Abdalkarim Awad

Efficient Data Management in Wireless Sensor Networks using Peer-to-Peer Techniques

Dissertation im Fach Informatik

Lehrstuhl für Informatik 7
Rechnernetze und Kommunikationssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

University of Erlangen · Dept. of Computer Science 7
Martensstr. 3 · 91058 Erlangen · Germany
www7.informatik.uni-erlangen.de

TECHNISCHE FAKULTÄT

Efficient Data Management in Wireless Sensor Networks using Peer-to-Peer Techniques

—

Effizientes Datenmanagement in drahtlosen Sensornetzen durch Peer-to-Peer Methoden

Der Technischen Fakultät der

Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Abdalkarim Awad

Erlangen, den 6. Juli 2009

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	**6. Juli 2009**
Tag der Promotion:	**30. November 2009**
Dekan:	**Prof. Dr.-Ing. Reinhard German**
Berichterstatter:	**PD Dr.-Ing. Falko Dressler**
	Prof. Dr.-Ing. Wolfgang Schröder-Preikschat

I dedicate this work to my mother, to my wife
and

to my sons Isaac, Adam and Elias

Acknowledgments

I'm indebted to many individuals who helped me during my doctoral studies. Without them, this dissertation would not have come into existence!

Firstly, I would like to express my appreciation to Prof. Reinhard German for giving me the opportunity to pursue my post graduate studies in the Computer Networks and Communication Systems group. I benefited tremendously from his wise counsel and indispensable advice. Thanks for continuous encouragement, support, and insightful comments. You have been a great supervisor.

Words cannot express my gratitude to Dr. Falko Dressler, the coordinator of our research group, for his patience and guidance. He has been infinitely patient and extremely supportive. I am blessed to have had him as my adviser. My first instinct in writing this was to say that Falko has been so much more than just an advisor. On second thought, an advisor is exactly what Falko has been -an incredible one- fulfilling each of the many roles required of an advisor to perfection. Falko is an outstanding teacher and a source of inspiration for me.

I am grateful to the other members of my thesis committee, especially to Prof. Wolfgang Schröder-Preikschat for his effort and the time he invested in supporting my research work.

I would also like to express my cordial thanks my fellow postgraduate students, Christoph Sommer, Isabel Dietrich and Tobias Limmer. They provided me constant encouragement and invaluable suggestions on my research, especially during the writing phase of my PhD thesis. I would like also to thank Ulrich Klehmet and Armin Heindl.

Appreciation is also extended to all the staff members in the Computer Networks and Communication Systems group, for their efforts in providing a friendly research environment. Special grateful thanks go to the active secretaries of the department Erika Hladky and Gerti Pastore for their willingness to help in any situation. Of course, I cannot forget Chris Moog for her help in developing electronic boards.

Particular acknowledgment is due to DAAD (German Academic Exchange Service) for the financial support during the study period in Germany.

Abstract

The data-centric nature of Wireless Sensor Networks (WSNs) and the severe resource constraints of the sensor nodes distinguish sensor data management from other communication networks. Both the client-server approach as well as the end-to-end communication principle that have been proposed in the for Mobile Ad Hoc Networks (MANETs) do not suit the characteristics of WSNs. Efficient lookup and routing of sensor data are of great significance for WSNs, especially as the size of these networks continues to grow. On an abstract level, structured peer-to-peer protocols, which rely on Distributed Hash Tables (DHTs) provide $O(1)$ complexity for storing and retrieving data in the network, seem overcome these restrictions. However, the fact that they rely on underlayer routing techniques leads to inefficient resource usage in the context of sensor networks. The combination of DHTs and underlayer routing led to the establishment of so called virtual coordinate routing techniques. Most of these algorithms are quite complicated and do not guarantee packet delivery on the shortest path. Additionally, only few of them are implemented in real sensor nodes. In this dissertation, we present the Virtual Cord Protocol (VCP), a virtual position based routing protocol that also provides means for data management such as identifying, storing, and retrieving data items. The key contributions of this protocol are independence of real location information by relying on virtual relative positions of neighboring nodes, the simplicity of obtaining the virtual positions, near optimal routing paths, and high scalability because only information about direct neighbors is needed for routing. Furthermore, VCP provides a unique position for each node and inherently prevents dead-ends. We extensively evaluated VCP in a number of simulation experiments. The simulation results show that VCP consistently provides high throughput and low overhead for a wide range of application scenarios. We compared VCP with Dynamic MANET on Demand (DYMO), a typical MANET protocol, and with Virtual Ring Routing (VRR), another virtual coordinate

based approach. In static networks, both VCP and VRR clearly outperform DYMO. In the case of frequent node failures, however, VCP benefits from its light-weight design. Our protocol is more failure tolerant compared to VRR. We finally integrated data replication techniques that support high success rates even in very unreliable sensor networks. A prototype implementation on real sensor nodes outlines the feasibility of our approach in a proof-of-concept study.

Kurzfassung

Die datenzentrische Arbeitsweise von drahtlosen Sensornetzen sowie die kritischen Ressourcenbeschränkungen von Sensorknoten unterscheiden die Datenverarbeitung in diesem Bereich von klassischen Kommunikationsnetzen. Sowohl der Client-Server-Ansatz als auch die Ende-zu-Ende Kommunikationsparadigmen, welche u.a. auch für mobile ad-hoc Netze erfolgreich eingesetzt werden, sind nur bedingt für Sensornetze nutzbar. Von besonderer Bedeutung in Sensornetzen sind effiziente Identifikations- und Routingmethoden. Dies betrifft vor allem die Skalierbarkeit für größer werdende Netze. Auf einer abstrakten Ebene scheinen strukturierte Peer-to-Peer-Ansätze eine Alternative darzustellen, da durch den Einsatz verteilter Hash-Tabellen eine $O(1)$ Komplexität für die Identifikation von Daten erreicht werden kann. Allerdings bedürfen diese Verfahren einer stabilen Routing-Infrastruktur auf den unteren Schichten. Die Kombination von verteilten Hash-Tabellen und Routingverfahren führte zur Entwicklung sogenannter Virtuelle-Coordinaten-Algorithmen. Viele dieser Methoden sind allerdings sehr kompliziert und geben keine Garantie für den Datenversand auf einem kürzesten Weg. Weiterhin sind nur wenige in einer realen Sensorumgebung implementiert. In dieser Dissertation wird das Virtual Cord Protocol (VCP) vorgestellt, welches virtuelle Knotenpositionen für das Datenmanagement und den Datentransport einsetzt. Die wesentlichen Beiträge können wie folgt zusammengefasst werden: Unabhängigkeit des Protokolls von realen Knotenpositionen, einfache Definition und Verwaltung der virtuellen Koordinaten, kurze Routingpfade nahe dem kürzesten Weg, hohe Skalierbarkeit durch die Beschränkung auf lokal verfügbare Nachbarschaftsinformationen. Weiterhin verwaltet VCP eindeutige Knotenadressen und vermeidet konzeptuell mögliche Sackgassen. VCP wurde im Rahmen der Arbeit in umfangreichen Simulationsexperimenten bewertet. Die Ergebnisse zeigen, dass VCP hohen Durchsatz und geringe Kosten für einen breiten Einsatzbereich ermöglicht. In einem direkten Vergleich mit Dynamic

MANET on Demand (DYMO), einem typischen ad-hoc Routing Protokoll, sowie Virtual Ring Routing (VRR), einem kompetitiven Ansatz basierend auf virtuellen Koordinaten, zeigte sich, dass VCP und VRR in statischen klassischen ad-hoc Verfahren überlegen sind. Für dynamische Netze mit häufig ausfallenden Knoten dominiert allerdings VCP durch sein leichtgewichtiges Design. Das Protokoll ist stärker fehlertolerant als VRR. Weiterhin wurden Replikationsmethoden integriert, die hervorragende Erfolgsraten auch in sehr unzuverlässigen Sensornetzen ermöglichen. Ein implementierter Prototyp zeigt die Fähigkeiten des neuen Ansatzes in einer Proof-of-Concept Studie.

Contents

Chapter 1

Introduction

The Semiconductor technology has made dramatic advances in the last decades. The raw cost of fabricating a given logic element (processor, memory, peripheral) has been always decreasing over the last four decades. Another way of looking at the technology advances is to compare the memory sizes which can be achieved on a single chip: from a few kByte in early 1970s sizes have increased to some GByte in late 2000s. In particular, the development of new technologies in the last decade, such as the widely used System-on-a-Chip approach, has produced small, embedded devices which offer several capabilities within one compact device. An important example of these embedded devices is the *sensor node*, which has sensing and communicating capabilities in addition to the traditional computing capabilities. Sensor nodes represent consequential improvement over traditional sensors [1]. Sensor nodes are capable of not only to measure physical phenomena, but also to process, store and disseminate these measurements. Wireless Sensor Networks (WSNs) are composed of cooperating sensor nodes that build an ad hoc network using wireless radio communication. Sensor Nodes can observe the environment to monitor physical phenomena and events of interest [2, 3]. Usually sensor nodes are deployed randomly inside an area of interest or in close proximity. These sensor nodes will perform significant signal processing, computation, and network self-configuration to achieve scalable, robust and long-lived networks.

Earlier generations of sensor networks which were deployed over a regular topographic mesh, like the radar network used in air-traffic control and nationwide weather stations, use special computers and communication protocols, which make them very expensive. For some application scenarios a network of sensors and actuators can be

built using the existing wired technologies. For many other application types, however, wiring is not practical, expensive, and can prevent the sensors from being close to the phenomenon [4]. Therefore WSNs present a cost-effective, practical and capable path to support many real-world applications. In spite of the fact that the capabilities of sensor nodes are very limited, WSN application domains are diverse and they can operate a variety of data types, including simple payload such as temperature, light intensity, and humidity, or more complex data types such as sound and images, or even more complicated data like video.

Specialists generally agree that WSNs will be made possible not only by the price-performance revolution of microelectronics but also by the development of efficient algorithms that suit the special characteristics of the sensor nodes. The advent of WSNs is rapidly changing the way by which data is captured, processed and disseminated for a variety of potential applications for homes, hospitals, factories, road traffic, environmental monitoring and so on [5]. WSNs are data-centric in nature, hence they can be treated as virtual distributed databases of continuous data streams from the physical world. Unlike traditional data communication networks, a sensor node may not need (and may not offer) an identity (e.g., an address) [1, 6]. That is, sensor network applications are unlikely to answer questions like "is there a bird at sensor N51 now?" Rather, applications focus on the data collected by sensors. Data is commonly annotated with attributes, enabling applications to request data matching certain attribute values. So, the users are more likely to be interested in questions such as: "when was a detection bird near the lake today?" or "how many birds were detected near the lake in the period between 2^{nd} and 18^{th} of July ?". This approach decouples data from the sensor that produced it. This allows for more robust application design: even if sensor node N51 dies, the data it generates can be replicated in other (possibly neighboring) sensors for later retrieval. The unique characteristics of WSNs, such as limited power sources, small storage capacity, or limited and poor connectivity, impose several and often conflicting design goals. A number of required characteristics when designing data management algorithms for WSNs are listed below:

- Self-organizing: the algorithm should autonomously adapt to changes resulting from external interventions, such as topological changes (e.g., node failure) without the influence of a centralized entity.

- Scalable: to provide scalability, protocols designed for WSNs have to be distributed and operate only on local information.
- Minimal administration: the algorithm should adjust its operational parameters according to the design requirements.
- Energy-efficient: in many scenarios sensor nodes have to rely on a battery. Replacing batteries is usually not practical, sometimes even impossible, hence the lifetime of a WSNs becomes a very important figure of merit. Evidently, because data transmission is the largest source of power consumption, employing an energy-efficient routing protocol is necessary.
- Fault-tolerant: although sensor nodes are assumed to be stationary, WSNs should be resilient to failures such as the physical destruction of nodes, energy depletion or communication link errors.
- Collaborative: sometimes a single node is not able to decide if an event has happened and only the joint data from several nodes provide enough information, so several nodes have to collaborate to detect the event. Collaborative behavior also allows for in-network processing of data instead of sending all raw data to an external computer. Finally, collaborative behavior can also entail the sharing of nodes' resources (e.g., storage capacity).
- Lightweight: because of low storage and computing capabilities on sensor nodes, a lightweight protocol stack is desired.
- Peer-to-Peer: the existence of a central entity in the network hinders scalability, therefore instead of relying on a client-server architecture, peer-to-peer communication is preferable.

Of course the challenges presented above certainly do not exhaust the design space of WSNs. Issues such as programmability and maintainability are also important, however they are less important for data management approaches.

1.1 Motivation

Many of the lessons learned from protocols used extensively on the Internet can be applied to the design of sensor network algorithms. Yet, they need to be adapted to

the special characteristics of WSNs. Over the past ten years, an enormous number of distributed and self-organizing Peer-To-Peer (P2P) systems have been designed and developed. Many of these systems are very successful and their use accounts for a sizable portion of today's world-wide Internet traffic. Motivated by individuals' contribution of file sharing, the sharing of data between users (peers), which are of equal rights, is the main application of these systems. At the beginning P2P systems have used either a central server to locate the data, which has the drawback that the system suffers from a single, central point of failure, or they have used a flooding technique to locate the data, which has the drawback that the scalability of these systems was bad. Since then, further advances have been achieved and new techniques have appeared. The reliance on the concept of distributed indexing is the main characteristic of these new techniques. A central idea was to use Distributed Hash Tables (DHTs) to associate nodes to data, so finding the node that hosts a specific piece of data would be an easy task. Because WSNs have very limited resources in terms of battery life, computational power and communication capabilities, efficient query dissemination and routing techniques are highly needed.

There are some commonalities between WSNs and typical P2P approaches that motivate our research on using peer-to-peer techniques for efficient data management in WSNs; below we list the major commonalities between WSNs and typical P2P systems, with the term node referring to both a peer in P2P systems and a sensor node in sensor networks:

- Sharing of resources: within a set of nodes, each utilizes its own as well as the resources provided by other nodes. The most prominent examples for such resources are the storage and processing capacity.

- Decentralized architectures: systems have no notion of global coordination at all.

- Transient connectivity: nodes in P2P systems frequently go off-line as a result of poor connectivity or of users leaving the system. In WSNs nodes fail for different reasons such as physical destruction of nodes, energy depletion or communication link errors.

- Equal rights and duties: nodes are equal partners with symmetric functionality. Each node is fully autonomous regarding its respective resources.

- Identity management: a node's Id typically changes so the node is not constantly reachable at the same address. Because of the transient connectivity, nodes are dynamically assigned a new Id every time they connect to the network.
- Routing and message forwarding: communication is handled entirely by nodes operating at a local level. Usually network communication implies the presence of a forwarding mechanism, i.e. nodes are forwarding messages on behalf of other nodes.

In addition to these similarities there are also differences that must be cared for. For example, sensor nodes usually have very limited computing, memory, and energy resources. In contrast peers in P2P systems are capable computers which have ample computing and memory resources and sustained power supply. Another important difference is the separation of physical and logical networks, which complicates the use of P2P concepts in WSNs.

1.2 Contribution

The main contribution of this thesis is VCP, a routing algorithm designed for WSNs. VCP is a DHT-like protocol that offers in addition to standard DHT functions (*put* and *get*) an efficient routing mechanism. Based on a pre-defined range of positions VCP derives relative positions to the nodes without flooding the network or considering the physical location of the nodes.VCP does not require nodes to have a global identity – in fact a local identity is enough. Using only the physical neighbors information, VCP guarantees packet delivery between any two joined nodes in the network. VCP is a light weight routing algorithm that can be implemented on top of any Medium Access Control (MAC) layer that provides a simple broadcast mode. In addition, we describe two extensions of VCP; efficient routing in the existence of frequent node failure and data replication schemes. Greedy forwarding guarantees packet delivery in static networks. However in the presence of frequent node failure, it was necessary to extend the routing approach to ensure high delivery ratios even if all nodes in the network do not have efficient communication capabilities. Data replication is important to ensure persistence of data in the existence of frequent node failure. We proposed two local replication schemes in addition to a global replication one. The main advantage of the proposed schemes is the on-demand and the locality of data copies transfer.

Throughout the thesis we used the terms “physical neighbors” and “radio range neighbors” interchangeably at which we always mean single-hop neighbors. Also when we say “VCP guarantees packet delivery ” we mean that VCP finds a route to the destination, however VCP does not use end-to-end acknowledgment.

1.3 Organization of the Dissertation

We will show that 1-D virtual Coordinates are enough to do efficient routing and data lookup in WSNs and requires the propagation of virtual position information for only a single hop: each node need only know its neighbors’ virtual positions. The relative position of nodes is the key to efficient routing without physical positions of nodes or a global overview of the entire network. The position of a packet’s destination and positions of the candidate next hops are sufficient to make correct forwarding decisions, without any other topological information.

- In Chapter 2 we describe some background on data management issues in WSNs. We concentrate on a performance comparison between different data lookup techniques and the requirements and implications of using each technique. We further introduce some details of the WSNs and P2P systems.
- We provide an overview in Chapter 3 of the related and previous literature on routing and data management in WSNs.
- The detailed design of VCP is presented in Chapter 4. We start with a detailed introduction of the join operation in VCP. We then describe the greedy routing approach of VCP at which point we prove that in static networks greedy forwarding guarantees packet delivery and is loop free. We describe also how to route in the existence of node failure and introduce three schemes for data replication. Finally we introduce further extensions to VCP.
- In Chapter 5, we evaluate VCP in simulations of different wireless networks scenarios, including the simulation of the full IEEE 802.11 physical and MAC layers. We show that VCP delivers user packets robustly, generates small routing protocol overhead,and delivers the vast majority of packets over near optimal paths. We simulate Dynamic MANET on Demand (DYMO) and Virtual Ring Routing (VRR) for comparison on the same networks and show that VCP performs

similar (or a little bit better) to VRR in static scenarios and outperforms DYMO. We also show that in dynamic networks with frequent node failure VCP outperforms VRR.

- In order to show the feasibility of implementing VCP in real life, in Chapter 6 we present a prototype implementation of VCP in the lab. We implemented VCP on sensor nodes called BTnodes, which has low-power radio channel and employed Berkeley MAC (BMAC) at the MAC layer.

- We conclude the thesis in Chapter 7 and we give possible direction for future research.

Chapter 2

Data Management in Wireless Sensor Networks

Data management in WSNs poses new challenges in comparison with the rich capabilities of traditional data base systems. The challenge comes from the low capabilities of WSNs. In most cases the challenges can be reduced to a single problem: *where to store and how to find a certain sensor reading (detected event) in a WSNs with efficient usage of the available resources (battery, bandwidth, storage) and without centralized control or coordination?*

The lookup problem can be defined as follows: some sensor node A wants to store a data item D in the WSN. D may be some (small) measurement, the location of some larger content, or coordination data, e.g. the address of the base station, etc. Then we assume that a node B wants to retrieve the data item later. The interesting questions are now:

- Where should A store the data item D?
- How do other nodes, e.g. node B, discover the location of D?
- How can the WSN be organized to assure scalability and efficiency?
- How to use nodes' resources efficiently?

Traditional approaches that have been used for tens of years can be adapted to the capabilities of WSNs. In this thesis we will leverage techniques used in P2P systems, namely the idea of DHTs to manage data efficiently in WSNs. A lot of work has been done recently to adapt techniques, such as traditional compression algorithms, which usually need a large amount of memory to work on few tens of kByte [7–9] in order

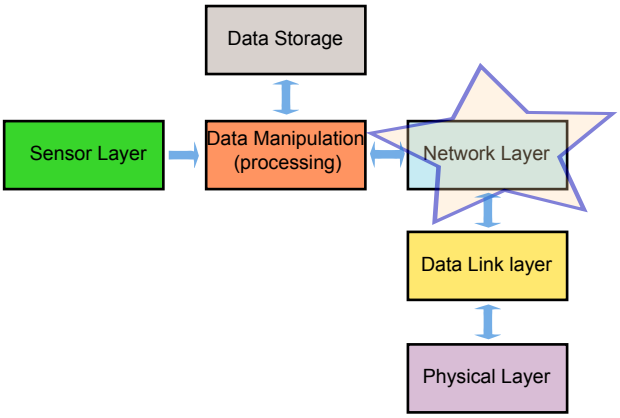

Figure 2.1: Sensor layers

to suit WSNs. There are several angles from which efficient data management can be viewed, from the collection of raw data to the reception of the data by the end user. Some approaches concentrate on data reduction, by compressing or aggregating the data, other approaches concentrate on traffic reduction, i.e efficient data look up and routing algorithms, as a key to reduce the communication cost. VCP presents a protocol to provide efficient routing as well as efficient data lookup mechanisms. Figure 2.1 shows a typical sensor network layer stack; a sensor measures a specific phenomenon, the resulting reading can be processed and stored locally or transmitted. In this chapter we will go over the these blocks and concentrate exclusively on the blocks that are related to data management in WSNs. VCP is located in the network block, but it can also find the storage place of a specific data item efficiently. Before we start to discuss approaches and preliminarily requirements used in the literature toward efficient data management in WSNs, we will introduce the basics of WSNs.

2.1 Wireless Sensor Networks

Tremendous advances have been made in sensor technology and many more are on the horizon [10]. A sensor is a device that converts a physical phenomenon into an electrical signal. As such, sensors represent part of the interface between the physical world and the world of electrical devices, such as computers. The other part of this interface is represented by actuators, which convert electrical signals into physical phenomena [10]. Usually a sensor node has more than one type of sensor to sample physical phenomena.

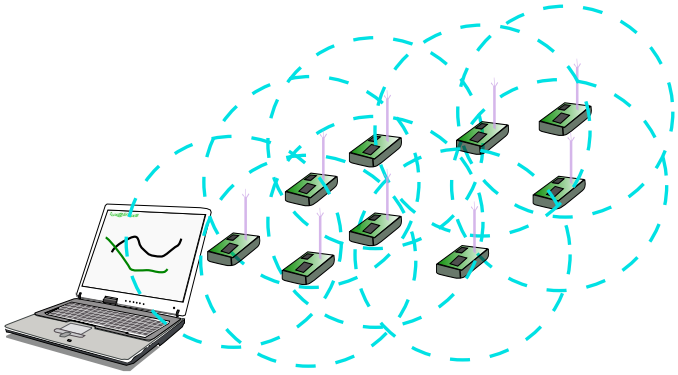

Figure 2.2: A typical wireless sensor network

The nature of the phenomena that have to be sensed determine the needed sensor. In turn, the sensor type determines the produced data type. Moreover, the required precision can have an effect on the size of the produced data for the same phenomenon. WSNs are composed of cooperating sensor nodes that build an ad hoc network using wireless radio communication [2, 3].

The main function of these networks is to sample the surrounding environment using the installed sensors [1]. In the early beginnings, these networks were considered to consist of a few sensor nodes connected to a central control unit. Today, however, WSNs are used with quite a large number of nodes. The WSNs design often employs approaches such as energy-aware techniques, in-network processing, multi-hop communication, and density control techniques to extend the network lifetime [11]. Figure 2.2 shows a typical WSN in which nine sensor nodes are deployed. Normally when a node detects an event it sends this event (maybe over multi-hop) to a base station, which is usually located outside the network.

2.1.1 General Characteristics of Sensor Nodes

The attractive characteristics of sensor nodes, like their price and their compactness, come at a price – the capabilities of sensor nodes are severely limited. Figure 2.3 depicts the basic components of a typical sensor node. In the following an overview of the characteristics of sensor nodes is presented.

- Computing: wireless sensor nodes usually use small, low power, low speed micro controllers (an Atmel ATmega128, for example) which can offer basic computing operations.

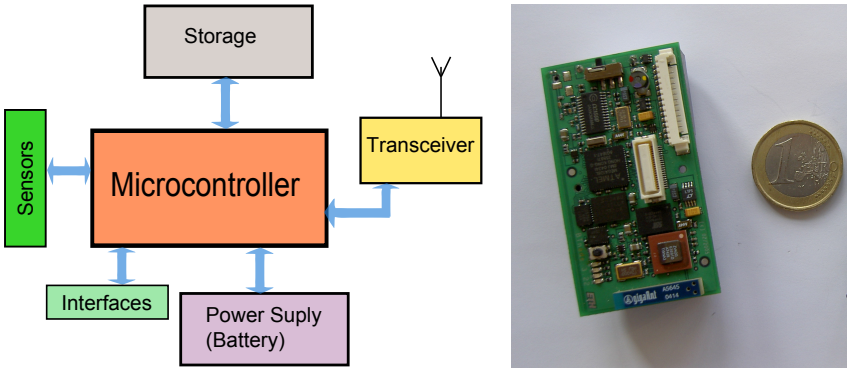

Figure 2.3: Sensor Node: (left) Components typically found on sensor nodes, (right) Sensor node BTnode developed at the ETH Zurich [2]

- Storage: normally the micro controller runs on only few kByte of RAM. Programs are stored in a few hundred kByte or a few MByte of flash memory, instead of the hard disks used on normal computers. Sometimes it also includes a few kByte of EEPROM which can be used for configuration.

- Communication: the communication capabilities suffer from short ranges and poor connectivity. Usually a low power radio transceiver (for example a Chipcon CC1000 operating in ISM band 433-915 MHz) is used. Sometimes the nodes offer only broadcasts as a communication paradigm [1].

- Power supply: the most important characteristic of sensor nodes is their relying on batteries. Because the power is drained mostly by communication between nodes [12, 13], this raises special concerns with regard to routing data between nodes.

- Interfaces: usually senor nodes include different interfaces to connect sensors and actuators. Interfaces may be serial (like UART, I2C), or parallel (like GPIO). Most micro controllers used in sensor nodes also include analog inputs which are connected to an internal ADC. These interfaces can be used to connect analog sensors.

- Sensors: different low power sensors are either built-in on the sensor node or attached using the interfaces. Sensor types include temperature, light, moisture and pressure sensors.

Also, in addition to the above limitations, in some cases sensor nodes do not even have a unique identifier [1]. Taking into consideration Moore's Law it is possible to produce these devices at a very low price, which makes it possible to use a large number of nodes. However, to really benefit from these sensor networks, they have to operate in a cooperative, distributed and self organized manner.

2.1.2 Applications

Although the first applications envisioned for WSNs were in the military field and therefore a typical application is battlefield surveillance, there is a broad domain of applications of WSNs in the civilian sector.

- Environmental monitoring is an appealing area for WSNs. In [14], a wireless sensor network is deployed to monitor a bird (Storm Petrels) and its surroundings on a small island off the coast of Maine. Some sensors have been placed inside the burrows of the bird, other sensors were deployed in the vicinity. Examples of other application fields in the environment that WSNs can be involved in include Structural Health monitoring [15] and volcano monitoring [16].

- Health monitoring is a good example of applying WSNs to improve the quality of life of the human being. In [17], a smart home scenario is presented. The applicability of using WSNs to ease the life of elderly and handicapped people is investigated. For example, the WSN can help the doctor to remotely monitor the health of the inhabitant of this smart home. WSNs can also be used to optimize the home's energy consumption, make emergency calls and secure the home.

- Awareness of location of objects is important for business administration, transportation, logistics and many other applications. In [18] an adaptive and cost-effective approach based on the Received Signal Strength Indicator (RSSI) values on sensor nodes was used to localize a mobile robot.

2.2 Data Transmission

Data transmission is the primary source of power consumption in a WSN [12, 13]. A lot of work has been done to optimize the data transmission on physical, data link and network layers. Basic physical layer duties are frequency selection and modulation. An

option for the communication channel is to use the Industrial, Scientific and Medical (ISM) bands, which offer license-free communication in most countries [19]. The data link layer is primarily responsible for medium access and error control. In [20], the authors study packet delivery performance at the physical and MAC layers in dense wireless sensor networks. They perform the experiments in three different environments: an indoor office building, a habitat with moderate foliage, and an open parking lot. Measurements indicate a significant asymmetry in realistic environments, and therefore the performance in these environments is fairly pessimistic.

This thesis focuses on the network layer, which is mainly responsible for routing strategies. The protocol proposed in this thesis, VCP, provides services for the network (routing) as well as for the application layers. In this section, we present a general survey of routing algorithms used in WSNs. Many topology-based ad hoc routing algorithms have been adapted to work in wireless sensor networks. Traditionally, routing algorithms for Mobile Ad Hoc Networks (MANETs) and WSNs can be classified as either proactive, reactive or hybrid [21].

Proactive algorithms, also called global state algorithms, employ classical routing strategies such as distance-vector routing (e.g. Destination-Sequenced Distance Vector (DSDV) [22]) or link-state routing (e.g. Optimized Link State Routing (OLSR) [23], Topology Broadcast based on Reverse Path Forwarding (TBRPF) [24]). They maintain routing information about available paths in the network even if these paths are not currently used. Moreover, they keep their routing tables current by flooding the network on topology changes (e.g. Global State Routing (GSR) [25]). Reactive algorithms on the other hand were developed as a response to this observation (e.g. Dynamic Source Routing (DSR) [26], Ad Hoc on Demand Distance Vector (AODV) [27], Temporally-Ordered Routing Algorithm (TORA) [28, 29]).

Reactive routing protocols are on-demand route acquisition systems wherein a node sends a route request (RREQ) whenever it needs to send a message to a node for which a route does not already exist. Reactive routing protocols are generally more scalable, since they generate less network traffic, and are thus suitable for highly dynamic ad hoc networks [30]. However, maintaining routes only while in use leads to a delay for the first packet to be transmitted.

Hybrid algorithms, such as Zone Routing Protocol (ZRP) [31, 32] combine features of the two mentioned categories. They maintain a global view only for a certain number

of hops for each node.

Geographic or location based routing algorithms eliminate some limitations of topology based routing by using the position of nodes. Geographic routing algorithms can be divided mainly into two types: real location based and virtual location based. In the first type it is necessary that each node knows its own physical location. Commonly, it is assumed that each node determines its own position through the use of Global Positioning System (GPS) [33] or some other type of positioning service [18, 34, 35]. In the second type, a virtual location is allocated for each node. Earlier geographic routing algorithms proposed only simple greedy forwarding [36, 37]. Due to local minima (i.e. dead ends) greedy forwarding cannot guarantee packet delivery.

Therefore several recovery algorithms have been suggested. For example, in [38], the authors propose to forward to the node with least backward (negative) progress when reaching a dead end. However, this raises the problem of routing loops. Other researchers proposed to drop the packet when reaching a dead end. Based on planar graph traversal, several routing algorithms, like Greedy Face Greedy (GFG) [39] or Greedy Perimeter Stateless Routing (GPSR) [40], appear to guarantee packet delivery. A packet enters the recovery mode when reaching a dead-end. Planar graphs are graphs with no intersecting edges.

GSpring [41] tries to improve the performance of greedy forwarding. In a first phase, each node assigns itself an initial coordinate. Subsequently, nodes adjust their coordinates by simulating a system of springs and repulsion forces. Based on this, greedy routing is performed with only about 15 % overhead compared to using real addresses. Landmark based routing algorithms are presented in Virtual Coordinate assignment protocol (VCap) [42], Beacon Vector Routing (BVR) [43], Gradient Landmark-Based Distributed Routing (GLIDER) [44], and HopID [45]. Coordinates are assigned to nodes based on their hop count distances to some landmark nodes called beacons. Routing is done by trying greedy forwarding based on a distance function. When greedy forwarding fails to reach the destination, they resort to scoped flooding. The major drawback of this approach, in addition to the dead-end problem, is that it is not easy to find the optimal number of beacons. Thus they usually use a large number of landmarks.

The special case of unidirectional links has been investigated in [46]. The developed virtual coordinate assignment protocol (ABVCap_Uni) supports routing in sensor networks with unidirectional links. Based on available unique network IDs of all nodes,

the protocol tries to assign nodes with unidirectional links into rings and to treat a ring as an extended node. Routing is performed on virtual addresses assigned to real nodes and extended nodes. In Chapter 3 we present several directly related protocols in more detail.

2.3 Data Naming and Indexing

There are two ways to make use of the data produced by the sensors: either pushing the data to a base station in real-time or pulling the data on demand by means of queries. Especially for the later solution, it is required to associate names with the produced data items. The nature of data in WSNs is different from those on the Internet. The users on the Internet are normally concerned about the data itself rather than when and where the data was created. On the other side, in WSNs, these attributes can have the same importance as the data itself. The second difference is that the data (files) on the Internet are usually named by manually. On the other hand, sensor nodes have to name produced data automatically. This lead to a large space domain for names on the Internet in comparison to a very limited naming space in WSNs. [47] discusses research challenges related to naming and indexing sensor data. The authors present provenance as a base for naming and addressing sensor data. Due to infeasibility of indexing every sensor reading, an indexing based on tuple sets is suggested. Furthermore the provenance are likely to be application specific.

2.4 Data Storage

Storage approaches in wireless sensor networks can be divided mainly into three categories. The first class is called local storage, in which the data is stored on the same node that produced it without placing a reference to the data. The second approach is to push the data to a base station in real-time. The last approach is store the data on a node that is not necessarily the one that produced it by distributing the data across a number of nodes and implementing a routing scheme which allows one to efficiently lookup the node on which a specific data item is located. In the next sections, the principle and implications of each approach are discussed.

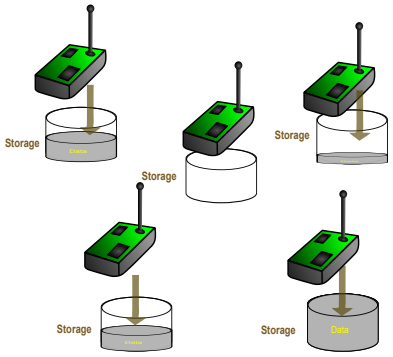

Figure 2.4: Local storage: the collected data is stored on the locally

2.4.1 Local Storage

Storing the data locally (Figure 2.4) is cost-effective in terms of communication needed for storing. Nevertheless to retrieve data from the network, the only chance is to ask as many participating nodes as necessary, whether or not they presently have the required data item or not. Although the complexity for storing is only $O(1)$, the complexity to retrieve data from the network is $O(N)$. The local storage approach has the advantage that there is no need for proactive efforts to maintain a routing table. Because of the limited storage space in the nodes, the storage capacity of nodes can get quickly exhausted. Hence, a special concern must be taken in the consideration when storing data locally on the nodes. Several approaches can be used to retrieve data from the network. An obvious approach is flooding the network with the query to find the answer. Flooding returns the answer very quickly, and is therefore highly tolerant of changing network dynamics, but it requires an excessive number of messages, which exhausts the node batteries and can congest the network, especially in dense networks.

A well-known example of the flooding technique used as a query algorithm in wireless sensor networks is directed diffusion [48, 49]. In directed diffusion, a sink announces its interest for a particular data item by flooding a query into the network containing attributes for the required event. The query is used to establish a gradient toward the sink. When the sink receives a response it can reinforce paths that have better quality. As a result of this high overhead several algorithms based on random or biased walk have been developed, including ACQUIRE [50], Rumor Routing [51], and biased Random Walk [52].

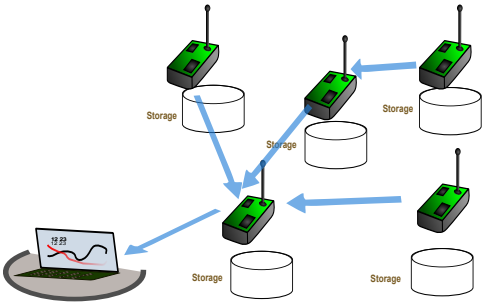

Figure 2.5: External storage: the collected data is transmitted to a base station

2.4.2 External Storage

The external storage approach (Figure 2.5) is analog to the traditional client-server approach. Each sensor reading should be transmitted to a base station in real time. In contrast to local storage, external storage does not impose communication cost to retrieve the data. Actually this storage strategy could be useful when all sensor readings are important, for example in habitat monitoring [14]. Typical systems employing external storage require efficient routing algorithms to ensure low communication cost for storing data. Hence the complexity for storing data is $O(\sqrt{N})$. Since all the gathered data are stored in an external server, there is no cost for data retrieval. However, the communication channel at the base station may encounter a large traffic load, consequently it represents a bottleneck in the whole system. Overall, the external storage approach is best for simple and small networks, since the cost for data retrieval is optimal and the amount of network load in the proximity of the base station is high. But the scalability is a vital property and this approach requires the availability of a path all the time from each node in the network to the storage element.

2.4.3 In-Network Storage

Both external and local storage exhibit bottlenecks that can affect the scalability and efficiency of a WSN. Indeed, external storage disqualifies itself with a linear complexity for communication at the base station. Local storage approaches avoid the management of references on other nodes and, therefore, they require a costly breadth-first search which leads to scalability problems in terms of communication overhead and energy consumption. A better solution for the lookup problem should avoid these drawbacks and should enable scalability by finding the golden path between the two approaches.

In this case the scalability is defined as follows: the search and storage complexity per node should not increase significantly even if the system grows by some orders of magnitude. Distributed Indexing, in the realm of P2P systems often implemented as a DHT, promises to be suitable method for this purpose. In Section 2.6 we will present P2P and hence DHTs in details. Distributed hash tables possess the following characteristics:

- By mapping nodes and data items into a common address space, routing to a node leads to the data items for which a node is responsible.
- Each node manages only a small number of references to other nodes
- By distributing the identifier of nodes and data items nearly equally throughout the system, the load for retrieving items should be balanced equally among all nodes.
- Because no node plays a distinct role within the system, the formation of hot spots or bottlenecks can be avoided.
- A distributed index provides a definitive answer about results. If a data item is stored in the system, the DHT guarantees that the data is found.

Like in the local storage approach, the collected data are stored on nodes inside the network. However, it is not necessarily on the node that gathered the data. Instead the data could be stored on any node in the network. This means that the sensed data are sent to a node(s) in the network so that all queries goes to this node(s). In a data-centric approach (Figure 2.6), sensed data names are associated with specific nodes that should host these data items, usually by using a mapping function. Thus queries are sent directly to the corresponding node. Geographic Hash Table (GHT) [53–55] is a typical in-network storage for WSNs. It hashes keys into geographic locations, so the data items are stored on the sensor node geographically nearest the hash of its key. Distributed Index for Features in Sensor Networks (DIFS) [56] and DIMENSIONS [57] are other examples of in-network storage schemes in WSNs. DIFS is another instance of GHT, it is built upon GHT and considers high-level events with multiple attributes. A multiple-rooted hierarchical index tree structure is constructed in DIFS to facilitate range query forwarding. [58] considers storage node placement in WSNs aiming to

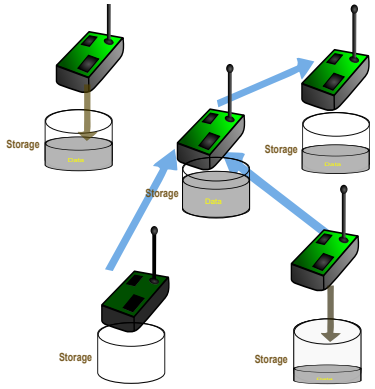

Figure 2.6: In-network storage: the collected data is stored on nodes associated with the data item within the network

minimize the total energy cost for gathering data to the storage nodes. The complexity for storing data as well as querying the data is $O(\sqrt{N})$.

There are two possibilities for storing data items in a DHT. In the first method, which is called direct storage, the data is copied upon insertion to the node responsible for it. The other possibility, called indirect storage, is to store references to the data. The inserting node only places a pointer to the data into the responsible node for this data. The data itself remains on this node. The first method incurs larger traffic in the insertion, but the query message will find its final destination faster than in the second method. Which method is adequate to WSNs depends on the size of the data. Direct storage is preferable for small data, while indirect storage is more convenient for large data sizes.

2.4.4 Analytical Comparison

To compare between the three storage schemes, we now derive analytical expressions for the energy costs of each approach. We did the analysis from a network layer point of view and concentrated on the average case. For simplicity we consider a wireless sensor network with n nodes deployed in a rectangle of area A. We consider a grid deployment in which each node can communicate only with its direct neighbors that are aligned either horizontally or vertically but not on the diagonal. This means that we have a connected network with a minimum node degree of 2 and a maximum of 4, while the average node degree depends on the size of the network and equals $4 - 4/\sqrt{n}$, where n is the number of nodes in the network. Figure 2.7 depicts a network of size 9. The

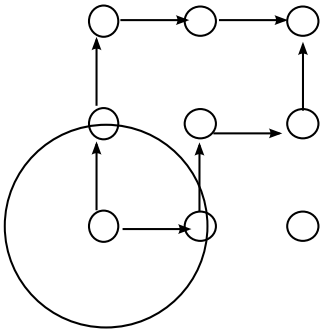

Figure 2.7: Example: network of 9 nodes deployed in a grid. Notice that the max routing path occurs when routing between end corners, traversing in either on the sides or diagonal leads to the same result ($4hops$)

maximum length of routing paths l_{max} in the grid occurs when routing between end corners. To find a mathematical solution we inspect simple networks. For a network with only one node, l_{max} is 0, for four nodes it is 2, for nine nodes it is 4, and so on. Assume $m = \sqrt{n}$ Thus, l_{max} can be calculated recursively as shown in Equation 2.2. The closed form solution is given in Equation 2.3.

$$l_{max}(1) = 0 \tag{2.1}$$

$$l_{max}(m) = l_{max}(m-1) + 2 \tag{2.2}$$

So, the closed form solution is:

$$l_{max} = 2\sqrt{n} - 2 \tag{2.3}$$

Equation 2.6 depicts the average length of routing paths to communicate with a node at the corner l_{avg-c} depending on the number of nodes n in the network. For simplified analysis, we still consider only nodes deployed in a grid network in a rectangular area.

$$l_{avg-c}(1) = 0 \tag{2.4}$$

$$l_{avg-c}(m) = l_{avg-c}(m-1) + 1 \tag{2.5}$$

So, the closed form solution is:

$$l_{avg-c} = \sqrt{n} - 1 \tag{2.6}$$

Following the same approach we can find the average length of a routing path between one random node and another within the network l_{avg}:

$$l_{avg}(1) = 0 \tag{2.7}$$

$$l_{avg}(m) = (m-1) - \frac{l_{avg}(m-1) \times (m-1)^2}{2m^2} \tag{2.8}$$

So, the closed form solution is:

$$\begin{aligned} l_{avg} &= (\sqrt{n}-1) - (\frac{\sqrt{n}}{3} + \frac{2}{3 \times \sqrt{n}} - 1) \\ &= \frac{2 \times \sqrt{n}}{3} - \frac{2}{3 \times \sqrt{n}} \end{aligned} \tag{2.9}$$

Although the big-O complexity for the communication overhead for sending data to a node at the border of the network and to a node the inside the network is $O(\sqrt{n})$, it is important to notice that average communication with a node outside the network consumes about 30% more than with nodes inside the network.

Before computing communication costs for the storage schemes, it is important to mention that sometimes it is more accurate to relate the path lengths to the area and the communication range of the nodes, rather than to the number of nodes. In fact sometimes it is misleading to use only the number of nodes in the network in the analysis of algorithms. For example if we have a network of size 100 in which all the nodes are in the communication range of each other (single-hop network), then the average as well as the maximum paths in the network are one hop. The maximum path length can be written as $\frac{2\times\sqrt{A}}{r}$ and the average is $\frac{\sqrt{A}}{r}$ where A is the area and r is the communication range. Figure 2.8(left) shows the shortest path between two nodes at the corners in two networks (maximum shortest path in the network). Notice that if there are no nodes at the diagonal in Figure 2.8(left), the packet will traverse either on the sides or on the diagonal, both will lead to the same result $l_{max} = 2\sqrt{49} - 2 = 12$. Now using the area we can calculate l_{max} as follows: the area $A = 900 \times 900$ and the communication range $r = 150$, hence $l_{max} = \frac{2\times\sqrt{900\times900}}{150} = 12$).

We can generalize this result to dense networks by allowing diagonal communication. Thus the maximum shortest path is the maximum distance between any two nodes in the network divided by the communication range of the sensor nodes. Thus it can be expressed as $\lceil \frac{\sqrt{2\times A}}{r} \rceil$. In Figure 2.8(right) the maximum shortest path is $\lceil \frac{\sqrt{2\times A}}{r} \rceil = (9)$.

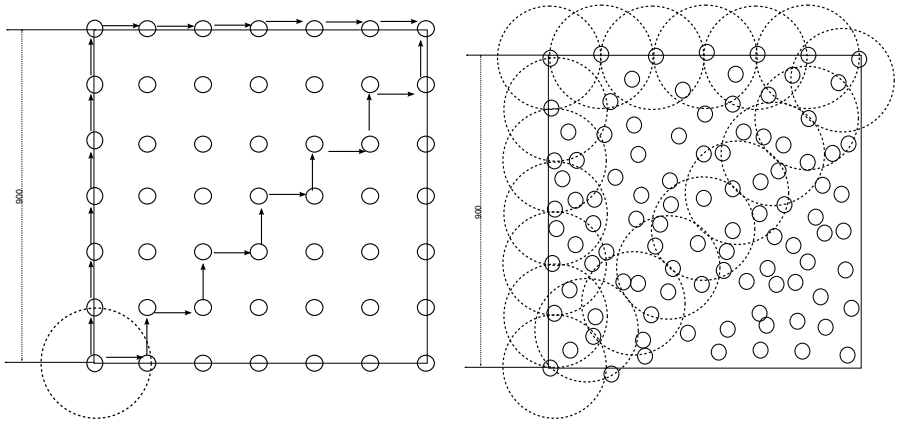

Figure 2.8: Shortest path in grid and random dense networks

For all storage schemes we assume a WSN with the following parameters: D is the total number of detected events (data items) in the network. Q are the total number of queries in the network. The base station is located at a corner. We also define the function C as the the average communication cost to store and retrieve data items in the network.

- External Storage:

  Because the external storage nodes are located outside the network (at a corner), the average cost to ship each data item to the external storage is $\sqrt{n}-1$. Because the data is already at the external storage node, there is no cost for queries.

  $$C_e = D \times (\sqrt{n}-1) + 0 \tag{2.10}$$

- Local Storage:

  Because the data will not be transported, there there is no communication cost for data storage. However, queries are flooded to all nodes in the network at average (and max) cost of (n). Answers are routed back to the source of the query at an average cost of $\sqrt{n}-1$.

  $$C_l = Q \times (n) + Q \times (\sqrt{n}-1) \tag{2.11}$$

- In-network Storage: There are two ways to store data, direct and indirect. Direct storage: here we transfer each data item to the node responsible for it. The average cost to transfer each data event to the node responsible for the data item

is $\frac{2\times\sqrt{n}}{3} - \frac{2}{3\times\sqrt{n}}$. The query source is located at the corner, therefore answers are routed back at an average cost of $\sqrt{n}-1$.

$$C_{id} = D \times ((\frac{2\times\sqrt{n}}{3} - \frac{2}{3\times\sqrt{n}})) + 2Q \times (\sqrt{n}-1) \tag{2.12}$$

Indirect storage: assume the number of reference updates is R_a

$$C_{ii} = (R_a + Q) \times ((\frac{2\times\sqrt{n}}{3} - \frac{2}{3\times\sqrt{n}})) + 2Q \times (\sqrt{n}-1) \tag{2.13}$$

Several lessons can be learned from the above calculations:

- Within a network with a large number of nodes, local storage incurs the highest cost.
- Local storage incurs the lowest cost for scenarios with a very low number of queries. For a large number of queries, external storage incurs the lowest cost. For a large number of nodes and intermediate number of queries, in-network storage incurs the lowest cost.
- Increasing the network density (i.e. increasing the number of nodes in the same area) will not increase the communication cost for external as well as for in-network storage schemes. The reason for this is that (as discussed above) the shortest path depends on the area and the communication range of nodes rather than the number of nodes in the network. However in case of local storage, the cost will increase linearly with the number of nodes, in addition to increasing the congestion in the network.
- In-network storage based on indirect storage incurs the lowest cost if the reference updates rate is very low.

Figure 2.9 compares the main characteristics of the presented approaches in terms of complexity, characteristics and requirements. According to their complexity in terms of communication overhead, external storage shows the lowest performance. But this approach suffers from two problems. First, the gateway to the external storage represents a bottleneck in the system, because it encounters huge traffic per time unit

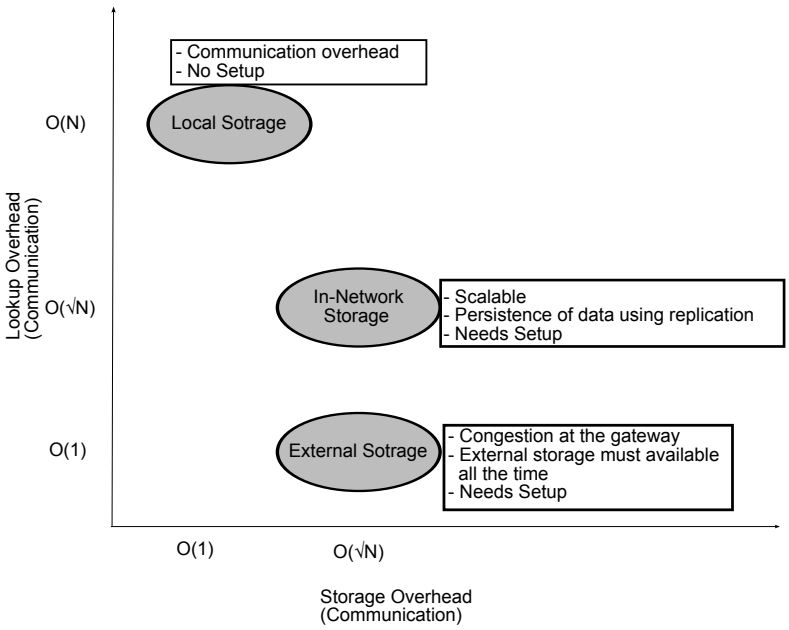

Figure 2.9: A comparison of the complexity of the three storage approaches in terms of lookup overhead (y-axis) and storage overhead(x-axis), special characteristics of each approach are named

(when a large number of nodes send data at the same time). The second drawback of this approach is that it requires the availability of a path to the external storage all the time. While local storage offers the lowest storage overhead, the lookup process requires a huge communication overhead, and search results are not guaranteed, since the lifetime of request messages is restricted to a limited number of hops. In-network storage which spreads data and routing information across multiple nodes represents a scalable approach since there are no hotspots in the network and it requires intermediate communication overhead. Indeed, DHTs provide an efficient approach for the lookup problem in distributed systems.

2.5 Data Processing

Usually, data processing will result in data reduction, which will reduce the usage of power sources, bandwidth, and storage in the network. Moreover, events in WSNs are detected by a group of spatially distributed sensors which collaborate to make decisions [19]. One approach is to compress sensor data before transmission to reduce

energy as some loss is acceptable without affecting the results of the application. Data collected by sensors that are in close proximity exhibit spatial correlation. If samples collected over time are from the same source(s), the data also show temporal correlation. Like data storage schemes, there are three places to process the data. Simple and correlated data can be processed on the node (local processing). Some statistical values (i.e. average, maximum) or event detection can be obtained by cooperation between nodes (in-network processing). Complicated data should be processed on high-end machines (external processing). The first two schemes (local and in-network processing) are usually used for data reduction. External processing is mainly used for visualization and data mining purposes.

2.5.1 Local Processing

The main goal of data processing on the node is to reduce the amount of data that has to be stored on the node and transmitted through the network to the sink. Therefore, the energy saving is proportional to the number of hops the data passes through the network. Data processing takes several shapes, for example data processing can be used to remove noise from the measured values, infer information from the data (e.g. detecting an animal from the picture instead of sending the picture) or compressing the data. [7, 8] are WSNs oriented data compression algorithms. [8] proposed compression algorithms for historical information in WSNs. The algorithms have been used to reduce the data needed to be disseminated. The techniques have been built on the observation that the values of the collected measurements exhibit similar patterns over time, or that different measurements are naturally correlated, as is the case between pressure and humidity in weather monitoring applications.

2.5.2 In-Network Processing

Despite the fact that sensor nodes are very resource-constrained devices, several nodes working together can produce considerable computing power. Hence the nature of WSNs induces the nodes to work cooperatively, and thus several distributed algorithms are used in WSNs. DISCUS [9] presents a distributed compression algorithm designed for WSNs. The Authors propose a way of removing the redundancy of data in a dense WSN in a completely distributed manner. Their collaborative framework enables highly effective and efficient compression across a sensor network without the need

to establish inter-node communication, using well-studied and fast error-correcting coding algorithms. In some application scenarios, the information of interest is not the data at an individual sensor node, but the aggregate statistics (aggregates) amid a group of sensor nodes [59, 60]. Tiny Aggregation [61] and Geographic Gossip [62] present approaches to compute aggregates like average, max/min, sum. [63] examines key applicative query-based approaches that utilize in-network processing for query resolution.

2.5.3 External Processing

For intensive data processing, it will be more efficient to process that data externally on a high-end machine. Moreover processing the data on the nodes usually needs much more time than on a high capability machine, therefore for real time applications it is preferable to process the data externally. It is obvious that data analysis and visualization make the data more accessible for human users.

2.6 Peer-To-Peer Technology

In traditional client-server systems the server is the only provider of service or content, e.g. a web server or a calender server. The peers (clients) in this context only request a content or service from the server. Thus generally the clients are lower performance systems and the server is a high performance system. In contrast, in peer-to-peer systems all resources, i.e. the shared content and services, are provided by peers. Some central facility may still exist, e.g. to locate a given content [64]. [65] gives a basic definition of what constitutes a peer-to-peer system: a self-organizing system of equal, autonomous entities (the peers), which aims for the shared usage of distributed resources in a networked environment avoiding central resources. The appearance of new P2Ps was in the late 1990s, the main usage of P2Ps being file sharing on the Internet [64] between users. In opposition to the traditional client-server paradigm, P2Ps follows a decentralized, self-organizing approach. Therefore P2P has drawn much attention in the last decade in both research and commercial domains. Usually P2P systems are built on the application layer as overlays and leverage the already existing routing services in the Internet. Therefore, traditional implementation of P2P systems requires the nodes to have a physical, a link, and a network layer.

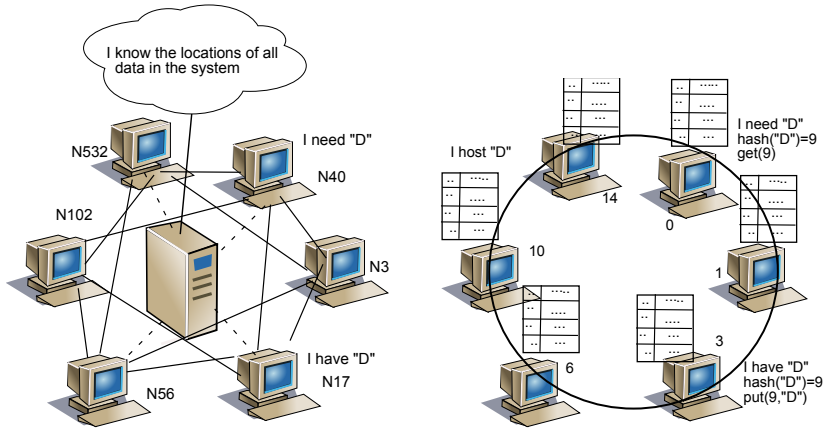

Figure 2.10: Peer-to-peer structures: (left) unstructured peer-to-peer network, (right) structured peer-to-peer network

2.6.1 Unstructured Peer-To-Peer

The term "Unstructured Peer-To-Peer" refers to peer-to-peer schemes at which the content stored on a given node and its name (i.e. IP address) are unrelated and do not follow any specific structure. Peer-to-peer networking started with the first generation centralized concept. In this case some central servers are still available. However, contrary to the client-server approach this server only stores information about peers where some content is available, thus greatly reduces the load of that server. The address of that server must be known to the peers in advance. Figure 2.10(left) depicts a typical first generation P2P network. If a node (i.e N40) needs some content it must first contact the server, which responds with the location (i.e N17) of the required content. After that the data transfer is done between the node holding the content (N17) and the requesting node (N40) without any interference from the server. A well known P2P application is Napster, its approach having later been called first-generation P2P. Napster was used to share music files between home users. It was not fully decentralized, instead it relied on a server to look up the needed file location. Therefore users have to contact a server to find the location of the file before being able to download files, which are stored on the peers. Because of the illegal content of most of the shared files in the Napster Network and because Napster relied on a central server, it was easily targeted by the authorities.

As a replacement for the the centralized scheme, decentrally organized schemes such as Gnutella 0.4 and Freenet became widely used. They used flooding to discover

the location of the requested files instead of a central server (except possibly for some bootstrap server to ease joining such network). An important drawback of these schemes that they generate a potentially huge amount of signaling traffic by flooding the requests. To avoid that, schemes like Gnutella 0.6 or JXTA introduced hierarchy by defining super peers, which store information about the content available at the connected peers. Thus super peers are often able to answer incoming requests by immediately providing the information about the respective peers, so that on average less hops are required in the search process, thus reducing the signaling traffic.

2.6.2 Structured Peer-To-Peer

New approaches have been proposed which establish a link between the stored content and the identifier of a node (e.g. an IP Address). Such networks are termed "Structured Peer-To-Peer" networks. The link between a content identifier and the node identifier is usually based on DHTs. The advantages of decentralized and self-organizing systems inspire researchers to focus on approaches for distributed, content addressable data storage. DHTs were designed to provide such distributed indexing as well as scalability, reliability and fault tolerance. DHTs manage the data by distributing it across the nodes in the network and implementing a routing scheme which allows one to efficiently look up the node on which a specific data item is located. The main idea of DHTs is simple: data items are associated with numbers and each node in the network is responsible for a range of these numbers. In contrast to flooding based searches in unstructured systems, each node in a DHT stores a partial view of the whole distributed system which effectively distributes the routing information. Based on this information, the routing procedure typically traverses several nodes, getting closer to the destination with each hop until the destination node is reached.

Figure 2.10(right) depicts a typical structured P2P system. Each node maintains a routing table which contains useful information (i.e IP Address, port number, etc.) to directly reach a few nodes in the network, through which all nodes can reach each other in the network. If a node has a content (e.g. node 3), is will insert it on the node responsible to hold this content (e.g. node 10). Now if a node needs this content, it can use the same mapping scheme (usually a hash function) to find the proposed node holding this content. Usually, DHTs are built on the application layer and rely on an underlying routing protocol that provides connectivity between the nodes. DHTs

introduce a new address space in which data items are mapped. Address spaces typically consist of large integer values (e.g. the range from 0 to $2^{160}-1$). DHT approaches differ mainly in how they internally manage and partition their address space. In most cases these schemes lend themselves to geometric interpretation of the address space. A data item in a DHT is associated with a unique number from the address space. This value can be chosen freely by the application, but it is often derived from the data itself via a collision-resistant hash function such as SHA-1 [66]. To store or access data in a distributed hash table, a node first needs to join the network. The arrival of new nodes leads to changes in the DHT infrastructure, to which the routing information and distribution of the data needs to be adapted. When a node fails or leaves the system, the DHT needs to detect and adapt to this situation.

Chord [67–69] is one of the best known DHT systems. Chord's elegance comes from its simplicity: It uses an $l-bit$ identifier forming a one-dimensional identifier circle modulo 2^l in the range $[0, 2^l-1]$. Using a hash function, data items and nodes are associated with an identifier. The identifier of the data is called key, while ID refers to the node identifier. The node whose ID is greater than or equal to the key hosts the data item. This node is called the successor of that key. Each node maintains a routing table, called $fingertable$, of size l and points to nodes that succeed the node in the identifier circle. Each entry in the table consists of a node ID, an IP address and a port. Other similar approaches like [70, 71] appeared almost at the same time and rely on the same fundamental idea, however their methods for organizing the identifier space and their routing mechanisms differ. Symphony [72] is a modified version of Chord which tries to reduce the per-node state and the network traffic when the overlay topology changes.

2.7 Summary

In this chapter, we have investigated the data management problem in WSNs. We have shown that because of the special characteristics of WSNs, efficient and scalable data management schemes are highly required. DHTs provide an efficient layer of abstraction for managing data in WSNs. Applying DHTs in WSNs faces fundamental challenges related to separation between physical and logical networks which need to be solved. We have shown in this chapter also that data lookup, data routing, and data processing are complementary and not contradicting methods aiming to optimal usage

of node's resources, which eventually leads to better performance of the whole system.

Chapter 3

Related Work

In this chapter, we provide an overview of the related work that uses Distributed Hash Tables (DHTs) to manage data in WSNs. In Section 3.1, we take Chord [67] [68] [69] as an example and show problems and drawbacks of implementing DHTs as an overlay in WSNs. We present DYMO as a candidate routing protocol that can be used with Chord to offer underlay routing services. After that, we describe Virtual Ring Routing (VRR), which is a routing protocol that provides DHTs services in Section 3.3. Section 3.4 covers Geographic Hash Tables (GHTs) which maps data items to geographical locations and leverages GPSR [40] for routing. Finally, we describe Geographic Routing Without Location Information (GRWLI) and Hop ID in Sections 3.5 and 3.6, respectively. Both algorithms rely on different types of virtual coordinate systems: Geographic Routing Without Location Information (GRWLI) is a geographic-like routing protocol using Cartesian virtual coordinates, while Hop ID employs a set of landmark nodes to construct other nodes' coordinates.

3.1 Chord

Chord is an efficient distributed lookup system based on consistent hashing [73]. It provides a unique mapping between an identifier space and a set of nodes. Chord uses a $l - bit$ identifier that forms a one-dimensional identifier ring modulo 2^l in the range $[0, 2^l - 1]$. Using a hash function, data items and nodes are associated with an identifier. Thereby the identifier of the data is called key and a node's identifier is called ID. The node whose ID is greater than or equal to the calculated key hosts the data item. This node is called the successor of that key. Each node maintains a routing

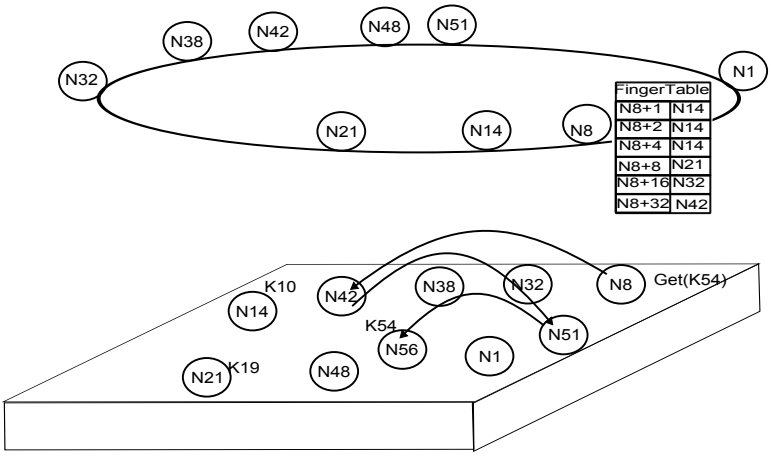

Figure 3.1: Chord example with 10 nodes

table called $fingertable$ of size l. The entries of the finger table point to nodes that succeed the node in the identifier ring. The i^{th} entry in the table at node n contains the successor node of $n + 2^i mod 2^l$, where $0 \leq i < l$. The chord routing algorithm exploits the information stored in the finger table of each node. A node forwards queries for a key k to the closest predecessor of k on the identifier according to its finger table. When a query reaches a node n such that k lies between n and the successor of n on the ring, then the key must be hosted by this successor. Consequently, the successor is communicated as a result of the query back to its originator.

Figure 3.1 illustrates ten nodes forming a chord ring. In this example, a 6-bit identifier is used. The successor of key K10 is the node with ID N14 where K10 is thus hosted. Each node maintains a routing table (*Finger Table*) with size 6. For example, if node N8 requests K54, it looks in its finger table and sends the query to the closest predecessor of the key, which is in this case node N42. This process continues until the query reaches node N56 which hosts key K54. N56 will then send K54 back to N8.

If a new node node wants to join the network, it must first get an identifier n, that e.g. can be randomly chosen. In addition, the new node should know another node o which already participates in the network. By querying o for n's own identifier, n retrieves its successor s. It notifies its successor s of its presence leading to an update of the predecessor pointer of s to n. Then node n builds its finger table by iteratively querying o for the successors of $n + 2^1$, $n + 2^2$, $n + 2^3$, etc. At this stage, n has a valid successor pointer and finger table. However, the join function does not make the

rest of nodes aware of n. In order to validate and update the successor pointers in a dynamic network with node joins and leaves, Chord introduces a *stabilization* protocol. It requires an additional pointer to a node's predecessor and is performed periodically on every node. The $stabilize()$ function at a node k requests its successor to return its predecessor p. If p equals k, then k and p are each other's predecessor and successor, respectively. The fact that p lies between k and its successor indicates that p recently joined the network as k's successor. Thus, node k updates its successor pointer to p and notifies p of being its predecessor.

3.2 Dynamic MANET on Demand

Chord is implemented as an overlay, hence one hop in Chord usually implies several hops in the real network. So Chord requires an underlay routing protocol that offers routing services in the underlayer. We chose Dynamic MANET on Demand (DYMO) as underlay routing protocol. DYMO is the most recent standard ad-hoc routing protocol and was developed by the IETF MANET working group. It is currently defined in an IETF Internet-Draft [74] in its twelfth revision and is still work in progress. DYMO is a successor of the AODV routing protocol [75] and is the current engineering focus for reactive routing in the IETF MANET working group. DYMO operates similarly to AODV and does not add extra features to the AODV protocol. It simplifies the protocol while retaining the basic mode of operation. DYMO is not the first attempt to make an enhanced version of AODV. AODV with Path Accumulation (AODV-PA) [76] extends AODV with DSR source route path accumulation feature. AODVjr, AODV simplified [77], removes all but the essential elements of AODV. Using AODV as a basis, DYMO combines the ideas originated in AODV-PA and AODVjr. It borrows path accumulation from AODV-PA (and DSR). Like AODVjr, DYMO removes features that may be regarded as extensions to the core functionality(i.e DYMO removes *hello* messages).

Routes in DYMO are discovered on-demand when a node needs to send a packet to a destination currently not in its routing table. A Route Request (RREQ) message is flooded in the network using broadcast. If the packet reaches its destination, a Route Reply (RREP) message is sent back containing the discovered and accumulated path. Each node must maintain its own sequence number and the sequence number is incremented each time the node sends a route request message. This allows other

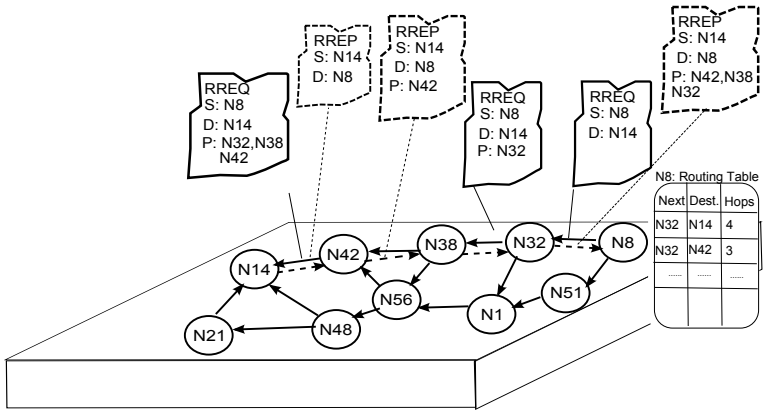

Figure 3.2: DYMO route discovery: node N8 wants to communicate with node N14

nodes to determine the order of discovery messages to avoid stale routing information, to detect duplicate messages, and to ensure loop freedom. An illustration of the route discovery process is shown in Figure 3.2. In the figure, Node N8 wants to communicate with node N14 (it is the successor of N8 in chord). Thus N8 is the source node (S) and node N14 is the destination (D). In the RREQ message, node N8 includes its own address, destination address and its sequence number, which is incremented before it is added to the RREQ. The message is flooded using broadcast in a controlled manner throughout the network, whereby a node only forwards an RREQ if it has not done so before. The sequence number is used to detect this. Each node forwarding an RREQ may append its own information. When node N32 receives the RREQ, it installs a route to node N8. After node N32 has forwarded the RREQ, it has added its own address to the RREQ, which means it now contains three addresses. Identical processing is performed at node N38, and additionally it installs a route to node N8 with a hop count of 2 and node N32 as the next hop node. When node N14 receives the RREQ, it contains five addresses and has traveled four hops. Node N14 processes the RREQ and installs routes using the accumulated information. As it is the target of the RREQ, it furthermore creates a RREP as a response. The RREP is sent back along the reverse route. Similar to the RREQ dissemination, every node forwarding the RREP adds its own address to the RREP and installs routes to node N14.

To maintain paths, nodes continuously monitor the active links and update the *Valid Timeout* field in its routing table when receiving and sending data packets. If a

node receives a data packet for a destination it does not have a valid route for, it must respond with a Route Error (RERR) message. When creating the RERR message, the node makes a list containing the address and sequence number of the unreachable node. In addition, the node adds all entries in the routing table that depend on the unreachable destination as next hop entry. The purpose for this action is to notify about additional routes that are no longer available. The node sends the list in a RERR message that is broadcasted. When a node receives a RERR, it compares the list of nodes contained in the message to the corresponding entries in its routing table. If a routing table entry for a node from the RERR message exists, it is invalidated if the next hop node is the node the RERR was received from, and the sequence number of the entry is greater than or equal to the sequence number found in the RERR. If a routing table entry is not invalidated, the corresponding entry in the list of unreachable nodes in the RERR must be removed. If entries still remain in the list, the node further propagates the RERR.

The main drawback of implementing DHTs as overlay in WSNs is the separation between the logical(overlay) and physical(underlay) networks. This separation poses extra overhead and complexity on the system, as each layer has its own routing schemes. To analyze the cost of the overlay, consider a network of size n nodes. Routing in the overlay requires $O(log(n))$ hops, while the underlay requires $O(\sqrt{n})$. Consequently, routing requires $O(log(n)\sqrt{n})$ hops. Although routing paths of length $O(\sqrt{n}log(n))$ might be not very bad, updating the routing tables still poses a significant overhead. In chord, the size of the finger table (routing table) is $log(n)$. Each entry in this table requires $O(\sqrt{n}log(n))$ hops, hence updating the routing table requires $O(log(n)^2\sqrt{n})$ hops. In addition to the overlay overhead, there is a separate underlay overhead. This overhead depends on the employed routing protocol, and updating its routing table also causes additional overhead. In addition to the communication overhead, implementations usually require extra memory for routing tables and programs.

3.3 Virtual Ring Routing

VRR [78] is a routing protocol inspired by overlay DHTs. Aside routing, it provides traditional DHTs functionality. VRR uses a unique key to identify nodes. This key is a location-independent integer. Like Chord, VRR organizes its nodes in a virtual ring in

the order of increasing identifiers. For routing purposes, each node maintains a set of virtual neighbors ($vset$) of cardinality r that are nearest to the node identifier in the virtual ring. Each node also maintains a physical neighbor set ($pset$) with the identifiers of nodes that it can communicate with directly. A routing table entry identifies the next hop towards a virtual neighbor. This information is maintained pro-actively, i.e. it is maintained even when there is no traffic along the path. The forwarding algorithm used by VRR is quite simple: VRR picks the node with the identifier closest to the destination from the routing table and forwards the message towards that node. The problem of these protocols is that adjacent nodes in the ring can be far away in the real network. As a result, forwarding to the nearest node can result in a very long path. Moreover, the scalability is a problem, as the protocol needs to maintain routing tables of increasing size, with increasing network sizes.

Figure 3.3 shows the mapping between the virtual ring and the physical network topology, and the routing table for the node with identifier N8. The first four entries in the table are the $vset$-paths to N8's virtual neighbors. The 5^{th} and 6^{th} entries are $vset$-paths that happen to be routed through N8, and the last three paths are N8's physical neighbors. The routing table also shows that the $vset$-path identifiers are not necessarily distinct. The $vset$-path identifier is assigned by endpoint A, which is the node that initiates the path setup, such that each $vset$-path is uniquely identified by the pair $< path-id, endpointA >$. $Vset$-path identifiers can be small: each node originates at most one $vset$-path to each of its r virtual ring neighbors and nodes can reuse the identifiers of paths that were torn down after a probation period to ensure that there are no routing table entries with those identifiers.

The joining process in VRR is quite similar to Chord. The joining node starts by looking for physical neighbors that are already active in the network and, therefore, can be used as proxies to route messages to others. It finds a proxy by sending and listening to $hello$ messages that VRR nodes periodically broadcast to physical neighbors. These messages are also used to initialize the $pset$ of the joining node. After finding a proxy, the joining node sends a $setup\_req$ message to its own identifier x through the proxy. This message is routed using the forwarding algorithm to the node whose identifier y is closest to x. Node y is one of the immediate virtual neighbors of the joining node in the virtual ring. It knows the identities of the other virtual neighbors of x. Node y replies with a $setup$ message that is routed back to the joining node through

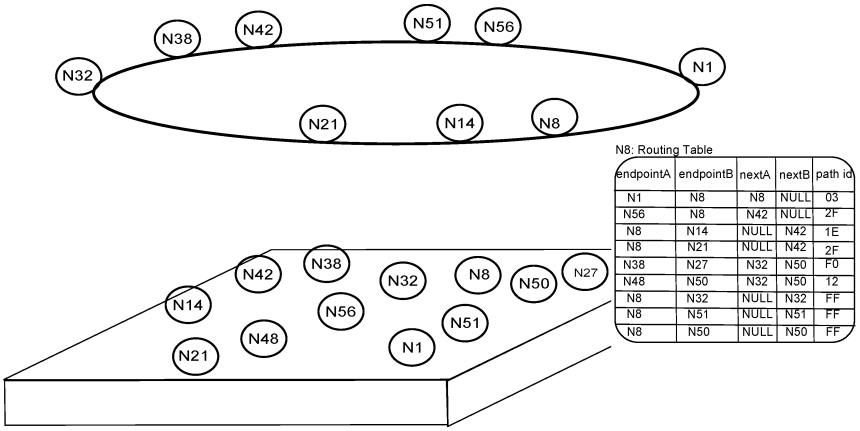

Figure 3.3: Relationship between the virtual and physical node places in VRR

the proxy, and it also adds x to its $vset$. This message sets up the $vset$-path between node y and the joining node by updating the routing tables of the nodes it visits. The joining node adds y to its $vset$ when it receives the message. The $setup$ message also includes y in its $vset$. The joining node uses the received $vset$ to initialize its own $vset$. It sends $setup\_req$ messages to the identifiers of its other virtual neighbors. The joining node adds these neighbors to its $vset$ when it receives setup messages from them. This completes all routing state initialization and the node becomes active. It is clear here that failure detection in $vset$ is not an easy task. To avoid the overhead of end-to-end probes or end-to-end heartbeats, the authors proposed a technique called *symmetric failure detection* to guarantee that if node x marks a neighbor y faulty, y will also mark x faulty. To do this, a $hello$ message containing two hops information is used. When a node x marks a node y as failed, it initiates the $teardown$ of any $vset$-paths in its routing table that have y as a next hop. It does this by calling $TeardownPath(p, null)$ for each identifier p of a failed $vset$-path. Additionally, x removes any one- and two-hop paths through y from its routing table.

VRR reduces the routing tables from two in Chord to one. Nevertheless, considering the fact that successor and predecessor of a node lie far away, it imposes not only an extra stretch of the routing path length, but also makes the joining process complicated, especially when several nodes concurrently join the network. Another drawback of VRR is the update of each node's $vset$ in the routing table. In the optimal case, if we do not take the stretch over the shortest path into consideration, updating each node in the $vset$ requires $O(\sqrt{n})$ hops.

3.4 Geographic Hash Tables

GHT [53–55] hashes keys into geographic locations, so only data items are stored on the sensor node that are geographically nearest to the hash of its key. They replicate the stored data locally to ensure persistence when nodes fail. Like ordinary DHTs, GHT is built as overlays and relies on underlay routing. In fact, it uses the Greedy Perimeter Stateless Routing (GPSR) [40] algorithm for low-level routing. GPSR uses the physical location of nodes for routing purposes. Thus, it is assumed that all nodes in the network know their location by using localization methodologies like GPS [79].

The main advantage of geographic routing algorithms is their optimal scalability, the reason of this, is the reliance only on local information for routing. GPSR uses greedy forwarding, based on local information about the real location of the physical neighbors, to forward packets to nodes that are always progressively closer to the destination. The sender includes the coordinates of the final destination while sending the packet. Since the greedy forwarding routing mode fails in case of voids (even in static networks), GPSR uses a second mode for routing, called perimeter mode. The perimeter mode uses a planar subgraph without crossing edges, wherein a packet consecutively traverses closer faces of a planar subgraph of the network connectivity graph until reaching a node closer to the destination. Here greedy forwarding resumes. Figure 3.4 illustrates how routing in GPSR works. Node N8 wants to communicate with node N14. The routing table of node N8 contains the addresses and locations of its single-hop radio neighbors. The position of N14 is known to node N8, consequently it sends the message to the node whose position is closer to the position of the destination, in this case node N32. Node N32 in turn sends the message to node N56. Until now, greedy forwarding succeeded to find the next best node. However, because the distance between nodes 56 and N14 (D1) is shorter than the distance between N38 and N14 (D2), as well as the distance between N48 and N1 (D3), there will be no progress toward the destination when sending to either N38 or N48. Thus, greedy forwarding fails at node N56. Now GPSR changes to the other mode (perimeter mode), in which, using the *right hand rule*, it traverses the perimeter nodes (N56,N48,N21,N14,N42,N38) to find a node closer to the destination.

GHT uses GPSR's perimeter mode to find *home nodes* for a packet, which is the node geographically nearest to the destination coordinates of the packet. In GHT, the packet

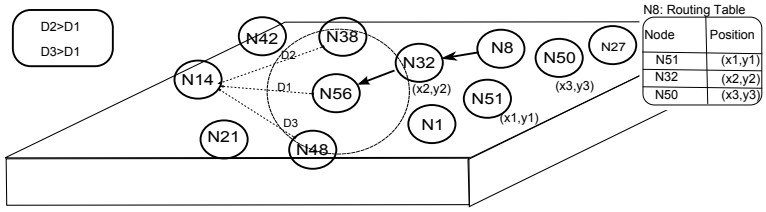

Figure 3.4: Routing using geographic positions

enters perimeter mode at the home node, as no neighbor of the home node can be closer to the destination. The packet then traverses the entire perimeter (called *home perimeter*) that encloses the destination, before returning to the home node [40].

The authors of GHT propose the Perimeter Refresh Protocol (PRP) to accomplish replication of key-value pairs and their consistent placement at appropriate home nodes when the network topology changes. PRP stores a copy of a key-value pair at each node on the *home perimeter*. A node becomes a home node for a particular key when the packet arrives after completing its tour of the home perimeter. Periodically, the home node for a key generates a refresh packet addressed to the hashed location of that key. Thus, the refresh packet will take a tour of the current home perimeter for that key. Figure 3.5(a) shows home perimeter nodes enclosing a key. The home perimeter nodes hold replicas of the data associated with the key and the home node is the node geographically closest to the key. GHT also employs a scheme called Structured Replication in DCS (SR-DCS) to achieve load-balancing in the network. SR-DCS uses a hierarchical decomposition of the key space and associates each event-type e with a hierarchy depth d. It hashes each event-type to a root location. For a hierarchy depth d, it then computes $4^d - 1$ images of root. When an event occurs, it is stored at the closest image node. Queries are routed to all image nodes, starting at the root and continuing through the hierarchy. Figure 3.5(b) shows a $d = 2$ decomposition and mirror images of the root (3,3) at every level.

To use GHT, four important things are assumed. Firstly, each node should know its physical location. The second important issue is that the deployment area should be known in advance in order to hash the keys to this area. Otherwise all packets hashed outside the area will be stored at the edge nodes after a long walk through the entire external perimeter before arriving at the home node. Thirdly, the correctness of the common (local) planarization algorithms, and hence the correctness of perimeter mode

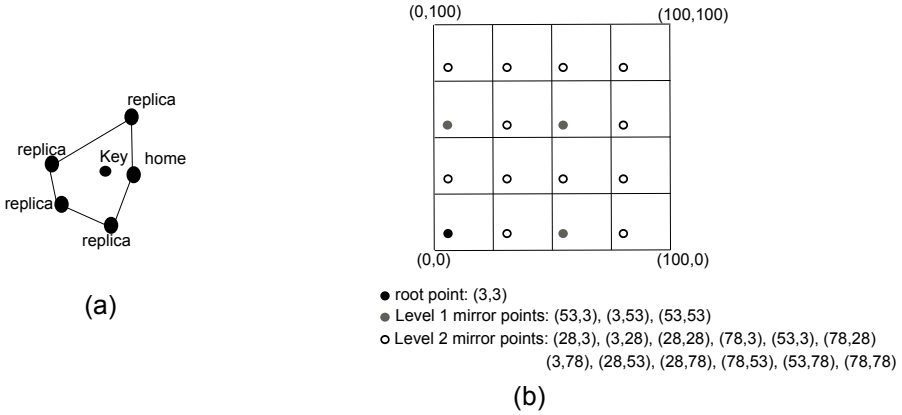

Figure 3.5: GHT: (a) Perimeter Refresh Protocol (PRP): the node geographically nearest to the key location is the home node, the nodes at the perimeter keep copies of the data associated with the key; (b) example of Structured Replication (SR): 16 mirror images using a 2-level decomposition

routing used intensively in GHT, relies on a unit disk graph assumption under which a node hears all transmissions from nodes within its fixed radio range and never hears transmissions from nodes outside this range [43]. Many studies have shown that this assumption is grossly violated by real radios [20, 80–82]. The final assumption is that nodes are deployed uniformly. Some WSNs may fit these assumptions, nevertheless many others may not. The replication algorithm (*PRP*) in GHT will always send refresh messages periodically, even if there is no any change in the network topology, which consequently wastes energy and bandwidth resources.

3.5 Geographic Routing Without Location Information

Motivated by the ideal scaling properties of schemes like GPSR, several recent proposals attempt to use geographic routing ideas without requiring geographic coordinates. GRWLI [83] creates synthetic coordinates through an iterative relaxation algorithm that embeds nodes in Cartesian space. Unlike GPSR, this routing protocol does not use real coordinates, which are costly, not available in many situations, and susceptible to localization errors. Instead, it constitutes an n-dimensional virtual coordinate system. The construction of the coordinates is based on finding the perimeter nodes and projecting them onto an imaginary circle if not pre-defined. This is done by designating two nodes as beacon nodes. Next, the nodes determine if they are perimeter nodes by using a heuristic based on their hop count from the beacons. After that, a relaxation

algorithm is used to find the virtual location of the nodes in the network. One of the reasons to project the perimeter nodes onto a circle is to have a well-defined area to implement the DHT.

The drawback of having many dimensions resulting from a large n is that the process of forming virtual coordinates requires a long time to converge [84]. Subsequently, it consumes more power for communication. The initialization technique for this scheme also requires roughly $O(\sqrt{n})$ nodes to flood the network, and each of these flooding nodes needs to store the entire $O(\sqrt{n} \times \sqrt{n})$ matrix of distances (in hops). So $O(n)$ state is kept at roughly $O(\sqrt{n})$ nodes, and this is an impractical burden in large networks. Again, the problem of possible dead ends exists here, so the greedy forwarding algorithm cannot guarantee reaching the correct destination.

3.6 Hop ID

In the Hop ID routing scheme [45], each node maintains a hop ID, which is a multidimensional coordinate based on the distance to some landmark nodes. Landmarks can be randomly selected in the network. However, to obtain better performance and reduce the effect of dead ends, the authors present several methods for landmark selection. To construct and maintain the hop ID system, three basic steps should be followed. First, a voluntary node floods the entire network to build a shortest path tree rooted at this node. Then landmarks are selected. Finally, each node adjusts its hop ID periodically and broadcasts its new hop ID using a hello message. To deal with dead ends when greedy forwarding fails, a landmark guided detour is designed and is applied with an expanding ring flooding search algorithm to route out of dead ends. Figure 3.6 illustrates the idea of the Hop ID system. Following a predefined order, the hop distance of a node to all the landmarks is combined into a vector, i.e., the node's hop ID. For example, G's hop ID is 432 in Figure 3.6, representing that G is four hops away from L1, three hop away from L2, and two hops away from L3. For greedy forwarding, the authors have used a the distance function $D_p = \sqrt[p]{\sum_{k=1}^{m} \mid H_k^{(1)} - H_k^{(2)} \mid^p}$, where m is the number of landmarks, the hop ID of node N1 is the vector H^1 and the hop ID of node N2 is the vector H^2. In Figure 3.6 for destination node C, node A is a dead end. Another source of dead ends is the existence of two or more nodes with same hop ID (i.e node D and E). In addition to the problems of dead ends and selecting the optimal number and

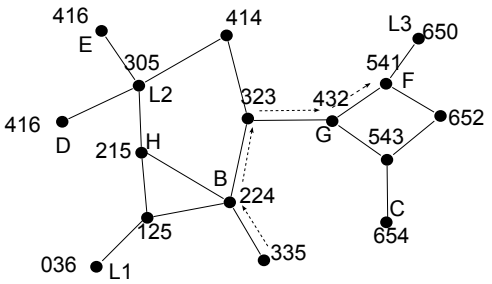

Figure 3.6: Example of Hop ID: 16 nodes including 3 landmarks (L1, L2, L3) constructing a 3-D Hop ID network, a node N's Hop Id xyz means N is x,y,z hops away from landmarks L1, L2 and L3 respectively [45]

location of landmarks, joining nodes as well as leaving nodes can affect large number of nodes in the network and requires a large number of nodes to update their hop ID.

3.7 Summary

In this chapter we have given a detailed overview of algorithms used in related work. We have shown the characteristics of each approach and the problems they face. Employing a traditional DHTs, like Chord, as overlay and replaying it on another protocol for underlayer routing services incurs avoidable extra overhead which makes the scalability of such an implementation is very bad. VRR reduces this overhead by combining both routing and data lookup, however updating the routing table requires multi-hop data transmission, and the traversal path can be much longer than the shortest path. Routing based on a physical location, like GPSR, has optimal scalability. GHT, which uses GPSR for routing, tries to combine data management with routing. However, it has some additional drawbacks in addition to the inherent drawbacks of GPSR: GHT always generates data refreshing messages, even if the topology does not change. GRWLI uses a virtual location to replace the physical location used in GPSR. The problem of dead-ends still exist in GRWLI. Additionally constructing the position requires considerable communication overhead. Constructing virtual positions in the Hop ID algorithm is easier than GRWLI, nevertheless it requires a number of landmarks. It is not trivial to determine the optimal number of landmarks. Additionally, the position itself is a vector that contains the hop counts to the landmarks. In GRWLI and Hop ID, the virtual positions of the nodes are not unique, which consequently complicates routing and mapping the data to nodes.

Chapter 4

Virtual Cord Protocol

In this chapter we describe the design of the Virtual Cord Protocol (VCP). VCP offers efficient routing as well as traditional DHT services for WSNs. The basic operations performed on VCP are insertion and lookup of (key, value) pairs.

4.1 Overview

In our design VCP offers underlay routing between sensor nodes in a WSNs. Each node maintains a small amount of routing information that is independent from the number of nodes in the network. Providing this hash-table-like interface then requires every node to support a single operation: given an input key, a node must be able to route messages to the node responsible for that key. As such, our design primarily addresses the issues related to supporting this data-centric routing operation in a manner that is completely distributed (requiring no form of centralized control, coordination or configuration), that is scalable (nodes maintain routing tables that are independent from the number of nodes in the network), that is simple and easy to construct, and that is robust to nodes failure.

The basic Idea of VCP is shown in Figure 4.1. Assume a cord connects all nodes in the network. The cord starts at a given value (say 0) and ends at another value (say 1). Each intermediate node has a unique value and communicates at least with the node whose value in the cord is immediately smaller than the node's own value (termed the predecessor) as well as the node whose value is immediately larger than the node's own value (termed the successor). In the example, node 0.91 generates a data item D. This data item should be stored in the network in a way that makes finding it by other

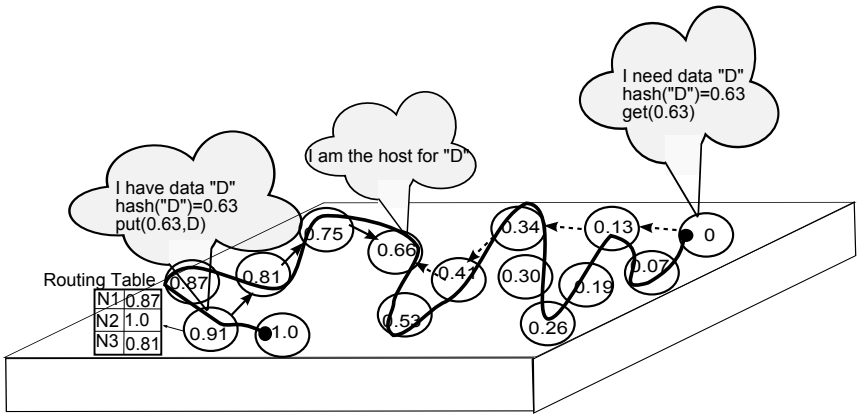

Figure 4.1: VCP organizes all the nodes in a structured cord and each node maintains a small amount of routing information. The routing table of each node contains the predecessor (first row), successor (second row) as well as other physical neighbors. When a node generates a data item (i.e. node 0.91 generates D), it stores this data item (or a reference to it) on the cord. Node 0 sends a request for item D to the node that hosts D.

nodes in the network an easy task. As discussed in the previous chapter this "lookup" problem can be easily solved using the concept of a DHT. The generating node stores the data item on a pre-defined node. Consequently all queries regarding this data item will be forwarded deterministically to that node. The pre-defined node is determined by a common hash function.

Now, the interesting questions are:

- How to set up the cord?
- How to route using the cord?
- How to replicate the data in order to assure maximum persistence of data with minimum usage of resources when nodes fail?

The idea behind VCP is to combine data lookup with routing techniques in an efficient way. VCP accomplishes this by placing all nodes on a virtual cord, which is also used to associate data items with. A hash function is used to hash data items to values in a pre-defined range $[S, E]$, which is completely covered by the participating nodes. Thus, each node maintains a part of the entire range. The routing mechanism relies on

two concepts: First, the virtual cord can be used to find a path to each destination in the network. Additionally, locally-available neighborhood information is exploited for efficient greedy routing towards the destination.

In comparison with the algorithms mentioned in the related work (Chapter 3), VCP borrows, and differs, from each of them. From Chord, VCP borrows its simple one-dimensional identifier, but nevertheless VCP has only one routing table in comparison to the two routing tables in overlay implementations. Moreover VCP incurs smaller routing states than the use of shortest path algorithms. From VRR, VCP borrows combining both routing and DHT services, though the details of the addressing and forwarding are entirely different. Moreover, VCP's routing table incurs only local information and (unlike VRR) needs not adhere to any far away node's information. VCP uses greedy forwarding over node coordinates, but (unlike GPSR) greedy forwarding guarantees packet delivery, does not require geographic information, and makes no assumptions about radio connectivity. From GRWLI, VCP borrows the notion of using virtual coordinates, but (unlike GRWLI) it uses a very simple coordinate construction algorithm, by which greedy forwarding guarantees packet delivery. As mentioned earlier, VCP focuses on efficient routing on a virtual cord. It features a predefined hashing range that allows applications to clearly associate data items to places in the cord.

4.2 Setting up the Cord

The design of VCP centers around a virtual, pre-determined range $[S, E]$, i.e. a one-dimensional cord. Each node in the network is responsible for a portion of the entire space, defined by its relative position to physical neighbors. This way it is possible to store data on the nodes, by mapping data items deterministically in the space using a hash function. The corresponding key-value pair is then stored at the node whose position is closest to the key. Routing of packets is done based only on the position of the physical neighbors. To retrieve data items, nodes have to apply the same hash function to find the key value, then they can route the request to the node whose position is closest to the key.

When a node joins the VCP network, it must set three important variables: its position, its predecessor, and its successor. Each node determines these values based on the positions of its single-hop neighbors. First, one node must be pre-programmed as

Table 4.1: Initial parameters for the join process

Parameter, Value	Description
Start $S = 0.0$	lowest position on the cord
End $E = 1.0$	highest position on the cord
Position $P = -1.0$	current position in the cord, -1 means the position is still unset
HelloPeriod $T_h = 1\,\text{s}$	time interval between `hello` messages
SetPosDelay $T_{ps} = 1\,\text{s}$	time interval before re-requesting a new position
SetVPosDelay $T_{vps} = 1\,\text{s}$	time interval before requesting a virtual position
BlockDelay $T_b = 1\,\text{s}$	blocking period to prevent assigning the same position to more than one node
Interval $I = 0.1$	interval between the two end positions $[S, E]$ and successor or predecessor position
VirtInterval $I_v = 0.9$	interval between node position and virtual node position
MaxVirtuals $V_{max} = 2$	Maximum number of virtual nodes that a node allowed to create

the initial node, i.e. it gets the position S. Furthermore, a number of initial variables are initialized in the startup phase as listed in Table 4.1.

We employ `hello` messages to discover the network structure, i.e. all neighboring nodes and their position in the cord. The `hello` message contains the position, predecessor and successor of a node. In the current implementation of VCP, the `hello` messages are transmitted by means of broadcasting, i.e. each node broadcasts a `hello` every T_h. T_{ps} is a delay that must have elapsed before a node may re-ask for a relative position in the case that the last attempt to get a relative position had failed. This delay is used to prevent asking multiple nodes at the same time. T_b is a delay we used to prevent assigning the same position to more than one node (i.e. to obtain a unique position for each node). We used intervals I and I_v to smooth the position distribution among nodes. Alternatively, the joining operation can also be executed using an on-demand mechanism, which has advantages in static networks or those with a high density.

Based on the periodically transmitted `hello` messages, the joining node gets information about its physical neighbors and their adjacent nodes. Algorithm 4.1 depicts

the handling of received `hello` messages: If the node that sent the `hello` message is not in physical neighbors table then we add it, otherwise we update its information in the table. Of course the `hello` message updates also the successor and predecessor information. If the node has not yet joined the network (i.e. its position equals -1), it calls the SetMyPosition() function listed in Algorithm 4.2 to get a relative position in the cord. An artificial join delay T_{ps} must have elapsed before re-asking for a relative position.

Algorithm 4.1: Handle `hello` Messages

Require: Locally stored state of all neighbors in set N
Ensure: Maintain neighbor set N and set virtual address
1: Receive (`hello`, Pos, SuccPos, PrePos) from node N_i
2: **if** $N_i \notin N$ **then**
3: $N \leftarrow N_i$
4: **else**
5: Update $N_i \in N$
6: **end if**
7: **if** $N_i == MySucc$ **then**
8: Update $MySucc$
9: **end if**
10: **if** $N_i == MyPre$ **then**
11: Update $MyPre$
12: **end if**
13: **if** $P == -1$ AND (Time() $-$ OldTime) $> T_{ps}$ **then**
14: OldTime $\leftarrow$ Time()
15: SetMyPosition()
16: **end if**

Each node joining the network has to receive at least one `hello` message from a node that already joined the cord in order to get a relative position in the cord. If a node can communicate with an end node (lines 2–20 in Algorithm 4.2), i.e. a node that has either position S or E, the new node takes over this end value as its virtual cord position by sending a packet `newpos` to the node that has that end. If it is the second node in the network (which can be easy detected by probing the old node position and successor) then the `newpos` message contains E as the new position. Otherwise the `newpos` message contains a new position between the end value and its successor or predecessor depending on the its old position. The receiving node `newpos` examines the "new position" field in the `newpos` message as shown in Algorithm 4.3. It can happen that multiple nodes ask for the end position at the same time, therefore we use a local variable to indicate if the node is already gave that end ($SentF$ and $Sent$). we

Algorithm 4.2: Position Update via SetMyPosition()

Require: Neighbor information stored in set N

1: **for** $\forall N_i \in N$ **do**
2: **if** Position(N_i) == S) **then**
3: **if** Successor(N_i) < S) **then**
4: SendNewPositionToNeighbor(N_i, E)
5: return;
6: **else if** Successor(N_i) == E **then**
7: $NeighbourNewPosition \leftarrow (S+E)/2$
8: **else**
9: $NeighbourNewPosition \leftarrow$ Successor(N_i) $-$ I $\times$ (Successor(N_i) $-$ Position(N_i))
10: **end if**
11: SendNewPositionToNeighbor(N_i, Pos_{old})
12: return;
13: **else if** Position(N_i) == E **then**
14: **if** Successor(N_i) == S **then**
15: $NeighbourNewPosition \leftarrow (S+E)/2$
16: **else**
17: $NeighbourNewPosition \leftarrow$ Predecessor(N_i) $-$ I $\times$ (Predecessor(N_i) $-$ Position(N_i))
18: **end if**
19: SendNewPositionToNeighbor(N_i, Pos_{old})
20: return;
21: **else**
22: **for** $\forall N_j \in N : i \neq j$ **do**
23: **if** Predecessor(N_i) == Position(N_j) **then**
24: $P_{temp} \leftarrow$ (Position(N_i) + Position(N_j))/2
25: temporarily store positions of N_i and N_j
26: SendBlockReq(N_j, P_{temp})
27: return;
28: **end if**
29: **end for**
30: **else**
31: **if** (Time() $-$ OldVTime) > T_{vps} **then**
32: OldVtime $\leftarrow$ Time()
33: temporarily store position of N_i
34: SendCreatVirtualNode(N_i)
35: return;
36: **end if**
37: **end if**
38: **end for**

use $SentF$ to indicate that the node is already sent the second node in the network its position, while $Sent$ indicates that the node gave the end it had and it is no anymore having that end. The old node should update its position, successor and predecessor accordingly, as well as acknowledge the `newpos`. At the beginning, when there is not yet another node in the network, the node pre-programmed as the first node sets the variable $SentF$, updates its successor information and sends a `firstack` message (lines 2–6 in Algorithm 4.3) to inform the new node that it will be the second node. Lines 7–14 and lines 15–22 in Algorithm 4.3 show the update and acknowledge when the old node was assigned the Start and the End position respectively. Here, the old node sends a `newpositionack` message instead of `firstack`. The purpose of these acknowledgments is to inform the new node that its attempt to get a position has succeeded, and it can now set its own variables and start to broadcast `hello` messages.

Algorithm 4.3: Handle New Position Messages

1: Receive (`newpos`, NewPos) packet from node N_i
2: **if** $((P == S) AND (NewPos == E) AND (!SentF)$ **then**
3: SentF$\leftarrow 1$
4: MySuccPos $\leftarrow E$
5: $MySucc \leftarrow N_i$
6: SendFirstAck(N_i)
7: **else if** $((P == S) AND (NewPos < MySuccPos) AND (!Sent))$ **then**
8: $Sent \leftarrow 1$
9: $MyOldPos \leftarrow P$
10: $MyPrePos \leftarrow P$
11: $MyPre \leftarrow N_i$
12: $P \leftarrow NewPos$
13: SendUpdatesMysuc()
14: SendAckNewPos(N_i,MyOldPos)
15: **else if** $((P == E) and (NewPos > MyPrePos) AND (!Sent))$ **then**
16: $Sent \leftarrow 1$
17: $MyOldPos \leftarrow P$
18: $MySuccPos \leftarrow P$
19: $MySuc \leftarrow N_i$
20: $P \leftarrow NewPos$
21: SendUpdatesMyPre()
22: SendAckNewPos(N_i,MyOldPos)
23: **end if**

The response to `firstack` and `newpositionack` is shown in Algorithm 4.4 and Algorithm 4.5 respectively. When the node is the second node in the network, it updates its predecessor as well as its own position information. In the current proactive

implementation we need to schedule `hello` messages. The aim of the `hello` message mechanism is to enable other nodes to join the network as well as to update their routing tables. As we describe in Section 4.5.2, if the protocol behaves reactively then there is no need to schedule `hello` messages. Instead the joining nodes have to send a `hello request` message. To handle the `newpositionack` messages, the nodes check the $NewPos$ field: if $NewPos$ is the End then the node updates the predecessor and its own position (lines 2–6 of Algorithm 4.5); if it was the $Start$ then it updates the successor and its own position (lines 8–12 of Algorithm 4.5). Like in the second node case, in both cases the node has to schedule `hello` messages because of the proactive implementation.

Algorithm 4.4: Handle Handle First Ack Messages

```
1: Receive (firstack, NewPos) packet from node N_i
2: if NewPos == E then
3:    MyPre ← N_i
4:    MyPrePos ← S
5:    P ← E
6:    ScheduleHelloMessage()
7: end if
```

Algorithm 4.5: Handle New Position Ack Messages

```
1: Receive (newpositionack, NewPos) packet from node N_i
2: if NewPos == E then
3:    MyPre ← N_i
4:    MyPrePos ← NeighbourNewPosition
5:    P ← E
6:    ScheduleHelloMessage()
7: end if
8: if NewPos == S then
9:    MySuc ← N_i
10:   MySuccPos ← NeighbourNewPosition
11:   P ← S
12:   ScheduleHelloMessage()
13: end if
```

If a node can communicate with two adjacent nodes in the cord, the new node gets a position between the values of these two adjacent nodes (lines 22–27 in Algorithm 4.2). The new node stores the position of predecessor and successor as well as its proposed position temporarily and then sends a `blockreq` message to the adjacent nodes. To reduce the number of transmitted messages we send only to the predecessor and

includes its successor in the `blockreq` message. Thus the node that received the `blockreq` message can verify if it was not blocked by another node and is still adjacent to the node contained in the message; if this test holds true, it blocks itself for a period of time T_b and then sends a `blockack` message. The blocking mechanism is used to prevent multiple nodes from getting the same relative position. Upon receiving a `blockack` message the new node activates the temporarily stored position information. Algorithm 4.6 and Algorithm 4.7 illustrate the details of block request and block acknowledgment processing. Note that the new node schedules `hello` messages and updates it successor.

Algorithm 4.6: Handle Block Req Messages

1: Receive (`blockreq`, P_{temp}, TempSuc) packet from node N_i
2: **if** (!$Blocked$)AND($MySuc == TempSuc$) **then**
3: $Block(T_b)$
4: $MySuc \leftarrow N_i$
5: $MySuccPos \leftarrow P_{temp}$
6: SendBlockAck()
7: **end if**

Algorithm 4.7: Handle Block Ack Messages

1: Receive (`blockack`) packet from node N_i
2: $MySuc \leftarrow MyTempSuc$
3: $MyPre \leftarrow MyTempPre$
4: $P \leftarrow P_{temp}$
5: ScheduleHelloMessage()
6: SendUpdatesMysuc()

Finally, if the new node can communicate with only one node in the network, which is neither at S nor E, then the new node asks this node to create a virtual position (lines 31–35). The reason for creating a virtual node is to maintain the connectivity and the consistency of the cord. The virtual node gets a position between that of the real node and its successor or predecessor. The newly joining node can now get a position between the real and the virtual position of the node in the cord. Notice that the node has to wait some time T_{vps} before asking for a virtual node. This timeout is used to encourage the node to find an end or adjacent neighbors and thus get a proper position in the cord without the need to setup a virtual position. We noticed that fewer virtual nodes lead to better routing paths. The mechanism for creating a virtual node starts

by storing the proposed position values and then sending a `virtualreq` message to the corresponding node. Each node can create up to $maxVirtuals$ virtual nodes. In order to ensure a unique position value for each node, the node has to block itself when creating a virtual node and send an acknowledgment as shown in Algorithm 4.8. When a node receives a `virtualack`, it activates the temporarily stored position information (see Algorithm 4.9).

Algorithm 4.8: Handle "createvirtual" Messages

1: Receive (`virtualreq`) packet from node N_i
2: **if** $numberofvirtuals < maxVirtuals) and (!Blocked)$ **then**
3: $VirtualPosition \leftarrow (MySuccPos - P) * I_v + P$
4: $MyTempSuccPos \leftarrow MySuccPos$
5: $VirtualSuc \leftarrow MySuc$
6: $MySuccPos \leftarrow (P + VirtualPosition)/2.0$
7: $VirtualSucPoistion \leftarrow MyTempSuccPos$
8: $MySuc \leftarrow N_i$
9: $VirtualPre \leftarrow N_i$
10: $VirtualPrePosition \leftarrow MySuccPos$
11: SendVirtualAck(N_i,MySuccPos,VirtualPosition)
12: **end if**

Algorithm 4.9: Handle Virtual Ack Messages

1: Receive (`virtualack`, NewPos, VirtualPosition) packet from node N_i
2: $MyPre \leftarrow MyTempPre$
3: $MyPrePos \leftarrow MyTempPrePosition$
4: $MySuc \leftarrow MyTempSuc$
5: $P \leftarrow NewPos$
6: $MySuccPostion \leftarrow VirtualPosition$

Figure 4.2 shows the joining process for a six node network with a position space of $[0,1]$. In each picture, a small circle indicates the communication range of the newly joining node. Initially, a pre-programmed node is assigned the lowest position, 0 (see Figure 4.2[a]). The newly joining node (Figure 4.2[b]) finds itself to be the second node, because it communicates with a node that has the lowest position and its successor is not set. Therefore the new node gets the end position, 1. The next node that joins the network (see Figure 4.2[c]) can communicate with a node that has the lowest end but its successor is set – therefore it gets this end position, 0, and hands the old node a position between the old one and its successor. The same happens in Figure 4.2[d], however this time with the other end, 1. The node joining in Figure 4.2[e] cannot

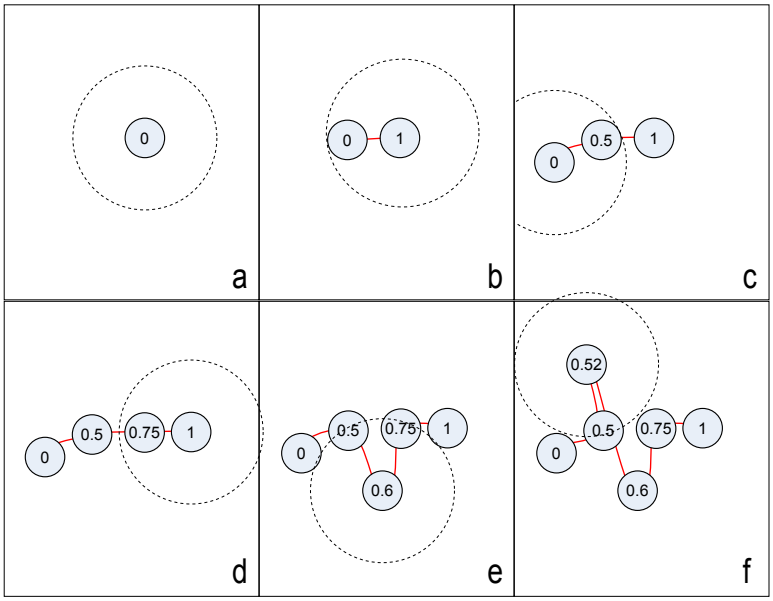

Figure 4.2: Basic join operation in VCP, six nodes are joining the network according to the rules described in Algorithm 4.2

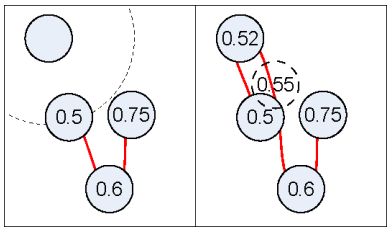

Figure 4.3: Operation of creating virtual node

communicate with any end, but it can communicate with two nodes adjacent on the cord. Thus, this new node gets a position between the positions of the adjacent nodes and it becomes the successor for the lower-numbered and predecessor for the higher-numbered node. In Figure 4.2[f] the new node can communicate only with one node whose position is not an end. Therefore the new node is forced to ask for a virtual node and get a relative position between the virtual node and the old successor. Figure 4.3 illustrates how a node gets a relative address by creating a virtual node.

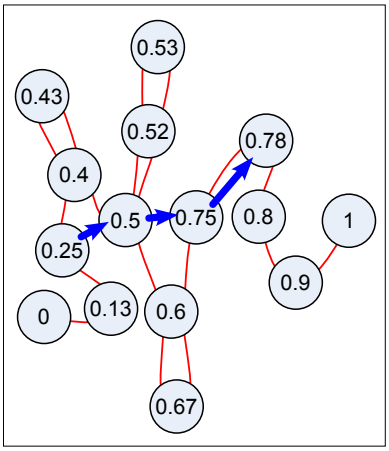

Figure 4.4: An example for a routing path using the virtual cord and greedy routing exploiting local neighborhood information

4.3 Routing on the Cord

We now describe routing in VCP. At the beginning we define the greedy forwarding rule and we prove that, for static networks Greedy forwarding based on local information guarantees finding path between any two nodes in the network. Then we address the failure of pure greedy routing to find paths in the presence node failure, by introducing algorithm for forwarding packets around failing nodes.

Figure 4.4 depicts the state of the network after adding 15 nodes. Routing in VCP is done using the virtual cord. Additionally, local neighborhood information is exploited for greedy routing. The greedy forwarding mechanism works as follows: a node with relative position P forwards a packet to its neighbor N_i which has the closest virtual position to the destination D_p. The forwarding is terminated if no more progress is possible, i.e. if the local coordinate P is closest to D_p. Based on the established cord, VCP will never get stuck in dead ends. Preliminary studies have shown that the path stretch is near optimal. Also, it is clear that the joining of a new node only affects a small number of nodes in the vicinity of the node and the complexity of the join operation is independent from the total number of nodes in the network. In fact, the insertion of a new node only affects $O(m)$ nodes, where m is the number of successors.

4.3.1 Greedy Routing

In VCP, packets are marked by their originator with their destinations' locations. As a result, a forwarding node can make a locally optimal, greedy choice when choosing a packet's next hop. Specifically, if a node knows its radio neighbors' positions, the locally optimal choice for next hop is the neighbor closest to the packet's destination. This forwarding logic results in successively smaller hops until the destination is reached. Thus for routing in VCP, each node has to first know its physical neighbors. Then a greedy algorithm, Algorithm 4.10, is employed to send packets to the node of the physical neighbor that has the position closest to the destination until no more progress is possible and the value lies between the positions of the predecessor and successor. VCP inherently relies on a previously established cord. Therefore, greedy routing will always find a path to the destination – it is not possible to run into a dead end. In addition, VCP allows taking shortcuts whenever a physical neighbor with a virtual number is available that is closer to the destination.

Continuing with our example, if node 0.25 in Figure 4.2 produces a data item, it first has to hash this item to the correspondent hash value – we assume a hash value of 0.781. Thus, node 0.25 will forward the message towards the destination node, i.e. in our case to node 0.5, which has the closest position to the value (0.781) among the physical neighbors. Afterwards, node 0.5 will send the message to node 0.75, and then node 0.75 will send it to node 0.78 as shown in Figure 4.2 (right). Node 0.78 will finally store the data and will not forward it any further because there is no more progress possible and the value lies between the positions of the predecessor and the successor.

Algorithm 4.10: Greedy Forwarding Algorithm

Require: Received data packet D for destination position D_p, locally maintained data set $[P_{min}, P_{max}]$
1: **if** $P_{min} \leq D_p \leq P_{max}$ **then**
2: StoreData()
3: **else if** $\exists N_i \in N : |\text{Position}(N_i) - D_p| < |P - D_p|$ **then**
4: Send(N_i, D)
5: **end if**

The great advantage of greedy forwarding is its reliance on only knowledge of the forwarding node's immediate neighbors [85]. The amount of state information that needs to be tracked is negligible and dependent on the density of nodes in the

wireless network, not the total number of destinations in the network. In networks where multi-hop routing is useful, the number of neighbors within a node's radio range will be substantially less than the total number of nodes in the network. Thus the scalability can by analyzed as follows; the most obvious measure of scalability of a routing protocol is the overhead associated with the maintenance of routing tables. In a WSN two measures of this overhead are important: the size of the routing table and the communication overhead required to keep it up to date. The size is not only refers to the memory size required to store the routing table, but also to how many entries of the table need to be adjusted when nodes join or leave. The communication overhead indicates how much communication is required to update each entry. In VCP the routing table of each node contains only its radio neighbors, hence the size is $O(m)$ where m is the node degree. Since a $hello$ message is enough to update each entry of the routing table, the communication overhead to update each entry is $O(1)$. From this analysis we can conclude that VCP has good scalability characteristics.

Theorem 4.3.1. *Given a static network and VCP, greedy forwarding based on local information guarantees packet delivery to the correct destination.*

Proof. Assume a packet, marked by its originator with a Key, gets stuck at a node that is not closest to the Key, say S, and there exists another node S' that is closest to the Key. Recall that in VCP each node can communicate at least with its predecessor and successor. The value of the successor is larger than S and the value of the predecessor is smaller than S. Therefore either the successor or the predecessor is closer to S' than S (if not, then $S = S'$). S' is closest to the Key. Consequently, either the successor or the predecessor is closer to the Key than S. Hence the packet will not get stuck at S and the packet will be greedily forwarded (based on the local information) either to the successor or the predecessor until it reaches a node for which neither the successor nor the predecessor is closer to the Key than the node itself. □

Theorem 4.3.2. *Given a static network, routing in VCP is loop free.*

Proof. A loop occurs when a packet revisits an already-visited node. In VCP each hop a packet travels shortens its distance toward the destination. Thus all visited nodes have a longer distance on the cord to the destination than the current node. Consequently, greedy forwarding will never choose to forward a packet to a node that has a longer distance to the destination than the current node. □

4.3.2 Failure management

The presented cord management is working very well as long as all nodes stay available after forming the cord. Greedy forwarding can guarantee the reachability of the destination only if there is no failure. However, in case of node failures, greedy

forwarding might fail and the cord might become unstable. To overcome the problem of finding a path towards the destination in case of node failures, we propose a new scheme to find an alternative path.

We use `hello` messages to identify failed nodes: in addition to successor and predecessor positions, we store the timestamps of the last `hello` message in the routing table, i.e. in the physical neighbor table. If a node did not receive a `hello` message from a neighbor for $k \times T_h$, where T_h is the hello message period, this neighbor is marked as a dead node. From the available information in the routing table, each node can locally check whether the final destination of a packet is one of its physical neighbors or not.

During packet routing there are two cases in which greedy forwarding cannot reach the correct destination because of a dead end in the cord. The first case means reaching a physical neighbor of the failed node. In this case, the packet can either be dropped or stored within the neighbor of the failed node. If the operation was to retrieve data items, then if replication is used, it is highly probable that the data is also on this node.

In the second case, the failed node is the next hop towards the destination, but it is not the final destination itself. In this case, we have to find an alternative path. The procedure is as follows. The neighbor of the failing node locally creates a so called *no path interval* NP-I. This interval corresponds to the range of keys that the dead node was responsible for. Then, the node sends a *no path* (NP) packet, which includes the NP-I to another active node in its neighborhood. This node is selected according to its position in the cord, which should be as close as possible to NP-I. In order to prevent routing loops, this information needs to be stored on all nodes involved in this process. However, the stored NP-I data is expected to expire after T_{np} s. Another function of *no path interval*, until T_{np} s expired, is that it prevents any new messages addressed with keys in this interval from entering zones that will not lead to the destination, i.e the calculated NP-I at a node is $[a, b]$ means that, at the moment this node can't find a path for packets addressed with values in the range $[a, b]$. Therefor any packet with destination $a + \epsilon < b$ should not be sent to this node. From now on, each node either transmits data using greedy forwarding towards the destination (if there is a neighboring node closer to the destination available), or it continues to send NP packets. Using the stored NP-I data, this information will never be sent twice. If a NP packet reaches a node that already has NP-I in its table, it has to send a *no path back* (NPB)

packet as an indicator of a detected loop.

The procedure of treating routing packets is shown in more detail in Algorithm 4.11. The interval $[P_{min}, P_{max}]$ is maintained by evaluating the distance between the current node and the neighbors on the cord. In particular, this interval is used to identify the final destination for each packet. The *no path interval* NP-I is calculated by the function $ComputeNoPathInterval()$ illustrated in Algorithm4.12. The functions $FindBiggerNode(BiggerNode)$ and $FindSmallerNode(SmallerNode)$ return (if available) the nodes with larger and smaller positions than the packet destination position respectively.

Algorithm 4.11: Handle Routing Packets

Require: Received data packet D for destination position D_p, locally maintained data set $[P_{min}, P_{max}]$
1: **if** $P_{min} \leq D_p \leq P_{max}$ **then**
2: StoreData()
3: **else if** $D \in$ NP-I **then**
4: Send(D->src,NPB, D)
5: **else if** $\exists N_i \in N$: |Position(N_i) $- D_p| < |P - D_p|$ **then**
6: Send(N_i,ROUT, D)
7: **else**
8: ComputeNoPathInterval()
9: $N_c \leftarrow$ FindClosestNode(N_i)
10: Send(N_c, NP, D)
11: **end if**

Algorithm 4.12: Compute No Path Interval

1: **if** $Key > P$ **then**
2: **if** $FindBiggerNode(BiggerNode)$ **then**
3: $NP\_E \leftarrow (FailedNodePosition + BiggerNodePosition)/2$
4: **else**
5: $NP\_End \leftarrow E$
6: **end if**
7: $NP\_Start \leftarrow (FailedNodePosition + FailedNodePrePosition)/2$
8: **else**
9: **if** $FindSmallerNode(SmallerNode)$ **then**
10: $NP\_Start \leftarrow (FailedNodePosition + SmallerNodePosition)/2$
11: **else**
12: $NP\_Start \leftarrow S$
13: **end if**
14: $NP\_End \leftarrow (FailedNodePosition + FailedNodeSuccPosition)/2$
15: **end if**

An example for packet forwarding in the case of a node failure is illustrated in

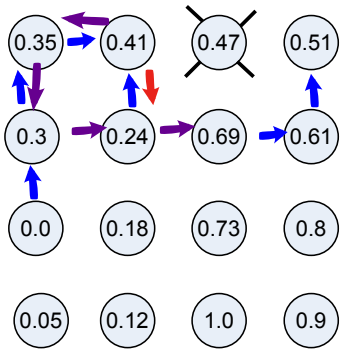

Figure 4.5: Routing in case of a node failure: node 0.41 generates a NP packet that is forwarded until node 0.69 continues to regularly forward the packet

Figure 4.5. In this example, node 0.0 produces a data packet address to 0.52. According to the greedy routing principles of VCP, the data packet will be forwarded by nodes 0.30 and 0.35 until it reaches node 0.41. At this node, a dead end is detected because the previously-existing node 0.47 has failed. Thus, node 0.41 will create a *no path interval* NP-I=[0.44,1] and send a NP packet back along the path to node 0.35. Similarly, the NP packet is forwarded until it reaches node 0.24. This one, according to the programmed rules, tries node 0.47 again. However, 0.47 detects a loop and sends a NPB packet to node 0.24. In turn, node 0.24 tries to find another path by sending a NP packet to node 0.69. Finally, this node can resume greedy forwarding toward destination node 0.51.

4.4 Data Replication

In order to avoid losing data when nodes fail, additional copies of the data items must be stored on other nodes. However, data replication does not only consume storage capacity, but also cause a communication overhead to increase, which can quickly lead wireless sensor networks in particular to be overloaded and eventually drain the battery. Therefore in WSNs data replication schemes should utilize the network resources as good as possible. In this section we propose replication techniques to ensure persistence of data when nodes fail. First we propose two localized replication techniques, which replicate each data item in the direct vicinity. Hence, these two schemes incur only local communication. The second approach replicates the data at dispersed places then at each place replicates the data locally.

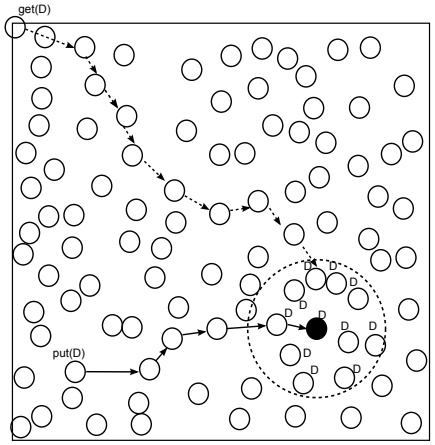

Figure 4.6: Replication of the inserted data on neighbors within a node's radio range

4.4.1 Local Replication

Replication on the Physical Neighbors

The first local replication scheme employs physical neighbors to accomplish local replication of key-value pairs (Figure 4.6). When a packet reaches its destination, it will be retransmitted to all physical neighbors. The retransmission of data is done using a broadcast message, thus we need only one message to store a data item in all physical neighbors. This way we can be sure that if a node failed, there are other nodes that stored the data items of the failed node. Note that we store for each data item a pointer that indicates the node from which it originated. When a node re-joins the network it will ask its physical neighbors for its data by broadcasting a *DATA_REQ* message. Here too we need only one broadcast to ask all neighbors. This message contains the physical neighbors of the joining node.

Algorithm 4.13 illustrates how VCP treats the *DATA_REQ* message. Nodes count the number of messages that have to be sent to the joining node. Then the nodes have to wain some time before sending acknowledgment. The waiting time is a sum of two components, the first component is deterministic and the second is random. we used the random time to prevent multiple nodes that have the same number of data form sending synchronously. The waiting is inversely proportional to the number of

data messages to be sent. Therefore, physical neighbor with more data items will send an acknowledgment by sending a *DATA_ACK* before other nodes. Upon receiving a *DATA_ACK* from the joining node, it will ask the answering node to send the data. This message is a broadcast that contains the position of the responding node, thus it will be heard by other nodes and will keep them from responding (Algorithm 4.14).

Algorithm 4.13: Handle DATA_REQ Messages

1: Receive (*DATA_REQ*, N) packet from node N_i
2: $computecommondata$
3: $wait$
4: **if** $!StopSending$ **then**
5: $Send(DATA_ACK)$
6: **end if**

Algorithm 4.14: Handle DATA_SEND Messages

1: Receive (*DATA_REQ*, NPos) packet from node N_i
2: **if** $NPos == P$ **then**
3: $SendData(N_i)$
4: **else**
5: $StopSending \leftarrow 1$
6: **end if**

The advantage of this approach is that nodes re-claim data items on demand and there is no need to periodically refresh the replica nodes. Thus if there is no node failure there will be no periodic refresh. Recall that VCP routes all packets (data and queries) to the responsible nodes. Thus the query will reach the node closest to the desired key. If the node has the desired data item it will answer the query. Otherwise it will broadcast the query to the physical neighbors, thus if the data item still exist in the network, it will be found (also here we need only one message to send the query to all neighbors). Therefore there is no extra communication overhead for querying a data item.

Replication on the Adjacent Neighbors

The second local replication technique utilizes the cord for replication as depicted in Figure 4.7. This replication scheme ensures exactly $2f + 1$ copies of the inserted data item independently from the network's node degree, where f is the maximum number of replications on each side of the cord. The node hosting a data item has to replicate

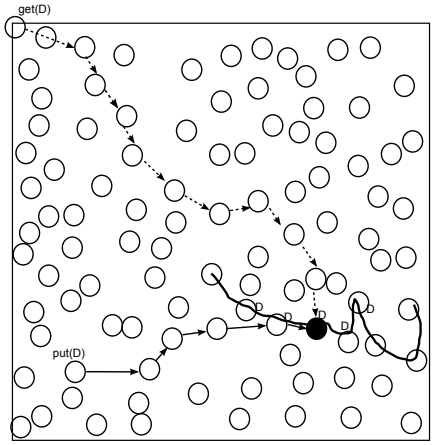

Figure 4.7: Replication of the inserted data on adjacent neighbors on the cord

this data on the f nodes succeeding and the f nodes preceding it on the cord. Because adjacent nodes can be located not only in the radio range of the node, broadcasting data items is not feasible. Hence the replication is done using unicast messages.

4.4.2 Global Replication

Relying only on local replication is of little use if all the nodes in an area fail at the same time (e.g., a fire destroys all nodes in a region). Therefore we also developed a global replication scheme (Figure 4.8). This replication approach is more resilient to clustered failures than local replication techniques. It stores each data item multiple times at dispersed locations (using multiple hash functions).

At each location we then use the local storage approach. The reason behind using local replication in addition to the global replication is to cut back on multi-hop data refreshing. Otherwise, the only chance to restore a data item after a node holding a copy fails, is to schedule data refresh messages by other nodes holding this data item. Given that these replica nodes can be far away from each other, the data refresh approach is very costly. To query a data item, three strategies can be used. We call the first strategy "sequential querying". Here, after asking the first location and, if the query did not find the data item, it continues from this location to other possible storage locations. If the sink node did not get answer it will retry the same query. This method is communication

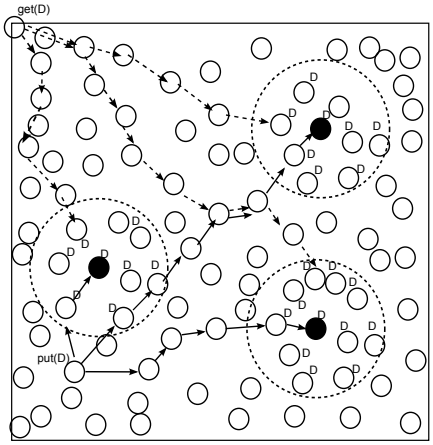

Figure 4.8: Replication of the inserted data to dispersed locations on the cord, applying the local replication technique at each location

efficient in the best case (finding the data at the first visited location or all locations have the data item) and it performs good in average cases, but it performs very bad in the worst case and can imply a very long delay.

The second strategy is to use multiple querying, i.e. to ask the first location and then, if there is no response, ask the second location and so on (modulo m). This method is better than the fist one in the worst case, but it can also imply a long delay. The last technique is parallel querying, i.e. to ask all locations at the same time. This method performs bad in the best case compared to the sequential and multiple querying methods, yet it has an optimal performance in the worst case. We used in our implementation the parallel querying method, nevertheless the other two methods are also applicable.

4.5 Further Extensions

At the beginning of this chapter we gave a detailed design of the two essential parts of VCP to work: the construction of the network and the routing algorithm. Then we described how to route around node failure and how to replicate the data in order to increased data availability when nodes fail. In this section we provide some extensions of VCP that can be useful for special applications. If the keys are supposed to be

uniformly distributed, then a uniformly distributed node positions can be desirable, therefore we introduce and approach for cod refinement. In the second extension we describe a reactive implementation of VCP, which can be useful for delay tolerant sensor networks. If the nodes are supposed to store data for long time, it is highly probable that the storage capacity of node get exhausted. Therefore, cooperative storage strategies are worth investigating.

4.5.1 Refinement of the Cord

As previously stated, greedy forwarding guarantees reachability between any two nodes in the cord, however there is no guarantee of a uniform distribution of the relative positions on the cord. Therefore if the cord is supposed to be used as an in-network storage mechanism and if the data is supposed to be uniformly distributed, then it is desirable to have a uniform cord. To refine the cord each node has to compare its position with the position of its predecessor and successor. If its position is far away (based on a refinement factor) from the middle, then the node should adjust its position towards the middle. To maintain a consistent cord it is important to block the predecessor and the successor before changing the position of the node. Otherwise it can happen that the successor position value becomes less than the node's position value or the predecessor position value becomes greater than node's position value. This refinement can be done during the construction of the cord or after constructing the cord and before starting to use the cord to store data.

There are two possible ways to do this refinement. The first possibility is to send special blocking messages (unicast) to the predecessor and the successor. Upon receiving a block acknowledgment from both the predecessor and the successor, a node can adjust its position. The second possibility is to integrate the refinement with the hello messages instead of sending special refinement messages. As shown in Algorithm 4.15 we extended the `hello` message with an extra field. During refinement this field is either set to `refineblockreq` if the node's position needs to be adjusted or it is set to `refineblockack` if the node is already blocked. Otherwise a normal `hello` message is sent. Upon receiving this `hello` message the normal `hello` message handler will be used in addition to testing the type field as depicted in lines(17–29) of Algorithm 4.16.

Algorithm 4.15: Hello Refinement

1: **if** $((Time() > start_refine_time) AND (Time() < end_refine_time))$ **then**
2: **if** $(!Blocked) AND (ABS((P - MyPrePos) > refine_fact * ABS(MySuccPos - P)))$ **then**
3: $Type \leftarrow refineblockreq$
4: **else if** $(Blocked)$ **then**
5: $Type \leftarrow refineblockack$
6: **end if**
7: **else**
8: $Type \leftarrow hello$
9: **end if**
10: $SendHello(Type)$

Algorithm 4.16: Handle `hello` Messages with Refinement

Require: Locally stored state of all neighbors in set N
Ensure: Refine the cord in addition to Maintain neighbor set N and set virtual address
1: Receive neighbor information from node N_i
2: **if** $N_i \notin N$ **then**
3: $N \leftarrow N_i$
4: **else**
5: Update $N_i \in N$
6: **end if**
7: **if** $N_i == MySucc$ **then**
8: Update $MySucc$
9: **end if**
10: **if** $N_i == MyPre$ **then**
11: Update $MyPre$
12: **end if**
13: **if** $P == -1$ AND (Time() − OldTime) $> T_{ps}$ **then**
14: OldTime ← Time()
15: SetMyPosition()
16: **end if**
17: **if** $(Type == refineblockreq) AND (!Blocked) AND (N_i \in (mySucc, myPre))$ **then**
18: $Block(T_b)$
19: **end if**
20: **if** $(Type == refineblockack) AND (!Blocked) AND (N_i \in (mySucc, myPre))$ **then**
21: **if** $(N_i == MySucc)$ **then**
22: $my\_succ\_blocked \leftarrow 1$
23: **end if**
24: **if** $(N_i == MyPre)$ **then**
25: $my\_pre\_blocked \leftarrow 1$
26: **end if**
27: **if** $(my\_suc\_blocked) And (my\_pre\_blocked)$ **then**
28: $P \leftarrow (\text{Position}(MySucc) + \text{Position}(MyPre))/2$
29: **end if**
30: **end if**

4.5.2 Reactive Implementation

The join operation of VCP, discussed in Section 4.2, is relying on periodic hello messages. However it is possible to implement VCP reactively; when a new node wants to join the cord, it simply asks for hello messages by broadcasting a hello message request to its physical neighbors. The physical neighbors that already joined the network respond by broadcasting a hello message. This way it is not necessary anymore to send periodic hello messages during the joining operation. The same holds true for updating routing tables. Instead of periodically updating the routing tables (physical neighbor tables) using the hello messages sent by physical neighbors, routing table can be updated on-demand by sending a hello request to the physical neighbors when there is a data item that needs to be routed. On one side this implementation saves hello messages which eventually saves power, on the other side this implementation imposes an extra delay.

4.5.3 Cooperative Storage with VCP

Because sensor nodes have limited resources, it can happen that the storage capacity of a single node gets exhausted. VCP offers a cost-effective mechanism to find alternative nodes which can offer part of their storage capacity to store data in place of other nodes. These nodes in VCP are the successors and predecessors on the cord. This way, if the storage capacity of a node is full, this node can send new data messages to either its successor or its predecessor. If the successor (or predecessor) is also full, it can in turn send new data items to its own successor (or predecessor) until reaching a node that can store the data item. To answer queries if a data item is not available at the node itself, it should ask its proceeding and succeeding node. To avoid the extra search overhead that can be introduced as a result of long searches on the preceding and succeeding nodes, Bloom filters [86] can be used. Bloom filters offer a way of representing a set of elements as a compact summary supporting membership queries. Consider a set of n elements to be represented using an m-bit long vector, initially set to 0. A set of k independent hash functions $h1, ..., hk$ is chosen where each function maps each item in the universe to a random number uniformly distributed in the range $1, ..., m$. For each element x to be represented, the bits at positions $hi(x)$ are set to 1 for $1 \leq i \leq k$. A bit may be set to 1 multiple times. This bit vector serves as a summary. To find out if an

element y is in the data-list, we check the positions $h1(y); ...; hk(y)$ in the bit vector. If they are all set to 1, then we can infer that y is in the data-list, though there is a (small) probability of being wrong (false positive). The critical feature of a Bloom filter is that the probability of false positives decreases exponentially with u if the number of hash functions k is chosen optimally. It is shown in [87] that the probability of a false positive is given by $(1-(1-\frac{1}{m})^{nk})^k$. We now explain how this concept can be applied to the problem of data storage in a sensor network. Each node maintains a Bloom filter for its successor and its predecessor. Thus the Bloom filter can be used to represent compactly which nodes store data on behalf of other nodes. By Bloom filter based matching, we can infer if a node stores a certain data item for other nodes. Hence a node can then forward the query to a restricted subset of the adjacent nodes.

4.6 Summary

In this chapter we have described VCP in detail. VCP is a virtual position based routing protocol that provides traditional DHTs services tailored to WSNs. VCP maintains the advantages and avoids the drawbacks of the protocols mentioned in the Related Work chapter. Unlike other, similar routing protocols that rely on local information for routing, in static networks greedy forwarding in VCP guarantees packet delivery. Because greedy forwarding relies only on local information, VCP scales good. The joining of a new node affects at most two nodes (the successor and the predecessor). The virtual position maintenance as well as the protocol itself are simple. Therefore VCP can be implemented on top of any MAC layer. We have also presented in this chapter a failure management approach to route around failing nodes, as well as replication approaches to improve persistence of data when nodes fail. We have further discussed some extensions that can be used in specific scenarios.

Chapter 5

Simulation and Evaluation

We evaluated VCP in a detailed simulation using OMNET++. The aim of the simulation part is to give a detailed performance analysis of VCP under different scenarios and to compare it with other similar protocols. Later, in Chapter 6 aiming to show that VCP's feasibility to run on low resource sensor devices, we implemented a prototype on sensor nodes in our lab.

We simulated the algorithm on a variety of scenarios [88–90] including different network sizes, data rates and topologies. A detailed description of the scenarios is provided in Section 5.2. Unless stated differently, we performed ten replications of each simulation experiment with different random seeds. In order to provide a better understanding of the performance of VCP, we compared VCP to standard ad hoc routing technique. This comparison outlines the usability of VCP especially in typical WSN scenarios where nodes are less mobile compared to MANETs. The most recent proposals for routing protocols in both the WSN and the MANET domain have been evaluated in comparison with AODV. We decided to explicitly use DYMO for our comparison as it is the designated successor of AODV. To compare the performance of VCP with prior work in DHT-based routing protocols for wireless sensor networks, we also simulate VRR [78], which has been shown to offer higher packet delivery ratios and lower latencies than several other ad-hoc routing protocols [78].

5.1 Simulation Environment

For the analysis of the VCP protocol, we implemented a simulation model of the protocol in OMNeT++ [91]. OMNeT++ is a discrete-event simulator which is free for academic

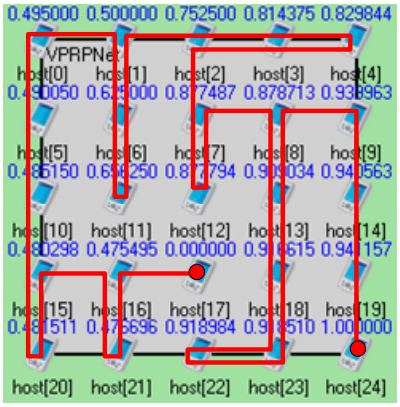

Figure 5.1: Simulation setup: nodes are deployed either in a grid or randomly a rectangular area

use. In addition, there are some extensions, such as the INET framework and the Mobility framework, which have been released under the GNU General Public License (GPL). We used the INET framework that provides detailed simulation models of the MAC and the physical layer. In our simulation, we built our protocol on the top of the IEEE 802.11 Wireless LAN protocols. An example of a 25 node network is depicted in Figure 5.1. The nodes are deployed in a grid on a rectangular area. Each node can communicate only with its direct neighbors that are aligned either horizontally or vertically, but not on the diagonal. The figure includes the virtual relative positions and the virtual cord connecting all the nodes.

For all communications, the complete network stack is simulated and wireless modules are configured to closely resemble IEEE 802.11b network cards transmitting at 2 Mbit/s with RTS/CTS disabled. On the MAC layer, each correctly received unicast packet is followed by an Acknowledgment (ACK) to the sender, which retransmits the packet up to 7 times until this ACK is received. Nevertheless, all routing protocols do not receives notification from the MAC layer when a packet exceeds its maximum number of retransmit retries. For the simulation of radio wave propagation, a plain free-space model is employed, with the transmission ranges of all nodes adjusted to a fixed value of 50 m. In order to compare the results with measurements for VRR [78] and DYMO [92], all simulation parameters used to parameterize the modules of the *INET Framework* are summarized in Table 5.1. For the constants that used as internal parameters of DYMO, we used the values presented in the paper [92] which reflect the

Table 5.1: INET Framework Module Parameters

Parameter	Value
mac.address	auto
mac.bitrate	2 Mbit/s
mac.retryLimit	7
mac.maxQueueSize	14 Pckts
mac.rtsCts	false
decider.bitrate	2 Mbit/s
decider.snirThreshold	4 dB
snrEval.bitrate	2 Mbit/s
snrEval.headerLength	192 bit
snrEval.snrThresholdLevel	3 dB
snrEval.thermalNoise	−110 dB
snrEval.sensitivity	−85 dB
snrEval.pathLossAlpha	2.5
snrEval.carrierFrequency	2.4 GHz
snrEval.transmitterPower	1 mW
channelcontrol.carrierFrequency	2.4 GHz
channelcontrol.pMax	2 mW
channelcontrol.sat	−85 dBm
channelcontrol.alpha	2.5

suggested values from IETF MANET working group. For the constants of VRR we stick with values used in the original paper [78].

5.2 Evaluation Metrics and Scenarios

Throughout our simulations, we used a number of metrics to evaluate and compare the performance of the protocols. We now present definitions and explanations for these metrics.

- Stretch ratio: the ratio between length of the path traversed by a routing protocol, in comparison with the optimal shortest paths in the network graph. If all packets are all transmitted with the same transmission power, this metric also reveals the radio power consumed to deliver a packet from the source to the destination.
- Join messages overhead: the number of control messages nodes need to join the network.
- Success rate: the ratio between the number of packets originated by the application layer sources and the number of packets received by the final destination. This metric represents the useful work done by a routing algorithm.

- Path length: how long are the routes found by a routing protocol?
- End-to-end delay: the delay a packet experiences on its path from source to destination. The path length can also serve as an indicator for the delay, but the two metrics may vary due to congestion in the network.
- MAC layer collision ratio: the ratio between collided packets and received packets on the MAC layer at the sink node. This metric indicates the overhead a routing protocol causes on the underlying layer.
- Total number of transmitted messages: the number of messages transmitted by all nodes during the simulation experiment. This is an important metric for comparing the protocols, as it measures the scalability of a protocol, i.e. the degree to which it will function in congested or low-bandwidth environments, and its efficiency in terms of node battery power consumption.
- Refresh messages: the total number of retransmitted data copies during the simulation. This metric indicates the communication overhead imposed by a protocol to increase data availability during node failure.
- Query messages: the total number of messages used to retrieve data items during the simulation. This metric indicates the communication overhead imposed by a protocol to find data items during node failure.

We used these metrics to evaluate the protocols in three different scenarios. In the first set of simulation experiments, the goal was to show that VCP guarantees reachability with a low cost and to show that VCP is a scalable protocol. Thus we measured the join overhead, stretch ratio, packet delivery ratio, and end-to-end delay under different network sizes and data rates. Additionally we examined the cord refinement.

We varied the network size from 25 to 400 nodes and adapted the size of the simulation area to keep the node density constant. Moreover, we studied two different traffic patterns. After joining the network, each node uniformly selects a start time in the time interval $[0, 100)$ s. Then, in the first traffic scenario, we evaluated bursty traffic, i.e. all messages were sent with a uniformly distributed inter-departure time in $[0, 1)$ s. In the second traffic scenario, a constant packet stream was analyzed to compare the protocol behavior under artificial traffic conditions. We varied the traffic load in

Table 5.2: Summary of simulation scenarios in the first set of experiments

Input Parameter	Value
Number of Nodes	25-400
Playground size	200 m × 200 m to 820 m × 820 m
Node placement	Grid, Random
Data rate	CBR, 1 pps ... 400 pps or VBR 2 pps ... 400 pps
Initialization time	40 s
Start of data transmission	uniformly distributed in $[0, 100)$ s
Number of data transmissions	100
Destination node	Upper left node or Random

the range of 1 pps ... 400 pps (packets per second). For statistical correctness, each experiment was executed five times for different data item keys. Table 5.2 summarizes the scenarios used in the first set of simulation experiments.

In the second set of experiments we evaluated VCP in comparison with DYMO and VRR. In this comparison we explored the performance of these protocols regarding the path length, end-to-end delay and MAC-layer collisions. In these experiments we fixed the network size at 100 nodes. Nevertheless, we changed other parameters as shown in Table 5.3. We also inspected the routing resilience to node failures, which indicates the ability of a routing protocol to withstand and adapt to node failures. We measured the success ratio when a part of the nodes are flipping between on and off state.

In the last set of experiments we investigated different data replication schemes. Here we investigated how VCP can be used to keep data available to queries in spite of node failures. Table 5.4 summarizes the parameters of the performed experiments.

We visualized the results using box plots. We chose box plots because they are more robust in the presence of outliers than the classical statistics based on the normal distribution. In a box plot, the median is used instead of the mean to indicate the central tendency. The interquartile range (IQR) is a robust way of describing the dispersion of the data. The IQR is the range within which the middle 50% of the ranked data are found. This is also the range between what is called the lower quartile value and the upper quartile value. The median and the IQR are used to construct the box. It has a height equal to the IQR and is drawn so that it starts at the lower quartile value and stops at the upper quartile value. A horizontal bar is drawn at the height of the

Table 5.3: Summary of simulation scenarios for experiments comparing several protocols

Input Parameter	Value
Number of Nodes	100
Playground size	180 m × 180 m or 400 m × 400 m
Node placement	Grid and random
Data rate	CBR, 1 pps or 0.1 pps
Initialization time	400 s
Start of data transmission	uniformly distributed in [400, 418] s or [400, 580] s
End of data transmission	490 s or 1 300 s
Destination node	Upper left node
Fraction of failing nodes	0 %, 20 %, 40 %, 60 %, 80 %, or 100 %
On time	uniformly distributed in [0, 120] s
Off time	uniformly distributed in [0, 60] s
Start of node failures	400 s
End of node failures	1 300 s

Table 5.4: Summary of simulation scenarios for data replication experiments

Input Parameter	Value
Number of Nodes	50-400
Playground size	160 m × 160 m
Node placement	Grid and random
Query rate	2 qps
Query period	300 s
Data Types	20
Sink node	Upper left node
Fraction of failing nodes	0 %, 20 %, 40 %, 60 %, 80 %, or 100 %
On time	uniformly distributed in [0, 120] s
Off time	uniformly distributed in [0, 60] s

median. A provision is also made for the representation of extreme values. An upper extreme value limit is computed as the upper quartile range $+1.5 \times IQR$ and the lower extreme value limit as the lower quartile range $-1.5 \times IQR$. The box plot emphasizes the presence of outliers by marking them with a circle.

5.3 Performance Evaluation of VCP

In this section, we show the first set of simulation results. The results have been collected from several simulation experiments to investigate the impact of different parameters like network size and traffic load on the performance of our protocol.

5.3.1 Join Overhead and Join Duration

At the beginning we explore the communication overhead and the time required to join the VCP. To measure the communication overhead in the joining operation, we count the number of control messages sent by each node to join the network in the initialization phase. The left-hand side of Figure 5.2 shows the number of messages sent by each node to join the network. On the x-axis we vary the number of nodes in the network, and the y-axis shows the number of sent packets by each node to join the network. The figure confirms that the control messages needed to join VCP are very low and independent from the network size. For example, when starting a network with 400 nodes, the average number of control messages per node is 4 messages. In contrast, a single flood of the network requires 400 messages. Moreover, it should be noted that the control messages in VCP are lightweight: its payload contains only few parameters as discussed in Section 4.2. We also measured the time when the last node joined the network. As you can see in the right-hand side of Figure 5.2, the time increased with increasing network size. This increase is a result of the fact that the virtual position is propagated throughout the network starting from the node that was pre-programmed as first node. Nevertheless, the time required to build the network is reasonable. We can see from the figure that even for the largest network all nodes were active in less than one minute.

As we explained in Section 4.2, the current implementation uses `hello` messages to stimulate new nodes to ask their physical neighbors who are already active in the network for a virtual position. Since VCP uses only local information for the joining

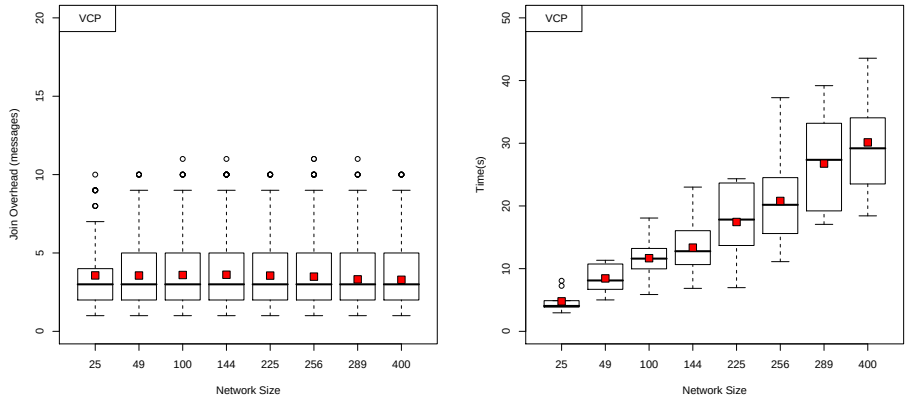

Figure 5.2: Initialization phase: number of control messages sent by each node and the required time in order to join the network

operation, concurrent joins of multiple nodes in different regions of the network are allowed. This has no effect on the consistency of the virtual cord. A problem may arise if multiple nodes join the network simultaneously in the same region of the network. In that case, the same position might be assigned to multiple nodes. We solved this problem by introducing the blocking mechanism discussed in Section 4.2. Thus, each node in the networks receives a unique virtual position.

5.3.2 Quality of Routing Paths

In order to inspect the quality of the routing paths, we examined the stretch ratio, i.e. the ratio between the length of the path traversed by VCP and the shortest path. For different network sizes, we measured the path length from all nodes to the upper left node. Recall that for a network of size n which is deployed in a rectangular area and in which nodes can communicate only with their direct neighbors, the average path length is $l_{avg-c} = \sqrt{n} - 1$.

Figure 5.3 shows the measured stretch ratio in the simulations as we varied the network size from 25 to 400 nodes. The stretch ratio increases with the network size, however, this increase is reasonable and the mean and median stay below 25%. This low stretch level outlines the optimal path selection of VCP.

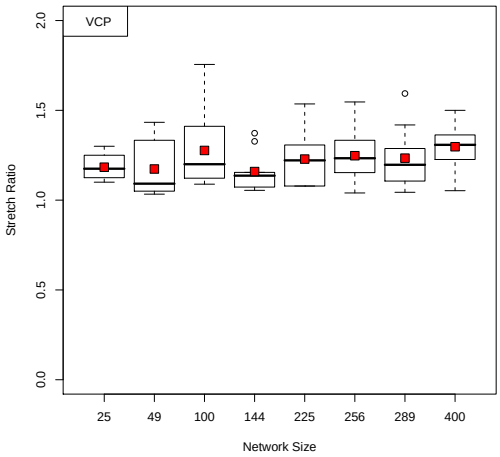

Figure 5.3: Stretch ratio for different network sizes

5.3.3 Influence of the Network Size

We performed a series of experiments to explore the effect of the network size on the performance of VCP. We simulated networks consisting of 100, 200, 300 and 400 nodes. We deployed the nodes randomly in rectangular planes of sizes 200 m × 200 m, 300 m × 300 m, 400 m × 400 m, and 500 m × 500 m, respectively. Each node in the network sends one packet per second to a random destination. All nodes start sending at a random time in the interval $[100, 280]$ s and stop sending at 490 s.

Figure 5.4 shows the results of the ten replications we performed for each network size. It is clear that the path length increases with the network size in a logarithmic manner. However, there are a few nodes that used a path length significantly larger than the shortest path to reach the destination. Taking a look on the average and mean path lengths, they are almost half of the worst case shortest path l_{avg}. Also, more than 75% of the nodes have a path length smaller than l_{max}.

For instance, if we assumed an optimal situation in which the nodes are deployed uniformly with adequate node density, then for a network with an area of 500 m × 500 m and nodes with a communication range of 50 m, $l_{max} = 15$(based on the euclidean distance between two corners). The results show that for this network 75 % of the paths were less than 12 hops, which is an indication of good path selection. On the other hand, the end-to-end delay is proportional to the path length as a result of the

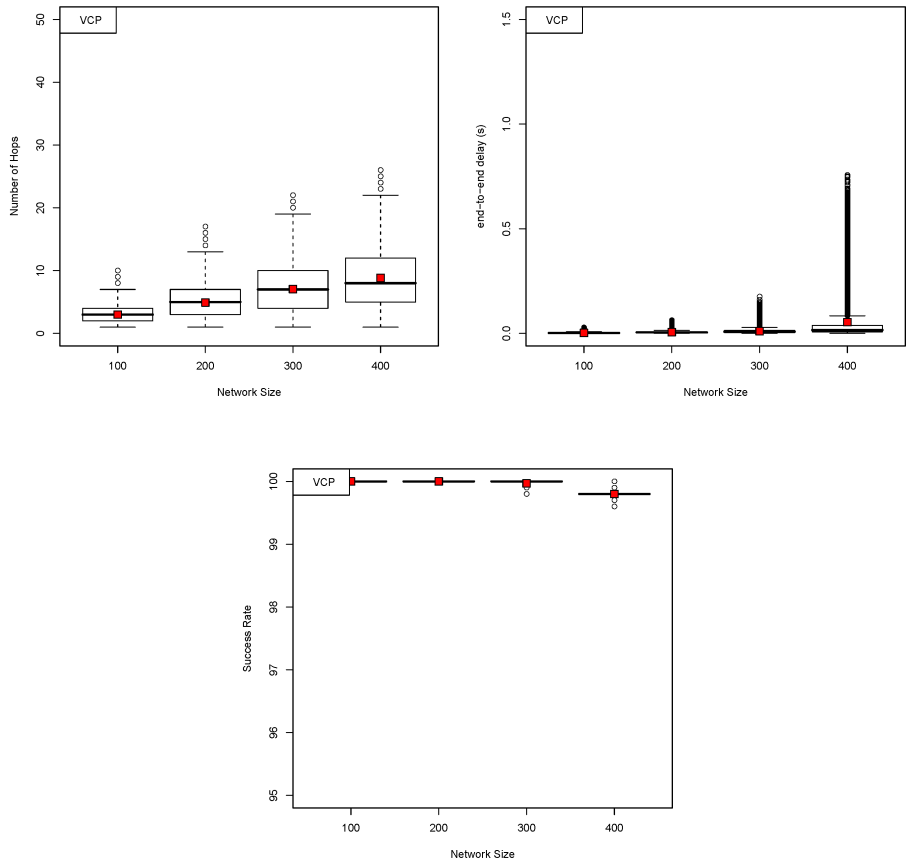

Figure 5.4: Performance evaluation results of path length, end-to-end delay and success rate with increasing network size

propagation delay because it is not necessary to queue packets. Moreover, the success ratio was 100 % for most network sizes. The packet losses at the lower layers cause a slight decrease in the success rate for the largest network size, but even then it is still above 99.6 %.

5.3.4 Influence of the Traffic Load

To study the behavior of our protocol under varying traffic load, we kept the number of nodes in the network constant at 100 nodes. Each node in the network sends 100 packets to the same destination. For the first experiments, the time between successive packets was randomly selected in the interval $[0, 0.5]$ s, which is equivalent to sending at least 2 pps. Afterwards, we decreased this time interval down to $[0, 0.005]$ s, which is

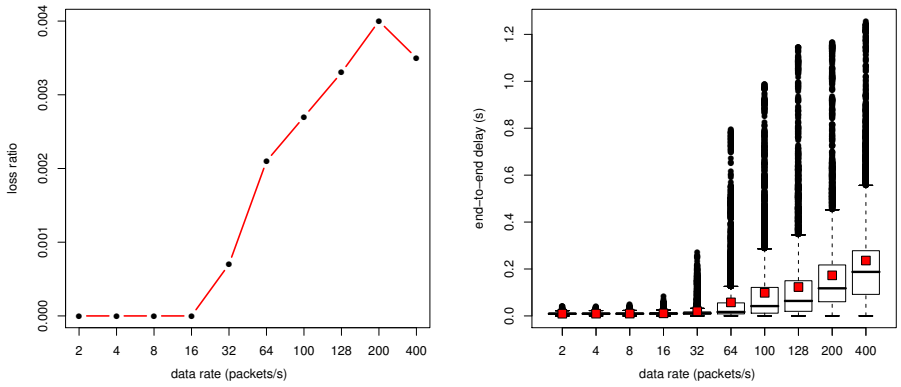

Figure 5.5: Performance evaluation results: effect of increasing VBR traffic rate on loss rate and end-to-end delay

equivalent to sending at least 400 pps. As shown in Figure 5.5, packets can be delayed at the MAC layer due to congestion. As a result, the end-to-end delay increases with increasing traffic load. Nevertheless, the delay is still in an acceptable range. For sending packets with a rate below 16 pps, the effect of congestion is negligible. However, the delay reached a peak of 1.2 s when sending with rate of 400 pps compared to only 0.03 s in the other case. The effect of increasing traffic was not so big on the packet delivery rate. As you can see from Figure 5.5, the loss ratio was below 0.4 %, therefore the success ratio is above 99.6 %. This loss is due to collisions on the MAC layer.

We repeated the same experiments using a constant packet rate. In these experiments, we started sending packets every 0.5 s and decreased this duration down to 0.005 s. As shown in Figure 5.6, the results slightly improved. There was no impact of increasing traffic load below 32 pps and the mean delay was lower than before. However, the peak delay was slightly higher compared to the experiments with varying traffic.

All previous experiments were done based on deployment on a square area. To explore the performance of VCP in an area of different shape, we performed simulations using a network consisting of 200 nodes deployed randomly in a 600 × 120 plane, which is similar to scenarios described in [78] for VRR. The packet rate was set to one or two packets per second, which is equivalent to 200 or 400 CBR flows, respectively. In less than 20 s, all the nodes joined the network. Each node starts sending a 100 byte

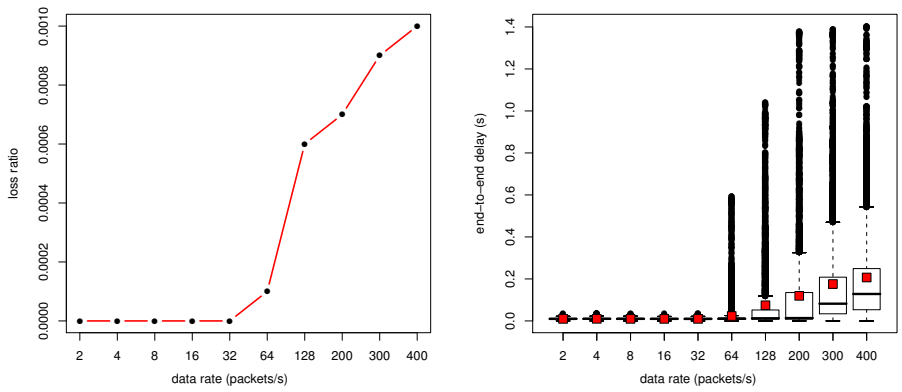

Figure 5.6: Performance evaluation results: effect of increasing CBR traffic rate on loss rate and end-to-end delay

Table 5.5: Influence of the traffic load for random deployment

Measure	1 pps	2 pps
success rate	100 %	99.95 %
average delay	0.006 8 s	0.017 4 s
max delay	0.065 s	0.234 s
average hop count	5.65	5.79
max hop count	16	16

packet to a random destination at a random time in the interval $[50, 230]$ s. All nodes stop transmission at time 950 s. As shown in Table 5.5, the results are very promising. For example, the success rate is almost 100 % – even for high traffic load.

5.3.5 Cord Refinement

To show the possibility of smoothing the cord, we ran an experiment where we enabled the cord refinement mechanism as discussed in Section 4.5.1. In this experiment, we simulated 100 nodes deployed randomly on a rectangular area of 180×180. Running several experiments yield to different position values for the nodes in each run, which consequently makes the difference between with and without refinement difficult to be noticed. Therefore we ran this experiment only for one time (this is the only experiment we ran only for one time). The left-hand side of Figure 5.7 shows a histogram for the distribution of node positions without refinement. The right-hand side of the figure shows the positions distribution after about 120 refinement iterations. It is important to

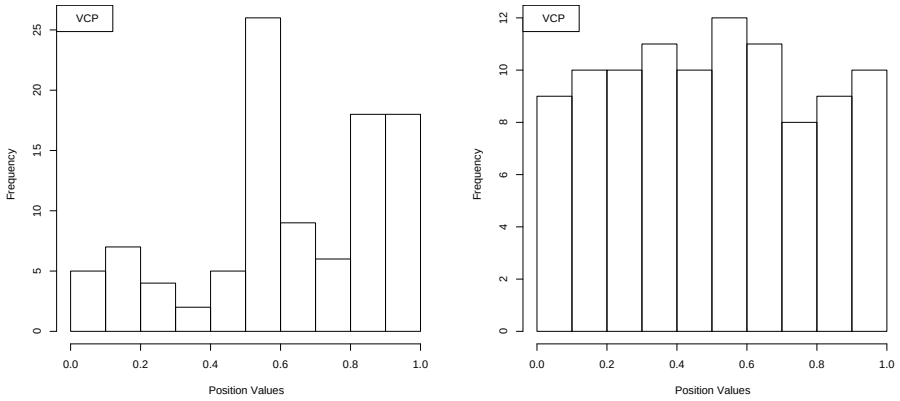

Figure 5.7: Distribution of virtual positions: (left) histogram of the position without refinement, (right) histogram of position with refinement

mention here that the routing performance of VCP does not rely on the smoothness of the Cord. In contrast, routing on similar protocols such as GRWLI [83] depends heavily on the the distribution of the nodes' virtual positions.

5.4 Comparison with Other Protocols

In this section, we present our evaluation of the routing performance of VCP in comparison with other protocols. In particular, we compare VCP to VRR, a competitive approach relying on virtual coordinates for routing, and to DYMO, which is the most recent standard of ad hoc routing protocols as developed by the IETF MANET working group. We rely on the simulation settings as described in Section 5.2. Basically, these settings represent a basis for analyzing the routing performance in sensor networks [78, 89].

5.4.1 Path Length

The first metric we evaluate is the stretch ratio, i.e. the deviation of path length calculated by the different routing protocols to the shortest path. It turned out that in all the scenarios the average measured stretch ratio was in the interval of $[1, 1.25]$. This measure was independent of the network size and density as well as of the deployment topology. Also, VCP and VRR provided almost similar results for this network size. A deviation of 25 % from the shortest path can be considered acceptable especially if the observed end-to-end latency is not being influenced to a large degree. Figure 5.8 depicts

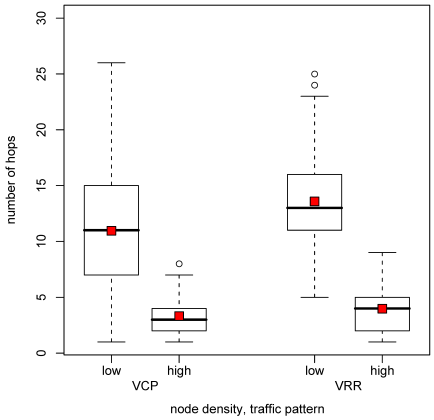

Figure 5.8: Path lengths of VCP and VRR for low and high density scenarios

the typical path lengths as observed during the simulation runs for the low and the high density scenarios. As can be seen, the path length used by VRR is slightly higher compared to the VCP paths.

5.4.2 Delay

We compared the end-to-end delay as observed by the application. In a number of different experiments, we analyzed the protocol behavior for different network densities and traffic rates. Also, we evaluated the influence of the network topology, i.e. grid or random placement of nodes. The results for the grid scenario are depicted in Figure 5.9, and for the random scenario in Figure 5.10. Because of the proactive implementation of VCP as well as VRR, both protocols incur low delays in comparison with the reactive protocol DYMO. For better comparability, we normalized the latency to the path length, i.e. to a per hop delay. As can be seen, the per hop delay for VCP and VRR is quite similar. Both the mean and the median are at about 1 ms. Some outliers can be observed up to about 10 ms. Both protocols are very robust with respect to the network density and the traffic load. Differently, the MANET routing protocol DYMO performed slightly worse for higher traffic load (depicted as 1 s traffic pattern). For lower traffic rates, the observed delay increases largely. This effect can be explained by the route timeouts used by DYMO in our experiment. In the ten seconds example, DYMO has to set up a new route for almost every packet because the available routes have timed out. Thus, each time an additional route setup delay adds to the transmission delay.

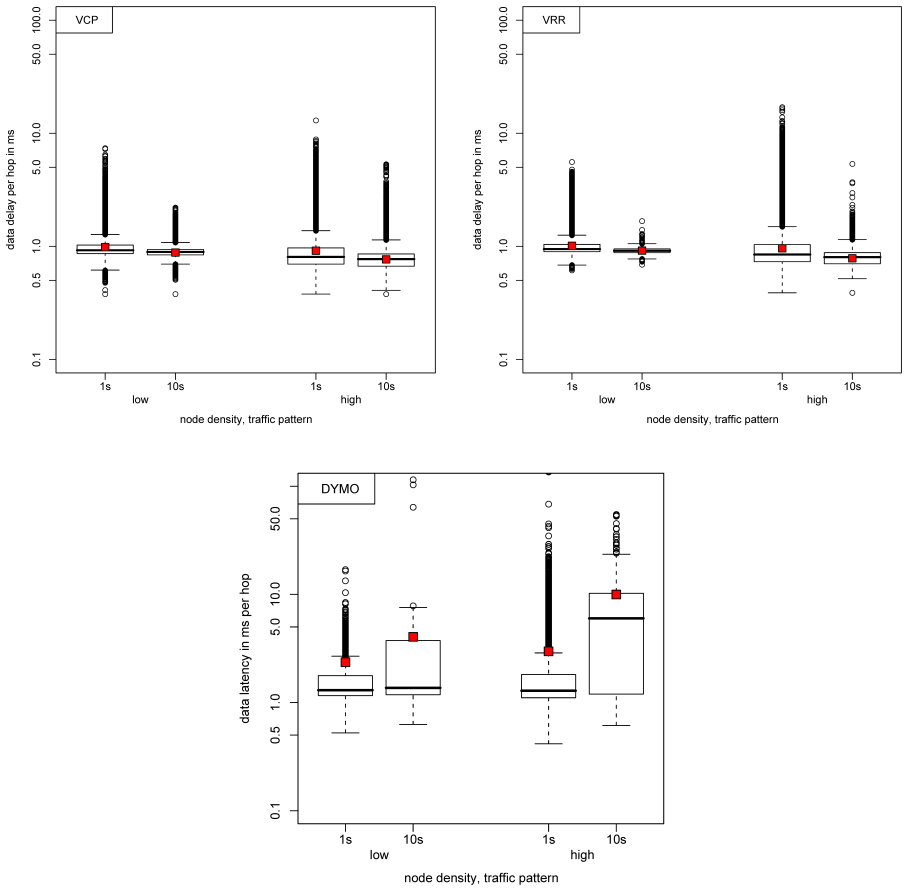

Figure 5.9: Delay performance of VCP, VRR, and DYMO in the grid scenario: depicted is the latency as observed by the application normalized to the path length; all figures are plotted using a log scale y-axis

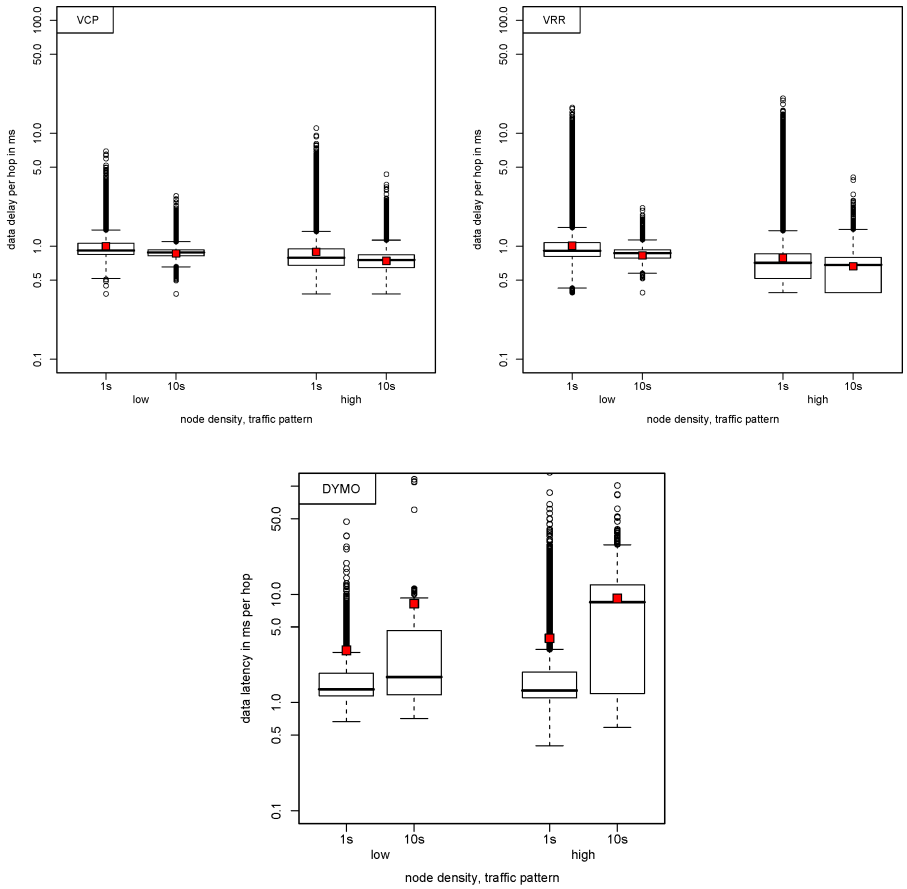

Figure 5.10: Delay performance of VCP, VRR, and DYMO in the random scenario: depicted is the latency as observed by the application normalized to the path length; all figures are plotted using a log scale y-axis

5.4.3 MAC Layer Collisions

For the analysis of routing protocols designed for wireless networks, a main measure to observe is the number of MAC layer collisions. This metric helps to evaluate the load introduced by a routing protocol on the lower layers. Figure 5.11 shows the results for the grid scenario. The results for random deployment, as shown in Figure 5.12, show a higher variance but a similar trend. In particular, the number of MAC collisions is almost zero for the virtual address based routing protocols (for the high density scenario, VRR shows about 5 % collisions per data packet sent, which is negligible). However, the number of collisions is quite high for DYMO. This is not an effect of collisions among data messages but of the periodic flooding of path setup messages. With decreasing data rates, the ratio of the messages required to setup paths to normal data messages increases.

5.4.4 Network Load

Figure 5.13 shows the total number of transmitted messages during the simulation. From this figure, we can infer two important pieces of information. The first one concerns the communication overhead of the protocol, while the second one is related to the routing path. For this experiment we used the low packet rate scenario where each node transmits at a data rate of 0.1 pps. The nodes start transmitting at random times in the interval $[0 - 18]$ s and stop sending after 90 seconds. Thus each node generates about 8 packets on average. The network size is 100 nodes, therefore the nodes will create about 800 data messages during one replication. Figure 5.13 shows that the average number of messages transmitted in the low density network is about 8000 (i.e. about 10×800).

The explanation for this number is that in the low density scenario 100 nodes are deployed in a 400 m $\times$ 400 m plane. Hence the average shortest path is 9 hops which means that in an optimal routing protocol each message would be transmitted 9 hops on average to reach the destination, i.e. the upper left node. In case of the dense network (i.e. 100 nodes deployed on a field of 180 m $\times$ 180 m) the optimal average shortest path is 4 hops. This explains why the number of messages in the dense network is about half the number of messages compared to the low density network. Thus we can conclude that VCP does not create extra routing overhead and the number of messages needed to

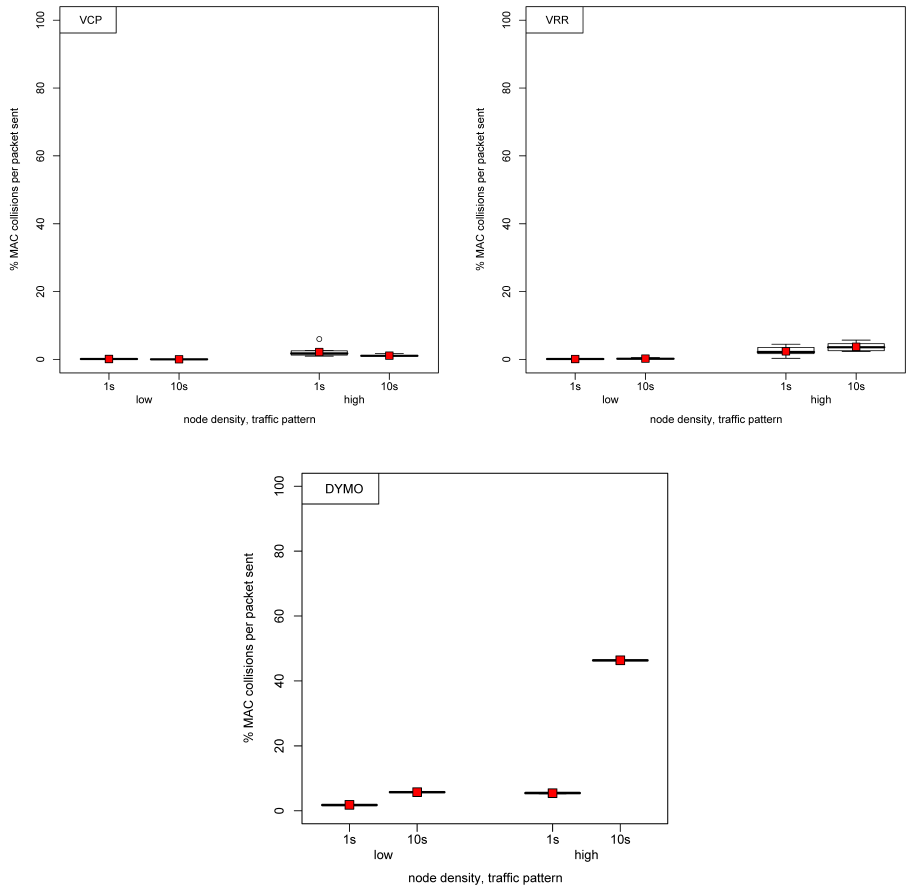

Figure 5.11: MAC layer collisions at the sink per data packet sent for VCP, VRR, and DYMO in the grid scenario

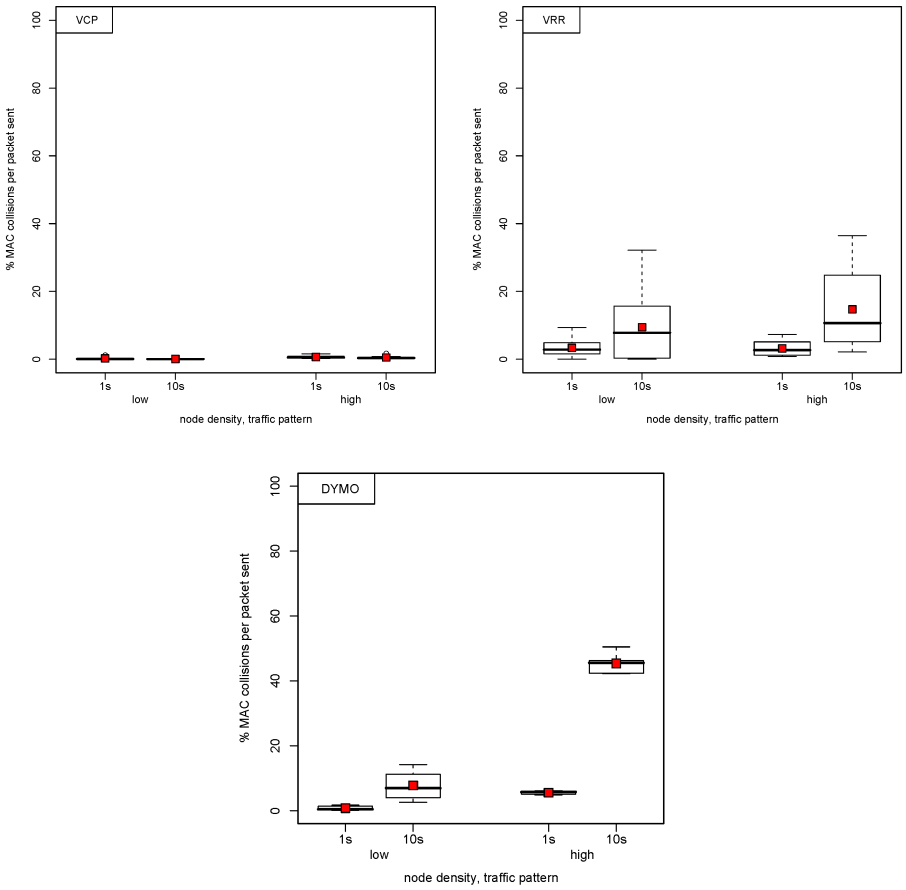

Figure 5.12: MAC layer collisions at the sink per data packet sent for VCP, VRR, and DYMO in the random scenario

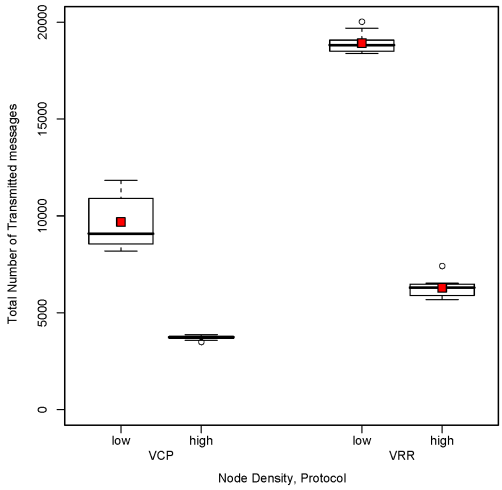

Figure 5.13: Communication overhead: total number of messages transmitted during the simulation

build the cord is negligible. The second conclusion confirms that VCP routing paths are near optimal. Concerning VRR, although there was only a small difference in the path length there is a clear difference in the number of transmitted messages. The reason for this increase is the number of required messages for a node to join the network. In the high density scenario there are about 2500 more messages than with VCP which means that on average each node transmits 25 messages to join the network. This number increases with the average path between nodes. In the low density network the number of joining packets is about 100 packets per node.

5.5 Failure Performance

In the next set of experiments, we focused on the protocol behavior in presence of frequent node failures. As a node failure, we consider any event that prevents communication to a particular node at a given time, e.g. complete energy outages and node replacement, or interrupted communications due to changes in the radio propagation. For these experiments, we only consider VCP and VRR because the network load (and therefore the collision probability) increases too much for MANET protocols such as DYMO. The general setup is the same as for comparative performance in the previous section. We follow the simulation setup described in [55]. In particular,

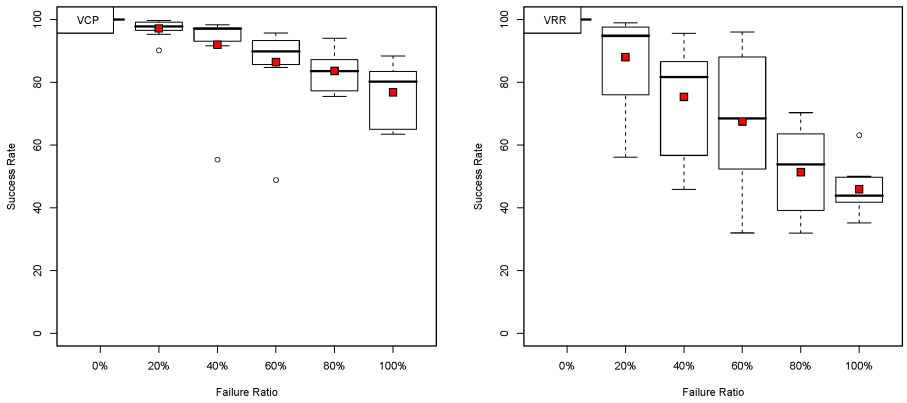

Figure 5.14: Failure performance: the plots show the success rates for VCP and VRR in the grid scenario

the failing of nodes is modeled as a uniformly distributed on/off process. We increase the number of nodes toggling their state from 0 % to 100 %. When a node is selected as unreliable, it remains up for a period selected uniformly at random in $[0, 120]$ s, then goes down for another period selected uniformly in $[0, 60]$ s.

5.5.1 Success Rate with Node Failures

First, we investigated the success rate, i.e. the number of transmissions that were completed successfully. Figure 5.14 depicts the measured success rate for VCP and VRR. As can be seen, the ratio of successful transmissions degrades with the number of failing nodes. However, VCP still maintains a success rate of about 70 %–80 %. In contrast, the success rate degrades much faster for VRR, down to 50 % in the worst case.

The main source of packet loss in VCP is the wrong assessment of the status of radio range neighbors. For example, if a node dies immediately after transmitting a `hello` message, this node will be considered live in the next 4 seconds (the time that should elapse before removing a node from the routing table). As we do not acknowledge the packets, the neighbors can send data packets to this node during this time without the data packets being received. Acknowledgment on the network layer level can reduce this problem. However we did not use network layer acknowledgments because they can cause extra traffic and delay with little benefit. Some routing protocols in the literature, [40, 93] check the MAC layer queue to retransmit packets to a different

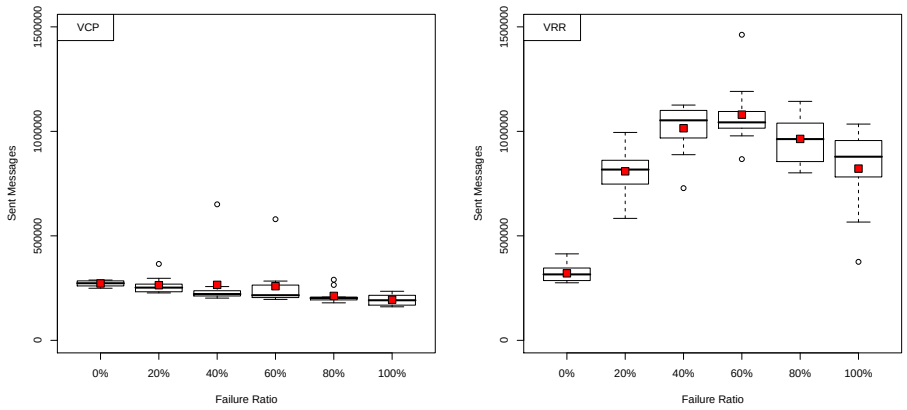

Figure 5.15: Failure performance: the plots show the total number of sent messages for VCP and VRR in the grid scenario

node. However, to keep VCP independent from the MAC layer, we did not use this trick. VRR has a similar problem in updating the physical neighbor states. In addition, VRR requires extra routing packets to update its virtual neighbors. We investigate the problem of communication overhead with node failures further in the next section.

5.5.2 Communication Overhead with Node Failures

We counted the total number of sent messages, i.e. the number of transmissions in the network, without `hello` messages during each simulation experiment. Figure 5.15 depicts the total number of messages sent for VCP and VRR, respectively. In a static network where the node failure ratio is 0, the total number of messages for VRR is larger than for VCP. As discussed before, this is due to nodes in VRR requiring more packets for joining and due to the larger stretch ratio of VRR. When a fraction of the nodes alternates between operation and failure, the number of transmitted messages in VRR increases dramatically, whereas the number decreases for VCP. The explanation for this is related to the mechanism for updating the routing tables in both routing protocols.

In VCP, `hello` messages are enough to update the routing table. Although length of the routing paths will be increased with node failure, there will be fewer nodes generating messages to be sent to the base station as the fraction of nodes increases. Consequently the total number of messages in VCP will decrease. On the other hand, in

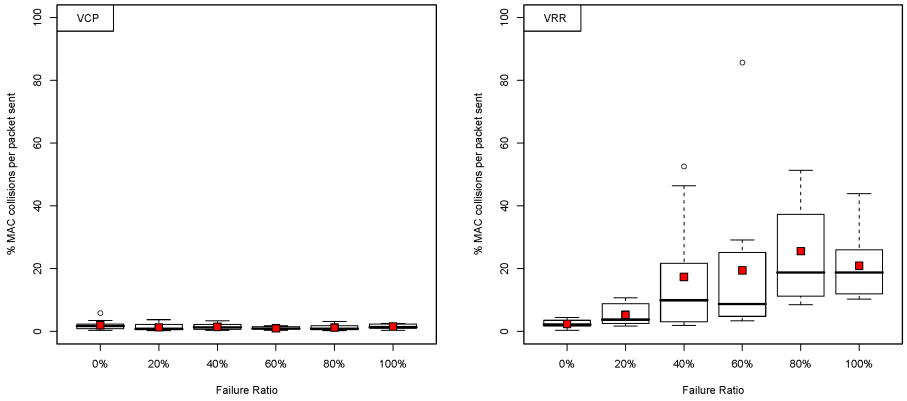

Figure 5.16: Failure performance: the plots show the MAC layer collisions for VCP and VRR in the grid scenario

VRR, the `hello` messages can only partially update the routing table (only the physical neighbors set). Because the virtual neighbors are often far away updating them incurs a high communication overhead.

5.5.3 Collision with Node Failures

A look at the network load reveals some effects of the extra messages of VRR on the lower layer. These effects also explain the reduced success rate of VRR compared to VCP. Figure 5.16 shows the number of MAC layer collisions. As can be seen, there are almost no collisions for VCP, which outlines the capability of this protocol to work even in extreme failure situations. On the other hand, VRR needs many state maintenance operations that lead to increased network congestion.

In addition to the problem of updating the routing table in VRR, there is another problem facing VRR when nodes are unreliable. When VRR enters the transmission phase after its initial join phase, it simply forwards packets to the node that has the ID closest to the packet ID. It needs to be noted that “closest” ID means the ID that is closest on the virtual ring. When the network size increases and many nodes fail periodically, the node’s forwarding table becomes incomplete and represents only a local view of the whole network. VRR has two different strategies to handle such failure situations: exact repair and local “$vset$-path” repair. The idea is to bypass the failed node. However, this technique works only for a few node failures.

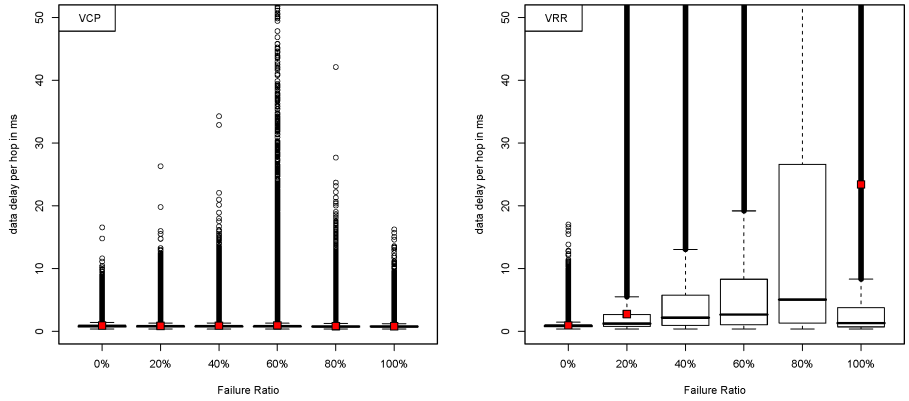

Figure 5.17: Failure performance: the plots show the delay for VCP and VRR in the grid scenario

5.5.4 Delay with Node Failures

The congestion not only reduces the success rate, but also, increases the delay in the network. Packets spend more time in the interface queues because of collision avoidance and packet retransmissions at the MAC layer. Figure 5.17 shows that the median per hop delay of VCP is not affected by the failing nodes. Differently, the delay of VRR increases with the failure ratio.

5.5.5 Path Length with Node Failures

The path length outlines the capability of a routing protocol to find short paths even in case of many node failures. Figure 5.18 depicts the simulation results. While the average path length is slightly shorter for VCP, some single outliers correspond to the special cases in which the virtual cord needs to be used for finding an alternative path when greedy fails. In some cases, such as when all nodes are unreliable, VRR has lower path lengths. The explanation for this lies in the fact that VCP has a higher success rate, which means it receives more packets than VRR. The source of these packets can be far way from the final destination, hence these packets can encounter long paths.

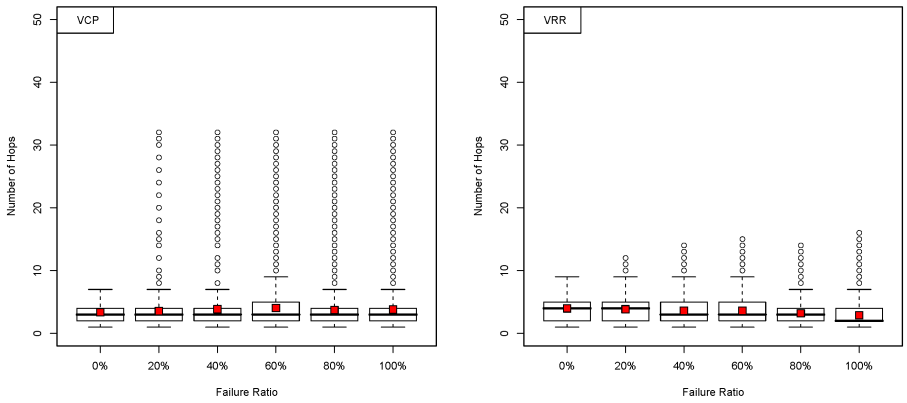

Figure 5.18: Failure performance: the plots show the path length for VCP and VRR in the grid scenario

5.6 Replication Performance

To explore the effectiveness of replicating data in VCP, we measured the availability of stored data to queries as well as the communication load placed on the nodes when a fraction of the nodes fails. To measure the availability we used the query success rate, i.e the fraction of queries that were answered correctly. We measure the communication load on nodes by counting the total number of data refresh operations during the simulation experiment as well as the number of messages transmitted during the query period. We used the same simulation parameters as shown in Table 5.4, which are similar to the scenarios used in the GHT paper [53]. Thus the simulation setup consists of 100 nodes deployed in rectangular area of size 160 m × 160 m. We placed a sink node on the upper left corner of the simulation area. After inserting 20 keys into the network, the sink starts to generate queries. If a query is not answered within one second, the sink will retransmit the query for up to 7 times. The sink generates 2 queries per second including both new and retransmitted queries. In the simulation we investigated the two local replication schemes as well as the global replication described in Section 4.4.

5.6.1 Local Replication on Neighbors

At the beginning we investigate the scalability of VCP to store and retrieve data. We varied the number of nodes from 50 to 400 and we scaled the playground to keep a constant density. As one would expect, VCP offers perfect availability of stored data

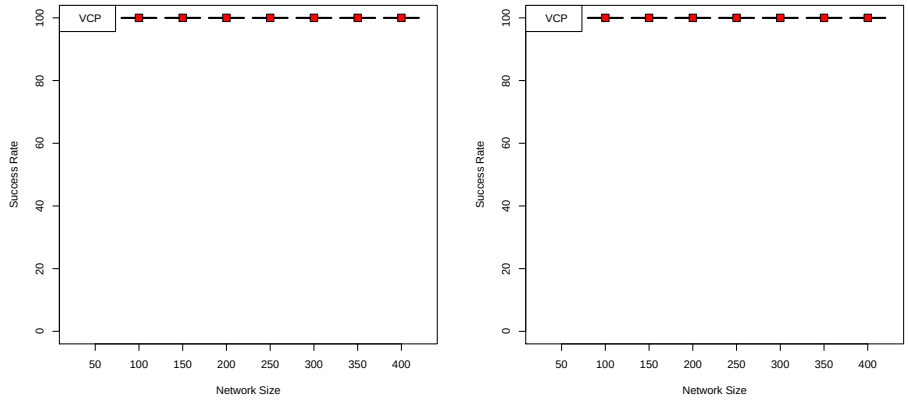

Figure 5.19: Success Ratio for Insertion (upper-left), Retrieval (upper-right) with increasing network size

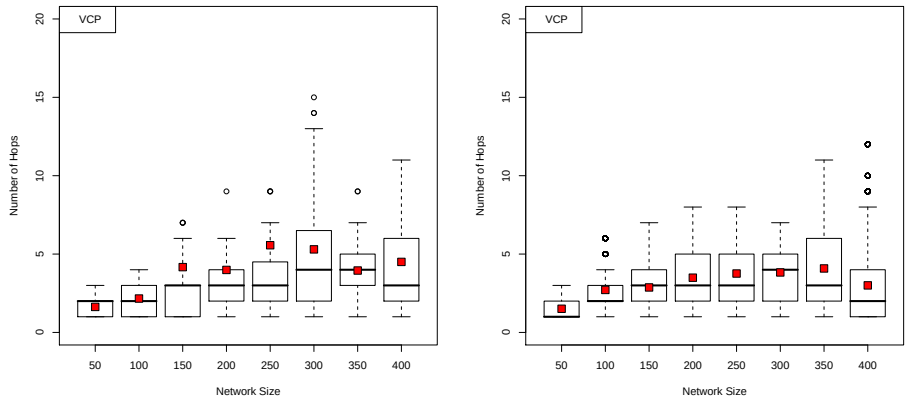

Figure 5.20: Path lengths for insertion (left) and retrieval (right) with increasing network size

items on static networks where the topology does not change. Figure 5.19 shows that at all network scales, all data items were inserted to the correct destination and all queries were answered without retransmission of the request. The number of refresh messages was zero because there is no node failure.

The paths traversed by the queries are slightly shorter than the data insertion paths as can be seen from Figure 5.20. The reason behind this decrease in the path length lies on the replication scheme. It is highly probable that queries will reach a node holding a copy of the requested data before reaching the node responsible for that data item.

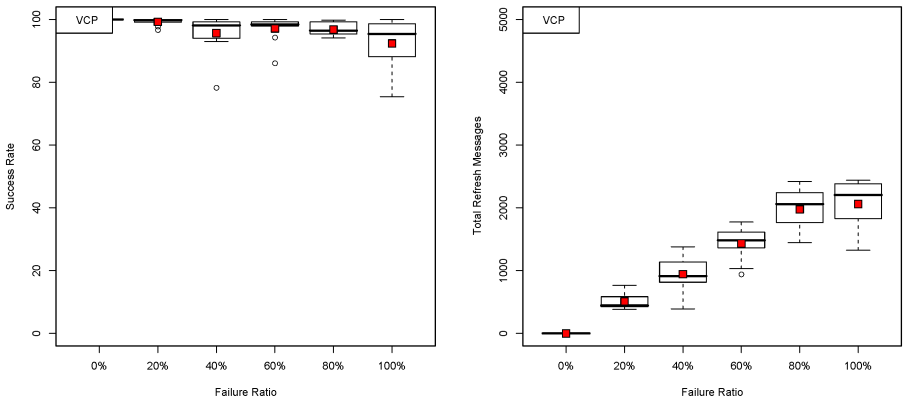

Figure 5.21: Replication on all physical neighbors: query success ratio and number of data refresh messages

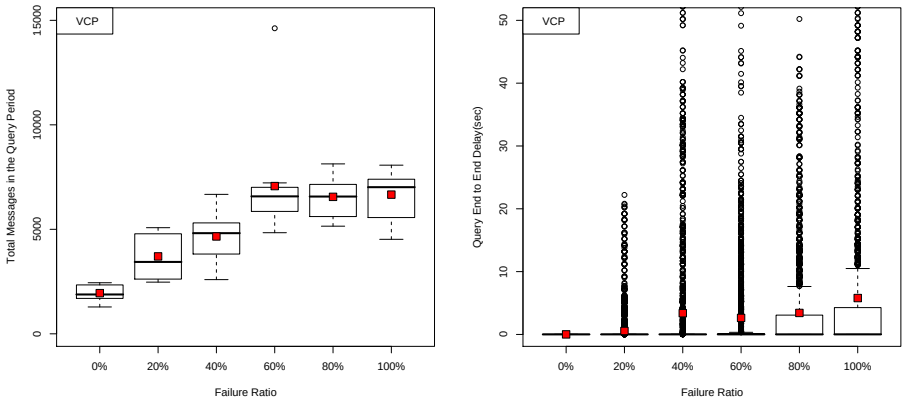

Figure 5.22: Replication on all physical neighbors: total number of transmitted messages in query period and end-to-end delay with increasing fraction of failing nodes in random scenario

We said “highly probable” since the data copies were sent using broadcasts without acknowledgments at the MAC layer. Consequently, in real world scenarios, some nodes (especially the far away nodes) may not receive the packet.

Figure 5.21 depicts the success ratio and total number of refresh messages for the random topology. As one would expect, the success ratio decreases as the number of failing nodes increases. Nevertheless, VCP maintains a good success ratio even when all the nodes alternate between available and unavailable states. Figure 5.22

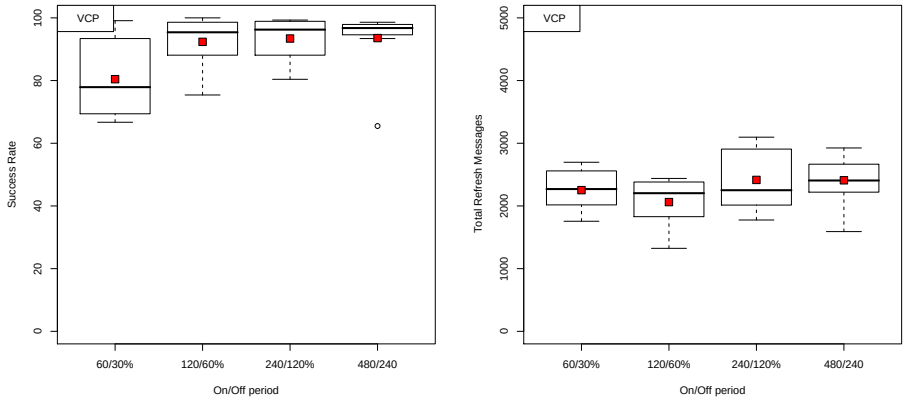

Figure 5.23: Performance of replication: query success rate and number of refresh messages, all nodes alternate between on and off state of varied length

depicts the number of messages used in retrieving the data as well as the querying delay. This success ratio is only a little higher than the success ratio achievable with GHT. Nevertheless, VCP has several advantages. First, nodes do not need to be aware of their physical location. Second, there is no assumption about the node connectivity. Finally, the refresh messages in VCP are generated on-demand, i.e if there is no node failure then there is no need for refresh messages. In contrast, the refresh messages in GHT are sent periodically and independently from node failures.

We also investigated the effect of on/off intervals. In these experiments, all nodes are flipping between on and off states. Figure 5.23 shows the performance of VCP with different on/off time intervals. The simulation experiments lasts five times the length of the down time interval. When nodes flip between on and off states more frequently, the ability of local nodes to hold data items is stressed heavily. Thus, the success rate for short on/off intervals is decreased. For longer on/off intervals, the success rate increases and reaches a good value at long on/off intervals. Notice that the number of refresh messages stays almost constant for all on/off intervals.

5.6.2 Local Replication on Adjacent Nodes

The effectiveness of data replication on radio range neighbors depends on the node degree, i.e. the number of direct neighbors. Therefore, the availability of data items can be affected negatively in a sparse network. We investigated the replication on adjacent

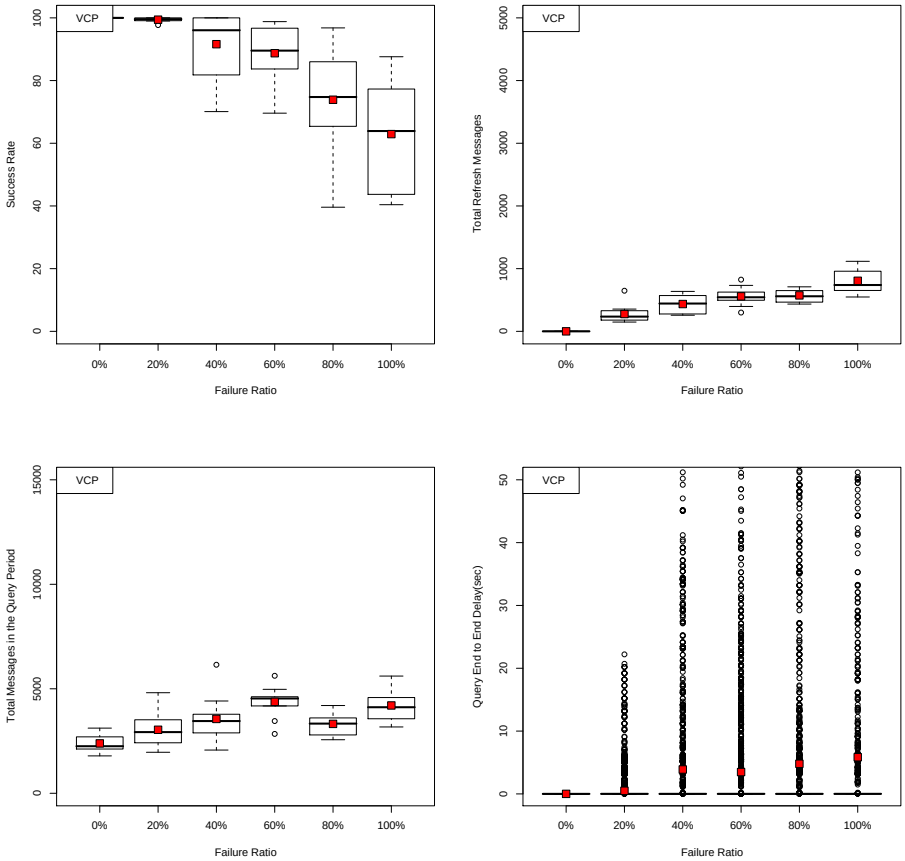

Figure 5.24: Replication on two adjacent nodes: query success ratio, number of data refresh messages, number of total transmitted messages in query period and end-to-end delay with increasing fraction of failing nodes in random scenario

nodes and the effect of the number of copies. Figures 5.24, 5.25, and 5.26 illustrate the effect of the number of copies on the success ratio, the required number of refresh messages and the number of messages used in the query period. As it is expected, the higher the number of copies the better the success rate. The success rate figures also indicate that all replication schemes perform similarly and provide a high success rate when a small number of nodes fail (20 %). In contrast, when a high number of nodes are failing, the success rate decreases significantly.

There is a clear relation between the number of copies and the number of refresh messages. The figures show a slight increase in the total number of messages used in the query period, which actually comes from the refresh messages and not from the

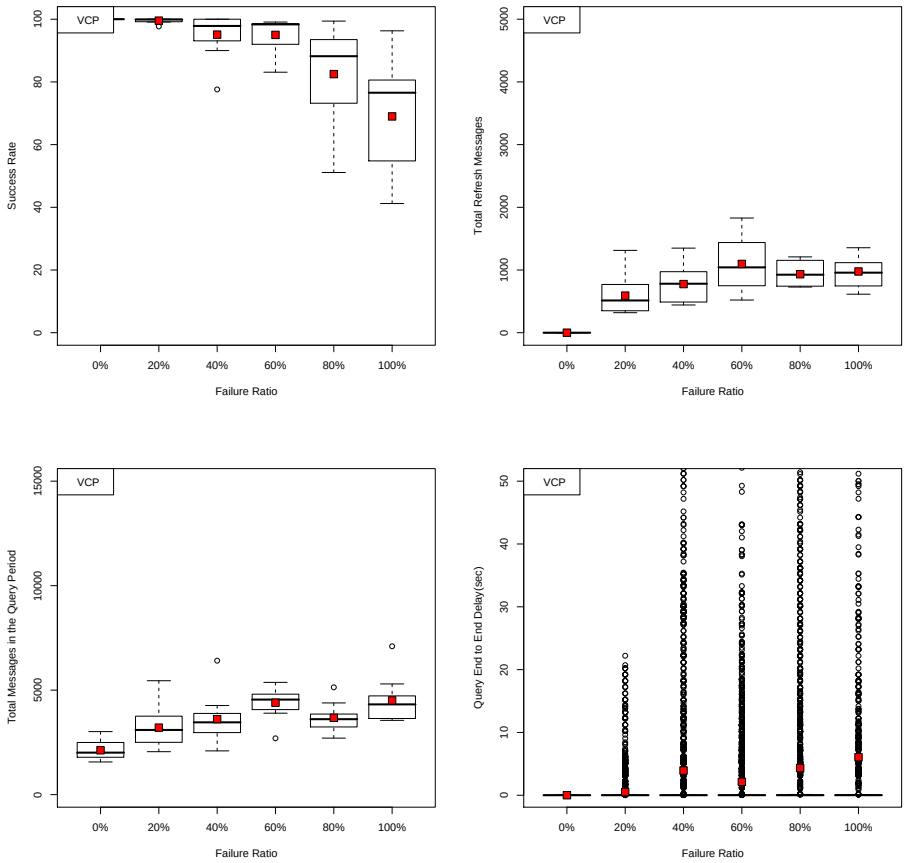

Figure 5.25: Replication on four adjacent nodes: query success ratio, number of data refresh messages, number of total transmitted messages in query period and end-to-end delay with increasing fraction of failing nodes in random scenario

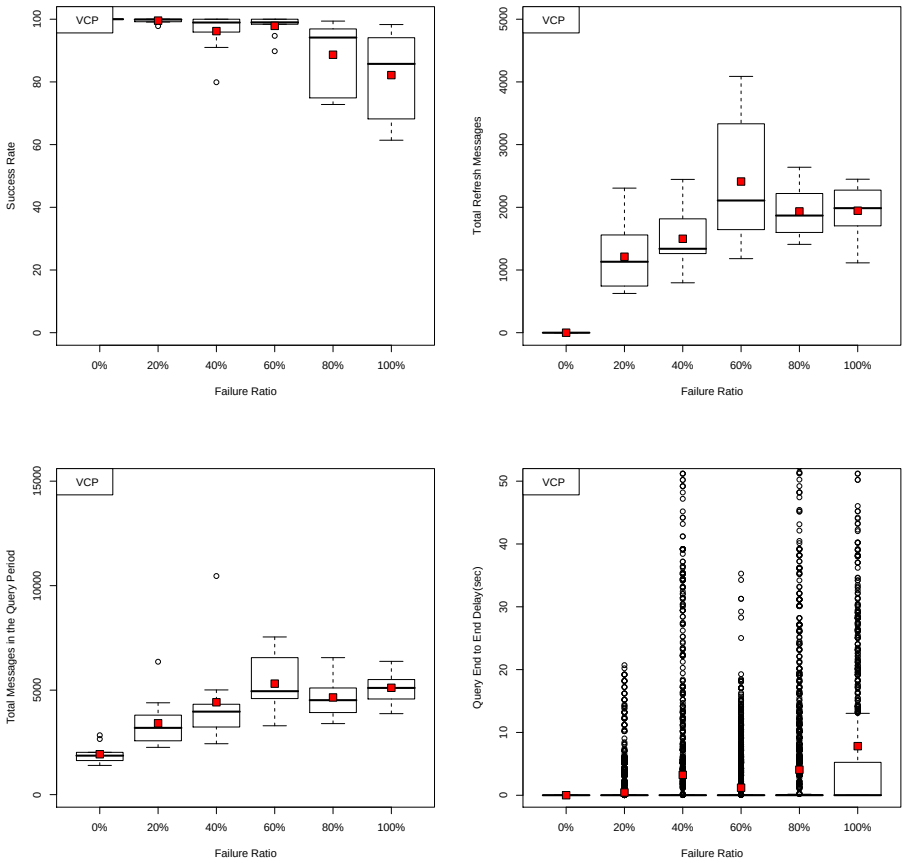

Figure 5.26: Replication on eight adjacent nodes: query success ratio, number of data refresh messages, number of total transmitted messages in query period and end-to-end delay with increasing fraction of failing nodes in random scenario

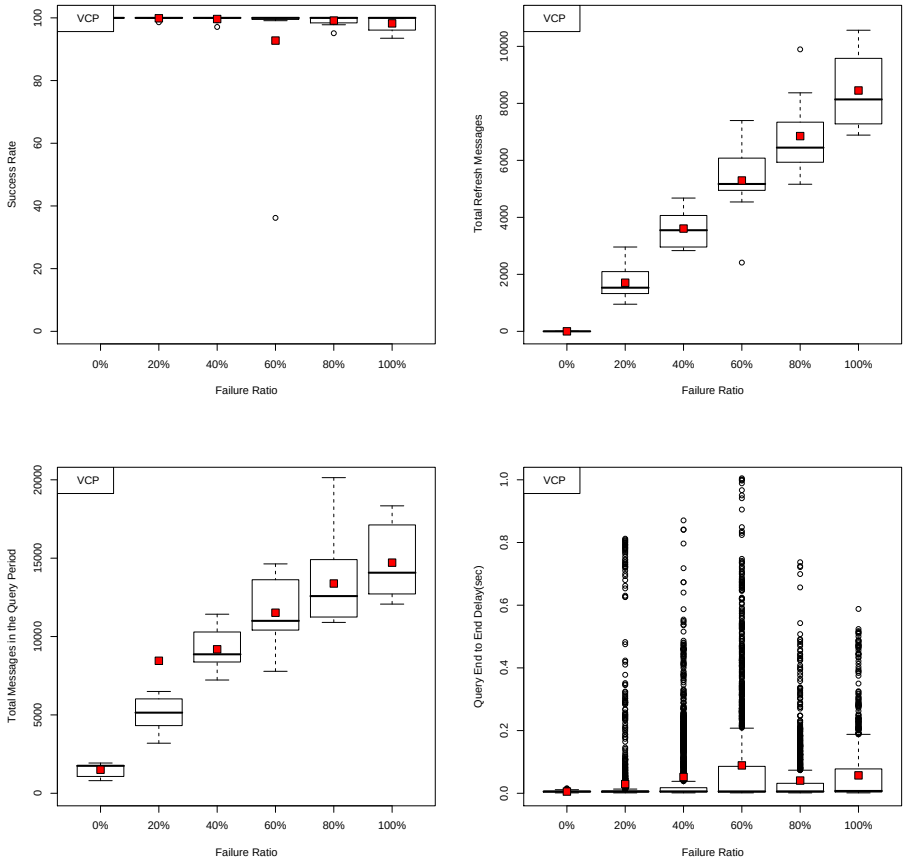

Figure 5.27: Global Replication: query success ratio and data refresh messages with increasing fraction of failing nodes in random scenario

messages needed to reach the data items. Finally, the end-to-end delay mostly results from the retransmission of requests.

5.6.3 Global Replication

In extreme scenarios a massive node failure can occur in the same region. Therefore, we examine here the effectiveness of storing data items at dispersed locations on the cord. We stored each data item on five different locations. In each location we also replicate the data items on the physical neighbors. The reason behind replicating the data locally is to save multi-hop data refreshing as discussed in Section 4.4. Figure 5.27 shows the success ratio and number of refresh messages. On one side, the success ratio becomes close to the 100 % even when all nodes are unreliable. On the other side,

the communication cost increases linearly with the number of storage locations. The end-to-end delay is low compared to local replication because we used the parallel querying method.

5.7 Summary

In this chapter we have intensively evaluated VCP using simulation. A detailed model of IEEE 802.11 radios has been used throughout the simulation experiments. We investigated different parameters over a large set of scenarios. We also compared VCP with competitive state of the art protocols. We have performed ten replications for most of the experiments and visualized the results using box plots. VCP has exhibited a robust performance throughout the experiments, i.e a low overhead, high scalability, near optimal routing paths, and a high capability to adapt to topology changes. We investigated the ability of different local replication algorithms to persist data in the presence of unreliable nodes. The results show that these maintain good persistence of the data in the network. We also investigated a more expensive algorithm that generates copies of data items on multiple places on the cord. This method shows a higher query success rate, but also incurs more communication overhead. The preferred replication algorithm thus depends on the intended application.

Chapter 6

Implementation in a Lab Environment

In order to show that VCP is feasible not only in simulation, but also in real world, we set up a proof-of-concept experimental setup in our lab. We implemented VCP on sensor node called BTnode[1] depicted in Figure 6.1. Because this sensor node does not contain any sensor, we implemented a sensor board that contains sensors measuring light and temperature, a motion detector and a buzzer. Moreover we implemented a switch to be used as an actuator. During the developing of VCP on the sensor nodes we made use of pimoto [94, 95] for the debugging and monitoring the transmitted packets.

6.1 The BTnode Sensor Node

The BTnode sensor node is is an autonomous wireless communication and computing platform based on a Bluetooth, low-power radio and a microcontroller. Both radios can be operated simultaneously or be independently powered off completely when not in use, considerably reducing the idle power consumption of the device. The low-power radio is the same as used on the Berkeley Mica2 Motes. It had been jointly developed at the ETH Zurich by the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems. It serves as a demonstration and prototyping platform for research in MANETs and WSNs.

The characteristics of BTnode are listed below:

- Microcontroller: Atmel ATmega 128L (8 MHz @ 8 MIPS)
- Memories: 64+180 Kbyte RAM, 128 Kbyte FLASH ROM, 4 Kbyte EEPROM

[1] `http://BTnode.ethz.ch/`

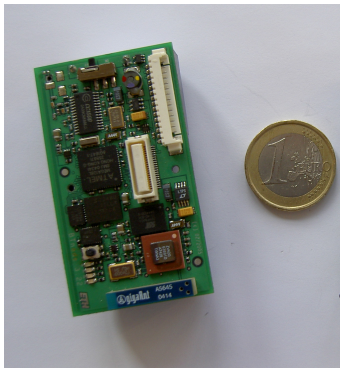

Figure 6.1: The BTnode wireless sensor node [2]

- Bluetooth subsystem: Zeevo ZV4002, supporting AFH/SFH
- Low-power radio: Chipcon CC1000 operating in ISM band 433-915 MHz
- External Interfaces: ISP, UART, SPI, I2C, GPIO, ADC, Timer, 4 LEDs
- Software: BTnut System Software, Standard C Programming, TinyOS compatible

6.2 Sensors and Actuator Boards

While the BTnode had been designed for conducting research in sensor networks, it does not carrying any onboard sensors. Therefore, in order to enable for sensing capabilities, the BTnode team designed the BTsense, which is a sensor board that can be attached by the J2 connector to the BTnode. However this Board is not available to external users. Therefore we built our own board. We used the same design but we used the J3 connector on the USB programmer board. In addition to the BTsense board, we also built an actuator.

Figure 6.2 shows a sensor board (left) and an actuator (right), which we developed in our lab for testing and demonstration purposes. They can be attached to the BTnode. The sensor board contains temperature and light sensors as well as a motion detector. The actuator (switch) contains a relay that can be biased by a sensor node to switch on/off high-voltage (220V) devices.

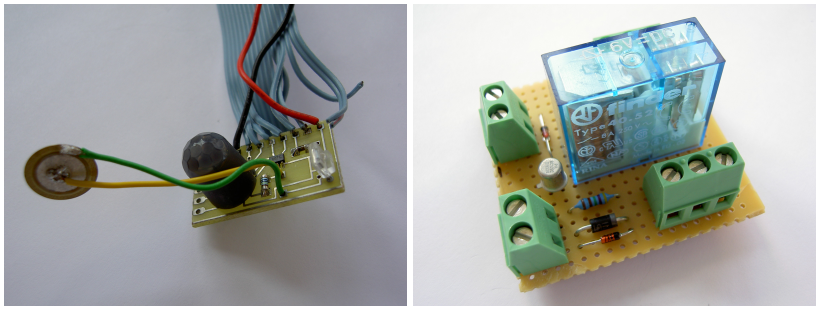

Figure 6.2: Left: Motion, temperature, and light sensors and a buzzer are mounted on a connector board; Right: a relay is used as an actuator and can be used together with the BTnode sensor node

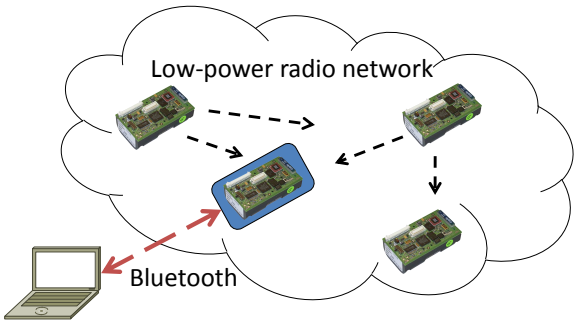

Figure 6.3: Basic architecture of Pimoto

6.3 Pimoto

Pimoto [94, 95] is a tool to passively monitor sensor networks. Its primary objective is to intercept radio data packets in a completely passive way. Additionally, it supports distributed monitoring in hierarchical way. Pimoto visualizes the packets collected by several monitoring nodes, which transmitted the monitoring data over a hierarchically operating system to a central server for further analysis.

In particular, the packets are sniffed at MAC level by sensor nodes that employ a second radio channel, i.e. a Bluetooth interface, for communication with a gateway PC. This gateway in turn sends the received data to the server. Pimoto uses the concept of “monitoring islands”, which has the advantage of monitoring several networks simultaneously by multiple monitoring nodes and, perhaps, multiple gateways. At the server, a Wireshark plugin for protocol analysis is used. This tool for the visualization and analysis of packet contents.

Field	Meaning
Source address (2 Byte)	BMAC source address
Destination address (2 Byte)	BMAC destination address
Length of Data (2 Byte)	Length of the data
Type (1 Byte)	Application type
Data (length of data)	Packet data

Table 6.1: BMAC data format

6.4 Implementation

We implemented VCP on BTnode running BTnut operating system. BTnut was developed based on the open source Nut/OS. Similar to TinyOS, it is a real-time embedded operating system compatible with ATmega processor family. In addition to minimal OS services, it provides a multithreaded system architecture. We used the multithread capability to schedule tasks for `hello`, `insert`, `query` messages.

The implementation is written in standard C and it is configured to use double for Position, and a `hello` period of 1 second. The employed BTnode sensor nodes primarily use the BMAC protocol. Table 6.1 shows the original packets format used by BMAC.

Because sensor nodes may offer only broadcast as a communication paradigm [1], we tested at the beginning the performance of VCP using broadcast mode and relying on the VCP position to identify nodes. Then we investigated the performance of VCP when putting the interface in promiscuous mode, because we observed that the monitoring nodes in Pimoto capture more packets than the destination node of that packet.

The size of the compiled system to be installed on the BTnode is about 87 kByte. The file contains both the text and the data sections. The text portion contains the actual instructions, while the data contains the program's data part. The resulting file size is reasonable for sensor nodes of this class.

To simplify the administration of the network we implemented some functions and registered them to be called from the terminal:

- `printN` displays some debugging information information about the node (like its virtual position).
- `setpos 0` is used to initialize the first node.
- `sink 1` assigns the current node as a sink.

6.5 Prototype and Demo

Based on our implementation, we prepared a demo setup to show the functionality of VCP in a real-world scenario [96]. Certainly, this is more a proof-of-concept than a large-scale deployment.

Initially each node should have a unique local identifier. We used the first 3 bytes of the Bluetooth address to identify the nodes (node ID). Other approaches can be used such as random numbers. All nodes were initialized with the relative position with a the negative value of the node ID. We used this value to communicate with the nodes before joining the network. We connected one node to the PC using a USB cable. This node acts as a sink node. Using a terminal we set the virtual position of this node to the Start value (0). This triggers the node to start broadcasting its position (`hello` messages). The `hello` messages enable all nodes to eventually get a virtual position on the cord.

By default all nodes are named as node. We used the terminal to change one node's name to `Sink`. Using a hash function, this information is inserted in the cord's data store and can be used to retrieve the sink's virtual position. After a short initial period, the same hash function is used to identify the sink's position by all the other nodes. Then, all the nodes start sending one data packet per second to the sink.

As one would expect, when the promiscuous modes is enabled the sink receives more packets than when just using the broadcast mode. Nevertheless this different is not that big. The success ratio is increased from about $60-78\%$ when using broadcast mode to $65-90\%$ when enabling promiscuous mode. The main reason for packet loss lies on the fact that we did not considered the quality of the link when transmitting packets between nodes. Greedy forwarding in VCP selects the node that make maximum progress toward destination without considering the link-quality to that node. Therefore we suggest to include link-quality (i.e signal strength of the received packets) in the routing table (radio range neighbors) and restricting data transmission to nodes with minimum link properties. This criterion can incur longer paths, however it improves the delivery ratio. Another obvious method to improve the delivery ratio is to acknowledge data packets. However this will incur extra communication overhead on the network.

For demonstration purposes, we used the received data packets on the sink to visualize the established cord and its dynamic updates as shown in Figure 6.4. We

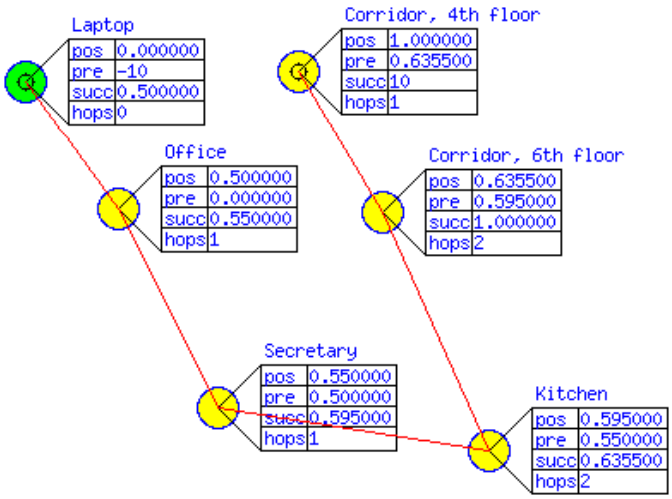

Figure 6.4: Cord visualization in the lab demo

deployed six nodes in the lab and in the corridor. We stored the IDs of the sensor nodes and there location in a text file. Hence, when a packet is received at the sink node, we would be able to know its original location. Each packet has a hop-counter as well as the value of successor and predecessor positions of the originating node. It can happen that different packets from the same node have different hop counts. The reason for this is the that the `hello` messages are not reliable and they can be easily lost, especially from nodes at the edges of transmission range. Thus a node that considered live at one time, can be be considered unavailable in another time because there was no `hello` messages form that node. In Figure 6.4 the yellow circles are normal nodes, the green is a sink, the double circle indicates an end of the cord.

6.6 Summary

In this chapter we proved that VCP can be implemented on real sensor nodes. From the demonstration, it can be seen that all nodes obtained a virtual position. Moreover all Sink position queries were answered correctly. However we noticed that the network experienced packet loses at the mac-layer level. Nevertheless, we can say in spite of the poor capabilities of the MAC-layer on BTnodes, VCP performance was acceptable.

Chapter 7

Conclusion

Wireless Sensor Networks (WSNs) represent an emerging technology that has the ability to bring the physical real world closer to the user. However, it also poses big challenges that must be solved to build functional systems. The challenges can be divided into two main related categories: miniaturization of node hardware and development of efficient data management solutions. In this thesis, the latter aspect has been addressed.

We have presented the Virtual Cord Protocol (VCP), a light-weight protocol that includes routing functionality as well as DHT-based data management functions. VCP uses a one-dimensional virtual cord to keep routing tables small and to provide low-complexity but robust packet delivery in sensor networks independently from the network density. VCP relies on greedy forwarding based on two concepts. First, the virtual cord can be used to find a path to each destination in the network. Secondly, locally available neighborhood information is exploited to find shortcuts towards the destination. Local broadcasting of hello messages is used to update the routing tables. We have shown that VCP is able to find almost optimal paths.

Furthermore, VCP uses a pre-defined range of virtual coordinates, i.e. the cord positions, that can be used to hash different contents such as collected sensor data or even service information such as the positions of base stations. This mapping between content and the node position on the cord enables other nodes to deterministically identify the searched items. In static networks, greedy forwarding guarantees packet delivery to the correct destination as well as a loop free path. In the case of frequent topology changes, greedy forwarding is impossible and VCP tries to find alternative paths in order to reach a region in the network where greedy forwarding can be resumed.

We evaluated VCP in extensive simulation experiments using different network sizes, topologies and node densities. The results demonstrate that VCP shows a robust performance across a wide range of different parameter settings. The results shows a nearly optimal stretch ratio in all investigated scenarios. Furthermore, the success rate was almost 100 %. The only reason for packet loss was congestions on the MAC layer.

We compared VCP with the state-of-the-art protocols DYMO and VRR in different scenarios. We demonstrated that the performance of VCP is consistently good and in most scenarios even better compared the other protocols. Both VCP and VRR clearly outperformed DYMO in static network scenarios. In the experiments with frequent node failures, VCP shows clear advantages over VRR. VCP consistently delivers more that 80 % of all data packets successfully even when all nodes flipping between on and off, whereas the success ratio for VRR decreased to about 40 %. The reason for this decrease is the extra communication overhead generated by VRR to update the $vset$.

In a final set of experiments, we investigated different replication algorithms. We have shown that VCP can be used to replicate data on physical neighbors or on neighbors on the virtual cord. The main advantage of the proposed algorithms is that data replication is done locally and only on demand. Hence there is no need for periodic update. The results shows that replicating on the physical neighbors significantly increases the data availability. In dense networks, even when all nodes are unreliable, the query success rate is still well above 90 %. Because the performance of replicating on physical neighbors depends heavily on the network density, we investigated replication on the cord, i.e. on the succeeding and the preceeding nodes. The query success rate was lower compared to replicating on all physical neighbors. However, when only small fraction (less than 20 %) of the nodes are unreliable, the results were perfect for all replication methods. We also investigated the replication on different places on the cord, which can be required for some cases, for instance when all nodes in the same region may fail due to localized disturbances.

List of Acronyms

ACK	Acknowledgment
AODV	Ad Hoc on Demand Distance Vector
AODV-PA	AODV with Path Accumulation
BMAC	Berkeley MAC
BVR	Beacon Vector Routing
DIFS	Distributed Index for Features in Sensor Networks
DHT	Distributed Hash Table
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
DYMO	Dynamic MANET on Demand
GFG	Greedy Face Greedy
GHT	Geographic Hash Table
GLIDER	Gradient Landmark-Based Distributed Routing
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
GRWLI	Geographic Routing Without Location Information
GSR	Global State Routing
ISM	Industrial, Scientific and Medical

MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
OLSR	Optimized Link State Routing
PRP	Perimeter Refresh Protocol
P2P	Peer-To-Peer
RREQ	Route Request
RREP	Route Reply
RERR	Route Error
RSSI	Received Signal Strength Indicator
SR-DCS	Structured Replication in DCS
TBRPF	Topology Broadcast based on Reverse Path Forwarding
TORA	Temporally-Ordered Routing Algorithm
VCap	Virtual Coordinate assignment protocol
VCP	Virtual Cord Protocol
VRR	Virtual Ring Routing
WSN	Wireless Sensor Network
ZRP	Zone Routing Protocol

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–116, August 2002.

[2] F. Dressler, *Self-Organization in Sensor and Actor Networks*. John Wiley & Sons, December 2007.

[3] A. Boukerche, Ed., *Algorithms and Protocols for Wireless Sensor Networks*. Wiley-IEEE Press, 2008.

[4] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.

[5] L. Q. Zhuang, J. B. Zhang, D. Zhang, and Y. Z. Zhao, "Data management for wireless sensor networks: research issues and challenges," in *International Conference on Control and Automation (ICCA 2005)*, vol. 1, June 2005, pp. 208–213.

[6] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in *5th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 1999)*. Seattle, WA: ACM, August 1999, pp. 263–270.

[7] C. M. Sadler and M. Martonosi, "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks," in *4th ACM Conference on Embedded Networked Sensor Systems (ACM SenSys 2006)*. Boulder, CO: ACM, November 2006, pp. 265–278.

[8] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Compressing historical information in sensor networks," in *ACM SIGMOD International Conference on Management of Data (ACM SIGMOD 2004)*. Paris, France: ACM, 2004, pp. 527–538.

[9] S. S. Pradhan, J. Kusuma, and K. Ramchandran, "Distributed Compression in a Dense Microsensor Network," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 51–60, March 2002.

[10] J. S. Wilson, *Sensor Technology Handbook*. Newnes, 2004.

[11] T. Arampatzis, J. Lygeros, and S. Manesis, "A Survey of Applications of Wireless Sensors and Wireless Sensor Networks," in *13th Mediterrean Conference on Control and Automation*, Limassol, Cyprus, June 2005, pp. 719– 724.

[12] C. Chevallay, R. E. V. Dyck, T. A. Hall, R. E. Van, and D. T. A. Hall, "Self-organization Protocols for Wireless Sensor Networks," in *36th Annual Conference on Information Sciences and Systems (CISS 2002)*, Princeton, NJ, March 2002.

[13] I. Dietrich and F. Dressler, "On the Lifetime of Wireless Sensor Networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 1, pp. 1–39, February 2009.

[14] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *1st ACM Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[15] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks," in *6th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2007), Poster Session*. Cambridge, MA: ACM, April 2007, pp. 254–263.

[16] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring Volcanic Eruptions with a Wireless Sensor Network," in *2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, Istanbul, Turkey, February 2005.

[17] S. Dengler, A. Awad, and F. Dressler, "Sensor/Actuator Networks in Smart Homes for Supporting Elderly and Handicapped People," in *21st IEEE International Conference on Advanced Information Networking and Applications (IEEE AINA-07): 1st IEEE International Workshop on Smart Homes for Tele-Health (SmarTel 2007)*, vol. II. Niagara Falls, Canada: IEEE, May 2007, pp. 863–868.

[18] A. Awad, T. Frunzke, and F. Dressler, "Adaptive Distance Estimation and Localization in WSN using RSSI Measures," in *10th EUROMICRO Conference on Digital*

System Design - Architectures, Methods and Tools (DSD 2007). Lübeck, Germany: IEEE, August 2007, pp. 471–478.

[19] C. S. Raghavendra, K. M. Sivalingam, and T. Znati, *Wireless Sensor Networks*. Kluwer Academic Publishers, 2004.

[20] J. Zhao and R. Govindan, "Understanding Packet Delivery Performance In Dense. Wireless Sensor Network," in *1st ACM Conference on Embedded Networked Sensor Systems (ACM SenSys 2003)*, Los Angeles, CA, November 2003.

[21] M. Mauve and J. Widmer, "A Survey on Position-Based Routing in Mobile Ad-Hoc Networks," *IEEE Network*, vol. 15, no. 6, pp. 30–39, 2001.

[22] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Computer Communications Review*, pp. 234–244, 1994.

[23] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *IEEE International Multi Topic Conference 2001: Technology for the 21st Century (IEEE INMIC 2001)*, Lahore, Pakistan, December 2001, pp. 62–68.

[24] B. Bellur and R. G. Ogier, "A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks," in *18th IEEE Conference on Computer Communications (IEEE INFOCOM 1999)*, vol. 1, New York, NY, March 1999, pp. 178–186.

[25] T.-W. Chen and M. Gerla, "Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks," in *IEEE International Conference on Communications (IEEE ICC 1998)*, Atlanta, GA, June 1998, pp. 171–175.

[26] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*, T. Imielinski and H. F. Korth, Eds. Kluwer Academic Publishers, 1996, vol. 353, pp. 152–181.

[27] C. E. Perkins and E. M. Royer, "Ad hoc On-Demand Distance Vector Routing," in *2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, February 1999, pp. 90–100.

[28] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *16th IEEE Conference on Computer Communications (IEEE INFOCOM 1997)*, 1997, pp. 1405–1413.

[29] V. Park and M. Corson, "A Performance Comparison of the Temporally-Ordered Routing Algorithm and Ideal Link-State Routing," in *3rd IEEE Symposium on Computers and Communications (ISCC 1998)*, Athens, Greece, June/July 1998, pp. 592–598.

[30] R. Bhaskar, J. Herranz, and F. Laguillaumie, "Efficient Authentication for Reactive Routing Protocols," in *20th IEEE International Conference on Advanced Information Networking and Applications (IEEE AINA-06)*, vol. 2. Vienna, Austria: IEEE, April 2006, pp. 57–61.

[31] Z. J. Haas, "A new routing protocol for reconfigurable wireless networks," in *IEEE International Conference on Universal Personal Communications (ICUPC)*, 1997, pp. 562–565.

[32] Z. J. Haas and M. R. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, pp. 427–438, August 2001.

[33] S. Capkun, M. Hamdi, and J.-P. Hubaux, "GPS-Free Positioning in Mobile ad-hoc Networks," in *34th Annual Hawaii International Conference on System Sciences (HICSS 2001)*, Big Island, Hawaii, January 2001.

[34] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The anatomy of a context-aware application," *ACM/Springer Wireless Networks*, vol. 8, pp. 187–197, March 2002.

[35] N. B. Priyantha, A. K. Miu, H. Balakrishnan, and S. Teller, "The cricket compass for context-aware mobile applications," in *7th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2001)*, Rome, Italy, July 2001, pp. 1–14.

[36] G. G. Finn, "Routing and Addressing Problems in Large Metropolitan-Scale Internetworks," USC/ISI, ISI Research Report ED290427, 1987.

[37] T.-C. Hou and V. Li, "Transmission Range Control in Multihop Packet Radio Networks," *IEEE Transactions on Communications*, vol. 34, no. 1, pp. 38–44, January 1986.

[38] H. Takagi and L. Kleinrock, "Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals," *IEEE Transactions on Communications*, vol. 32, no. 3, pp. 246–257, March 1984.

[39] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," in *3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM 1999)*. Seattle, Washington, United States: ACM, 1999, pp. 48–55.

[40] B. Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," in *6th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2000)*, Boston, MA, 2000, pp. 243–254.

[41] B. Leong, B. Liskov, and R. Morris, "Greedy Virtual Coordinates for Geographic Routing," in *15th IEEE International Conference on Network Protocols (ICNP 2007)*, Beijing, China, October 2007, pp. 71–80.

[42] A. Caruso, S. Chessa, S. De, and A. Urpi, "GPS Free Coordinate Assignment and Routing in Wireless Sensor Networks," in *24th IEEE Conference on Computer Communications (IEEE INFOCOM 2005)*, Miami, FL, March 2005.

[43] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica, "Beacon Vector Routing: Scalable Point-to-Point Routing in Wireless Sensornets," in *USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005)*. San Francisco, CA: USENIX, 2005, pp. 329–342.

[44] Q. Fang, J. Gao, L. J. Guibas, V. de Silva, and L. Zhang, "GLIDER: Gradient Landmark-Based Distributed Routing for Sensor Networks," in *24th IEEE Conference on Computer Communications (IEEE INFOCOM 2005)*, Miami, FL, March 2005.

[45] Y. Zhao, Y. Chen, B. Li, and Q. Zhang, "Hop ID: A Virtual Coordinate-Based Routing for Sparse Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 9, pp. 1075–1089, September 2007.

[46] C.-H. Lin, B.-H. Liu, H.-Y. Yang, C.-Y. Kao, and M.-J. Tasi, "Virtual-Coordinate-Based Delivery-Guaranteed Routing Protocol in Wireless Sensor Networks with Unidirectional Links," in *27th IEEE Conference on Computer Communications (IEEE INFOCOM 2008)*. Phoenix, AZ: IEEE, April 2008.

[47] J. Ledlie, C. Ng, D. Holland, K.-K. Muniswamy-Reddy, U. Braun, and M. Seltzer, "Provenance-Aware Sensor Data Storage," in *21st International Conference on Data Engineering Workshops (ICDEW 2005), Workshop on Networking Meets Databases (NetDB)*. Atlanta, GA: IEEE, April 2005, pp. 1189–1189.

[48] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *6th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2000)*, Boston, MA, August 2000, pp. 56–67.

[49] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed Diffusion for Wireless Sensor Networking," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 2–16, February 2003.

[50] N. Sadagopan, B. Krishnamachari, and A. Helmy, "Active query forwarding in sensor networks," *Elsevier Ad Hoc Networks*, vol. 3, no. 1, pp. 91–113, January 2005.

[51] D. Braginsky and D. Estrin, "Rumor Routing Algorithm For Sensor Networks," in *1st Workshop on Sensor Networks and Applications (WSNA)*, Atlanta, GA, September 2002.

[52] C. Avin and C. Brito, "Efficient and robust query processing in dynamic environments using random walk techniques," in *3rd ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN 2004)*. Berkeley, CA: ACM, April 2004, pp. 277–286.

[53] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage," in *1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, Atlanta, GA, September 2002.

[54] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Data-Centric Storage in Sensornets," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 137–142, January 2003.

[55] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table," *ACM/Springer Mobile Networks and Applications (MONET), Special Issue on Wireless Sensor Networks*, vol. 8, no. 4, pp. 427–442, August 2003.

[56] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker, "DIFS: A Distributed Index for Features in Sensor Networks," in *1st IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003, pp. 163–173.

[57] D. Ganesan, D. Estrin, and J. Heidemann, "Dimensions: why do we need a new data handling architecture for sensor networks?" *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 143–148, January 2003.

[58] B. Sheng, Q. Li, and W. Mao, "Data Storage Placement in Sensor Networks," in *7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2006)*, Florence, Italy, May 2006, pp. 344–355.

[59] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," in *18th ACM Symposium on Operating Systems Principles (SOSP)*. Banff, Alberta, Canada: ACM, 2001, pp. 146–159.

[60] B. Krishnamachari, D. Estrin, and S. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," in *International Workshop Distributed Event Based System (DEBS 2002)*, Vienna, Austria, July 2002.

[61] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," in *5th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[62] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright, "Geographic Gossip: Efficient Aggregation for Sensor Networks," in *5th ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN 2006)*. Nashville, TN: ACM, April 2006, pp. 69–76.

[63] T. E. Daniel, R. M. Newman, E. I. Gaura, and S. N. Mount, "Complex query processing in wireless sensor networks," in *2nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM2HW2N 2007)*. Chania, Crete Island, Greece: ACM, 2007, pp. 53–60.

[64] R. Steinmetz and K. Wehrle, Eds., *Peer-to-Peer Systems and Applications*. Springer, 2005, vol. LNCS 3485.

[65] A. Oram, Ed., *Peer-to-Peer - Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[66] D. Eastlake and P. Jones, "US secure hash algorithm 1 (SHA1)," IETF, RFC 3174, September 2001.

[67] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM 2001*. San Diego, CA: ACM, August 2001, pp. 149–160.

[68] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 17–32, February 2003.

[69] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, "Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service," in *8th Workshop on Hot Topics in Operating System (HOTOS)*, Ilmau, Germany, May 2001, pp. 81–86.

[70] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[71] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *ACM SIGCOMM 2001*, San Diego, CA, 2001, pp. 161–172.

[72] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed Hashing in a Small World," in *4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, Seattle, WA, March 2003, pp. 127–140.

[73] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," in *29th ACM Symposium on Theory of Computing*. El Paso, TX: ACM, 1997, pp. 654–663.

[74] I. Chakeres and C. Perkins, "Dynamic MANET On-Demand (DYMO) Routing," Internet-Draft (work in progress) draft-ietf-manet-dymo-10.txt, July 2007.

[75] C. E. Perkins, E. M. Belding-Royer, and S. R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561, July 2003.

[76] S. Gwalani, E. M. Belding-Royer, and C. E. Perkins, "AODV-PA: AODV with Path Accumulation," in *IEEE International Conference on Communications (IEEE ICC 2003)*, Anchorage, AK, May 2003.

[77] I. D. Chakeres and L. Klein-Berndt, "AODVjr, AODV simplified," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 3, pp. 100–101, July 2002.

[78] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron, "Virtual Ring Routing: Network routing inspired by DHTs," in *ACM SIGCOMM 2006*. Pisa, Italy: ACM, September 2006.

[79] T. A. Herring, "The Global Positioning System," *Scientific American*, vol. 274, no. 2, pp. 44–50, February 1996.

[80] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *1st ACM Conference on Embedded Networked Sensor Systems (ACM SenSys 2003)*. Los Angeles, California, USA: ACM, November 2003, pp. 14–27.

[81] D. S. J. De Couto, D. Aguayo, B. A. Chambers, and R. Morris, "Performance of multihop wireless networks: shortest path is not enough," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 83–88, 2003.

[82] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker, "Geographic Routing Made Practical," in *USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005)*. San Francisco, CA: USENIX, 2005.

[83] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic Routing without Location Information," in *9th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2003)*, San Diego, CA, September 2003.

[84] K. Liu and N. Abu-Ghazaleh, "Aligned Virtual Coordinates for Greedy Routing in WSNs," in *3rd IEEE International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS 2006)*. Vancouver, Canada: IEEE, October 2006, pp. 377–386.

[85] B. N. Karp, "Geographic routing for wireless networks," PhD Thesis, Harvard University, 2000.

[86] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[87] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, January 2005.

[88] A. Awad, R. German, and F. Dressler, "P2P-based Routing and Data Management using the Virtual Cord Protocol (VCP)," in *9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2008), Poster Session*. Hong Kong, China: ACM, May 2008, pp. 443–444.

[89] A. Awad, C. Sommer, R. German, and F. Dressler, "Virtual Cord Protocol (VCP): A Flexible DHT-like Routing Service for Sensor Networks," in *5th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE MASS 2008)*. Atlanta, GA: IEEE, September 2008, pp. 133–142.

[90] A. Awad, L. R. Shi, R. German, and F. Dressler, "Advantages of Virtual Addressing for Efficient and Failure Tolerant Routing in Sensor Networks," in *6th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (IEEE/IFIP WONS 2009)*. Snowbird, UT: IEEE, February 2009, pp. 111–118.

[91] A. Varga, "The OMNeT++ Discrete Event Simulation System," in *European Simulation Multiconference (ESM 2001)*, Prague, Czech Republic, June 2001.

[92] C. Sommer, I. Dietrich, and F. Dressler, "A Simulation Model of DYMO for Ad Hoc Routing in OMNeT++," in *1st ACM/ICST International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008): 1st ACM/ICST International Workshop on OMNeT++ (OMNeT++ 2008)*. Marseille, France: ACM, March 2008.

[93] 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 *4th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 1998)*. Dallas, TX: ACM, October 1998.

[94] F. Dressler, R. Nebel, and A. Awad, "Distributed Passive Monitoring in Sensor Networks," in *26th IEEE Conference on Computer Communications (IEEE INFOCOM 2007), Demo Session*. Anchorage, AK: IEEE, May 2007.

[95] A. Awad, R. Nebel, R. German, and F. Dressler, "On the Need for Passive Monitoring in Sensor Networks," in *11th EUROMICRO Conference on Digital System Design - Architectures, Methods and Tools (DSD 2008)*. Parma, Italy: IEEE, September 2008, pp. 693–699.

[96] A. Awad, R. German, and F. Dressler, "Efficient Routing and Service Discovery in Sensor Networks using Virtual Cord Routing," in *7th ACM International Conference on Mobile Systems, Applications, and Services (ACM MobiSys 2009), Demo Session*. Kraków, Poland: ACM, June 2009.