on parameters like link state and battery lifetime. However, PBS lacks the ability to predict the future behavior of nodes, which makes the selected server set unstable in some situations, especially when the mobiliy of the nodes is high.

1.1.1 Scope of the Thesis Project

The aim of this thesis is to explore the benefits of using mobility prediction for distributed server selection by adding prediction to the PBS algorithm. This should help making the elected set of servers more stable and avoiding costly re-elections. By means of simulation, the stability of the server set will be compared with and without mobility prediction. In order to meet these goals, the thesis is split into the following tasks:

1. *Literature exploration*: In order to gain an overview of the current state of research in mobility prediction in MANETs, a comprehensive literature exploration has to be performed.

2. *Requirements and constraints in SIRAMON*: This task will answer the following questions: What are the requirements of SIRAMON for a mobility prediction algorithm? Where and how can such an algorithm be integrated into the framework? What parameters provides the system for mobility prediction?

3. *Evaluation criteria*: Evaluation criteria and test scenarios with and without mobility prediction will be defined along with the desired performance of the algorithms with these criteria and scenarios.

4. *Algorithm selection*: Based on the previous tasks a mobiliy prediction algorithm will be defined.

5. *Analytical evaluation*: The chosen algorithm will be evaluated analytically regarding the defined criteria.

6. *Simulation*: The behavior and performance of the algorithm will be simulated in a network simulator. Therefore the algorithm will be implemented in ns-2 [7]. The simulated scenarios will be evaluated.

7. *Implementation in SIRAMON*: The algorithm will be integrated in SIRAMON with all the necessary adaptations of the system. Tests of the implementation with the defined test scenarios and, if possible, in real scenarios will be evaluated to gain a good overview of the performance of the algorithm with and without mobility prediction.

8. *Thesis writing*: A detailed report of the performed work will be written.

1.1.2 Working Plan

Table 1.1 shows the working plan of this thesis. The task numbers refer to the points defined above.

1.1.3 General Regulations

The project will be guided by Károly Farkas. At the end of the project, a written thesis report describing the work and outcome as well as the documentation of the implemented code have to be delivered. The master student understands and accepts the terms and regulations of ETH in regard to the developed code which will be published as open source under the terms of the GNU General Public License (GPL) [8]. In the course of the work two intermediate and a final presentation have to be given. An accepted thesis report and successfully accomplished presentations are prerequisites of getting the final grade of the master thesis work.

Start:	Monday, 24th October 2005
End:	Friday, 21st April 2006

Zurich, 29th September 2005
Theus Hossmann

Week	**Date**	**Tasks**							
		1	2	3	4	5	6	7	8
1	24th Oct. - 30th Oct.	X							
2	31st Oct. - 6th Nov.	X							
3	7th Nov. - 13th Nov.		X						
4	14th Nov. - 20th Nov.		X	X					
5	21st Nov. - 27th Nov.			X	X				
6	28th Nov. - 4th Dec.				X				
7	5th Dec. - 11th Dec.				X				
8	12th Dec. - 18th Dec.				X				
9	19th Dec. - 25th Dec.				X	X			
10	26th Dec. - 1st Jan.					X			
11	2nd Jan. - 8th Jan.						X		
12	9th Jan. - 15th Jan.						X		
13	16th Jan. - 22nd Jan.						X		
14	23th Jan. - 29th Jan.						X		
15	30th Jan. - 5th Feb.						X		
16	6th Feb. - 12th Feb.						X		
17	13th Feb. - 19th Feb.							X	
18	20th Feb. - 26th Feb.							X	
19	27th Feb. - 5th Mar.							X	
20	6th Mar. - 12th Mar.							X	
21	13th Mar. - 19th Mar.							X	
22	20th Mar. - 26th Mar.								X
23	27th Mar. - 2nd Apr.								X
24	3rd Apr. - 9th Apr.								X
25	10th Apr. - 16th Apr.								X
26	17th Apr. - 23th Apr.								X

Table 1.1: Working plan of the thesis project

1.2 Context of the Thesis

As mobile devices such as laptops, PDAs and mobile phones are getting more and more ubiquitous, and their capabilities to communicate with one another increase, a new desire to be connected emerges for the user. With the ability to connect directly via technologies like the widespread wireless LAN 802.11 standard [9] or Bluetooth the user is able to fulfill this desire, without having to rely on the infrastructure of network service providers and paying for the connections. Today, devices can communicate spontaneously in so-called *wireless mobile ad hoc networks (MANETs)*. While MANETs historically were mainly propagated for situations like disaster recovery or military applications as providing connectivity between the troops on a battlefield, the paradigm today is accepted as useful in a broader range of every day applications. One promising field where MANETs are applicable is *mobile gaming*. Imagine a group of people having some free time, for instance spending their break at a schoolyard, or waiting for something to happen, such as for their train to arrive at the destination. As they are in a close range one to another, they can spontaneously form a network, possibly spreading over several hops, and initiate a game session. In this case, a paid connection to a service provider is not necessary to achieve connectivity.

However, the self-organizing mobile and error prone environment of MANETs poses a number of challenges for the design of new communication protocols and applications. One important aspect to deal with is the mobility of the nodes. The most prominent example, where mobility of the nodes is a problem, is *routing*. Frequent changes in the topology of a network require a big communication overhead to establish new routes and slow down communication or causes packet loss. Another field where frequent changes of topology cause problems is related to *service management* in mobile wireless ad hoc networks. In such a network there are no designated servers, thus nodes acting as servers have to be determined distributedly. High mobility of the nodes can result in the selection of unstable servers, leading to frequent changes of server nodes and frequent handovers of clients from one server to another. This also causes high traffic overhead or even service disruption (during the time a new server is elected).

Mobility prediction is a tool to deal with the problems emerging from the mobile nature of MANETs. It can help increase the stability of such networks by predicting future changes of the network topology based on observations from the past. This thesis describes such a mobility prediction algorithm and shows how it can support the selection of stable servers for services such as gaming or any other application in a MANET. In the following sections of this introductory chapter, a more detailed overview of what mobility prediction is and why currently used methods are not satisfactory is given. Furthermore, the concept of *service provisioning*, which is a key problem in the self-organizing MANETs, is introduced.

1.2.1 Mobility Prediction in MANETs

Knowing the future topology of a network helps the service management algorithms determine as stable as possible servers and routing algorithms select the most stable routes between two nodes. Different mobility prediction methods for mobile ad hoc networks have been published so far (e.g. [10], [11], [12]), however, all of these examples use *specialized hardware*, e.g. Global Positioning System (GPS) [13] receivers, in order to determine the current state of mobility of the nodes. Though it can be assumed, that GPS (or the european pendant, Galileo) will get more and more widespread in the future, this dependence is not desirable, as the assumption of having such devices in each node is a limiting factor. GPS devices are still costly today and only a few mobile devices are equipped with them. Furthermore, they do not work well in indoor environments. Thus, assuming that the nodes are aware of their geographical location is a limiting constraint for a prediction algorithm which is supposed to work on a wide range of devices and in different physical environments.

The question addressed in this thesis is, how well the future network topology of a MANET can be predicted without using specialized positioning hardware. This restriction allows to operate with a broader range of networks, consisting of heterogeneous devices like laptops, PDAs or mobile phones, without having any constraints on the devices' capabilities. In order to do so, a large part of the work has to be dedicated to the observation of the current mobility state. This is the main point which distinguishes this work from others, as many other mobility prediction approaches assume that the mobility state of a node is given.

Abstracting from mobility prediction in MANETs and taking a glance at the general problem of prediction, it is essential to note that there are two major ingredients for any form of prediction. The first is the existence of observable parameters which are related to the predicted quantity. If, for example, one wants to predict the weather, the observable parameters could be the temperature, the air pressure and the wind velocity and direction[1]. The process of measuring these quantities is the *state observation* part. The second major ingredient is that some knowledge about the possible behavior of the system is required. For the weather forecast this knowledge may come from physical laws, which give a model of the system, but also from observations made in the past, used as training data. The part of creating a model which is able to predict future behavior from this knowledge is the *state prediction*.

Mapping these general requirements of prediction to the mobility prediction problem, it can be noted that for the state observation there are several options. As mentioned before, a popular choice is to use a GPS device in order to observe the geographic position, speed and moving direction. However, for predicting the future topology of a wireless network, this might not be the best choice. If a link failure should be predicted, it is more meaningful to predict the future link quality between two nodes instead. Link quality is a parameter directly related to whether a direct connection between the two nodes can be established or not. For predicting the failure of a link from information obtained by a GPS device instead, more information about the environment of the network is required in order to be able to choose a radio propagation model and map the distance of two nodes to a link quality. While this might be a viable way for some special cases, for instance if a free space model can be assumed, generally in MANETs there is no a priori knowledge of the physical environment of the network. It might be located in an office building, in a train, on a schoolyard or in any other imaginable environment. However, this lack of information can be compensated by learning from the past behavior of the nodes, by measuring *time series* of link qualities and using them as training data for the prediction. Of course, it is crucial that this training data show some structure, as it is clear that

[1]Of course, predicting the weather is a far more complicated science and the literature about it might fill whole libraries. But the readers may excuse this simplistic and unqualified view of weather prediction as it is only used as an example to show some general principles of prediction.

the more random the links behave, the less one is able to reliably estimate the future. While the physical environments mentioned above may have different degrees of randomness (for instance on the schoolyard the nodes can move in any direction they want, while in a typical office environment, the nodes may only move along the corridors and stop in the office rooms), they usually all show a certain pattern and therefore are predictable.

The goal of this work is to provide an algorithm which is able to cope with the rather hostile environment of a MANET for mobility prediction. That means, each node should, without knowing its environment, be able to detect patterns in the behavior of its links and therefore be able to predict the future topology of its neighborhood.

1.2.2 Service Management in MANETs

One key problem which arises with the mobile ad hoc networking paradigm is *service provisioning*. If, for instance, the users of a MANET would like to play a multiplayer game, this can, because of the self-organizing nature of the network, be a serious challenge. As there is no central infrastructure, even basic functions, such as determining a game server which maintains the game state, has to be done in a distributed manner. However, the challenge does not stop with having a server. Because of the freedom of the nodes to move wherever they want, the elected server may decide to move away from its clients, or even leave the network completely. To guarantee the continuation of the service for the other nodes, a *redundant* server should take over in such a case. Thus, a set of redundant servers should be formed and maintained as the network topology changes. To allow services in a network to run smoothly is the aim of *service provisioning*. Service provisioning is a very broad term, which covers all functionalities used for supporting a *service*[2] during its whole life cycle. An overview of the aspects of service provisioning is given in Table 1.2.

To provide a framework for service provisioning in MANETs a project called SIRAMON (Service provIsioning fRAMework for self-Organized Net-

[2]The term service alone is very broad. It covers all kinds of applications which provide a benefit for the user. Services can be roughly classified in *information providers* (e.g. a news service), *software providers* (e.g. downloading an offline game), *resource providers* (e.g. storage space or computational resources), *action providers* (e.g. printing) and *interaction providers* (e.g. an online multiplayer game).

Function	Description
Service Specification	Describing a service
Service Indication	Advertising a service to other users
Service Deployment	Requesting, downloading, installing and configuring a service
Service Management	Maintaining the service while it is running

Table 1.2: Service provisioning aspects

works) [6] was established at the Computer Engineering and Networks Laboratory [3] of the Swiss Federal Institute of Technology (ETH) in Zürich [2]. A short overview of the SIRAMON framework is given in Appendix B. For *service management* in SIRAMON, a distributed algorithm for server selection named *Priority Based Selection (PBS)* [5] has been developed. PBS distributedly computes a set of redundant servers. In order to do this, a *Dominating Set (DS)*[3] of the network graph is computed in a way that the nodes which are best suited to act as servers (in terms of available resources, position in the network, etc.) are more likely to be part of this Dominating Set. The nodes in the DS are then used as servers for a certain service. The question of how mobility prediction can be used to improve the stability of the computed DS is addressed later in this thesis.

1.2.3 Organisation of the Thesis

This thesis is organized as follows: In Chapter 2 some background information about time series prediction in general, as well as mobility prediction in MANETs and its application to server selection are provided. Related work is also presented in this chapter. With this necessary background, the chosen method of mobility prediction is described in Chapter 3. Chapter 4 describes the implementation of the mobility prediction algorithm in the network simulator ns-2 [7]. The results of the evaluation performed with this implementation are given in Chapter 5. Chapter 6 finally concludes this thesis and gives an outlook on possible future work.

[3] A Dominating Set is a concept in *graph theory*. It is a subset of the nodes of a graph, such that all the nodes are either part of the Dominating Set or are directly connected to a member of the Dominating Set.

2

Fundamentals

This chapter provides the essential background information for this thesis. Section 2.1 offers some general information about common prediction methods found in the literature. Section 2.2 then lays the focus on mobility prediction in a wireless mobile ad hoc environment, points out the assumptions and requirements such an environment has and shows a general structure of the mobility prediction problem. Server selection is presented as a possible application of mobility prediction in Section 2.3 and finally, in Section 2.4 other work related to this thesis is presented.

2.1 Time Series Prediction

As pointed out in Section 1, the aim of this thesis is to make a prediction about the future network topology, based on measured link qualities from the past. These measurements are available in form of a *time series*. Generally, a time series is a number of data points, measured in uniform time intervals and can be denoted by

$$\boldsymbol{x} = \{x_1, x_2, x_3..x_k\}, \tag{2.1}$$

where x_n can be a scalar or vector value.

The field of making predictions from an available time series is called *time series prediction* and is an area of research in the field of *machine learning* [14]. An overview of the field of time series prediction is presented

in the introduction chapter of [15]. The basic goal of time series prediction is to generate a model of the process under observation, which is able to predict values that have not yet been measured.

The quest of predicting future values of a time series is a problem which is common in many different areas of research. Be it forecasting the weather or predicting the prices in the stock markets, the task is always to build a model which is able to estimate future values from observations made in the past. Figure 2.1 shows several examples of different time series encountered in different applications which are subject to prediction[4] The different physical systems encountered in different fields which are subject to prediction, lead to a large number of different structures showed by the time series. From almost periodic time series to very complex or chaotic ones, everything is possible.

Historically, the first approach for time series prediction was to find a global model fitting the data best possible. This changed in 1927, when George Udny Yule published his research of an *autoregressive* technique for predicting the annual number of sun spots. He invented the autoregressive prediction which provides an algorithm, that estimates the next value in a series as a weighted sum of previous ones. This method was widely used and perfected until in the 80ies of the last century, when new ideas were developed for cases where autoregression did not fit well. The promising approach of *neural networks*, emerging from research in artificial intelligence, was studied for the detection of patterns in time series. Another approach, *state-space reconstruction* from time-delay embedding, emerged from research in the field of dynamical systems and is used to learn about the underlying structure of a system, where the time series was created by deterministic mathematical equations.

Even though there are many different approaches to time series prediction, one may find some common issues among them, which allows a certain classification of different prediction methods. The following sections will

[4]The examples are taken from the Santa Fe contest on time series prediction issued in the 1990ies by Neil A. Gershenfeld and Andreas S. Weigend. The results of the competition were assembled in the book "Time Series Prediction: Forecasting the Future and Understanding the Past" ([15]). The data sets of the competition together with the information about its origins and the reasons why they were chosen can be found at http://www-psych.stanford.edu/ andreas/Time-Series/SantaFe.html.

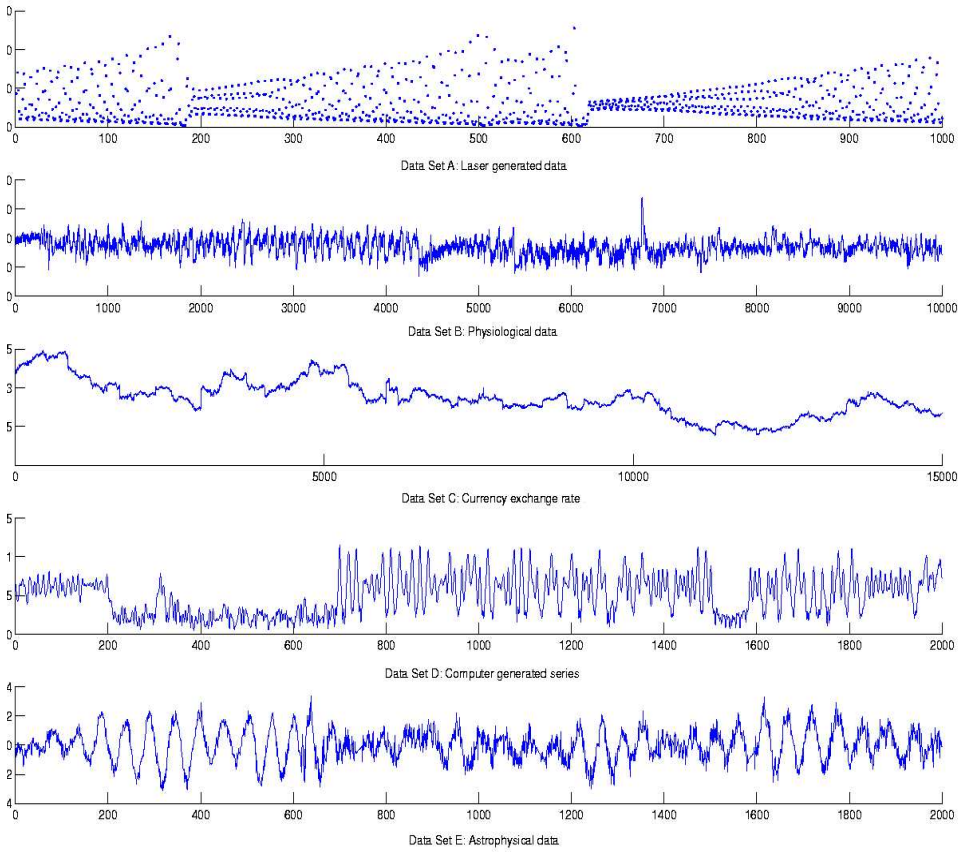

Figure 2.1: Time Series from the Santa Fe time series prediction contest

discuss some general features of different prediction methods and provide a background for the work presented in this thesis.

2.1.1 Understanding vs. Learning

An important criterion for prediction is how much information on the system is available *a priori*. The most comfortable situation is a deterministic system of which the mathematical equations and the initial state are known. In this case, solving the equations will in general lead to an accurate prediction of the future behavior. Unfortunately, these assumptions are rarely fulfilled, and so the rules driving the system and the actual state have to be deduced from observations of the past.

[15] introduces a distinction between *learning* and *understanding*. In some cases it is possible to understand the system, that means to get some mathematical insight of how the system works. In other cases, it is only possible (or desirable) to be able to *learn* from the past, which means the structure in a time series can be emulated without deducing anything about how the system works. An example for a learning system is one that compiles a library of patterns from past measurements and, at prediction time, tries to find the observed pattern in this library.

2.1.2 Lazy Learning vs. Eager Learning

One option is, whether the prediction method should be a lazy learner or an eager learner. Lazy learning basically means, that the input data is not processed until the time a prediction *query*[5] is issued. In [16], the three basic characteristics of lazy learning are described as:

- Lazy learning algorithms defer the processing of their inputs until they receive requests for information. They simply store the input for future use.
- They reply to information requests by combining their stored data.
- They discard the constructed answer and any intermediate results.

[5] A query in this context is a request for a prediction with a given input.

On the other hand, eager learners compile the model for prediction while the measurements arrive. They discard the input values after they are incorporated into the model.

This distinction leads to differences in terms of memory and computing resource usage for lazy and eager learners. Lazy learning requires more memory for storing the history, while eager learning stores only the model, which usually requires little memory. In terms of computing power, eager learning requires a certain amount of computation during the training phase for adopting the model, whereas lazy learning requires almost no computation at that time. At prediction time, eager learning uses very little resources, as the application of the precomputed model is inexpensive, where lazy learners require a bigger amount of computing, as the model has to be generated on the fly.

2.1.3 Global Model vs. Local Model

A further, very important distinction that can be made concerns the model that is created, such as *global model* or *local model* (see [17] for a detailed comparison). Global models describe the relationship between the input and the output values as a single analytical function over the whole input domain. On the other hand, local modeling does not describe the whole physical system in one model, but creates a specific model for a given input. This means that generally not a global model has to be created, but only a model that describes the systems behavior for a given input. Because the input for which the prediction has to be performed is only known at the prediction time, local model algorithms are generally lazy learners.

This concept of global and local models requires some more clarification. In order to explain it, a simple example is assumed, in which, from a set of training data, the next value shall be predicted. For reasons of simplicity, scalar input and output values are assumed. The training data is shown in Figure 2.2 (a). Two possible *global* models, a linear and a non-linear one, are shown in plots (b) and (c). Note that these models take as input the time variable k, thus, if the query is $k = 15$ it gives the value of $y(15)$ at that specific time. Another approach is to create a model which takes as input the last $y(k)$ value and outputs the following value $y(k+1)$ (*autoregression*). A possible global autoregressive model is shown in plot (d) in Figure 2.2. In

this plot, the axes are different to the others. While in the first three plots, there was a y axis for the output and a k axis for the k-th value in the time series, now there is the $y(k+1)$ and the $y(k)$ axis, respectively. Creating a global model generally makes sense if a physical law is assumed behind the process under observation. The benefit of creating a global model is, that it can be stored in a very limited amount of memory. For instance, a linear model $y(k) = ak + c$ requires only two parameters, a and c to be stored. Besides the described linear and non-linear models, another typical representative of global models is a *neural network*.

The goal of *local* modeling is not to get a model which describes the whole process, but instead to simply give a reasonable output for a given input (the *query*). A possible local linear model is shown in Figure 2.2 (e). As time independent local models, in which the next value is predicted based on the query, there are several strategies found in literature. One prominent approach is called the *nearest neighbor*. In this case, in the training data the value which is *closest* to the query is taken and the value which followed this nearest neighbor is taken as the prediction. *Weighted average* models, on the other hand, take an average of the outputs of training values in the neighborhood of the query point, inversely weighted with their distance to the query point. From the description of these examples it is obvious, that the computation of a local model requires a measure of distance between two sample points. The *distance* $d(x_i, q)$ between the i-th measurement and the query point q is a function with a scalar output value. A typical distance is the *Euclidean distance*, defined as

$$d_E(\boldsymbol{x}, \boldsymbol{q}) = \sqrt{\sum_j (\boldsymbol{x}_j - \boldsymbol{q}_j)^2}. \tag{2.2}$$

2.1.4 Iterative Prediction vs. Direct Prediction

Time series prediction algorithms usually predict only the next value of the time series. However, it is often the case that the prediction should go further in the future (*long term* prediction). In this case, *iterative prediction* can be applied, as described in [18]. Iterative prediction basically means, that the result of the *one-step-ahead* prediction is fed back to the input of the predictor, which then uses the predicted value as the base for the prediction of the next step. Repeating this k times leads to a *k-steps-ahead* prediction.

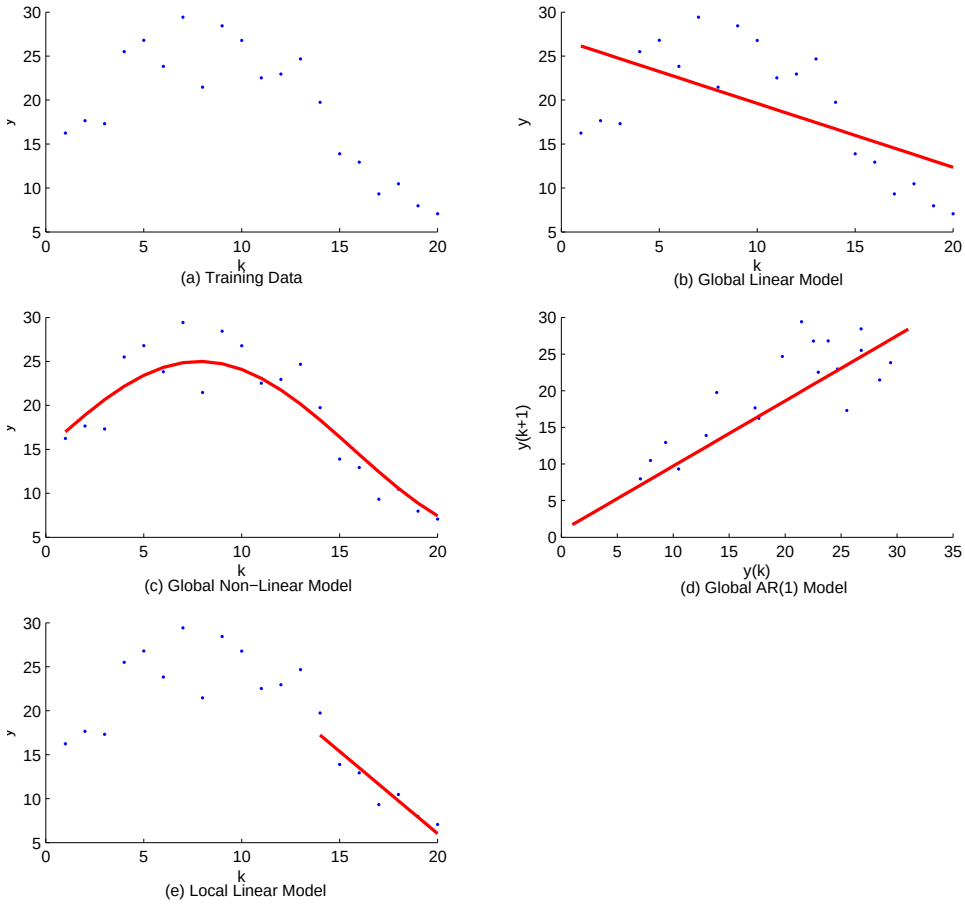

Figure 2.2: Local and global linear and non-linear models

Another way to predict k time steps ahead is to make a *direct prediction* instead of iterative prediction. This means, that the predictor is a function which does not predict only one step, but instead directly predicts k steps ahead.

2.2 Mobility Prediction in MANETs

Mobility prediction in general is the problem of estimating the trajectory of future positions of the nodes in mobile networks. It has been a research topic for some time in different areas, mainly in cellular networks and routing for wireless mobile ad hoc networks. In *cellular networks*, estimating the future position of the mobile nodes helps predicting handovers of mobile nodes from one cell to the next and can be used to reserve resources and speed up the handover process. It is obvious, that the application field of cellular networks operate with vastly different prerequisites for mobility prediction than ad hoc networks, as the structure of network, the hardware of which the networks are built and the behavior of the nodes are fundamentally different. However, it reveals that the structure of the problem of mobility prediction is the same, whether used in wireless networks with fixed infrastructure or in wireless mobile ad hoc networks.

2.2.1 Assumptions

The common structure of the mobility prediction problem comes from the fact that all application scenarios share some very general assumptions:

1. The current mobility states of the nodes can be observed.

There exist a broad range of possible parameters which can be used for state observation, ranging from the Received Signal Strength (RSS) of the radio signal to the absolute geographic location, speed and moving direction of a node measured by a GPS receiver. While the choice which parameter is used is usually restricted by the structure and the capabilities of the hardware of a network, in any case an input space of variables which are used to observe the mobility state can be defined.

2. The behavior of the nodes show some pattern.

This assumption is essential, because it intuitively seems impossible to predict the future state of the network if the nodes behave completely random. Here also, a wide range of patterns can be observed, ranging from the possibilities, that the network is located in an office building and the nodes usually move along the corridors and stop in the offices, or the network is located on a freeway and the nodes have only the possibilities to move along the street. This pattern allows to map the past and current behavior of a node to its future state.

To pin down these observations, the general definition of mobility prediction can be written as:

$$Pred : Mob\_State_{past,current} \mapsto Mob\_State_{future} \tag{2.3}$$

Additionally to the described assumptions, which are valid for mobility prediction in any type of network, there are some others coming from the fact of operating in wireless mobile ad hoc networks. In order to define these, again, a short glance over the fence to mobility prediction in cellular networks can help by pointing out the differences to the ad hoc environment. While having information about the geographic conditions of a cell in cellular networks, in MANETs very little or no clue at all about the physical environment of the network is available. While in cellular networks one end of a communication link (the base station) has fixed and known position, in the ad hoc case both ends of the link are assumed to be mobile. And finally, while mobility prediction in cellular networks is usually the affair of the fixed part of the network (the network has to reserve resources on a base station), in case of a MANET each node predicts its own future neighborhood distributedly. These differences between cellular networks and MANETs show that in MANETs the prerequisites for mobility prediction are more hostile, as there is less information on which the prediction can be based. However, there is a ray of hope that even in MANETs the nodes do not behave completely random. This comes from the fact that their movement is restricted by geographical properties of the network environment and the intent of the users carrying around the nodes, and it will be further clarified in Section 3.1.

Having described the differences between cellular environments and MANETs, it is now time to define some additional assumptions which are vital for mobility prediction in wireless mobile ad hoc networks:

3. There is no a priori information about the geographic environment of the network.

4. Each node in the network is mobile.

5. Each node has to predict distributedly the future state of its neighborhood.

As stated in Section 1.2.1, the lack of a device to measure the geographic coordinates of the nodes and having no information about the physical environment in which the network is located are limiting factors for the choice of input parameters which restrict the prediction method to operating with link qualities as a measure of the mobility state of a node. However, this is enough to predict the future network topology in MANETs, taking the past time series of link quality measurements as training data for regularly observed patterns of link quality behavior between two nodes and learning about possible future link qualities. In Figure 2.3, possible time series of measurements in an imaginary network of five nodes from Node A to Node E (for sake of simplicity only the observed link qualities between a few node pairs are depicted) are shown. Thus, each node is supposed to measure the observed link qualities to its neighbors and base the prediction on these observations from the past. Adopting the general form of Equation 2.3, the specific problem with link quality as the measure of mobility state leads to the following equation:

$$Pred : SigQ_{past,current} \mapsto SigQ_{future} \tag{2.4}$$

2.2.2 General Structure of a Mobility Prediction Algorithm

With these prerequisites in mind, now the general building blocks of a mobility prediction algorithm can be defined. Figure 2.4 shows the basic structure of the problem. There are two major parts, the state observation and the prediction. In the following, the tasks of these two parts will be described.

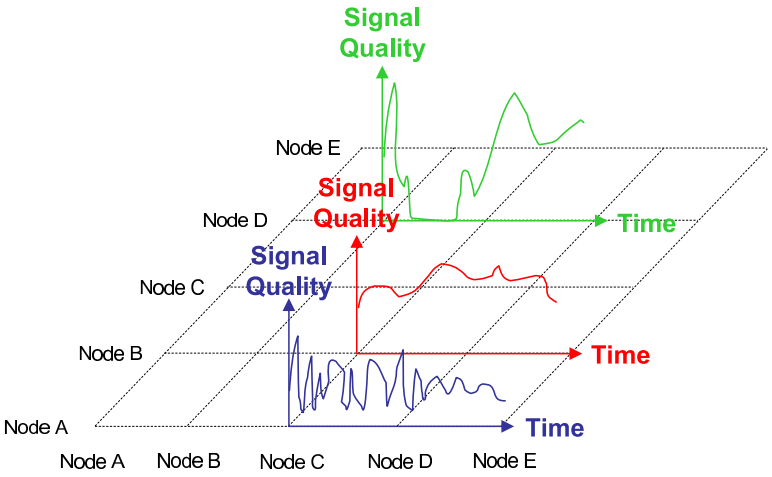

Figure 2.3: 'Trajectory' of signal quality versus time in a visionary network consisting of five nodes

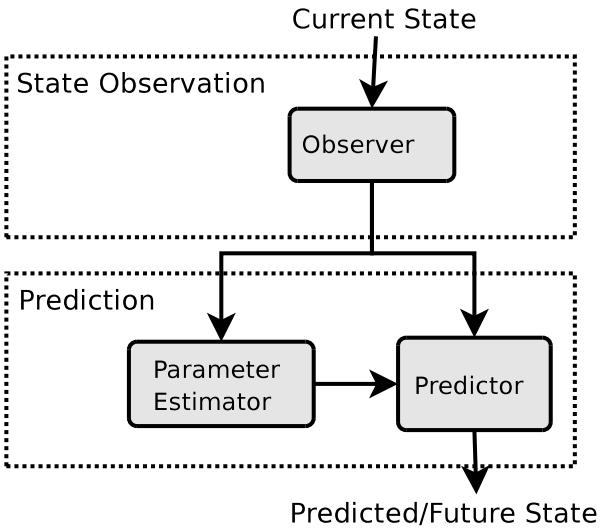

Figure 2.4: The problem of mobility prediction, split in the two major parts of observation and prediction

State Observation

The task of the state observation is to keep track of the mobility state of the node. By doing this, it assures that the first assumption defined above, the observability of the mobility state is fulfilled. Its input comes from the defined set of parameters which form the input space. The output of the Observer is the input of the prediction part. If the observed parameters (the input space of the Observer) are not the same as the input required by the Predictor, the Observer has to make a mapping of the parameters. For instance, if a GPS device is used as the Observer, the time difference of arrival measured from the signals of at least three satellites form the input space (see [13] for an explanation on how GPS works). What the predictor needs is a geographic location, thus the GPS device has to make a transformation between these two quantities, which is the actual process of geolocation with GPS. Another example of mapping of the input to the output space might be, if the input is a measurement of RSS and the output is a time series of measurements of RSS values, the Observer has to store the past values and arrange them in a time series which it can pass to the prediction part.

State Prediction

The prediction part can be split in two major tasks, the first of which is the Predictor. The Predictor makes the actual prediction and is basically the model of the system. Its inputs come from the Observer and from the Parameter Estimator, which is the second part of the prediction. The Parameter Estimator gets as input the training data from the Observer and computes from this a set of parameters of the system model. As a simple example for clarifying this, consider a linear model of a nodes movement. In order to make a linear prediction of geographic position, the system model needs the actual coordinates as input and the speed and acceleration as parameters. So the Parameter Estimator gets a time series of measured coordinates, computes the velocity and moving direction from this series and hands them over to the model as parameters. With these parameters and the input being the actual position, the linear model is able to predict the future position of the node.

2.3 Application of Mobility Prediction: Server Selection

Because of the mobility of the nodes and the lack of fixed infrastructure, MANETs are especially challenging environments to run services. The classical client server architecture is not well suited in such networks, as (1) there are no dedicated server nodes and (2) as the nodes are free to move and even leave the network whenever they want, the server may simply disappear and disrupt the service. A promising new architecture for MANETs is the *zone based architecture* [19] which is able to cope with these challenges. With this approach, the network is split in zones, each of which contains a special node, the *zone server* being responsible for its zone. Because of the redundancy with having several servers in the network, each of which is responsible for a small number of nodes, this architecture is more stable than the classical centralized server approach and therefore better suited for wireless mobile ad hoc networks. In the following, an overview of the *Priority Based Selection (PBS)* [5] algorithm, which distributedly creates such a set of zone servers, will be given. As an application of mobility prediction in MANETs, an extension of PBS, increasing the stability of the selected set of zone servers, is described in Section 3.3.

The basic idea behind the PBS algorithm is to calculate a *Dominating Set (DS)* (recall that a DS in graph theory is defined as a subset of the nodes of a graph, such that each node is either part of the DS or has a neighbor which is part of the DS). This Dominating Set is then used as a set of servers, each building a zone with its direct neighbors. Thus, each node is either a server itself or has a direct link to a server. In order to construct the DS, PBS defines the following four states in which the nodes of the network can be:

DOMINATOR - The node is in the DS and will act as zone server.

DOMINATEE - The node is not in the DS but is covered by one or more DOMINATOR nodes (it has at least one DOMINATOR neighbor).

INT_CANDIDATE - The node participates in the service and does not yet have a DOMINATOR or DOMINATEE state. It is an internal candidate to become one of them.

EXT_CANDIDATE - The node does not participate in the service but it is possible that the algorithm chooses the node as DOMINATOR. Thus, the node can be considered as an external candidate[6].

In order to determine the states of the nodes distributedly, each node keeps track of its neighborhood and maintains a *Neighborlist*. In this Neighborlist, the node stores information about its direct neighbors:

ID - The unique ID of the neighbor node.

Address - The network address of the neighbor node.

Node weight - The weight of the neighbor node.

Span - The span value of the neighbor node.

State - The state of the neighbor node.

One thing that deserves special attention is the *node weight*. Saying that PBS constructs a Dominating Set is only half of the truth. What it really does is constructing a *Weighted Dominating Set*. This means that each node of the graph is assigned a weight, which indicates how good the node is able to act as server for a given service. The weight is assigned depending on certain parameters, such as the *computing power*, available *memory resources*, *battery lifetime* and the *position in the network*. How this is done is further explained in [20].

The PBS algorithm performs in rounds, until each node is either in DOMINATOR or DOMINATEE state. Each round consists of exchanging the Neighborlist with its neighbors and based on this deducing its own state. The pseudo-code of the algorithm for one specific node v is shown in Listing 2.1. In line 6, the condition for switching to DOMINATOR mode is given as having the *highest priority*. This requires further explanation. As already mentioned, a Weighted Dominating Set should be constructed, thus, the priority to switching to DOMINATOR state is bound on having a high node weight. However, PBS does not stop here, but also defines what should be done in case of several nodes having the same node weight. The following hierarchy of priorities for being server is used:

[6]Even non-participating nodes can be selected as servers because a basic assumption of the PBS algorithm is, that all the nodes in the network are cooperative and willing to help running a service.

```
1: status = INT_CANDIDATE or EXT_CANDIDATE;
2: while v has INT_CANDIDATE neighbors within distance 2 do
3: – send neighborlist;
4: – receive neighborlist;
5: – change to DOMINATEE, if neighbor is DOMINATOR;
6: – change to DOMINATOR, if v has highest priority within
     distance 2 among the nodes with INT_CANDIDATE status;
7: od
```

Listing 2.1: The PBS algorithm

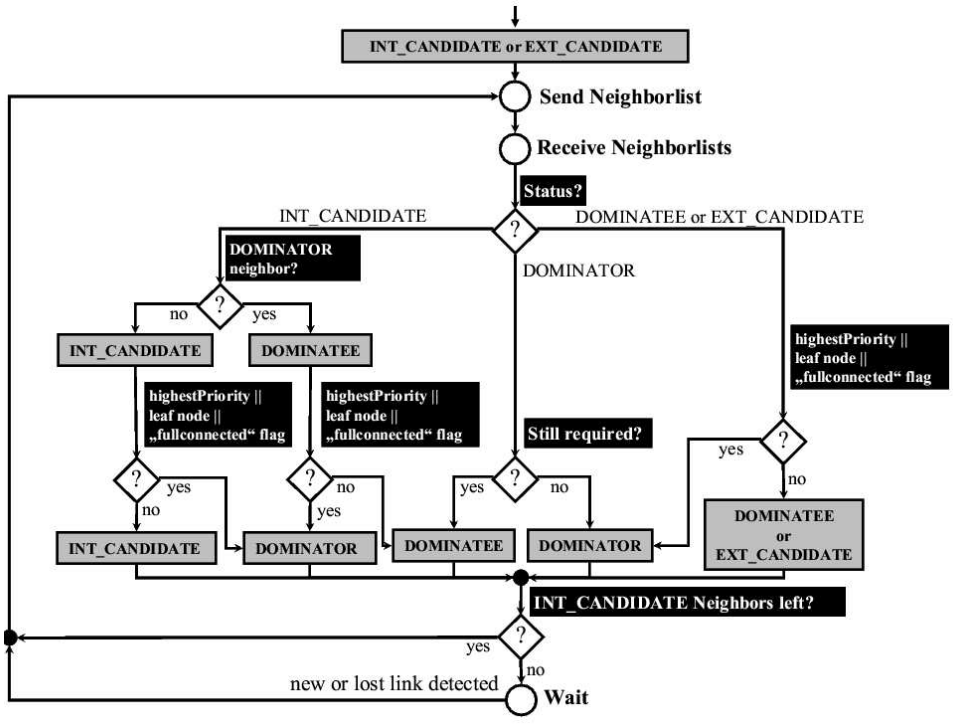

Figure 2.5: Flow chart of the PBS algorithm

1. The node has a higher node weight;
2. If tie: the node has a higher span[7] value;
3. If tie: the node has more neighbors with DOMINATOR state;
4. If tie: the node has a lower ID.

Figure 2.5 shows a detailed flow chart of the PBS algorithm. How prediction can help to increase the stability of the Dominating Set will be further described in Section 3.3.

2.4 Related Work

In the literature, different approaches to mobility prediction in wireless networks have been proposed. This section explains different methods observed mainly in the fields of cellular networks and routing in ad hoc networks. The purpose of this overview is not to be complete, but to explain with examples taken from the literature the different approaches the problem of mobility prediction offers.

2.4.1 Mobility Prediction with a Linear Model

Creating a linear (in time) mobility model of mobility of the nodes means basically assuming that the probability that the nodes keep on moving in the same direction and with the same speed they currently have. In mobile ad hoc networks, determining the current speed and moving direction of the nodes usually requires special hardware like a GPS device. Such a method has been observed in different mobility prediction algorithms, for example in [10]. In this approach, different schemes to improve routing protocol performance by using mobility prediction are proposed. The *expiration time of a link* is calculated with the assumption of having the GPS position information of both ends of the link. With assuming a free space radio propagation model, where the received signal strength solely depends on the distance

[7] The *span value* is the number of nodes which could switch from INT_CANDIDATE to DOMINATEE state if the given node would switch to DOMINATOR state. This priority is used in order to get a Dominating Set with the least number of Dominators possible.

between sender and receiver, the amount of time two mobile hosts will stay connected can be computed with a simple formula:

$$D_t = \frac{-(ab+cd) + \sqrt{(a^2+c^2)r^2 - (ad-bc)^2}}{a^2+c^2}, \tag{2.5}$$

where
$a = v_i cos\Theta_i - v_j cos\Theta_j, b = x_i - x_j, c = v_i sin\Theta_i - v_j sin\Theta_j, d = y_i - y_j.$

2.4.2 Mobility Prediction with an Autoregressive Model

In [21], a mobility tracking[8] scheme based on an *autoregressive model* is described. Estimation of the position, velocity and acceleration of the mobile station in a cellular network is accomplished with an extended Kalman filter[9]. The Kalman filter used in this example applies an autoregressive model of the mobility state of the mobile node.

An autoregressive model of order p defines the n-th value as a weighted sum of the p previously measured ones and is mathematically defined as

$$x_n = \alpha_0 + \sum_{i=1}^{p} \alpha_i x_{n-p} + \epsilon_n, \tag{2.6}$$

where ϵ_n is an independent identically distributed noise term with zero mean. Autoregression will be further discussed in Section 3.2.2.

In a cellular network, where one end of the link is at fixed position (the base station) and the other one is mobile, such an autoregressive model leads to good results for mobility tracking. Experiments with an autoregressive model of the link qualities used not only for mobility tracking, but also for iterative prediction have shown, that it is hard to tune the parameters like the model order p.

[8] Mobility tracking is the task to determine a trajectory of the mobile nodes' position in time.

[9] Kalman filters are used for determining the actual values in a set of noisy measurements and are further explained in Section 3.2.2.

2.4.3 Mobility Prediction with Neural Networks

A *neural network* (see e.g. [22]) is a network of simple processing elements (neurons) which can exhibit complex global behavior. The idea behind neural networks historically was to imitate the central nervous system in its way of performing operations. Although current neural networks do not follow this analogy in detail, they still have in common with the central nervous system, that the tasks are performed collectively and in parallel by the units, instead of assigning each of them a certain subtask. Just as the human brain, also neural networks are well suited for pattern recognition, which makes them useful for mobility prediction.

A mobility prediction algorithm for cellular networks based on a back-propagation neural network[10] has been described in [23]. In this approach, the moving trajectory of a mobile node is determined as a sequence of base stations the node was attached to. The neural network is trained with sequences observed in the past in order to detect the current movement pattern in the past behavior of the node.

2.4.4 Mobility Prediction with Pattern Matching

Another approach for prediction based on pattern matching was proposed in [24]. The algorithm was designed for the use in cellular networks, and adapted for smart environments[11]. It uses an information theoretic approach to mobility tracking and prediction. The approach is similar to the one presented above in terms that it uses the history of the base stations (or closest sensors) for encoding the trajectory of user movement. However, instead of using a neural network for pattern recognition, it uses the LZ78

[10] The main idea behind a back-propagation network is, that it starts out with a random pattern encoded in it and as it is trained modifies this random pattern based on how well the pattern performs on the training data. In other words, the neural net starts with guessing what the output should be given a certain input and then compares its guess with the desired output. Depending on how far off the guess is, the network adjusts its internal state and proceeds to the next training point.

[11] A smart environment is one that is able to acquire and apply knowledge about humans and their surroundings, and also adapt to improve their experience. Examples are smart homes, smart offices, etc.

compression algorithm[12] to generate a sort of dictionary of observed paths in the past.

2.5 Chapter Summary

The chapter presented an overview of the basic concepts used in this thesis. The principles of time series prediction were explained in order to get some insight in the different options one has to create a model of a link for prediction. Furthermore, some basic assumptions which are used for mobility prediction in MANETs were given. These assumptions will be vital for the explanations of the design concepts in the next section. As a base for the application of mobility prediction to server selection, an overview of the PBS algorithm has been given. Finally, related work in the field of mobility prediction has been shown.

Based on these prerequisites, the next chapter will explain the approach taken in this thesis and show how prediction can be integrated in the PBS algorithm to create more stable Dominating Sets.

[12]The LZ78 algorithm for lossless compression was published in 1978 by Lempel and Ziv and is based on Shannon's entropy. Variants of it are widely used today for instance in the Unix 'compress' utility and in the GIF image format.

3

Design Concepts

This chapter explains the design concepts of the developed mobility prediction method. Mapping the following sections to the general structure of a mobility prediction algorithm shown in Figure 2.4, Section 3.1 deals with the part of the state observation. Section 3.2 covers the prediction part with the estimation of the model parameters described in Section 3.2.1 and the model itself in Section 3.2.2. After making the prediction of future link qualities, one step is missing in order to use the algorithm for distributed server selection. Section 3.3 explains the concept of the link stability criterion, and how it is integrated in the PBS algorithm to get a more stable Dominating Set.

3.1 State Observation

As elaborated in Section 2.2, the variable under observation is the *Signal to Noise Ratio (SNR)* of the links of a node. Predicting the SNR was chosen as this is an appropriate measure of the quality of a link, not only taking in account the *Received Signal Strength (RSS)*, but also to amount of *noise*, which can prevent a link from being established even though the RSS is high. This allows to predict the changes in the network topology[13] directly instead of having to conclude this information from a geographical distance of the nodes by means of a radio propagation model.

[13]Topology in a sense of stating whether or not there is a connection between two nodes of the network, not in a geographical sense.

3.1.1 Lazy Learning

Figure 3.1 shows an example of a time series, measured from a link in a MANET in a typical office environment[14], with two nodes moving around on the floor and pausing in the offices.

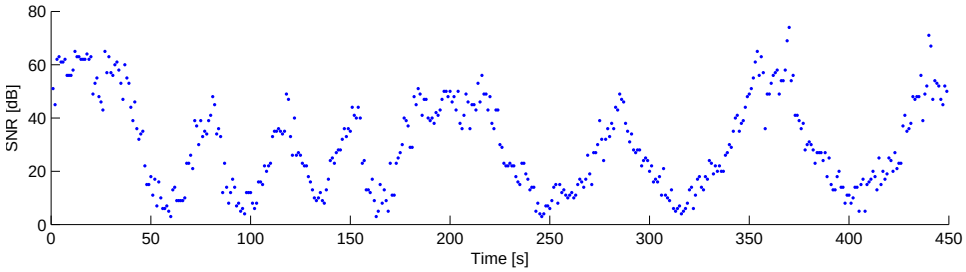

Figure 3.1: Measurement of Signal Strength in an office environment

With this example of how a time series of measurements might look and remembering the information about time series prediction in Section 2.1, it is now time to clarify what of the described options is best suited for mobility prediction in MANETs. The concepts are explained with the example of a MANET in an office environment[15].

The most important aspect is to realize, that there are *two driving forces* which determine the patterns of behavior of the nodes. The first is that they are *restricted by the physical environment*. In an office building the people carrying the nodes are usually walking along the floors (for instance, going to the printer to fetch a document, walking to the coffee corner and returning to their office). Another physical restriction is that they will never move with speed larger than about 2 m/s. These restrictions are reflected in the patterns observed in the measurements. The second driving force is the *intention of the user*. For example, when the user passes a door of an office (s)he decides based on intentions, whether (s)he wants to enter the office or continue walking down the floor.

[14] The measurement was performed in the SIRAMON testbed.

[15] An office environment might not be the most typical environment of a mobile ad hoc network, as in such an environment usually an infrastructure network exists. Though this scenario is well suited for the explanation of the concepts and the statements are valid for other physical environments, (e.g. buildings, cities, freeways, trains, schoolyards, etc.) as well.

In Section 2.1.1, the difference between *understanding* the underlying process from a time series and *learning* from it was explained. Understanding would mean, that from the past measurements a model is created of where in the office the nodes are and what the probabilities of the intention of the users are. For example, the model should recognize a situation where one end of a link is currently in the coffee corner (the user is drinking a coffee) and the other is somewhere on the floor. Additionally, the model should have learned from the past, that the user walking down the floor enters with a certain probability a certain office or will walk to the exit and leave the building with another probability. To achieve this seems not realistic as there is no a priori information about the physical environment (assumption 3 in Section 2.2.1). On the other hand, learning from the past seems more realistic, as there clearly are some repeated patterns visible in Figure 3.1. This means, when looking at the recent past, trying to find a similar pattern in the training data is possible. Having found such a pattern match in the past and assuming that the link will behave similarly, a prediction by learning instead of understanding seems reasonable.

Thus, the task is to detect patterns similar to the currently observed behavior in the past. Addressing the question of *lazy learning* versus *eager learning*, this means that for eager learning a set of observed patterns would be compiled during the measurements. While this is a viable approach in certain cases (as, for instance, the work described in Section 2.4.4), with having the noisy measurements of SNR identifying patterns is hard. Existing methods to classify similar data[16] require that a fixed length of a pattern would have to be defined before classification. This would mean that the training data would have to be split in a number of patterns of predefined length and then grouped. With lazy learning this classification can be circumvented, which is a benefit. With the knowledge about the recent past of the SNR curve, a lazy learner can create a *local model* of the link by searching for similar patterns in the history and assume that the nodes repeat their behavior with a high probability.

Looking back at Section 2.1, where the options of understanding vs. learning, lazy learning vs. eager learning and local modeling vs. global

[16]'Statistical classification' is the field of grouping objects or, in this case, patterns according to their similarities. Examples for classification algorithms are neural networks, decision trees, hidden markov models, etc.

modeling were presented, the choices have now been justified to *learn* with a *lazy learner* and create a *local model*. One question that remains is, what this learner should learn from, whether the prediction for one specific link should be performed with taking only the history of this very same link into account, or base the prediction on the information from all the links of the node. The first would be a good idea, if the links showed fundamentally different behavior and therefore the information contained in the history of one link is not useful for another link. However, in a MANET the assumption that *different links behave similarly* is reasonable, therefore the available information about all the links should be taken into account for the prediction.

3.1.2 Signal to Noise Ratio

One benefit of taking the SNR as a measure of the mobility state of a node is that it can be easily measured. For instance, in wireless LAN network interfaces according to the 802.11 standard [9], the firmware and driver usually provide some measurements of signal strength and background noise observed on the channel. In the following paragraphs, a short overview of the physics behind the SNR is given.

The SNR is generally defined as

$$SNR[dB] = 10log_{10}(\frac{P_{signal}}{P_{noise}})[dB], \tag{3.1}$$

where P_{signal} is the power level of the signal and P_{noise} is the power level of noise, respectively. This can also be written as

$$SNR[dB] = 10log_{10}(\frac{P_{signal}}{P_0})[dBm] - 10log_{10}(\frac{P_{noise}}{P_0})[dBm], \tag{3.2}$$

with $P_0 = 1\ mW$ being the reference power.

The signal power is influenced by several parameters of the communication system. At the sender, it is depending on the *transmission power* of the sending device and the *antenna gain*. During propagation, the signal experiences a *propagation loss*, which is usually modeled with the *freespace*

model [17]. It is further influenced mainly by three effects, *reflection*, *scattering* and *diffraction*. Together, all these physical effects are building the *radio propagation model*. Therefore, the SNR is connected to the current mobility state of the two nodes, sender and receiver, by a propagation model (so assumption 1 in Section 2.2.1 is fulfilled), but there is no information about how this propagation model looks, as it is influenced by the unknown physical environment (assumption 3). On the other hand, the noise power is usually modeled as *receiver noise* (e.g. thermal noise in the receiver modeled as Additive White Gaussian Noise - AWGN), *environmental noise* (from different sources in the environment, usually also modeled as AWGN) and *interference* caused by other transmissions on the same channel or nearby, overlapping channels. Thus, P_{noise} can be written as

$$P_{noise} = P_{interference} + P_{environment} + P_{receiver}. \tag{3.3}$$

After this little overview of what the SNR is, it is now time to describe the concepts behind the state observation part of the prediction algorithm.

In order to get a time series of the history of SNR, each node has to make measurements at fixed time intervals of length T defined in the following:

Definition 3.1 *The* ***measurement interval T*** *is the time between two sequent measurements performed for one link of the node.*

T is a design parameter of the algorithm and was chosen to be 1 second. The reason for this choice is to have a balance between having too frequent measurements which increases the computational costs and having too little measurements which means loosing information about the behavior of the node (e.g. quick changes of moving direction). Assuming that the behavior of the users carrying the nodes generally have a length of at least several seconds (for instance, walking down a floor would be such a pattern), the choice of $T = 1\ s$ seems reasonable.

In order to account for breaking links, the special value of $SNR = 0$ is defined for having no connection between two nodes. That means, each node, even if the connection to a neighbor breaks, keeps the history of the

[17] The freespace loss is proportional to the square of the distance between the transmitter and receiver.

according link and fills it with zeros in order to have the information when and for how long the connection broke. Having this information is important to predict failing links and therefore is essential to estimate the future network topology.

As the resources of mobile devices are generally scarce, the number of measurements which are stored must be limited. The history of each link is therefore stored in a *circular buffer* of N elements. This means that after the time $N \times T$, the buffer is full and the new measurements start to overwrite the oldest ones. N is another design parameter, which has to be chosen as a balance of memory use and having enough training data for the prediction. Assuming that each measurement is stored as a *float* value, typically using 4 Bytes of memory, reserving 8 kBytes of memory allows to store 2048 measurements per link. With $T = 1\ s$ this results in storing roughly the last 35 minutes of each link. Assuming that each node has no more than about 10 connections, this requires approximately 80 kBytes of memory for storing the time series. Even the most limited devices should nowadays have this amount of free memory available, so $N = 2048$ seems a reasonable choice.

This is how a node stores the history of its links in the state observation. When the state observation gets a *prediction request*, it hands over two things to the prediction algorithm: the *training data* and the *query*.

Definition 3.2 *The **training data t** are the measured time series of all links of the node from the past from time $t = (n - NT) \dots n$, where n is the **query time**.*

Definition 3.3 *The **query q** is the recent part of the time series of measurements, which is used for creating the model. The **query order** o is the length of the query, that means the number of measurements which are used to create the model.*

In Equation 2.4 the training data is represented by $SigQ_{past}$ and the query is $SigQ_{current}$. The plot in Figure 3.2(a) visualizes the training data[18], query and query order for an example training data.

[18]Note that for reasons of simplicity the training data of only one link is shown.

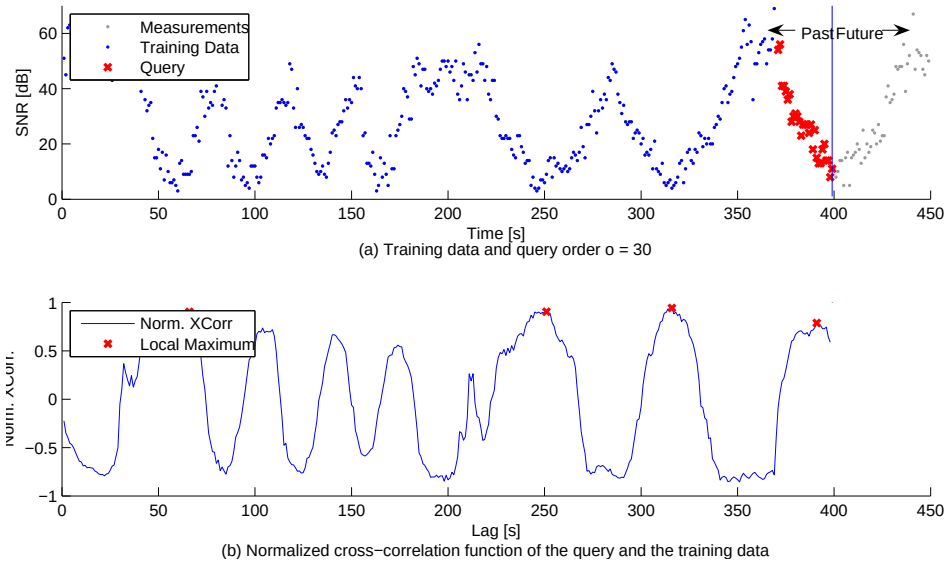

Figure 3.2: Training data and query, normalized cross-correlation

While the length of the training data is determined by the size of the circular buffer in which they are stored, the question of the query order o is yet another, very important design parameter. The choice of o will be discussed in the evaluation part (Section 5.2) of this thesis.

3.1.3 Smoothing with Kalman Filter

One problem that arises with this approach is, that the measurements of the training data are noisy. In order to get rid of this noise as far as possible, the predictors are filtered with a *Kalman filter*[19]. This filtering of the training data presents a little deflection of the pure lazy learning principle, as the measurements are manipulated before the time of prediction. Thus, the approach may be called *almost lazy learning*. What follows is a general overview of how Kalman filters work and a closer description of the Kalman filter designed especially for filtering the SNR values.

[19]Kalman filters were first published in 1960 by K.E. Kalman, who developed them for space craft navigation. Since then, they turned out to be useful in various application areas like GPS, radar, etc.

Kalman Filter

Kalman filters are generally used to determine the system state that can be observed only indirectly or inaccurately. An introduction can be found in [25]. In the following discussion, vectors of $\mathbf{R}^n$ are generally printed in bold and matrices are written with capital letters. The general problem, which is addressed by the Kalman filter is to determine the state $\mathbf{x} \in \mathbf{R}^n$ of a system given by the equation

$$\mathbf{x_k} = A\mathbf{x_{k-1}} + B\mathbf{u_{k-1}} + \mathbf{w_{k-1}}, \tag{3.4}$$

with measurements $\mathbf{z} \in \mathbf{R}^m$

$$\mathbf{z_k} = H\mathbf{x_k} + \mathbf{v_k}. \tag{3.5}$$

A is the $n \times n$ *system matrix*, which is determined by the nature of the system. B is an $n \times l$ matrix, that relates the optional control input $u \in \mathbf{R}^l$ to the state $\mathbf{x}$. The $m \times n$ matrix H relates the actual state of the system to the measurement $\mathbf{z}$. $\mathbf{w_k}$ and $\mathbf{v_k}$ are random variables, which are *independent, white and with normal distributions*

$$p(w) \sim \mathcal{N}(0, Q), \tag{3.6}$$

$$p(v) \sim \mathcal{N}(0, R). \tag{3.7}$$

$\mathbf{w}$ represents the *process noise* and $\mathbf{v}$ represents the *measurement noise*.

In order to filter out the noise and get an estimate of the system state which is more accurate than the measured state, the Kalman filter essentially is estimating the system state as a *linear combination* of a prediction (using the system model described above) and the measured value. In order to get an appreciation about the Kalman filter, it is vital to understand that, while the exact state $\mathbf{x_k}$ is not known at any time, there are *three* estimates of $\mathbf{x_k}$. The first is the measured value $\mathbf{z_k}$, which is related to $\mathbf{x_k}$ by Equation 3.5. However, because of the noise term in this equation the mapping from $\mathbf{z_k}$ to $\mathbf{x_k}$ is not so straight forward. The second is an *a priori* estimate of $\mathbf{x_k}$ and is denoted with $\hat{\mathbf{x}}_\mathbf{k}^-$. A priori means in this context:

without taking the information from the measurement in account. $\hat{\mathbf{x}}_\mathbf{k}^-$ is predicted[20] by the system Equation 3.4. The third value is the *a posteriori* estimate representing the linear combination of the measurement and the *a priori* estimate and denoted by $\hat{\mathbf{x}}_\mathbf{k}$.

The Kalman filter works *recursively* with two steps: the *time update*, in which the a priori value is predicted, and the *measurement update*, in which the prediction is corrected with having the measured value of the state. These two steps are illustrated in Figure 3.3. This recursive function of the Kalman filter is a nice feature for filtering the SNR measurements, as each step requires only little computation power, whereas filtering the whole time series at the same time would be costly.

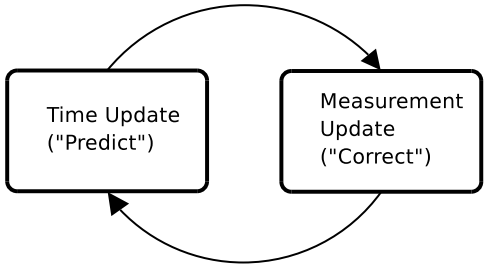

Figure 3.3: Kalman filter - time update and measurement update cycle

The a priori and a posteriori state estimates both contain an *error* (a priori error and a posteriori error, respectively)

$$\mathbf{e}_\mathbf{k}^- = \mathbf{x}_\mathbf{k} - \hat{\mathbf{x}}_\mathbf{k}^-, \tag{3.8}$$

$$\mathbf{e}_\mathbf{k} = \mathbf{x}_\mathbf{k} - \hat{\mathbf{x}}_\mathbf{k} \tag{3.9}$$

with the a priori estimate error covariance and a posteriori estimate error covariance

[20]Note that the Kalman filter has some inherent predictive power, as the whole process of assuming a model, setting its parameters from the past measurements, applying this model to estimate the next value, is basically the same process which is described in this whole thesis for prediction. However, while the autoregressive model used for predicting the next value of the time series in the filter proves well for one-step-ahead prediction, experiments with using it in an iterative way to produce a k-step-ahead prediction have shown that its performance is not optimal.

Time update step	**Measurement update step**
$\hat{\mathbf{x}}_{\mathbf{k}}^{-} = A\hat{\mathbf{x}}_{\mathbf{k-1}} + B\mathbf{u}_{\mathbf{k-1}}$	$K_k = \frac{P_k^- H^T}{HP_k^- H^T + R}$
$P_k^- = AP_{k-1}A^T + Q$	$\hat{\mathbf{x}}_{\mathbf{k}} = \hat{\mathbf{x}}_{\mathbf{k}}^{-} + K_k(\mathbf{z}_{\mathbf{k}} - H\hat{\mathbf{x}}_{\mathbf{k}}^{-})$
	$P_k = (I - K_k H)P_k^-$

Table 3.1: Kalman filter time update and measurement update equations

$$P_k^- = E[\mathbf{e}_{\mathbf{k}}^{-}\mathbf{e}_{\mathbf{k}}^{-\mathbf{T}}], \tag{3.10}$$

$$P_k = E[\mathbf{e}_{\mathbf{k}}\mathbf{e}_{\mathbf{k}}^{\mathbf{T}}]. \tag{3.11}$$

In the measurement update step, the linear combination of the a priori estimated state and the measured state is calculated as

$$\hat{\mathbf{x}}_{\mathbf{k}} = \hat{\mathbf{x}}_{\mathbf{k}}^{-} + K_k(\mathbf{z}_{\mathbf{k}} - H\hat{\mathbf{x}}_{\mathbf{k}}^{-}). \tag{3.12}$$

The term $(z_k - H\hat{x}_k^-)$ is called the *measurement innovation* and K_k, an $n \times m$ matrix, is called *gain*. The gain matrix is calculated in each time step in order to minimize the a posteriori estimate error covariance (see Equation 3.11):

$$K_k = \frac{P_k^- H^T}{HP_k^- H^T + R}. \tag{3.13}$$

In the time update step, the a priori prediction of the state is computed as

$$\hat{\mathbf{x}}_{\mathbf{k}}^{-} = A\hat{\mathbf{x}}_{\mathbf{k-1}} + B\mathbf{u}_{\mathbf{k-1}}. \tag{3.14}$$

A summary of the time update and measurement update steps is given in Table 3.1.

Applying a Kalman filter to the measurements of signal quality means that the parameters A, B, H, Q and R have to be set in a reasonable way. One of the parameters, in this case, is trivial. The matrix H relates the system state to the measurable quantity in the case of indirect observation of the system. However, as the SNR is measured directly, H can be set simply as a unitary matrix. So, what remain are the variance of the process noise Q, the variance of the measurement noise R, the system matrix A and the matrix B. These will be discussed in the following.

Autoregressive Link Model

As the system model, an *autoregressive model* (see 2.4.2) of *order 1*, denoted by $AR(1)$, was chosen. An $AR(1)$ model describes a linear dependency of the next value on the latest measured value:

$$x_{k+1} = \alpha x_k + c + w_k \tag{3.15}$$

The parameters α and c in the case of a scalar x_k are both scalar values. They are calculated using the *least squares method* [26], which minimizes the mean squared error of the last o measured values $x_{k-o-1} \dots x_{k-1}$ to the estimated values. In case of the $AR(1)$ model, the least squares method minimizes the following equation:

$$S = \sum_{i=1}^{o} (x_{k-i} - \hat{x}_{k-i})^2 = \sum_{i=1}^{o} (x_{k-i} - (\alpha x_{k-i-1} + c))^2, \tag{3.16}$$

where $\hat{x}_k$ denotes the estimated value at step k. S gets minimal, when α and c are chosen according to the following equations:

$$\alpha = \frac{\sum_{i=0}^{o-1} x_{k-i} x_{k-i-1} - (o-1)\bar{x}^2}{\sum_{i=0}^{o-1} x_{k-i-1}^2 - (o-1)\bar{x}^2}, \tag{3.17}$$

$$c = (1-\alpha)\bar{x}, \tag{3.18}$$

where $\bar{x}$ denotes the mean of the vector of the o latest training samples.

These model parameters are recomputed at every filter step, thus for every measurement made. Hence, the autoregressive model presents a *local*

model of the SNR. For the integration of the AR(1) model in the Kalman filter equations, the model has to be put in the form of Equation 3.14 for the time update step. One possibility to do this would be to set $A = (\alpha\ c)$ and $B = (0\ 0)$. In this case, the system state x_k would become a two dimensional vector $\boldsymbol{x_k} = \begin{pmatrix} x_k \\ 1 \end{pmatrix}$, so that the following can be written:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} = (\alpha\ c) \begin{pmatrix} \hat{x}_k \\ 1 \end{pmatrix} + (0\ 0)\, u_{k-1} = \alpha\hat{x}_k + c \tag{3.19}$$

However, the equations can be simplified with a little change. Abusing the control input u_k by setting it constantly to 1, the scalar nature of the system state x_k can be conserved by setting $A = \alpha$ and $B = c$:

$$\hat{x}_k^- = A\hat{x}_{k-1} + B \cdot 1 = \alpha\hat{x}_k + c \tag{3.20}$$

As there is no external control input in the SNR model, there are no conflicts with u_k. So, for simplicity this solution was chosen for the matrices A and B.

The Process and Measurement Noise Covariance

What remains for discussion is the noise covariances of the process and the measurement noise. When designing a Kalman filter, usually the process noise covariance R is determined by preliminary measurements and set offline to a constant value. In wireless networks, the process noise is usually modeled by a normally distributed random variable with typical standard deviation from 4 - 8 dB (cf. [21]). A value of $P = 49$ was chosen for the process noise covariance, which relates to assuming a standard deviation of noise of 7 dB. This seems a reasonable choice, since without having information about the environment of where the network is located assuming a rather high noise value is better.

On the other hand, the measurement noise covariance can be dynamically estimated with the following formula (cf. [21]):

$$Q_k = \frac{\sum_{i=1}^{o} \hat{x}_i - (\alpha\hat{x}_{i-1} + c)}{o} \tag{3.21}$$

Time update step	Measurement update step
$\hat{x}_k^- = \alpha\hat{x}_{k-1} + c$	$K_k = \frac{P_k^-}{P_k^- + R}$
$P_k^- = \alpha^2 P_{k-1} + Q_k$	$\hat{x}_k = \hat{x}_k^- + K_k(x_k - \hat{x}_k^-)$
	$P_k = (1 - K_k)P_k^-$

Table 3.2: Kalman filter time update and measurement update equations for the SNR filter

Having discussed the system matrices A and B, process and measurement noise covariances Q and R, the Kalman filter for the link process can now be formulated. It shows, that using an $AR(1)$ model, the equations are simplified a lot, as all the values are scalar. The resulting equations are summarized in Table 3.2.

Thus, all the building blocks of the state observation are discussed.

3.2 State Prediction

The decision of having a lazy learning algorithm, as discussed in the last section, has influence on the *prediction part* of the algorithm. As visualized in Figure 2.4, the prediction part can be split in two tasks like Parameter Estimation and Model Creation. Parameter Estimation is explained in the following section, while the creation of the model is discussed in 3.2.2.

3.2.1 Parameter Estimation with Cross-Correlation

The parameters which the Parameter Estimator should hand over to the model part of the algorithm are references to points in the history, where a similar situation has been observed. The information it gets from the state observation for doing so are the training data and the query. Thus, the task of the Parameter Estimator is to find in the training data patterns which are similar to the query. In order to do this, a *measure of distance* is required, a distance from the query to the past measurements.

Normalized Cross-Correlation

The use of cross-correlation for pattern recognition is motivated by the *squared Euclidean distance* (see [27]). The squared Euclidean distance between the query $\boldsymbol{q}$ and the piece of training data $\boldsymbol{t_{j,k}}$ [21] at times $m \dots (m+o)$ is defined as

$$d^2_{\boldsymbol{q},\boldsymbol{t_{j,k}}}(m) = \sum_{i=1}^{o} [q(i) - t_{j,k}(m+i)]^2. \tag{3.22}$$

Expanding this equation leads to

$$d^2_{\boldsymbol{q},\boldsymbol{t_{j,k}}}(m) = \sum_{i=1}^{o} q^2(i) - 2\sum_{i=1}^{o} q(i)t_{j,k}(m+i) + \sum_{i=1}^{o} t^2_{j,k}(m+i). \tag{3.23}$$

In this equation, $\sum q^2(i)$ is constant. If also $\sum t^2_{j,k}(m+i)$ is assumed to be approximately constant, then the cross-correlating term

$$c(m) = \sum_{i=1}^{o} q(i)t_{j,k}(m+i) \tag{3.24}$$

is a measure of similarity between the query and the training data at times $m \dots (m+o)$. The time shift m (the time in the training data at which the similarity is calculated) is called *lag*.

However, using the cross-correlation as defined in 3.24 for pattern recognition arises a problem. If the expression $\sum t^2_{j,k}(i)$ in Equation 3.23 varies with the lag m, pattern matching can fail. This is because, even with an exact match between the query and the training data, the cross-correlation can be a smaller value than the correlation between the query and a region of high signal quality. This problem can be solved with using the *normalized cross-correlation*. The normalized cross-correlation subtracts the mean of the query and the mean of the piece of training data under observation and scales the value in order to get results in the interval from $[-1 \dots 1]$:

$$\gamma(m) = \frac{\sum_{i=1}^{o} [q(i) - \bar{q}][t_{j,k}(m+i) - \bar{t}_{j,k}]}{\sqrt{\sum_{i=1}^{o} [q(i) - \bar{q}]^2 \sum_{i=1}^{o} [t_{j,k}(m+i) - \bar{t}_{j,k}]^2}} \tag{3.25}$$

[21] Note that, as the training data $\boldsymbol{t}$ consist of the time series from all links of node j, $\boldsymbol{t_{j,k}}$ represents the time series of the link of node j to node k.

The *normalized cross-correlation function* indicates the normalized cross-correlation for every lag, that means for $m = 0 \dots (N - o)$. An example of the normalized cross-correlation function of query and training data is shown Figure 3.2 (b) on Page 37. Note that, while in the figure only one time series of training data is shown. When the node has k neighbors, this results in creating k normalized cross-correlation functions.

Finding Local Maxima

The plot in Figure 3.2 (b) shows that there are several good matches resulting in local maxima at lags around $m = \{65, 100, 140, \dots\}$ with the global maximum being at $m = 316$ with $\gamma(316) = 0.94$. This rises the question of which of these lags should be applied for creating the model used for prediction. One obvious option is to simply use the global maximum, as this represents the closest match. However, creating the model only based on the best match is not a good choice for two reasons:

- While the global maximum is certainly the series of measurements which are the most similar to the query, this does not obligatory mean that the physical situation at this lag resembles most to the current situation, because the results may be distorted by the noise in the measurements.
- As noted before, one driving force of what kind of patterns are observed in the measurements is the intention of the user. Frequently the user has in a given situation several options of how to behave. A good prediction must account for this and create the model based on what behavior was the most probable in the past. In order to do this, the model must be based not only on one match, but on several.

Hence, the goal of the Parameter Estimator is to hand over a *set* of lags representing good matches to the Model Creation part. Therefore, local maxima of the correlation function have to be determined. Thus, a *threshold* γ_{min} has to be defined, such that m is a good match, if $\gamma(m) \geq \gamma_{min}$.

Definition 3.4 *The **match threshold** γ_{min} is a scalar value, above which the correlation of the query with the training data is considered to be a **match** and is used for the prediction.*

γ_{min} has to be chosen as a balance between being not too strict about what a good match is, as this leads to having none or only a small number of matches, and being not too loose, as this would mean to define situations as matches which are not really similar to the query. Experimenting with different γ_{min} has shown, that a threshold of 0.5 is a good choice. This value is further discussed in the evaluation section (Section 5.2) of this thesis. In order to find the locale maxima of $\gamma(m) \geq 0.5$, first all the regions of m where the normalized cross-correlation function is above that value are determined. Then, for each of these regions the maximum is searched and inserted in the set of lags which are handed over to the modeling part. In Figure 3.2 (b), the maxima found with this procedure are highlighted with red points.

3.2.2 Creating the Local Model

Getting the set of matches from the parameter estimation and the training data from the state observation, the question is now, how the modeling part of the prediction algorithm can create the local model of the link. The model will be based on the parts of the training data following the lags of the matches. This means, if a lag of m is received from the Parameter Estimation, the part of the training data used for creating the model is the measurements from $m \ldots (m+k)$ for a $k-step-ahead$ prediction. So, the first step in modeling is to create, from the set of lags, a *set of predictors*.

Definition 3.5 *If the set of matches contains i lags* $\{m_1 \ldots m_i\}$*, that i parts of the training data* $\{t(m_1 \ldots m_1 + k) \ldots t(m_i \ldots m_i + k)\}$ *form the* ***set of predictors P****. The i-th* ***predictor*** *is denoted by* $\boldsymbol{p_i}$*.*

The parameter k, the length of the predictors, is called the *prediction order*. As discussed below, k determines, how far in the future the prediction will reach. It is a design parameter and can be set according to the needs of the application for which the prediction is used. In case of server selection with PBS, k was set to 40, as this results in maximum stability of the Dominating Set in the simulations. This choice of the prediction order is further discussed in Section 5.4.

The question is now, how the link model can be created from the predictors. In the set of predictors, each $\boldsymbol{p_i}$ represents a past situation where the link was in a similar state as it currently is. It can be assumed, that in

these predictors different patterns of SNR changes appear. The reason for this is that, in a given situation, the nodes have typically several options of how to behave, which will be reflected in the patterns of the predictors. In order to predict the most probable one of these patterns, the pattern which appeared the most often in the past should be chosen. This can be done by looking at which predictor has the most similarities to the other predictors. Again, the normalized cross-correlation is used for measuring the similarity of predictor $\boldsymbol{p_i}$ to $\boldsymbol{p_j}$:

$$\gamma_{i,j} = \frac{\sum_{n=1}^{k} [p_i(n) - \bar{\boldsymbol{p_i}}][p_j(n) - \bar{\boldsymbol{p_j}}]}{\sqrt{\sum_{n=1}^{k} [p_i(n) - \bar{\boldsymbol{p_i}}]^2 \sum_{n=1}^{k} [p_j(n) - \bar{\boldsymbol{p_j}}]^2}} \tag{3.26}$$

A measure of how often a pattern appeared in the past is the *average similarity*, which is for predictor $\mathbf{p_i}$ defined as

$$\gamma_i = \frac{\sum_{j=1}^{N} \gamma_{i,j}}{N}, \tag{3.27}$$

where N is the total number of predictors.

As the prediction, the predictor with the maximum average similarity among all the predictors is chosen. Note that choosing one predictor from the set of predictors directly as the prediction represents in some way a *'the winner takes it all'* strategy, as opposed to, for instance, the *weighted average* approach, where a weighted average over all the predictors is taken (see Section 2.1.3). The reason for choosing this approach is, that in this case the prediction has some clear meaning. In order to understand this, the predictors should be thought of as physical situations. An example of such a physical situation would be that one end of the link is in the coffee corner, while the other end of the link exits an office and walks along the floor. The Parameter Estimator handed over a set of such situations from the past, which are similar to the current. The model creation then chooses one of these situations and predicts that the nodes will behave the same way. By choosing the one which has the most similarities to the others, it makes sure that the situation is chosen which was the most 'common' in the past. Taking an average of the predictions instead would mean in an abstract sense, that some average of the past situations would be calculated. Such an average would no longer represent a clear physical situation. Thus,

this approach is some mixture of the *nearest neighbors* (because it uses only one predictor for the prediction) and the *weighted average* (because not only one, but several neighbors are taken into account) approaches.

Taking one of the predictors directly as the prediction means that the prediction has the same length as the predictors. Thus, if the predictor contains k measurements, a *k-steps-ahead* prediction is performed. Such an approach was defined in Section 2.1.4 as *direct prediction*.

3.2.3 Fallback Model: Autoregression

The question is, what happens if the Parameter Estimator does not find any match in the training data? This can happen due to two reasons:

- The training data are too short, as the order of training measurements has to be at least the length of the query o (for being able to compute the cross-correlation) plus the prediction order k (as the k training samples after a matching part of the training data are used as a predictor).
- The cross-correlation does not contain a value above the match threshold γ_{min}, because the pattern in the query was not observed before.

In such a case, a *fallback model* is created. In order to do this, the inherent predictive power of the Kalman filter in the state observation is exploited. For the Kalman filter, an *autoregressive link model* AR(1) is created in the time update step (see Section 3.1.3). Using this model for an *iterative k-steps-ahead prediction*[22] has the benefit, that the model already exists and can simply be applied. Thus, in any case, even if the current situation was not observed before, a prediction will be available.

3.3 Link Stability

Having discussed the state observation and prediction part of the algorithm, the nodes are now capable of predicting the future SNR values of their links.

[22]Recall that iterative prediction means the predicted next-step-value is used again as input for the model. Doing this k times leads to a k-steps-ahead prediction.

The question now is, how this can be used in the PBS algorithm in order to elect a stable Dominating Set. Note that the link quality prediction could basically be used for other applications and is not limited to server selection. For instance, one might use it in a routing protocol to establish stable routes. However, in this thesis the prediction is used for improving the stability of the Dominating Set.

The basic idea behind the integration in PBS is, that each client should have a link to a server which is predicted to be stable for a certain time. Thus, a link availability criterion is introduced. A link is said to be *stable*, if it is available for the next $k \times T$ seconds, where k is the prediction order and T is the measurement interval. As k is the number of values in the time series of the prediction, in order to check whether a link is stable the prediction simply has to be scanned for zeros[23]. If a zero appears in the prediction, the link is *unstable*.

Looking at the pseudo code of the PBS algorithm in Listing 2.1, only little adaptations have to be done in order to introduce the link availability criterion. The only thing that changes is in line 5, where the state of the node is decided. Instead of

```
5: – change to DOMINATEE, if neighbor is DOMINATOR;
```

the new condition for switching to DOMINATEE state is introduced as

```
5: – change to DOMINATEE, if neighbor is stable DOMINATOR;
```

where *stable*, as defined above, means that the link to the DOMINATOR is expected to remain stable for $k \times T$ seconds.

Note that this link availability criterion for Dominator nodes holds only during the election time. Once all the nodes have decided their state to either DOMINATEE or DOMINATOR, a Dominatee is not required to have a stable link to a Dominator anymore. That is, just because a Dominatee has no more stable link to a Dominator it does not switch back its state to CANDIDATE and triggers a new election round. Letting the criterion be valid not only at prediction time would have the benefit, that the Dominating Set would be updated proactively, before a change is really necessary. Once a node predicts that it might loose its connection to the Dominator, it would

[23]Recall that a zero was defined as a special value for having no connection between two nodes.

already trigger a re-election before the link really breaks. This might prevent a service interruption during the re-election for the node to whom the link broke. However, in this thesis the goal was to increase the Dominating Set stability by decreasing the number of changes in the Dominating Set, and not to improve the service availability.

3.4 Chapter Summary

In this chapter, a link quality prediction algorithm based on time series prediction was described. The measurement of the SNR values of the links was explained and a Kalman filter with an autoregressive link model was introduced in order to get rid of the noise in the measurements. For the prediction part, a method of finding similar situations in the past by pattern matching with normalized cross-correlation was explained and based on taking the most 'common' of these situations in the training data the link model was created. Furthermore, for cases where no predictors are found with the pattern matching method, a fallback solution with autoregressive prediction was introduced. In order to use the link quality prediction for choosing stable Dominating Sets, a link availability criterion for the PBS algorithm was described, such that a node only becomes Dominatee if it has a stable connection to a Dominator. In the following chapters, the implementation in the network simulator ns-2 of this algorithm is explained and based on that the predictions and the Dominating Set stability are evaluated.

4

Implementation

This chapter describes the implementation of the mobility prediction algorithm described in Chapter 3. The algorithm has been implemented in the *network simulator ns-2* in order to be able to investigate its accuracy and impact on server selection with a large number of experiments. A brief overview of ns-2 is given in Section 4.1. For a realistic simulation of the SNR prediction two crucial things have to be considered. First, the nodes have to behave in a reasonable way. As already mentioned, the more random the movements of the nodes the less can be predicted. In order to account for this, realistic *mobility models* which determine the movement have to be chosen. The used mobility models, namely the Random Waypoint Model and the Freeway Model are described in Section 4.2. Second, a *realistic model of the SNR* is necessary in order to get meaningful results. The SNR model of ns-2 is explained in Section 4.3. Finally, Section 4.4 describes the integration of link quality prediction in the PBS Agent.

4.1 Ns-2 Network Simulator

Ns-2 is a free and open source network simulator widely used in research projects. It can be downloaded from the ns-2 homepage [7] and is available for several Unix and Windows platforms. A handy introduction can be found in [28]. The development of ns-2 started back in 1989 and since then the simulator has evolved into a complex and powerful tool supporting a rich number of protocols and applications which is maintained by the VINT (Virtual InterNetwork Testbed) [29] project. Ns-2 is implemented in two

programming languages: the core of the simulator is written in `C++` and for configuring and running the simulations `OTcl`[24] is used.

In a former project, the PBS algorithm was implemented in the `C++` space of the ns-2 simulator (see [5]). Thus, as the final goal of this thesis is to improve the stability of the Dominating Set computed with PBS, this PBS implementation was used as a base for implementing the prediction algorithm. Most of the implementation details described in Section 4.4 are therefore also written in `C++` and integrated into the ns-2 PBS classes. Further explanations on ns-2 and its PBS implementation can be found in Appendix A.

4.2 Mobility Models

There are two possibilities to model the mobility of the nodes in a simulation. The first is that node trajectories are measured in a real network, for instance, node positions can be measured with a GPS device, and then used as input for driving the simulations. While this method is desirable because node movement is modelled realistically, it is often not possible because of the lack of such data. The second possibility is to use a so-called *mobility model*, which basically is a set of rules of how nodes behave. The benefit of this method is, that without having to perform extensive measurements in real world scenarios, large numbers of node trajectories can be used for simulation. However, the drawback of this approach is, that the mobility model reflects the real behavior of mobile nodes only to a certain degree. A good overview of the most common mobility models used for simulation can be found in [31].

For the evaluation of the mobility prediction algorithm two mobility models have been used, which are discussed in the following.

4.2.1 Random Waypoint

The *Random Waypoint Model (RWP)* was used as a representative of a mobility model where the motion of the nodes show little structure. Thus, it is hard to predict the future SNR values. With the RWP model, the

[24] `OTcl` is the object oriented variant of the `Tcl` script language. For more details see [30].

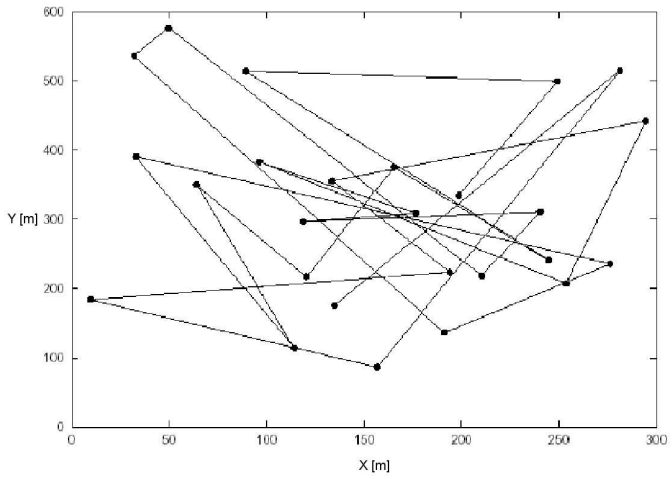

Figure 4.1: Travelling pattern of a node using the RWP mobility model

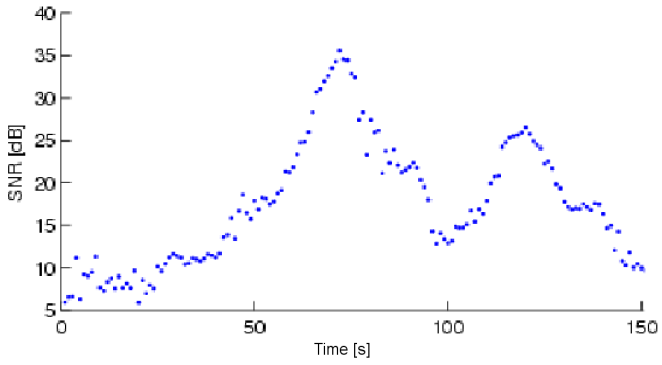

Figure 4.2: Typical SNR pattern of a link driven by the RWP mobility model

nodes start at random, uniformly distributed positions spread over the whole area under simulation. Each node chooses a random destination within the simulation area and a random speed, which is uniformly distributed within the interval of $[minspeed, maxspeed]$. After arriving at its destination, the node pauses for a certain amount of time and then starts all over with selecting a new destination and speed. A typical travelling pattern for the RWP mobility model is shown in Figure 4.1. An exemplary pattern to which such movement leads is plotted in Figure 4.2.

4.2.2 Freeway Model

As a representative of models which show clear movement patterns, the Freeway mobility model described in [32] has been used. Cars are modelled as nodes on a straight line representing a lane on the freeway. The number of lanes and their directions can be configured, as well as the desired minimal and maximal speed for each lane. The speed of the vehicles are set according to the *Intelligent-Driver Model (IDM)* [33]. As a model of lane change and overtaking maneuvers, the Freeway model uses the *MOBIL (Minimizing Overall Breaking Induced by Lane-Changes)* [33] strategy. An exemplary pattern in the SNR measurements is shown in Figure 4.3. It can be clearly seen how the two nodes are moving towards, then crossing and driving away from each other. Patterns with such a short lifetime and such a clear structure are typical for two nodes travelling in opposite directions. For two nodes moving in the same direction the patterns look similar but are longer. The faster car approaches from behind, then overtakes the slower one and the distance increases until the connection is lost.

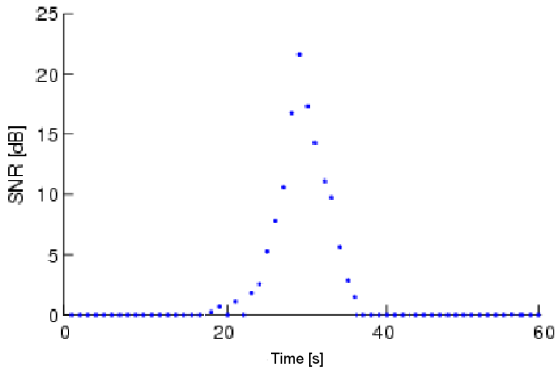

Figure 4.3: Typical SNR pattern of a link driven by the Freeway mobility model

4.3 SNR in Ns-2

As mentioned in Section 3.1.2, a realistic model of the SNR has to take into account the radio propagation (path loss, and effects like reflection, scattering and diffraction) and noise (environmental noise, receiver noise and interference). In the following, the SNR model is explained step by

step with an example of a link between two nodes. The movement of the nodes was generated with the random waypoint mobility model. Note that typical values of the signal power in wireless LAN 802.11 networks range from $-90\ dBm$ to $-40\ dBm$, while a typical noise level observed is around $-90\ dBm$. Thus, with Equation 3.2, typical SNR values range from 0 dB to 50 dB.

4.3.1 Shadowing Model

A model which is commonly used as radio propagation model is *shadowing* [34]. The shadowing model consists of two parts, the path loss model, which defines a deterministic relation between distance and received signal strength, and a random variable, which reflects the variation of the signal strength at a certain distance.

The path loss is usually measured in dB with the equation

$$\frac{P_r(d)}{P_r(d_0)} = -10\beta log(\frac{d}{d_0})[dB], \tag{4.1}$$

where d is the distance between the nodes and d_0 is a reference distance. The parameter β is called *path loss exponent* and is determined by the physical environment. Some typical values of β can be found in [34], for the simulation $\beta = 4$ was chosen, which is a typical value observed in *obstructed in building environments*. The SNR measurements of an example link with taking only the path loss into account is shown in Figure 4.4.[25]

The second part of the shadowing model is an added random variable which is used to model different effects on the received signal strength when the nodes of the link are in motion:

$$\frac{P_r(d)}{P_r(d_0)} = -10\beta log(\frac{d}{d_0}) + X_{dB}[dB] \tag{4.2}$$

X_{dB} has a Gaussian distribution with zero mean and standard deviation σdB. σdB is called *shadowing deviation* and is also determined by the physical environment. Typical values range from 3 to 12 (in [34] the values for different environments can be looked up). For the simulation, a value

[25] In order to show the effect of the radio propagation model, without accounting for noise, a constant noise level of $-90dBm$ has been assumed.

```
$ns node-config -propType Propagation/Shadowing

Propagation/Shadowing set pathlossExp_ 4.0 ;# path loss exponent
Propagation/Shadowing set std_db_ 7.0 ;# shadowing deviation (dB)
Propagation/Shadowing set dist0_ 1.0 ;# reference distance (m)
Propagation/Shadowing set seed_ 0 ;# seed for RNG
```

Listing 4.1: Deploying shadowing propagation model

of $\sigma dB = 7$ was chosen, which is representing an *office environments with hard partition*. The same link as above, this time with the added random variable X_{dB} is shown in Figure 4.5.

How the shadowing model is deployed and configured in ns-2's `Tcl` scrips is shown in Listing 4.1.

4.3.2 Background Noise and Interference

In order to account for *environmental noise* and *receiver noise*, another Gaussian distributed random variable was added to the SNR values. The mean was set to -90 dBm with a standard deviation of 4 dBm.

Ns-2 implements a very simplistic model of *interference* for detecting packet collisions. Collision detection is included in the *MAC (Medium Access Control) Layer* (`mac/mac-802_11.cc`) of the 802.11 implementation. When a packet arrives, the receiving function simply checks, whether another packet is currently being received. If this is the case, the signal powers of the two packets are compared. If the power of the incoming packet is smaller than the power of the packet currently being received by at least the constant `CPThresh` (10 dB by default), a collision is detected and the packet is dropped.

This model has several drawbacks:

- Only two packets are considered. If more packets are being received simultaneously, the interference is not additive.
- The duration of the interference generated by a packet is not considered.

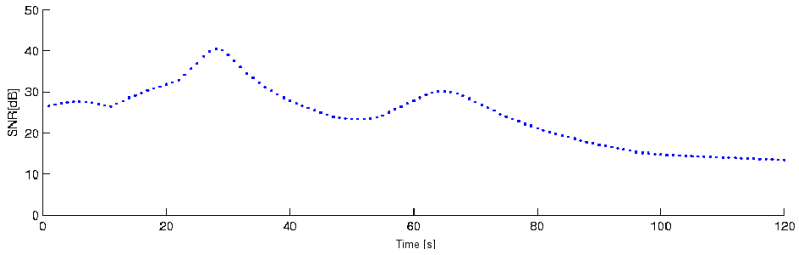

Figure 4.4: SNR with deterministic distance to signal strength relation

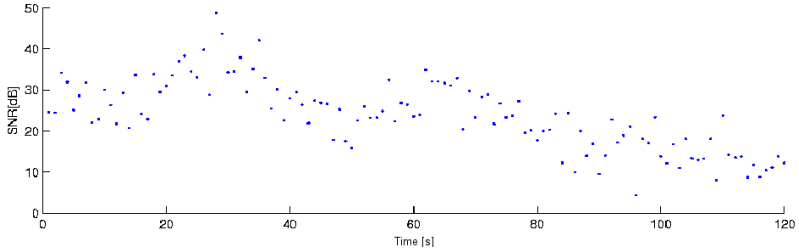

Figure 4.5: SNR with shadowing model

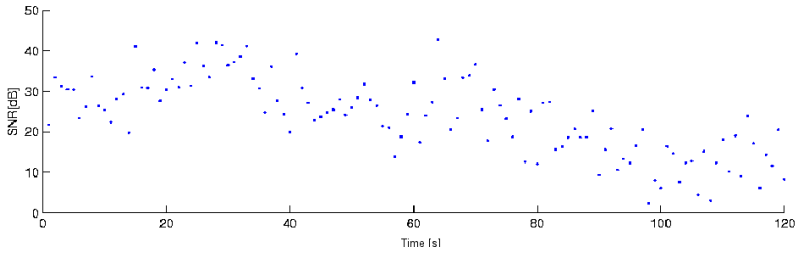

Figure 4.6: SNR with shadowing model plus AWGN plus interference

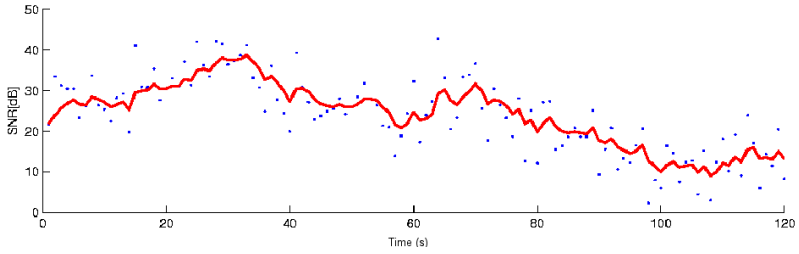

Figure 4.7: SNR measurements filtered with Kalman filter

- Only packets with signal power higher than the *receive threshold* constant `RxThresh_`[26] are taken into account.
- The interference is implemented in the MAC layer instead of the physical layer, to where it belongs.

However, in lack of a better interference model this model was used for the simulations. If during the reception of a packet another packet arrives, the signal power of the latter is added to the noise power as a value for interference.

The SNR is calculated in the MAC layer in ns-2. The MAC layer's `recv` function, which processes incoming packets, gets the received signal strength information from the propagation model in the variable `p->txinfo_.RxPr`. In order to hand over the SNR value to the state observation of the mobility prediction algorithm, the packet header (file `common/packet.h`) was extended with a property `snr_`. The code of SNR calculation is listed in Listing 4.2.

4.4 Implementation of Mobility Prediction in the PBS Algorithm

The PBS algorithm is implemented in the `ZSSpbsAgent` class of ns-2 in `C++` (cf. [5]). An overview of this implementation is given in Appendix A.2. For the extension with mobility prediction two classes were defined: `MobilityStateObservation` for the state observation part and `MobilityPrediction` for the prediction part. The `ZSSpbsAgent` was extended such that each object creates instances of both of these classes.

4.4.1 State Observation

In order to keep track of the links of a node, a structure called `LinkMeasurements` was defined. Each state observation object contains an array of such `LinkMeasurements`, one for each link it currently has or had in the past (recall that the broken links of a node contain valuable training data which should be stored). The

[26] The receive threshold is a constant signal strength value defined in ns-2, above which the packet is received correctly. It is typically set to -95 dBm.

```
// mac/mac-802_11.cc
// Mac802_11::recv(Packet *p, Handler *h)
[..]
// calculate snr and write it in the packet header
double snr;
// initializing ns-2 random number generator
RNG noise;
noise.reset_next_substream();
// calculate snr
if(rx_state_ == MAC_IDLE) {
        // channel idle, no interference
        snr = 10*log10(p->txinfo_.RxPr) - noise.normal(-90,4);
} else {
        // channel occupied, interference
        snr = 10*log10(p->txinfo_.RxPr)
                - 10*log10(pktRx_->txinfo_.RxPr)
                - noise.normal(-90,4);
}
// write to packet header
hdr->snr_ = snr;
```

Listing 4.2: Calculating the SNR in ns-2

`nodeId`	The node ID of the peer of this link
`lastMeasurement`	Stores the last measured SNR value of the actual measurement interval before it is saved in the time series
`measurements[]`	Saves the time series of the filtered values, used as a circular buffer of size `SNR_BUFFER_SIZE`

`kalmanParPkApriori`
`kalmanParPk`
`kalmanParKk`
`kalmanParA`
`kalmanParC`
`kalmanParQ`
	The actual Kalman filter parameters (A, C, P_k^-, P_k, K_k, Q) of the time series as they were described in Section 3.1.3

`prediction`	Stores the last prediction in a vector made for this link
`lastPredictionTime`	The time when the last prediction of this link was made. It is used for checking whether the prediction in the `prediction` field is still actual

Table 4.1: Overview of the `LinkMeasurements` data structure

most important fields of the `LinkMeasurements` structure are summarized in Table 4.1.

The `MobilityStateObservation` class has two important interfaces. The first is used by the `recv` function (the `recv` function handles all the incoming packets) of the `ZSSpbsAgent`. The `recv` function calls the `insertMeasurement` function of the state observation in order to store the measured SNR value for each received packet in the `lastMeasurement` variable of the according `LinkMeasurements`. Because measurements should only be performed once every T seconds (recall that T is the measurement interval defined in Section 3.1.2), a *timer* was added to the `ZSSpbsAgent`. The `makeMeasurement` function, which is called every T seconds, checks if a new value has arrived during the last measurement interval and if so, applies the Kalman filter to this value and stores the filtered value in the time series of the according link. If no new value was saved by the `insertMeasurement` function, it is assumed that the connection is broken and a 0 is inserted in the time series. Note that each node should frequently receive so-called *Hello Messages* for all of its active links. The Hello Messages are used by PBS in order to keep

Filename	Description
`config_trainingorder.inc`	The number of training samples for the autoregressive model of the Kalman filter
`config_queryorder.inc`	The query order
`config_stabletime.inc`	The time for which a link has to be available in order to be considered as stable

Table 4.2: Configuration files for the state observation class

the list of direct neighbors accurate and are sent by default every 0.1 Seconds. Thus, in order to make sure that a measurement for each link can be made during a measurement interval, T should not be set to values smaller than 0.1.

The second important interface of the `MobilityStateObservation` class is used by the `ZSSpbsAgent` to request a prediction for a certain link. In order to check, whether a link to a given neighbor is stable, the `isLinkStable` function is called, which returns a boolean value. The `isLinkStable` function first checks, whether the prediction in the `prediction` field of the according `LinkMeasurements` structure is still actual. The life time of a prediction was set to be 2 seconds. This value should make sure that during one election each link has to be predicted only once, as an election usually should take less than 2 seconds time. When the prediction has expired, a new one is triggered. Then the `isLinkStable` function scans the prediction for zeros in order to decide whether the link is stable or not.

Additionally to these two interfaces for the communication with the `ZSSpbsAgent`, the `MobilityStateObservation` provides some functions for the `MobilityPrediction` class. These are mainly used to hand over the training data and query to the prediction part.

In order to set the parameters of the state observation, three config files are used. They are listed in Table 4.2 and can be found in the `zoneserver/tcl/wireless/` directory. Each of the files simply contains a value of the parameter.

4.4.2 Prediction

The `MobilityPrediction` class has one important public function called `predict`, which is used by the `MobilityStateObservation` in order to get a prediction of a certain link. The `predict` function first fetches the query and training data and tries to find predictors by calculating the normalized cross-correlation function. If this is successful, the most 'common' predictor is chosen according to the strategy explained in Section 3.2.2. If no predictor was found, it gets the autoregressive model parameters from the according `LinkMeasurements` structure and performs an iterative prediction with them. Finally, the `predict` function stores the predicted values in the `prediction` field of the `LinkMeasurements`.

The `MobilityPrediction` has only one parameter which can be configured via a config file. The match threshold is read from the file `config_threshold.inc` in the `zoneserver/tcl/wireless/` directory.

4.4.3 Link Stability Criterion

In order to implement the link stability criterion in PBS, several changes in the `ZSSpbsAgent` code and in the `Neighborlist` class had to be made. The `Neighborlist` class was extended with two new functions. `getAllStableNeighbors` and `getStable1HopNeighbors` are the pendants to the default `Neighborlist` functions `getAllNeighbors` and `get1HopNeighbors` which return only the neighbors for which the stability criterion holds. Additionally, a configuration parameter was introduced to the `Neighborlist` class, which controls whether prediction should be used or not. Therefore, the file `config_prediction.inc` is read which can contain a zero for disabling prediction or a one for enabling it.

The `ZSSpbsAgent` class was changed in two places. First, the function that sends the neighborlist was changed to include only those neighbors for which the stability criterion holds. In order to do this, the `sendNeighborlist` uses the `getStable1HopNeighbors` function. The second change concerns the `determineStatus` function, which was changed in several places to use the `getAllStableNeighbors` and `getStable1HopNeighbors` functions in order to determine the state of the node as described in Section 3.3.

4.5 Chapter Summary

This chapter provided an overview of the ns-2 implementation of the prediction algorithm. The concept of mobility models was discussed and two representatives, the RWP model and the Freeway model were explained. Furthermore, a detailed explanation of the SNR in ns-2 was given. The implementation of the state observation part and the prediction part of the algorithm was shown together with the changes in the adaptations required in the PBS implementation. The following chapter will now present the evaluation of the algorithm based on this implementation.

5

Evaluation

In this chapter, the mobility prediction algorithm is evaluated by simulation in ns-2. It starts with determining the optimal choice of some design parameters which have been described in the previous sections. Section 5.1 shows how the *number of training samples* for the autoregressive model in the Kalman filter was set. In Section 5.2, the question of the *query order* and *match threshold* are addressed, both have influence on the number of predictors and on the accuracy of the predicted SNR values.

After having set the parameters, the accuracy of the prediction is evaluated in Section 5.3 using the Random Waypoint and the Freeway mobility models. Finally, in Section 5.4 the influence of the link stability criterion on the server selection algorithm is analyzed by comparing the Dominating Set stability with and without prediction.

5.1 Kalman Filter Parameters

In Section 3.1.3, the *autoregressive link model* for the *Kalman filter* was described. In order to create the model of the link at each filtering step, that is at each time a new measurement is made, the past measurements of the link have to be used as training data for setting the model parameters α and c. The question of how far in the past values should be used as training data (the *training data order*) was left unanswered in Section 3.1.3. Intuitively, it seems clear that using only a small number of training measurements for creating the model should give better results than choosing a large training

data order, as this creates a more accurate model of the *actual* state of the link (*local model*). Recall that, as the Kalman filter needs only 1-step-ahead predictions, this situation differs a lot from the problem of predicting network topology changes, which requires a *long term* prediction. Thus, for the Kalman filter a model taking into account only the recent past of the link should be created, opposed to the model for the SNR prediction, where having more training data is better.

In order to determine the optimal choice of training data order, simulations have been run with different model orders and the quality of the models was compared. As a measure of quality of autoregressive models the *coefficient of determination* R^2 is widely used (cf. [21]). It is defined as

$$R^2 = 1 - \frac{\sum_i \hat{e_i}^2}{\sum_i x_i^2} = 1 - \frac{\sum_i (\hat{x_i} - x_i)^2}{\sum_i x_i^2}, \tag{5.1}$$

where the estimated (by the model) value at time i is denoted by $\hat{x_i}$ and the measured value by x_i. The values of R^2 lie in the interval $[0, 1]$, 0 means a bad fit and 1 means a perfect fit of the model. The procedure to compute the R^2 value is the following:

1. Take the last o measurements and create the autoregressive model by calculating the model parameters α and c, according to Equations 3.17 and 3.18.

2. For each of the o measurements, calculate the value that would have been estimated with the above calculated model based on the previous measurement according to the following equation: $\hat{x}_k = \alpha x_{k-1} + c$. Compute the squared error of this estimated value: $\hat{e_i}^2 = (\hat{x_i} - x_i)^2$.

3. With having all the $\hat{e_i}^2$ values, compute R^2 with Equation 5.1.

5.1.1 Simulation Setup

For the evaluation of the *coefficient of determination*, the *Random Waypoint Mobility Model* (see Section 4.2.1) was used on a small scenario with 10 nodes. The simulation setup is summarized in Table 5.1. The simulated area of 1000×1000 m^2 was chosen in order to have a not fully connected network. The relatively short simulation time of 300 seconds is enough, as

Mobility Model	Random Waypoint
Max. speed	5 m/s
Pause time	5 s
Simulated area	$1000x1000\ m^2$
Number of nodes	10
Simulation time	300 s

Table 5.1: Simulation setup for determining the training data order of the autoregressive model

for each link in the network each second a model is created, which gives enough data to make some statistical evaluation. Assuming that each node has an average of 5 links to other nodes[27], each node computes $5 \times 300 = 1500$ models during the simulation time. With 10 nodes in the network this gives a set of 15000 computed R^2 values. For each model order in the range from 3 to 13[28], the simulation ran 10 times with different random seeds of the mobility model.

5.1.2 Results

The resulting *average* R^2 *values* depending on the training data order are plotted in Figure 5.1. In the plot, the intuitively assumed decrease of the coefficient of determination is approved. The best fit of the model with the training data with $R^2 = 0.91$ is achieved at training data order 3, then it goes steadily down to a value of $R^2 = 0.59$ at model order 13.

These results suggest to chose a model order of 3. However, it was observed that this small amount of training data occasionally leads to unstable predictions. If the 3 training values are in an unfavorable constellation because of the noise, the model fit might be good, but the 1-step-ahead prediction is an unrealistic value. Thus, in order to get more stable predictions, a training data order higher than 3 had to be chosen. A choice of training data order of 7 is a reasonable value, as the coefficient of determination is

[27]The number of 5 links was only guessed and not verified, as it is only used to make a rough estimation of how many models are created.

[28]Note that model order 1 and 2 were omitted because with one single value no model can be created and having only two training values leads in any case to a R^2 value of 1, as the model would simply be a straight line through the two points without any error.

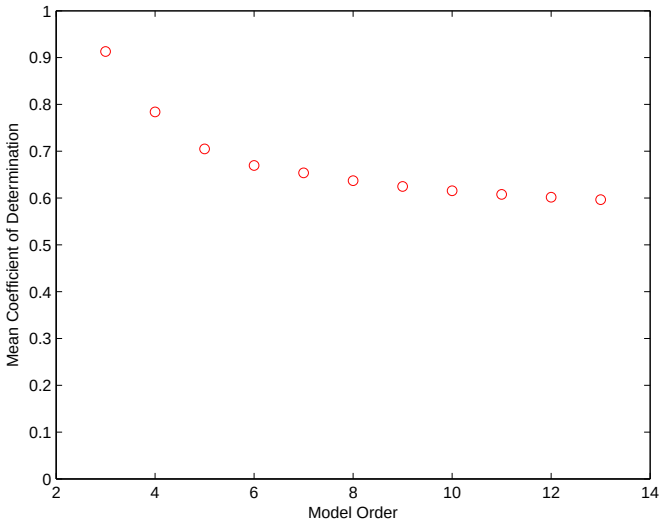

Figure 5.1: Average values of the coefficient of determination of autoregressive model for different training data orders

with $R^2 = 0.65$ still high enough and the probability of unstable predictions is already much smaller.

With this, the discussion of the Kalman filter and the whole state observation part is complete. The next step is to go to the prediction part and determine its remaining parameters.

5.2 Prediction Parameters

For the prediction part, the parameters which remain to be discussed are the *query order* and the *match threshold*. They both have a direct influence on the accuracy of the prediction and should be set in a way that gives as accurate as possible predictions. Another question still open is the *order of prediction k*. While having already mentioned that a long term prediction is required, it was not yet discussed how far in the future this prediction should go. However, this discussion is depending on the application of the mobility prediction algorithm and is therefore postponed to Section 5.4, where the *link stability criterion* is evaluated. In the following, the influence of the query order and of the match threshold on the prediction accuracy are first discussed separately. Then, the simulation setup and results are given.

Query Order

The question of the *query order* (see Definition 3.3) is related to how long a pattern in the movement of the nodes is assumed to be. However, as there are no clearly splittable patterns with a unique length in the training data, the optimal query order cannot be set analytically but has to be chosen by means of simulation instead. Furthermore, different physical environments of the network may lead to different lengths of the observed patterns, thus the query order should be set as some compromise between environments that show short patterns and others which show longer patterns.

Anticipating the results of the simulation, two main effects of the query order on the prediction accuracy can be assumed:

- A short query leads to a large number of predictors. This is a benefit, as the decision of which predictor should be used as prediction (see Section 3.2.2) can be based on many predictors. However, if the query order is too small, the predictors are bad representations of the current node behavior. This may lead to a reduced accuracy of the prediction.
- A large query order leads to a small number of higher quality predictors, with the risk that the number of predictors gets too small or even none is found at all. This should be avoided, as in this case the fallback solution (see Section 3.2.3) has to be applied.

Thus, by means of simulation a balance between these two controversial effects should be found.

Match Threshold

The second parameter which requires some discussion is the *match threshold* (see Definition 3.4). The match threshold is the value above which the correlation of the query and the training data at a certain lag m is considered to be a match. Intuitively, the match threshold, just as the query order, influences the number of matches found and therefore the number of predictors and the accuracy of the prediction. The influence of the match threshold on the number of predictors and the prediction accuracy is quite similar to the influence of the query order:

- A small match threshold leads to a big number of predictors, as the match must not be perfect. However, a too small threshold can be harmful, as patterns are considered as matches, which are not really similar to the query.
- On the other hand, choosing a high match threshold leads to a small number of predictors, as only few situations are considered similar enough to take into account at the choice of the prediction. This again is risky, as the number of predictors may be too small or no predictor may be found at all.

Thus, for this parameter also a balance between the two effects has to be found.

As the final goal is to optimize the accuracy of the prediction which is influenced by both, the query order the and match threshold, the goal of the simulation was to find an *optimal combination* of them. In order to account for this, the simulation was conducted with possible combinations of query order and match threshold.

5.2.1 Simulation Setup

In order to account for different physical environments in which a MANET can possibly be deployed, two mobility models, the RWP and the Freeway model, with vastly different behaviors of the nodes were chosen for the simulation. The RWP model was chosen to have a simulation where the patterns are not so clear and the behavior of the nodes is highly random. In the Freeway model, on the other hand, the observed patterns are quite limited as the nodes have basically only the freedom to move in either of the directions with varying speeds.

Random Waypoint Scenario

The simulation parameters for the Random Waypoint scenario are summarized in Table 5.2. The parameters are basically the same as in the simulation described in Section 5.1, except for the simulation time. The first 600 seconds of the overall 630 seconds simulation time, were used for collecting training data. At time 600, all the nodes predicted the future SNR values for

Mobility Model	Random Waypoint
Max. speed	5 m/s
Pause time	5 s
Simulated area	$1000x1000\ m^2$
Number of nodes	10
Simulation time	630 s

Table 5.2: RWP simulation setup for determining the query order and the match threshold

each of their links. In order to get an overview of how much data the local link model was based on, the number of predictors for each of the predictions was saved. The prediction order k was set to 30 seconds, thus a 30-steps-ahead prediction was performed. The following 30 seconds, the accuracy of this prediction was measured by comparing at each measurement interval the predicted value with the measured one[29]. In order to get an estimation of the accuracy of the prediction, the average absolute prediction error of each of the links in the network for each of the predicted time steps was computed. 10 simulation runs with this scenario have been performed with different random seeds in order to get a broad data base for the evaluation. The results and interpretation of this simulation will be presented in the next section, after the discussion of the Freeway scenario.

Freeway Scenario

The simulation setup of the Freeway scenario is shown in Table 5.3. This scenario is supposed to model a typical Swiss highway with 2 lanes in either direction. The speeds are set according to typical behavior on such a highway with a speed limit of 120 km/h. The node density is, with 25 nodes in 5 km, rather small. It was chosen at such a low level in order to account for the fact that it is not realistic to assume each car on a freeway equipped with ad hoc networking capabilities. Thus, only a certain percentage of the cars participate in the network. The simulation time is, with 330 seconds, shorter than in the RWP scenario. The reason for this is that the variety of typical patterns observed in such a physical environment is by far smaller than in

[29]Note that for the calculation of the prediction error, not the noisy measurement was taken, but the Kalman filtered value instead.

Mobility Model	Freeway
Lanes	4 (2+2)
Speed	Fast lane: 110 km/h ... 130 km/h
	Slow lane: 80 km/h ... 110 km/h
Simulated freeway length	5000 m
Number of nodes	25
Simulation time	330 s

Table 5.3: Freeway simulation setup for determining the query order and the match threshold

a Random Waypoint environment and the patterns are usually shorter. Of these 330 seconds, again 30 seconds are used to verify the accuracy of the prediction made after 300 seconds. The first 300 seconds are used to collect training data. With this scenario also 10 simulation runs with different random seeds have been performed.

5.2.2 Results

As mentioned above, the simulation of the above described scenarios have been run with different combinations of query order and match threshold in order to get some insight of how these two parameters affect the prediction accuracy. The match threshold has been varied in the interval [0.5, 0.9] with steps of 0.05, which gives 9 different values. The query order has been chosen in the interval of [20, 100] in steps of 20, thus 5 different values were simulated. All possible combinations of these values give a total number of $9 \times 5 = 45$ configurations, each being simulated 10 times. That gives a total number of 450 simulations per scenario.

Figures 5.2 and 5.3 show the average number of predictors per prediction depending on the query order and the match threshold. The figures confirm what has already been assumed, that small query order and small match threshold lead to high numbers of predictors. The absolute numbers shown in these figures are not really interesting, as they depend highly on the length of training data, the number of links the nodes have and other parameters. However, what can be taken from these figures is some upper bounds of the query order and match threshold. In case of a match threshold in the range of 0.8 ... 0.9, the number of predictors is, especially in the RWP scenario,

approaching zero. Thus, in order to avoid this the match threshold should be set to a smaller value. The same statement can be made for the query order for values around 80 ... 100, where the number of predictors get very small.

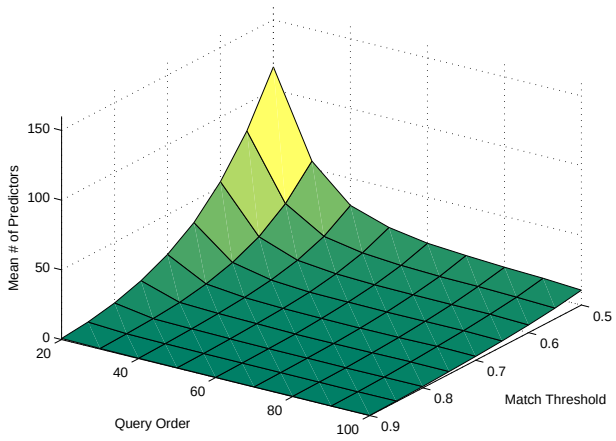

Figure 5.2: Average number of predictors with different query orders and match thresholds using the RWP mobility model

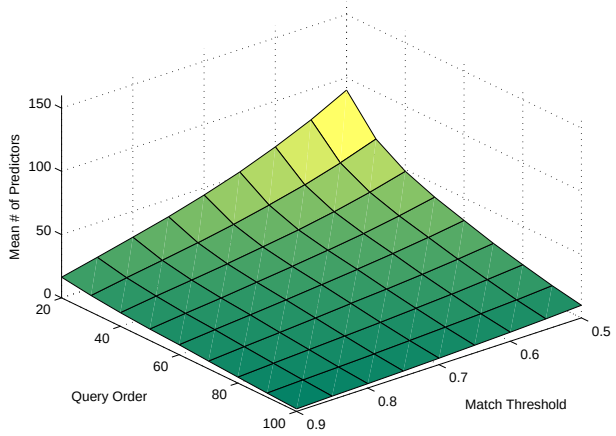

Figure 5.3: Average number of predictors with different query orders and match thresholds using the Freeway mobility model

The more important information comes from the average absolute prediction errors, shown in Figure 5.4 for the RWP scenario and in Figure 5.5

for the Freeway model. The prediction error in dB, depending on the query order and the match threshold was plotted for different prediction orders, like 1, 5, 10, 20 and 30 steps-ahead predictions, respectively. First looking at the RWP model, the 1-step-ahead prediction error does not really depend on the two parameters and is about constant at 2 dB. For longer term predictions, starting at around 10 steps, the error starts to significantly increase for higher match thresholds. The reason for this is one of the above mentioned effects, namely that for high thresholds no predictors can be found and the fallback model is used. These results suggest to choose a small match threshold of about 0.5.

Looking at the prediction errors for the Freeway model plotted in Figure 5.5 shows a different situation. First, it is obvious that all in all the errors for this model are smaller than in the RWP case. This was already assumed before, because of the clearer structure of the patterns using the Freeway model. For short term predictions, an average error of about 1 dB can be observed, which is again more or less independent of the query order and match threshold. For longer term predictions, an important difference to the RWP case can be seen. Where, using the RWP model, the error depended mainly on the match threshold and not so much on the query order, using the Freeway model the opposite is the case. The error is more or less independent of the match threshold but increases significantly with a smaller query order. The reason for independence of the match threshold again lies in the clear patterns observed with this model. If a situation in the past is really similar to the current situation, the patterns will match even with a high match threshold. Thus, the matches with small match thresholds are the same as those with high match thresholds. This can be verified in Figure 5.3, where the number of predictors is not as strongly dependent on the match threshold as in the RWP case in Figure 5.2. The other effect, the errors increase with too short query orders, is a sign that a query order of below 60 is too small for the Freeway case. This is an indication that with this model usual patterns are about 60 seconds long[30].

As a compromise of the observed effects using the RWP and the Freeway model, a query order of 70 and a match threshold of 0.5 were chosen for the

[30]Recall that in Figure 4.3, a typical pattern lasting only 20 seconds was shown. However, this was a pattern of nodes moving in opposite directions. When the nodes move in the same direction, the patterns are longer.

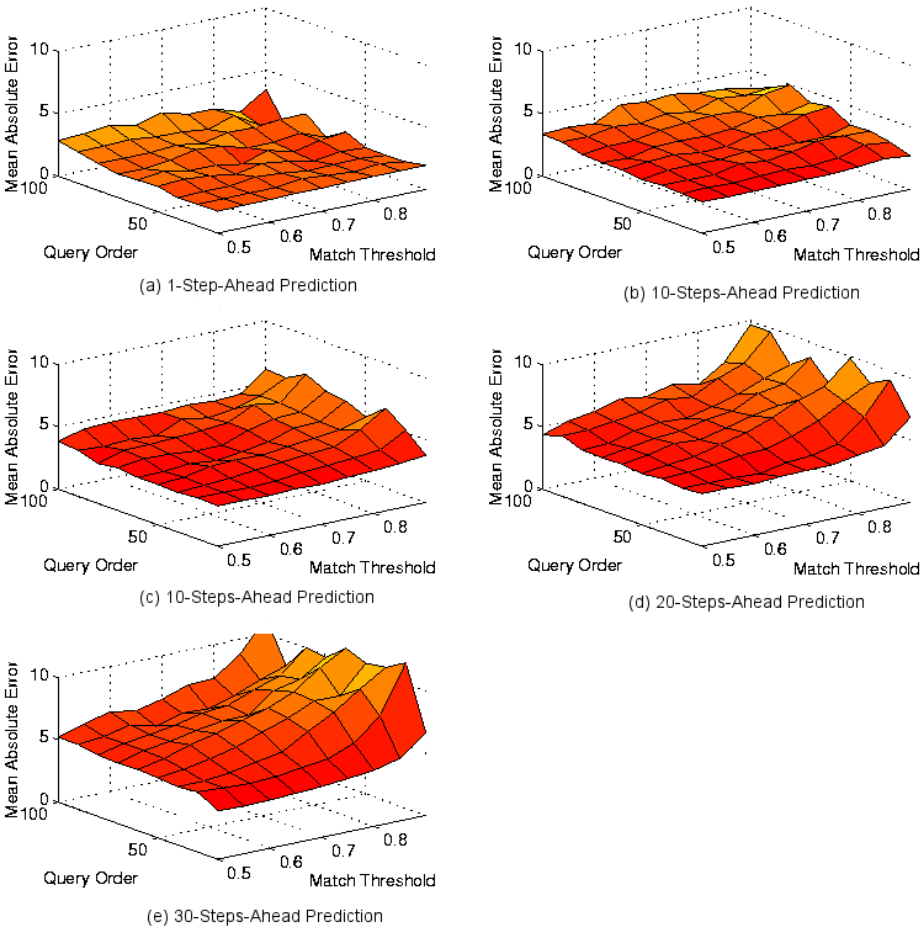

Figure 5.4: Mean prediction error with different query orders and match thresholds using the RWP mobility model

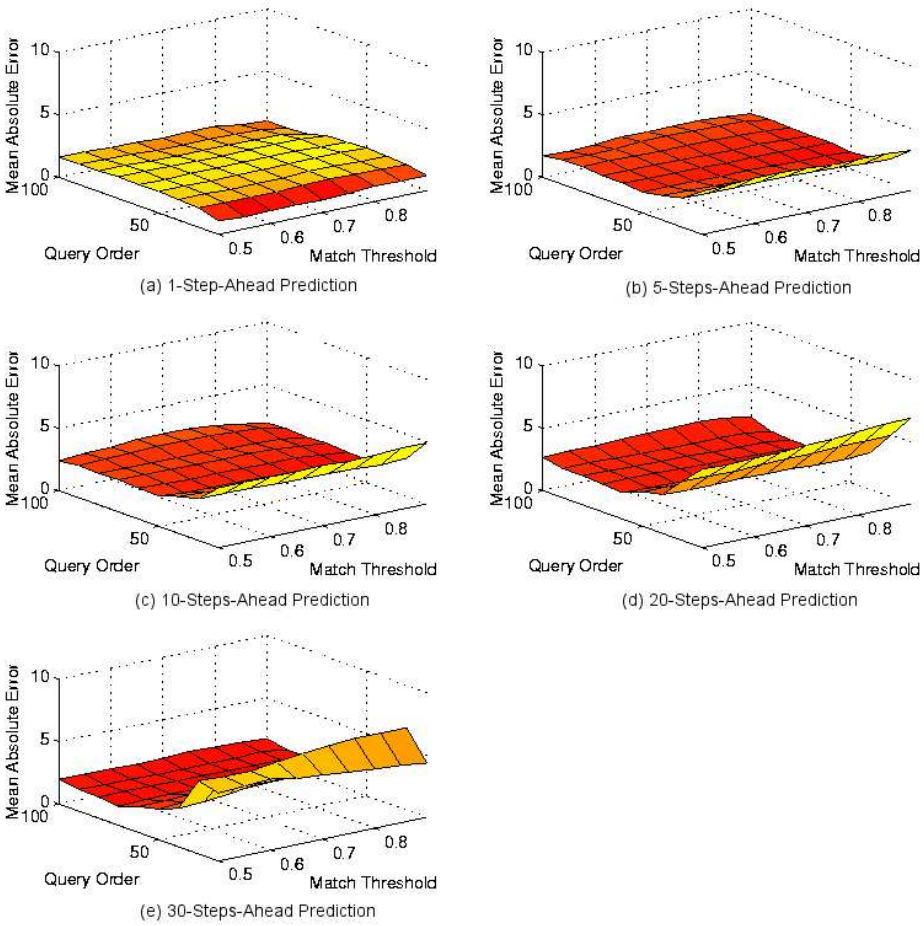

Figure 5.5: Mean prediction error with different query orders and match thresholds using the Freeway mobility model

further evaluation of the prediction algorithm. These are values which lead in both scenarios to good results. Note that if there is some knowledge about the physical environment of the network and the mobility structures of the nodes available, these two parameters may be tuned accordingly in order to get the best possible results.

Having set the remaining parameters, the algorithm is now ready for a more detailed evaluation of its accuracy, which is presented in the following section.

5.3 Prediction Accuracy

Figures 5.6 and 5.7 show in more detail the dependence of the average absolute prediction error on the prediction order for the RWP and the Freeway model with query order 70 and match threshold 0.5. For the RWP case, the results are straight forward. For a one-step-ahead prediction, the error lies at 2 dB, it then steadily increases with the prediction order up to a value of around 5 dB for a 30-steps-ahead prediction. In case of the Freeway model, the results are a bit more surprising at first sight. The error first increases up to a maximal value of around 3 dB for a 12-steps-ahead prediction. Then the error starts to decrease again, until it reaches a value of about 2 dB for a 30-steps-ahead prediction. This decrease of error stems from the rather short lifetime of the links, especially between nodes driving in opposite directions (cf. Figure 4.3). As more and more links break, the predictions in average get more accurate because the absence of a link can usually be predicted without any error. Predicting the exact SNR value of an existing link will always contain some error, while predicting that there is no link anymore is perfectly accurate if the link really breaks.

5.4 Dominating Set Stability

One question that remains is the influence of the link stability criterion on the Dominating Set stability in the PBS algorithm. In order to clarify this, another round of simulations was run with the parameters set to the values discussed above. The simulation was again performed using both, the RWP and the Freeway scenarios, with simulation parameters similar to

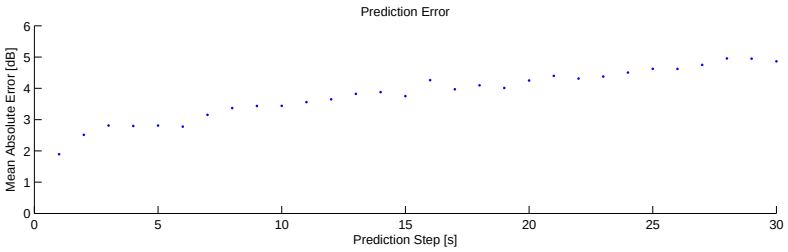

Figure 5.6: Mean prediction error for different prediction times using the RWP mobility model, query order 70 and match threshold 0.5

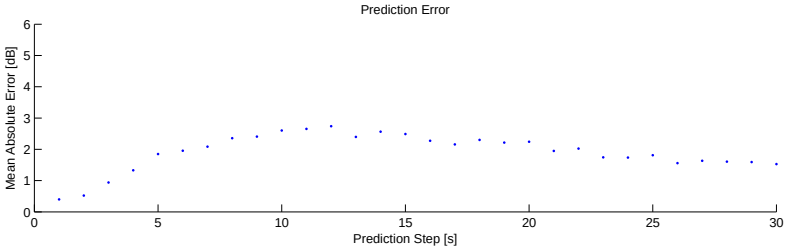

Figure 5.7: Mean prediction error for different prediction times using the Freeway mobility model, query order 70 and match threshold 0.5

those given in Tables 5.2 and 5.3. The only difference in the simulation setup was concerning the simulation time. Instead of the 630 seconds in the RWP scenario, the simulation this time ran for 600 seconds, with 300 seconds training time. That means, after 300 seconds the Dominating Set was constructed and then maintained for the next 300 seconds. In case of the Freeway scenario, instead of 330 seconds 200 seconds training and 100 seconds for the construction and maintenance of the DS was used.

The results of these simulations are given in Table 5.4 for the RWP case and in Table 5.5 for the Freeway model. The tables show the average number of Dominators and the average number of Dominating Set changes[31], depending on the link stability criterion. For instance, a link stability of $k = 30$ implies that a link is assumed to be stable, if it is still available in 30 seconds (cf. Section 3.3). The first row of the tables shows the numbers for the DS without using prediction ($k = 0$). In the following rows, the stability criterion got more and more strict ($k = 10 \ldots 60$). Additionally to the number of Dominators and DS changes, the standard deviation and the

[31] A Dominating Set change occurs when a node in CANDIDATE or DOMINATEE state switches to DOMINATOR state or vice versa.

percentage values are given with the values without prediction set to 100 %.

The results for the RWP model show that the number of Dominators increases with using the link stability criterion by more than 30 % from a value of 3.45 up to around 4.5. This increased number of Dominators is the expense which is paid for the increased DS stability, as Candidates do not just accept any Dominator in their neighborhood but require one with a stable link instead. In terms of DS changes, the average number of changes could be reduced with the stability criterion from a value of 20.4 down to 16.6. This is a reduction by 19 % in the optimal case of requiring 40 seconds (or 30 seconds which gives the same reduction) of stability in the links.

In general, one can argue that decreasing the number of DS changes is usually worth the price of having a few more Dominators, because changes in the Dominating Set are expensive. A change in the DS generally means a *service disruption* for at least the nodes that lose the connection to their Dominators and triggers a re-election. Additionally, a re-election presents a large communication overhead, which should be avoided whenever possible. However, while this argument is true for most usual types of services, in case of a service which requires a big synchronization overhead between the Dominators it might be desirable to have fewer Dominators and more DS changes instead. In such a case, the link stability criterion should not be used.

In the Freeway scenario, the results in Table 5.5 look similar but better. In general, the number of Dominators and the number of DS changes are higher than with the RWP model, because the mobility of the nodes is higher. A nice difference to the RWP scenario is, that the costs of a more stable DS are much smaller and the increase of stability is higher. In the optimal case of $k = 40$, the number of DS changes could be reduced by 26 % from 72.6 changes to 53.3 changes, while the number of Dominators is only increased by 11 % from an average of 11.8 to 13.04. Figure 5.8 shows the variation over time of the number of Dominators in an exemplary case with and without using prediction. It shows that a lot of the fluctuations in the number of Dominators can be avoided.

In both scenarios, an optimum of 40 seconds link stability was found, thus the remaining parameter, the *prediction order k* can be set to this value.

k	Avg. # of Dominators	σ	%	Avg. # of DS changes	σ	%
0	3.45	0.25	100	20.4	6.50	100
10	4.62	0.52	134	20.0	6.29	98
20	4.64	0.46	134	19.0	4.06	93
30	4.63	1.13	134	16.6	4.59	81
40	**4.68**	**0.45**	**135**	**16.6**	**3.62**	**81**
50	4.54	0.68	132	18.2	3.54	89
60	4.49	0.89	130	17.4	5.81	85

Table 5.4: Average number of Dominators and changes in the Dominating Set depending on the prediction order using the Random Waypoint model

k	Avg. # of Dominators	σ	%	Avg. # of DS changes	σ	%
0	11.80	0.43	100	72.6	9.60	100
10	12.51	0.56	106	63.4	5.68	87
20	12.78	0.35	108	58.4	12.17	80
30	12.58	0.33	107	54.2	7.98	75
40	**13.04**	**0.68**	**111**	**53.4**	**10.05**	**74**
50	12.82	0.65	109	55.0	8.90	76
60	12.94	0.50	110	56.6	8.04	78

Table 5.5: Average number of Dominators and changes in the Dominating Set depending on the prediction order using the Freeway model

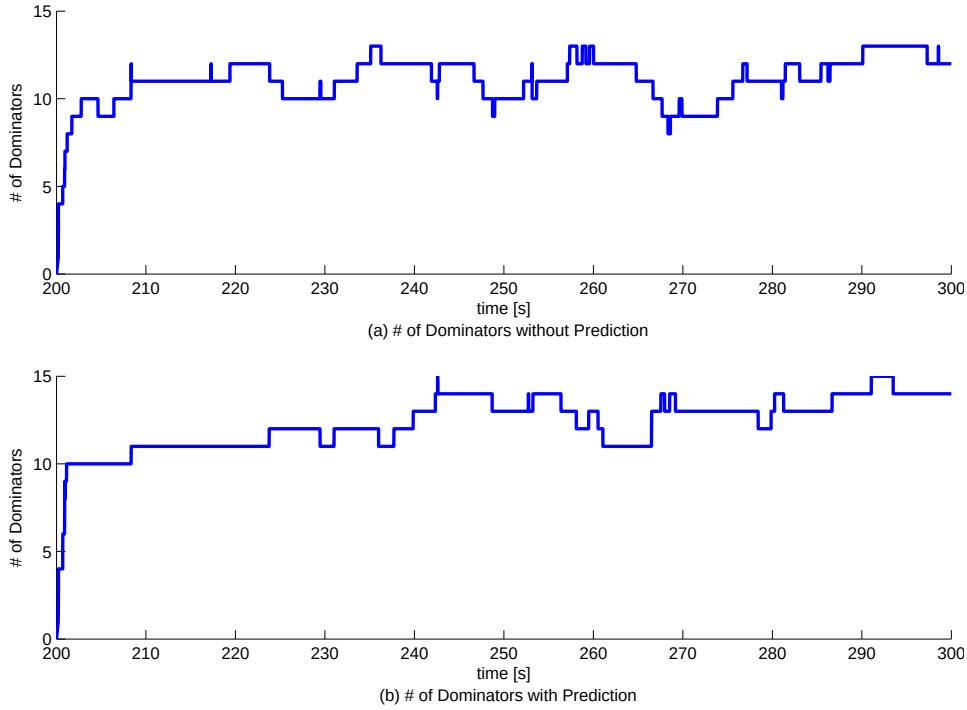

Figure 5.8: Exemplary number of Dominators in the DS with and without prediction using the Freeway model

5.5 Chapter Summary

In this chapter, the prediction algorithm was evaluated. The parameters, such as the training data order of the autoregressive model, the query order and the match threshold and their influence on the prediction accuracy were analyzed. This helped to set these parameters in an optimal way. Furthermore, having the parameter values set, the average prediction errors were evaluated using the RWP mobility model and the Freeway model. Finally, the evaluation of the influence of the link stability criterion on the number

of DS changes was shown with the result that the stability of the computed Dominating Set improved significantly.

6

Conclusions and Outlook

This chapter concludes this thesis by giving a short overview of the work and the achieved results and presents some ideas for future work and following projects.

6.1 Conclusions

In this thesis, a prediction algorithm based on *pattern matching* was developed. To the best knowledge of the author, such an approach to mobility prediction is new in the area of MANETs, as most of the existing methods use linear models and are based on having localization information from dedicated hardware, such as GPS devices. In order to observe the mobility state of a node and avoiding the use of dedicated hardware, the *Signal to Noise Ratio* of the links is monitored and filtered with a *Kalman filter*. The pattern matching approach was justified with the assumptions that (1) the movements of the nodes are restricted by the physical environment of the network and the intentions of the users and (2) the behavior of the nodes is repetitive. In order to recognize situations similar to the current in the past, the *normalized cross-correlation function* of the current pattern with the history of the links was used to obtain a set of predictors. As it is desirable to use the most probable one of these predictors as a base of the prediction, a method of choosing the most common predictor among the set of predictors by correlating them with each other was chosen. For cases where no match can be found in the past, a fallback solution based on an *autoregressive model* was defined.

In order to verify the algorithm, it was implemented in the *network simulator ns-2*. With this implementation it was possible to find optimal choices of some design parameter of the algorithm, such as the *query order* and the *match threshold*. It was shown that the obtained predictions in case of the used two mobility models, the *Random Waypoint model* and the *Freeway model*, are reasonable. However, the accuracy of the predictions depends on how much structure the mobility of the nodes shows. For the RWP model as a representative of having little structure in the mobility, the accuracy ranges from 2 dB of absolute average prediction error for a 1-second-ahead prediction to 5 dB for a 30-seconds-ahead prediction. In case of the Freeway model, which shows clear patterns, a maximal average error of around 3 dB was found, independent on how far in the future the prediction reaches.

Furthermore, as an application of the prediction algorithm, a *link stability criterion* was introduced and implemented in the PBS algorithm for distributedly computing and maintaining a Dominating Set of zone servers. With the chosen approach clients (Dominatees) accept only neighbors as servers (Dominators) when having a link to them which is predicted to be stable for a certain time in the future. With simulations it could be shown that this extension leads up to a reduction of Dominating Set changes by 19 % in case of the RWP model and 26 % in case of the Freeway model. Thus, the prediction has proved to be useful for the application of distributed server selection. However, it is not limited to server selection and could also be used for instance in the routing layer.

As mentioned in the first chapters of this thesis, mobile ad hoc networks are a challenging environment for mobility prediction if the assumption of being able to localize the nodes by means of localization hardware is dropped. As there are no fixed points in the network with known position, the only thing that remains is focus on relative distances between the nodes. By using the SNR as a measure of distance, not in geographical space but instead in 'signal space', the discussed algorithm is able to cope with this challenge and predict changes in the network topology. With the approach of pattern matching this is done not in a simple linear way, instead the nodes learn from the past behavior of their links and therefore adopt the predictions to the specific properties of a given network. This is especially a benefit in networks showing a clear structure in mobility as the simulations with

the Freeway mobility model have shown. Of course, an approach based on learning requires some fine tuning of the parameters. In this thesis the parameters' values were carefully set in order to provide good results in a wide range of possible mobility patterns. However, if the physical environment of a network is given, the parameters could still be adjusted accordingly.

One thing to note is that the described algorithm focuses on predicting the failure of existing links. While this is enough for the presented application, the link stability criterion, it might be useful for other applications to predict the whole future topology of the network. Thus, while in this thesis the approach was to use mobility prediction in terms of predicting what influence the mobility of a node has on the future state of its links, predicting the future topology of a network would mean to use mobility prediction in terms of predicting the future position of a node in a network. In order to do so the existing approach could be extended by an algorithm that concludes from the distances between nodes[32] on the network topology.

One drawback of the presented approach is that it is costly. Though a detailed complexity analysis of the algorithm was not performed in this thesis, one may imagine that computing the normalized cross-correlation of the queries and the training data as well as correlating the predictors with one another presents a big computation overhead for the nodes. Especially for devices with very limited resources such as mobile phones this might be to much and a simpler or more optimized solution might be preferable. Analyzing the complexity and optimizing the algorithm are left as future work.

6.2 Outlook

There are several interesting possibilities of following projects and further research based on the presented work. For example:

Computational complexity: As mentioned above, it is a known issue of the presented algorithm that the pattern matching method used is computationally expensive. Computing the normalized cross-correlation

[32]Here again, distance is not used in geographical sense but rather as 'signal space' measure.

functions and the correlations for selecting the most common predictor is costly and a more efficient method might be found. One viable approach might be to use the *fast normalized cross-correlation* in the frequency domain as it is described in [27].

Small Dominating Set vs. small number of changes: As mentioned in Section 5.4, with the current approach of the *link stability criterion*, the reduction of Dominating Set changes comes with the cost of having an increased number of Dominators. While this should not be problematic for most types of services, in some cases the increased traffic for synchronization might be undesirable. A study of the overhead of a Dominating Set change and the overhead of synchronization traffic generated by the increased number of Dominators should be done.

Future network topology: As noted before, the presented algorithm focuses on predicting *changes* in the network topology in terms of failing links. However, for some applications it might be desirable to predict the whole network topology. In order to achieve this, the current approach should be extended by an algorithm which is able to deduce from the future SNR values on the network topology. This might enable the algorithm to not only predict the failure of existing links but also predict *upcoming* links. One possible approach to do so is to use *Multidimensional Scaling (MDS)* as it is described in [35] for sensor networks.

Testbed implementation: In order to gain some experience with real world scenarios, the prediction algorithm should be implemented in the *SIRAMON testbed*. However, such an implementation does not make sense with the current testbed, as it is too small[33] to see the impact of prediction on the server selection. Thus, a larger testbed is required and measurements in real world scenarios should be performed.

[33]The testbed consists currently of 3 laptops and 2 PDAs, thus having only 5 nodes (see Appendix B.2).

Appendix A

Ns-2

This Appendix gives a short overview of the network simulator ns-2. Furthermore, it provides some basic information about the ns-2 implementation of the PBS algorithm and gives some instructions on how to install ns-2 with the PBS agent and the prediction algorithm.

A.1 About Ns-2

Ns-2 is a free and open source network simulator, available for download at the ns-2 homepage [7]. A good introductory tutorial can be found at [28]. For the implementation of the prediction algorithm as it was described in Section 4, the ns-allinone package release 2.29 was used.

The simulator core is written in `C++`. An object oriented variant of the script language `Tcl`, called `OTcl` (see [30] for more details) is used for the configuration of the simulator. The combination of these two languages offers a compromise between performance and ease of use of the simulator. A simulation usually creates trace files in order to analyze the results. For viewing these traces, an animation tool called Network Animator (NAM) can be used.

The PBS algorithm was implemented in ns-2 as an *agent* (`ZSSpbsAgent`) that can be attached to the nodes of the network in the simulation configuration scripts. This agent will be described in the following section. For a more detailed explanation, see [5].

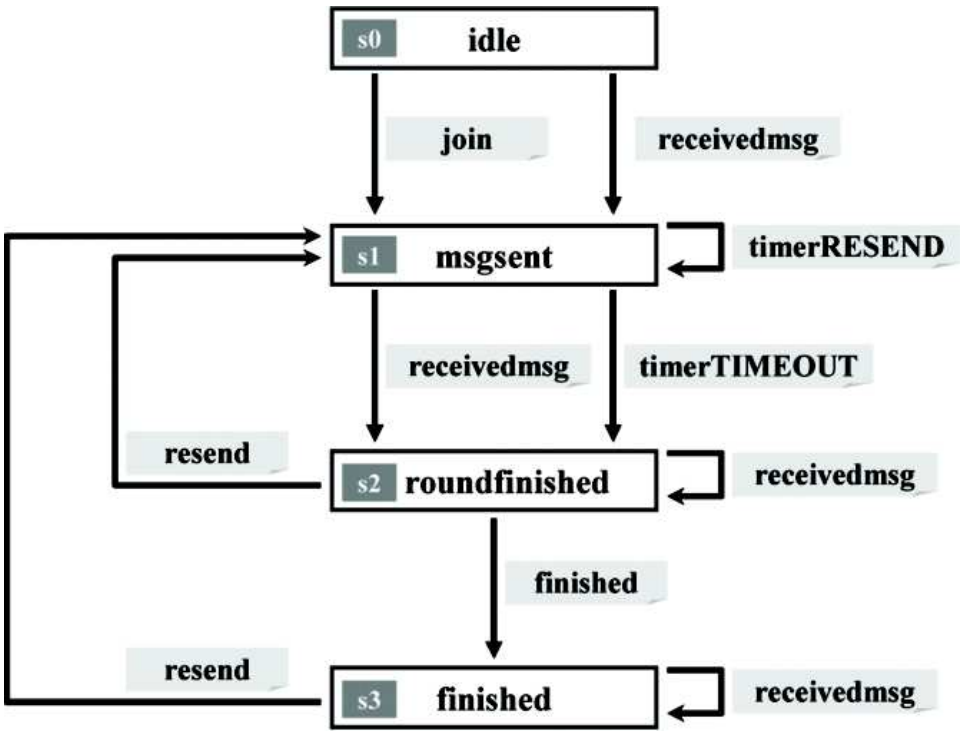

Figure A.1: The PBS finite state machine, taken from [5]

A.2 ZSS PBS Agent

The `ZSSpbsAgent` agent is basically an implementation of the algorithm shown in Listing 2.1 on Page 25. The `ZSSpbsAgent` is implemented as a *Finite State Machine (FSM)* consisting of the four states "idle", "msgsent", "roundfinished" and "finished". The "idle" state is the initial state of the FSM. Once a node joins a service it sends its neighborlist and waits in the "msgsent" state to get the neighborlists from its direct neighbors. When all the neighborlists have been received or a timeout occurred, the state changes to "roundfinished" and the node determines its own state. If there are no more CANDIDATE neighbors, the nodes switches to "finished" state, otherwise a new round is started by changing again to "msgsent" state. The FSM is shown in Figure A.1 and a detailed description of the transitions can be found in Table A.1.

From State	To State	Event	Action
idle	msgsent	join	The node joins the Service and starts sending out neighborlists.
idle	msgsent	receivedmsg	The node is still waiting in the idle state, but received a neighborlist. It starts now sending out neighborlists as well.
msgsent	msgsent	timerRESEND	The resend timer expired and not all neighbors sent a neighborlist back. Therefore, the already sent neighborlist will be resent.
msgsent	roundfinished	receivedmsg	All required neighborlists have arrived. The node determines its own state.
msgsent	roundfinished	timerTIMEOUT	Not all required neighborlists arrived, but the timeout timer expired and the node determines its own state.
roundfinished	msgsent	resend	There are still INT_CANDIDATE neighbors and a new round of the PBS algorithm needs to be started.
roundfinished	finished	finished	All nodes determined their status. There are no INT_CANDIDATE neighbors left.
finished	finished	receivedmsg	Handles incoming neighborlist even the node is already in the finished state. If required it sends a neighborlist back.
finished	finished	resend	Changes in the network and/or some INT_CANDIDATE neighbors have been detected. A new round of the PBS algorithm needs to be started.

Table A.1: Transitions of the PBS finite state machine, taken from [5]

A.3 Installation Guide

In order to install ns-2 with the above described PBS implementation including the link stability criterion, use the following procedure:

1. Download the ns-allinone package from the official ns-2 homepage and install it according the instructions.
2. Copy the `zoneserver` directory found on the enclosed CD-ROM in the `implementation` directory into the ns-2 main directory (e.g., `ns-allinone-2.29/ns-2.29`).
3. In order to extend the mac layer with the SNR monitoring as described in Section 4.3, copy the file `packet.h` from the `implementation/common/` directory of the CD-ROM to the ns-2 main directory and the file `mac-802_11.cc` from the `implementation/mac/` directory to the `mac/` directory.
4. Some changes in the ns source files are required to add the new agent, especially because it uses a new packet format:
   - The file `tcl/lib/ns-default.tcl` has to be edited, too. This is the file where all default value for the `Tcl` objects are defined. Insert the line `Agent/ZSS set weight_ 0` to set the default weight for `Agent/ZSS`.
   - The new packet has also to be added in the file `tcl/lib/ns-packet.tcl` in the `foreach prot{}` loop with the entry "ZSS".
   - The last change is a change that has to be applied to the `Makefile`. All sourcefiles have to be added after the `OBJ_CC=` list:
     ```
     zoneserver/zss.o
     zoneserver/zss_pbs.o
     zoneserver/debugger.o
     zoneserver/timer_pbs.o
     zoneserver/timer_monitor.o
     zoneserver/monitor.o
     zoneserver/neighborlist.o
     zoneserver/utils/fsm.o
     zoneserver/utils/state.o
     zoneserver/mobility_state_observation.o
     ```

```
zoneserver/mobility_prediction.o
```

5. Recompile ns-2 by using `make clean`; `make depend`; `make`.

6. After recompilation the `tcl` scripts can be used to run simulations with the command `ns example.tcl`. The prediction can be configured as explained in Section 4.4.

Appendix B

SIRAMON

B.1 About SIRAMON

SIRAMON (Service provIsioning fRAMwork for self-Organized Networks) [6] is a service provisioning framework with a decentralized and modular design. SIRAMON integrates the functions required to deal with the full lifecycle of services, such as service specification, look-up, deployment and management. It is designed as a Middleware, thus residing between the applications and the operating system and providing the service provisioning functions to the applications via an *API (Application Programming Interface)*. The basic structure of SIRAMON is depicted in Figure B.1. This figure also shows the different modules of which SIRAMON consists. Each module represents a different aspect of service provisioning:

Service Specification: The Service Specification module defines a universal service description language to describe the heterogeneous services and assist applications in the useage of this language. For the service description *XML Information Sets* [36] are used. Such an XML infoset defines an abstract data set in a tree structure and is normally described using a well-formed XML document [37].

Service Indication: The Service Indication module of the framework enables to advertise and discover services. Due to the mobile environment, the service indication is completely distributed and no central service directory can be used.

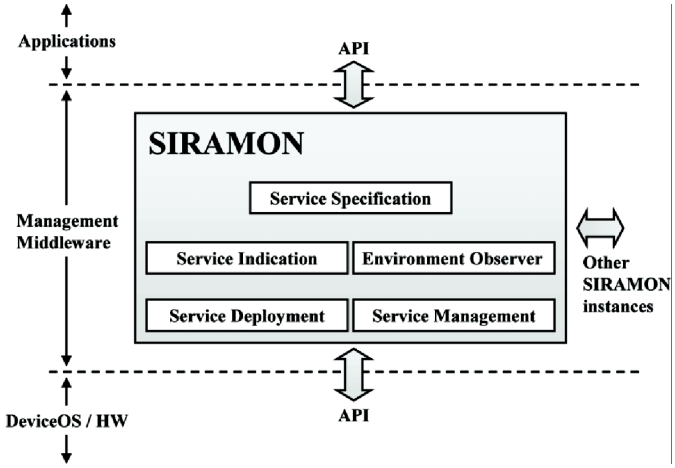

Figure B.1: SIRAMON structure

Service Deployment: The Service Deployment module is responsible for deploying a service. This requires the following actions:

- Requesting and downloading software according to the specification
- Discovering and gathering of resources
- Mapping of the specification to resources
- Configuring and installing the downloaded software
- Activating the service in a synchronized manner along with the other service participants
- Transferring the control to the Service Management module

Service Management: The Service Management module is responsible for the maintenance of running services. Maintenance includes the control of the service execution, the dynamic service adaptation to resource and environment variations, as well as user triggered service adjustment and the support of communication between both local and remote services. The PBS algorithm is part of this module and handles the distributed server selection.

Environment Observer: The Environment Observer module monitors the network, node and user context and makes this information available to the framework and its services. An application can query the Environment Observer about certain resource characteristics. It is also

possible to register watch statements, by which the application gets information when a resource characteristics falls below or rises above a certain threshold.

B.2 Testbed

In order to gain experience with SIRAMON not only by simulation but also in real world environments a testbed was installed. The mobile part of the testbed consists of four devices, three laptops and two PDAs[34]. Furthermore, in order to have more than five nodes for experimenting with, there are two desktop PCs which are also part of the network but are not mobile. All of the devices are running Linux as operating system and have SIRAMON installed. For the testbed, a SIRAMON implementation based on Java is used. The communication is done via wireless LAN 802.11b [9] (even between the desktop PCs), though the PDAs are also Bluetooth enabled. A snapshot of the testbed is shown in Figure B.2.

Figure B.2: SIRAMON testbed

The testbed is currently mainly used for demonstration purposes. In

[34] As PDAs Compaq iPAQ H3870 are used.

order to be able to demonstrate not only the dry working of SIRAMON and its server selection using PBS, an ad hoc multiplayer game called *Clowns* was developed using the Service Management functions of SIRAMON. *Clowns* is a jump-and-run game (a screenshot of *Clowns* is shown in Figure B.3).

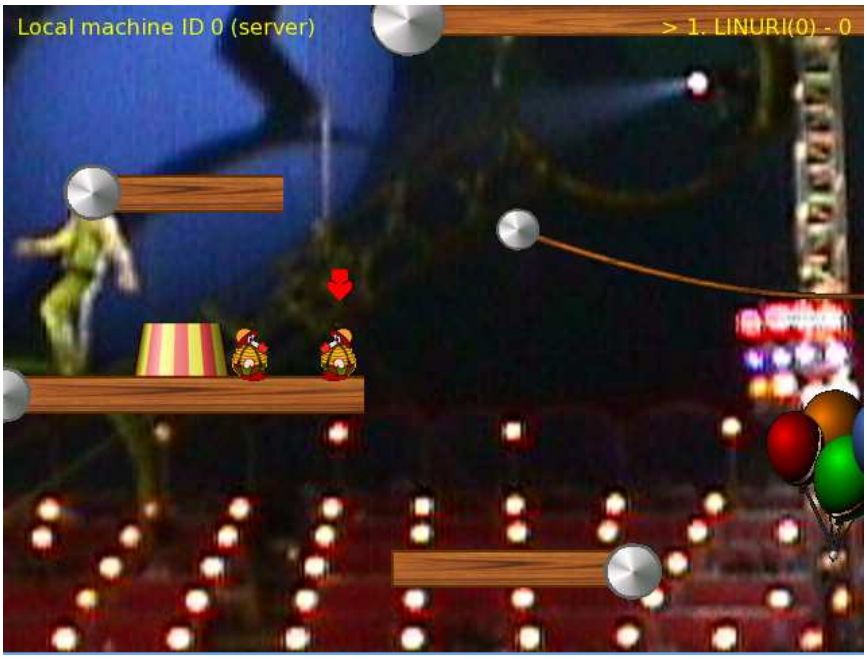

Figure B.3: Clowns screenshot

For more information about the current state of the SIRAMON project, please visit the project website [38].

Bibliography

[1] Department of Information Technology and Electrical Engineering. http://www.ee.ethz.ch/.

[2] Swiss Federal Institute of Technology Zurich. http://www.ethz.ch/.

[3] ETH Zurich. Computer Engineering and Networks Laboratory. http://www.tik.ee.ethz.ch.

[4] GAV - Gpl Arcade Volleyball. http://sourceforge.net/projects/gav/.

[5] F. Maurer. Service management procedures supporting distributed services in mobile ad hoc networks. Master's thesis, ETH Zurich, Computer Engineering and Networks Laboratory, August 2005. MA-2005-14.

[6] K. Farkas, L. Ruf, M. May, and B. Plattner. Real-Time Service Provisioning in Spontaneous Mobile Networks. In *Proceedings of the Students Workshop of The 24th Annual Conference on Computer Communications and Networking, (IEEE INFOCOM 2005)*, Miami, Florida, USA, March 2005.

[7] Information Sciences Institute ISI. The Network Simulator ns-2, February 2005. http://www.isi.edu/nsnam/ns/.

[8] The GNU General Public License. http://www.gnu.org/licenses/licenses.html.

[9] IEEE-SA Standards Boards. Part11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, June 2003. http://standards.ieee.org/getieee802/802.11.html.

[10] W. Su, S. Lee, and M. Gerla. Mobility prediction and routing in ad hoc wireless networks. *International Journal of Network Management*, 2000.

[11] H. S. Hassanein, H. Du, and C-H Yeh. Robust route establishment in high mobility manets. In *International Computer Engineering Conference ICENCO. Cairo, Egypt*, December 2004.

[12] Guojun Wang, Lifan Zhang, and Jiannong Cao. A virtual circle-based clustering algorithm with mobility prediction in large-scale MANETs. In *Networking and Mobile Computing, Third International Conference, ICCNMC 2005, Zhangjiajie, China, Proceedings*, August 2005.

[13] J.B. Tsui. *Fundamentals of Global Positioning System Receivers: A Software Approach*. New York: John Wiley and Sons, 2000.

[14] Wikipedia. Machine learning. http://en.wikipedia.org/wiki/Machine_learning.

[15] Andreas S. Weigend and Neil A. Gershenfeld. Time series prediction: Forecasting the future and understanding the past. *International Journal of Forecasting*, 10(1):161–163, 1994.

[16] David W. Aha. Editorial. *Artificial Intelligence Review*, 11(1-5):7–10, 1997.

[17] G. Bontempi. *Local Learning Techniques for Modeling, Prediction and Control*. PhD thesis, 1999.

[18] Gianluca Bontempi, Mauro Birattari, and Hugues Bersini. Local learning for iterated time series prediction. In *Proc. 16th International Conf. on Machine Learning*, pages 32–38. Morgan Kaufmann, San Francisco, CA, 1999.

[19] S.M. Riera, O. Wellnitz, and L. Wolf. A Zone-based Gaming Architecture for Ad-Hoc Networks. In *Proceedings of the Workshop on Network and System Support for Games (NetGames2003)*, Redwood City, USA, May 2003.

[20] E. Silva. Management of distributed services in manets. Master's thesis, ETH Zurich, Computer Engineering and Networks Laboratory, April 2006. MA-2006-07.

[21] Z. Zaidi and B. Mark. Mobility estimation based on an autoregressive model. Submitted to IEEE Transactions on Vehicular Technology, Jan. 2004. (Pre-print) Available at URL: http://mason.gmu.edu/zzaidi. 2004.

[22] Wikipedia. Artificial neural network. http://en.wikipedia.org/wiki/Artificial_neural_network.

[23] Joe Capka and Raouf Boutaba. Mobility prediction in wireless networks using neural networks. In *Management of Multimedia Networks and Services: 7th IFIP/IEEE International Conference, MMNS 2004, San Diego, CA, USA, Proceedings*, October 2004.

[24] Amiya Bhattacharya and Sajal K. Das. Lezi-update: An information-theoretic approach to track mobile users in PCS networks. In *Mobile Computing and Networking*, pages 1–12, 1999.

[25] Greg Welch and Gary Bishop. An introduction to the Kalman filter. Technical report, 1995.

[26] R. Jain. *The Art of Computer Systems Performance Analysis*. New York: John Wiley and Sons, 1991.

[27] J. Lewis. Fast normalized cross-correlation. in vision interface., 1995.

[28] Marc Greis. Tutorial for the Network Simulator ns. http://www.isi.edu/nsnam/ns/tutorial/index.html.

[29] Information Sciences Institute ISI. Virtual Inter-Network Testbed (VINT) project, October 1997. http://www.isi.edu/nsnam/vint/index.html.

[30] Eitan Altman and Tania Jiménez. Ns simulator for beginners, 2003. http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf.

[31] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC) - Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[32] Marc Bechler Sven Jaap and Lars Wolf. Evaluation of routing protocols for vehicular ad hoc networks in typical road traffic scenarios. In *Proceedings of the 11th Open European Summer School EUNICE 2005, Colmenarejo, Spain*, July 2005.

[33] Dirk Helbing Martin Treiber. Realistische mikrosimulation von strassenverkehr mit einem einfachen modell. In *16. Symposium "Simulationstechnik" ASIM 2002, Tagungsband, Rostock*, 2002.

[34] K. Fall. Ns notes and documentation. *The VINT Project*, February 2000.

[35] Y. Shang and W. Ruml. Improved mds-based localization. In *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China*, March 2004.

[36] World Wide Web Consortium (W3C). Xml information set. http://www.w3.org/TR/xml-infoset/.

[37] World Wide Web Consortium (W3C). Extensible markup language (xml). http://www.w3.org/XML/.

[38] Siramon: Service provisioning framework for self-organizing networks. http://www.siramon.org.