Diss. ETH No.
TIK-Schriftenreihe Nr. 83

Supporting Distributed Services in Mobile Ad Hoc Networks

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Sciences

presented by

KÁROLY FARKAS

M.Sc. in Technical Informatics,
Technical University of Budapest
born July 23, 1975
citizen of Hungary

accepted on the recommendation of
Prof. Dr. Bernhard Plattner, examiner
Prof. Dr. Lars Wolf, co-examiner

November 22, 2006

Abstract

As mobile devices, such as laptops, PDAs or mobile phones, are getting more and more ubiquitous and are able to directly communicate with each other via wireless technologies, the paradigm of wireless Mobile Ad hoc NETworks (MANETs) is gaining popularity. Especially distributed real-time applications, like multiplayer games, group-work or multimedia entertainment, are attractive for their users in this mobile ad hoc environment. However, MANETs impose new challenges because of their self-organizing, mobile and error-prone nature. Thus, provisioning distributed services in such an environment is quite difficult.

In this thesis, we focus on service management related issues and design, develop, implement and evaluate new mechanisms, which can aid the efficient creation of appropriate service management architecture for mobile ad hoc networks. Our contribution is three-fold.

First, we have designed and developed an algorithm called PBS (Priority Based Selection) to help manage distributed applications and support their smooth running in mobile ad hoc networks. PBS is based on graph theory and computes an appropriate Dominating Set (DS) of the network graph in a fully distributed manner applying node priority. The nodes in the DS then can be used as servers creating a robust, redundant client/server based service management architecture. Moreover, PBS is the first approach, according to our knowledge, in contrast to the existing DS computation algorithms that offers continuous maintenance of this set in dynamically changing network topologies. It shows a stable performance even in case of high node mobility, keeping the DS computation time nearly constant.

Second, we have designed and developed a mechanism called NWC (Node Weight Computation) to be applied in node priority comparison which reflects from the viewpoint of a given service the node's available computation and communication resources and its position in the network. NWC computes the node weight, on which the priority comparison is based, as the weighted linear combination of the node parameters. The parameter weights are extracted from the so-called service profile which reflects the characteristics and requirements of the given service. To create this profile containing the appropriate parameter weights we have applied factorial design. This technique assigns the highest priorities to the best suited nodes for a given service type and thus designates the most powerful nodes to be selected as servers by PBS.

And third, taking node mobility into account we have designed and developed a prediction mechanism called XCoPred to increase the stability of the selected server set. XCoPred predicts link quality variations based on pattern matching which can be exploited for mobility prediction. In contrast to most of the mobility prediction techniques applied today, XCoPred does not require the use of any external hardware nor reference points. Each node in the ad hoc network monitors the Signal to Noise Ratio (SNR) of its links to obtain a time series of SNR measurements. These measurements are filtered with a Kalman filter to decrease the level of measurement noise. When a prediction is required, the node tries to detect patterns similar to the current situation in the history of the SNR values of its links by applying the normalized cross-correlation function. The found matches are then used as the base of the prediction. For cases where no match can be found, we apply a fallback solution based on an autoregressive model. With this technique, fairly accurate link quality predictions around 2 dB of absolute average prediction error can be achieved in case of appropriate parameter settings and scenarios showing clear node mobility patterns. Integrating XCoPred into the PBS algorithm we can improve the stability of the selected server set and decrease the number of server changes substantially, in some cases even by approximately 25 %.

Moreover, all these algorithms/mechanisms, i.e., PBS, NWC and XCoPred, were implemented and investigated in the network simulator NS-2. As a proof of concept, we also implemented them in our mobile ad hoc testbed together with a demo multiplayer game application.

Kurzfassung

Durch die allgegenwärtige Verfügbarkeit drahtloser, kommunikationsfähiger, mobiler Endgeräte wie Laptops, PDAs oder Mobiltelefone, gewinnen neue Konzepte, wie das der Mobilen Ad Hoc Netze (MANET), immer mehr an Bedeutung. Speziell verteilte Echtzeit Anwendungen wie Spiele, Gruppenanwendungen oder Unterhaltungsmedien sind besonders attraktiv für die Teilnehmer in einer mobilen ad hoc Umgebung. Allerdings stellen diese MANETs neue Herausforderungen an die Entwickler, denn diese Netze setzen sich aus mobilen Endgeräten zusammen, müssen sich selbst organisieren und sind von Natur aus sehr unzuverlässig. Die Schwierigkeit liegt nun darin, verteilte Dienste auf einer derartigen Plattform anzubieten.

Diese Doktorarbeit konzentriert sich auf Aspekte der Dienstverwaltung, erstellt ein Konzept, und entwickelt sowie beurteilt neuartige Mechanismen für die Verwaltung von verteilten Diensten in ad hoc Netzen. Hierfür wurden die folgenden drei Beiträge geleistet:

Wir haben einen Algorithmus (Priority Based Selection - PBS) entworfen und entwickelt, welcher verteilte Anwendungen verwaltet und deren reibungslosen Ablauf ermöglicht. PBS ist ein graphenbasierter Ansatz der, mit Hilfe von Prioritäten, ein “Dominating Set (DS)” auf verteilte Art und Weise berechnet. Die Knoten des DS können dann als Dienstanbieter in einer robusten und redundanten Verwaltungsarchitektur eingesetzt werden. So weit uns bekannt ist PBS der erste Ansatz der ein DS dynamisch an die sich verändernden Verhältnisse im ad hoc Netz anpasst. Wir können dabei zeigen, dass, selbst in sehr dynamischen Netzen mit hoher Knotenmobilität, der Algorithmus eine stabile Leistung bei gleich bleibender Berechnungszeit erreicht.

Zweitens haben wir einen Mechanismus namens NWC (Node Weight Computation) entworfen und entwickelt, der die Wichtigkeit der Knoten berücksichtigt. Die Priorität eines Knoten wird dabei bestimmt durch seine Position und die zur Verfügung stehenden Rechenressourcen. Genauer gesprochen berechnet NWC seine Wichtigkeit mit Hilfe einer linearen Kombination verschiedener Knotenparameter. Diese Parameter werden vom so genannten Knotenprofil ausgelesen welches die Anforderungen an einen gegebenen Dienst beinhaltet. Dabei haben wir die Profile mit den dazugehörigen Parametergewichten mit Hilfe von “factorial design” erstellt. Diese Methode weist dem am besten geeigneten Knoten die höchste Priorität zu und erreicht damit, dass PBS immer die leistungsfähigsten Knoten auswählt.

Als dritten Beitrag haben wir einen Mechanismus zur Vorhersage der Knotenbewegung entworfen und entwickelt: XCoPred. Dank dieses Algorithmus wird die Stabilität der ausgewählten Knoten entscheidend erhöht. XCoPred ermittelt die Qualitätsschwankungen auf einem Link anhand von “Pattern Matching”. Im Gegensatz zu den meisten angewandten Techniken, ist XCoPred unabhängig von externen Informationsquellen denn XCoPred verwendet weder zusätzliche Hardware noch basiert er auf Referenzpunkten. XCoPred funktioniert folgendermaßen: Jeder Knoten im ad hoc Netz misst zu jeder Zeit die Link-Qualität (Signal to Noise Ratio - SNR) und speichert den Zeitverlauf dieser Messungen. Sobald eine Vorhersage erstellt werden soll, sucht der Knoten eine Messreihe mit ähnlichem Zeitverlauf. Hierfür wird eine Methode namens “normalized cross-correlation” verwendet. Gefundene Übereinstimmungen dienen dann als Vorlage für die Vorhersage. Falls keine Übereinstimmung gefunden wird, wird eine Alternativmethode basierend auf einem Autoregressionsmodel angewandt. Wir konnten zeigen, dass dank dieser Technik sehr genaue Vorhersagen mit einem Fehler von zirca 2 dB getroffen werden, falls die Systemparameter gut gewählt werden und die Knotenbewegung klare Strukturen aufweist. Durch die Integration von XCoPred in den PBS Algorithmus, könnten wir eine Stabilitätssteigerung von bis zu 25 % erreichen.

Alle erwähnten Algorithmen (PBS, NWC und XCOPred) wurden in den Netzwerksimulator NS-2 implementiert und getestet. Darüber hinaus haben wir die genannten Methoden in einem mobilen ad hoc Testnetz in eine Mehrspieler-Anwendung integriert und evaluiert.

Áttekintés

A mobil eszközök (például laptopok, mobil telefonok, palmtopok) széleskörű elterjedésével és ezen eszközök egymással való közvetlen kommunikációjában rejlő lehetőségekből következően az ún. vezeték nélküli mobil ad hoc hálózati (MANET) kommunikációs paradigma egyre nagyobb népszerűségnek örvend manapság. Különösen a valós idejű elosztott alkalmazások, többek között a többrésztvevős számítógépes játékok, csoport munka vagy multimédia alkalmazások számára nyújtanak ígéretes környezetet a mobil ad hoc hálózatok. Azonban ezen hálózatok új kihívásokat tartogatnak a kutatók számára az önmenedzselő, mobil és megbízhatatlan működésük következtében. Így elosztott alkalmazások támogatása meglehetősen nehéz feladat mobil ad hoc környezetben.

Ebben a disszertációban szolgáltatás menedzsmenttel kapcsolatos kérdésekre koncentrálunk és ismertetjük a megtervezését, kifejlesztését, implementációját és vizsgálatát olyan új eljárásoknak, melyek elősegíthetik a menedzselését elosztott alkalmazásoknak mobil ad hoc hálózatokban. Eredményeink a következőképpen összegezhetők:

Kidolgoztunk egy olyan algoritmust PBS néven (Priority Based Selection – prioritás alapú kiválasztás), amely segítségével egyszerűen kialakítható elosztott alkalmazások menedzselésére alkalmas architektúra mobil ad hoc környezetben. A PBS algoritmus egy gráfelméleten alapuló eljárás, amely az adott hálózati topológia gráfjának egy megfelelő csomóponthalmazát, ún. Dominating Set-jét (dominációs halmaz) vagy röviden DS-ét jelöli ki a csomópontok prioritása alapján teljesen elosztott módon. A DS-ben szereplő csomópontok szerverként használhatók, amelyek így egy robusztus, redundáns, szerver-kliens alapú szolgáltatás menedzsment architektúrát alkotnak. Továbbá, tudomásunk szerint ez az algoritmus az első olyan eljárás, ellentétben a manapság használatos DS számító algoritmusokkal, amely lehetővé teszi a ki-

jelölt DS folyamatos fenntartását még dinamikusan változó hálózati topológia esetén is.

Kidolgoztunk egy eljárást NWC néven (Node Weight Computation – csomópont súlytényező számítás), amelyen a PBS algoritmusban alkalmazott csomópont prioritások összehasonlítása alapul, és amely hűen reprezentálja a csomópont számítási és kommunikációs képességeit, valamint hálózati pozícióját az adott szolgáltatás szempontjából. Ez az eljárás a csomópont súlytényezőjét a csomópont paraméterek súlyozott lineáris kombinációjaként állítja elő. A paraméterek súlytényezőjét az ún. szolgáltatás profil tartalmazza, amely tükrözi az adott szolgáltatás követelményeit és sajátosságait. Ennek a profilnak az előállításához faktoriális tervezést (factorial design) alkalmaztunk, amelynek segítségével a megfelelő csomópontokhoz a legmagasabb prioritások rendelhetők, és így a PBS algoritmus ezen csomópontokat ki tudja választani a szerver szerepkörre.

Továbbá kidolgoztunk egy predikciós mechanizmust XCoPred néven, amely segítségével, figyelembe véve a csomópontok mobilitását, a kiválasztott szerver halmaz stabilitása növelhető. Ez a mechanizmus előrejelzi a csomópontok közötti vezeték nélküli linkek változásait mintaillesztés (pattern matching) használatával. Ellentétben a legtöbb, manapság alkalmazott predikciós mechanizmussal, ehhez az eljáráshoz nem szükséges egyéb külső eszköz vagy referencia pont használata. Minden egyes ad hoc hálózati csomópont folyamatosan méri a linkjei SNR (Signal to Noise Ratio – jel/zaj arány) értékét. A mérési zaj csökkentése egy megfelelő Kalman szűrő segítségével történik. Amikor előrejelzésre van szükség, a csomópont az aktuális SNR értékek alkotta mintához hasonló mintákat keres az SNR mérések sorozatában normalizált kereszt korreláció (normalized cross-correlation) alkalmazásával. Az így talált mintákat követő SNR értékek felhasználásával a predikció már könnyen elvégezhető. Amennyiben nincs az aktuális mintához eléggé hasonló minta az SNR mérések sorozatában, a predikcióhoz egy autoregresszión alapuló modellt használunk. Az XCoPred mechanizmus segítségével akár 2 dB pontosságú előrejelzés is elérhető megfelelő paraméter beállítások esetén. Továbbá ennek a predikciós eljárásnak a használatával kiegészített PBS algoritmus által kiválasztott szerver halmaz stabilitása tekintetében is lényeges növekedés figyelhető meg, némely esetben akár 25 %-os is.

És végül, az összes említett algoritmust/eljárást (azaz a PBS-t, NWC-t és az XCoPred-et) implementáltuk és intenzív vizsgálatoknak vetettük alá az ‘NS-2’ hálózati szimulátorban. Ezen felül implementáltuk őket mobil ad hoc teszthálózatunkban is egy többrésztvevős játék alkalmazással együtt.

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

This chapter gives an overview about the content of this thesis. First, it provides the necessary information about the thesis context describing the area and motivating the research reported in this thesis, explaining the used taxonomy, and giving the problem statement. Then, the thesis goals and contributions are summarized. And finally, the structure of the thesis is presented.

1.1 Thesis Context

1.1.1 Motivations

The number of powerful mobile devices, such as laptops, PDAs or smart mobile phones, with wireless networking support is constantly increasing these days. Direct communication between these devices makes data exchange quick and cheap leading to the formation of multi-hop, where direct connection is not possible between the end nodes and intermediate nodes help data delivery, wireless Mobile Ad hoc NETworks (MANETs) [1]. In such networks, the devices communicate directly in a spontaneous, ad hoc manner without relying on any pre-existing infrastructure or central administration (cf. Figure 1.1).

MANETs historically were mainly propagated for situations like disaster recovery or military applications providing connectivity between the troops on a battlefield. We expect, that in the near future the ad hoc communication paradigm will be a useful way of data exchange in a broader range of everyday applications. For example, distributed real-time applications, such as multi-

Figure 1.1: *Mobile Ad Hoc Network*

player games, document sharing, multimedia entertainment, Voice-over-IP or the so-called 'edutainment' area are the most attractive candidates to be used over mobile ad hoc networks [2]. Using MANETs the users do not have to be continuously connected to infrastructure-based networks (such as GPRS [3], UMTS [4] or WLAN [5]) to be able to communicate with other users, and thus they do not have to pay for communicating via a network provider's infrastructure either. The exceptional feature of a mobile ad hoc network is that nobody owns it but everybody can be part of it. Furthermore, an ad hoc network can be easily created even where infrastructure-based networks are hardly available, like in a mountain area or in a desert.

1.1.2 Terminology

To put MANETs into operation the participating nodes must approach each other to be in communication range and organize themselves spontaneously into a multi-hop network. Moreover, they must provide data relaying and service provisioning functionalities for distant nodes and not only act as terminals. In this context, we use the term *service* as a collection of useful functions and procedures for the users implemented by the application. Thus, in our terminology the terms service and application have similar meaning. For example, a file sharing service is a collection of file lookup/advertisement and file download/upload procedures implemented by a file sharing application like Kazaa [6]. To be able to find and use the services in the network, common *service provisioning* functionalities, such as service description, discovery, deployment and management, have to be implemented and provided by the nodes forming the ad hoc network. In our terminology, *service description* specifies the role of the device in the service, and the functions and connections of service elements to build the service. *Service discovery* handles service advertisement if the node hosts a service, or service lookup if the node intends to use a service. *Service deployment* covers the creation, installation and configuration of services. And *service management* performs service maintenance, which handles *state management* to synchronize the service state on the different service nodes, service reconfiguration and termination.

In this thesis, we mainly focus on service management issues. There are three basic models to implement the service management functions in communication networks, namely, the *centralized*, the *fully distributed* and the *hybrid* model [7]. In the centralized model, the so-called *client/server* architecture is used where the service client nodes connect to a single server node which exclusively handles the service management issues. When it is required, the clients send service state updates to the dedicated server machine and the server, after completing the necessary computations, sends authoritative service updates back to the clients. In the fully distributed model, which is the other end of the spectrum, the so-called *peer-to-peer* architecture is used, where the role of every node is equal (client and server at the same time) and the nodes implement the service management functions together, in a distributed way. Each node maintains a local copy of the service state and informs every other node whenever the service state changes. As a trade-off between these two extreme solutions, the hybrid model uses the *zone-based* architecture [8], where the network is divided into separate zones and in every zone a dedicated server node handles the service management issues. The zone server receives service state updates from its clients belonging to the zone and, if it is necessary, synchronizes the service state information among the other zone servers which serve their clients with this information.

1.1.3 Problem Statement

Using an appropriate service management architecture in the mobile ad hoc environment is crucial to help the spreading of MANETs in everyday life.

This architecture must be able to cope with the inherent properties of MANETs. In general, a mobile ad hoc network consists of resource constrained (e.g., limited computation, storage and/or battery power) heterogeneous devices which can move freely and which organize themselves in an ad hoc way to form a network. The communication links between the nodes are unreliable and error-prone wireless connections and there is no pre-established central infrastructure to administer and manage such networks. The architectures of traditional service provisioning/management solutions used in communication and data networks (e.g., Jini [9], UPnP [10], SDP [11], Chameleon [12]) are not well suited for MANETs and thus it is difficult to adapt them for the mobile ad hoc environment. Usually they are either based on the centralized service management model using the client/server architecture or the fully distributed model using the peer-to-peer architecture. The shortcomings of the client/server architecture are the low fault tolerance due to the reliance on a single, central server node and the limited scalability of the server. On the other hand, the main drawback of the peer-to-peer approach is the limited scalability due to the high communication overhead of state management, though this architecture provides good fault tolerance properties. As a compromise, the hybrid service management model using the zone-based architecture offers a high level of fault tolerance due to the inherent redundancy and provides much better scalability properties than the peer-to-peer architecture, thus it is well suited for MANETs.

However, the main challenge is the creation of the zones and the selection of the zone servers in MANETs to be able to implement the zone-based service management architecture. This has to be carried out in an efficient and distributed way. The number of zones has to be as small as possible to reduce the communication overhead of state management between the zone servers, but at the same time the most powerful nodes (concerning available communication and computation resources) have to be used as zone servers and the client nodes in the zones have to be as close as possible, usually one hop away, to their server node to keep the communication delay low. This latter condition is especially critical in case of real-time applications.

1.2 Thesis Goals and Contributions

The primary goal of this thesis is to help the implementation of the zone-based service management architecture in MANETs and thus design, develop, implement and evaluate new mechanisms, which can aid the efficient creation of zones and the selection of zone servers. Moreover, to prove the viability of our concepts we intended to implement the developed mechanisms together with a demo distributed application in a mobile ad hoc testbed.

To create the zones and select the zone servers we have developed a distributed Dominating Set (DS) computation algorithm called PBS (Priority Based Selection) [13–15]. A Dominating Set is a subset of the graph's nodes, such that all the nodes are either part of the DS or are directly connected to a member of the DS. Our PBS algorithm is a graph theory based procedure that computes an appropriate DS of the ad hoc network graph in a distributed manner containing nodes which can be used as zone servers. The client nodes one hop away from their zone server together with this server node form the zones. This structure fits well with our requirements mentioned above to implement the zone-based architecture. Moreover, to ensure the smooth running of the distributed application, the set of zone servers must be maintained and recomputed on the fly when it is required (e.g., in case of network topology changes or link failures). PBS is the first algorithm, according to our knowledge, that offers continuous maintenance of the DS when the network graph changes dynamically. PBS shows a stable performance even in case of high node mobility, keeping the DS computation time nearly constant.

To select the most powerful nodes as zone servers, PBS compares the priority of the nodes which reflects from the viewpoint of a given service the node's available computation and communication resources and its position in the network. We have developed a mechanism called NWC (Node Weight Computation) [16] to be applied in node priority comparison based on a set of node parameters and the characteristics of the service being used. NWC computes the node weight, on which the priority comparison is based, as the weighted linear combination of the node parameters. The parameter weights are extracted from the so-called service profile. The role of this profile is to reflect the characteristics and requirements of the given service, and it simply contains the appropriate parameter weights which are computed or given by the service designer in advance. As the technique to compute these parameter weights we have applied factorial design which can be used by the service designer to create the service profile. With this technique we were able to assign the highest priorities to the best suited nodes for a given service type and thus designate the most powerful nodes to be selected as zone servers.

To increase the stability of the selected server set, taking node mobility into account is indispensable. High mobility of the nodes can result in the selection of unstable zone server set leading to frequent changes of the server nodes and thus frequent handovers of clients between them. This also causes high state management traffic overhead or even service disruption. Mobility prediction can mitigate this problem. It can help increase the stability of the server set by predicting future changes of the network topology and using this information in server selection. We have developed a mechanism called XCoPred [17,18] to predict the variations of the wireless link quality based on pattern matching which can be exploited for mobility prediction. In contrast to most of the mobility prediction techniques applied today, XCoPred does not require the use of any external hardware nor reference points. Each node in the ad hoc network monitors the signal quality of its links to obtain a time series of link quality metric which is derived from the Signal to Noise Ratio (SNR) measurements of the links. These measurements are filtered with a Kalman filter to decrease the level of measurement noise. When a prediction is required, the node tries to detect patterns similar to the current situation in the history of its links' SNR values and tries to obtain a set of predictors. For this purpose, the node computes the normalized cross-correlation between the current pattern and the history of the links' quality. From the detected set of predictors the most probable predictor is used as the prediction of future link quality. For cases where no predictors can be found, we apply a fallback solution based on an autoregressive model. Using XCoPred, highly accurate link quality predictions can be achieved with around 2 dB of absolute average prediction error in case of appropriate parameter settings and scenarios showing clear node mobility patterns. Moreover, integrating XCoPred into the PBS algorithm we can improve the stability of the selected server set and decrease the number of zone server changes substantially, in some cases even by approximately 25%. Note that extending XCoPred with network topology prediction features, e.g., using MDS (MultiDimensional Scaling) [19] on top of XCoPred to predict topology changes, its applicability could be substantially increased from supporting routing decisions via traffic engineering to even application layer usage in mobile ad hoc networks.

In summary, we make the following contributions in this thesis:

1. We design and develop a distributed Dominating Set computation algorithm called PBS. This algorithm computes and maintains an appropriate DS of the ad hoc network graph based on node priority in a fully distributed manner containing nodes which can be used as zone servers.

2. We design and develop a mechanism called NWC to calculate node weight to be used in the node priority comparison of PBS. NWC calcu-

lates the node weight based on a set of node parameters and the characteristics of the given service. It assigns the highest priorities to the best suited nodes for a given service type and thus designates the most powerful nodes to be selected as zone servers.

3. We design and develop a mechanism called XCoPred to help increase the stability of the selected zone server set via prediction. XCoPred predicts the variations of the wireless link quality based on pattern matching which can be exploited in assessing future changes of the network topology. Then, using this information in server selection the stability of the selected zone server set can be increased.

Moreover, all these algorithms/mechanisms, i.e., PBS, NWC and XCoPred, were implemented and investigated in the network simulator NS-2 [20]. As a proof of concept, we also implemented them in our mobile ad hoc testbed together with a demo multiplayer game application.

1.3 Thesis Structure

The remainder of this thesis is organized as follows:

- In Chapter 2 (Service Provisioning in Mobile Ad Hoc Networks), we survey the difficulties of service provisioning in mobile ad hoc networks and identify the different service provisioning phases using a sample mobile ad hoc application scenario. Moreover, we introduce and briefly describe our service provisioning framework called SIRAMON which accommodates and implements common service provisioning functions for mobile ad hoc environments.

- In Chapter 3 (Dominating Set Based Service Management Architecture in MANETs), first we discuss the centralized client/server and the distributed peer-to-peer architecture used today for implementing service management functions in traditional networks and introduce the zone-based architecture which is more suitable for MANETs than the previous ones. After that, we present our PBS algorithm we developed for zone server selection. Moreover, we discuss our NWC mechanism applied to compute the node weights for comparing node priorities in PBS.

- In Chapter 4 (Mobility Prediction in Mobile Ad Hoc Networks), first we survey the fundamentals of mobility prediction which can be exploited in increasing the selected zone server set's stability. Then, we present our link quality prediction mechanism called XCoPred we have developed for mobility prediction in MANETs. Moreover, we point out how XCoPred can be used in the PBS algorithm.
- In Chapter 5 (Implementation), we provide an overview about how we implemented the algorithms and mechanisms being developed during this thesis work to prove the viability of our concepts. The implementation also forms the basis for our investigations and evaluation. Thus, we first describe the implementation of PBS, NWC and XCoPred in the NS-2 network simulator. Then, we discuss their implementation in our SIRAMON framework, pointing out the SIRAMON testbed we built and a demo application, a simple real-time multiplayer game called Clowns, which we implemented to demonstrate the usefulness of SIRAMON.
- In Chapter 6 (Evaluation), first we present the specification of a pseudo real-time multiplayer game what we have used as the test application in our simulations. After this, we show our evaluation of the PBS algorithm, the technique of service profile creation for node weight computation using factorial design and simulations, and our evaluation of the XCoPred prediction mechanism together with its application in PBS. Moreover, we give a short, simulation based comparative study of the centralized client/server service management architecture, the distributed peer-to-peer architecture and the zone-based architecture using PBS.
- In Chapter 7 (Related Work), we give a brief overview about the state-of-the-art approaches related to our work. First, we present previous work related to the selection of management nodes in MANETs and compare these approaches to our PBS algorithm and NWC mechanism. Furthermore, we discuss the most interesting proposals related to mobility prediction in mobile networks comparing them to our XCoPred prediction mechanism.
- Finally, in Chapter 8 (Conclusions) we recall the context of our work and briefly review our contributions. Moreover, we give a critical assessment of the achievements followed by a short discussion about future work.

Chapter 2

Service Provisioning in Mobile Ad Hoc Networks

Service provisioning in mobile ad hoc networks faces special difficulties due to the constraints of the ad hoc environment—such as lack of central infrastructure, high level of device heterogeneity, degree of mobility, limited device and network resources. In this chapter, we survey these difficulties and identify the different service provisioning phases using a sample mobile ad hoc application scenario. Moreover, we introduce and briefly describe our service provisioning framework called SIRAMON (Service provIsioning fRAMework for self-Organized Networks) which accommodates and implements common service provisioning functions for mobile ad hoc environments.

2.1 Sample Application Scenario

We use an imaginary scenario of a real-time multiplayer computer game in a mobile ad hoc network to explain the different phases of service provisioning.

A tourist, travelling by train, wants to spend his travel time playing a multiplayer computer game. To do so, he has a mobile device (e.g., a laptop) with a game software installed on it. Some other passengers on the train are supposed to possess also mobile devices that are able to directly communicate among each other and, at least, a couple of them are willing to play. While only some devices are capable to actively participate in the game, all may act as a relay to forward data. Since they have come close together, a mobile ad hoc network has been setup spontaneously.

Upon establishment of the ad hoc network, the initiator device advertises the availability of a new application/service, i.e., the game, on the network. When each of the connected devices is informed their users can decide whether to join the game or not. Thereafter, the service will be automatically deployed on the selected devices (and in the network) and the game can be started. Further passengers are allowed to join (if the game allows for late arrivals) and leave the ongoing multiplayer game at any time. Even the initiator must be able to leave without interrupting the game session for the remaining players.

2.2 Basic Assumptions

With regard to the network, the devices and the game application we assume some prerequisites. These are the following:

Mobile ad hoc network: The ad hoc network is connected and provides routing functionality running an ad hoc routing protocol. Thus, any node can communicate with any other node in the network. Moreover, network resource discovery and management are also given (e.g., set up and maintain multicast communication infrastructure if the network is multicast capable).

Device: On every device an Operating System is running which includes a Network Software part and a Device Resource Manager part. The Network Software handles data transfer, routing and network resource discovery/management related issues. The Device Resource Manager monitors and controls the local resources of the device. Moreover, every device is furnished with our service provisioning framework (see section 2.4.1).

Game application: A multiplayer computer game is a multiuser application, a group service. The players in the game world interact with the game objects and one another via their device running the game client software to form the game story. Usually the player who collects the most points or kills the most enemies wins the game. Today most of the multiplayer computer games are implemented in a centralized manner following the client/server model [7]. The game clients are connected to the game server which controls the game running. However, in a mobile ad hoc network there is no dedicated game server node, thus one of the devices has to run the game server software and thus host the game service, or the game control has to be implemented in a distributed manner and included in the game client software. We assume, that the required software(s) to run the game is/are installed at least on one device in advance before the ad hoc network is formed and this/these software(s) can be distributed in the network and installed on the other devices to make them capable to join the game.

2.3 Phases of Service Provisioning

Analyzing the sample application scenario we can identify the following phases of service provisioning.

In the first phase, the service to be provisioned has to be specified and named. We call this phase **Service Description**. The service specification must be able to describe the *role* of the node in the service, the *functions* of the service elements and the *connections* among them. For example, in our game service the specification has to describe the participating device's role (e.g., active player or auxiliary node which just aids the service to run), the device-level local view (the game is composed from moving pictures with music, sounds and voice requiring real-time interaction from the player, etc.), and the network-wide global view (the device has to communicate real time with all other player devices, etc.) of the game.

When the ad hoc network is established, the initiator device (the host) has to *advertise* the game service such that new participants can join. On the other hand, the devices of the other passengers must be able to *lookup* the service before deploying and using it. This phase is called **Service Discovery**. Mobile networks have special constraints which must be addressed to develop appropriate service discovery mechanisms. Usually, MANETs cannot provide a permanent, central directory where the announced services can be registered and from where the available services can be read out. Moreover, the service hosting role can change dynamically in a mobile ad hoc group service. For example, in our scenario the initiator device hosted the game service but when it quits the game some other device has to take this role over or this role has to be implemented in a distributed manner.

When the devices in the ad hoc network are informed about the game service the **Service Deployment** phase is entered with *requesting, downloading, installing, configuring* and *activating* the game software. Every device has to accomplish these procedures separately but in a synchronized manner to get a consistent state by when the game starts. In general, heterogeneous devices form the ad hoc networks (e.g., laptops, PDAs, mobile phones) representing several different platforms. This requires all the ported versions of the service software to these platforms or the service has to be implemented in a platform independent way.

After the game started the service *maintenance, reconfiguration* and *termination* functions come to the front. We call this phase **Service Management**. In the course of service maintenance the dynamic adaptation of the service to the resource variations must be assured. The players may experience frequently degraded performance of the service due to dynamic changes in network conditions (e.g., sudden drop in bandwidth when one of the player devices moves away from the others hereby reducing its communication capabilities) or other resources such as local computational cycles and memory space. Service reconfiguration takes place fundamentally in two forms, locally and globally. We are talking about the former if a service user device modifies the configuration of its local service instance (e.g., selecting a better resolution for the pictures). In the latter, the network-wide global view of the service session changes, for example, when a new player joins the game session. Service provisioning must support global reconfiguration. Service termination can be considered as a special reconfiguration if a service device stops running the service instance, such as the game initiator when it quits the game. This incurs also the release of the reserved resources.

Dynamic adaptation to the resource variations and reconfiguration of the service session demand continuous monitoring of the resources and the service context. This comprises the gathering and transformation of resource and context information into an appropriate form which can be used as input in the aforementioned functions (e.g., in service maintenance or reconfiguration).

Figure 2.1 depicts the logical/chronological sequence of the functions and phases discussed above.

2.4 Generic Framework for Service Provisioning

After surveying the characteristics of service provisioning in self-organized or ad hoc networks we collected the discussed functions into a generic service provisioning framework, called SIRAMON (Service provIsioning fRAMework for self-Organized Networks) [21–26]. Since these functions are to be implemented in case of every application it is a reasonable design choice to gather and implement them at one place in the form of a middleware/frame-

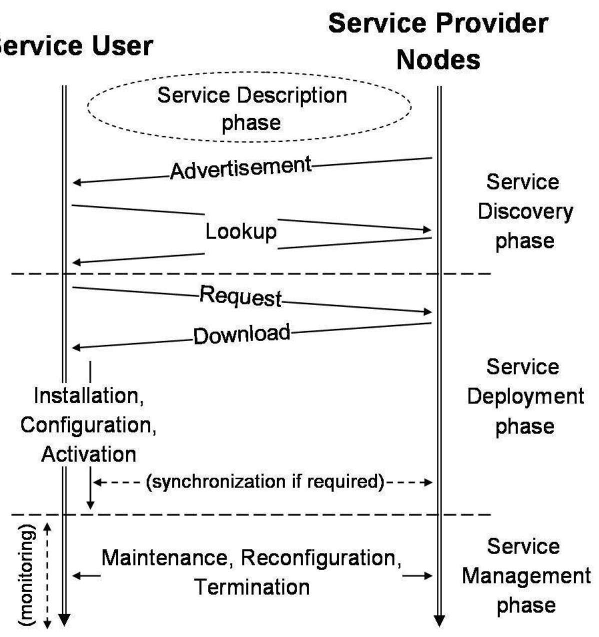

Figure 2.1: *Sequence of Service Provisioning Functions*

work. Hence, the application developers do not have to deal with these common functions rather focus on only the application specific issues.

2.4.1 SIRAMON, Our Service Provisioning Framework

Our service provisioning framework is based on a decentralized and modular design. Every device runs an instance of SIRAMON which handles the control and synchronization among the devices, as well. We integrated the functions of the different service provisioning phases discussed above into separate modules. This makes the framework generic giving the possibility to replace the functions of a module with others which may fit better with the actual environment or the application's needs. Moreover, we do not want to re-invent the wheel, thus with a modular design we can easily integrate procedures which have been already developed for given functions (such as service discovery or service deployment).

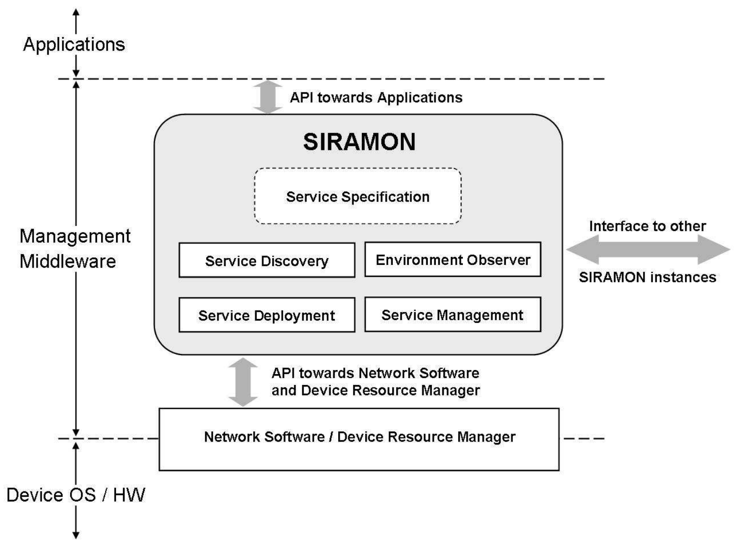

Figure 2.2: *Ad Hoc Device Model with SIRAMON*

In our device model (see Figure 2.2), we introduced a management middleware layer where SIRAMON, integrating the service provisioning functions, is located. This layer provides an API (Application Programming Interface) towards the device's local and network resources and towards the applications. Moreover, it provides an interface to SIRAMON instances running on other devices. As we mentioned above, the Network Software handles data transfer, routing and network resource discovery/management related issues. The Device Resource Manager is responsible for monitoring/controlling the local resources and mapping them to service elements. The Network Software and the Device Resource Manager are not part of the SIRAMON framework, rather they belong to the device Operating System. In the following, we briefly describe the different SIRAMON modules.

Service Specification

The Service Specification module defines the service descriptor used in the given service as we discussed in Section 2.3. SIRAMON is not bounded to any specific service description technique. The only requirement is that the service descriptor must be able to describe the role of the node in the service, the functions of the service elements and the connections among them. In case of different type of services, it can be reasonable to use different service description techniques if the other technique suits better for describing the given service. For example, a simple, location based service (e.g., a network printer) can use an attribute-value based compound service description method [27]. In this case, the service is described by using a hierarchical attribute-value pair tree structure which reflects the location of the resource (i.e., the printer). On the other hand, a complex, distributed group service, such as a multiplayer game, can be more appropriately described by using a component-based service description technique [21] in which the service is composed by simpler, reusable service components in a well defined way. However, the service descriptor should be carefully selected because it constitutes the basis of service provisioning and it is used in the course of the whole provisioning activity.

Service Discovery

This module covers service advertisement and service lookup procedures (see Section 2.3). However, due to the lack of central infrastructure the traditional central directory model [7] cannot be applied in ad hoc environments. One solution is to replicate this directory on every device connected to the ad hoc network [28]. This introduces synchronization and propagation overhead to maintain a consistent database on all the devices but makes service lookup very fast and efficient. Another solution is to establish a so-called virtual backbone in the ad hoc network selecting some devices which store a copy of the service directory [29]. The virtual backbone must be reachable by every other device (i.e., every device has to be able to reach at least one backbone node within one hop). This approach slows down the service lookup and is less efficient compared to the previous one. However, it introduces less administrative overhead, but requires an extra procedure to select and maintain the set of virtual backbone nodes. In case of mobile ad hoc group applications, the latter solution seems to be a better choice, since the devices participating in the given service constitute by themselves an overlay network which can be considered as part of the virtual backbone.

Service Deployment

Deploying a service, as we discussed in Section 2.3, requires the following functions which are included in the Service Deployment module:

- Requesting and downloading the service software according to the specification;
- Discovering and gathering resources;
- Mapping the service specification to resources;
- Configuring the resources, installing and configuring the downloaded softwares;
- Activating the service in a synchronized manner along with the other service participants;
- And handing the control on to the management module.

Traditionally, in infrastructure-based networks service deployment consists of two levels: the network level and the node level [12]. Briefly, on the network level the participants of the service are to be explored and the required network resources to be discovered and selected, since on the node level the appropriate service components are to be deployed on the selected resources. However, in ad hoc networks these two levels cannot be clearly distinguished due to the lack of central infrastructure and the mobile behavior of the devices. It can easily happen that a previously selected relay node moves away even during the deployment phase and another has to be selected on the network level of the deployment activity. Thus, the network and node level service deployment functions have to synchronize continuously each other during the whole deployment phase.

The Service Deployment module has to tightly co-operate with the Environment Observer for getting information about the available resources and with the Network Software/Device Resource Manager for mapping the resources to the service components.

Service Management

This module integrates the service maintenance, reconfiguration and termination functions (cf. Section 2.3).

Service maintenance is responsible for the dynamic adaptation of the service to the resource variations of the nodes in order to optimize the user's perceived service quality. Moreover, it has to keep up the smooth running of the service even in case of network topology changes caused by node mobility or link failures.

Service reconfiguration, as discussed above, has two fundamental forms, local and global reconfiguration. Local reconfiguration takes place when the context of the device changes or the service user intends to modify the running service session. In global reconfiguration, the network-wide global view of the service session changes (e.g., a new user joins). The Service Management module gives support for global reconfiguration while local reconfiguration is in charge of the application.

Service termination can be considered as a special form of reconfiguration when a device stops running the service instance. All the other service participants have to be informed about the termination and the reserved resources have to be released.

The Service Management module also has to tightly co-operate with the Environment Observer and the Network Software/Device Resource Manager.

Note that in the rest of this thesis, we only focus on mechanisms (PBS, NWC, XCoPred) what we have developed to aid the management of distributed applications in mobile ad hoc networks. These mechanisms have been integrated into SIRAMON and implemented as part of the Service Management module. The detailed discussion of the other SIRAMON modules is out of scope of this thesis, though in our SIRAMON testbed we implemented also these modules (see Section 5.2.1 and Section 5.2.2).

Environment Observer

This module is responsible for monitoring the device resources and the service context. Resource and context information have to be gathered and transformed into an appropriate form, which can serve as input for the deployment and management modules.

2.5 Chapter Summary

In this chapter, we have surveyed the different service provisioning phases in mobile ad hoc networks via a sample multiplayer game application scenario and discussed their challenging aspects. Moreover, we have introduced and briefly described our generic service provisioning framework called SIRAMON which collects and implements the common service provisioning functions. More information about SIRAMON can be found on its homepage [30].

The rest of this thesis covers only mechanisms we have developed to aid the management of distributed applications in mobile ad hoc networks. These mechanisms, namely PBS, NWC and XCoPred, have been part of SIRAMON and implemented in its Service Management module.

Chapter 3

Dominating Set Based Service Management Architecture in MANETs

An appropriate architecture is essential for implementing service management functions in mobile ad hoc networks. In this chapter, first we discuss the centralized client/server and the distributed peer-to-peer architecture and introduce the zone-based architecture which is more suitable for MANETs than the previous ones. After that, we present our PBS (Priority Based Selection) algorithm we developed for zone server selection. And finally, we show our NWC (Node Weight Computation) mechanism we are using to compute the node weights for comparing node priorities in the PBS algorithm.

3.1 Fundamentals

In the following, we discuss briefly the different service management architectures which are used in the implementation of the service management functions (see Section 2.3 for discussion on these functions) in communication networks nowadays. Moreover, we point out a hybrid, zone-based architecture proposed for mobile ad hoc environments and its relation to Dominating Sets. Then, we give a short explanation of the notations and definitions relevant to our proposed Dominating Set computation algorithm.

3.1.1 Service Management Architectures

Today's service management architectures in infrastructure based networks are following two fundamental models: the centralized, client/server and the fully distributed, peer-to-peer model. However, the majority of the management architectures uses the client/server model [7].

In case of the *client/server based* architecture, every service client node connects to a single central server machine that exclusively handles the service management issues. When it is required, the clients send service state updates to the dedicated server node and the server, after completing the necessary computations, sends authoritative service updates back to the clients. The main problems with this model, which make it unsuitable to be used in MANETs, are reduced fault tolerance (a centralized server represents a single point of failure), limited scalability (computation and bandwidth problems may arise if too many service clients[1] are connected to the dedicated server node and the management of the given service is computation or communication intensive) and required central administration.

The fully distributed *peer-to-peer* architecture follows the opposite philosophy. Here each node maintains a local copy of the service state and informs every other node whenever the service state changes. With this architecture, a good fault tolerance level can be achieved because there is no single point of failure. However, this architecture suffers from limited scalability (as everybody communicates to everybody else, the bandwidth or computation resources required at the nodes can be pretty high) and synchronization issues.

As a compromise between these two extreme solutions, the *zone-based* service management architecture proposed in [8] provides a robust, redundant client/server model that is more appropriate for the mobile ad hoc environment. In this approach, some nodes act as zone servers and each zone server is in charge of a small group of clients. For efficiency reasons, this group should be close to its zone server. The zone server receives service state updates from its clients belonging to the zone and, if it is necessary, synchronizes the service state information among the other zone servers which serve their clients with this information. When a zone server loses connection, shuts down or disappears its clients still will be able to keep running the service by reconnecting to another zone server. The zone servers are topolog-

[1]A few dozen clients, depending on the application, can already overload a server node in a mobile ad hoc environment.

ically distributed across the network and the clients connect to their closest server. Figure 3.1 shows an example of the zone-based service management architecture in which each group has its own zone server. Node 1, 3 and 4 are connected to zone server 2 while node 6, 7, 8 and 9 are connected to zone server 10.

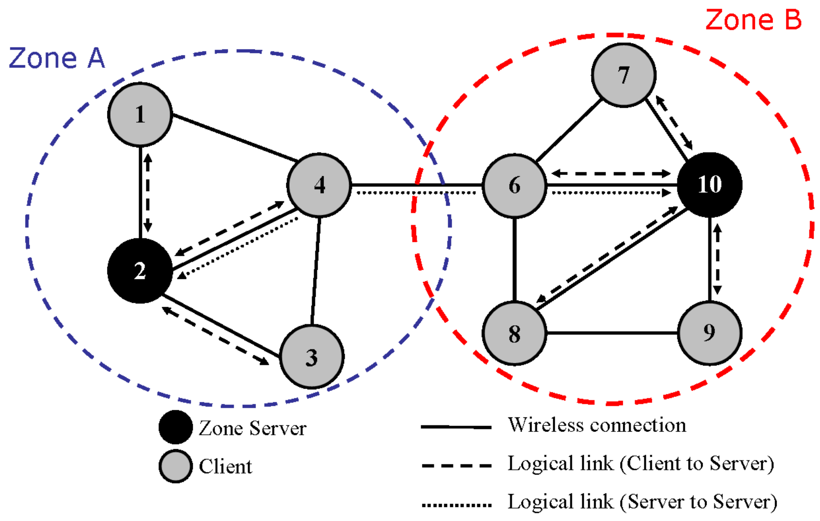
Zone A
Zone B
1
2
3
4
6
7
8
9
10
Zone Server
Client
Wireless connection
Logical link (Client to Server)
Logical link (Server to Server)

Figure 3.1: *Zone-Based Service Management Architecture in MANETs*

However, the zone server nodes must be selected carefully and this selection must be maintained even in case of changes in the network topology or the available resources. Since there are no dedicated zone server nodes in mobile ad hoc networks, the most powerful user devices (concerning available computation and communication resources) also taking into account their position in the network should act as zone servers and run the server software. On the other hand, the client nodes should be close to their zone servers to decrease the latency of their responsiveness. Thus, we have developed a distributed Dominating Set (DS) computation algorithm called PBS (Priority Based Selection) for selecting the zone server nodes because the DS structure fits well with these requirements. In this sense, an ad hoc network can be considered as a graph and the problem can be mapped into the computation and maintenance of an appropriate Dominating Set of this graph containing the most suitable (most powerful with 'good' position in the network) nodes.

3.1.2 Dominating Set Related Notations, Definitions

For sake of simplicity, we assume symmetrical communication links between the neighboring nodes in the wireless ad hoc network[2]. In this case, the network topology can be modelled as a Unit Disk Graph (UDG) [31] $G(V,E)$, which is defined by a set V of vertices (nodes), and a set E of edges (links). In this graph, there is an edge between two nodes if and only if their distance is at most one (they are in communication range).

A Dominating Set and its different variants can be defined as follows:

Dominating Set (DS): A Dominating Set of a graph G = (V, E) is a subset $S \subseteq V$ of the nodes such that for all nodes $v \in V$, either $v \in S$ or a neighbor u of v is in S. For example, in Figure 3.1 node 4 and 10 form a DS of the given network graph.

Connected Dominating Set (CDS): If the nodes of the subset S form a connected subgraph, S is spanning a Connected Dominating Set.

Minimum Dominating Set (MDS): A Dominating Set is called a Minimum Dominating Set if the number of nodes in S is minimal. Note that finding a Dominating Set of minimum size is an NP-hard problem [32].

Node Weighted Dominating Set (NWDS): If the nodes have weights, a Node Weighted Dominating Set is a subset S of the nodes which is selected based on the node weights and forms a Dominating Set of the graph.

A Dominating Set computation algorithm can be evaluated according to some quality and construction cost factors. The quality factors are:

- **Connected:** This property indicates whether the Dominating Set is connected or not. Note that it is not always a requirement to get a connected DS (it depends on the application for which the DS is built).
- **Approximation:** As already mentioned, to find an MDS is a classical NP-hard optimization problem and we have to use heuristics to approximate it. The approximation factor is defined as the ratio of the computed Dominating Set's size to that of the MDS:

$$\frac{|DS|}{|DS_{MDS}|} = Approximation \tag{3.1}$$

[2]If this is not the case and the links are asymmetrical, the network topology has to be modelled by using directed graph which makes the network topology handling more complex. However, our PBS algorithm works even in this case though it may compute huge Dominating Sets, e.g., selecting all the nodes in the network as dominators.

where $|DS|$ is the number of nodes in the DS and $|DS_{MDS}|$ is the number of nodes in the MDS, respectively.

The construction costs can be characterized by:

- **Time Complexity:** The time complexity is measured in rounds as it is usual in DS computation algorithms. One round consists of sending a message, receiving a message and some local computation. How the rounds are synchronized via the network is an implementation issue. In the implementation of our PBS algorithm, we use periodic 'alive' messages for monitoring the neighborhood of the nodes (this synchronizes the starting time of round execution at the different nodes), and timers to execute the different parts of the round. For more details see Chapter 5 and Appendix B.

- **Message Complexity:** This factor indicates how many messages with what size have to be sent.

3.2 Zone Server Selection

This section describes first the assumptions and requirements for zone server selection in an ad hoc network based on building a Dominating Set. Then we present PBS, our zone server selection algorithm together with some examples.

3.2.1 Assumptions

Our assumptions consider some prerequisites for the network and the nodes themselves. These are the following:

Connected ad hoc network: The ad hoc network consists of several nodes that span a connected network. Thus, any node can communicate with any other node in this network using a routing protocol.

Existing routing functionality: We assume that there is a running routing protocol that provides routing functionality and guarantees that every node knows its neighbors and is able to detect new nodes and nodes that left the network.

Unique node ID: Every node has a unique integer ID[3] that is used as a final tie breaker in the zone server selection procedure, if the expected decision cannot be made based on the other metrics (see later). We assume, that the nodes can get/generate this ID automatically.

Weight assigned to every node: In addition to the node ID, every node has a positive real number as node weight (assigned or generated automatically, see Section 3.3 for our weight computation method), as well. This weight indicates the node's capability to act as zone server and should be based on the available node resources (e.g., CPU power, memory, remaining battery power), the node's mobility and link connectivity (indicating the quality of links that a server can offer to its clients). In general, a laptop is better suited to act as zone server than a mobile phone due to its more powerful resources.

Co-operative behavior: Furthermore, for sake of simplicity we assume co-operative behavior from every node, similarly to offering routing functionality to other nodes, as a social contribution. Thus, every node is assumed to contribute to a service (e.g., can be selected as zone server) even if the node is not participating in the specific service session itself as a client. If this is not the case in a given MANET, the nodes participating in the service form an overlay network, and this overlay can be considered instead of the underlaying physical network. To build this overlay an appropriate neighbor discovery mechanism has to be implemented instead of using the connectivity information gained from the physical network.

3.2.2 Requirements

Our requirements concerning the properties of the DS that should be built and the algorithm that determines the DS are the following:

Node weighted DS: As a first priority, the DS should consist only of nodes that have high weights, hence are better suited to act as zone servers. If there are several nodes with the same weight, those nodes should be chosen that give a good Minimal Dominating Set (MDS) approximation.

The minimum number of nodes in the DS must be at least two: For every network topology, even for fully connected (mesh) graphs, there must

[3]This ID can be used as the node's name/network address, too. However, this is not necessarily the best solution. There are a number of other approaches to handle the naming/addressing issue in mobile ad hoc networks, e.g. see [33] or [27], which can provide also interoperability between different network clouds.

be at least two nodes[4] in the DS for fault-tolerance and redundancy purposes.

Good MDS approximation: With the increasing number of zone servers, the produced overhead as needed for consistency and synchronization mechanisms will increase as well. Therefore, the built DS should have a good approximation of the MDS. But note that choosing nodes with high weights will be prioritized over the requirement to get a good MDS approximation.

Completely distributed: The algorithm has to run locally on every node in a distributed manner because there is no pre-established infrastructure in mobile ad hoc networks that can be used for central administration.

Low time complexity: The time complexity that is measured in rounds is to be kept low.

Low message complexity: The message complexity (number of messages per round and their size) is to be kept also low for saving resources.

Included mobility maintenance: The algorithm has to provide mobility maintenance and react properly to links going down or coming up if nodes are moving around or joining/leaving the service session.

3.2.3 Priority Based Selection Algorithm

Providing support for a zone-based service management architecture and for the selection of zone servers we have developed a DS computation algorithm called PBS (Priority Based Selection) [13–15].

Quite a few DS computation algorithms have been already developed (see Section 7.1.2), however, PBS is the only one, according to our knowledge, that offers continuous maintenance of the DS when the network graph changes dynamically.

The PBS algorithm compares the priorities of the different nodes and chooses the highly prioritized nodes into the DS which will act as zone servers.

Notations, Definitions Used in PBS

In our algorithm, we are using the following notations and definitions:

Status: A node can have one of the following states:

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

[4]Of course, one can chose any higher number here. We have selected two because in this case a minimum level of fault-tolerance can already be achieved.

- 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 as a client (e.g., as a player in the game) and still does not have a DOMINATOR or DOMINATEE status but it is an internal candidate to become one of them.

- EXT_CANDIDATE - The node does not participate in the service as a client but based on the co-operative behavior assumption it is possible that the algorithm chooses the node as DOMINATOR into the DS. Thus, the node runs the service server software (e.g., the game server software) and is considered as an external candidate for becoming a DOMINATOR.

  In the beginning, every node has an INT_CANDIDATE or EXT_CANDIDATE status depending on whether the user of the node wants to participate in the service as a client or not[5]. The task of the algorithm is to put at least every INT_CANDIDATE node of the graph into either DOMINATOR or DOMINATEE status.

Span: The $span(v)$ of a node v is the number of INT_CANDIDATE neighbors (including itself).

Fully connected node: If a node has a link to all other nodes in the network, the node is fully connected.

Leaf node: A node having degree 1 is a leaf node.

Neighborlist: Every node is tracking a neighborlist that contains all relevant information about the node's neighbors.

Coverage: All the nodes that are directly connected to a DOMINATOR node are covered by this node. Every DOMINATEE node is covered by at least one DOMINATOR node.

Dominating Set Computation

Building a DS using the PBS algorithm is based on comparing priorities. A node has a **higher priority** than another node if

[5]Note that we have only INT_CANDIDATE nodes in the graph if our co-operative behavior assumption does not hold and the overlay network of service client nodes has to be considered instead of the physical network (see Section 3.2.1).

1. the node has a higher node weight;
2. if tie: the node has a higher span value;
3. if tie: the node has more neighbors with DOMINATOR status;
4. if tie: the node has a lower node ID.

The node weight indicates the node's capability to act as zone server and having high weighted nodes in the DS has the highest preference. The span value is an indication of how many INT_CANDIDATE nodes will change into DOMINATEE status if this node becomes a DOMINATOR and is used as the second criterion leading to a better MDS approximation. Note that if all nodes have the same node weights and their priority decisions are based on the span value, our algorithm becomes quite similar to a greedy DS computation algorithm [34]. If no decision can be made based on the span values the number of DOMINATOR neighbors will be considered. It is reasonable to have as few hops as possible between the DOMINATOR nodes due to the consistency, synchronization and the according messages exchange mechanisms required by the applications. The final tie breaker for comparing priorities of different nodes is the node ID which is unique within the network. A node with lower node ID has higher priority.

The choice of the criteria and their order to compare the priorities of different nodes are based on the discussions in the previous subsections. They are influencing the quality and property of the DS built by the PBS algorithm. An advantage of our algorithm is that the Dominating Set to be built can be influenced purely changing the criteria and their order. For example, if it is important to get a connected DS fast the criterion about the number of DOMINATOR neighbors (Criterion 3) can be put at the top, and the algorithm constructs a connected DS. In this case, no compulsory nodes with the highest weights will be chosen into the DS.

The logic of the PBS algorithm is illustrated in Figure 3.2. The black boxes contain the conditions for the different branches, and the grey boxes indicate the status of the node at the given point. Moreover, Listing 3.1 shows the pseudo code of the algorithm.

PBS is performed in rounds. Every round consists of three steps, such as sending the own neighborlist to the neighbors, receiving neighborlists from the neighbors and recalculating the own status. Based on the neighborlist exchange, every node has knowledge about the network topology up to a distance of 2 hops and is therefore able to determine the nodes with the highest

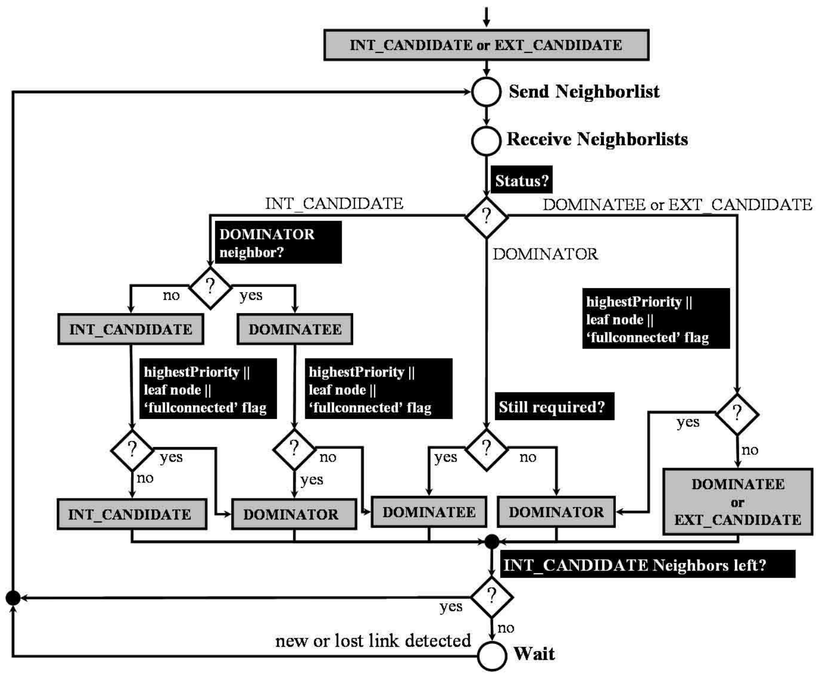
INT_CANDIDATE or EXT_CANDIDATE
Send Neighborlist
Receive Neighborlists
Status?
INT_CANDIDATE
DOMINATEE or EXT_CANDIDATE
DOMINATOR
DOMINATOR neighbor?
no
yes
INT_CANDIDATE
DOMINATEE
highestPriority || leaf node || 'fullconnected' flag
highestPriority || leaf node || 'fullconnected' flag
highestPriority || leaf node || 'fullconnected' flag
Still required?
yes
no
INT_CANDIDATE
DOMINATOR
DOMINATEE
DOMINATOR
DOMINATEE or EXT_CANDIDATE
INT_CANDIDATE Neighbors left?
yes
no
new or lost link detected
Wait

Figure 3.2: *Flow Chart of the PBS Algorithm*

priorities. In the beginning, the rounds are performed as long as there are any INT_CANDIDATE neighbors left within the distance of 2 hops. Afterwards, a node starts a round again, if it detects lost or new links to some neighboring nodes. Every node set its initial status to EXT_CANDIDATE when it joins the ad hoc network. If the node wants to join a service session, it changes its status to INT_CANDIDATE and starts sending its neighborlist.

After this, a round of PBS can be summarized as follows:

Step 1: Send neighborlist to all neighbors;

Step 2: Receive neighborlist from all neighbors;

Step 3: Determine status:

- If the node's status is INT_CANDIDATE:
  - change status to DOMINATEE if at least one neighbor has DOMINATOR status (`isNeighbor(DOMINATOR)==true`).

  - change status to DOMINATOR if
    * the node has the highest priority among its 1-hop INT_CANDIDATE neighbors and the neighbors of these INT_CANDIDATE nodes which are two hop away from the original node (`highestPriority`);
    * the node has a leaf INT_CANDIDATE neighbor, i.e., an INT_CANDIDATE neighbor node with degree 1 (`haveLeafNeighbor`);
    * a DOMINATOR neighbor reported fully connected status (`fullconnected_reported`).

    If the node is fully connected and changed to DOMINATOR, check the number of other DOMINATOR nodes among its neighbors. If there is not any other DOMINATOR node set the 'full-connected' flag in the neighborlist that is sent to the node with the highest priority among its 1-hop neighbors (`report_fullconneced(v)`).

- If the node's status is DOMINATEE or EXT_CANDIDATE:
  - change status to DOMINATOR if
    * the node has the highest priority among its 1-hop INT_CANDIDATE neighbors and their two hop neighbors (`highestPriority`);
    * the node has a leaf INT_CANDIDATE neighbor (`haveLeafNeighbor`);
    * at least one DOMINATOR neighbor reported fully connected status (`fullconnected_reported`).

    If the node is fully connected and changed to DOMINATOR, check the number of other DOMINATOR nodes among its neighbors. If there are not any other DOMINATOR node set the 'full-connected' flag in the neighborlist that is sent to the node with the highest priority among its 1-hop neighbors (`report_fullconneced(v)`).

- If the node's status is DOMINATOR:
  - change status back to DOMINATEE if there is another DOMINATOR node with lower node ID covering the same set of DOMINATEE nodes (`!is_still_required`).

```
myStatus = INT_CANDIDATE or EXT_CANDIDATE
do until (all_neighbors_determined) {
    send_neighborlist;      // Step 1
    receive_neighborlist;   // Step 2
    // Step 3
    if(myStatus==INT_CANDIDATE) {
        if(isNeighbor(DOMINATOR)==true) {
            myStatus=DOMINATEE;
            updateCommonNeighbors;
        }
        if(highestPriority ||
            haveLeafNeighbor ||
            fullconnected_reported) {
            myStatus=DOMINATOR;
            if(fullconnected) {
                v=neighborWithHighestPriority;
                report_fullconnected(v);
            }
        }
    } else if(myStatus==EXT_CANDIDATE ||
                myStatus==DOMINATEE) {
                if(highestPriority ||
                    haveLeafNeighbor ||
                    fullconnected_reported) {
                    myStatus=DOMINATOR;
                    if(fullconnected) {
                        v=neighborWithHighestPriority;
                        report_fullconnected(v);
                    }
                }
    } else if(myStatus==DOMINATOR) {
                if(!is_still_required) {
                    myStatus=DOMINATEE;
                }
    }
}
```

Listing 3.1: *Pseudo Code of the PBS Algorithm*

Special Cases

A node will change to DOMINATOR not only if it has the highest priority, but also if it has a leaf INT_CANDIDATE neighbor (because this node is the only choice to cover the leaf node), or a neighboring DOMINATOR set a ‘fullconnected’ flag in its sent neighborlist. This flag is used to fulfill the requirement of having always minimum two nodes in the DS regardless of the network topology (in small, mesh networks this requirement would not be met just using a general DS building algorithm). For example, consider the simple network topology shown in Figure 3.3 consisting of four nodes. The numbers inside the circles represent the node IDs since the numbers outside indicate the different node weights. All the nodes have INT_CANDIDATE status.

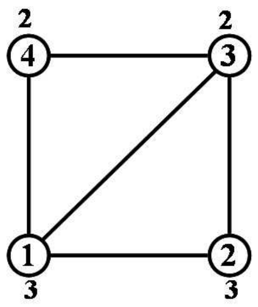

Figure 3.3: *Graph with 2 Fully Connected Nodes*

Based on the node weight and on the higher span value node 1 will change its status to DOMINATOR in the first round of the algorithm. In the second round, all other nodes change to DOMINATEE and without the requirement of having a minimum number of two zone servers the DS building algorithm would stop. A node can easily detect whether it is fully connected or not by receiving the neighborlist from the neighboring nodes. If none of these nodes has a neighbor that is not already an own neighbor, then the node is fully connected. Being such a fully connected node has one big advantage because this node knows the whole network topology. Therefore, if a fully connected node is selected as DOMINATOR it checks its neighbors whether there are other DOMINATOR nodes among them. If not it determines the node with the highest priority among its neighbors and sends in the next round the neighborlist with the flag ‘fullconnected’ that forces the selected neighbor node to join the DS, as well. In the considered example in Figure 3.3, after switching to DOMINATOR in round 1 node 1 set in round 2 the ‘fullconnected’ flag to node 2, because it is the node with the highest priority among the neighbors of node 1. Thus, node 2 will also switch to DOMINATOR in the next round forming a DS with 2 nodes as shown in Figure 3.4.

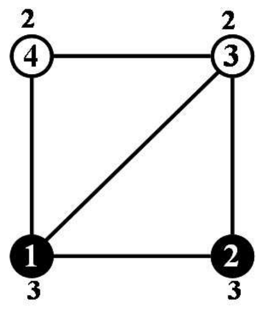

Figure 3.4: *Chosen DS for the Graph with 2 Fully Connected Nodes*

Mobility Maintenance in PBS

An exclusive advantage of the PBS algorithm is that it can react to node mobility. If a node leaves or joins the network, or if it simply moves around, this leads always to changes in the status of some links (some links are going down or coming up). If a node detects new or lost links it will start a new round of the algorithm to distribute the information about the link changes and to determine possible new INT_CANDIDATE nodes. Note that a DOMINATEE node that lost a connection will check first whether there is a neighboring DOMINATOR node left. If not it switches its status back to INT_CANDIDATE before exchanging the neighborlists (if required for the application, the node may connect to the closest DOMINATOR node known from its neighbors for the recomputation period). Similarly, a DOMINATOR node checks first whether there are DOMINATEE neighbors left, and will change back to DOMINATEE if only DOMINATOR but no DOMINATEE nodes are left in the neighborhood.

DS Computation Example Using PBS

In Figure 3.5, a general graph is shown. The numbers inside the circles represent the node ID and the numbers outside the node weight. We assume that in the beginning all nodes but node 7 and 9 want to join the service session. Thus, every node has INT_CANDIDATE status except node 7 and 9 that have EXT_CANDIDATE status. Note that in the figures INT_CANDIDATE nodes are white, DOMINATEE nodes are grey, and DOMINATOR nodes are black colored. EXT_CANDIDATE nodes have grey dashed circle. The following actions will take place using the PBS algorithm:

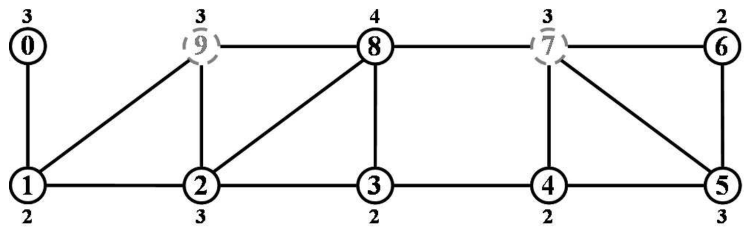

Figure 3.5: *Example Graph*

Round 1: All nodes send their neighborlist to all neighbors. Based on this list node 8 will change to DOMINATOR because it has the highest node weight among its neighbors and therefore the highest priority. Similarly, node 1 changes to DOMINATOR because its neighbor, node 0 is a leaf node in the graph and can only be covered by node 1.

Round 2: Node 1 and 8 inform their neighbors via sending neighborlist that they changed to DOMINATOR. Upon receiving this message all neighboring INT_CANDIDATE nodes (node 0, 2 and 3) change to DOMINATEE. This situation is outlined in Figure 3.6.

Round 3: The neighborlists will be exchanged again. Based on this, node 5 gets now the information that node 8 changed to DOMINATOR. The node with the highest priority that can cover the remaining INT_CANDIDATE nodes has to be determined once more. Node 7 and 5 have now the same highest node weight, and the same span value. But because node 7 has 1 DOMINATOR neighbor it has higher priority and will change to DOMINATOR. Note that node 7 is actually not participating in the service session, but due to the co-operative behavior assumption it still can act as zone server.

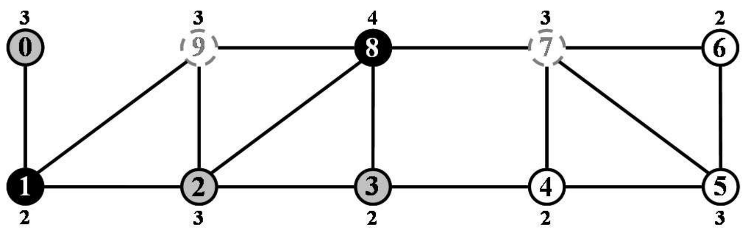

Figure 3.6: *Situation After 2 Rounds*

Round 4: Nodes 4, 5 and 6 get now the information that node 7 changed to DOMINATOR and they will switch to DOMINATEE.

As long as the nodes do not move around the DS is built and, as outlined in Figure 3.7, nodes 1, 7 and 8 will act as zone servers in the given network.

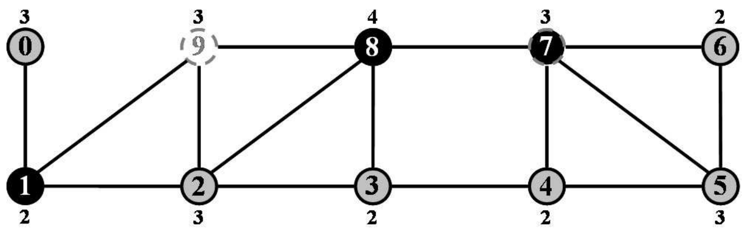

Figure 3.7: *Chosen DS by the PBS Algorithm*

3.2.4 Analysis of the Priority Based Selection Algorithm

In the following, we analyze the correctness and performance of the PBS algorithm based on the requirements we defined in Section 3.2.2.

Node weighted DS: PBS compares priorities to build the DS and the first criterion to get a high priority is the weight of the node. Therefore, only nodes with high weights will be chosen into the DS.

The minimum number of nodes in the DS must be at least two: It is guaranteed that there are at least two DOMINATOR nodes for any network topology (of course, the network should consist of at least two nodes). If there is a fully connected node, this node will force another node to turn to DOMINATOR status if required. If there is no fully connected node in the graph with n nodes, the maximum possible degree for a node is $n-2$. This means, that if a node changes to DOMINATOR status, there is minimum one node left that will not be covered by this node, and a second DOMINATOR node will be determined by the algorithm.

Good MDS approximation: The approximation factor of the PBS algorithm is strongly influenced by the distribution of the node weights. In the worst case, the nodes with the lowest span values will have the highest node weights deteriorating the approximation factor. If all nodes have the same weight or the nodes with the highest span values also have the highest weights, the span value will determine the priority between the different nodes. This is similar to a greedy algorithm [34] and the approximation factor at the beginning of the algorithm is $log\Delta$, where Δ is the maximum node degree in the network graph. During the service session due to node mobility the approximation factor can change and become worse, because the algorithm tries to keep the existing DS even if there could be a better MDS approximation achieved. However, we consider zone server changes and the required transfer of the service states as a more expensive solution than having some additional servers.

Completely distributed: The decisions of a node to turn to DOMINATOR or DOMINATEE status are only based on the received information by exchanging the neighborlists with the direct neighbors. Thus, the algorithm is completely distributed and runs locally on every node.

Low time complexity: The required number of rounds to determine the DS can be rapidly reduced if as many nodes as possible can change from INT_CANDIDATE status to either DOMINATOR or DOMINATEE status per round. However, there are some worst case scenarios where the higher priority can only be determined by comparing the node IDs. This means that the DOMINATOR nodes have to be chosen step by step. For example, in Figure 3.8 a fragment of a network graph is shown, where all nodes have the same node weight and all edges are identical. At the beginning, all nodes have INT_CANDIDATE status. The white numbers in the black boxes show the current span values of the nodes. In this case, there is a chain of nodes whose priorities can be compared only based on the unique node IDs.

In the first round, node 2 declares itself as a DOMINATOR because it has the lowest node ID among the nodes with span value 3. In the second round, node 1 and 3 change to DOMINATEE. In the third round, node 4 gets the information about the new status of node 3. Node 5 recognizes the situation as shown in Figure 3.9 only in the fourth round and changes itself to

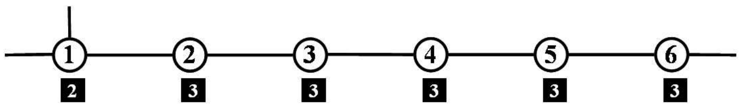

Figure 3.8: *Worst Case Scenario Concerning Time Complexity*

DOMINATOR because it has the lowest node ID among the remaining nodes with span value 3.

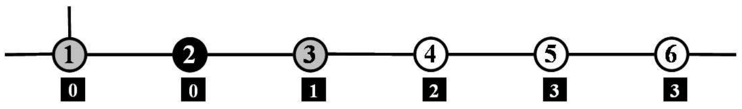

Figure 3.9: *Situation in the Worst Case Scenario After 3 Rounds*

This means that before node 5 as the next DOMINATOR can be determined 3 rounds have been required. During this time 3 nodes changed their status from INT_CANDIDATE to either DOMINATOR or DOMINATEE. Therefore, in the worst case when a graph consists only of such fragments shown in Figure 3.8, the PBS algorithm can be upper bounded by $O(n)$ rounds.

In general, if

$$DEG = \{deg(v_1), ..., deg(v_n)\} \tag{3.2}$$

is the set of the different node degrees in a graph, and

$$\delta = min(DEG \backslash \{deg(v_i) | deg(v_i) = 1\}) \tag{3.3}$$

the minimum degree (excluding degree 1 for leaf nodes), and it is assumed that in the worst case

$$\forall v_i \in V, \qquad deg(v_i) = \delta \tag{3.4}$$

then it is possible to change $\frac{\delta+1}{3}$ nodes from the INT_CANDIDATE status to either DOMINATOR or DOMINATEE in one round. Therefore, in total

$$\frac{n}{\frac{\delta+1}{3}} = \frac{3}{\delta+1}n \tag{3.5}$$

rounds are needed for determining all nodes and the time complexity can be upper bounded by $O(n)$.

The required time for building the DS is influenced by the way the node weights are assigned and the span values that the nodes with high weight have. In the best case, already in the first round DOMINATOR nodes are determined that cover all remaining nodes. Then the PBS algorithm requires only 3 rounds to determine the status of all the nodes: In the first round, the nodes exchange their neighborlists and some of the nodes change to DOMINATOR. In the second round, the remaining nodes change to DOMINATEE and finally, in the third round every node recognizes that all nodes changed to either DOMINATEE or DOMINATOR. Therefore, the time complexity can be lower bounded by $\Omega(3)$.

Low message complexity: In general, every node sends a neighborlist to its neighbors and every node can have maximum Δ neighbors. Therefore, per round Δn messages will be sent, and the message complexity can be upper bounded by $O(\Delta n)$. Note that this bound can be drastically reduced, if the different nodes are connected via a broadcast medium like in case of WLAN [5]. In such a medium, all nodes that are in the transmission range of a certain node can receive all messages from this node. Because a node sends the same neighborlists to all its neighbors, it is enough to send the message once to the broadcast address with the TTL (Time To Live) field set to one. For a broadcast medium the message complexity can be upper bounded by $O(n)$ messages that need to be sent per round. In general, only the nodes that detected new or lost links and their neighbors up to a distance of 2 hops will exchange neighborlists. The upper bounds will only be achieved at the beginning of the algorithm and if all nodes need to update their neighborlists due to mobility maintenance.

The size of a message is determined by the number of entries in the neighborlist. A neighborlist contains for all neighbors and the node itself an entry and its size can be upper bounded by $O(k(\Delta+1))$ bytes, whereas k represents the size of an entry in the neighborlist in bytes. The lower bound of a message size is $\Omega(k(\delta+1))$. These bounds can be decreased as well, if only incremental updates are sent instead of sending always the full neighborlist.

Mobility maintenance: The algorithm can react to links that are going down or coming up. If a node detects new or lost links it will start new rounds of the algorithm to distribute the information about the link changes and to determine the status of possible new INT_CANDIDATE nodes.

Summary of the PBS Algorithm's Analysis

Table 3.1 gives a summary of the PBS algorithm's analysis. In the table, n indicates the number of nodes, Δ the maximum degree in the network graph, and k is the size of a neighborlist entry (in bytes).

Property	**Value**
MDS Approximation	$O(log\Delta)$
Time Complexity [round]	Upper bound: $O(n)$ Lower bound: $\Omega(3)$
Message Complexity	
Messages per round	$O(n)$
Message size per round [byte]	Upper bound: $O(k(\Delta+1))$ Lower bound: $\Omega(k(\delta+1))$
Connected DS	No (optional)

Table 3.1: *Summary of the PBS Algorithm's Analysis*

3.3 Node Weight Computation

In this section, after describing the assumptions and requirements concerning our NWC mechanism we present NWC what we have developed to compute node weights being used in the PBS algorithm's node priority comparison.

3.3.1 Assumptions and Requirements

Besides the assumptions and requirements we defined in Section 3.2.1 and Section 3.2.2 we consider some additional prerequisites for the weight computation:

Local resource/parameter values: Every node is able to extract/collect all the necessary information related to its local resource parameters, such as CPU, memory, battery capacity and load.

Link quality information: The nodes are able to collect link quality information of their links from their network interface devices measuring the links' SNR (Signal to Noise Ratio) values.

Service profile: In case of various service types, different importance levels (parameter weights[6]) are to be assigned to the same parameter set, which is called service profile. Since this profile is created by the service developer, a mechanism has to be developed aiding the service developers in this procedure. Moreover, we assume that the service description document [21] used in SIRAMON has an entry with the service profile.

3.3.2 Node Weight Computation Mechanism

To provide a systematic way for computing node weights to be used in the PBS algorithm, we have developed a mechanism called NWC (Node Weight Computation) [16].

The few mechanisms existing today (see Section 7.1.3) use only a simplified heuristic to classify/weight the nodes in the network. In these mechanisms, just one or very few node properties are taken into account aiming to achieve a specific objective (e.g., compute a good MDS approximation or minimize energy consumption).

However, different services have different requirements. In NWC, several node properties/parameters are used to accomplish the node weight computation. In case of each service type, the most important properties from the viewpoint of the service are considered with high parameter weights resulting in the best suited DS selection for the given service.

Node Parameters

The parameters of the node reflect the node's capabilities to act as zone server. In NWC, we consider the following five parameters grouped into three classes:

Processing Power:

CPU - It represents the available CPU capacity of the node. This can be important for computation intensive services.

Memory - It represents the available memory capacity of the node. This also can be important for computation intensive services.

[6]Note that the parameter weight and the node weight are different things. NWC computes the node weight as the linear combination of the node parameters. In this computation, every parameter is weighted by its own parameter weight extracted from the service profile.

Energy:

Battery - Represents the available energy level of the node. This parameter allows the selection of nodes with long life time.

Connectivity:

Link quality - The link quality parameter represents the quality of the wireless connections between the node and its neighbors. This parameter can be important for communication intensive services.

Position - The position parameter is related to the number of neighbors the node has (node degree). This reflects the node position in the network.

The parameter values are computed in the following way:

$$P_{CPU}(t) = 1 - load_{CPU}(t), \tag{3.6}$$

where $load_{CPU}(t)$ is the actual fraction (between 0 and 1) of the node's CPU load. The *CPU* parameter value is time dependent and nodes with higher CPU load have lower value of this parameter.

$$P_{mem}(t) = 1 - \frac{load_{mem}(t)}{MAX_{mem}}, \tag{3.7}$$

where MAX_{mem} is the maximum memory capacity, while $load_{mem}(t)$ is the actual memory load in MByte on the node, respectively. The *memory* parameter value is also time dependent and higher memory load on the node results in lower parameter value.

$$P_{bat}(t) = 1 - load_{bat}(t), \tag{3.8}$$

where $load_{bat}(t)$ is the actual fraction (between 0 and 1) of the node's battery load. The *battery* parameter value is time dependent and higher battery load results in lower parameter value giving an indication about the remaining battery power. Note that this way of computation does not take into account the speed of the node's energy consumption. However, when the energy level of the node's battery is below a certain threshold, the node may/must be excluded from the zone server selection process.

$$P_{link}(t) = \sum_{i=1}^{N} \frac{SNR_{link\_i}(t)}{MAX_{SNR_{link\_i}}} \times \frac{1}{N}, \tag{3.9}$$

where $MAX_{SNR_{link\_i}}$ is the maximum and $SNR_{link\_i}(t)$ is the actual Signal to Noise Ratio value of $link\_i$ of the node, respectively. Moreover, N is the number of the node's neighbors. The *link quality* parameter is time dependent and directly proportional to the measured SNR values of the node's links.

$$P_{pos}(t) = 1 - \frac{1}{node\_deg(t)}, \tag{3.10}$$

where $node\_deg(t)$ is the actual degree of the node. The *position* parameter value is time dependent and nodes with higher node degree have higher parameter value.

As we can see, all these parameter values are normalized and fall in the range of [0..1]. This is convenient to compute a normalized value also for the node weight which makes it possible to compare these weights easily.

NWC Algorithm

When the node weight is required the node does the following procedure:

Step 1: Collect the CPU load, memory load and battery energy level and compute the parameter values $P_{CPU}(t)$, $P_{mem}(t)$ and $P_{bat}(t)$;

Step 2: Get the list of neighbors;

Step 3: Compute the link quality parameter value $P_{link}(t)$;

Step 4: Compute the position parameter value $P_{pos}(t)$;

Step 5: Collect the parameter weights $W_{CPU}(serv)$, $W_{mem}(serv)$, $W_{bat}(serv)$, $W_{link}(serv)$, $W_{pos}(serv)$ from the service profile;

Step 6: Compute the node weight based on the following equation:

$$\begin{aligned} W_{node\_x}(t) &= \frac{W_{CPU}(serv)P_{CPU}(t)}{NUM_{param}} + \frac{W_{mem}(serv)P_{mem}(t)}{NUM_{param}} + \\ &+ \frac{W_{bat}(serv)P_{bat}(t)}{NUM_{param}} + \frac{W_{link}(serv)P_{link}(t)}{NUM_{param}} + \\ &+ \frac{W_{pos}(serv)P_{pos}(t)}{NUM_{param}}, \qquad (3.11) \end{aligned}$$

where NUM_{param} is the number of the parameters which is 5 in our case.

Note that with this computation we get a normalized node weight value falling in the range of [0..1] because the parameter values and the parameter weights are also normalized (see Figure 3.10). Node weight 1 means that the node is powerful, since node weight 0 means that the node is not powerful enough to act as a zone server.

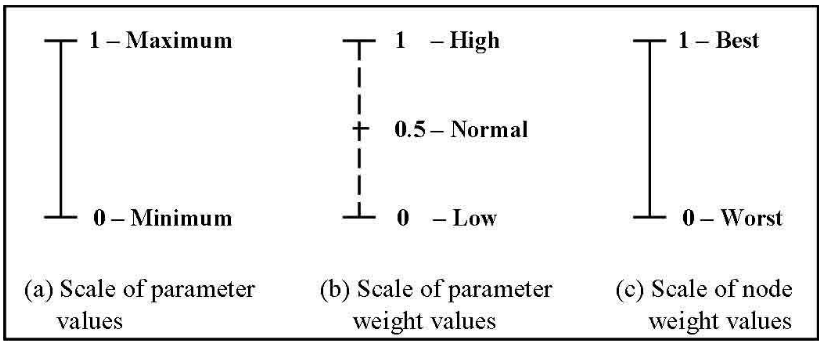
1 – Maximum
0 – Minimum
(a) Scale of parameter values
1 – High
0.5 – Normal
0 – Low
(b) Scale of parameter weight values
1 – Best
0 – Worst
(c) Scale of node weight values

Figure 3.10: *Value Scales in NWC*

When a node is selected as a zone server, its resources are monitored to assure a minimal performance for the service. Such monitor runs periodically and checks the three local resource parameter values (CPU, memory and battery) whether they drop below a certain threshold in which case an anomaly is generated (see Section 6.3.4 for simulation results). The threshold values we used are given in Table 3.2.

Parameter	Threshold
CPU	10 %
Memory	10 %
Battery	5 %

Table 3.2: *Threshold Values for Local Resource Parameters in NWC*

Service Profile

As we mentioned above, different importance levels and thus different parameter weights are to be assigned to the same parameter set in case of various service types. These parameter weights are collected in the service profile created by the service developer. The approach we have taken here consists of the definition of *static* service profiles. Hence, each service developer assigns the parameter weights in advance for the given service. Then this profile is used by all the nodes for the node weight computation.

Note that with such a static approach it is not possible to accurately predict the future service context (e.g., network scenario, mobility behavior of the nodes) in advance where the given service will be deployed. Thus, even knowing the basic characteristics of the service an optimal profile (parameter weight assignment) is not granted for every situation. To achieve this, a more complex dynamic approach should be developed which remains as future work.

To define the service profile and thus assign the parameter weights we have developed a mechanism using factorial design [35]. It is based on simulation and presented in Section 6.3. Note that the parameter weights can also be set directly according to some simple objective(s) to be achieved. For example, if the aim is to get the best MDS approximation in the zone server selection, the node degree parameter weight has to be set to 1 and all the other parameter weights to 0.

3.4 Chapter Summary

In this chapter, we have discussed the centralized client/server and the distributed peer-to-peer architecture used today for implementing service management functions in traditional networks and introduced the zone-based architecture for being used in mobile ad hoc networks. Moreover, we have presented our PBS algorithm developed for zone server selection and our NWC mechanism to compute the node weights used in PBS.

PBS computes a Dominating Set and also takes care of the continuous maintenance of this DS. Besides analyzing the basic properties of PBS in this chapter we give its detailed, simulation based performance analysis later in Section 6.2. Moreover, for node weight computation our NWC mechanism is applied which computes the node weight as the linear combination of the node parameters. In this computation, every parameter is weighted by its own parameter weight extracted from the service profile. To define this profile and assign the parameter weights we have developed another mechanism using factorial design. It is based on simulation and presented later in Section 6.3.

Chapter 4

Mobility Prediction in Mobile Ad Hoc Networks

To increase the stability of the selected server set taking node mobility into account is indispensable. High mobility of the nodes can result in the selection of unstable zone servers leading to frequent changes of the server nodes and thus frequent handovers of clients between them. This may also cause high traffic overhead or even service disruption. In this chapter, first we survey the fundamentals of mobility prediction which can be exploited in increasing the selected zone server set's stability. Then, we present our link quality prediction mechanism called XCoPred we have developed for mobility prediction in MANETs. And finally, we point out how XCoPred can be used in the PBS algorithm.

4.1 Fundamentals

Using mobility prediction can help increase the stability of the selected server set by predicting future changes of the network topology and using this information in server selection. In the following, we point out the relation between link quality and mobility prediction, define the problem of mobility prediction in MANETs and discuss the basic structure of a prediction algorithm.

4.1.1 Link Quality as Measure of Mobility State

One of the main reasons of link quality variations in MANETs, besides changes in the environment, is node mobility. Thus, predicting the future link quality of a node is strongly related to predict the movements of the nodes which is generally the quest of *mobility prediction*. Therefore, we heavily use the concept of mobility prediction in order to motivate and explain our link quality prediction work. Moreover, it is possible to predict future network topology changes merely from connectivity information and its variations, e.g., by applying the approach of Multidimensional Scaling (MDS) [19]. Thus, our link quality prediction method can be exploited for topology prediction, too.

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 but also in MANETs to improve routing (see Section 7.2). In cellular networks, estimating the future node position helps predicting handover of the node from one cell to the next and can be used to reserve resources and speed up the handover process. However, the application domain of cellular networks operates with vastly different prerequisites for mobility prediction than in case of ad hoc networks, as the network structure, the hardware of which the networks are built and the behavior of the nodes are fundamentally different. But still the prediction problem is the same, whether considered in wireless networks with fixed infrastructure or in MANETs.

Definition of Mobility Prediction in MANETs

Our assumptions for mobility prediction in MANETs are the following:

1. The current mobility states of the nodes can be observed;
2. The behavior of the nodes are determinable and show some pattern;
3. There is no a priori information about the geographic environment of the network;
4. Each node predicts the future state of its neighborhood in a distributed manner.

Considering the first assumption, there exist a broad variety 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 the node. The choice of the parameter is usually restricted by the structure and the capabilities of the network and the nodes. The second assumption is also essential, because if the nodes behave completely random it is impossible to predict their future status. The pattern allows to map the past and current behavior of the node to its future state. The third assumption reflects our intention to avoid the use of any external hardware or infrastructure in the prediction method. And the last assumption complies with the lack of central coordination property of MANETs.

After this, the general definition of mobility prediction can be formulated in the following way:

$$Pred : Mob\_State_{past,current} \mapsto Mob\_State_{future} \tag{4.1}$$

The lack of a positioning device and having no information about the ad hoc network's physical environment restrict the choice of state observation parameters. We selected link quality as a measure of the node's mobility state because it is easy to monitor without any special hardware or external information and it reflects well the node's connectivity. Moreover, if the absolute geographical position of the node is not important, which is usually the case in MANETs, monitoring the signal quality is a better choice than using physical positions and derive connectivity information from that. The reason for this is because it is hard to find a one-to-one mapping between distance and signal quality due to noise, interference, fading, etc.

To predict the future network topology in MANETs it is enough to know the communication channel (signal) quality and its variation over time between each node pair in the network, if we assume similar behavior of the nodes in the future. Thus, taking the past time series of link quality measurements as training data for regularly observed patterns we can predict probable future connectivity of the node. Figure 4.1 illustrates this with a hypothetical network consisting of five nodes from Node A to Node E[7]. The different graphs depict the signal quality variations in the past between the given node pairs (solid lines) and the predicted future signal quality (dashed lines). Note that tracking signal quality between every node pair has scalability issues and it is practically unfeasible. However, usually it is enough to monitor and predict the signal variation only in the close neighborhood of the node because far nodes are out of communication range anyway. Hence, each node is

[7]For sake of simplicity, we depicted the signal variation between only a few node pairs.

supposed to measure the observed link quality to its neighbors and base the prediction on these observations from the past. Adopting the general form of Equation 4.1, the specific problem with link quality as the measure of mobility state leads to the following equation:

$$Pred : SigQ_{past,current} \mapsto SigQ_{future} \tag{4.2}$$

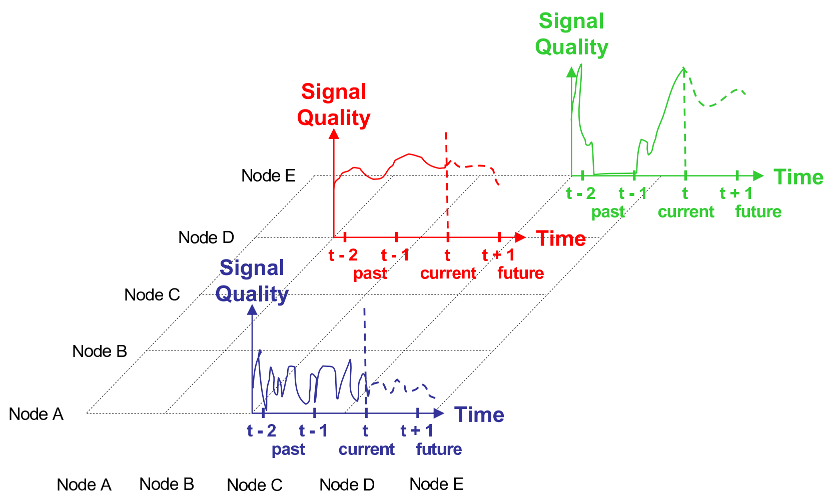

Figure 4.1: *'Trajectory' of Signal Quality Versus Time in a Hypothetical Network*

4.1.2 Structure of a Prediction Algorithm

With these prerequisites in mind, the general building blocks of a prediction algorithm can be defined as shown in Figure 4.2. The two major parts are the *state observation* and the *prediction*.

The task of state observation is to keep track of the node's mobility state by the Observer. 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 prediction part, the Observer has to make a mapping of the parameters. For instance, if the input is an RSS measurement

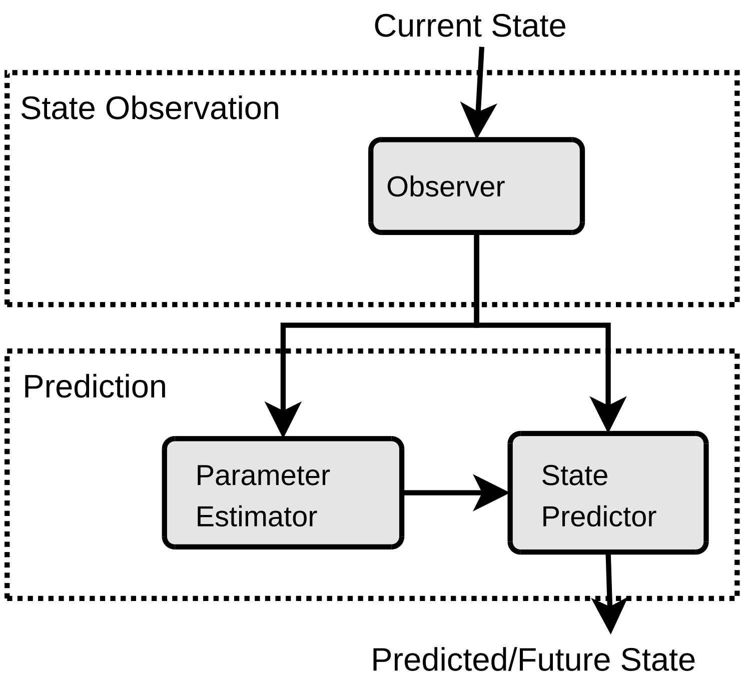

Figure 4.2: *General Structure of a Prediction Algorithm*

and the output is a time series of RSS measurements, the Observer has to store the past values and arrange them in a time series which it can pass to the prediction part.

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

4.2 Link Quality Prediction Using Pattern Matching

In this section, we present our link quality prediction method called XCo-Pred [17, 18] which is based on pattern matching. As stated in Assumption 2 in the previous section, we assume that the behavior of the nodes shows some patterns. There are two driving forces which determine how these patterns look. The first is that the possible patterns are heavily influenced by the physical environment. For instance, in an office building, people carrying the nodes are usually walking along the floors (e.g., going to the printer to fetch a document, walking to the coffee corner and returning back to their office). Another physical restriction is that in this environment humans are rarely moving with higher speed than about 2 m/s. The second driving force is the intention of the user. For example, when the user passes an office door (s)he decides based on intentions, whether (s)he wants to enter the office or continue walking down the floor. These driving forces of node mobility are reflected in the observed patterns of link quality.

Assuming that the behavior of the nodes is repetitive, patterns similar to the recent past in the history of the link measurements contain information about the node's current mobility state. Thus, these past situations can be used as the base for the prediction. Therefore, the quest is to find these similar patterns in the past.

4.2.1 State Observation

As the measure of link quality, we are using the Signal to Noise Ratio (SNR) value of the link. The node periodically monitors the SNR values of its links and these values serve as input for the Observer. Figure 4.3 shows an example of a time series of a link's SNR values with recognizable patterns. The samples were measured in a MANET of a typical office environment with two nodes moving around on the floor and pausing in the offices.

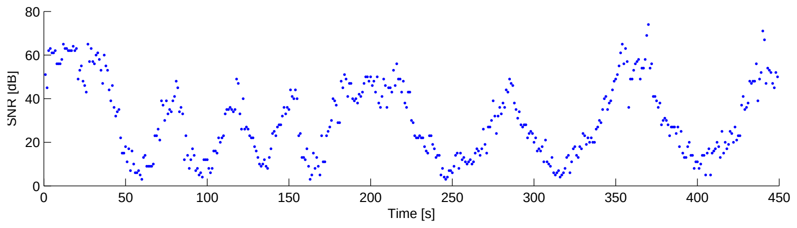

Figure 4.3: *SNR Measurement Example*

Signal to Noise Ratio

We have chosen to use the SNR value as a measure of link quality because the SNR not only takes into account the signal strength, but also the amount of noise in the signal. Moreover, it can be easily measured. For instance, in wireless LAN network interfaces according to the 802.11 standard [36] the firmware and the driver usually provide some measurements of signal strength and background noise observed on the channel.

The Signal to Noise Ratio is generally defined as

$$SNR[dB] = 10log_{10}(\frac{P_{signal}}{P_{noise}})[dB], \tag{4.3}$$

where P_{signal} is the power level of the signal and P_{noise} is the power level of noise, respectively. 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 modelled with a freespace model. It is further influenced mainly by three effects, such as reflection, scattering and diffraction. Together, all these physical factors are building the radio propagation model. On the other hand, the noise power is usually modelled as receiver noise (e.g., thermal noise in the receiver modelled as Additive White Gaussian Noise - AWGN), environmental noise (from different sources in the environment, usually also modelled as AWGN) and interference caused by other transmissions on the same channel or nearby, overlapping channels. For more details, see [37].

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

Definition 4.2.1 *The **measurement interval T** is the time between two sequent measurements performed on one link of the node.*

T is a design parameter and was chosen to be 1 sec in our case. It is a trade-off between having too frequent measurements which increases the computational costs and having too few measurements which means loss of information about the node's behavior.

In order to account for breaking links, the special value of $SNR = 0$ is defined as having no connection between the nodes. Each node keeps the history of the according link and, in case of link failure, fills it with zeros to have the information when and for how long the connection broke. Having this information is important to predict future link failures.

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. N is another design parameter, which has to be chosen as a trade-off between memory usage 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 allows to store 2048 measurements per link. With $T = 1$ sec this results in storing roughly the last 35 minute history of each link. Counting 10 connections per node, this requires approximately 80 kBytes of memory. Even the most limited devices should nowadays have this amount of free memory available, so $N = 2048$ seems a reasonable choice.

When the Observer gets a prediction request, it hands over two things to the prediction algorithm: the training data and the query.

Definition 4.2.2 *The **training data t** is the measured time series of all the node links from the past in the time range $t = (n - NT) \dots n$, where n is the **query time**.*

Definition 4.2.3 *The **query q** is the recent part of the time series of measurements, which is used for creating the link model. The **query order o** is the length of the query, i.e., the number of measurements used in the query.*

In Equation 4.2 the training data is represented by $SigQ_{past}$ and the query is $SigQ_{current}$. Figure 4.4(a) visualizes an example training data gathered on a link, query and query order.

The query order o is another important design parameter. We discuss the choice of o in the evaluation part (see Section 6.4.3) of this thesis.

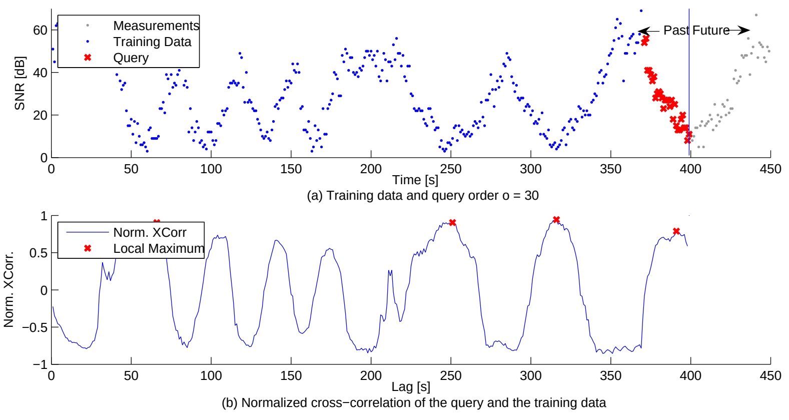

Figure 4.4: *Sample Training Data and Query*

Smoothing with Kalman Filter

One problem with this approach is, that the measurements of the training data are usually noisy. In order to get rid of this noise as far as possible, we filter the measurements with a Kalman filter [38]. Figure 4.5 shows a time series of SNR measurements from an exemplary link simulated in NS-2 [20], together with the filtered values.

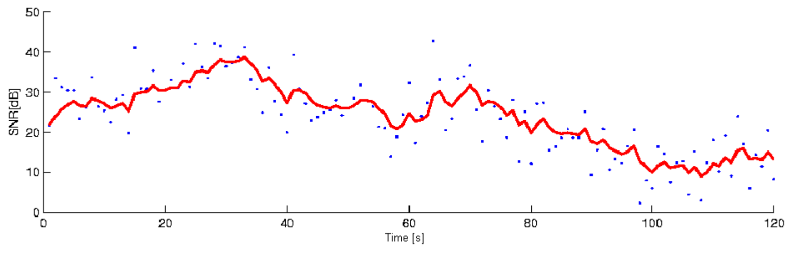

Figure 4.5: *SNR Time Series Filtered with Kalman Filter*

The basic idea behind the Kalman filter is essentially to estimate the state $\mathbf{x_k} \in \mathbf{R}^n$ of a system at time k, which can be observed only indirectly or in-accurately, as the linear combination of a prediction, based on the last value, and the measured value. Systems which can be filtered with a Kalman filter are described with the following equations:

$$\mathbf{x_k} = A\mathbf{x_{k-1}} + B\mathbf{u_{k-1}} + \mathbf{w_k}, \tag{4.4}$$

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

$$\mathbf{z_k} = H\mathbf{x_k} + \mathbf{v_k}. \tag{4.5}$$

A is the $n \times n$ system matrix, which is determined by the nature of the system. B is an $n \times b$ matrix, that relates the optional control input $\mathbf{u_k} \in \mathbf{R}^b$ to the state $\mathbf{x_k}$. The $m \times n$ matrix H relates the actual state of the system to the measurement $\mathbf{z_k}$. $\mathbf{w_k}$ and $\mathbf{v_k}$ are independent random variables with normal distribution representing the process and the measurement noise, respectively:

$$p(w) \sim \mathcal{N}(0, Q), \tag{4.6}$$

$$p(v) \sim \mathcal{N}(0, S). \tag{4.7}$$

Though the real state $\mathbf{x_k}$ of the system at time k is never exactly known, the Kalman filter works with three estimates of the system state: (1) the a priori estimate $\hat{\mathbf{x}}_\mathbf{k}^-$, which is a prediction of the next value based on the previous ones; (2) the measurement $\mathbf{z_k}$, related to $\mathbf{x_k}$ by Equation 4.5; and (3) the a posteriori estimate $\hat{\mathbf{x}}_\mathbf{k}$ which combines the a priori estimate and the measurement, and is the actual output of the filter. In order to compute $\hat{\mathbf{x}}_\mathbf{k}$, the Kalman filter works recursively with two steps: the time update, in which the a priori estimate of the value is predicted; and the measurement update, in which the prediction is corrected with having the measured value. This two step cycle is illustrated in Figure 4.6.

In the time update step, the next value of the system state is predicted based on the previous value using the following equation:

$$\hat{\mathbf{x}}_\mathbf{k}^- = A\hat{\mathbf{x}}_\mathbf{k-1} + B\mathbf{u_{k-1}}, \tag{4.8}$$

For the SNR filter, where the system state is the scalar SNR value, this equation can be written as

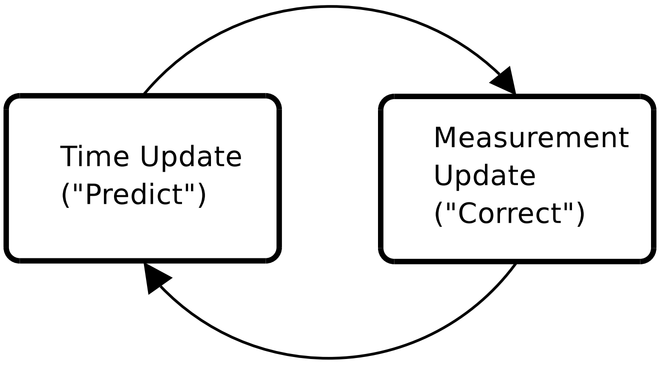

Figure 4.6: *The Two Step Cycle of a Kalman Filter*

$$\hat{x}_k^- = A\hat{x}_{k-1} + B \cdot 1 = \alpha\hat{x}_{k-1} + c \tag{4.9}$$

with the model parameters α and c. This approach of the next SNR value, being dependant solely on the previous one and a constant c, is an autoregressive model of order 1 denoted by AR(1). The parameters α and c of the autoregressive model are calculated at each filtering step using the least squares method [35]:

$$\alpha = \frac{\sum_{i=0}^{O-1} x_{k-i}x_{k-i-1} - (O-1)\overline{x}^2}{\sum_{i=0}^{O-1} x_{k-i-1}^2 - (O-1)\overline{x}^2}, \tag{4.10}$$

$$c = (1-\alpha)\overline{x}, \tag{4.11}$$

where $\overline{x}$ denotes the mean of the O latest training sample vector. O is the order of the training data. We discuss the choice of O later in the evaluation part (cf. Section 6.4.2). Furthermore, in the time update step, the a priori estimate error covariance is calculated according to

$$P_k^- = AP_{k-1}A^T + Q_k, \tag{4.12}$$

where P_{k-1} is the a posteriori estimate error covariance (see below). The measurement noise covariance Q_k can be dynamically estimated with the following equation (cf. [39]):

$$Q_k = \frac{\sum_{i=1}^{O} \hat{x}_i - (\alpha \hat{x}_{i-1} + c)}{O} \tag{4.13}$$

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

$$\mathbf{\hat{x}_k} = \mathbf{\hat{x}_k^-} + K_k(\mathbf{z_k} - H\mathbf{\hat{x}_k^-}). \tag{4.14}$$

The measurement innovation K_k is defined as

$$K_k = \frac{P_k^- H^T}{H P_k^- H^T + S}. \tag{4.15}$$

The a posteriori estimate error covariance is calculated according to

$$P_k = (I - K_k H) P_k^-. \tag{4.16}$$

Time Update Step	Measurement Update Step
$\alpha = \frac{\sum_{i=0}^{O-1} x_{k-i} x_{k-i-1} - (O-1)\bar{x}^2}{\sum_{i=0}^{O-1} x_{k-i-1}^2 - (O-1)\bar{x}^2}$	$K_k = \frac{P_k^-}{P_k^- + S}$
$c = (1-\alpha)\overline{x}$	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{x}_k^-)$
$\hat{x}_k^- = \alpha \hat{x}_{k-1} + c$	$P_k = (1 - K_k) P_k^-$
$Q_k = \frac{\sum_{i=1}^{O} \hat{x}_i - (\alpha \hat{x}_{i-1} + c)}{O}$	
$P_k^- = \alpha^2 P_{k-1} + Q_k$	

Table 4.1: *Kalman Filter Equations for the SNR Filter*

In wireless networks, the process noise $\mathbf{w_k}$ is usually modelled by a normally distributed random variable with typical standard deviation from 4 - 8 dB (cf. [39]). A value of $S = 49$ was chosen in our case for the process noise covariance, which relates to assuming a standard deviation of noise of 7 dB.

In case of filtering the SNR the above equations are simplified because the system state $\mathbf{x_k}$ and its estimates are scalar values, thus all matrices and vectors become scalar. Furthermore, the equations are simplified by setting $H = 1$, as the SNR is directly measured. The resulting equations for filtering the SNR values are summarized in Table 4.1.

4.2.2 Prediction

As depicted in Figure 4.2, the prediction part can be split in two functions, namely Parameter Estimator and State Predictor.

Parameter Estimation with Cross-Correlation

The parameters which the Parameter Estimator hands over to the State Predictor are references to points in the link history, where similar situations to the actual one have been observed. The information what the Parameter Estimator gets from the Observer is the training data and the query. Thus, its task is to find patterns in the training data which are similar to the query.

In order to do this, it computes the normalized cross-correlation [40] of the query and the training data. The normalized cross-correlation $\gamma(m)$ at lag m is a measure of similarity of the query to the appropriate part of the training data at this lag. It is defined as

$$\gamma(m) = \frac{\sum_{i=1}^{o} [q(i) - \overline{q}][t_{j,f}(m+i) - \overline{t_{j,f}}]}{\sqrt{\sum_{i=1}^{o} [q(i) - \overline{q}]^2 \sum_{i=1}^{o} [t_{j,f}(m+i) - \overline{t_{j,f}}]^2}}, \tag{4.17}$$

where $q(i)$ denotes the ith value of the query, $t_{j,f}(i)$ denotes the ith value of the training data of the link between node j and node f, and $\overline{q}$ and $\overline{t_{j,f}}$ denote the mean of the query and the according part of the training data, respectively.

An example of the normalized cross-correlation function is depicted in Figure 4.4(b). Note that in this example only one time series of training data was used. When the node has y neighbors, this results in computing y normalized cross-correlations. The plot shows that there are several good matches resulting in local maxima at lags around $m = \{65, 100, 140, \ldots\}$ and the global maximum is at $m = 316$ with $\gamma(316) = 0.94$.

The Parameter Estimator should hand over these lags representing good matches to the State Predictor. Thus, it has to determine the local maxima of the correlation function. For doing this, a threshold γ_{min} is defined, such that m is a good match, if $\gamma(m) \geq \gamma_{min}$.

Definition 4.2.4 *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 trade-off between being not too strict and not too loose about what a good match is. The former case could lead to having none or only a small number of matches, since the latter one would define situations as matches which are not really similar to the query. In order to find the local maxima of $\gamma(m)$, first all the regions of m where the normalized cross-correlation is above γ_{min} 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 Predictor.

We discuss the choice of the match threshold γ_{min} later in the evaluation part (see Section 6.4.3) of this thesis.

Prediction by Creating the Local Model

Getting the set of matches from the Parameter Estimator and the training data from the Observer, the task of the State Predictor is to create the local model of the link. The model is based on the training data regions following the lags of the matches. This means, that if a lag of m is received from the Parameter Estimator, the part of the training data used for creating the model is the measurements of $m \ldots (m+l)$ for an *l-steps-ahead* prediction. So, the first step in modelling is to create from the set of lags a set of predictors[8].

Definition 4.2.5 *If the set of matches contains i lags $\{m_1 \ldots m_i\}$, that i parts of the training data $\{t(m_1 \ldots m_1 + l) \ldots t(m_i \ldots m_i + l)\}$ form the **set of predictors p**. The i-th predictor is denoted by p_i.*

Parameter l, the length of the predictors, is called the prediction order. l determines how far in the future the prediction will reach. It is a design parameter

[8]Note that these predictors are not the same as the State Predictor, as they are patterns in the link history since the State Predictor is a functional part of a prediction algorithm.

and can be set according to the needs of the application for which the prediction is used.

Having a set of predictors, the link model can be created in the following way. In the set of predictors, each p_i represents a past situation where the link was in a similar state to its current status. It can be assumed, that in these predictors different patterns of SNR changes appear because in a given situation the nodes have typically several options of how to behave. 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 ones. Again, the normalized cross-correlation function is used for measuring the similarity of predictor p_i to p_j, which is denoted by $\gamma_{i,j}$. After this, the average similarity of a predictor p_i is defined as

$$\gamma_i = \frac{\sum_{j=1}^{L} \gamma_{i,j}}{L}, \tag{4.18}$$

where L is the total number of predictors.

As the prediction, the predictor with the maximum average similarity among all the predictors is chosen and the link future behavior is modelled by the selected predictor. Taking one of the predictors directly as the prediction means that the prediction has the same length as the predictor. Thus, if the predictor contains l measurements, an *l-steps-ahead prediction* is performed.

Fallback Solution Using Autoregression

It can happen that the Parameter Estimator does not find any match in the training data due to the following reasons:

- The training data are too short, as the order of the 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 l (as the l 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, we use a fallback solution exploiting the inherent predictive power of the Kalman filter. As our Kalman filter, a first order autoregressive link model AR(1) is created in the time update step of the state observation (see Section 4.2.1). Using this model for an iterative *l-steps-ahead prediction*[9] 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.

4.2.3 Summary of the Design Parameters

After discussing our XCoPred prediction mechanism, Table 4.2 gives a summary of the design parameters. The complete table with all the parameter values chosen is presented in Section 6.4.4.

Design Parameter	**Value**
Kalman filter parameters:	
α	computed according to Eq. 4.10
c	computed according to Eq. 4.11
process noise covariance S	49
training data order O	see Section 6.4.2
Prediction parameters:	
measurement interval T	1 sec
no. of stored measurements N	2048
query order o	see Section 6.4.3
match threshold γ_{min}	see Section 6.4.3
prediction order l	depends on the application

Table 4.2: *Summary of the XCoPred Mechanism's Design Parameters*

[9]Iterative prediction means that the predicted next-step-value is used again as input for the model. Doing this l times leads to an *l-steps-ahead* prediction.

4.3 Using XCoPred in the Priority Based Selection Algorithm

In order to select a stable Dominating Set, during the selection process each client should have a link to a server which is predicted to be stable for a certain amount of time. Thus, we introduce a *link stability criterion* into the PBS algorithm:

Definition 4.3.1 *A link is predicted to be stable, if it is available for the next $l \times T$ seconds, where l is the prediction order and T is the measurement interval.*

As l is the number of values in the time series of the prediction, to check whether a link is stable the prediction simply has to be scanned for zeros[10]. If a zero appears in the prediction, the link is predicted to be unstable.

Considering the pseudo code of the PBS algorithm in Listing 3.1, only a little adaptation has to be done in order to incorporate the link stability criterion into PBS. In line 7, where the state of the node is determined, instead of

```
7:          if(isNeighbor(DOMINATOR)==true) {
```

the following condition for switching to DOMINATEE state is to be used:

```
7:          if(isStableNeighbor(DOMINATOR)==true) {
```

where `isStableNeighbor(DOMINATOR)==true` means that the node has at least one DOMINATOR neighbor and the link to this DOMINATOR node is expected to remain stable for $l \times T$ seconds[11]. See Section 6.4.6 for evaluation results of using XCoPred in the PBS algorithm.

Note that this link stability requirement holds only during the DS selection period. Once all the nodes have decided their state as either DOMINATOR or DOMINATEE, a DOMINATEE node is not requested to have a stable link to

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

[11]Though we did not carry out a detailed complexity analysis of XCoPred, it can be easily seen that computing the normalized cross-correlation of the queries and the training data as well as correlating the predictors with one another present a big computation overhead for the nodes. Especially for devices with limited resources, such as mobile phones, this might be a too big burden and a simpler or more optimized solution might be preferable. However, analyzing the complexity and optimizing XCoPred are left as another topic for future research (see Section 8.4 for possible ideas).

the given DOMINATOR node anymore. Thus, just because a DOMINATEE node has no stable link to the given DOMINATOR node anymore it does not switch back its state to CANDIDATE and does not trigger a new selection round. However, to require this would be a further improvement possibility. Letting the link stability criterion be always valid would have the benefit, that the DS would be updated proactively before a change is really necessary. Once a node predicts that it might loose the connection to its DOMINATOR neighbor, it would already trigger a new DS selection round before the link really breaks. This might prevent the interruption of the service for the given node.

4.4 Chapter Summary

In this chapter, we have surveyed the fundamentals of mobility prediction in MANETs. Moreover, we have presented our XCoPred mechanism we have developed for mobility prediction and pointed out its use in the PBS algorithm.

XCoPred predicts the variations of the wireless link quality based on pattern matching without using any external hardware or reference point. Each node in the ad hoc network monitors the signal quality (Signal to Noise Ratio) of its links to obtain a time series of link quality measures. These measurements are filtered with a Kalman filter to get rid of noise. When a prediction is required, the node tries to detect patterns similar to the current situation in the history of its links' SNR values and to obtain a set of predictors. For this purpose, the node computes the normalized cross-correlation between the current pattern and the history of the links' quality. As the prediction itself, the most probable predictor is used. For cases where no match can be found, we apply a fallback solution based on an autoregressive model. To incorporate XCoPred into the PBS algorithm we defined a link stability criterion. Taking into account link stability based on prediction in the DS selection, the stability of the selected DS can be improved.

We show later in Section 6.4 based on a simulation study how the prediction parameters can be set appropriately, how accurate predictions we can achieve and how XCoPred can improve the performance of the PBS algorithm.

Chapter 5

Implementation

In this chapter, we provide an overview about how we implemented the algorithms and mechanisms being developed during this thesis work to prove the viability of our concepts. The implementation also forms the basis for our investigations and evaluation. Thus, first we describe the implementation of PBS, NWC and XCoPred in the NS-2 network simulator. Then, we discuss their implementation in our SIRAMON framework with pointing out the SIRAMON testbed we have built and a demo application, a simple real-time multiplayer game called Clowns, which we implemented to demonstrate the usefulness of SIRAMON.

5.1 Implementation in NS-2 Network Simulator

In the following, we give a brief overview about NS-2, then we present the main points of our PBS, NWC and XCoPred implementation in this simulator.

5.1.1 NS-2 Network Simulator

NS-2 is a free and open source discrete event network simulator widely used in research projects. It can be downloaded from the NS-2 homepage [20] and is available for several Unix and Windows platforms. A handy introduction to NS-2 can be found in [41]. The development of NS-2 started in 1989 and it is maintained by the VINT (Virtual InterNetwork Testbed) [42] project. In our work, the *ns-allinone* package release 2.28 of the simulator has been used. Since 1989 the simulator has evolved into a complex and powerful tool supporting a rich number of protocols and applications. NS-2 is implemented in two programming languages, the core is written in `C++` and to configure and run the simulations `OTcl`[12] is used. The combination of these two languages offers a compromise between performance and ease of use of the simulator. The simulation of a specific protocol consists of four steps:

1. Implementation of the protocol in `C++` and `OTcl`;
2. Description of the simulation scenarios in `OTcl`;
3. Running the simulations;
4. Analysis of the generated trace files.

For inspecting network simulation traces and real world packet traces a `Tcl/TK` based animation tool called NAM (Network Animator) [44] can be used.

5.1.2 Implementation of the Priority Based Selection Algorithm in NS-2

The PBS algorithm has been implemented as an agent (`ZSSpbsAgent`) in the core part of the NS-2 simulator that can be attached to different nodes in the simulation configuration scripts. `ZSSpbsAgent` is inherited from the main zone server selection agent called `ZSSAgent` that provides some general functionalities for selecting zone servers. This gives the flexibility to implement any modified PBS or even completely different algorithms by just adding a new agent inherited from the `ZSSAgent`.

The workflow and the required actions of the PBS algorithm are controlled by a Finite State Machine (FSM) defined by states and transitions between these states. The transitions are executed based on the incoming events. The specification and details of this FSM can be found in Appendix B.1. Note that we used the same state machine for the implementation in the SIRAMON framework, too.

Figure 5.1 depicts the basic structure of the `ZSSpbsAgent`. Every agent has a `recv()` and a `send()` function beside the FSM to receive and send the

[12]`OTcl` is the object oriented variant of the `Tcl` script language. For more details see [43].

PBS messages, respectively. The `recv()` function handles the incoming messages, extracts the relevant information found in the message into the own neighborlist and forwards the appropriate event to the FSM. Based on the received event, the FSM performs the corresponding action. On the other hand, when it is required to send out a PBS message, the FSM compiles the information, forwards it to the `send()` function which creates the PBS message and sends it out.

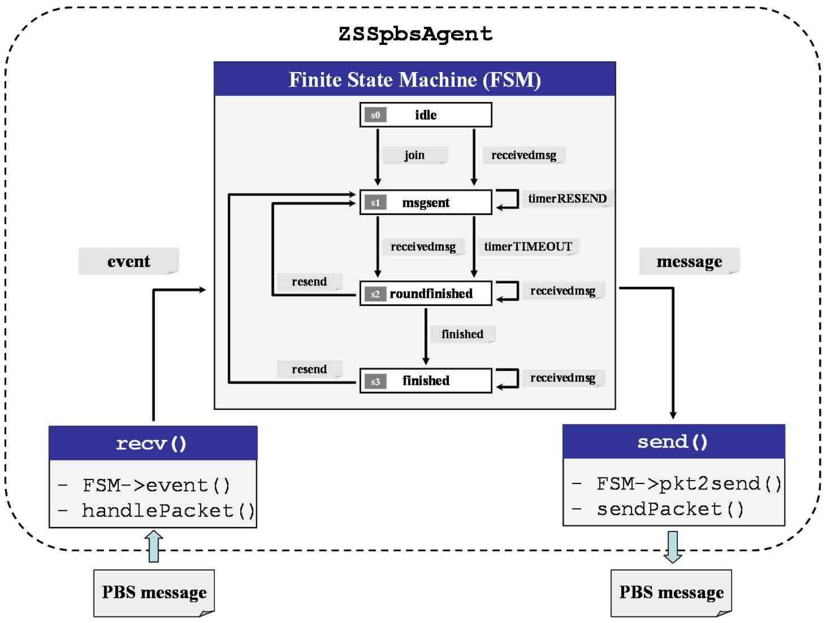

Figure 5.1: *Structure of the* `ZSSpbsAgent` *in NS-2*

Communication Between the PBS Agents

For the communication between the agents we have defined and developed an own protocol called PT_ZSS, which is based on the UDP transport layer protocol, and an own message format illustrated by Figure 5.2. This message consists of a PBS header, the localnode fields containing information about the sender node, and several neighbornode fields containing information about the neighbors of the node. Moreover, the neighbornode fields contain a flag called ‘fullconnected’ that is used by the DOMINATOR nodes when they have to force another node to join the DS (cf. Section 3.2).

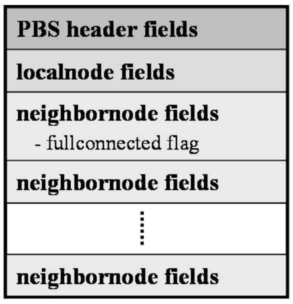

Figure 5.2: *PBS Message*

Network Monitor

To cope with node mobility and detect changes in the topology of the network a basic Network Monitor has been implemented, as well. This monitor uses periodically sent ‘alive’ messages. If a node sends no alive message for a specified amount of time, we assume that the node disappeared and it is removed from the neighborlist. New nodes can easily be detected based on newly received alive messages.

More information about the PBS implementation in NS-2 can be found in Appendix B.2.

5.1.3 Implementation of the Node Weight Computation Mechanism in NS-2

The NWC mechanism, which computes the node weight, has been implemented as an extension to the PBS algorithm. Figure 5.3 shows the structure of the NWC implementation in NS-2.

The implementation follows the programming technique of NS-2. In the `OTcl` space, the necessary variables are assigned, set and then passed to the `C++` space where the NWC mechanism is implemented as an extension to PBS. The three groups of these variables are:

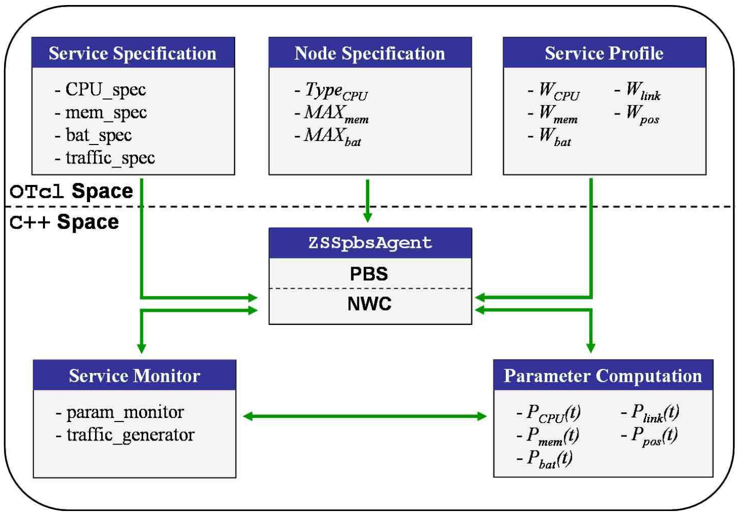

Figure 5.3: *Structure of the NWC Implementation in NS-2*

- **Service Specification:** It is used to characterize the service to be simulated. Variables related to and describing the service such as CPU/memory load, battery consumption and also traffic rate between the zone servers and their clients are defined and set here.
- **Node Specification:** The node properties are assigned and set here, such as the CPU type, memory and battery capacity. This will allow the computation of the parameter values during the simulations.
- **Service Profile:** The parameter weights are defined and set here for the given service. These weights are used, together with the parameter values, to compute the node weight.

In the `C++` space, the `ZSSpbsAgent` is extended with the node weight computation mechanism. Moreover, we have implemented also the following functionalities:

- **Service Monitor:** This functionality is responsible for monitoring and checking the parameter values. It collects statistics about the parameter values periodically. Additionally, it checks in case of the local resource parameters (CPU, memory and battery) whether their values are below the preset thresholds. If yes, an anomaly is generated (see Section 3.3.2). Furthermore, it controls the traffic generation between the zone servers and their clients, and the zone servers themselves.

- **Parameter Computation:** It computes the parameter values using the node and service specification settings and the information collected from the Service Monitor.

5.1.4 Implementation of the XCoPred Prediction Method in NS-2

For the implementation of our mobility prediction mechanism we defined two new classes in the `C++` space of NS-2: the `MobilityStateObservation` class for the state observation part and the `MobilityPrediction` class for the prediction part (see Figure 5.4). Then, we extended the `ZSSpbsAgent` such that each PBS agent creates instances of both of these classes. Moreover, using XCoPred in the PBS algorithm, i.e., incorporating the link stability criterion (see Section 4.3), required also some changes in the `Neighborlist` class and in the `ZSSpbsAgent` code.

State Observation

In order to keep track of a node's links, a structure called `LinkMeasurements` was defined (see Appendix B.3.1). 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 also be stored). Furthermore, the `MobilityStateObservation` class has two important interfaces.

The first interface is used by the `recv()` function of the `ZSSpbsAgent`. The `recv()` function calls the `insertMeasurement()` function of `Mobility-StateObservation` in order to store the measured SNR value for each received packet in the `lastMeasurement` variable of the according `LinkMea-surements` structure. Because measurements should only be performed once every T second (recall that T is the measurement interval defined in Section 4.2.1), a timer was added to the `ZSSpbsAgent`. The `makeMeasurement()` function, which is called every T second, checks whether 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

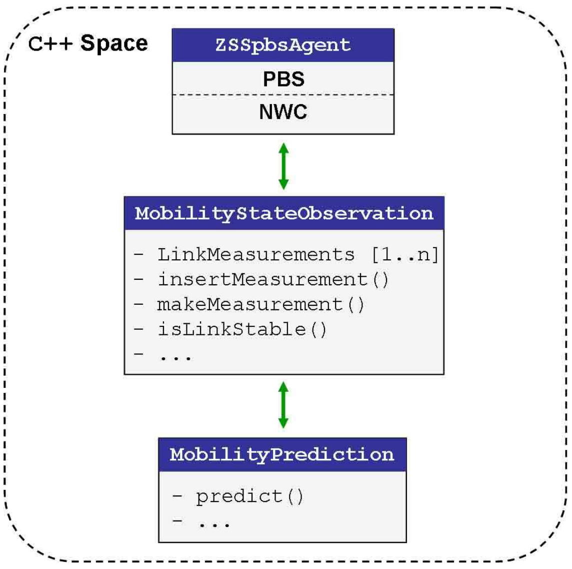

Figure 5.4: *Structure of the XCoPred Implementation in NS-2*

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 ‘alive’ messages for all of its active links. The alive messages are used by PBS in order to keep the list of direct neighbors accurate and are sent by default every 0.1 second. 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 lifetime of a prediction was set to 2 seconds. This value should make sure that during a zone server selection round each link is predicted only once. 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.

Beyond these two interfaces the `MobilityStateObservation` class provides some additional functions for the `MobilityPrediction` class, too. These are mainly used to hand over the training data and query to the prediction part.

More information about the state observation part can be found in Appendix B.3.2.

Prediction

The `MobilityPrediction` class has one important public function called `predict()`, which is used by the `MobilityStateObservation` class 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 4.2.2. 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 adequate `LinkMeasurements` structure.

Link Stability Criterion

To implement the link stability criterion in PBS (see Section 4.3), we had to make several changes in the `Neighborlist` class and in the `ZSSpbsAgent`.

The `Neighborlist` class was extended with two new functions. The `getAllStableNeighbors()` and the `getStable1HopNeighbors()` functions are the pendants to the default functions `getAllNeighbors()` and `get1HopNeighbors()` returning only the neighbors on which the stability criterion holds. Additionally, a configuration parameter was introduced to the `Neighborlist` class, which controls whether prediction is to be used or not.

The `ZSSpbsAgent` class was changed in two places. First, the `sendNeighborlist()` function was modified to include only those neighbors on which the stability criterion holds. In order to do this, it uses the `getStable1HopNeighbors()` function of the `Neighborlist` class. The second change concerns the `determineStatus()` function, which was modified in several places to use the `getAllStableNeighbors()` and `getStable1HopNeighbors()` functions in order to determine the PBS status of the node.

5.2 Implementation in SIRAMON

In the following, we give a brief overview about the structure and implementation of our SIRAMON framework. Then, we present shortly our SIRAMON testbed we have built and the demo game application we have implemented to demonstrate the working of SIRAMON. And finally, we discuss the main points of our PBS, NWC and XCoPred implementation in SIRAMON.

5.2.1 SIRAMON Framework

As we discussed in Section 2.4.1, SIRAMON is our service provisioning framework for self-organized networks based on a modular and distributed design. Its components can be replaced according to specific demands which makes it possible to support different type of services. SIRAMON consists of the following modules (see Figure 5.5): (i) Service Specification; (ii) Service Discovery; (iii) Service Deployment; (iv) Service Management; (v) Environment Observer.

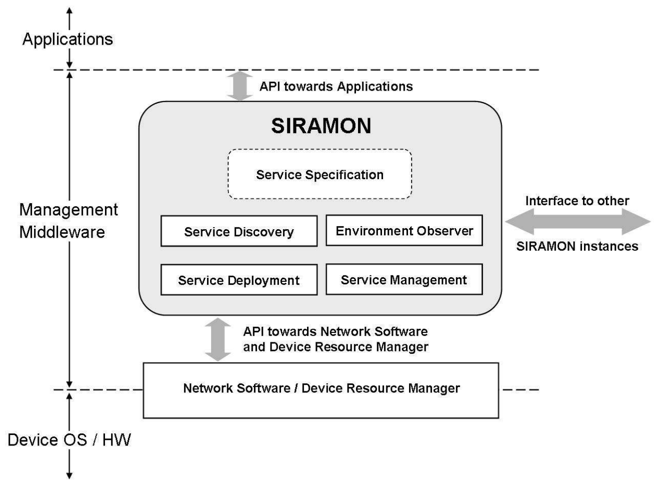

Figure 5.5: *Ad Hoc Device Model with SIRAMON*

Service Specification contains the used Service Model which describes the role of the device in the service, the functions and connections of service elements to build the service. Service Discovery is responsible for service advertisement if the node hosts a service, or service lookup if the node intends to use a service. By the Service Deployment module, creation, installation and configuration of services are carried out. The Service Management component controls the service maintenance, reconfiguration and termination functions. And the Environment Observer module deals with the monitoring of the node resources and the service context.

We have implemented SIRAMON in a platform independent way using Java programming language. For more details on SIRAMON's design and implementation see [21].

5.2.2 SIRAMON Testbed

In order to have a proof of concept implementation and gain some experience with SIRAMON in real world environment we have built a small testbed. The mobile part of this testbed consists of five devices, such as three laptops and two PDAs. Furthermore, there are two desktop PCs also included in the testbed which are not mobile but equipped with wireless interfaces, too. All of the devices are running Linux as operating system and SIRAMON is installed on them. In the testbed, the nodes are communicating with each other via 802.11b Wi-Fi interfaces [36] in ad hoc mode, though the PDAs are also Bluetooth enabled. A snapshot of this testbed is shown in Figure 5.6.

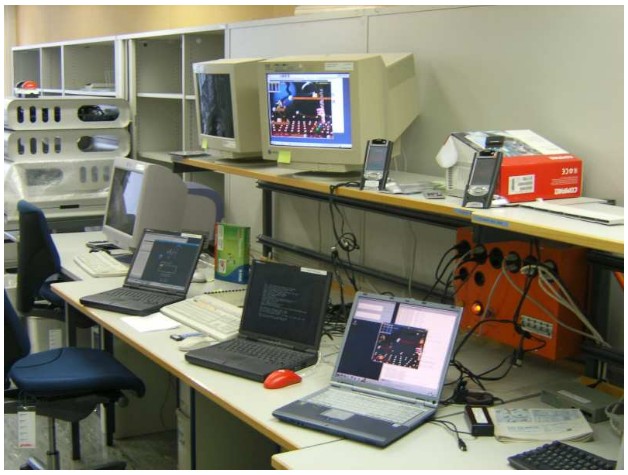

Figure 5.6: *SIRAMON Testbed*

To demonstrate the working of SIRAMON from the application's point of view, we also have developed and implemented a real-time ad hoc multiplayer game called Clowns (a screenshot of Clowns is shown in Figure 5.7) [45]. It is a simple jump-and-run type game with a modest 2D graphical interface. The players' clowns have to catch and kick each other's clowns, in this way collecting points. Who collected the most points wins the game. Clowns has been implemented in C++ on top of SIRAMON. It uses a distributed zone-based server architecture with real-time synchronization between the servers. All the communications are done via wireless interfaces of the testbed nodes in ad hoc mode. With the help of our PBS algorithm, the Clowns clients are able to cope with the dynamic changes of the network. Thus, to handle mobility and disappearing zone servers they are able to select new servers and reconnect to them on the fly.

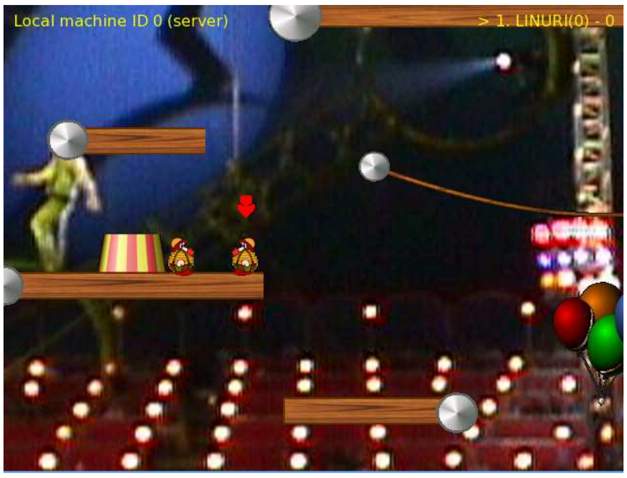

Figure 5.7: *Screenshot of the Demo Game Called Clowns*

5.2.3 Implementation of the Priority Based Selection Algorithm in SIRAMON

The zone server functionality and the PBS algorithm have been implemented in the Service Management module of the SIRAMON framework. Like the implementation in NS-2, the PBS algorithm is implemented in the class called

`ZSS_PBS` that is inherited from the basic `ZSS` class that provides some general functionality for selecting zone servers. Again, this gives the possibility to implement modified algorithms or completely different approaches for zone server selection in future projects. The Service Management module is implemented in its own java package (`Siramon.Management`) inside the framework. For zone server selection two subpackages, `Siramon.Management.zss` and `Siramon.Managament.zss.utils`, have been added. A short description of these new subpackages can be found in Appendix B.4.

Communication Between the Nodes

For the communication between the different nodes, the existing procedures provided by the framework have been used. The neighborlists are embedded in the message part of the SIRAMON packets. In order to be able to distinguish between different running instances of the PBS algorithm supporting different services, a unique Session ID is assigned to every instance. This ID needs to be included as well in the header of the SIRAMON packet. Based on this information, the network receiver thread of the framework is able to detect to which instance the given packet belongs. In addition, the information about the server component to be started is also transmitted inside the packet. The structure of a SIRAMON packet as it used by PBS is described in Table 5.1. The Service Identifier and the Message fields contain the most important PBS information. The Service Identifier contains, beside the identifier value 'zss', the Session ID and the Server Component information. In the Message field, the neighborlist containing the information about the neighbors is stored.

Field	**Description**
Prefix	'Siramon'
Source Address	Contains the address of the sending node
Flooding Flag	Not used
Timestamp	Not used
Service Identifier	'zss:<SessionID>:<ServerComponent>'
Message	Contains the neighborlist

Table 5.1: *Structure of the SIRAMON Packet as It Used by PBS*

The SIRAMON packets are wrapped into UDP packets and sent to the 1-hop neighbors of the node. For saving resources, these packets are sent only once in a given time period using a broadcast address with the Time To Live (TTL) value set to 1 instead of using unicast connections. Unfortunately, in Java the TTL field can be set only to multicast sockets. Thus, every node has to join a specified multicast address and send the packet containing the neighborlist to this address with TTL set to 1.

Network Monitor

Because the PBS algorithm has to know the 1-hop neighbors of the node, we have implemented a simple network monitor, similarly to the PBS implementation in NS-2, in the `Siramon.Network` package of the SIRAMON framework (see Appendix B.4). The monitor running on the node sends 'alive' broadcast messages with TTL set to 1 into the network periodically and receives alive messages from the neighbors. If a node sends no alive message for a specified amount of time, the monitor assumes that the neighbor is disappeared and removes it from the neighborlist. On the other hand, appearing nodes sending alive messages are added to the neighborlist.

5.2.4 Implementation of the Node Weight Computation Mechanism in SIRAMON

To implement the NWC mechanism we have modified three of the basic SIRAMON modules:

- **Service Specification:** To support the NWC functionality the service description document (it is an XML [46] file in the actual version of SIRAMON) has been extended with the service profile containing the appropriate parameter weights of the given service type. These fields are optional and are used only if the service requires the zone server selection functionality.

- **Service Management:** The node weight computation has been implemented in this module as an extension to the basic PBS algorithm. To compute the node weight the necessary values of the parameters and the parameter weights are obtained from the Service Specification and the Environment Observer module.

- **Environment Observer:** In this module, the Service Monitor part of NWC is implemented which monitors the CPU and memory load, the battery energy level and the link quality of the connections to the node's neighbors. We had to create four external procedures to monitor these parameters, because Java does not have native support to extract the required information from the hardware device drivers.

5.2.5 Implementation of the XCoPred Prediction Method in SIRAMON

To implement our mobility prediction mechanism in SIRAMON, we have followed the same strategy as in its implementation in NS-2. So, we created two new classes (`MobilityStateObservation`, `MobilityPrediction`) wrapped into the newly created `Siramon.Management.mobilitypred` subpackage of SIRAMON's Service Management module for state observation and prediction. Moreover, we adopted the PBS implementation in the `ZSS_PBS` class to make possible the use of prediction.

Measuring the Link SNR Value

As the basis of XCoPred, we have to be able to measure and collect the SNR values of the node's wireless links. The IEEE 802.11 standard defines the wireless signal strength as the Received Signal Strength Indicator (RSSI) [47] which is a vendor dependent measurement in arbitrary units and does not directly refer to the SNR value. However, the RSSI values from common vendors can easily be converted into SNR values (cf. [48]). The employed DLink wireless cards in our testbed are based on an Atheros chip-set controlled by the Madwifi device driver [49]. In order to pass the RSSI values of the wireless link from the device driver to the XCoPred code in SIRAMON, socket buffers are used. These buffers have a control buffer field, where protocols can put their private data. We used this field to store the RSSI values. Thus, we has been extended the Madwifi driver to copy the RSSI values of the wireless link into this control buffer field whenever XCoPred needs actual SNR data.

5.3 Chapter Summary

In this chapter, we have surveyed the implementation of PBS, NWC and XCoPred in NS-2 network simulator and in our SIRAMON service provisioning framework (for more implementation details see Appendix B), respectively. Moreover, we pointed out the SIRAMON testbed we had built to get real world experience with SIRAMON and the demo real-time multiplayer game application called Clowns what we had implemented to demonstrate the working of SIRAMON.

The implementation of PBS, NWC and XCoPred in NS-2 constitutes the basis of our investigation and evaluation of these algorithms and mechanisms what we present in the next Chapter. Unfortunately, we could not accomplish thorough evaluation in the SIRAMON testbed due to its limited size and node mobility. However, this work was still very useful and let us gain a lot of real world experience with mobile ad hoc networking.

Chapter 6

Evaluation

Evaluation and validation of the developed algorithms/mechanisms are an important part of research. Thus, we laid special emphasis on evaluating our approaches via simulations. In this chapter, first we present the specification of a pseudo real-time multiplayer game what we have used as the test application in our simulations. After this, we show our evaluation of the PBS algorithm, the technique of service profile creation for node weight computation using factorial design and simulations, and our evaluation of the XCoPred prediction mechanism together with its application in PBS. And finally, we give a short, simulation based comparative study of the centralized client/server service management architecture, the distributed peer-to-peer architecture and the zone-based architecture using PBS.

6.1 Test Application

As the test application for the investigations of our developed algorithms/ mechanisms, we have used real-time multiplayer gaming. In the following, we give a brief overview about the properties and requirements of real-time multiplayer games.

6.1.1 Real-Time Multiplayer Games

Real-time multiplayer gaming is one of the most popular spare time activities today. Just consider the biggest on-line multiplayer Internet games like

Half-Life, Counter-Strike, Warcraft, Unreal Tournament or Diablo which are played by millions daily all over the world [50]. As technology advances, mobile devices are now capable to be used also in playing distributed games. Playing via ad hoc networks of these mobile devices increases the flexibility of players while offering a convenient way to play at any time and at any place without the need of permanent network infrastructure. Thus, mobile ad hoc multiplayer gaming is one of the most promising future development directions of existing Internet-based multiplayer games.

Quality of Service Properties of Real-Time Multiplayer Games

The most important QoS (Quality of Service) properties of real-time multiplayer games are the end-to-end communication delay, delay jitter and packet loss to a certain degree, while the available network bandwidth is of less importance (see Table 6.1) [51].

QoS Property	**Value**
Max. latency	100 – 150 msec
Max. packet loss	3 – 5 %
Max. jitter	As low as possible
Required bandwidth	Some kbit/s

Table 6.1: *QoS Properties of Real-Time Multiplayer Games*

For most real-time multiplayer games, a maximum of 100 – 150 msec round trip delay is still acceptable. The upper bound of the affordable skew for interactive audio and video applications is estimated around 120 msec. For car racing games, a network delay in the range of 50 to 100 msec is already noticeable but is tolerated by most of the players, whereas a delay of more than 150 msec results in remarkable degradation of the end users' performance. Similar results are applied for first-person shooter games [52]. The effects of jitter, however, are not so well-researched yet. Latency and jitter are strongly coupled in the Internet with the ratio of jitter to latency being 0.2 or smaller [53]. This also means that jitter in general is very small in the Internet if latency is below 150 ms and therefore has little impact on the game. Usually, players' perception of jitter is game-dependant because high level of jitter leads to packets not arriving in time thus requiring the use of game prediction mechanisms such as dead-reckoning [54]. As the quality of these prediction mechanisms differ from game to game, players also perceive jitter differently. Furthermore, prediction mechanisms cannot always anticipate players' actions accurately so high level of jitter usually degrades the players' experience. Hence, the jitter level must be kept as low as possible. The packet loss rate has similar effects and it must be kept in the range of 3 – 5 % [51]. And finally, the network bandwidth requirement of these games is not so critical. With appropriate game state coding the generated game traffic between the player and server node is usually full duplex CBR (Constant Bit Rate) traffic in the range of some kbit/s (e.g., 5 kbit/s for Warcraft III or 34 kbit/s for Counter-Strike) [55].

6.2 Evaluation of the Priority Based Selection Algorithm

It would be interesting to compare the behavior of PBS to other Dominating Set computation algorithms. However, due to the lack of built-in mobility maintenance of the other algorithms we cannot easily perform a simulation based comparison study (we compare analytically the initial DS computation property of the different algorithms in Section 7.1.2). Thus, here we restrict our investigations to the performance evaluation of the PBS algorithm via simulations [13, 15].

6.2.1 Simulation Settings

As we discussed in the previous chapter, we had implemented the PBS algorithm using the NS-2 network simulator [20] and its wireless extensions developed by the Monarch group [56]. Throughout the simulations, each mobile node shares a 2 Mbit/s radio channel with its neighboring nodes using a two-ray ground reflection model [57], IEEE 802.11 MAC (Medium Access Control) protocol [47] and AODV (Ad Hoc On-Demand Distance-Vector) routing protocol [58]. We have simulated 3 different scenarios with 2 different node densities[13]. First, with 15 nodes from which 10 nodes are participating in the game session, and second with 35 nodes from which 25 are participating in the game in average. The following scenarios have been used in the

[13]Note that it would have been more desirable to run the simulations with many more different node densities (e.g., with 15, 20, 25, 30, 35 nodes) but the long simulation time required by even a single simulation prevented us from carrying out this more thorough investigation.

simulations:

- **School Yard Scenario:** The School Yard scenario can be characterized by a group of people (students) who are standing on a school yard of 400x400 m^2 and are playing a multiplayer game together. Because usually there are no huge obstacles on a school yard the transmission range of a device is assumed to be 250 meters, which is a typical value for WLAN in a free area. Due to this wide transmission range, most of the players will have 1 hop connections to each other. The movements of the nodes are simulated by using the Random WayPoint (RWP) mobility model (see Appendix A.1.1 and [56]) in which the non-player nodes are moving freely on the whole area of the school yard between different destination points, since the distance between two destination points of the player nodes is uniformly distributed in the range of 0-15 meters. We assume, that people participating in the current game session are moving only slightly, because it is quite difficult to move and play at the same time for most of the existing games nowadays. The speed of the nodes is uniformly distributed in the range of 0-6 km/h.

- **Train Scenario:** In the second scenario, a train is assumed where some passengers are playing with each other. For the simulation with 15 nodes the geometrical dimensions are 240x5 m^2 (about 8 wagons), and 450x5 m^2 (about 15 wagons) for 35 nodes. Due to the narrow but long area and the assumption that the disjunction between the wagons reduces the transmission range of the devices (which is set to 40 meters), most of the nodes will be connected by several intermediate hops between each others. Because every playing passenger is expected to sit most of the time during a game session, we use the probability value of 50 % that they will move around. This movement can be characterized by moving towards a destination point like the toilet or restaurant wagon, spending some time there and moving back to the seat. Non-player nodes will move around more and will not implicitly move back to the original starting-point. The speed of the nodes are again uniformly distributed in the range of 0-6 km/h.

- **Test Scenario:** For testing the robustness and reliability of the PBS algorithm we have used a testing scenario that faces the algorithm with some more challenging conditions than in the previous scenarios. This scenario has the geometric dimension of 400x400 m^2 for the simulation with 15 nodes and 800x800 m^2 for 35 nodes, respectively. All the nodes are moving around with uniformly distributed speed in the range of 0-30 km/h using again the RWP mobility model. The increased speed leads to much more mobility of the nodes and causes the algorithm to recalculate the Dominating Set more frequently.

Variable	**Setting**
Scenario	School Yard, Train, Test
Number of nodes	15 with 10 player nodes and 35 with 25 player nodes
Node weight	Uniformly distributed between 1-99
Duration of the simulation time	900 sec
Number of repetitions	10
Game joining and leaving points	Randomly distributed during the simulation time
Used mobility model	Random WayPoint
Game traffic	Server ⇔ client: full duplex CBR - 10 kbit/s
	Server ⇔ server: forwarded traffic coming from the clients
Background traffic	CBR - 5 kbit/s
	5 (15 nodes) and 15 (35 nodes) parallel connections

Table 6.2: *Simulation Settings of the PBS Evaluation*

Table 6.2 summarizes the main simulation settings. To increase the confidence level [35] of the simulations, we had repeated every simulation 10 times using different seed values then averaged the results. This means 10 simulations per scenario and node density, and 60 simulations in total. We set the duration of the game session, and thus the simulation time, to 900 seconds in every scenario. The players' game joining and leaving points in time were randomly distributed during the simulation. To simulate a real environment we generated some traffic, as well. Between the servers and their clients we generated a full duplex 10 kbit/s CBR data flow (20 packet/second with 64 byte packet size) simulating the game traffic [55]. For server to server synchronization, we applied the simplest solution without any optimization or sophisticated data coding. Thus, every server simply forwarded the traffic to the other servers coming from its clients. We also added some background traffic with the following parameters: In the scenarios with 15 nodes, the background traffic was generated by 5 parallel connections being active at the same time during the whole simulation between any two random nodes. Per connection, the sender produced a 5 kbit/s (10 packet/second with 64 byte packet size) data flow for 30 seconds, then a new connection was established. In the scenarios with 35 nodes, we increased the number of parallel connections being active at the same time to 15. Moreover, we set the node weight randomly between 1 and 99 using uniform distribution[14].

6.2.2 Simulation Results

We investigated the time PBS requires to compute and maintain a Dominating Set of the network graph and the signaling traffic generated during this time. For this we used the following metrics:

- **Bandwidth:** It indicates the bandwidth required by the PBS overhead data from the viewpoint of a node (the total available bandwidth was 2 Mbit/s in these simulations). The used bandwidth is strongly influenced by the size of the sent messages and increases as more neighbors a node has. To avoid counting some messages several times, only the sent data has been considered. The traffic characteristics of the nodes contain some bursts, because the nodes only start exchanging neighborlists, when some changes in the network topology have been detected and no messages are sent if there is no change. In Figure 6.1, an example of the PBS overhead data (without the periodic ‘alive’ messages) sent by a single node during a game session is shown. We can see that at the beginning of the game session a lot of data is transmitted, but once the DS is built the burst behavior can be detected. Thus, during some time periods no PBS data is transferred, because there are no changes in the network topology.

  The results of the bandwidth investigation (counting also the periodically sent ‘alive’ messages) are shown in Table 6.3 (we calculated the minimum, maximum, average and standard deviation values based on

[14]For sake of simplicity we used positive integer numbers as node weights in this set of simulations.

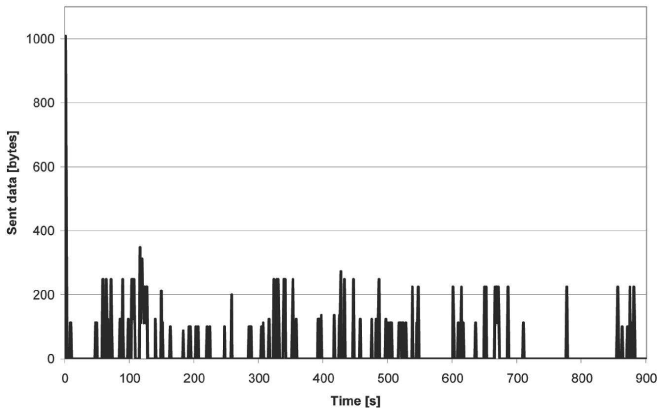

Figure 6.1: *Example of PBS Data Sent by a Node During a Game Session*

the simulation traces). We can see from the table that in the School Yard scenario the required bandwidth is increasing from 440 bit/s to roughly 1.7 kbit/s on average if the number of nodes increases from 15 to 35. In the bigger Test scenario with 35 nodes the simulation area has been substantially expanded from 400x400 m^2 to 800x800 m^2 and the required bandwidth has decreased to 582 bit/s comparing with the School Yard scenario because there were less 1 hop connections. Moreover, using higher speed wireless interfaces in the mobile ad hoc network (e.g., 11 or 54 Mbit/s according to the IEEE 802.11b/a standards [47]) the required bandwidth by the PBS algorithm will not cause any serious problem, since even in the worst case of our simulations it occupied only 0.1 % of the available 2 Mbit/s channel.

- **Determination Delay:** This delay indicates the time that is needed until a node gets a neighboring DOMINATOR node or becomes itself a DOMINATOR. This happens at the beginning of the game session, or during the game when the connection to a DOMINATOR node gets lost due to the node's mobility and a new DOMINATOR node needs to be selected in order to rebuild the Dominating Set. The results of the determination delay investigation are shown in Table 6.4. We can see that in all the three scenarios with 15 nodes, the average delay is in the same order between 111 and 185 milliseconds. However, the maximum delay in the Train scenario is much higher (1.039 seconds) than in the other

Scenario	**Min.**	**Max.**	**Avg.**	σ
School Yard w/ 15 nodes	410	480	440	8
Train w/ 15 nodes	402	430	412	3
Test w/ 15 nodes	416	572	490	15
School Yard w/ 35 nodes	676	2,412	1,722	180
Train w/ 35 nodes	402	498	442	11
Test w/ 35 nodes	408	936	582	53

Table 6.3: *Used Bandwidth by the PBS Algorithm [bit/sec]*

two scenarios. This is, because the narrow but long area in the train creates a situation similar to the worst case scenario as shown in Figure 3.8 of Section 3.2.4. This ‘chain’ topology can lead to situations where the DOMINATOR nodes need to be determined step by step. With the growing number of nodes, the determination delay increases as well. In the scenarios with 35 nodes, there are more considerable differences in the average values of the determination delay that are caused by the different number of 1 hop neighbors, but the maximum values are all in the same order.

Scenario	**Min.**	**Max.**	**Avg.**	σ
School Yard w/ 15 nodes	18	690	159	172
Train w/ 15 nodes	7	1,039	111	178
Test w/ 15 nodes	21	674	185	188
School Yard w/ 35 nodes	3	1,130	283	251
Train w/ 35 nodes	4	1,175	153	193
Test w/ 35 nodes	12	1,358	510	311

Table 6.4: *Determination Delay [msec]*

- **Number of DOMINATOR Neighbor Changes:** This metric indicates from the view of a node how often it looses the connection to a neighboring DOMINATOR node and a new node has to be determined as

DOMINATOR. Obviously, this number should be as small as possible. As we can see observing the determination delay it can take more than one second until a new DOMINATOR is determined in a dense scenario with unfavorable topology. During this time the client needs to connect to a server that is more than 1 hop away if this is possible.

In Table 6.5, we indicated how often a node had lost the connection to its DOMINATOR node in the scenarios being used. We can notice that the average values are between 0.1 and 0.4. This means that most of the nodes had constant connection to a neighboring DOMINATOR node during the whole game session. In the Train scenarios, however, a node can loose its DOMINATOR node more frequently because of the narrow but long geometrical shape of the train and the limited transmission range if the DOMINATOR node starts moving around. Clearly, the higher level of mobility also has essential influence on the number of required changes. For example, in the Test scenarios this number is relatively high because the topology can change much faster and the Dominating Set needs to be recomputed more frequently. An interesting observation is that there are nodes in all scenarios never losing their DOMINATOR nodes.

Scenario	**Min.**	**Max.**	**Avg.**	σ
School Yard w/ 15 nodes	0	1	0.1	0.2
Train w/ 15 nodes	0	2	0.2	0.4
Test w/ 15 nodes	0	2	0.4	0.7
School Yard w/ 35 nodes	0	2	0.1	0.4
Train w/ 35 nodes	0	3	0.2	0.6
Test w/ 35 nodes	0	4	0.2	0.7

Table 6.5: *Number of DOMINATOR Neighbor Changes*

- **Number of DS Changes:** This metric is used to give an indication about the stability of the DS. It shows how often the initial Dominating Set needs to be changed during the game session from a global viewpoint. Note that in our implementation, a DOMINATOR node remains a DOMINATOR, even if its clients are covered by other DOMINATOR nodes. A DOMINATOR node switches back to DOMINATEE status only if there are no DOMINATEE nodes left in its neighborhood. We chose this solution, because it could lead to an oscillation problem if a DOMINATOR node having the lowest priority switched back immediately when it detected other DOMINATOR nodes also covering its DOMINATEE neighbors.

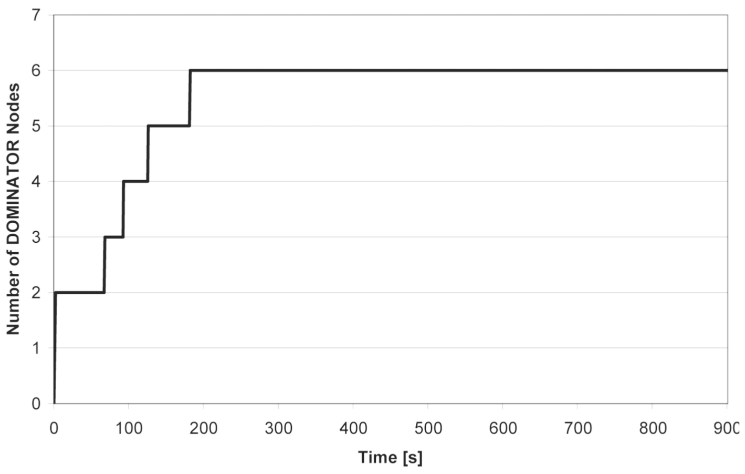

Figure 6.2: *DOMINATOR does not Switch Back to DOMINATEE*

In Figure 6.2, the number of DOMINATOR nodes during the simulated time in one of the School Yard scenario simulations with 35 nodes is shown when the DOMINATOR nodes do not switch back to DOMINATEE status at all. We can see, that after building the initial Dominating Set consisting of 2 nodes, 4 further nodes were added to this set until the end of the simulation which led to 4 changes in the DS during the whole session.

In Figure 6.3, the same situation is outlined if the DOMINATOR nodes immediately turn to DOMINATEE status when they detect that their clients are covered by other DOMINATOR nodes. This causes a frequent oscillation in the number of DS nodes massively increasing the number of DS changes. The oscillation can be alleviated if a DOMINATOR node waits a given amount of time then checks again whether it is still unnecessary. Figure 6.4 depicts the situation when a DOMINATOR node waits 10 seconds before switching back to DOMINATEE status.

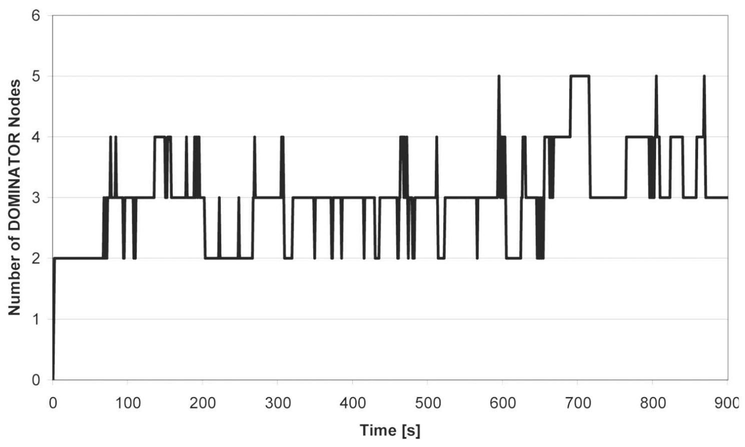

Figure 6.3: *DOMINATOR Switches Back Immediately to DOMINATEE*

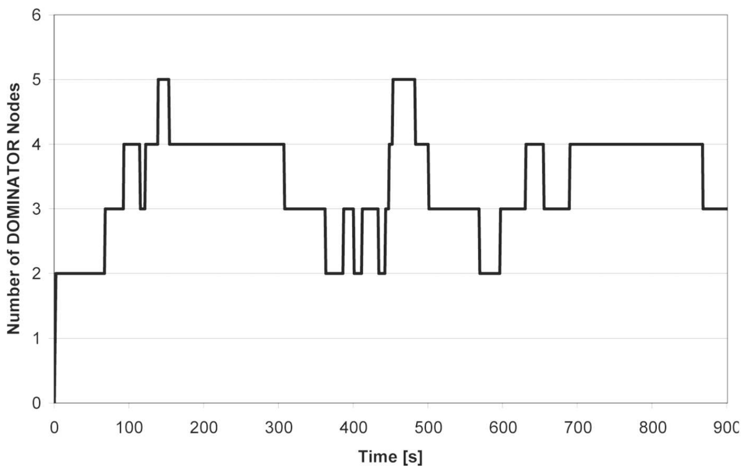

Figure 6.4: *DOMINATOR Switches Back to DOMINATEE After 10 Seconds*

Concerning the approximation factor of the DS, the results shown in Figure 6.3 and 6.4 are better than what we can see in Figure 6.2 because the number of DOMINATOR nodes in average is less decreasing the synchronization complexity for the application. On the other hand, the various changes force the clients to switch between servers more frequently which causes a lot of overhead. Therefore, we selected and used in all the scenarios presented here the solution when a DOMINATOR node never switches back to DOMINATEE status. To avoid the oscillation problem but still get a good approximation of MDS a possible improvement can be the use of node mobility prediction, as we discussed earlier in this thesis. For the evaluation results of applying our XCoPred prediction mechanism in the PBS algorithm see Section 6.4.6.

In Table 6.6, the number of DS changes in all the different scenarios is summarized. With the increasing number of nodes and the higher mobility level in the Test scenario the required changes are also increasing.

Scenario	**Min.**	**Max.**	**Avg.**	σ
School Yard w/ 15 nodes	0	1	0.6	0.5
Train w/ 15 nodes	0	4	1.6	1.3
Test w/ 15 nodes	0	2	0.8	0.6
School Yard w/ 35 nodes	2	5	3.6	0.8
Train w/ 35 nodes	3	7	4.4	1.2
Test w/ 35 nodes	1	9	5.6	2.6

Table 6.6: *Number of DS Changes*

- **Number of DOMINATOR Nodes:** This metric indicates the number of DOMINATOR nodes in the computed DS. We collected the minimum and maximum numbers of DOMINATOR nodes in case of the different scenarios (moreover the minimum, maximum, average values and the standard deviation of these numbers through the several runs with different seed values of a given scenario) in Table 6.7 and 6.8.

  Investigating Table 6.7 we can notice, that in all the scenarios the DS consists always of minimum 2 nodes as required. However, the average value in case of the Train scenario is a bit higher than in the other two scenarios (3.1 and 7.6) because the reduced transmission range and the narrow but long geometry require more DOMINATOR nodes. In regard

Scenario	**Min.**	**Max.**	**Avg.**	σ
School Yard w/ 15 nodes	2	3	2.4	0.5
Train w/ 15 nodes	2	4	3.1	0.5
Test w/ 15 nodes	2	3	2.4	0.8
School Yard w/ 35 nodes	2	4	3.2	1.1
Train w/ 35 nodes	6	13	7.6	1.9
Test w/ 35 nodes	5	8	7.0	0.6

Table 6.7: *Minimum Number of DOMINATOR Nodes*

Scenario	**Min.**	**Max.**	**Avg.**	σ
School Yard w/ 15 nodes	2	4	2.9	0.9
Train w/ 15 nodes	3	6	4.5	1.5
Test w/ 15 nodes	2	4	3.0	0.5
School Yard w/ 35 nodes	5	9	6.6	0.6
Train w/ 35 nodes	10	16	12.6	1.7
Test w/ 35 nodes	10	17	13.0	1.8

Table 6.8: *Maximum Number of DOMINATOR Nodes*

to the maximum number of DOMINATOR nodes (cf. Table 6.8) we can observe very similar tendency. Moreover, the number of DOMINATOR nodes is increasing with the increasing size of the geometrical area and the increasing number of nodes, such as in the Test scenario, which is not so surprising. The interesting thing is, however, that the Train and Test scenario show very similar behavior from this respect. This indicates to us, that the careful selection of the scenario and the mobility pattern is very important and they should capture real world situations as close as possible.

- **Range of the Node Weights:** In the simulations shown above, the node weights were distributed randomly in the range of [1..99]. With this choice, most of the nodes had different weight value assigned and the

PBS algorithm could determine the node priority solely based on its weight in the most cases. This reflects the situation when most of the nodes are assumed to have different capabilities to act as zone server. However, we have carried out some investigations also in that case when the majority of the nodes had the same node weight assigned being not able to differentiate between the node priorities only based on the weight value. Thus, the other criteria to determine the node priority, as described in Section 3.2, got more importance. We ran the simulations of the School Yard scenario with 35 nodes again distributing the node weights randomly only in the range of [1..4]. This time, we computed just the determination delay (cf. Table 6.9). We can see, that in this case to determine the servers requires slightly more time but the difference is not really significant. It indicates, that the PBS algorithm gives the same performance regardless how the node weights are assigned. In the upcoming section, we investigate another weight assignment/computation solution, our NWC mechanism, which reflects more realistically the node capabilities.

Node Weight Range	**Min.**	**Max.**	**Avg.**	σ
[1..99]	3	1,130	283	251
[1..4]	16	1,143	290	248

Table 6.9: *Determination Delay with Different Node Weight Ranges [msec]*

6.3 Service Profile Creation

As we discussed in Section 3.3.2, in our NWC mechanism different importance levels and thus different parameter weights are to be assigned to the node parameters in case of various service types. These parameter weights are collected in the service profile created by the service developer. To define this profile and assign the parameter weights we have developed a mechanism using factorial design and multi-objective optimization [16]. This experimental design is based on simulation and discussed in the following.

6.3.1 Pseudo Game Service Specification

We have defined a pseudo service, a real-time multiplayer game, via which we demonstrate here the procedure of the factorial design to assign the node parameter weights, and thus create the service profile.

The main characteristics of this pseudo service are the following:

- **Processing load generation:** The service generates a certain processing load on the nodes. This is associated with the game playing activity but also with the zone server role if the given node acts as a zone server. Unfortunately, in a simulation environment it is very hard to realistically simulate the changes of the processing load (CPU and memory load) generated by the service without implementing the service itself. Moreover, this load can be different on divers device types (e.g., on laptop, PDA or mobile phone). Thus, for sake of simplicity we assume that, whatever the device type is, being a zone server of our game has a minimum requirement of 500 MHz processor and 50 Mbyte free memory, since the game playing activity requires 700 MHz processor and 30 Mbyte free memory, respectively. Moreover, the game server role generates a 10 % CPU load and occupies 50 Mbyte memory continuously, since playing the game (being a client) generates a 30 % CPU load and occupies 30 Mbyte memory continuously.

  After this, the minimum requirements of the service are checked before selecting a zone server or deploying the game client on a node. Moreover, the generated processing load of the service can be easily computed now.

- **Energy consumption:** For sake of simplicity, in computing the consumed energy by the service we consider only the traffic generated by the game on the node. Thus, we define the energy consumption of the service as the linear function of the game traffic, i.e., every sent/received packet consumes 1 mW energy.

  We have implemented this simple mechanism in the simulator. Moreover, we used this strategy also in computing the battery level of the node taking into account only the traffic the node sent or received.

- **Generated traffic:** We define the game traffic, likewise in Section 6.2.1, as a full duplex 10 kbit/s CBR data flow (20 packet/second with 64 byte packet size) between the zone servers and their clients. For server to server synchronization, we assume that the traffic coming from the clients of the given server is simply forwarded to the other servers.

The characteristics of our pseudo game service are summarized in Table 6.10.

Property	**Value**
Min. CPU (server)	500 MHz
Min. CPU (client)	700 MHz
CPU load (server)	10 %
CPU load (client)	30 %
Min. free memory (server)	50 MByte
Min. free memory (client)	30 MByte
Memory load (server)	50 MByte
Memory load (client)	30 MByte
Energy consumption	1 mW per packet sent/received
Traffic (server ⇔ client)	Full duplex CBR - 10 kbit/s
Traffic (server ⇔ server)	Forwarded traffic coming from the clients

Table 6.10: *Characteristics of the Pseudo Game Service*

6.3.2 Factorial Design

Factorial design is an experimental design technique especially useful to measure the effects of a group of factors on the output of an experiment [35]. Applying this technique it is possible to determine that combination of the factor values which gives the best performance of the system. The complete analysis of the factors is called *Full Factorial Design*, where every possible combination of the factor values is created and analyzed. Usually, a full factorial design is expensive, time consuming and not possible to carry out due to the huge number of combinations to be investigated. However, in most of the cases some of the factor values can be eliminated intuitively which are not important or obviously have no or just very small influence on the system's output. In this case, we are talking about *Fractional Factorial Design*.

In NWC, we have five factors (the parameter weights) with three different values of each (see Section 3.3.2). Our objective is to determine via simulations that combination of the factor values, called service profile[15], which gives the best performance of the selected metrics (see Section 6.3.4).

Applying full factorial design for our case, $3^5 = 243$ different simulation runs would be required not counting the repetitions of the simulations with various seed values to increase the confidence level. This is neither practical nor feasible due to the high computation and time requirement of the simulations. Thus, we eliminated some of the factor values and carried out a fractional factorial design.

Fractional Factorial Design

In the NWC mechanism, three possible values can be assigned to each factor. The values are shown in Table 6.11 (0 - low; 0.5 - normal; 1 - high).

Weight of				
CPU	**Memory**	**Battery**	**Link Quality**	**Position**
0 ; 0.5 ; 1	0 ; 0.5 ; 1	0 ; 0.5 ; 1	0 ; 0.5 ; 1	0 ; 0.5 ; 1

Table 6.11: *Values of the Different Parameter Weight Factors*

To reduce the number of combinations and thus the number of simulations to run, we eliminated some of the values (see Table 6.12), taking into account the properties of real-time multiplayer games, from the further investigations based on the following intuitions:

- **CPU weight:** Real-time applications, hence real-time multiplayer games are usually processing intensive services. To avoid selecting weak nodes as zone servers, it is important to take into account the nodes' processing power in the node weight computation. Thus, we used two values, such as *normal* and *high*, for the CPU weight factor in our factorial design.
- **Memory weight:** The same arguments hold for the memory weight

[15]Note that this combination can be different in case of different network scenarios. Thus, using a static service profile cannot provide a general ideal combination for all cases. The use of some dynamic service profile would solve this problem, what remains as future work.

Comb. No.	CPU	Memory	Battery	Link Quality	Position
1	1	1	0	1	1
2	1	1	0	1	0.5
3	1	1	0	0.5	1
4	1	1	0	0.5	0.5
5	1	0.5	0	1	1
6	1	0.5	0	1	0.5
7	1	0.5	0	0.5	1
8	1	0.5	0	0.5	0.5
9	0.5	1	0	1	1
10	0.5	1	0	1	0.5
11	0.5	1	0	0.5	1
12	0.5	1	0	0.5	0.5
13	0.5	0.5	0	1	1
14	0.5	0.5	0	1	0.5
15	0.5	0.5	0	0.5	1
16	0.5	0.5	0	0.5	0.5

Table 6.12: *Considered Combinations of the Parameter Weight Factor Values*

factor, as well. So, we investigated two levels of this factor, like *normal* and *high*.

- **Battery weight:** The available battery level can be important for energy intensive or long-life services. However, ad hoc games do not have special energy requirements comparing to other applications and they are usually short time games mainly to kill waiting or travelling time. Thus, we neglected this factor from our investigations and assigned always the *low* weight level.

- **Link Quality weight:** Real-time multiplayer games have strict QoS requirements towards the underlying network, as we discussed in Section 6.1.1, and in this respect the quality of the nodes' links are important in the zone server selection. Hence, we used again two values, *normal* and *high*, for this factor in our factorial design.

- **Position weight:** The creation of a small DS has the advantage of reducing the inter server synchronization delay and overhead traffic. Clearly, this can be achieved by selecting nodes with good network position into the DS. Thus, the position parameter is important to take into account and we used the *normal* and *high* values of this factor in the investigations.

As we can see in Table 6.12, we had managed to reduce the number of combinations of the different parameter weight factor values to 16 what we finally considered in our factorial design. Thus, we reduced the minimal number of simulations to run, not counting the repetitions, from 243 to 16.

6.3.3 Simulation Settings

To select the best combination of the factor values for the given service, we did run simulations. In these simulations, we investigated 16 combinations (cf. Table 6.12) and picked that combination as the service profile which had given the best performance of the measured metrics.

As earlier, we used the NS-2 network simulator and its wireless extensions for the simulations. We applied the PBS algorithm to compute and maintain the zone server set in the simulations, but this time the node weights were not assigned randomly rather computed by our NWC mechanism implemented as an extension to PBS (cf. Section 5.1.3). We used the same basic settings as in the previous simulations, thus each mobile node shared a 2 Mbit/s radio channel with its neighboring nodes using the two-ray ground reflection model, IEEE 802.11 MAC protocol and AODV routing protocol.

This time, we did run the simulations using only two different scenarios:

- **School Yard Scenario:** We used the same settings in this scenario, as in case of the PBS evaluation. Thus, we set the area of the school yard to 400x400 m^2, the transmission range of every device to 250 meters and the node density to 15 nodes from which 10 nodes were participating in the game session in average. The movements of the nodes were modelled by using the RWP mobility model with the same settings as earlier. The speed of the nodes was uniformly distributed in the range of 0-6 km/h.

- **Test Scenario:** Again, we used the same settings as in case of the PBS evaluation. We set the geometric area of this scenario to 800x800 m^2, the transmission range of the devices to 250 meters and the node density to 35 nodes from which 25 nodes were participating in the game session in average. All the nodes were set to move around with uniformly distributed speed in the range of 0-30 km/h using again the RWP mobility model.

Variable	**Setting**
Scenario	School Yard (400x400 m^2), Test (800x800 m^2)
Number of nodes	School Yard - 15 with 10 player nodes
	Test - 35 with 25 player nodes
Node weight	In the range of $[0..1]$, computed by NWC
Duration of the simulation time	900 sec
Number of repetitions	10
Game joining and leaving points	Randomly distributed during the simulation time
Used mobility model	Random WayPoint
Game traffic	Server $\Leftrightarrow$ client: full duplex CBR - 10 kbit/s
	Server $\Leftrightarrow$ server: forwarded traffic coming from the clients
Background traffic	CBR - 5 kbit/s
	School Yard - 5, Test - 15 parallel connections
Node settings: CPU type	500 - 1000 MHz
Memory	256 or 512 MByte
Battery capacity	1500 - 5000 mW

Table 6.13: *Settings of the Simulations to Create the Service Profile*

Table 6.13 summarizes the main simulation settings. We had repeated every simulation 10 times using different seed values then averaged the results and computed the 95 % confidence interval of the average. This gives 10 simulations per scenario and combination, and 320 simulations in total. We set the duration of the game session, and thus the simulation time, to 900 seconds. The players' game joining and leaving points in time were randomly distributed during the simulation. To simulate the game traffic, we used the characteristics of the pseudo game service we specified above (cf. Section 6.3.1). Thus, we generated a full duplex 10 kbit/s CBR data flow between the servers and their clients and as synchronization traffic between the servers every server forwarded the traffic to the other servers coming from its clients. In the School Yard scenario, the background traffic was generated by 5 parallel connections being active at the same time during the whole simulation between any two random nodes (consuming the same amount of energy as in case of the game service but not generating CPU and memory load on the given node). Per connection, the sender produced a 5 kbit/s data flow for 30 seconds, then a new connection was established. In the Test scenario, we increased the number of parallel connections being active at the same time to 15.

For computing the parameter values in NWC, besides the service specification we have to specify also the node properties. Thus, in the simulations we used nodes with 500 - 1000 MHz CPU regardless the exact processor type, 256 or 512 MByte memory and 1500 - 5000 mW battery capacity. The values were randomly assigned to the nodes at the beginning of the simulations.

6.3.4 Simulation Results

The simulation results are depicted in Figure 6.5 using the School Yard scenario and in Figure 6.6 using the Test scenario, respectively. The figures show the average values of the applied metrics and their 95 % confidence interval in case of the investigated factor value combinations. To measure the performance of the different combinations, we used the following metrics:

- **Number of DS Nodes:** This metric indicates the number of DOMINATOR nodes in the computed DS. The size of the DS is highly influenced by the node weight assignment, i.e., how the weight numbers are distributed in the network. If only the node degree is considered in the DS creation, which is usually the case in other DS computation algorithms, the computed DS tends to be a good MDS approximation. However, in

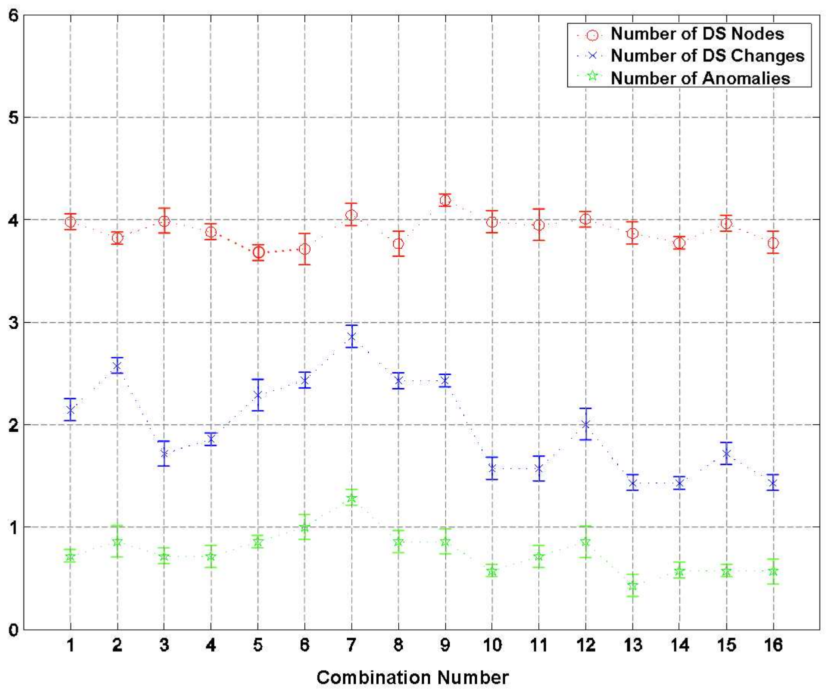

Figure 6.5: *Simulation Results Using the School Yard Scenario*

our case the node weight depends not only on this but also on several other parameters. Thus, the size of the computed DS can be much bigger than the size of MDS even if the position parameter of the node has a *normal* or *high* parameter weight value assigned in the node weight computation.

Using the School Yard scenario, we always observed a quite small DS, see Figure 6.5, which can be explained basically by the scenario properties. The simulated geometric area in this scenario is 400x400 m^2 and the network contains 15 nodes, which results in the complete coverage of all the nodes by a DS consisted of usually 3 to 5 nodes. The best performance regarding this metric was produced by combination 5. Note that in this combination *high* parameter weight value is assigned to the

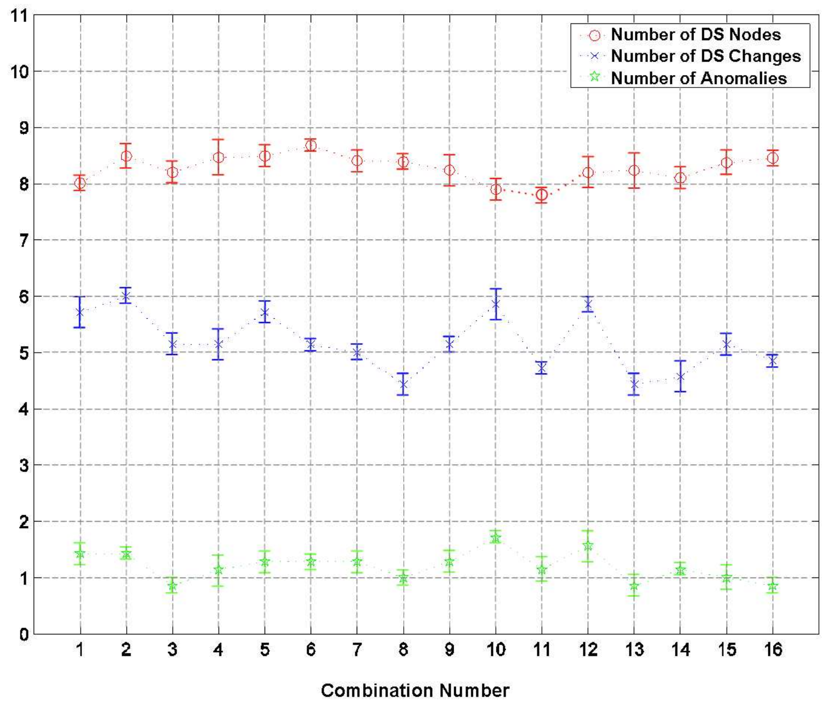

Figure 6.6: *Simulation Results Using the Test Scenario*

position parameter of the node.

In the Test scenario, we observed slightly different results (see Figure 6.6). The main difference is the increased number of the DOMINATOR nodes. The geometric area of this scenario is larger than in the previous case (800x800 m^2), the network consists of more nodes (35) and the nodes are moving with higher speed (in the range of 0-30 km/h). Due to these reasons the computed DS contains usually 6 to 10 nodes. In this scenario, combination 11 showed the best performance from the viewpoint of the DS size. And again, *high* parameter weight value is assigned to the position parameter of the node in combination 11.

- **Number of DS Changes:** This metric measures the stability of the DS.

It shows how often the initial DS needs to be changed during the simulation from a global viewpoint. As earlier, in our implementation a DOMINATOR node remains a DOMINATOR even if its clients are covered by other DOMINATOR nodes, and switches back to DOMINATEE status only if there are no DOMINATEE nodes left in its neighborhood.

We can observe that the number of DS changes shows the biggest difference when the two investigated scenarios are compared. The School Yard scenario showed a very stable DS performance (cf. Figure 6.5) due to the relatively low level of network dynamics in this scenario. In most of the cases, only a few or no changes occurred during the whole simulation. Combination 13 produced the best performance with an average of 1.43 DS changes, in which combination *normal* parameter weight value was assigned to the CPU and memory parameters whereas *high* parameter weight value was assigned to the link quality and position parameters, respectively.

Using the Test scenario, the same combination, i.e., combination 13, showed the best performance (cf. Figure 6.6). However, in this scenario the observed stability level of the computed DS was obviously smaller in case of all investigated combinations than in the School Yard scenario resulting in a much higher number (usually 4 to 7) of DS changes.

- **Number of Anomalies:** This metric indicates how often DOMINATOR nodes are not able to support properly the running of the deployed service. It counts how many times the available CPU, memory or battery capacity of DOMINATOR nodes drops below a preset threshold value (10 % in case of CPU and memory, 5 % in case of battery, see Section 3.3.2) in global. If such a thing happens, an anomaly is generated triggering a new DS selection round.

  We observed very low number of anomalies with small variations in both scenarios. All of these anomalies were created by the dropping battery power. This reflected our expectations because with limited resource properties only a small number of nodes had been used in the simulations as we had assigned the property values randomly to the nodes (cf. Section 6.5.1). Moreover, the used services in the simulations (the game service and the generated background traffic) were not processing intensive services. Only the available battery power could drop below the preset threshold during the simulation time when a node with small initially assigned battery capacity had to forward relatively high amount of traffic (cf. Section 6.3.1 and Section 6.5.1).

The number of anomalies was usually in the range of 0 to 1 using the School Yard scenario (see Figure 6.5) and in the range of 0 to 2 using the Test scenario (see Figure 6.6), respectively. Again, combination 13 showed the best performance in case of the both scenarios.

Note that in the simulations, we monitored the properties of the game traffic, as well. We wanted to see whether they were in compliance with the QoS requirements of real-time multiplayer games (cf. Section 6.1.1). We observed, that to keep the end-to-end communication delay and delay jitter below the required limits (i.e., to keep the delay below 150 msec and the jitter as low as possible) was easily achievable in our scenarios using the standard IEEE 802.11 MAC protocol and AODV routing protocol. However, we could not always keep the packet loss rate below the maximum allowable 3 – 5 %. To comply also with the packet loss rate requirement, we should introduce and implement Quality of Service mechanisms instead of merely using best effort type communication with contention based medium access. We have investigated and elaborated also this issue (see [59–61] for more details) but we do not discuss it further in this thesis.

Determine the Service Profile

We used several metrics in the performance investigation of the different factor value combinations and obviously not always the same combination showed the best performance. Thus, the selection of the combination appropriate for the given service is a trade-off based on how important the various metrics are for the service developer. To be able to compare the different combinations, we used the *multi-objective optimization* technique [35].

This technique takes each objective function (metric) and multiplies it by a *weighting coefficient* w_i. The modified functions are then added together to obtain a single cost function as given in Equation 6.1:

$$f(x) = \sum_{i=1}^{k} w_i f_i(x), \tag{6.1}$$

where $0 \leq w_i \leq 1$ and $\sum_{i=1}^{k} w_i = 1$. The objective functions, i.e. $f_i()$-s, must be optimized before computing the cost value. Moreover, the weighting coefficients must be determined beforehand, thus the service designer is expected to set these values according to how important the various metrics are for him/her.

In our example, the objective functions represent the computed average values of the metrics and we set the weighting coefficients according to Table 6.14.

Metric	**Weight**
Number of DS Nodes (NUM_{nodes})	2/10
Number of DS Changes ($NUM_{changes}$)	5/10
Number of Anomalies (NUM_{anoms})	3/10

Table 6.14: *Weighting Coefficients of the Used Metrics*

After this, the cost value of the different combinations can be computed in the following way:

$$f(cn) = \frac{2 \cdot NUM_{nodes}(cn) + 5 \cdot NUM_{changes}(cn) + 3 \cdot NUM_{anoms}(cn)}{10}, \quad (6.2)$$

where *cn* represents the combination number. Note that before the cost value is computed all the metric values are normalized. This way the cost value of the different combinations can be easily compared, and the combination with the lowest cost value can be selected as the service profile.

Figure 6.7 and 6.8 depict the cost values of the different combinations together with their 95 % confidence interval using the School Yard and the Test scenario, respectively.

We can see, that in both scenarios combination 13 has the lowest cost value. Thus, we can select the parameter weight values represented by combination 13 (cf. Table 6.15) as the service profile of real-time multiplayer games.

Weight of				
CPU	**Memory**	**Battery**	**Link Quality**	**Position**
0.5	0.5	0	1	1

Table 6.15: *Service Profile of Real-Time Multiplayer Games*

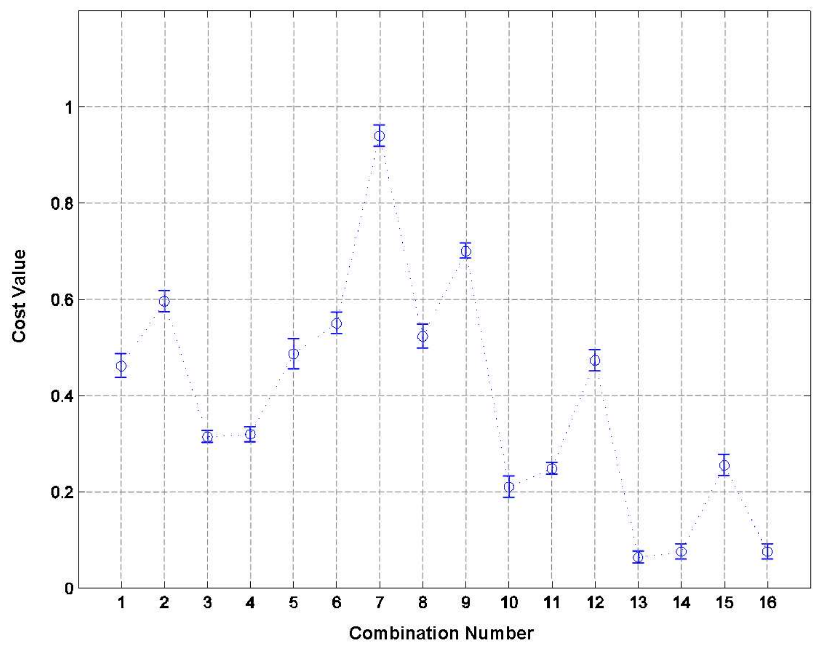

Figure 6.7: *Cost Values Using the School Yard Scenario*

Note that the selected metrics to evaluate our factorial design were up to our choice and any other metrics could have been used also. This is true for the weighting coefficient assignments, too. Moreover, recall that the best performing combination of the factor values can be different in case of different network scenarios. For example, it could have happened that we got a different best performing combination using the School Yard scenario than using the Test scenario. Thus, using a static service profile cannot provide a general ideal combination for all the cases. The use of some dynamic service profile would solve this problem, whose investigation remains as future work.

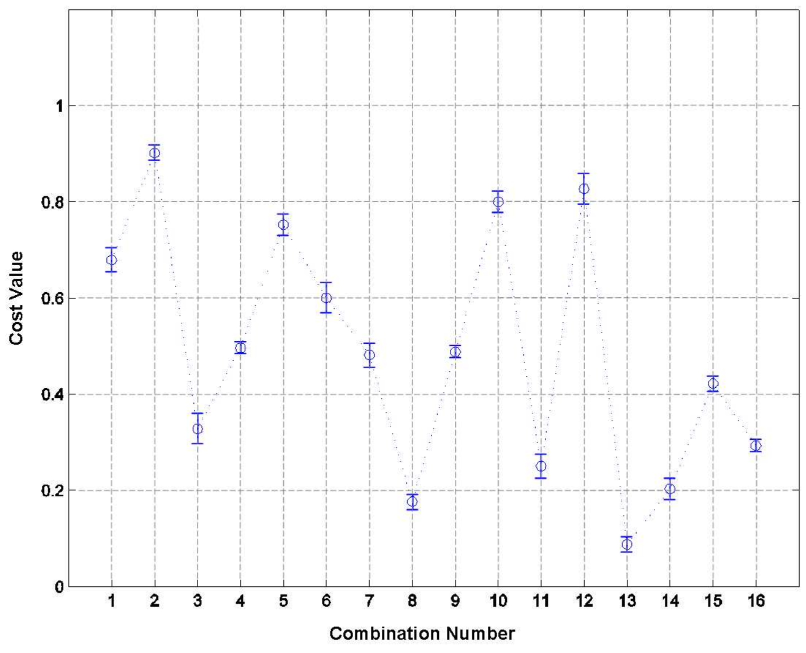

Figure 6.8: *Cost Values Using the Test Scenario*

6.4 Evaluation of the XCoPred Prediction Mechanism

In this section, we evaluate XCoPred via simulations. The evaluation starts with determining the optimal choice of the design parameters described in Section 4.2. First, we show how the number of training samples for the Kalman filter's autoregressive model was set. Then the query order and the match threshold parameters are investigated, both have influence on the number of predictors and on the accuracy of the predicted SNR values. After having set the parameters, we evaluate the accuracy of the prediction using the Random WayPoint and the Freeway mobility models. And finally, to illustrate the application of our prediction method we show how XCoPred can improve the performance of the PBS algorithm.

6.4.1 Simulation Settings

This time again, we have implemented XCoPred in the NS-2 network simulator (cf. Section 5.1.4). For the simulations, we used NS-2 and its wireless extensions with the same basic settings as in the previous investigations, thus each mobile node shared a 2 Mbit/s radio channel with its neighboring nodes using the two-ray ground reflection model, IEEE 802.11 MAC protocol and AODV routing protocol.

Concerning our prediction mechanism, there are two critical factors for getting realistic and meaningful simulation results. First, a realistic model of the SNR is required. Second, the simulations have to be run with mobility models representing a wide range of different possible physical environments.

SNR in NS-2

As the propagation model, we have used the shadowing model which comes with NS-2 (cf. [62]). The shadowing model calculates the received signal power $P_r(d)$ at distance d according to

$$\frac{P_r(d)}{P_r(d_0)} = -10\beta log(\frac{d}{d_0}) + X_{dB}[dB], \tag{6.3}$$

where d_0 is a reference distance, β is the path loss exponent and X_{dB} is a random variable with Gaussian distribution, zero mean and standard deviation σ_{dB}. Both β and σ_{dB} are depending on the physical environment. Typical values for them can be found in [62]. We have chosen $\beta = 4$ and $\sigma_{dB} = 7$ as values representing office environments.

In order to account for environmental noise and receiver noise, another random variable with Gaussian distribution was added to the SNR values. The mean was set to -90 dBm with a standard deviation of 4 dBm. As for interference, we used the collision detection model of NS-2. When a packet arrives, the receiving function simply checks, whether some other packet is currently being received. If this is the case, the signal power of this packet is accounted as interference.

More details can be found about NS-2's SNR implementation in Appendix A.2.

Mobility Models

As stated in Assumption 3 in Section 4.1.1, we have no a priori information about the physical environment in which the network is located. Thus, in order to get simulation results which are meaningful for a broad range of physical environments, we have used two different mobility models with vastly different node behaviors.

We have chosen the Random WayPoint model as a representative of node mobility showing high level of randomness. The simulation setup of our RWP scenario is summarized in Table 6.16.

Variable	**Setting**
Mobility model	RWP
Max. speed	5 m/s
Pause time	5 sec
Simulated area	1000x1000 m^2
Number of nodes	10

Table 6.16: *Simulation Setup Using the RWP Mobility Model*

As a representative of a mobility model which shows a clear structure concerning the behavior of the nodes, we have chosen the Freeway mobility model (see Appendix A.1.2 and [63]). Using this model it is possible to simulate the traffic on a highway with high level of realism. The parameters we used in case of this model are shown in Table 6.17.

Variable	**Setting**
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

Table 6.17: *Simulation Setup Using the Freeway Mobility Model*

6.4.2 Setting the Kalman Filter Parameters

As discussed in Section 4.2.1 previously, the Kalman filter parameters α and c can be computed according to Equation 4.10 and 4.11. In order to set them 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. But we have not discussed yet the training data order O, i.e., how many values in the past should be used as training data. Using only a small number of training measurements for creating the model should give better results than choosing a large training data order, since this creates a more accurate model of the actual link state. As the Kalman filter operates only with 1-step-ahead predictions, this is a different case than predicting the link quality changes which requires long term predictions. Thus, for the Kalman filter a model taking into account only the recent past of the link should be created.

Simulation Results

To determine the optimal choice of the training data order, we ran simulations using the RWP scenario described above. We had repeated every simulation 10 times using different seed values then averaged the results and computed the 95 % confidence interval of the average. Each simulation ran for 300 seconds. At every second a link model was created for each of the links in the network giving enough data for evaluation. We ran the simulations with different model orders in the range from 3 to 13 and compared the quality of the models[16]. This means 110 simulation rounds with the repetitions in total. As the measure of the quality of autoregressive models, the coefficient of determination R^2 is widely used (cf. [39]) and we also applied this. It is defined as

$$R^2 = 1 - \frac{\sum_i \hat{e_i}^2}{\sum_i x_i^2} = 1 - \frac{\sum_i (\hat{x}_i - x_i)^2}{\sum_i x_i^2}, \tag{6.4}$$

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 fall in the interval of $[0..1]$, where 0 means a bad fit and 1 means a perfect fit of the model.

[16]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.

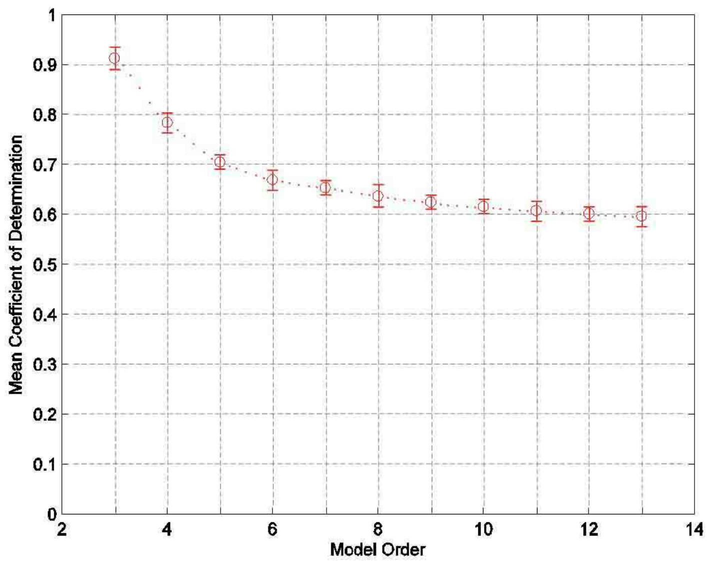

Figure 6.9: *Model Order Investigation of the Kalman Filter*

The resulting average R^2 values, depending on the training data order, together with the 95 % confidence interval of the values are plotted in Figure 6.9. The plot approves our intuitively assumed decrease of the coefficient of determination. The best fit of the model to 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, we have 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. The choice of 7 is a reasonable value, as the coefficient of determination with $R^2 = 0.65$ is still high enough and the probability of unstable predictions is already much smaller.

6.4.3 Setting the Prediction Parameters

For the prediction part, the parameters to be discussed are the query order and the match threshold. They both have a direct influence on the prediction accuracy and should be set in a way that gives as accurate predictions as possible.

Query Order

The query order o (see Definition 4.2.3) is related to how long a pattern in the movement of the nodes is assumed to be. However, as there are no clearly defined patterns with a unique length in the training data, the optimal query order cannot be set analytically, rather it has to be chosen by other means, e.g. 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 trade-off between environments showing short patterns and those showing longer patterns.

The two main effects of the query order on the prediction accuracy are:

- 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 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 degraded 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 4.2.2) has to be applied.

Match Threshold

The match threshold γ_{min} (see Definition 4.2.4) 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. 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. Its influence is quite similar to the influence of the query order:

- A small match threshold leads to a big number of predictors, as the match need not be perfect. However, a too small threshold can be harmful, since 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 the query. Again, this is risky as the number of predictors may be too small or no predictor may be found at all.

Simulation Results

Our final goal is to optimize the accuracy of the prediction influenced by both of the query order and the match threshold. Thus, we have run simulations with several possible combinations of these parameters to find the optimal one.

For these simulations we used both of the RWP and the Freeway scenario described above. With each of the scenarios we had repeated the simulations 10 times using different seed values then averaged the results and computed the 95 % confidence interval of the average. In case of the RWP scenario, the simulations ran for 630 seconds, split in 600 seconds for collecting training data and 30 seconds for checking the accuracy of even the longest (30-seconds-ahead) predictions made at time point 600. In case of the Freeway scenario, the simulations ran only for 330 seconds, as the training phase did not have to be as long as in the RWP case because the observed SNR patterns were more limited. Thus, we used 300 second training phase and 30 seconds for checking the accuracy.

To get some insight of how these two parameters, the query order and the match threshold, affect the prediction accuracy we investigated the following combinations. The match threshold has been varied in the interval of [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. Thus, together with the repetitions we ran 450 simulations per scenario and 900 simulations in total.

Figure 6.10 and 6.11 show the average number of predictors per prediction depending on the query order and the match threshold (this time we did not depict the confidence intervals for sake of visibility). The figures confirm,

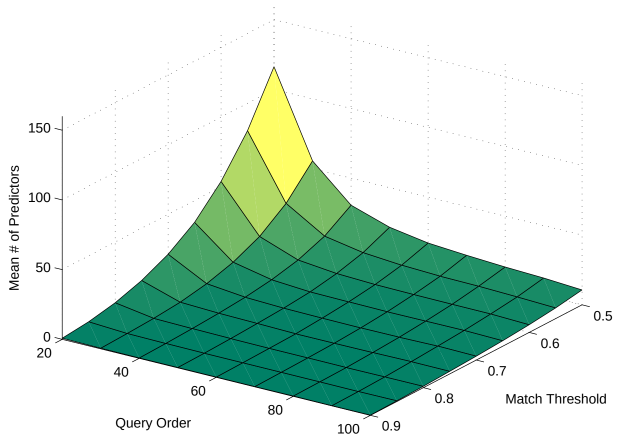

Figure 6.10: *Average Number of Predictors Using the RWP Model*

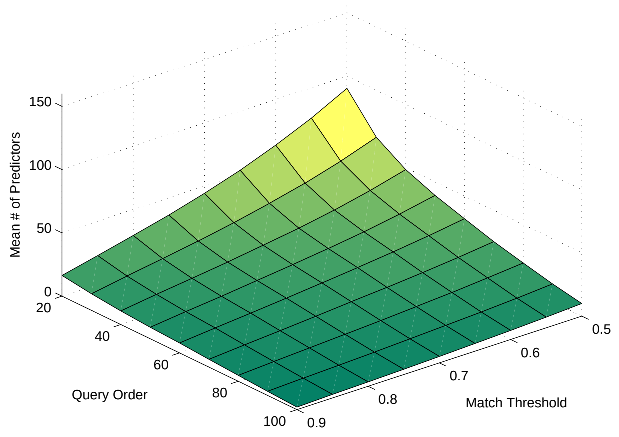

Figure 6.11: *Average Number of Predictors Using the Freeway Model*

that small query order and small match threshold lead to high number of predictors. The absolute numbers are not relevant, as they depend highly on the length of the training data, the number of links the nodes have and other settings. However, what we can extract from the plots are some upper bounds of the parameters. 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. The same can be observed for query order values in the range of [80..100], where the number of predictors gets very small.

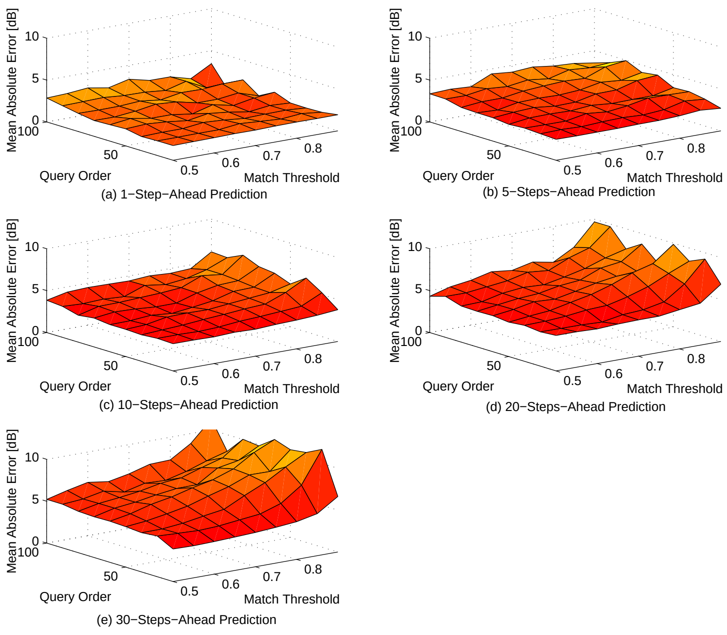

Figure 6.12: *Mean Prediction Error Using the RWP Model*

We have also investigated the average absolute prediction error shown in Figure 6.12 using the RWP model and in Figure 6.13 using the Freeway model. In these figures[17], the prediction error in dB, depending on the query

[17]Note that in these figures the scaling order was reversed on the Query Order and Match Threshold axes compared to Figure 6.10 and 6.11 for sake of better representability.

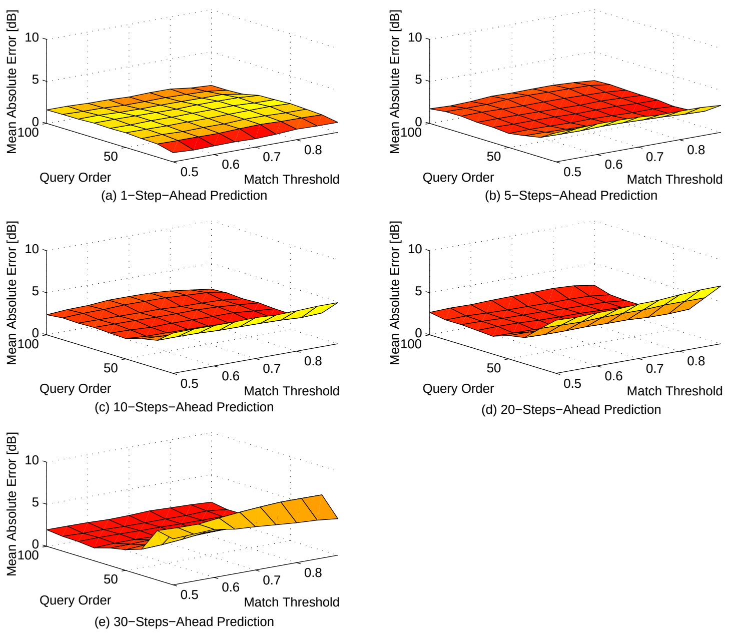

Figure 6.13: *Mean Prediction Error Using the Freeway Model*

order and the match threshold, is plotted for different prediction orders, like 1, 5, 10, 20 and 30-steps-ahead predictions, respectively. First considering the RWP model, the 1-step-ahead prediction error does not really depend on the two parameters and is constant with a value about 2 dB. For longer term predictions above 10 steps, the error starts to significantly increase for higher match thresholds. The reason is that for high thresholds no predictors can be found and the fallback model (see Section 4.2.2) is used. These results suggest to choose a small match threshold of about 0.5.

Considering the prediction error using the Freeway model shows a different situation. In this case, the error values are smaller due to the clearer structure of the patterns. For short term predictions, we can observe an average error of about 1 dB which is again independent of the query order and the match threshold. However, for longer term predictions we can see an important difference to the RWP case. Where, using the RWP model, the error depends 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. If a situation in the past is really similar to the current situation, the patterns will match even with a high match threshold. The other effect, the error increase with too short query orders, is a sign that a query order of below 60 is too small for the Freeway case.

As a trade-off of the observed effects using the RWP and the Freeway model, we have chosen a query order of 70 and a match threshold of 0.5 for the further evaluations of XCoPred. These values led in both scenarios to good results. Note that if we have some knowledge about the physical environment of the network and the mobility patterns of the nodes, these parameters may be tuned accordingly in order to get the best possible results.

6.4.4 Complete Setting of the Design Parameters

Table 6.18 shows the complete setting of the XCoPred prediction mechanism's design parameters.

Design Parameter	**Value**
Kalman filter parameters:	
α	computed according to Eq. 4.10
c	computed according to Eq. 4.11
process noise covariance S	49
training data order O	7
Prediction parameters:	
measurement interval T	1 sec
no. of stored measurements N	2048
query order o	70
match threshold γ_{min}	0.5
prediction order l	depends on the application

Table 6.18: *Complete Setting of XCoPred's Design Parameters*

6.4.5 Prediction Accuracy

In order to evaluate the prediction accuracy in more detail, we used again the RWP and the Freeway mobility models. The simulation time was set to 630 seconds in the RWP case and to 330 seconds in the Freeway case. We had repeated every simulation 10 times with different seed values then averaged the results and computed the 95 % confidence interval of the average again. This gives 300 simulations per scenario and 600 simulations in total.

Figure 6.14 and 6.15 depict in more detail the dependence of the average absolute prediction error on the prediction order using the RWP and the Freeway mobility models with query order 70 and match threshold 0.5 (the confidence intervals are not shown in the figures for sake of visibility). In the RWP case, for a 1-step-ahead prediction the error is about 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 maximum value of around 3 dB for a 12-steps-ahead prediction. Then it starts to decrease again, until it reaches a value of about 2 dB for a 30-steps-ahead prediction. This decrease stems from the rather short lifetime of the links, especially between nodes driving in opposite directions. As more and more links break, the predictions in average get more accurate. The reason is that the absence of a link can usually be predicted without any error whereas predicting the exact SNR value of an existing link will always contain some error.

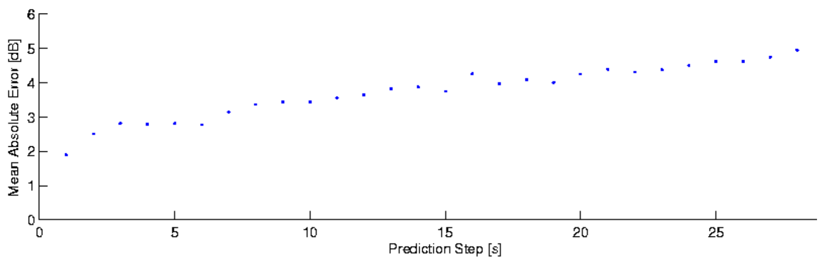

Figure 6.14: *Mean Prediction Error for Different Prediction Length Using the RWP Mobility Model, Query Order 70 and Match Threshold 0.5*

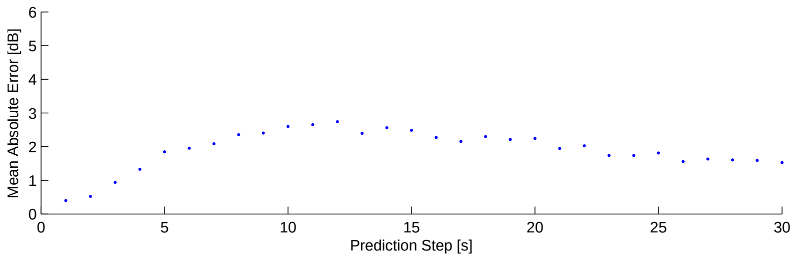

Figure 6.15: *Mean Prediction Error for Different Prediction Length Using the Freeway Mobility Model, Query Order 70 and Match Threshold 0.5*

6.4.6 Applying XCoPred in the Priority Based Selection Algorithm

To illustrate the application of XCoPred we had integrated it into the PBS algorithm (cf. Section 4.3) and ran another round of simulations to see whether it could improve the stability of the computed DS.

Simulation Results

Again, we used both the RWP and the Freeway mobility models with simulation settings given in Table 6.16 and 6.17 and repeated the simulations 10 times with different seeds to increase the confidence level then averaged the results. This means 70 simulation rounds per scenario and 140 simulations in total. In case of the RWP scenario, the simulations ran for 600 seconds, using 300 seconds for gathering training data. At time 300, the DS was computed and then maintained for the next 300 seconds. With the Freeway scenario, the simulations ran for 300 seconds, split in 200 second training phase and 100 seconds for maintaining the DS. The parameters of the prediction algorithm were set according to our investigation results in Section 6.4.3. Thus, we set the query order to 70 and the match threshold to 0.5.

The results of these simulations are summarized in Table 6.19 and 6.20 for the RWP case and for the Freeway model, respectively. The tables show the average number of server nodes and the average number of DS changes depending on the link stability criterion (see Section 4.3). For instance, a link stability of $l = 30$ implies that a link is assumed to be stable if it is still available during the upcoming 30 seconds. The first row of the tables shows the results without using prediction ($l = 0$). In the following rows, the stability criterion got more and more strict ($l = 10..60$). Additionally to the number of servers and DS changes, the standard deviation and the percentage values are also given with values without prediction set to 100 %.

l	No. of Servers	σ	%	No. 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 6.19: *Average No. of Servers and DS Changes With the RWP Model*

l	No. of Servers	σ	%	No. of DS Changes	σ	%
0	11.80	0.43	100	71.2	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 6.20: *Average No. of Servers and DS Changes With the Freeway Model*

The results using the RWP model show that the number of servers increases applying the link stability criterion by more than 35 % from the value of 3.45 up to around 4.7. This larger number is the cost which has to be paid for the increased DS stability. Considering the DS changes, the average number could be reduced applying the stability criterion from 20.4 to 16.6. This is a 19 % reduction in the optimal case of requiring 40 seconds (or 30 seconds which gives the same reduction) of link stability.

In the Freeway scenario, the results in Table 6.20 look similar but better. In general, the number of servers and the number of DS changes are higher than in case of using the RWP model, because the mobility of the nodes is higher. A nice difference to the RWP scenario is, that the cost of a more stable DS is much smaller and the increase of stability is higher. In the optimal case of $l = 40$, the number of DS changes could be reduced by 25 % from 71.2 to 53.4, while the number of servers is only increased by 11 % from the average of 11.8 to 13.04.

In our case dealing with real-time services, decreasing the number of DS changes is usually worth the price of having a few more servers because changes in the Dominating Set are expensive. A change in the DS generally involves service disruption for at least the nodes that lose the connection to their servers. Additionally, new server selection rounds present large communication overhead, which should be avoided whenever possible. However, in case of services which require huge synchronization overhead by default between the servers it might be desirable to have fewer servers and more DS changes instead. In such a case, the link stability criterion should not be used.

6.5 Simulation Based Comparison of Service Management Architectures

We have run another round of simulations to see, whether the zone-based service management model with PBS to select the zone servers really offers a reasonable compromise while solving the main problems of the other two common models, such as the reduced fault tolerance of the centralized client/server, and the limited scalability of the fully distributed peer-to-peer model. Thus, in the final part of our evaluation we have compared the performance of these three different service management models.

6.5.1 Simulation Settings

We used again the NS-2 network simulator and its wireless extensions for the simulations with the same basic settings as earlier. Thus, each mobile node shared a 2 Mbit/s radio channel with its neighboring nodes using the two-ray ground reflection model, IEEE 802.11 MAC protocol and AODV routing protocol.

This time, we did run the simulations using only one scenario, the School Yard scenario, but with four different node densities. We set the area of the school yard to 800x800 m^2, the transmission range of every device to 250 meters and the node density to 15, 30, 45, 60 nodes from which always 2/3 of the nodes (i.e., 10, 20, 30, 40, respectively) were participating in the game session in average. The movements of the nodes were modelled by using the RWP mobility model with the same settings as earlier. The speed of the nodes was uniformly distributed in the range of 0-6 km/h.

We had repeated every simulation 10 times using different seed values then averaged the results and computed the 95 % confidence interval of the average. This gives 10 simulations per node density and service management architecture, and 120 simulations in total. We set the duration of the game session, and thus the simulation time, to 900 seconds. The player nodes' game joining and leaving points in time were randomly distributed during the simulated time. In case of the client/server model, the server node was randomly selected in every simulation. Moreover, in the zone-based model we used PBS extended with NWC and XCoPred to select the server nodes.

As service management traffic, we used periodically sent 'alive' messages between the server and its clients in case of the client/server model, between all the peer nodes in case of the peer-to-peer model, and between the nodes and their neighbors and even between all the servers in case of the zone-based model generating roughly 400 bit/s full duplex CBR traffic per connection. Moreover, in the zone-based model the PBS messages were also counted as management traffic. To simulate the game traffic, we used the characteristics of the pseudo game service we specified above (cf. Section 6.3.1). Thus, we generated a full duplex 10 kbit/s CBR data flow in case of the client/server model between the server and its clients, in case of the zone-based model between the servers and their clients, and in case of the peer-to-peer model between the peer nodes. As synchronization traffic between the servers of the zone-based model, every server sent 1/3 of the game traffic received from its clients to all the other servers. The background traffic was generated in case of the 15, 30, 45 and 60 network nodes by 5, 15, 25 and 35 parallel connections being active at the same time during the whole simulation between any two random nodes, respectively. Per connection, the sender produced a 5 kbit/s data flow for 30 seconds, then a new connection was established. Note that all the service management, game and background traffic was sent as unicast traffic in the simulations.

Table 6.21 summarizes the main simulation settings discussed above.

Variable	**Setting**
Scenario	School Yard (800x800 m^2)
Number of nodes	15 (10 players), 30 (20 players), 45 (30 players), 60 (40 players)
Service management	Client/server, Zone-based, Peer-to-peer
Duration of the simulation time	900 sec
Number of repetitions	10
Game joining and leaving points	Randomly distributed during the simulation time
Used mobility model	Random WayPoint
Management traffic	
Client/server, peer-to-peer	Periodic ‘alive’ messages
Zone-based	Periodic ‘alive’ + PBS messages
Game traffic	
Client/server	Server ⇔ client: full duplex CBR - 10 kbit/s
Zone-based	Server ⇔ client: full duplex CBR - 10 kbit/s
	Server ⇔ server: 1/3 of the game traffic received from the clients
Peer-to-peer	Peer node ⇔ peer node: full duplex CBR - 10 kbit/s
Background traffic	CBR - 5 kbit/s
	5 (15 nodes), 15 (30 nodes), 25 (45 nodes), 35 (60 nodes) parallel connections

Table 6.21: *Simulation Settings to Compare the Three Service Management Models*

6.5.2 Simulation Results

We investigated the performance of the different service management models from the viewpoint of the service and the network. To compare the different models, we used the following metrics:

- **Number of Service Interruptions:** This measure indicates how many times the game service was interrupted during the simulation from a global viewpoint. We counted an interruption event if the majority of the nodes participating in the service had to wait for a service update more than 0.5 seconds (approximately 3 times the maximum acceptable round trip delay in real-time games). Table 6.22 shows the averaged results and their standard deviation we got applying the different service management models in the scenarios with various node densities.

Scenario / Man. Model	**Client/Server**	**Zone-Based**	**Peer-to-Peer**
15 Nodes	5, $\sigma = 0.51$	0, $\sigma = 0$	0, $\sigma = 0$
30 Nodes	4, $\sigma = 0.44$	0, $\sigma = 0$	0, $\sigma = 0$
45 Nodes	3, $\sigma = 0.40$	0, $\sigma = 0$	0, $\sigma = 0$
60 Nodes	3, $\sigma = 0.38$	0, $\sigma = 0$	0, $\sigma = 0$

Table 6.22: *Average Number of Service Interruptions*

As the table indicates, we observed service interruptions only using the traditional client/server management model, as expected. However, this does not mean that applying the other two models none of the service nodes suffers from service interruption for shorter or longer time. But the number of these nodes is substantially lower and they never covered the majority of the service nodes in our simulations. In the client/server model, the server constitutes a single point of failure and if something happens with the server (e.g., moves away or their links break) most of the client nodes realize this and may suffer from service interruption. Moreover, this model does not offer any solution for the problem of taking over the server role if the server disappears, thus in this case the service will be permanently disrupted. An interesting observation is that the number of interruptions is decreasing when the node density is increasing in the network. Its explanation is that when some links break or the server node moves into a peripheral region of the network the possibility of finding alternative routes to the client nodes is increasing with the growing node density. And since in our simulations service interruptions happened exclusively due to communication failures (we assumed low and easily treatable computation load on the server even in the most dense scenario) increasing node density can result in less service interruptions.

- **Management Traffic Overhead:** It indicates the bandwidth required to administer the service in the given service management architecture from the viewpoint of the different service nodes. Thus, we measured the management traffic overhead at the server and at the clients in the client/server, at the zone servers and at their clients in the zone-based, and at the peer nodes in the peer-to-peer model, respectively. Recall that in the client/server and the peer-to-peer model, only some 'alive' messages were generated as management traffic. This is presumably different and involves more management traffic in real implementations, especially using the peer-to-peer model when the nodes are performing service discovery and maintaining the peer-to-peer management structure.

  Figure 6.16 shows the averaged results of the management traffic overhead measurements together with their 95 % confidence interval. We can see, that in case of the client/server model the management traffic requires a small, constant bandwidth around 400 bit/s at the client nodes (for sending the 'alive' messages to and receiving them from the server) whatever the node density is in the network, whereas the occupied bandwidth is increasing linearly with the number of client nodes at the server, as expected. In the peer-to-peer model, every peer node behaves also as a server. Hence, the required bandwidth for management traffic in this model at each of the peers is approximately the same as at the server node in the client/server model, and it is increasing linearly with the number of peer nodes. In the zone-based model, the required bandwidth for management traffic at the client nodes is a bit higher than in case of the client/server model and it is increasing roughly linearly with the increasing node density due to the growing number of neighbors and the bigger PBS messages. This also holds at the server nodes, but there the extra 'alive' messages are counted too as management traffic to keep track of the other zone servers. As we can see from the figure, the management traffic is more evenly distributed among the service nodes in the zone-based model than using the client/server model, and its amount per node is much lower than the management traffic per peer node using the peer-to-peer model. This is a nice property of the zone-based model from load balancing point of view, even if the management traffic is only a small portion of the overall network traffic.

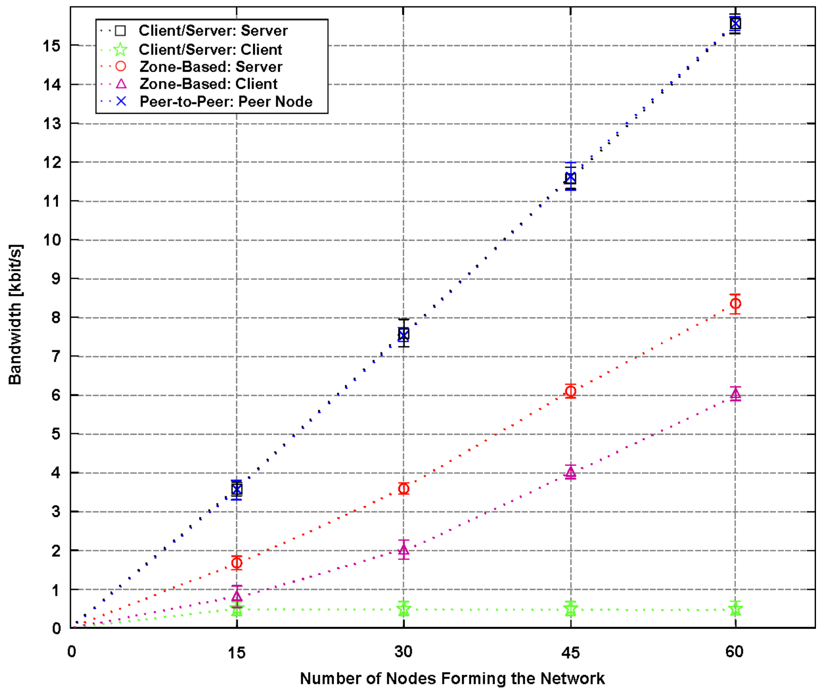

Figure 6.16: *Management Traffic Overhead Comparison*

- **Game Traffic:** This metric indicates the bandwidth occupied by the application traffic from the viewpoint of the different service nodes. We measured the game traffic at the server and at the clients in the client/server, at the zone servers and at their clients in the zone-based, and at the peer nodes in the peer-to-peer model, respectively. Recall that as game traffic a full duplex 10 kbit/s CBR data flow was generated in case of the client/server model between the server and its clients, in case of the zone-based model between the servers and their clients, and in case of the peer-to-peer model between the peer nodes. Moreover, to synchronize the servers of the zone-based model, every server sent 1/3 of the game traffic received from its clients to all the other servers.

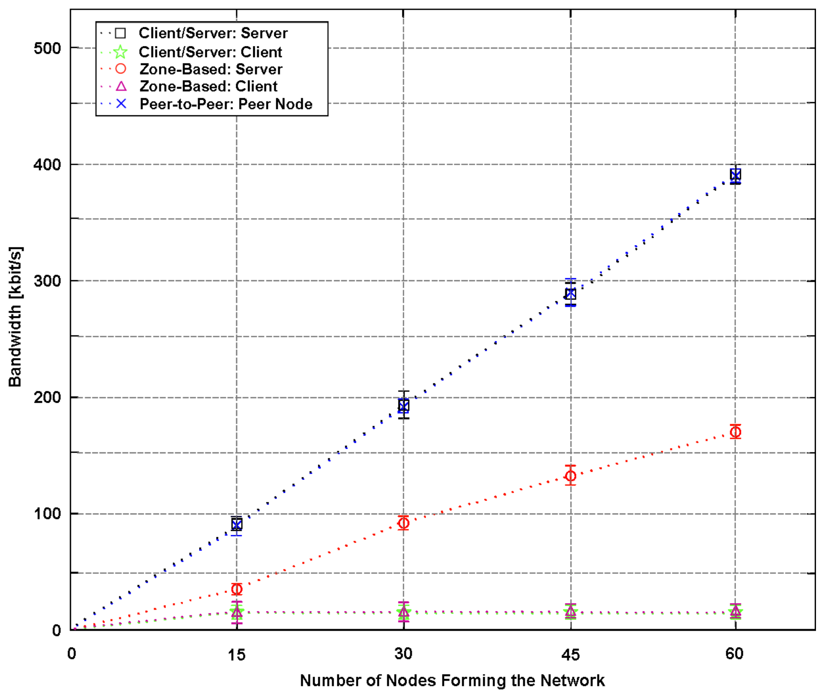

Figure 6.17: *Game Traffic Comparison*

Figure 6.17 shows the averaged results of the measured game traffic together with their 95 % confidence interval. In case of the client/server model, the game traffic occupies a constant bandwidth of 10 kbit/s at the client nodes in all scenarios, whereas the used bandwidth is increasing linearly with the number of client nodes at the server. The occupied bandwidth in the peer-to-peer model at each of the peers is approximately the same as at the server node in the client/server model, and it is increasing linearly with the number of peer nodes. In the zone-based model, the used bandwidth of the game traffic at the client nodes is the same as at the clients in the client/server model, whereas at the server nodes it is increasing nearly linearly with the increasing node density but at a slower pace than at the server using the centralized model. As we can see from the figure, the server node in the client/server and any of the peer nodes in the peer-to-peer model can become a bottleneck if the number of the nodes participating in the service is too high, which reflects the scalability concerns with these models. In this respect, the zone-based model is a better choice being capable to handle more service nodes than the other models.

- **Network Load:** The network load gives an indication about the network saturation caused by the management and game traffic. It sums up the amount of traffic transmitted in the network in total.

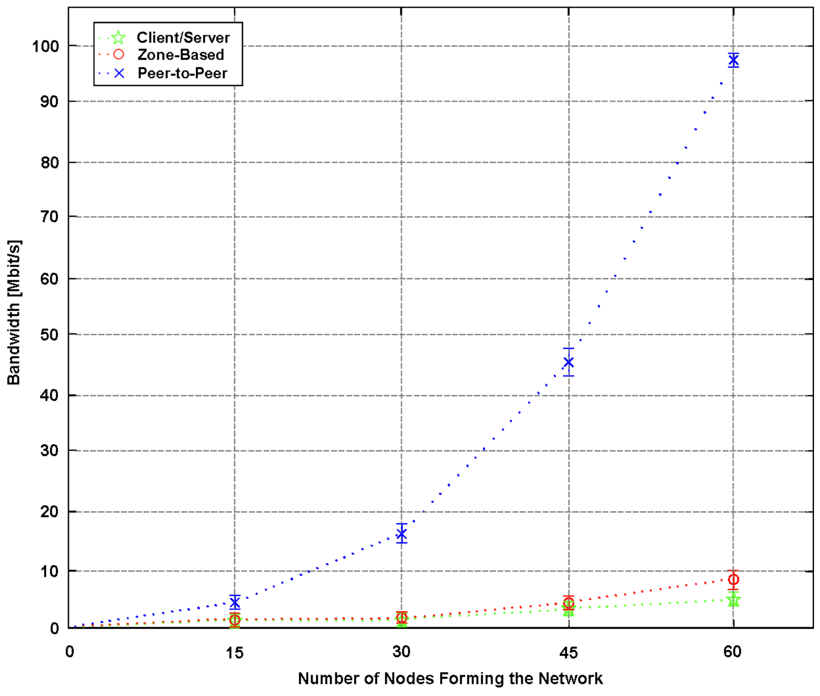

Figure 6.18: *Network Load Comparison*

> The averaged results of the network load[18] together with their 95 % confidence interval are presented in Figure 6.18. As we can see, the network load is moderate and increasing slowly with the growing node density using the client/server and the zone-based model. However, it is increasing drastically with an exponential pace using the peer-to-peer model, which is the other main scalability concern of this model. Of course, this situation can be mitigated by putting in place appropriate techniques for data transfer (e.g., using multicast communication to send game state updates to the other peer nodes), but usually this is not an easy task in a mobile ad hoc environment.

We can conclude from our simulation results, that the zone-based service management model is a reasonable alternative of the client/server model offering a redundant, fault tolerant solution to avoid service interruptions. Moreover, it shows much better scalability properties than the other models being capable to handle a higher number of service nodes. Thus, the zone-based model is a valid trade-off between the centralized client/server and the fully distributed peer-to-peer model for mobile ad hoc networks.

6.6 Chapter Summary

In this chapter, first we presented our pseudo real-time multiplayer game specification used in our simulations as the test application. Then, based on 2260 simulation rounds in total, we showed our evaluation of the PBS algorithm, the technique of service profile creation for node weight computation and our evaluation of the XCoPred prediction mechanism together with its application in PBS. Moreover, we gave a comparison of the server-client, the peer-to-peer and the zone-based service management architectures.

Our PBS algorithm computes quickly an appropriate DS of the network graph (its nodes can be used as zone servers) and offers its continuous maintenance generating moderate management traffic overhead, as we saw from our simulation results. With our NWC mechanism, a static service profile can be created and the node weights can be computed easily giving the basis of priority comparison in PBS. However, note that using such a static approach cannot provide a general ideal solution for all the different service contexts. To increase the stability of the computed DS, mobility prediction can be used.

[18]Notice the difference in magnitude of the bandwidth scale values in the figures: some bit/s in Figure 6.16, some hundred kbit/s in Figure 6.17 and some tens of Mbit/s in Figure 6.18.

XCoPred, our prediction mechanism, can provide fairly accurate link quality predictions around 2 dB of absolute average prediction error in case of appropriate parameter settings and scenarios showing clear node mobility patterns. Integrating XCoPred into PBS the stability of the selected server set can be improved, in some cases even by roughly 25 %. And finally, we can conclude that the zone-based service management model is a reasonable compromise between the centralized server-client and the fully distributed peer-to-peer model for mobile ad hoc networks offering fault tolerance and good scalability properties.

Chapter 7

Related Work

Before developing new algorithms/mechanisms it is inevitable to be aware of the previously proposed approaches in the related research areas. Thus, in this chapter we give a brief overview about the state-of-the-art approaches related to our work. First, we present previous work related to the selection of management nodes in MANETs and compare these approaches to our PBS algorithm and NWC mechanism. Then, we discuss the most interesting proposals related to mobility prediction in mobile networks comparing them to our XCoPred prediction mechanism.

7.1 Selection of Management Nodes in Mobile Ad Hoc Networks

In this section, we give a brief overview about approaches related to our PBS algorithm and NWC mechanism.

7.1.1 Clustering Mechanisms

Some of the hierarchical routing schemes proposed for mobile networks also include online clustering algorithms that can be performed by the network nodes. We discuss here two examples of such clustering proposals.

Lauer in [64] outlines a clustering mechanism for mobile networks in which nodes cooperate to elect clusterheads and super-clusterheads. Each node then affiliates itself with the closest clusterhead, and all nodes affiliated with a given clusterhead form a cluster. Each clusterhead affiliates itself with a super-clusterhead, thus affiliating all cluster members with the same super-cluster-head. All nodes affiliated with a given super-clusterhead form a supercluster. However, Lauer does not give specific election algorithm to elect the clusterheads and super-clusterheads. Moreover, the adjustment of the initial clustering hierarchy in case of network topology changes is not clear.

Ramanathan and Steenstrup in [65] give a clustering algorithm for mobile ad hoc networks. The algorithm uses link-state information distributed among network nodes and recursive bisection to produce connected level-1 clusters. A single node, the cluster leader, performs the clustering. The cluster leader is the node with the lowest-numbered identifier in a group of mutually reachable nodes, but each node is capable of assuming the role of cluster leader if necessary. To begin, the cluster leader selects two seed nodes and performs a bisection of the initial large cluster. To form each cluster, the cluster leader alternates between clusters, attempting to add to a cluster another node that is not yet a member of any cluster and is adjacent to a node in that cluster, until each node is a member of one of the two clusters. This procedure is repeated until all clusters are within the prescribed size limit. The problem with this approach is that the selection of the cluster leaders does not take into account the capabilities of the nodes (it is based only on the node identifiers) and the cluster maintenance is not clear in case of node mobility.

7.1.2 Dominating Set Computation Algorithms

Several static algorithms have been already proposed that determine a Dominating Set in a given graph, but they don’t specify how the DS can be maintained in case of topology changes. Other dynamic approaches try to handle also the DS maintenance issue, but it is not always clear how.

In Table 7.1 and 7.2, some of the static DS computation algorithms with their quality and construction properties are presented together with our PBS algorithm. Most of these algorithms have been developed for the purpose of providing routing functionality in ad hoc networks. In these tables, n indicates the number of nodes, Δ the maximum degree in the network graph, and k is an arbitrary parameter.

Algorithm	**Time Complexity [round]**	**MDS Approx.**
Largest-ID [66]	2	$O(\sqrt{n})$
LRG [34]	$O(\log\Delta\log n)$	$O(\log\Delta)$
Marking [67]	2	N/A
LP Relaxation [68]	$O(k^2)$	$O(k\Delta^{\frac{2}{k}}log\Delta)$
Dominator [69]	$O(n)$	$O(4)$
Removing Cycles [70]	$O(n)$	$O(log\Delta)$
PBS	$O(n)$	$O(log\Delta)$

Table 7.1: *Summary of the Existing DS Computation Algorithms I*

Algorithm	**CDS**	**Message Complexity**
Largest-ID [66]	No	$O(n)$ messages
LRG [34]	No	N/A
Marking [67]	Yes	$O(\Delta n)$ messages
LP Relaxation [68]	No	$O(k^2\Delta)$ messages, size $O(log\Delta)$
Dominator [69]	Yes	$O(n)$ messages, size $O(logn)$
Removing Cycles [70]	Yes	$O(n(n+2logn))$ messages
PBS	No (optional)	$O(n)$ messages, size $O(k(\Delta+1))$

Table 7.2: *Summary of the Existing DS Computation Algorithms II*

We can see, that the different algorithms have different quality and construction properties. From the view of time complexity, there are two algorithms (Largest-ID [66] and Marking [67]) that perform in the minimal number of 2 rounds. But this is paid with a higher approximation factor (for the Marking algorithm we didn't find exact analytical expression, only simulation results were given). The best approximation result, $log\Delta$, can be achieved with a greedy (LRG - Local Randomized Greedy [34]), the Removing Cycles [70] and our PBS algorithm. The Dominator algorithm [69] is a distributed approach that constructs in the first step an independent Dominating Set. In the second step, every node in the Dominating Set from the previous phase detects the best paths to the other dominator nodes (that consist at most of 2 intermediate nodes), and forces the intermediate nodes to join the DS, too.

At the end a Connected Dominating Set is constructed. A similar approach could be used by our zone server selection algorithm, if the set of zone servers needs to be connected (in the default version of PBS, we do not require this). Comparing PBS to these algorithms, it is the modification of the distributed implementation of the greedy LRG algorithm. We extended the criteria to choose a node into the DS similarly to the Marking algorithm. Moreover, the nodes exchange neighborlists for getting the relevant information about the ad hoc network. As we can see from the tables, our algorithm shows comparable analytical performance to the state-of-the-art DS computation algorithms. Moreover, it offers the maintenance of the computed DS in case of topology changes, too.

As a dynamic DS computation approach, Kozat and Tassiulas propose a distributed service discovery architecture in [29] which relies on a virtual backbone for locating and registering available services within a dynamic network topology. It consists of two independent mechanisms: the formation of a virtual backbone, which includes the nodes acting as service brokers, together with assigning into clusters the non backbone nodes; and the distribution of service registrations, requests and replies. The virtual backbone creation mechanism selects a subset of the network nodes to form a relatively stable Dominating Set, discovers the paths between the dominating nodes and adapts to the topology changes by adding or removing network nodes into this DS. Similarly to PBS, the DS nodes to form the backbone are selected in a distributed way, but it is based only on the node degree, the link failure frequency of the node and the node's ID, and does not take into account other node properties. Moreover, the maintenance of the DS is not detailed enough in [29] to see how it works exactly.

Another dynamic DS computation algorithm has been proposed by Acharya and Roy in [71], which computes a power aware minimum connected Dominating Set for routing in mobile ad hoc networks. However, this approach considers only the energy level of the node and the node degree to compute the DS. Moreover, it is not clear how the proposed algorithm maintains the DS in case of network topology changes.

7.1.3 Node Weight Computation Mechanisms

The simplest approaches in regard to selecting the DS nodes optimize the selection for a single objective. For example, minimizing the power consumption to increase the lifetime of the network/DS, or minimizing the number of DS nodes to decrease the message overhead and simplify the DS synchronization mechanism. These approaches are referred to as *single metric* mechanisms, because the heuristic for weight assignment is based on a solely metric. There are other approaches that use some combination of several metrics, thus combining multiple objectives in the selection algorithm. We refer to these approaches as *multiple metric* mechanisms. In the following, we discuss two single metric and two multiple metric mechanisms.

Single Metric Mechanisms

The *largest-ID/lowest-ID* mechanism [66, 72], also known as *identifier-based* clustering, is one of the simplest and most common approaches in the construction of a Dominating Set. In this approach, each node of the network is assigned with a unique ID and the nodes with the largest/lowest IDs are selected for the DS. This algorithm does not show good adaptation properties, since every node is assigned with a given ID in advance which remains during the whole network's life-cycle. On the other hand, changing or re-numbering the node IDs according to the network situation is not practical, because it is a complex procedure and can create a lot of overhead traffic.

The *highest-degree* mechanism [73], also known as *connectivity-based* mechanism, is another simple approach. Each node broadcasts its ID to its neighbors. The node with the maximum number of neighbors, or maximum degree, is chosen as clusterhead, and any tie is broken by the nodes' IDs, which are unique in the network. The neighbors of a clusterhead become members of that cluster, and can no longer participate in the election process. The constructed DS has a low rate of change, but the throughput is also low. If the number of cluster members increases, the throughput decreases and a gradual degradation in the system's performance can be observed.

Multiple Metric Mechanisms

Shaikh et al. propose a mechanism in [74] to select the DS based on the node degree and the remaining battery power of the nodes. In this case, the nodes frequently alternate between their dominating and dominatee status, thus balancing energy consumption and prolonging network lifetime. Comparing this technique to the single metric mechanisms we can observe, that the energy consumption is better balanced when the available battery power as the only metric is used in the DS selection. But in this case, more nodes are selected into the DS and therefore the overall network energy consumption is higher.

On the other hand, DS computation based on the node degree as the only metric tends to result in a smaller size DS, thus reducing energy consumption in each round. However, shifting roles between nodes is slow in this case, which makes nodes with higher degree energy critical. Combining the two metrics in the node weight computation can lead to an ‘equilibrium’, which consequently increases the network life. However, this approach doesn’t consider the requirements of the service for which the DS is created.

Chatterjee et al. in [75] propose a weighted clustering mechanism called WCA (Weighted Clustering Algorithm), which is a distributed clustering approach for multi-hop packet radio and mobile ad hoc networks. The main idea of this approach, similarly to our NWC mechanism, is to perform the weight computation according to the requirements of the application/system. Four parameters are considered in the weight computation, namely the node degree, the battery power, the mobility of the node and the distance between the node and its communication counterparts. The node weight is the weighted linear combination of the parameter values, such as in NWC, where the weight co-efficients are dependent on the service type being deployed in the network. With this technique, always an appropriate DS can be selected which suits best to the requirements of the service.

Our NWC mechanism is similar to WCA and based on the same idea and node weight computation method. However, in WCA it is not specified, how the parameter weight co-efficients are computed. For this in NWC, we have developed a technique based on factorial design which can be easily applied by service developers.

7.2 Mobility Prediction in Mobile Networks

In the literature, several approaches of mobility prediction in wireless mobile networks have been proposed. Here we discuss different methods appearing mainly in the field of cellular networks or proposed to improve routing in MANETs. As the research area of mobility prediction is fairly broad, we cannot give a complete overview, rather point out the most interesting techniques which are related to our XCoPred mechanism.

7.2.1 Using a Linear Model

Creating a linear (in time) model of node mobility means basically, that we assume the nodes to keep on moving in the same direction with the same speed as they currently do. In mobile ad hoc networks, determining the current speed and moving direction of the nodes usually requires special hardware, like a GPS (Global Positioning System) receiver [76]. Such a method has been investigated in different mobility prediction algorithms, for example in [77]. 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. Using the freespace radio propagation model, where the received signal strength solely depends on the distance between the sender and the receiver, the amount of time two mobile hosts will stay connected can be computed with a simple formula:

$$D_t = \frac{-(ab+cd)+\sqrt{(a^2+c^2)r^2-(ad-bc)^2}}{a^2+c^2}, \tag{7.1}$$

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

However, assuming that every node is equipped with a positioning device is a fairly limiting factor.

7.2.2 Using an Autoregressive Model

Zaidi and Mark describe a mobility tracking[19] scheme based on an autoregressive model in [39] and [78]. The position, velocity and acceleration of the mobile station in a cellular network are estimated by applying an extended Kalman filter. The Kalman filter creates an autoregressive model of the node's mobility state which is applied to the RSS (Received Signal Strength) or TOA (Time Of Arrival) measurements of the mobile node's signal in order to determine the user's actual position. It computes, in case of a model order p, the actual value as the weighted sum of the p previously measured values, or formally:

$$x_n = \alpha_0 + \sum_{i=1}^{p} \alpha_i x_{n-p} + \varepsilon_n, \tag{7.2}$$

where ε_n is an independent identically distributed noise term with zero mean. While in a cellular network, where one end of the link is fixed at the base

[19]Mobility tracking is the task to determine the movement trajectory of the nodes in time.

station and the other one is mobile, such an autoregressive model can lead to good results for mobility tracking.

However, our experiments to predict link quality in MANETs with autoregressive models have shown, that it is fairly difficult to find the right parameter settings which can lead to accurate results.

7.2.3 Using Neural Networks

A neural network (see e.g. [79]) is a network of simple processing elements (neurons) which can exhibit complex global behavior. Neural networks historically were to imitate the central nervous system of the human body 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.

Capka and Boutaba propose a mobility prediction algorithm for cellular networks based on a back-propagation neural network in [80]. 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 network 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. In this approach, the movement 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.

Because of the encoding of the movement trajectory this approach cannot simply be adopted to our mobility prediction case, as we tried to avoid the use of such reference points.

7.2.4 Using Pattern Matching

Another approach for mobility prediction based on pattern matching has been proposed by Bhattacharya and Das in [81]. The algorithm has been designed for the use in cellular networks and adapted for smart environments[20]. It uses an information theoretic approach for mobility tracking and prediction. This proposal is similar to the previous one in terms of using the history of the base stations (or closest sensors) for encoding the trajectory of the node movement. However, instead of applying a neural network for pattern recognition, it uses the LZ78 compression algorithm[21] to generate a smart dictionary which stores the observed paths from the past measurements.

This approach resembles our XCoPred prediction method the most since it is based on pattern matching, however it is also using location information for the prediction what we tried to avoid.

7.2.5 Other Methods

Yang and Wang propose enhancements using mobility prediction to several routing protocols in [82]. They show two methods of how to predict the expiration time of a wireless link. The first approach takes the location and mobility information provided by GPS, the other one uses the received signal strength of transmitted packets. In both cases, the users' behaviors are captured by linear models. Bahl and Padmanabhan in [83] describe an RF-based user location and tracking system for in building environments called RADAR. It uses the measurement of signal strength information gathered at multiple receiver locations to triangulate the user's coordinates. Triangulation is done via both empirically-determined and theoretically computed signal strength information. Liu et al. in [84] propose a location prediction algorithm, called Hierarchical Location Prediction (HLP), which is used to advance resource reservation in wireless ATM networks. HLP uses a first order autoregressive model to estimate the position, velocity and acceleration of the mobile nodes. In order to determine the current mobility state, Kalman filters are applied to the measurements of the user's signal.

In all of these methods, either a positioning device or the use of some external reference points is required. Thus, they cannot be easily applied in mobile ad hoc networks.

[20]A smart environment, like smart homes, is one that is able to acquire and apply knowledge about humans and their surroundings, and also adapt to improve their experience.

[21]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.

7.3 Chapter Summary

In this chapter, we briefly surveyed the state-of-the-art research approaches related to our work. Thus, first we described previous work related to the selection of management nodes in MANETs and compared these approaches to our PBS algorithm and NWC mechanism. Then, we discussed the most interesting proposals related to mobility prediction in mobile networks and pointed out their differences compared to our XCoPred prediction mechanism.

We can conclude that numerous approaches have been proposed to create clusters for management purposes in mobile networks either using Dominating Set computation or other algorithms. Most of these proposals take into account only a single metric (e.g., the node ID or the node degree) in selecting the clusterhead nodes which does not capture accurately the node properties. Moreover, these approaches either do not deal with cluster maintenance in case of network topology changes or they do not specify the required procedures in detail. Our PBS algorithm offers the continuous maintenance of the selected DS and using our NWC mechanism the computed node weight captures more accurately the node properties making possible to select the most appropriate nodes into the DS. Concerning mobility prediction in mobile networks, a huge number of approaches have been developed, as well. However, almost all of them require either the use of a positioning device (e.g., a GPS receiver) or some external reference points. Thus, they cannot be easily applied in mobile ad hoc networks. Our XCoPred prediction mechanism does not rely on any external device nor reference point, only the signal quality between the communicating nodes has to be monitored to predict the future link quality.

Chapter 8

Conclusions

As the closing of the dissertation, this final chapter gives a summary and concludes the thesis. Thus, first we recall the context of our work and briefly review our contributions. Then, we give a critical assessment of the achievements followed by a short discussion about future work. And in the end, we conclude the thesis with our final remarks.

8.1 Thesis Context

Wireless mobile ad hoc networks (MANETs) form a promising paradigm for future mobile communication. The devices in these networks can communicate directly in an ad hoc manner without requiring any external infrastructure. These networks can appear anytime and anywhere providing new ways to run distributed services, especially real-time multiuser applications.

Making a MANET functional the participating nodes must organize themselves spontaneously. Moreover, they must provide data relaying and service provisioning functionalities for distant nodes beyond acting as terminals. In this thesis, we focused on service management issues. Using appropriate service management architecture in the mobile ad hoc environment is crucial to help the spreading of MANETs in everyday life. This architecture must be able to cope with the inherent properties of MANETs, such as the lack of a central infrastructure, the high level of device heterogeneity, the degree of mobility, the self-organizing and error-prone properties and often the resource constraints of the participating devices. The architectures of traditional service provisioning/management solutions used in communication and data networks are not well suited for MANETs. Usually they are either based on the centralized service management model using the client/server architecture, which provides low fault tolerance, or the fully distributed model using the peer-to-peer architecture, which has scalability concerns. As a compromise, the hybrid service management model using the zone-based architecture offers high level of fault tolerance and provides much better scalability properties than the peer-to-peer architecture, thus it is well suited for MANETs. However, the main challenge is the creation of the zones and the selection of the zone servers in mobile ad hoc networks. This has to be carried out in an efficient and distributed way.

In this thesis, we presented the design, development, implementation and evaluation of our novel, distributed algorithm (PBS) together with two other, newly developed mechanisms (NWC, XCoPred) to create the zones and select the zone servers to be able to implement the zone-based service management architecture.

8.2 Review of Contributions

The primary goal of this thesis was to help the implementation of the zone-based service management architecture in MANETs and thus design, develop, implement and evaluate new mechanisms, which can aid the efficient creation of zones and the selection of zone servers. Keeping this goal in view, below we review the main contributions of the thesis:

1. *We design and develop a distributed Dominating Set computation algorithm called PBS. This algorithm computes and maintains an appropriate DS of the ad hoc network graph based on node priority in a fully distributed manner containing nodes which can be used as zone servers.*

   In the zone-based service management model, the nodes participating in the service are divided into separate zones. In every zone, a dedicated server node handles the clients belonging to the zone and synchronizes with the other zone servers. As zone server nodes, the most appropriate and most powerful nodes should be selected in an efficient and distributed way.

   We have developed a Dominating Set computation algorithm called PBS (Priority Based Selection) [13–15] which, based on graph theory, computes an appropriate DS of the ad hoc network graph in a fully distributed manner containing nodes which can be used as zone servers (cf. Section 3.2). PBS performs in rounds and is based on exchanging neighborlists that contain the relevant information about the neighboring nodes. In every round, the node sends the current neighborlist to its neighbors, receives the neighborlists from them and determines its own status. A node can be in DOMINATOR (acts as zone server), DOMINATEE (regular client), INT_CANDIDATE (participates in the service as a client, but not yet determined whether DOMINATOR or DOMINATEE) or EXT_CANDIDATE (does not participate in the service as a client, but can be chosen as DOMINATOR) status. The algorithm compares the node priorities, and chooses the highly prioritized nodes as DOMINATORs. Priority comparison is based on the node weight, the span value (number of INT_CANDIDATE neighbors), the number of DOMINATOR neighbors, and finally the node's ID.

To ensure the smooth running of the distributed application, the set of zone server nodes must be maintained and recomputed on the fly when it is required (e.g., in case of network topology changes or link failures). PBS is the first algorithm, according to our knowledge, that offers continuous maintenance of the DS when the network graph changes dynamically. The nodes start to exchange neighborlists when they notice some change and the algorithm performs some new rounds in the vicinity of the change if it is required. PBS shows a stable performance even in case of high node mobility keeping the DS computation time nearly constant (cf. Section 6.2).

2. *We design and develop a mechanism called NWC to calculate node weight to be used in the node priority comparison of PBS. NWC calculates the node weight based on a set of node parameters and the characteristics of the given service. It assigns the highest priorities to the best suited nodes for a given service type and thus designates the most powerful nodes to be selected as zone servers.*

   To select the most powerful nodes as zone servers, PBS compares the priority of the nodes which reflects from the viewpoint of a given service the node's available computation and communication resources and its position in the network. We have developed a mechanism called NWC (Node Weight Computation) [16] to be applied in node priority comparison based on a set of node parameters and the characteristics of the service being used (cf. Section 3.3).

NWC computes the node weight, on which the priority comparison is based, as the weighted linear combination of the node parameters. In contrast with the existing algorithms, which usually consider only one or a small group of node parameters aiming to optimize the DS selection for a unique objective, NWC uses five parameters, such as the available CPU, memory, battery power, the quality of the node's links and the node's position in the network, to capture the node's capabilities as close as possible.

The parameter weights are extracted from the so-called service profile which reflects the characteristics and requirements of the given service. To create this profile containing the appropriate parameter weights we have applied factorial design and multi-objective optimization based on simulation (cf. Section 6.3). With this technique it is possible to assign the highest priorities to the best suited nodes for a given service type and thus designate the most powerful nodes to be selected as zone servers.

3. *We design and develop a mechanism called XCoPred to help increase the stability of the selected zone server set via prediction. XCoPred predicts the variations of the wireless link quality based on pattern matching which can be exploited in assessing future changes of the network topology. Then, using this information in server selection the stability of the selected zone server set can be increased.*

   To increase the stability of the selected server set taking node mobility into account is indispensable. High mobility of the nodes can result in the selection of unstable zone servers leading to frequent changes of the server nodes and thus frequent handovers of clients between them. This also causes high traffic overhead or even service disruption. Mobility prediction can mitigate this problem. It can help increase the stability of the server set by predicting future changes of the network topology and using this information in server selection.

   We have developed a mechanism called XCoPred [17,18] to predict the variations of the wireless link quality based on pattern matching which can be exploited for mobility prediction (cf. Chapter 4). To the best of our knowledge, such an approach to mobility prediction is novel in the area of MANETs, as most of the existing techniques use linear models and are based on having localization information from dedicated hardware, such as a GPS receiver. In order to observe the mobility state of a node without using dedicated hardware, the Signal to Noise Ratio (SNR) of the links is monitored and filtered with a Kalman filter to get rid of noise in XCoPred. When a prediction is required, the node tries to detect patterns similar to the current situation in the history of its links' SNR values and tries to obtain a set of predictors. For this purpose, the node computes the normalized cross-correlation between the current pattern and the history of the links' quality. From the detected set of predictors the most probable predictor is used as the prediction of future link quality. For cases where no predictors can be found, we apply a fallback solution based on an autoregressive model. Using XCoPred highly accurate predictions of link quality can be achieved with around 2 dB of absolute average prediction error in case of appropriate parameter settings and scenarios showing clear node mobility patterns (cf. Section 6.4.5).

Furthermore, to increase the stability of the selected zone server set, we introduced and implemented a link stability criterion into the PBS algorithm. Using this criterion, a client node (DOMINATEE) accepts only a neighbor as server (DOMINATOR) when it has a link to it which is predicted to be stable for a certain time in the future. Integrating XCoPred into the PBS algorithm we could improve the stability of the selected server set and decrease the number of zone server changes substantially, in some cases even by approximately 25 % (cf. Section 6.4.6).

Moreover, all these algorithms/mechanisms, i.e., PBS, NWC and XCoPred, were implemented and evaluated in the network simulator NS-2 (cf. Section 5.1). As a proof of concept, we also implemented them in our SIRAMON framework running in our mobile ad hoc testbed together with a demo multiplayer game application (cf. Section 5.2).

8.3 Self-Assessment

In this thesis, we intended to propose algorithms/mechanisms which were able to aid the creation of appropriate service management architecture in MANETs coping with the constraints of the mobile ad hoc environment. Our contributions (the PBS algorithm, the NWC mechanism and the XCoPred mechanism) support the creation of a zone-based service management architecture which is more appropriate for self-organized networks than the traditional management architectures, such as the client/server or peer-to-peer architecture.

As the novelty of our distributed Dominating Set computation algorithm, PBS is the first approach, according to our knowledge, that offers continuous maintenance of the DS in case of dynamically changing network topologies. It is important to see the difference between the DS maintenance and simply select a new DS whenever some change happens. Clearly, it is not practical to rerun the DS computation algorithm in case of changes in the whole network because it can create a huge management overhead traffic, requires considerable amount of time to select again and again the DS nodes and can compromise the stability of the initially selected DS. Rather, PBS tries to 'fix' the DS in the vicinity of the change which usually does not propagate through the whole network making the recomputation fast and preserving the not concerned part of the DS. Another handy feature of PBS is that the properties of the selected DS can be easily adjusted by merely reordering the criteria (node weight, span value, number of DOMINATOR neighbors, node ID) in the priority comparison. For example, if our goal is to get a minimum DS the span value has to be considered at the first place in the comparison. Our algorithm shows comparable analytical performance to the state-of-the-art DS computation algorithms and provides stable behavior generating only a small amount of management overhead traffic, as we saw it from our simulation results. The weak point of PBS is the oscillation problem (oscillation between DOMINATOR and DOMINATEE status of the node, cf. Section 6.2.2), which can arise if a DOMINATOR node switches back immediately when it detects other DOMINATOR nodes also covering its DOMINATEE neighbors. However, this problem can be avoided by taking into account also node mobility in the DS selection and delaying a bit the switch back decision.

To select the most powerful nodes into the DS our NWC mechanism computes the node weight using a representative group of node parameters and a so-called service profile. The novelty of NWC is taking into account several parameters in the weight computation and the technique to create the service profile in an offline/static way applying factorial design and multi-objective optimization based on simulation. The weak point of NWC is the static approach to create this profile, because with such an approach it is not possible to accurately predict the future service context (e.g., network scenario, mobility behavior of the nodes) in advance where the given service will be deployed. Thus, even knowing the basic characteristics of the service an optimal profile is not granted for every situation. This problem can be solved with a more complex, dynamic approach which adapts continuously the service profile to the actual service context. However, the development of this dynamic service profile creation mechanism requires more research and remains as a topic for future work.

To increase the stability of the selected zone server set mobility prediction can play an important role. The novelty of our XCoPred mechanism is that it predicts the variations of the wireless link quality based on pattern matching without relying on any positioning hardware or external reference point. According to our simulation results, highly accurate predictions can be achieved with XCoPred in most of the cases. However, these results still have to be justified in real environments via measurements which requires the setup of a test ad hoc network consisting of enough nodes. Another thing to note is that XCoPred focuses on predicting the future variations of existing links. While this is enough for increasing the stability of the selected DS, it might be useful for other applications to predict the whole future topology of the network. This would mean to use mobility prediction in terms of predicting the future position of a node in the network. In order to do so, our approach can be extended by an algorithm, e.g., using MDS (Multidimensional Scaling) [19], that concludes the network topology from the distances between the nodes[22]. The weak point of XCoPred is that it is costly. It can be easily seen, though we did not carry out a detailed complexity analysis in this thesis, that computing the normalized cross-correlation of the queries and the training data as well as correlating the predictors with one another present a big computation overhead for the nodes. Especially for devices with very limited resources, such as mobile phones, this might be a too big burden and a simpler or more optimized solution might be preferable. However, analyzing the complexity and optimizing XCoPred are left as another topic for future research.

As we mentioned earlier, we had implemented all the developed approaches in our SIRAMON testbed. Unfortunately, we could not accomplish thorough evaluation in this testbed due to its limited size and node mobility, but this work was still very useful and let us gain a lot of real world experience with mobile ad hoc networking.

[22]Distance is not used in geographical sense here, rather as ‘signal space’ measure.

8.4 Future Work

The research presented in this thesis has created a set of contributions, such as the PBS algorithm, the NWC and XCoPred mechanisms, in the area of service management in mobile ad hoc networks. Based on these contributions further research can be built and it is also worth to consider how our achievements could be commercialized. Below we give a set of ideas for follow-up research.

Concerning PBS, it can be useful to examine further the scalability limits of the algorithm and the influence of a constantly changing network topology on the application which can put the algorithm into a continuous convergence situation. Moreover, it would be interesting to see how the performance of PBS compares to other clustering approaches or newly proposed algorithms, thus the development of a comparison framework can be also worthwhile. In this thesis, we have investigated PBS only via simulations. However, in order to be able to assess its real world applicability, it is inevitable to investigate our algorithm also in a real, testbed environment.

With regard to our node weight computation mechanism, above we referred to its weak point which was the static approach to create the service profile. This could be replaced with a dynamic approach which adapts continuously the service profile to the actual service context. Moreover, the performance investigation of NWC in a real environment, which implements also the application, would be even more important here because the changes of the node parameter values (CPU, memory, battery power) can be imitated in a simulator only with a coarse fidelity.

In the area of mobility prediction, we believe that our XCoPred prediction approach has great potentials. Extending it with network topology prediction features its applicability could be substantially increased from supporting routing decisions via traffic engineering to even application layer usage in mobile ad hoc networks. However, as we mentioned above, the pattern matching method used in XCoPred is computationally expensive. Computing the normalized cross-correlation functions and the correlations for selecting the most common predictor is costly and a more efficient method might be found. One viable approach is to use fast normalized cross-correlation in the frequency domain [40]. And finally, the investigation of XCoPred in a real testbed environment is also desirable and could help further develop our prediction method.

8.5 Final Remarks

Mobile ad hoc networking is an emerging communication paradigm and it has promising features to become a widely used technique in the future. However, it imposes numerous challenges which reserve room for continuous research.

One challenge is to develop appropriate service provisioning procedures which can cope with this dynamic environment. We believe that our contributions constitute a step further on the research highway and hope that the community can profit from our work.

Acknowledgements

I would like to express my gratitude to a number of people without whom this thesis would have been never carried out.

First of all, I would like to thank my supervisor Prof. Dr. Bernhard Plattner for his continuous support and for the numerous invaluable discussions which guided me on the right way towards my dissertation. Many thanks for letting me pursue this not so usual research topic covering computer games related networking issues and for the possibility to work and spend several years in the CSG (Communication Systems Group) research group at ETH Zurich.

I am also very grateful to my co-advisor Prof. Dr. Lars Wolf from the Technical University of Braunschweig for the co-operation during the last couple of years and for his comments and suggestions which greatly improved this dissertation. Thanks for spending the huge amount of time and effort on reviewing my dissertation draft.

Many thanks also to Prof. Dr. Polly Huang whom I was so lucky to start working with when I arrived at ETH Zurich. She was the first person who seriously guided me in the labyrinth of research and I learnt a lot from her.

My special thanks go to Dr. Lukas Ruf for his friendship and research collaborations. I will never forget the long discussions and sleepless nights we spent together on paper writing in the accompany of some pizza and beer. Moreover, I would like to thank all my colleagues from the CSG laboratory for the great time we spent together: Rainer Baumann, Marcel Baur, Dr. Matthias Bossardt, Daniela Brauckhoff, Dr. Thomas Dübendorfer, Dr. Ulrich Fiedler, Placi Flury, Stefan Frei, Dr. Jan Gerke, Dr. Hasan Hasan, Dr. David Hausheer, Simon Heimlicher, Dr. Konstantinos Katrinis, Dr. Ralph Keller, Pascal Kurtansky, Dr. Vincent Lenders, Dr. Martin May, Dr. Jan Mischke, Georgios Parissidis, Dr. Marc Rennhard, Daniel Sigg, Mario Strasser, Svetlena Taneva, Bernhard Tellenbach, Arno Wagner and Dr. Nathalie Weiler. And of course, I have to express my gratitude to Caterina Sposato, our great secretary, who tirelessly arranged the administrative issues day after day.

I would like to acknowledge the contributions from students doing a master or semester thesis under my supervision, in particular to Rolf Grüninger, Florian Maurer, Dirk Budke, Eduardo Silva, Theuss Hossmann, Samuel Korpi, Christian Meyer, Peter Bänziger, Patrick Bönzli and Benjamin Grauer.

I am also thankful to Prof. Dr. Tamás Henk who was my first advisor at Technical University of Budapest in Hungary when I started my Ph.D. studies. Moreover, I am grateful to Prof. Dr. László Jereb for always encouraging me on the bumpy road towards my Ph.D. title and offering me a follow-up position in his group in Hungary.

My gratefulness goes also to the Hungarian company in Zurich which made my stay funny and more enjoyable during the last couple of years. Especially to my friends in other ETH research laboratories, such as to Tamás Kovács, Dr. Gábor Szabó, Dr. János Zátonyi, Zoltán Sárosi and Ádám Darvas, to the Hungarian soccer team in Zurich, and to the Hungarian Folk Dance Club in Zurich led by Ágnes Simon and Éva Ferrari-Avar for helping the lonely hours forget.

And last but not least I would like to thank my parents Ilona and József and my brother Gábor for their unconditional support, and my partner Anna for her great love and understanding over my stay in Switzerland.

Appendix A

NS-2 Details

A.1 Mobility Models

In the evaluation of the developed mechanisms, we have used basically two mobility models in NS-2, the Random WayPoint and the Freeway model. These are discussed in the following. A good overview of the most common mobility models used for simulations can be found in [85].

A.1.1 Random WayPoint Model

The Random WayPoint mobility model (RWP) [56] is a representative of such a model where the motion of the nodes shows little structure. Thus, it is hard to predict the future SNR values. With the RWP model, the 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 of the RWP mobility model is shown in Figure A.1. An exemplary SNR pattern to which such movement leads is plotted in Figure A.2.

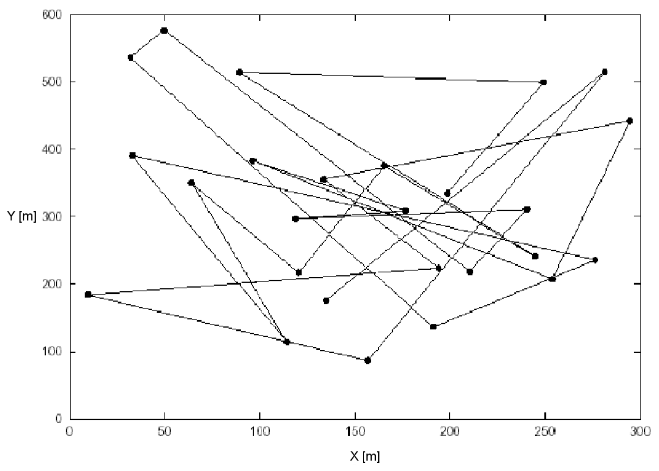

Figure A.1: *Travelling Pattern of a Node Using the RWP Model*

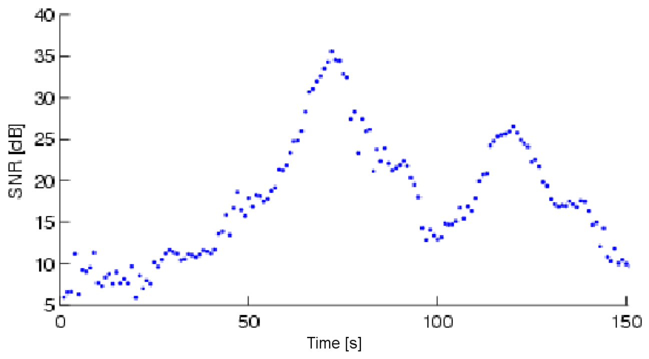

Figure A.2: *Typical SNR Pattern of a Link Driven by the RWP Model*

A.1.2 Freeway Model

The Freeway mobility model [63] is a representative of models which show clear movement patterns. 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) [86]. As a model of lane change and overtaking maneuvers, the Freeway model uses the MOBIL (Minimizing Overall Breaking Induced by Lane-Changes) [86] strategy. An exemplary SNR measurement pattern using the Freeway mobility model is shown in Figure A.3. It can be clearly seen how the nodes (cars) 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 nodes moving in the same direction the patterns look similar but they are more spread in time. The faster car approaches from behind, then overtakes the slower one and the distance increases until the connection is lost.

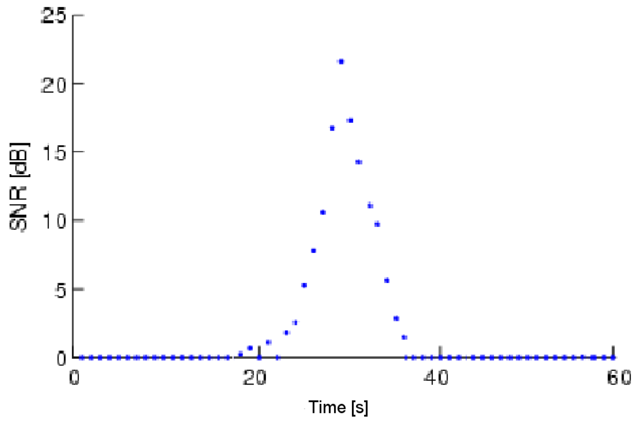

Figure A.3: *Typical SNR Pattern of a Link Driven by the Freeway Model*

A.2 Signal to Noise Ratio in NS-2

For implementing our mobility prediction mechanism a realistic model of the Signal to Noise Ratio[23] is to be used taking into account the radio propagation (path loss, and effects like reflection, scattering and diffraction) and noise (environmental noise, receiver noise and interference) properties of the signal. In the following, the SNR model used in NS-2 is explained.

[23]Note that typical values of the signal power in wireless 802.11 LAN networks range from -90 dBm to -40 dBm, while a typical noise level observed is around -90 dBm. Thus, typical SNR values range from 0 dB to 50 dB.

A.2.1 Radio Propagation

As radio propagation model the *shadowing model* [62] is the most commonly used in NS-2. 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

$$\frac{P_r(d)}{P_r(d_0)} = -10\beta log(\frac{d}{d_0})[dB], \tag{A.1}$$

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 [62]. In our simulations we chose $\beta = 4$, 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 A.4[24].

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:

$$\frac{P_r(d)}{P_r(d_0)} = -10\beta log(\frac{d}{d_0}) + X_{dB}[dB] \tag{A.2}$$

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 of σdB range from 3 to 12 (in [62] the values for different environments are discussed in more detail). In our simulations, we chose the value of $\sigma dB = 7$, which is representing an office environment with hard partitioning. The SNR pattern of the same link as above, this time with the added random variable X_{dB} is shown in Figure A.5.

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

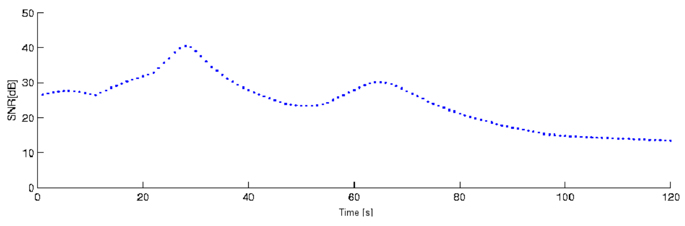

Figure A.4: *SNR Using Deterministic Distance to Signal Strength Relation*

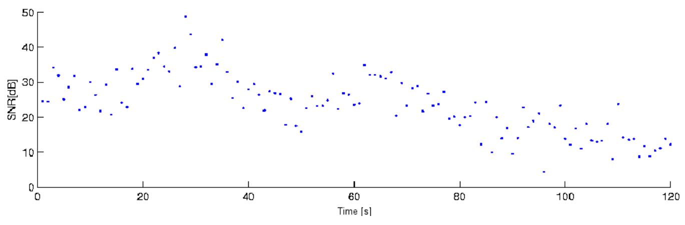

Figure A.5: *SNR Using the Shadowing Model*

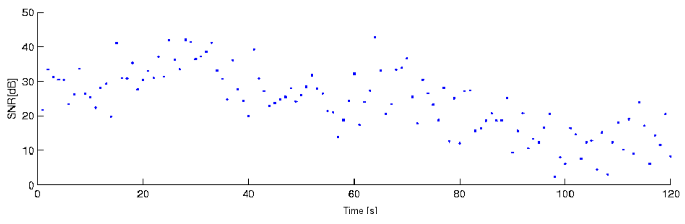

Figure A.6: *SNR Using the Shadowing Model Plus Noise and Interference*

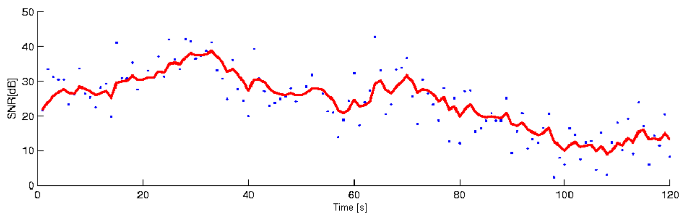

Figure A.7: *SNR Filtered with Kalman Filter*

How the shadowing model is deployed and configured in the `Tcl` scrips of NS-2 is shown in Listing A.1.

```
$ns node-config -propType Propagation/Shadowing
# path loss exponent
Propagation/Shadowing set pathlossExp_ 4.0;
# shadowing deviation (dB)
Propagation/Shadowing set std_db_ 7.0;
# reference distance (m)
Propagation/Shadowing set dist0_ 1.0;
# seed for RNG (Random Number Generator)
Propagation/Shadowing set seed_ 0;
```

Listing A.1: *Usage of the Shadowing Propagation Model in NS-2*

A.2.2 Noise and Interference

In order to account for environmental noise and receiver noise, another Gaussian distributed random variable is added to the SNR values. Its mean is 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 power of the two packets is 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:

- The interference is not additive. Thus, only two packets are considered. This is problematic if more packets are being received simultaneously.
- The duration of the interference generated by a packet is not considered.
- Only packets with signal power higher than the *receive threshold* constant `RxThresh_`[25] are taken into account.

[25]The *receive threshold* is a constant signal strength value defined in NS-2, above which the

- The interference is implemented in the MAC layer instead of the physical layer, to where it belongs.

However, in the lack of a better interference model we used this model in our 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. Figure A.6 shows the SNR pattern of the same link as above with added noise and interference to the shadowing model. Moreover, Figure A.7 depicts the SNR pattern of the given link after applying our Kalman filter.

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, we extended the packet header (file `common/packet.h`) with a property `snr_`. The code of the SNR calculation is listed in Listing A.2.

```
double snr;
RNG noise;
noise.reset_next_substream();
if(rx_state_ == MAC_IDLE) {
     snr = 10*log10(p->txinfo_.RxPr) -
              - noise.normal(-90,4);
} else {
     snr = 10*log10(p->txinfo_.RxPr)
           - 10*log10(pktRx_->txinfo_.RxPr)
           - noise.normal(-90,4);
}
hdr->snr_ = snr;
```

Listing A.2: *SNR Calculation in NS-2*

packet is received correctly. It is typically set to -95 dBm.

Appendix B

Implementation Details

In the following, we give some details related to the implementation of PBS, NWC and XCoPred in the NS-2 simulator and our SIRAMON framework.

B.1 Finite State Machine in the PBS Implementation

We have used a Finite State Machine (FSM) to implement the PBS algorithm in the NS-2 simulator and the SIRAMON framework. An FSM is a model of behavior composed of states, transitions and actions. Between different states of the machine transitions are defined which perform the appropriate actions based on the incoming events. The specification of the used FSM is shown in Figure B.1.

B.1.1 Finite State Machine States

The FSM consists of four states, such as `idle`, `msgsent`, `roundfinished` and `finished`. The `idle` state is the initial state. Once the PBS algorithm is started it performs in rounds. In every round, it sends the Neighborlist to its neighbors and waits in the `msgsent` state for the neighborlists from the neighbors. If all required neighborlists are arrived or the timeout timer has expired, it goes to the `roundfinished` state and determines the own status. If there are still INT_CANDIDATE neighbors, the neighborlist is resent

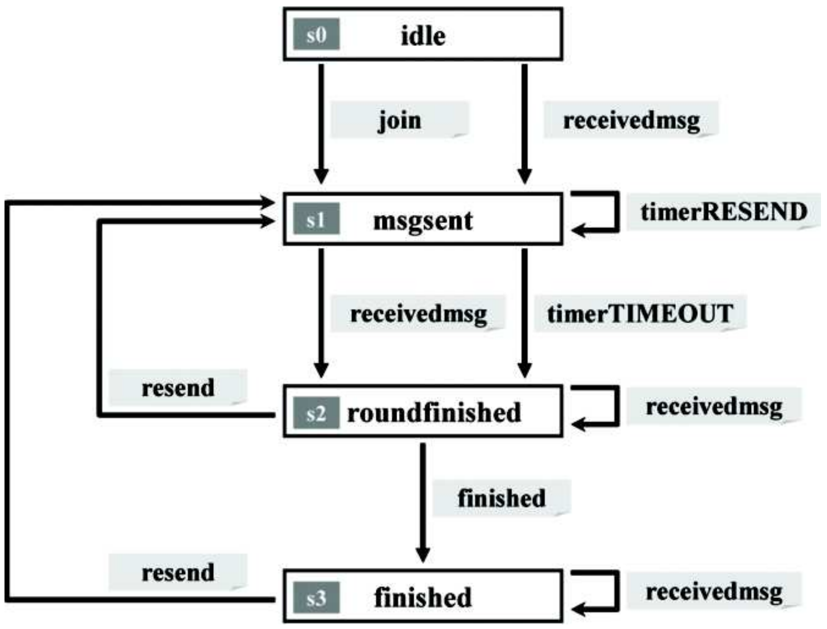

Figure B.1: *Finite State Machine of the PBS Implementation*

and the FSM is waiting again in the `msgsent` state. If all INT_CANDIDATE neighbors have switched to DOMINATEE or DOMINATOR status, the FSM goes to the `finished` state. It is waiting in the `finished` state unless some changes in the network topology are detected or there are again some new INT_CANDIDATE neighbors. If this is the case, a new round of the PBS algorithm will be started and the FSM turns again to the `msgsent` state.

B.1.2 Finite State Machine Transitions and Actions

All transitions, the corresponding events and a short description of the related action of the FSM are presented in Table B.1 and Table B.2.

From State	**To State**	**Event**	**Action**
`idle`	`msgsent`	`join`	The node joins the service session 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 the neighborlist back. Therefore, the already sent neighborlist will be resent
`msgsent`	`roundfinished`	`receivedmsg`	All required neighborlists have arrived. The node determines its own status
`msgsent`	`roundfinished`	`timerTIMEOUT`	Not all required neighborlists have arrived, but the timeout timer is expired and the node determines its own status

Table B.1: *Transitions, Events and Actions of the PBS FSM (I)*

From State	To State	Event	Action
`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 if the node is already in the `finished` state. If required it sends a neighborlist back
`finished`	`msgsent`	`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 B.2: *Transitions, Events and Actions of the PBS FSM (II)*

B.2 PBS Implementation Details in NS-2

B.2.1 Directory Structure

In Table B.3, the directory structure of the PBS implementation in NS-2 is shown. All the necessary files are located under the NS-2.28 main directory. Table B.4 shows the files used for the implementation in the C++ core of the simulator and gives a short description about them.

Directory	**Description**
ns-2.28	The NS-2 main directory
ns-2.28/zoneserver	The zoneserver main directory
ns-2.28/zoneserver/utils	Some utils for the zone server selection agents (ZSSAgent)
ns-2.28/zoneserver/doc	Source code documentation
ns-2.28/zoneserver/tcl/wired	Tcl scripts for wired environments
ns-2.28/zoneserver/tcl/wireless	Tcl scripts for wireless environments

Table B.3: *Directory Structure of the PBS Implementation in NS-2*

File	**Description**
`zss{.h .cc}`	The main class for any zone server selection (ZSS) implementation
`zss_pbs{.h .cc}`	Implements the PBS algorithm
`timer_pbs{.h .cc}`	Timer functionality used by the PBS algorithm
`neighborlist{.h .cc}`	Stores and handles the relevant information about the neighbors
`node_information.h`	Contains the structure of a node entry in the Neighborlist and the structure of the sent PBS messages
`monitor{.h .cc}`	Monitors the 1-hop neighborhood
`timer_monitor{.h .cc}`	Timer functionality used by the monitor
`debugger{.h .cc}`	Debugger used to write outputs
`utils/fsm{.h .cc}`	Implementation of a PBS FSM
`utils/state{.h .cc}`	State of the PBS FSM

Table B.4: *Files of the PBS Implementation in NS-2*

B.2.2 Running Simulations Using PBS in NS-2

To run simulations in NS-2 Tcl scripts are used to build different topologies and to simulate different scenarios. In Listing B.1, an example of the basic Tcl commands used for attaching the PBS agent to nodes is shown. Note that the shown script does not build a topology between the nodes or add movements of the nodes. It only attaches the PBS agent to the given nodes.

```
set ns [new Simulator]
#create 3 nodes
set node(0) [$ns node]
set node(1) [$ns node]
set node(3) [$ns node]
#create agents
set agent(0) [new Agent/ZSS/PBS]
set agent(1) [new Agent/ZSS/PBS]
set agent(2) [new Agent/ZSS/PBS]
#attach agents to the nodes
$ns attach-agent node(0) agent(0)
$ns attach-agent node(1) agent(1)
$ns attach-agent node(2) agent(2)
#set node weights
$agent(0) set weight_ 10
$agent(1) set weight_ 20
$agent(2) set weight_ 30
#Schedule events
$ns at 0.00001 ‘‘$agent(0) init_zss’’
$ns at 0.00001 ‘‘$agent(1) init_zss’’
$ns at 0.00001 ‘‘$agent(2) init_zss’’
$ns at 0.1 ‘‘$agent(0) join’’
$ns at 0.1 ‘‘$agent(1) join’’
$ns at 0.1 ‘‘$agent(2) join’’
$ns at 900.0 ‘‘finish’’
```

Listing B.1: *Tcl Commands to Attach the PBS Agent to the Nodes in NS-2*

B.3 XCoPred Implementation Details in NS-2

B.3.1 Data Structures

The most important fields of the `LinkMeasurements` data structure are summarized in Table B.5.

Field	**Description**
`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 4.2.1
`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 B.5: *Overview of the* `LinkMeasurements` *Data Structure*

B.3.2 Configuration Files

In order to set the parameters in the state observation part of XCoPred, three config files are used. They are listed in Table B.6 and can be found in the `zoneserver/tcl/wireless/` directory. Each of the files simply contains the value of the parameter.

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 B.6: *Configuration Files of the State Observation Part of XCoPred*

B.4 PBS Implementation Details in SIRAMON

Table B.7 gives a short description of the packages containing the PBS code in the SIRAMON framework. Moreover, Table B.8 lists the files and their short description that are contained in these packages.

Package	**Description**
`Siramon.Management.zss`	Contains the main classes needed by the zone server selection functionality
`Siramon.Management.zss.utils`	Contains some utilities for the zone server selection functionality like a basic debugger and the FSM

Table B.7: *Packages Containing the PBS Code in SIRAMON*

File	**Description**
`ZSS.java`	The main class for any zone server implementation
`ZSS_PBS.java`	Implements the PBS algorithm
`ZSS_PBSfsm.java`	Implements the FSM used by the PBS algorithm
`ZSS_PBSpacket.java`	PBS packet format to be sent in the content part of the SIRAMON packet
`TimerPBS.java`	Timer functionality used by the PBS algorithm
`Neighborlist.java`	Stores and handles the relevant information about the neighbors
`NodeInformation.java`	Used by the Neighborlist to store the information about the neighbor
`utils/Debugger.java`	Debugger used to write outputs
`utils/FSM.java`	Basic implementation of an FSM
`utils/State.java`	Implementation of the FSM states

Table B.8: *Files of the PBS Implementation in SIRAMON*

Table B.9 shows the new classes of the `Siramon.Network` package implementing the network monitor functionality.

File	**Description**
`NetMonitor.java`	The monitor that sends and receives packages to detect the 1-hop neighbors
`NetMonitor_Callback.java`	Callback interface used by other classes to be informed about changes in the neighborhood

Table B.9: *Files of the Network Monitor Implementation in SIRAMON*

Appendix C

Abbreviations

AODV	Ad Hoc On-Demand Distance-Vector
AR(1)	First Order Autoregressive Model
ATM	Asynchronous Transfer Mode
AWGN	Additive White Gaussian Noise
CBR	Constant Bit Rate
CPU	Central Processing Unit
CDS	Connected Dominating Set
DS	Dominating Set
FSM	Finite State Machine
GIF	Graphics Interchange Format
GPRS	Generalized Packet Radio Service
HLP	Hierarchical Location Prediction
IEEE	Institute of Electrical and Electronics Engineers
LP	Linear Programming
LRG	Local Randomized Greedy
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network

MDS	Minimum Dominating Set
NAM	Network Animator
NS-2	Network Simulator version 2
NWC	Node Weight Computation
NWDS	Node Weighted Dominating Set
PBS	Priority Based Selection
PDA	Personal Digital Assistant
QoS	Quality of Service
RADAR	In-Building RF-Based User Location and Tracking System
RF	Radio Frequency
RSS/RSSI	Received Signal Strength/Received Signal Strength Indicator
SDP	Service Discovery Protocol
SIRAMON	Service provIsioning fRAMework for self-Organized Networks
SNR	Signal to Noise Ration
Tcl	Tool command language
TTL	Time to Live
TOA	Time Of Arrival
OTcl	Object Tcl
UDG	Unit Disk Graph
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications Service
UPnP	Universal Plug and Play
WCA	Weighted Clustering Algorithm
WLAN	Wireless Local Area Network
XCoPred	Cross-Correlation Based Prediction
XML	eXtensible Markup Language
ZSS	Zone Server Selection
2D	2 Dimensional

Curriculum Vitae

Károly Farkas received the M.Sc. degree in Computer Science in 1998 and a postgraduate degree in Bank Information Technology in 1999, both from Budapest University of Technology and Economics (BUTE), Hungary. He carried out his Master's thesis project at the research laboratory of Telia, the Swedish national telecommunication company, in Stockholm, Sweden. Then he had spent 3 years as research and teaching assistant at BUTE in Budapest before he joined in 2001 the Communication Systems Group (CSG), led by Prof. Dr. Bernhard Plattner, of the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland.

His research interests cover the field of communication networks, especially autonomic, self-organized and wireless mobile ad hoc networks. His core research area deals with service provisioning in mobile ad hoc networks which led to the SIRAMON framework and became the topic of his Ph.D. thesis. Based on SIRAMON, he developed a business plan by which he managed to get in the winners of the 'Venture Leaders 2006' contest and he was the School Winner of the Young Entrepreneur Price, issued by the French External Trade Advisors, at ETH Zurich in 2006. He has published more than 30 scientific papers in different journals, conferences and workshops and he has given a plenty of regular and invited talks.

In the years past, he supervised a number of student theses, participated in several research projects, coordinated the preparation of an EU IST research project proposal and acted as reviewer and organizer of numerous scientific conferences. He served in the program committees of the 1st and 2nd International Conference on Telecommunications and Computer Networks (IADAT-tcn 2005, 2006) and he would act as technical co-chair of the 3rd Annual International Wireless Internet Conference (WICON 2007). He is member of the IEEE and fellow of the European Multimedia Academy (EADiM).

Education

1982 - 1989	Primary School, Dabas
1989 - 1993	“Apáczai Csere János” Secondary School, Budapest
1993 - 1998	Graduate Studies, TU Budapest, Faculty of Electrical Engineering and Technical Informatics
	Graduation: M.Sc. in Technical Informatics
1998 - 1999	Postgraduate Studies, TU Budapest
	Graduation: 2nd M.Sc. in Bank Information Technology
1998 - 2001	Ph.D. Studies, Budapest University of Technology and Economics (BUTE, formerly TU Budapest), Faculty of Electrical Engineering and Technical Informatics
2001 - 2006	Ph.D. Studies, Swiss Federal Institute of Technology (ETH) Zurich, Department of Information Technology and Electrical Engineering, Institute of Computer Engineering and Networks Laboratory (TIK)
	(Graduation: Ph.D. in Doctor of Sciences)

Technical Experience

1998, 2000	Project work at Telia Research AB, Stockholm
2003	Joint research project, NTT DoCoMo EuroLab, Munich
2003 - 2005	Webmaster of the CSG research group at ETH Zurich

References

TIK, ETH Zurich	Prof. Dr. Bernhard Plattner (plattner@tik.ee.ethz.ch)
BUTE	Prof. Dr. László Jereb (jereb@hit.bme.hu)

Scientific Publications

Journal Papers

[J10] **Károly Farkas**, Theus Hossmann, Lukas Ruf, Bernhard Plattner: Link Quality Prediction in Wireless Mobile Ad Hoc Networks, *submitted to IEEE Transactions on Mobile Computing*, August, 2006.

[J9] Roland Vida, **Károly Farkas**, Dan Grigoras, Utz Rödig, Jorge sa Silva, Petia Todorova, Andreas Pitsillides, Vasos Vassiliou, Lars Wolf: Mobile and Ambient Environments of the Future - Research Challenges for Wireless Ad-Hoc and Sensor Networks. *China Communications*, 3(3):99-106, June, 2006.

[J8] Qing Wei, **Károly Farkas**, Christian Prehofer, Paulo Mendes, Bernhard Plattner: Context-aware Handover Using Active Network Technology. *Elsevier Computer Networks*, 50(15):2855-2872, October, 2006.

[J7] **Károly Farkas**, Oliver Wellnitz, Matthias Dick, Xiaoyuan Gu, Marcel Busse, Wolfgang Effelsberg, Yacine Rebahi, Dorgham Sisalem, Dan Grigoras, Kyriakos Stefanidis, Dimitrios N. Serpanos: Real-time Service Provisioning for Mobile and Wireless Networks. *Elsevier Computer Communications*, 29(5):540-550, March, 2006.

[J6] **Farkas Károly**: IPv6 - A jövő Internetprotokollja? (IPv6 - The Protocol of Future Internet?). *Invited paper in Híradástechnika (Communications, Hungarian journal)*, LX(2005/10):2-7, October, 2005.

[J5] **Károly Farkas**: IPv6 - das zukünftige Internetprotokoll? (IPv6 - The Protocol of Future Internet?). *Invited paper in Electrosuisse Bulletin SEV/AES (Swiss journal)*, 2005/19:9-12, October, 2005.

[J4] **Károly Farkas**, Lukas Ruf, Martin May, Bernhard Plattner: Generic Service Provisioning Framework for Mobile Networks. *Invited paper in IADAT Journal of Advanced Technology on Telecommunications and Computer Networks*, 1(1):14-16, September, 2005. [26]

[J3] **Károly Farkas**: Traffic Engineering and MPLS in IP networks. *Hungarian Telecommunication*, 2000/VIII:19-22, August, 2000.

[J2] **Károly Farkas**: OMP Technique in IP Traffic Engineering. *Periodica Polytechnica*, 2000/44(1):5-12, April, 2000.

[J1] **Károly Farkas**, Markosz Maliosz: Interactive Television Service with Platform-Independent User Interface on ATM Networks. *Hungarian Telecommunication*, 1998/II:21-25, February, 1998.

Conference and Workshop Papers

[C23] Lukas Ruf, Tilman Wolf, **Károly Farkas**, Bernhard Plattner: Specification of Network Services and Mapping Algorithms. In *Proceedings of the 25th Military Communications Conference (MILCOM 2006)*, Washington, DC, USA, October, 2006.

[C22] **Károly Farkas**, Theus Hossmann, Lukas Ruf, Bernhard Plattner: Pattern Matching Based Link Quality Prediction in Wireless Mobile Ad Hoc Networks. In *Proceedings of the 9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2006)*, Torremolinos, Malaga, Spain, October, 2006. [18]

[C21] **Károly Farkas**, Florian Maurer, Lukas Ruf, Bernhard Plattner: Dominating Set Based Support for Distributed Services in Mobile Ad Hoc Networks. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, Vancouver, Canada, April, 2006. [15]

[C20] **Károly Farkas**, Dirk Budke, Oliver Wellnitz, Bernhard Plattner, Lars Wolf: QoS Extensions to Mobile Ad Hoc Routing Supporting Real-Time Applications. In *Proceedings of the 4th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-06)*, Dubai, UAE, March, 2006. [61]

[C19] Dirk Budke, **Károly Farkas**, Oliver Wellnitz, Bernhard Plattner, Lars Wolf: Real-Time Multiplayer Game Support Using QoS Mechanisms in Mobile Ad Hoc Networks. In *Proceedings of the Third IEEE/IFIP Annual Conference on Wireless On demand Network Systems and Services (WONS 2006)*, Les Ménuires, France, January 2006. [60]

[C18] Lukas Ruf, **Károly Farkas**, Hanspeter Hug, Bernhard Plattner: Network Services on Service Extensible Routers. In *Proceedings of the Seventh Annual International Working Conference on Active and Programmable Networks (IWAN 2005)*, Nice, France, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg New York, November, 2005.

[C17] **Károly Farkas**, Florian Maurer, Bernhard Plattner: Algorithmic Support for Distributed Service Architectures in Mobile Ad Hoc Networks. In *Proceedings of the Second International Conference on Telecommunications and Computer Networks (IADAT-tcn 2005)*, Portsmouth, UK, September, 2005. [14]

[C16] **Károly Farkas**, Bernhard Plattner: Supporting Real-Time Applications in Mobile Mesh Networks, In *Proceeding of MeshNets 2005*, Visegrád-Budapest, Hungary, July, 2005. [25]

[C15] **Károly Farkas**, Lukas Ruf, Martin May, Bernhard Plattner: Real-Time Service Provisioning in Spontaneous Mobile Networks (awarded by E-Next Travel Grant and the Workshop's Travel Grant). In *Proceedings of the Student's Workshop of INFOCOM 2005*, Miami, Florida, USA, March, 2005. [24]

[C14] **Károly Farkas**, Lukas Ruf, Martin May, Bernhard Plattner: Framework for Service Provisioning In Mobile Ad Hoc Networks. In *Proceedings of the First International Conference on Telecommunications and Computer Networks (IADAT-tcn 2004)*, San Sebastian, Spain, December, 2004. [22]

[C13] Lukas Ruf, Arno Wagner, **Károly Farkas**, Bernhard Plattner: A Detection and Filter System for Use Against Large-scale DDoS Attacks in the Internet Backbone. In *Proceedings of the Sixth Annual International Working Conference on Active Networks (IWAN 2004)*, Kansas, USA, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg New York, October, 2004.

[C12] **Károly Farkas**, Lukas Ruf, Bernhard Plattner: Service Provisioning Framework for Self-Organized Networks. *Invited paper at Dagstuhl Seminar on Service Management and Self-Organization in IP-based Networks*, Dagstuhl, Germany, October, 2004. [23]

[C11] Lukas Ruf, Arno Wagner, **Károly Farkas**, Bernhard Plattner: A Flexible Router Platform for Next Generation Network Services. *Invited paper at Dagstuhl Seminar on Service Management and Self-Organization in IP-based Networks*, Dagstuhl, Germany October, 2004.

[C10] Qing Wei, **Károly Farkas**, Paulo Mendes, Christian Prehofer, Bernhard Plattner, Nima Nafisi: Context-aware Handover Based on Active

Network Technology. In *Proceedings of the Fifth Annual International Working Conference on Active Networks (IWAN 2003)*, Kyoto, Japan, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg New York, December, 2003.

[C9] **Károly Farkas**, Polly Huang, Balachander Krishnamurthy, Yin Zhang, Jitendra Padhye: Impact of TCP Variants on HTTP Performance. In *Proceedings of the High Speed Networking'02, International Workshop*, Budapest, Hungary, May, 2002.

[C8] **Károly Farkas**: OMP Simulation Results in IP Networks. In *Proceedings of the Polish-Czech-Hungarian Workshop on Circuit Theory, Signal Processing and Telecommunication Networks*, Budapest, Hungary, September, 2001.

[C7] **Károly Farkas**: IP Traffic Engineering using OMP Technique. In *Proceedings of the 12th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2000)*, Las Vegas, USA, November, 2000.

[C6] **Károly Farkas**, Zoltán Balogh, Henrik Villför: IP Traffic Engineering over OMP Technique. In *Proceedings of the CSCS 2000 International Conference*, Szeged, Hungary, July, 2000.

[C5] **Károly Farkas**, Zoltán Balogh, Henrik Villför: OMP Simulation Results in Opnet Simulation Tool. In *Proceedings of the High Speed Networking'00, International Workshop*, Balatonfüred, Hungary, May, 2000.

[C4] Markosz Maliosz, **Károly Farkas**, István Cselényi: Agent Based Interactive Television Service for an Experimental Multimedia System. In *Proceedings of the APCC/ICCS'98 Conference*, Singapore, November, 1998.

[C3] **Károly Farkas**, István Cselényi, Gábor Fehér: Rule Based Resource Reservation Scheme for Multiparty, Multimedia TeleServices. In *Proceedings of the APCC/ICCS'98 Conference*, Singapore, November, 1998.

[C2] István Cselényi, **Károly Farkas**, Gábor Fehér, Markosz Maliosz, Norbert Végh, Gergely Záruba: SIGNE - A Dream Comes True. In *Proceedings of the High Speed Networking'98, International Workshop*, Balatonfüred, Hungary, May, 1998.

[C1] **Károly Farkas**, Markosz Maliosz: Interactive Television Service with Platform-Independent User Interface on ATM Networks. *TDK'97 Conference*, Technical University of Budapest, Budapest, Hungary, November, 1997.

Theses

[T2] **Károly Farkas**: PET Technologies in Digital Paying Systems. *2nd master thesis in Bank Information Technology*, Technical University of Budapest, Budapest, Hungary, June, 1999.

[T1] **Károly Farkas**: SIGNE Rule-base Subsystem for Multimedia Teleservices. *Master thesis*, Technical University of Budapest, Budapest, Hungary, June, 1998.

Bibliography

[1] C. S. R. Murthy and B. S. Manoj. *Ad Hoc Wireless Networks - Architectures and Protocols*. 2004.

[2] Mobile Entertainment Industry and Culture (mGain). Mobile Entertainment in Europe: Current State of the Art. http://www.mgain.org/MGAIN-wp3-d311-revised-final.pdf, April 2003.

[3] G. Sanders, L. Thorens, M. Reisky, O. Rulik, and S. Deylitz. *GPRS Networks*. 2003.

[4] B. H. Walke, P. Seidenberg, and M. P. Althoff. *UMTS: The Fundamentals*. 2002.

[5] M. S. Gast. *802.11 Wireless Networks: The Definitive Guide (O'Reilly Networking)*. 2002.

[6] Kazaa Peer-to-Peer File Sharing Client, 2006. http://www.kazaa.com.

[7] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. 2001.

[8] 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.

[9] SUN Microsystems. JINI Architecture Specification, November 1999.

[10] Microsoft Corporation. Universal Plug and Play: Background. www.upnp.org/resources/UPnPbkgnd.htm, 1999.

[11] Specification of the Bluetooth System. www.bluetooth.com, December 1999.

[12] M. Bossardt, L. Ruf, R. Stadler, and B. Plattner. A Service Deployment Architecture for Heterogeneous Active Network Nodes. In *IFIP International Conference on Intelligence in Networks (SmartNet)*, Saariselka, Finland, April 2002. Kluwer Academic Publishers.

[13] 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.

[14] K. Farkas, F. Maurer, and B. Plattner. Algorithmic Support for Distributed Service Architectures in Mobile Ad Hoc Networks. In *Proceedings of the Second International Conference on Telecommunications and Computer Networks (IADAT-tcn 2005)*, Portsmouth, UK, September 2005.

[15] K. Farkas, F. Maurer, L. Ruf, and B. Plattner. Dominating Set Based Support for Distributed Services in Mobile Ad Hoc Networks. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, Vancouver, Canada, April 2006.

[16] E. Silva. Management of Distributed Services in MANETs. Master's thesis, ETH Zurich, Computer Engineering and Networks Laboratory, April 2006. MA-2006-07.

[17] T. Hossmann. Mobility Prediction in MANETs. Master's thesis, ETH Zurich, Computer Engineering and Networks Laboratory, April 2006. MA-2006-08.

[18] K. Farkas, T. Hossmann, L. Ruf, and B. Plattner. Pattern Matching Based Link Quality Prediction in Wireless Mobile Ad Hoc Networks. In *Proceedings of the 9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2006)*, Torremolinos, Malaga, Spain, October 2006.

[19] Y. Shang and W. Ruml. Improved MDS-based Localization. In *The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2004)*, Hong Kong, China, March 2004.

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

[21] R. Grueninger. Service Provisioning in Mobile Ad hoc Networks. Master's thesis, ETH Zurich, Computer Engineering and Networks Laboratory, September 2004. MA-2004-12.

[22] K. Farkas, L. Ruf, M. May, and B. Plattner. Framework for Service Provisioning in Mobile Ad Hoc Networks. In *Proceedings of the First International Conference on Telecommunications and Computer Networks, (IADAT-tcn2004)*, San Sebastian, Spain, December 2004.

[23] K. Farkas, L. Ruf, and B. Plattner. Service Provisioning Framework for Self-Organized Networks. In *Online Proceedings of Dagstuhl Seminar on Service Management and Self-Organization in IP-Based Networks, (Dagstuhl Seminar 04411)*, Schloss Dagstuhl, Wadern, Germany, October 2004.

[24] 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, (INFOCOM 2005)*, Miami, Florida, USA, March 2005.

[25] K. Farkas and B. Plattner. Supporting Real-Time Applications in Mobile Mesh Networks. In *Proceedings of MeshNets 2005*, Visegrád-Budapest, Hungary, July 2005.

[26] K. Farkas, L. Ruf, M. May, and B. Plattner. Generic Service Provisioning Framework for Mobile Networks. *IADAT Journal of Advanced Technology on Telecommunications and Computer Networks*, 1(1):14–16, September 2005.

[27] D. Grigoras. Service-Oriented Naming Scheme for Wireless Ad Hoc Networks. In *Proceedings of the NATO Advanced Research Workshop, (CIPC 2003)*, Sinaia, Romania, July 2003.

[28] S. Helal, N. Desai, V. Verma, and C. Lee. Konark – A Service Discovery and Delivery Protocol for Ad-hoc Networks. In *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC)*, New Orleans, USA, March 2003.

[29] U. C. Kozat and L. Tassiulas. Network Layer Support for Service Discovery in Mobile Ad Hoc Networks. In *Proceedings of the IEEE INFOCOM*, San Francisco, USA, April 2003.

[30] SIRAMON: Service provIsioning frAMework for self-Organed Networks. http://www.siramon.org.

[31] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit Disk Graphs. *Discrete Mathematics*, 86:165–177, 1990.

[32] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, New York, USA, 1983.

[33] H. Zhou, L. Ni, and M. Mutka. Prophet Address Allocation for Large Scale MANETs. In *Proceedings of the The 22nd Annual Conference on Computer Communications and Networking, (INFOCOM 2003)*, San Francisco, USA, April 2003.

[34] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, December 2002.

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

[36] 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.

[37] Theodore Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[38] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical report, 1995.

[39] Z. R. Zaidi and B. L. Mark. Real-Time Mobility Tracking Algorithms for Cellular Networks Based on Kalman Filtering. *IEEE Transactions on Mobile Computing*, 4(2):195–208, March/April 2005.

[40] J. Lewis. Fast Normalized Cross-Correlation. *In Vision Interface*, 1995.

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

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

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

[44] Information Sciences Institute ISI. Nam: Network Animator, July 2003. http://www.isi.edu/nsnam/nam/.

[45] C. Meyer and P. Baenziger. Multiplayer Gaming in Mobile Ad Hoc Networks. Semester Thesis, ETH Zurich, Computer Engineering and Networks Laboratory, March 2006. SA-2006-04.

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

[47] IEEE-SA Standards Boards. Part11-wireless lan medium access control (mac) and physical layer (phy) specifications. 12 June 2003. http://standards.ieee.org/getieee802/802.11.html.

[48] J. Bardwell. Converting Signal Strength Percentage to dBm Values, November 2002.
http://web.archive.org/web/20040405102601/http://www.wildpackets.com/-elements/whitepapers/Converting_Signal_Strength.pdf.

[49] Sam Leffler. Multiband Atheros Driver for WiFi (MADWIFI), 2005. http://sourceforge.net/projects/madwifi/.

[50] Entertainment Software Association. 2005 Sales, Demographics and Usage Data - Essential Facts about the Computer and Vido Game Industry, 2005.

[51] T. Henderson and S. Bhatti. Networked Games: A QoS-Sensitive Application for QoS-Insensitive Users. In *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*, Karlsruhe, Germany, August 2003.

[52] M. Dick, O. Wellnitz, and L. Wolf. Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames 2005)*, Hawthorne, USA, October 2005.

[53] G. Armitage and L. Stewart. Limitations of Using Real-World, Public Servers to Estimate Jitter Tolerance of First Person Shooter Games. In *Proceedings of ACM SIGCHI ACE2004*, Singapore, June 2004.

[54] L. Pantel and L. Wolf. On the Suitability of Dead Reckoning Schemes for Games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames 2002)*, April 2002.

[55] J. Müller and S. Gorlatch. GSM: A Game Scalability Model for Multiplayer Real-time Games. In *Proceedings of The 24th Annual Conference on Computer Communications and Networking, (IEEE INFOCOM 2005)*, Miami, Florida, USA, March 2005.

[56] J. Broch, D. A. Maltz, D. B. Johnson, Y-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 85–97, October 1998.

[57] T. S. Rappaport. *Wireless Communications, Principles and Practice*. PRENTICE HALL INTERNATIONAL, 1996. ISBN: 0-13-042232-0.

[58] C. E. Perkins, editor. *Ad Hoc Networking*. 2001.

[59] D. Budke. Quality of Service for Multiplayer Game Provisioning in Mobile Ad Hoc Networks. Master's thesis, TU Braunschweig and ETH Zurich, September 2005.

[60] D. Budke, K. Farkas, O. Wellnitz, B. Plattner, and L. Wolf. Real-Time Multiplayer Game Support Using QoS Mechanisms in Mobile Ad Hoc Networks. In *Proceedings of The Third Annual Conference on Wireless On demand Network Systems and Services (WONS 2006)*, Les Ménuires, France, January 2006.

[61] K. Farkas, D. Budke, O. Wellnitz, B. Plattner, and L. Wolf. QoS Extensions to Mobile Ad Hoc Routing Supporting Real-Time Applications. In *Proceedings of the 4th ACS/IEEE International Conference on*

Computer Systems and Applications (AICCSA-06), Dubai, UAE, March 2006.

[62] K. Fall. Ns Notes and Documentation. *The VINT Project*, February 2000.

[63] Marc Bechler Sven Jaap and Lars Wolf. Evaluation of Routing Protocols for Vehicular Ad Hoc Networks in Typical Road Traffic Scenarios. In *The 11th Open European Summer School (EUNICE 2005)*, Colmenarejo, Spain, July 2005.

[64] G. S. Lauer. Hierarchical Routing Design for SURAN. In *Proceedings of IEEE ICC'86*, pages 93–102, ???, June 1986.

[65] R. Ramanathan and M. Steenstrup. Hierarchically-Organized, Multihop Mobile Wireless Networks for Quality-of-Service Support. *ACM/Baltzer Mobile Networks and Applications Journal*, 3(1):101–119, January 1998.

[66] R. Wattenhofer. Chapter 8 - Dominating Sets. Course material mobile computing, Distributed Computing Group, ETH Zurich, 2004. http://dcg.ethz.ch/lectures/ss04/mobicomp/index.html.

[67] J. Wu and H. Li. *Handbook of wireless networks and mobile computing*, chapter A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks, pages 425–450. 2003. ISBN:0-471-41902-8.

[68] F. Kuhn and R. Wattenhofer. Constant-Time Distributed Dominating Set Approximation. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'03)*, pages 25–32, Boston, Massachusetts, USA, July 2003.

[69] K. M. Alzoubi, P-J. Wan, and O. Frieder. Message-optimal Connected Dominating Sets in Mobile Ad Hoc Networks. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing 2002*, pages 157–164, Lausanne, Switzerland, June 2002.

[70] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivasan. Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms 2003*, pages 717–724, Baltimore, Maryland, USA, January 2003.

[71] T. Acharya and R. Roy. Distributed algorithm for power aware minimum connected dominating set for routing in wireless ad hoc network. In *Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW'05)*, pages 387–394, Washington, DC, USA, 2005. IEEE Computer Society.

[72] D. J. Baker and A. Ephremides. A Distributed Algorithm for Organizing Mobile Radio Telecommunication Networks. In *Proceedings of the 2nd International Conference in Distributed Computer Systems*, 1981.

[73] M. Gerla and Tsai J. Multicluster, Mobile, Multimedia Radio Network. *ACM/Baltzer Journal of Wireless Networks*, 1(3):255–265, 1995.

[74] J. A. Shaikh, J. Solano, I. Stojmenovic, and J. Wu. New Metrics for Dominating Set Based Energy Efficient Activity Scheduling in Ad Hoc Networks. In *Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks (LCN'03)*, 2003.

[75] M. Chatterjee, S. Das, and D. Turgut. WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 5:193–204, 2002.

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

[77] W. Su, S. Lee, and M. Gerla. Mobility Prediction and Routing in Ad Hoc Wireless Networks. *International Journal of Network Management*, 2000.

[78] Z. R. Zaidi and B. L. Mark. Mobility Estimation for Wireless Networks Based on an Autoregressive Model. In *Proceedings of IEEE GLOBECOM 2004*, Dallas, Texas, USA, Nov./Dec. 2004.

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

[80] Joe Capka and Raouf Boutaba. Mobility Prediction in Wireless Networks using Neural Networks. In *7th IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS 2004)*, San Diego, CA, USA, October 2004.

[81] Amiya Bhattacharya and Sajal K. Das. LeZi-Update: An Information-Theoretic Approach to Track Mobile Users in PCS Networks. In *Mobile Computing and Networking*, 1999.

[82] Z. Yang and X. Wang. Joint Mobility Tracking and Hard Handoff in Cellular Networks via Sequential Monte Carlo Filtering. In *Proceedings of The 21st Annual Conference on Computer Communications and Networking, (IEEE INFOCOM 2002)*, volume 2, pages 968–975, New York, NY, USA, June 2002.

[83] P. Bahl and V.N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *Proceedings of The 19th Annual Conference on Computer Communications and Networking, (IEEE INFOCOM 2000)*, volume 2, pages 775–784, Tel-Aviv, Israel, March 2000.

[84] T. Liu, P. Bahl, and I. Chlamtac. Mobility Modeling, Location Tracking, and Trajectory Prediction in Wireless ATM Networks. *IEEE Journal on Selected Areas in Communications*, 16(6):922–936, August 1998.

[85] 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.

[86] M. Treiber and D. Helbing. Realistische mikrosimulation von strassenverkehr mit einem einfachen modell. In *16. Symposium "Simulationstechnik" ASIM 2002, Tagungsband, Rostock*, 2002.